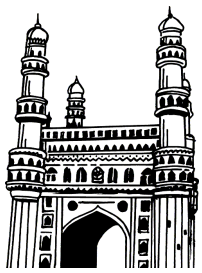
Rahul's ✔
Topper's Voice

AS PER
CBCS SYLLABUS



B.Sc.

III Year V Sem

Latest 2023 Edition

NO SQL DATA BASES

DATA SCIENCE PAPER - V(B)

- ☞ **Study Manual**
- ☞ **Important Questions**
- ☞ **Short Question & Answers**
- ☞ **Multiple Choice Questions**
- ☞ **Fill in the blanks**
- ☞ **One Mark Answers**
- ☞ **Solved Model Papers**

Price ` 189-00

- by -

WELL EXPERIENCED LECTURER

Rahul Publications™

Hyderabad. Ph : 66550071, 9391018098

B.Sc.

III Year V Sem

NO SQL DATA BASES

DATA SCIENCE PAPER - V(B)

Inspite of many efforts taken to present this book without errors, some errors might have crept in. Therefore we do not take any legal responsibility for such errors and omissions. However, if they are brought to our notice, they will be corrected in the next edition.

Price ` . 189-00

Sole Distributors :

☎ : **66550071, Cell : 9391018098**

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.

Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

CONTENTS

NO SQL DATA BASES

DATA SCIENCE PAPER - V(B)

STUDY MANUAL

SOLVED MODEL PAPERS

SYLLABUS

UNIT - I

Why NoSQL: The Value of Relational Databases, Impedance Mismatch, Application and Integration Databases, Attack of the Clusters, The Emergence of NoSQL Aggregate

Data Models: Aggregates, Column-Family Stores, Summarizing Aggregate Oriented

Databases More Details on Data Models: Relationships, Graph Databases, Schemaless Databases, Materialized Views, Modeling for Data Access.

UNIT - II

Distribution Models: Single Server, Sharding, Master-Slave Replication, Peer-to-Peer Replication, Combining Sharding and Replication.

Consistency: Update Consistency, Read Consistency, Relaxing Consistency, Relaxing Durability, Quorums

Version Stamps: Business and System Transactions, Version Stamps on Multiple Nodes **Map-Reduce:** Basic Map-Reduce, Partitioning and Combining, Composing Map-Reduce Calculations

UNIT - III

Key-Value Databases: What Is a Key-Value Store, Key-Value Store Features, Suitable Use Cases,

When Not to Use Document Databases: What Is a Document Database, Features, Suitable Use Cases, When Not to Use.

UNIT - IV

Column-Family Stores: What Is a Column-Family Data Store, Features, Suitable Use Cases,

When Not to Use Graph Databases: What Is a Graph Database, Features, Suitable Use Cases, When Not to Use

Contents

Important Questions

UNIT - I

1. What is a relational database? Explain the importance of it.

Ans :

Refer Unit-I, Q.No. 1

2. State the benefits of relational databases

Ans :

Refer Unit-I, Q.No. 2

3. Explain about the Applications and Integration of Databases?

Ans:

Refer Unit-I, Q.No. 5

4. What is Aggregate Data Model? Give with an example.

Ans:

Refer Unit-I, Q.No. 8

5. What is Aggregate-Oriented Databases in NoSQL?

Ans :

Refer Unit-I, Q.No. 10

6. Explain about the relations in NoSQL databases?

Ans:

Refer Unit-I, Q.No. 11

7. How does a schemaless database work?

Ans :

Refer Unit-I, Q.No. 13

8. Write about Modeling Access Techni- ques?

Ans :

Refer Unit-I, Q.No. 15

UNIT - II

1. Explain detail about single server in distribution models?

Ans :

Refer Unit-II, Q.No. 1

2. Elaborate on Master-Slave Replication in distribution models?

Ans :

Refer Unit-II, Q.No. 3

3. Explain in detail about Combining Sharding and Replication?

Ans:

Refer Unit-II, Q.No. 5

4. Explain Business and system Transac-tions with an example?

Ans :

Refer Unit-II, Q.No. 11

5. What are Version Stamps on Multiple Nodes in No SQL Database?

Ans :

Refer Unit-II, Q.No. 12

6. What is Basic Map-Reduce? Explain in detail how the data is processing?

Ans:

Refer Unit-II, Q.No. 13

7. Explain about Partitioning and Combin-ing with an example?

Ans :

Refer Unit-II, Q.No. 14

8. Explain Composing Map-Reduce Calculation with an example?

Ans :

Refer Unit-II, Q.No. 15

UNIT - III

1. What is a Key-Value (Model)Database? Explain with advantages and disadvantages.

Ans:

Refer Unit-III, Q.No. 1

2. What is a Key-Value Store? Explain the popular Key-Value Store databases

Ans:

Refer Unit-III, Q.No. 2

3. What are the Suitable Use Cases of Key - Value Database

Ans:

Refer Unit-III, Q.No. 4

4. Explain about Document Database in NoSQL

Ans:

Refer Unit-III, Q.No. 7

5. Explain about Suitable Use Cases of Document Database

Ans :

Refer Unit-III, Q.No. 10

UNIT - IV

1. Explain in detail about Column-Family Data Store.

Ans:

Refer Unit-IV, Q.No. 1

2. What is column store database? Give its advantages and disadvantages.

Ans:

Refer Unit-IV, Q.No. 2

3. What are the Features of Column Store Databases.

Ans:

Refer Unit-IV, Q.No. 3

4. What is column store database? Explain about When Not to Use column-family databases.

Ans:

Refer Unit-IV, Q.No. 5

5. What are the features of Graph Data bases?

Ans:

Refer Unit-IV, Q.No. 7

6. What are the Use Cases of Graph Data bases? Explain its advantages and disadvantages?

Ans:

Refer Unit-IV, Q.No. 9

UNIT I

Why NoSQL: The Value of Relational Databases, Impedance Mismatch, Application and Integration Databases, Attack of the Clusters, The Emergence of NoSQL Aggregate **Data Models:** Aggregates, Column-Family Stores, Summarizing Aggregate Oriented **Databases More Details on Data Models:** Relationships, Graph Databases, Schemaless Databases, Materialized Views, Modeling for Data Access.

1.1 WHY NOSQL

1.1.1 The Value of Relational Databases

Q1. What is a relational database? Explain the importance of it.

Ans : **(Imp.)**

Relational databases are a type of database that allows users to access data that is stored in various tables connected by a unique ID or "key." Using this key, users can unlock data entries related to that key on another table, to help with inventory management, shipping, and more. On relational database management systems (RDBMS), users can input SQL queries to retrieve the data needed for specific job functions.

In a relational database, each row in the table has a key. In the columns are data attributes. Each record has a value for each attribute, so users can understand the relationships between data entries for functions like product marketing, manufacturing, UX research, and more.

As an example, for a shoe store processing online orders, a relational database might have two tables with related data. In the first table, each record includes the customer's name, shipping address, email, and billing information, in columns. A key is assigned to each row. In the second table, that key is listed alongside the product ordered, quantity, size, color, and more. The two tables are related, and toggled to each other, with the key. When an order comes in, the key allows the warehouse to pull the correct product from the shelf and ship it to the customer.

Importance of Relational Database

A relational database's main benefit is the ability to connect data from different tables to create useful information. This approach helps organizations of all sizes and industries decipher relationships between different sets of data, from various departments, to create meaningful insights.

Q2. State the benefits of relational databases

Ans : **(Imp.)**

Relational databases provide plenty of benefits for companies. Here are a few primary advantages of relational databases:

(i) Simple and centralized database

Relational databases are simple. Toggling between tables provides a wealth of information that can be used for various purposes. Plus, ERP systems are built on relational databases, so they help users manage clients, inventory, and much more.

(ii) Easy to use

Many companies use relational databases, and ERP, to organize and manage large amounts of data. Their continued use helps to drive improvements to these systems-such as migrating to the cloud. Using SQL, users can easily navigate data sets to retrieve, filter, and ideate the information they need.

Save time and money: By using relational databases, companies can stay organized and efficient. The unique IDs help

eliminate duplicate information, whether it is tracking a customer's order or museum visitors. Instead of taking time to input logs of customer data, a relational database reduces redundancy, thus saving employees time. Companies can save money by allocating that labor elsewhere.

Q3. Explain the Features of relational databases.

Ans :

Relational databases tend to be used for processing and managing transactions. They are often used in retail, banking, and entertainment industries, where an exact amount (of money, tickets, or product) is withdrawn from one location or account and deposited into another. Transactions like these have properties that can be represented by the acronym ACID, which stands for:

Atomicity

All parts of a transaction are executed completely and successfully, or else the entire transaction fails.

Consistency

Data remains consistent throughout the relational database. Data integrity, or the accuracy and completeness of the data at hand, is enforced in relational databases with integrity constraints (similar to rule enforcers).

Isolation

Each transaction is independent of other transactions. Data from one record does not spill onto another, so it is secure.

Durability

Even if the system fails, data from completed transactions is safely stored.

By taking the relational approach to data queries, analysts can perform specific functions to obtain the information they need to organize query results by name, date, size, and location. This relational model also means that the logical data structures, such as tables and indexes, are completely separate from physical storage.

1.1.2 Impedance Mismatch

Q4. Explain in detail about Impedance Mismatch in relational databases.

Ans :

Impedance mismatch is the term used to refer to the problems that occurs due to differences between the database model and the programming language model. The practical relational model has 3 components these are:

1. Attributes and their data types
2. Tuples
3. Tables

Problems

Following problems may occur due to the impedance mismatch:

1. The first problem that may occur is that is data type mismatch means the programming language attribute data type may differ from the attribute data type in the data model.

   Hence it is quite necessary to have a binding for each host programming language that specifies for each attribute type the compatible programming language types. It is necessary to have different data types, for example, we have different data types available in different programming languages such as data types in C are different from Java and both differ from SQL data types.

2. The second problem that may occur is because the results of most queries are sets or multisets of tuples and each tuple is formed of a sequence of attribute values. In the program, it is necessary to access the individual data values within individual tuples for printing or processing.

   Hence there is a need for binding to map the query result data structure which is a table to an appropriate data structure in the

programming language. A mechanism is needed to loop over the tuples in a query result in order to access a single tuple at a time and to extract individual values from the tuple.

The extracted values are typically copied to appropriate program variables for further processing by the program.

A cursor or iterator is a variable which is used for looping over the tuples in a query result. Individual values within each tuple are extracted into different or unique program variables of the appropriate data type.

Impedance mismatch is less of a problem when a special database programming language is designed that uses the same data model and data type as a database model for example Oracle's PL/SQL.

1.1.3 Application and Integration Databases

Q5. Explain about the Applications and Integration of Databases?

Ans: **(Imp.)**

Applications of NoSQL Databases

1. Data Mining

When it comes to data mining, NoSQL databases are useful in retrieving information for data mining uses. Particularly when it's about large amounts of data, NoSQL databases store data points in both structured and unstructured formats leading to efficient storage of big data.

Perhaps when a user wishes to mine a particular dataset from large amounts of data, one can make use of NoSQL databases, to begin with. Data is the building block of technology that has led mankind to such great heights.

Therefore, one of the most essential fields where NoSQL databases can be put to use is data mining and data storage.

2. Social Media Networking Sites

Social media is full of data, both structured and unstructured. A field that is loaded with tons of data to be discovered, social media is one of the most effective applications of NoSQL databases.

From comments to posts, user-related information to advertising, socialmedia marketing requires NoSQL databases to be implemented in certain ways to retrieve useful information that can be helpful in certain ways.

Social media sites like Facebook and Instagram often approach open-source NoSQL databases to extract data that helps them keep track of their users and the activities going on around their platforms.

3. Software Development

The third application that we will be looking at is software development. Software development requires extensive research on users and the needs of the masses that are met through software development.

However, a developer must be able to scan through data that is available.

Perhaps NoSQL databases are always useful in helping software developers keep a tab on their users, their details, and other user-related data that is important to be noted. That said, NoSQL data bases are surely helpful in software development.

Integration Databases in NoSQL

Nowadays, an enormous amount of information is been made each second. This information is of different schemes-unstructured, structured, and semi-structured data. The variety and volume of this information can't be managed by traditional databases. Therefore, NoSQL frameworks have emerged, which is another age of database framework.

To handle data that is heterogeneous, NoSQL databases are more proficient at this. Tools that can be used to scale for accommodating a large volume

of information are required by NoSQL data integration, however, manual complicated coding is required by conventional SQL ETL tools. and also they include methods disturbing creation sources.

A database serving as a store for numerous applications is called an integration database and therefore, data is integrated across applications. A schema is needed by an integration database, and all applications of clients are taken by the schema into account. Either the resultant schema is general or complicated or both.

Here is an example for a better understanding of the integration database. For example, the computation data of an organization is stored in the Oracle database and client information is stored in Salesforce. The employees can get the integrated data of the two frameworks in a single spot with the help of database integration processes. website database integration is used by a few organizations for managing and bringing together information from different site pages. Database integration is only viable with the consolidation of data from on-premise systems, legacy systems, and cloud databases. Different software is used by each company.

Benefits

Here are a few benefits of database integration:

- Database integration helps in managing a large amount of data of an enterprise coming from a central location hence improving the experience for customers and also reducing delivery time. It helps in gaining control of data.
- Database integration makes it easy to ensure compliance of the business by enabling centralized management.
- The consolidation of data is allowed by the tools of database integration from a vast assortment of sources, which is then edited, changed, and loaded into the selected target database.

Data Integration Solutions for NoSQL Systems

Here are some methods for solving data integration problems in reference to NoSQL:

- Save Our Systems (SOS) is integration by a Middleware system permitting access to data stored in various NoSQL databases within a single application utilizing an API.
- SQL++ is a unifying semi-structured data model and query language for SQL-on-Hadoop, NoSQL, and new SQL databases following a uniform data access technique for solving issues of variations of databases query languages.
- A metamodel-based data merging approach is another implementation of the constant data access method. It is a user-friendly interface that helps to query data from heterogeneous databases without any programming skills.
- Query Arrow is integrated by the Middleware approach and is a generic software that queries and updates data from numerous databases.

1.1.4 Attack of the Clusters

Q6. Explain about Attack of the Clusters in NoSQL?

Ans:

Websites started tracking activity and structure in a very detailed way. Large sets of data appeared: links, social networks, activity in logs, mapping data. With this growth in data came a growth in users - as the biggest websites grew to be vast estates regularly serving huge numbers of visitors.

Coping with the increase in data and traffic required more computing resources. To handle this kind of increase, you have two choices: up or out. Scaling up implies bigger machines, more

processors, disk storage, and memory. But bigger machines get more and more expensive, not to mention that there are real limits as your size increases. The alternative is to use lots of small machines in a cluster. A cluster of small machines can use commodity hardware and ends up being cheaper at these kinds of scales. It can also be more resilient - while individual machine failures are common, the overall cluster can be built to keep going despite such failures, providing high reliability.

As large properties moved towards clusters, that revealed a new problem - relational databases are not designed to be run on clusters. Clustered relational databases, such as the Oracle RAC or Microsoft SQL Server, work on the concept of a shared disk subsystem. They use a cluster-aware file system that writes to a highly available disk subsystem - but this means the cluster still has the disk subsystem as a single point of failure. Relational databases could also be run as separate servers for different sets of data, effectively sharding the database. While this separates the load, all the sharding has to be controlled by the application which has to keep track of which database server to talk to for each bit of data. Also, we lose any querying, referential integrity, transactions, or consistency controls that cross shards. A phrase we often hear in this context from people who've done this is "unnatural acts."

These technical issues are exacerbated by licensing costs. Commercial relational databases are usually priced on a single-server assumption, so running on a cluster raised prices and led to frustrating negotiations with purchasing departments.

This mismatch between relational databases and clusters led some organization to consider an alternative route to data storage. Two companies in particular - Google and Amazon - have been very influential. Both were on the forefront of running large clusters of this kind; furthermore, they were capturing huge amounts of data. These things gave them the motive. Both were successful and growing companies with strong technical components, which gave them the means and opportunity. It was no wonder they had murder in mind for their relational databases. As the 2000s drew on, both companies produced brief but highly influential papers about their efforts: BigTable from Google and Dynamo from Amazon.

1.1.5 The Emergence of NoSQL

Q7. Explain in detail about Emergence of NoSQL?

Ans :

NoSQL databases emerged in the late 2000s as the cost of storage dramatically decreased. Gone were the days of needing to create a complex, difficult-to-manage data model in order to avoid data duplication. Developers (rather than storage) were becoming the primary cost of software development, so NoSQL databases optimized for developer productivity.

As storage costs rapidly decreased, the amount of data that applications needed to store and query increased. This data came in all shapes and sizes - structured, semi-structured, and polymorphic and defining the schema in advance became nearly impossible. NoSQL databases allow developers to store huge amounts of unstructured data, giving them a lot of flexibility.

Additionally, the Agile Manifesto was rising in popularity, and software engineers were rethinking the way they developed software. They were recognizing the need to rapidly adapt to changing requirements. They needed the ability to iterate quickly and make changes throughout their software stack all the way down to the database. NoSQL databases gave them this flexibility.

Cloud computing also rose in popularity, and developers began using public clouds to host their applications and data. They wanted the ability to distribute data across multiple servers and regions to make their applications resilient, to scale out instead of scale up, and to intelligently geo-place their data. Some NoSQL databases like MongoDB provides these capabilities.

Types

Over time, four major types of NoSQL databases emerged: document databases, key-value databases, wide-column stores, and graph databases.

- Document databases store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.
- Key-value databases are a simpler type of database where each item contains keys and values.
- Wide-column stores store data in tables, rows, and dynamic columns.
- Graph databases store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

1.2 Aggregate Data Models

1.2.1 Aggregates

Q8. What is Aggregate Data Model? Give with an example.

Ans: **(Imp.)**

Meaning

NoSQL are databases that store data in another format other than relational databases. NoSQL deals in nearly every industry nowadays. For the people who interact with data in databases, the Aggregate Data model will help in that interaction.

Features

- **Schema Agnostic**

  NoSQL Databases do not require any specific schema or s storage structure than traditional RDBMS.

- **Scalability**

  NoSQL databases scale horizontally as data grows rapidly certain commodity hardware could be added and scalability features could be preserved for NoSQL.

- **Performance**

  To increase the performance of the NoSQL system one can add a different commodity server than reliable and fast access of database transfer with minimum overhead.

- **High Availability**

  In traditional RDBMS it relies on primary and secondary nodes for fetching the data, Some NoSQL databases use master place architecture.

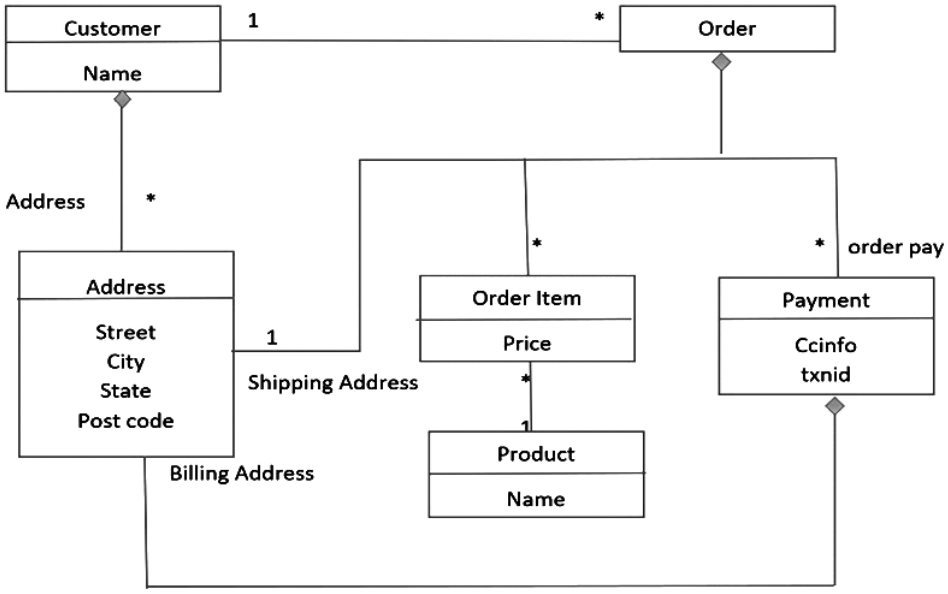
- **Global Availability**

  As data is replicated among multiple servers and clouds the data is accessible to anyone, this minimizes the latency period.

Aggregate Data Models

The term aggregate means a collection of objects that we use to treat as a unit. An aggregate is a collection of data that we interact with as a unit. These units of data or aggregates form the boundaries for ACID operation.

Example of Aggregate Data Model



Here in the diagram have two Aggregate

- Customer and Orders link between them represent an aggregate.
- The diamond shows how data fit into the aggregate structure.
- Customer contains a list of billing address
- Payment also contains the billing address
- The address appears three times and it is copied each time
- The domain is fit where we don't want to change shipping and billing address.

Consequences of Aggregate Orientation

- Aggregation is not a logical data property It is all about how the data is being used by applications.
- An aggregate structure may be an obstacle for others but help with some data interactions.
- It has an important consequence for transactions.
- NoSQL databases don't support ACID transactions thus sacrificing consistency.
- Aggregate-oriented databases support the atomic manipulation of a single aggregate at a time.

1.2.2 Column-Family Stores

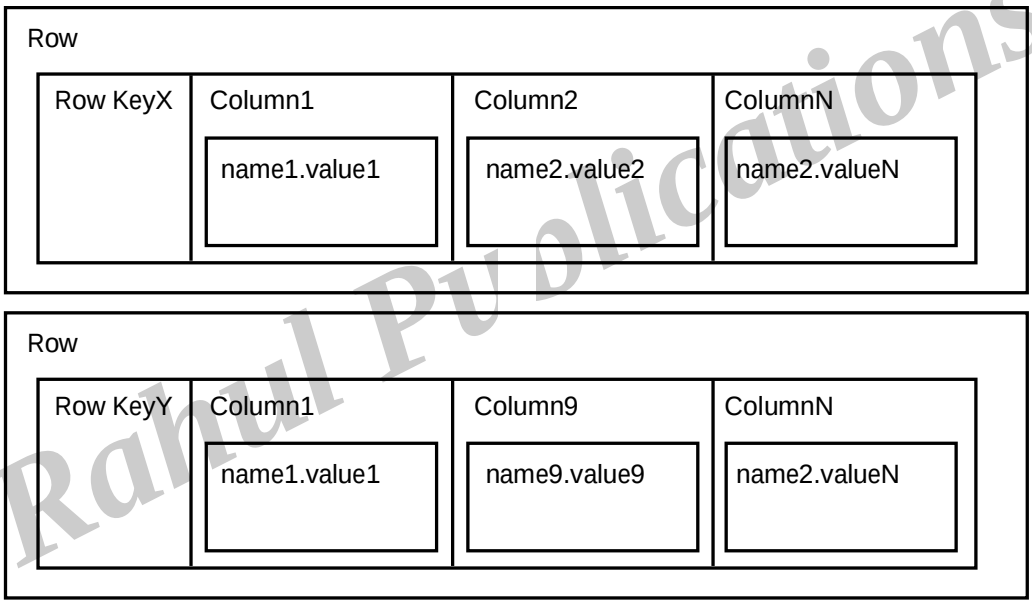
Q9. What is Column-Family Stores? Explain in detail?

Ans:

Column-family databases store data in column families as rows. These rows have many columns associated with a particular row. Column families basically contain the group of correlated data which we can access together.

- Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns.
- Rows do not need to have the same columns, and columns can be added to any row at any time without having to add it to other rows.
- When a column consists of a map of columns, we have a super column. A super column consists of a name and a value which is a map of columns. Think of a super column as a container of columns.
- Some Column-family databases are:
  - Cassandra
  - Hbase
  - Hypertable
  - Amazon DynamoDB

Cassandra is faster and more scalable as compared to other column-family databases with write operations because data is spread across the cluster.



Four Major Benefits of Column Family Database

- **Compression:** Column-based data storage stores data efficiently through data compression and by using data partitioning.
- **Aggregation Queries:** Due to the structure of the column family data structure, they perform particularly well with aggregation queries (such as SUM, COUNT, AVG etc).
- **Scalability:** Column databases are more scalable as compared to other databases. They are well suited for a data structure where data is spread on a large cluster of machines, often thousands of machines.

Fast to load and query

Columnar stores can be loaded fast. A table containing millions of rows can be loaded within a few seconds. We can start querying and analyzing immediately on the loaded data.

1.2.3 Summarizing Aggregate-Oriented Databases in NoSQL

Q10. What is Aggregate-Oriented Databases in NoSQL?

Ans : **(Imp.)**

The Aggregate-Oriented database is the NoSQL database which does not support ACID transactions and they sacrifice one of the ACID properties. Aggregate orientation operations are different compared to relational database operations. We can perform OLAP operations on the Aggregate-Oriented database. The efficiency of the Aggregate-Oriented database is high if the data transactions and interactions take place within the same aggregate. Several fields of data can be put in the aggregates such that they can be commonly accessed together. We can manipulate only a single aggregate at a time. We can not manipulate multiple aggregates at a time in an atomic way.

Aggregate

Oriented databases are classified into four major data models. They are as follows:

1. Key-value
2. Document
3. Column family
4. Graph-based

Each of the Data models above has its own query language.

1. Key-value Data Model

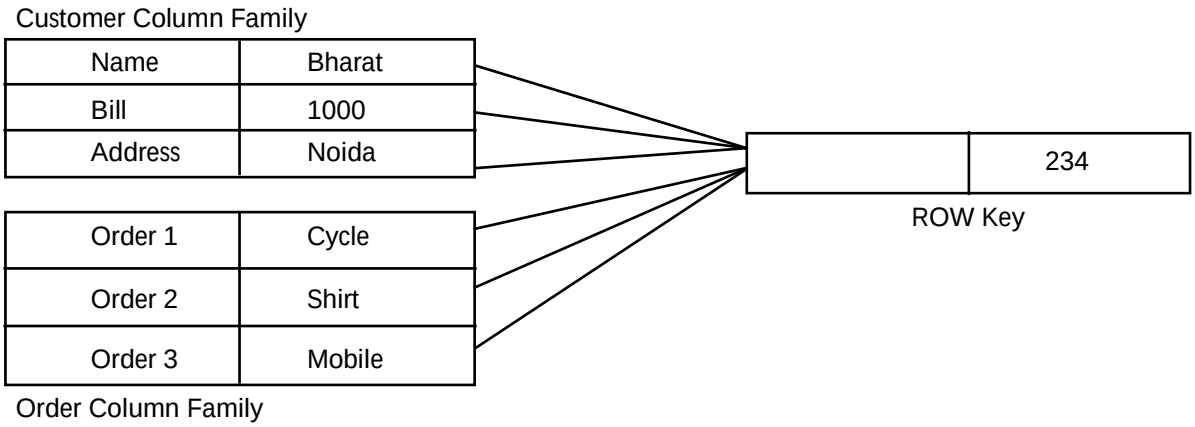
Key-value and document databases were strongly aggregate-oriented. The key-value data model contains the key or Id which is used to access the data of the aggregates. key-value Data Model is very secure as the aggregates are opaque to the database. Aggregates are encrypted as the big blog of bits that can be decrypted with key or id. In the key-value Data Model, we can place data of any structure and datatypes in it. The advantage of the key-value Data Model is that we can store the sensitive information in the aggregate. But the disadvantage of this model the database has some general size limits. We can store only the limited data.

2. Document Data Model

In Document Data Model we can access the parts of aggregates. The data in this model can be accessed inflexible manner. We can submit queries to the database based on the fields in the aggregate. There is a restriction on the structure and data types of data to be paced in this data model. The structure of the aggregate can be accessed by the Document Data Model.

3. Column family Data Model

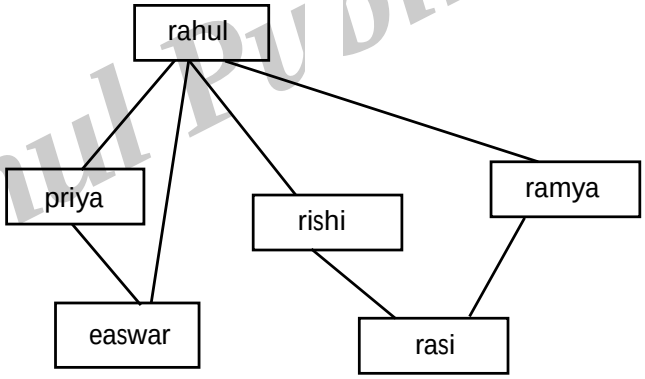
The Column family is also called a two-level map. But, however, we think about the structure, it has been a model that influenced later databases such as HBase and Cassandra. These databases with a big table-style data model are often referred to as column stores. Column-family models divide the aggregate into column families. The Column-family model is a two-level aggregate structure. The first level consists of keys that act as a row identifier that selects the aggregate. The second-level values in the Column family Data Model are referred to as columns.



in the above example, the row key is 234 which selects the aggregate. Here the row key selects the column families customer and orders. Each column family contains the columns of data. In the orders column family, we have the orders placed by the customers.

4. Graph Data Model

In a graph data model, the data is stored in nodes that are connected by edges. This model is preferred to store a huge amount of complex aggregates and multidimensional data with many interconnections between them. Graph Data Model has the application like we can store the Facebook user accounts in the nodes and find out the friends of the particular user by following the edges of the graph.



We can find the friends of a person by observing this graph data model. If there is an edge between two nodes then we can say they are friends. Here we also consider the indirect links between the nodes to determine the friend suggestions.

1.3 More Details on Data Models

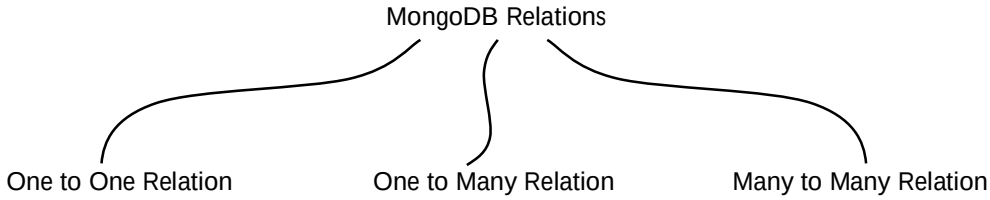
1.3.1 Relations

Q11. Explain about the relations in NoSQL databases?

Ans: **(Imp.)**

Relations are the crux of any database and relations in NoSQL databases are handled in a completely different way compared to an SQL database. There is one very important difference that you need to keep in mind while building a NoSQL database and that is, NoSQL databases usually always have a JSON

like Schema. Once you're familiar with that, then handling relations will be a lot easier. ideally, there are 3 basic kinds of relationships, they are as below:



1. One to One Relation

One to one relation, as the name suggests requires one entity to have an exclusive relationship with another entity and vice versa. Let's consider a simple example to understand this relationship better… The relationship between a user and his account. One user can have one account associated with him and one account can have only one user associated with it.

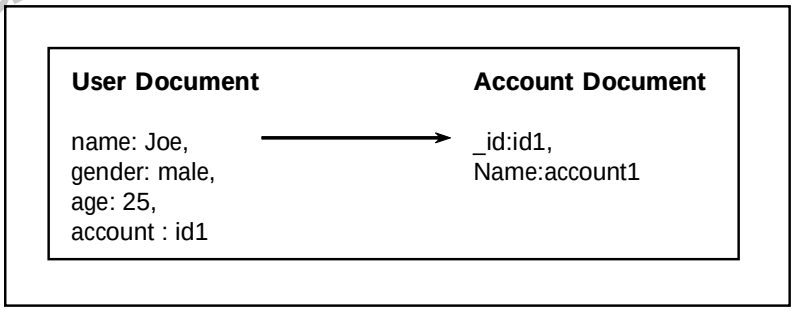
One to one relationships can be handled in two ways…

First and the easiest one is to have just one collection, the 'user' collection and the account of that particular user will be stored as an object in the user document itself.

User Document

```
name : Joe,
gender: male,
age : 25,
account : {
name: account 1,
}
```

The second way is to create another collection named account and store a reference key (ideally the ID of the account) in the user document.



This way is usually used when one of the following three scenarios occur...

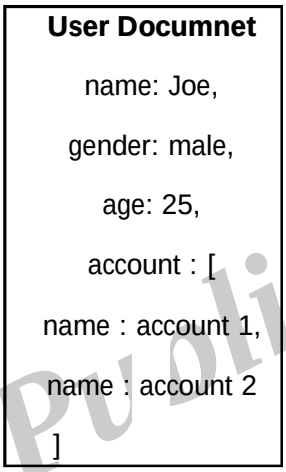
1. The main document is too large (MongoDB documents have a size limit of 16mb)
2. When some sensitive information needs to be stored (you might not want to return account information on every user GET request).
3. When there's an exclusive need for getting the account data without the user data (when 'account' is requested you don't want to send 'user' information with it and/or when a 'user' is requested you don't want to send 'account' information with it, even though both of them are connected).

2. One to many Relation

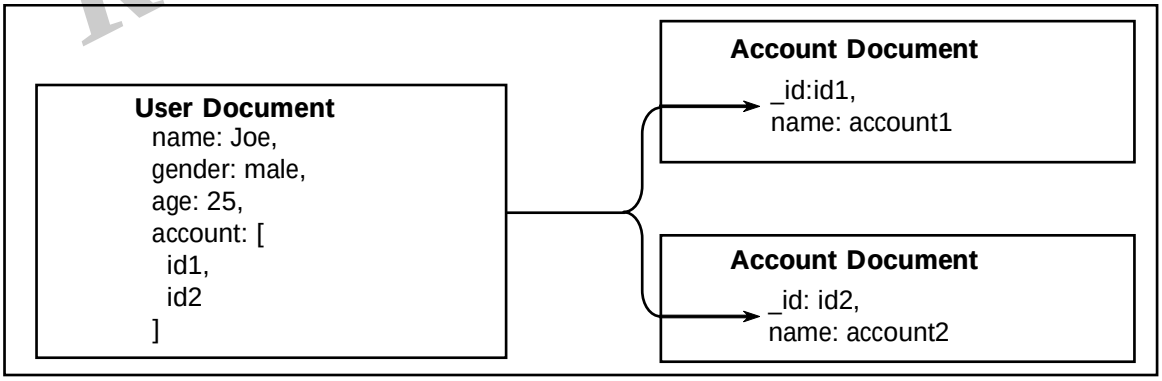
One to many relation, requires one entity to have an exclusive relationship with another entity but the other entity can have relations with multiple other entities. Let's consider a simple example to understand this relationship better…

Consider, a user has multiple accounts, but each account can have a single user associated with it (think about these accounts as bank accounts, it'll let you understand the example better). In this case, again there are two ways to handle it.

The first is to store an array of accounts in the user collection itself. This will let you GET all the accounts associated with a user in a single call. MongoDB also has features to push and pull data from an array in a document, which makes it quite easy to add or remove accounts from the user if need be.



The second way is to create another collection named 'account' and store a reference key (ideally the ID of the account) in the 'user' document. The reasons to do this are the same as in the case of one to one relations.



One issue with this approach is that when a new account needs to be created for a particular user, we need to create a new account and also update the existing user document with the id of this new account (basically requires 2 database calls). Obviously you can store the user ID in Account collection as well, in that way, you'll only need one call to create a new account but it depends on the system you're planning to build.

Before building the schema, it's important that you plan out what kind of calls will be used more in your system and plan your schema accordingly.

For example, in this case, since this is a bank application (assumption), you know that most of the calls you'll make would be getting a single user (while logging in maybe) and another call to get the accounts associated with that user (when he goes to the accounts tab maybe) and hence the above schema seems a pretty good one for this use case. In fact, storing user_id in the accounts' collection would be an even better approach in this case.

Now consider another scenario, this time it's a public forum, users can create posts and these posts can be viewed by the public. In this case, it's better to store user_id in posts collection, instead of storing post_ids in users collection, since you know that your selling point is the posts list that the users can view and hence the calls you mostly make would be to get the posts list, with the user data associated with it (maybe in the homepage itself, like Facebook's timeline). This way, while updating you wouldn't need to update two collections.

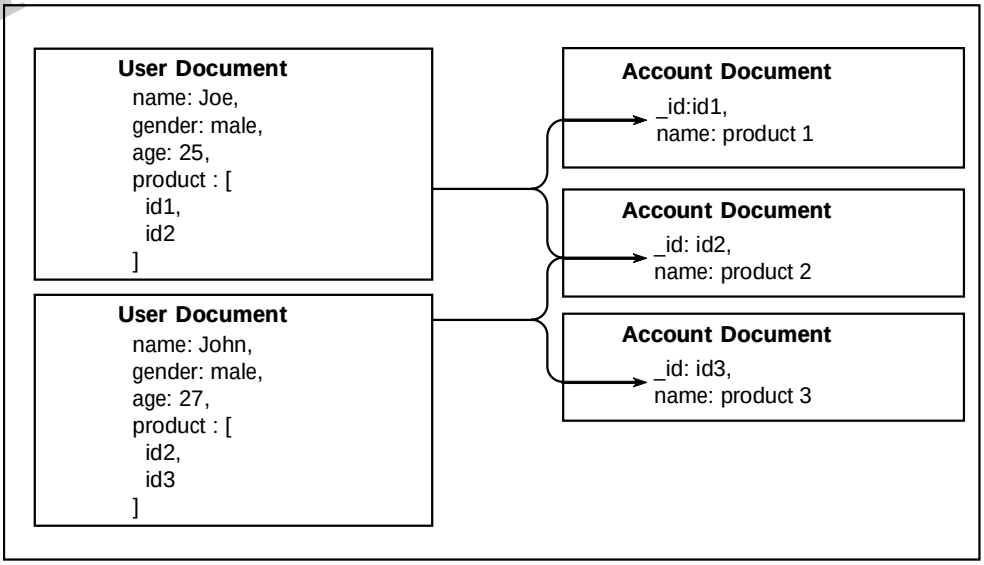
Another scenario would be that you need both of them, that is, you need posts in users' data as well and users in posts data as well. This will make creating new posts a bit slow (since you need to add IDs to the users' collection as well), but getting data in both cases would be fast.

3. Many to many Relation

Many to many relation, doesn't require any entity to have exclusive relations. Both entities can have multiple relations. Let's consider a simple example to understand this relationship better.

Consider the relationship between users and products in an eCommerce environment. There is a list of users and there is a list of products. Any user can buy any product, meaning a user can buy multiple products and a product can be bought by multiple users. In this case, there is just one ideal way to handle it.

There'll be two collections, one a collection for users and the other a collection from products. Whenever a user buys a product, add the ID of the product as a reference in the user's collection, and since the user can buy multiple products, these IDs need to be stored as an array.



When a product needs to be updated, only that product in the product collection needs to be updated and every user who has bought the product will automatically get the updated product.

1.3.2 Graph Databases

Q12. Graph Based Databases in NoSQL with its applications.

(OR)

Graph Based Data Model in NoSQL with its applications

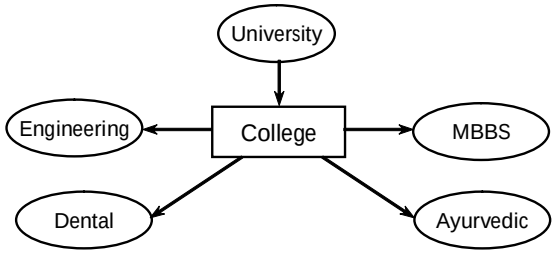
Ans :

Graph Based Data Model in NoSQL is a type of Data Model which tries to focus on building the relationship between data elements. As the name suggests Graph-Based Data Model, each element here is stored as a node, and the association between these elements is often known as Links. Association is stored directly as these are the first-class elements of the data model. These data models give us a conceptual view of the data.

These are the data models which are based on topographical network structure. Obviously, in graph theory, we have terms like Nodes, edges, and properties, let's see what it means here in the Graph-Based data model.

- **Nodes:** These are the instances of data that represent objects which is to be tracked.
- **Edges:** As we already know edges represent relationships between nodes.
- **Properties:** It represents information associated with nodes.

The below image represents Nodes with properties from relationships represented by edges.



Working of Graph Data Model

In these data models, the nodes which are connected together are connected physically and the physical connection among them is also taken as a piece of data. Connecting data in this way becomes easy to query a relationship. This data model reads the relationship from storage directly instead of calculating and querying the connection steps. Like many different NoSQL databases these data models don't have any schema as it is important because schema makes the model well and good and easy to edit.

Examples

- **JanusGraph:** These are very helpful in big data analytics. It is a scalable graph database system open source too. JanusGraph has different features like:
- **Storage:** Many options are available for storing graph data like Cassandra.
- **Support for transactions:** There are many supports available like ACID (Atomicity, Consistency, Isolation, and Durability) which can hold thousands of concurrent users.
- **Searching options:** Complex searching options are available and optional support too.
- **Neo4j:** It stands for Network Exploration and Optimization 4 Java. As the name suggests this graph database is written in Java with native graph storage and processing. Neo4j has different features like:
- **Scalable:** Scalable through data partitioning into pieces known as shards.
- **Higher Availability:** Availability is very much high due to continuous backups and rolling upgrades.
- **Query Language:** Uses programmer-friendly query language Cypher graph query language.DGraph main features are:
- **DGraph:** It is an open-source distributed graph database system designed with scalability.

- **Query Language:** It uses GraphQL, which is solely made for APIs.
- **Open-source system:** support for many open standards.

Applications

- Graph data models are very much used in fraud detection which itself is very much useful and important.
- It is used in Digital asset management which provides a scalable database model to keep track of digital assets.
- It is used in Network management which alerts a network administrator about problems in a network.
- It is used in Context-aware services by giving traffic updates and many more.
- It is used in Real-Time Recommendation Engines which provide a better user experience.

1.3.3 Schemaless Databases

Q13. How does a schemaless database work?

Ans : **(Imp.)**

A schemaless database manages information without the need for a blueprint. The onset of building a schemaless database doesn't rely on conforming to certain fields, tables, or data model structures. There is no Relational Database Management System (RDBMS) to enforce any specific kind of structure.

In schemaless databases, information is stored in JSON-style documents which can have varying sets of fields with different data types for each field. So, a collection could look like this:

```
{
name : "Joe", age : 30, interests : 'football' }
{
name : "Kate", age : 25
}
```

The data itself normally has a fairly consistent structure. With the schemaless MongoDB database, there is some additional structure the system namespace contains an explicit list of collections and indexes. Collections may be implicitly or explicitly created indexes must be explicitly declared.

Benefits

- **Greater flexibility over data types:** By operating without a schema, schemaless databases can store, retrieve, and query any data type perfect for big data analytics and similar operations that are powered by unstructured data. Relational databases apply rigid schema rules to data, limiting what can be stored.
- **No pre-defined database schemas:** The lack of schema means that your NoSQL database can accept any data type including those that you do not yet use. This future-proofs your database, allowing it to grow and change as your data-driven operations change and mature.
- **No data truncation:** A schemaless database makes almost no changes to your data; each item is saved in its own document with a partial schema, leaving the raw information untouched. This means that every detail is always available and nothing is stripped to match the current schema. This is particularly valuable if your analytics needs to change at some point in the future.
- **Suitable for real-time analytics functions:** With the ability to process unstructured data, applications built on NoSQL databases are better able to process real-time data, such as readings and measurements from IoT sensors. Schemaless databases are also ideal for use with machine learning and artificial intelligence operations, helping to accelerate automated actions in your business.

- **Enhanced scalability and flexibility:** With NoSQL, you can use whichever data model is best suited to the job. Graph databases allow you to view relationships between data points, or you can use traditional wide table views with an exceptionally large number of columns. You can query, report, and model information however you choose. And as your requirements grow, you can keep adding nodes to increase capacity and power.

When a record is saved to a relational database, anything (particularly metadata) that does not match the schema is truncated or removed. Deleted at write, these details cannot be recovered at a later point in time.

A lack of rigid schema allows for increased transparency and automation when making changes to the database or performing a data migration. Say you want to add GPA attributes to student objects held in your database. You simply add the attribute, resave, and the GPA value has been added to the NoSQL document. If you look up an existing student and reference GPA, it will return null. If you roll back your code, the new GPA fields in the existing objects are unlikely to cause problems and do not need to be removed if your code is well written.

1.3.4 Materialized Views

Q14. Explain in detail about Materialized View.

Ans :

A materialized view is a replica of a target master from a single point in time. The master can be either a master table at a master site or a master materialized view at a materialized view site. Whereas in multimaster replication tables are continuously updated by other master sites, materialized views are updated from one or more masters through individual batch updates, known as a refreshes, from a single master site or master materialized view site, as illustrated in Figure The arrows in Figure represent database links.

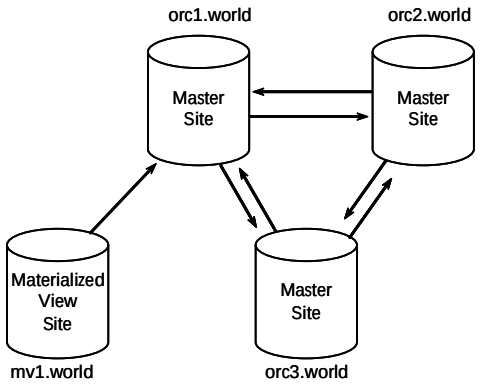


Fig.: Materialized View Connected to a Single Master Site

When a materialized view is fast refreshed, Oracle must examine all of the changes to the master table or master materialized view since the last refresh to see if any apply to the materialized view. Therefore, if any changes where made to the master since the last refresh, then a materialized view refresh takes some time to apply the changes to the materialized view. If, however, no changes at all were made to the master since the last refresh of a materialized view, then the materialized view refresh should be very quick.

Use of Materialized Views

Materialized views to achieve one or more of the following goals:

- Ease Network Loads
- Create a Mass Deployment Environment
- Enable Data Subsetting
- Enable Disconnected Computing

Ease Network Loads

If one of your goals is to reduce network loads, then you can use materialized views to distribute your corporate database to regional sites. Instead of the entire company accessing a single database server, user load is distributed across multiple database servers. Through the use of multitier materialized views, you can create materialized views based on other materialized views, which enables you to distribute user load to an even greater extent

because clients can access materialized view sites instead of master sites. To decrease the amount of data that is replicated, a materialized view can be a subset of a master table or master materialized view.

Create a Mass Deployment Environment

Deployment templates allow you to precreate a materialized view environment locally. You can then use deployment templates to quickly and easily deploy materialized view environments to support sales force automation and other mass deployment environments. Parameters allow you to create custom data sets for individual users without changing the deployment template. This technology enables you to roll out a database infrastructure to hundreds or thousands of users.

Enable Data Subsetting

Materialized views allow you to replicate data based on column- and row-level subsetting, while multimaster replication requires replication of the entire table. Data subsetting enables you to replicate information that pertains only to a particular site. For example, if you have a regional sales office, then you might replicate only the data that is needed in that region, thereby cutting down on unnecessary network traffic.

Enable Disconnected Computing

Materialized views do not require a dedicated network connection. Though you have the option of automating the refresh process by scheduling a job, you can manually refresh your materialized view on-demand, which is an ideal solution for sales applications running on a laptop. For example, a developer can integrate the replication management API for refresh on-demand into the sales application. When the salesperson has completed the day's orders, the salesperson simply dials up the network and uses the integrated mechanism to refresh the database, thus transferring the orders to the main office.

1.3.5 Modeling for Data Access

Q15. Write about Modeling Access Techniques?

Ans : **(Imp.)**

NoSQL Data Modeling Access Techniques

All NoSQL data modeling techniques are grouped into three major groups:

i) Conceptual techniques

ii) General modeling techniques

iii) Hierarchy modeling techniques

(i) Conceptual Techniques

There are three conceptual techniques for NoSQL data modeling:

- **Denormalization**: Denormalization is a pretty common technique and entails copying the data into multiple tables or forms in order to simplify them. With denormalization, easily group all the data that needs to be queried in one place. Of course, this does mean that data volume does increase for different parameters, which increases the data volume considerably.
- **Aggregates**: This allows users to form nested entities with complex internal structures, as well as vary their particular structure. Ultimately, aggregation reduces joins by minimizing one-to-one relationships.

  Most NoSQL data models have some form of this soft schema technique. For example, graph and key-value store databases have values that can be of any format, since those data models do not place constraints on value. Similarly, another example such as BigTable has aggregation through columns and column families.
- **Application Side Joins:** NoSQL doesn't usually support joins, since NoSQL databases are question-oriented where joins are done during design time. This is compared to

relational databases where are performed at query execution time. Of course, this tends to result in a performance penalty and is sometimes unavoidable.

(ii) General Modeling Techniques

There are five general techniques for NoSQL data modeling:

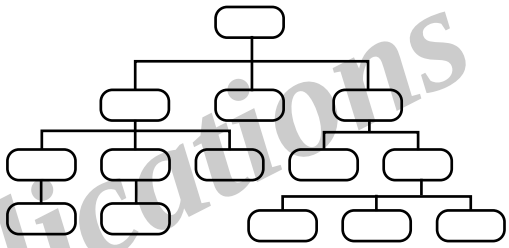
- **Enumerable Keys**: For the most part, unordered key values are very useful, since entries can be partitioned over several dedicated servers by just hashing the key. Even so, adding some form of sorting functionality through ordered keys is useful, even though it may add a bit more complexity and a performance hit.
- **Dimensionality Reduction**: Geographic information systems tend to use R-Tree indexes and need to be updated in-place, which can be expensive if dealing with large datavolumes. Another traditional approach is to flatten the 2D structure into a plain list, such as what is done with Geohash.

  With dimensionality reduction, you can map multidimensional data to a simple key-value or even non-multidimensional models.

  Use dimensionality reduction to map multidimensional data to a Key-Value model or to another non-multidimensional model.
- **Index Table:** With an index table, take advantage of indexes in stores that don't necessarily support them internally. Aim to create and then maintain a unique table with keys that follow a specific access pattern. For example, a master table to store user accounts for access by user ID.
- **Composite Key Index**: While somewhat of a generic technique, composite keys are incredibly useful when ordered keys are used. If you take it and combine it with secondary keys, you can create a multidimensional index that is pretty similar to the above-mentioned Dimensionality Reduction technique.
- **Inverted Search – Direct Aggregation:** The concept behind this technique is to use an index that meets a specific set of criteria, but then aggregate that data with full scans or some form of original representation.

  This is more of a data processing pattern than data modeling, yet data models are certainly affected by using this type of processing pattern. Take into account that random retrieval of records required for this technique is inefficient. Use query processing in batches to mitigate this problem.

(iii) Hierarchy Modeling Techniques



There are seven hierarchy modeling techniques for NoSQL data:

- **Tree Aggregation:** Tree aggregation is essentially modeling data as a single document. This can be really efficient when it comes to any record that is always accessed at once, such as a Twitter thread or Reddit post. Of course, the problem then becomes that random access to any individual entry is inefficient.
- **Adjacency Lists:** This is a straightforward technique where nodes are modeled as independent records of arrays with direct ancestors. That's a complicated way of saying that it allows you to search nodes by their parents or children. Much like tree aggregation though, it is also quite inefficient for retrieving an entire subtree for any given node.
- **Materialized Paths:** This technique is a sort of denormalization and is used to avoid recursive traversals in tree structures. Mainly,

we want to attribute the parents or children to each node, which helps us determine any predecessors or descendants of the node without worrying about traversal. Incidentally, we can store materialized paths as IDs, either as a set or a single string.

- **Nested Sets**: A standard technique for tree-like structures in relational databases, it's just as applicable to NoSQL and key-value or document databases. The goal is to store the tree leaves as an array and then map each non-leaf node to a range of leaves using start/end indexes.

  Modeling it in this way is an efficient way to deal with immutable data as it only requires a small amount of memory, and doesn't necessarily have to use traversals. That being said, updates are expensive because they require updates of indexes.

- **Nested Documents Flattening: Numbe-red Field Names:** Most search engines tend to work with documents that are a flat list of fields and values, rather than something with a complex internal structure. As such, this data modeling technique tries to map these complex structures to a plain document, for example, mapping documents with a hierarchical structure, a common difficulty you might encounter.

  Of course, this type of work is pain-staking and not easily scalable, especially as the nested structures increase.

- **Nested Documents Flattening: Proximity Queries:** One way to solve the potential problems with the Numbered Field Names data modeling technique is to use a similar technique called **Proximity Queries.** These limit the distance between words in a document, which helps increase performance and decrease query speed impact.

- **Batch Graph Processing:** Batch graph processing is a great technique for exploring the relationships up or down for a node, within a few steps. It is an expensive process and doesn't necessarily scale very well. By using Message Passing and MapReduce we can carry out this type of graph processing.

Short Question & Answers

1. What is a NoSQL data model?How is data stored in NoSQL?

Ans :

It's a model that is not reinforced by a Relational Database Management System (RDBMS). Therefore, the model isn't explicit about how the data relates – how it all connects together.

In one of the main non-relational database models, such as a key-value store, document store, graph data model, time series store, column-oriented. Data can be stored on disk, in-memory, or both.

2. What are the advantages and disadvantages of Aggregate Data Model?

Ans :

Advantage

- It can be used as a primary data source for online applications.
- Easy Replication.
- No single point Failure.
- It provides fast performance and horizontal Scalability.
- It can handle Structured semi-structured and unstructured data with equal effort.

Disadvantage

- No standard rules.
- Limited query capabilities.
- Doesn't work well with relational data.
- Not so popular in the enterprise.
- When the value of data increases it is difficult to maintain unique values.

3. What are the Advantages and Disadvantages of Graph Data Model?

Ans :

Advantages

- **Structure:** The structures are very agile and workable too.
- **Explicit Representation:** The portrayal of relationships between entities is explicit.
- **Real-time O/P Results:** Query gives us real-time output results.

Disadvantages

- **No standard query language:** Since the language depends on the platform that is used so there is no certain standard query language.
- **Unprofessional Graphs:** Graphs are very unprofessional for transactional-based systems.
- **Small User Base:** The user base is small which makes it very difficult to get support when running into a system.

4. What are the Applications of Graph Data Model?

Ans:

Applications

- Graph data models are very much used in fraud detection which itself is very much useful and important.
- It is used in Digital asset management which provides a scalable database model to keep track of digital assets.
- It is used in Network management which alerts a network administrator about problems in a network.

- It is used in Context-aware services by giving traffic updates and many more.
- It is used in Real-Time Recommendation Engines which provide a better user experience.

5. What is a relational database?

Ans :

Relational databases are a type of database that allows users to access data that is stored in various tables connected by a unique ID or "key." Using this key, users can unlock data entries related to that key on another table, to help with inventory management, shipping, and more. On relational database management systems (RDBMS), users can input SQL queries to retrieve the data needed for specific job functions.

In a relational database, each row in the table has a key. In the columns are data attributes. Each record has a value for each attribute, so users can understand the relationships between data entries for functions like product marketing, manufacturing, UX research, and more.

As an example, for a shoe store processing online orders, a relational database might have two tables with related data. In the first table, each record includes the customer's name, shipping address, email, and billing information, in columns. A key is assigned to each row. In the second table, that key is listed alongside the product ordered, quantity, size, color, and more. The two tables are related, and toggled to each other, with the key. When an order comes in, the key allows the warehouse to pull the correct product from the shelf and ship it to the customer.

6. What areApplications of NoSQL Databases?

Ans :

Applications of NoSQL Databases

(i) Data Mining : When it comes to data mining, NoSQL databases are useful in retrieving information for data mining uses. Particularly when it's about large amounts of data, NoSQL databases store data points in both structured and unstructured formats leading to efficient storage of big data.

(ii) Social Media Networking Sites: Social media is full of data, both structured and unstructured. A field that is loaded with tons of data to be discovered, social media is one of the most effective applications of NoSQL databases. From comments to posts, user-related information to advertising, social media marketing requires NoSQL databases to be implemented in certain ways to retrieve useful information that can be helpful in certain ways.

(iii) Software Development: The third application that we will be looking at is software development. Software development requires extensive research on users and the needs of the masses that are met through software development.

(iv) Integration Databases in NoSQL: Nowadays, an enormous amount of information is been made each second. This information is of different schemes-unstructured, structured, and semi-structured data. The variety and volume of this information can't be managed by traditional databases. Therefore, NoSQL frameworks have emerged, which is another age of database framework.

7. What are the Features of NoSQL Databases?

Ans :

Features

- **Schema Agnostic:** NoSQL Databases do not require any specific schema or s storage structure than traditional RDBMS.
- **Scalability:** NoSQL databases scale horizontally as data grows rapidly certain commodity hardware could be added and scalability features could be preserved for NoSQL.

- **Performance:** To increase the performance of the NoSQL system one can add a different commodity server than reliable and fast access of database transfer with minimum overhead.
- **High Availability:** In traditional RDBMS it relies on primary and secondary nodes for fetching the data, Some NoSQL databases use master place architecture.
- **Global Availability:** As data is replicated among multiple servers and clouds the data is accessible to anyone, this minimizes the latency period.

Aggregate Data Models

The term aggregate means a collection of objects that we use to treat as a unit. An aggregate is a collection of data that we interact with as a unit. These units of data or aggregates form the boundaries for ACID operation.

8. What is Column-Family Stores?

Ans:

Column-family Stores (Database)

Column-family databases store data in column families as rows. These rows have many columns associated with a particular row. Column families basically contain the group of correlated data which we can access together.

- Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns.
- Rows do not need to have the same columns, and columns can be added to any row at any time without having to add it to other rows.
- When a column consists of a map of columns, we have a super column. A super column consists of a name and a value which is a map of columns. Think of a super column as a container of columns.
- Some Column-family databases are -
  - Cassandra
  - Hbase
  - Hypertable
  - Amazon DynamoDB

9. What is graph based Data Model in NoSQL?

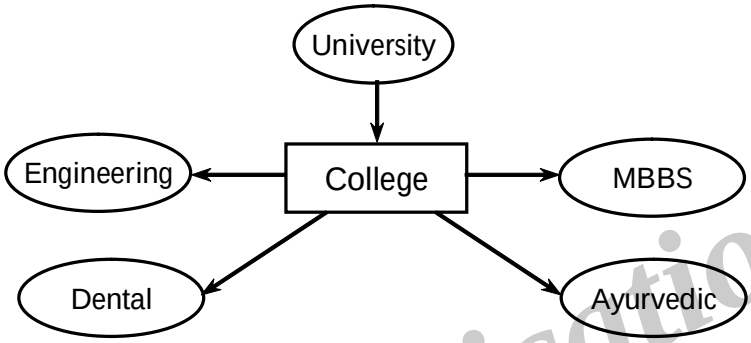
Ans :

Graph Based Data Model in NoSQL is a type of Data Model which tries to focus on building the relationship between data elements. As the name suggests Graph-Based Data Model, each element here is stored as a node, and the association between these elements is often known as Links. Association is stored directly as these are the first-class elements of the data model. These data models give us a conceptual view of the data.

These are the data models which are based on topographical network structure. Obviously, in graph theory, we have terms like Nodes, edges, and properties, let's see what it means here in the Graph-Based data model.

- **Nodes:** These are the instances of data that represent objects which is to be tracked.
- **Edges:** As we already know edges represent relationships between nodes.
- **Properties:** It represents information associated with nodes.

The below image represents Nodes with properties from relationships represented by edges.



10. What are the Major Benefits of Column Family Database?

Ans :

Four Major Benefits of Column Family Database:

- **Compression:** Column-based data storage stores data efficiently through data compression and by using data partitioning.
- **Aggregation Queries:** Due to the structure of the column family data structure, they perform particularly well with aggregation queries (such as SUM, COUNT, AVG etc).
- **Scalability:** Column databases are more scalable as compared to other databases. They are well suited for a data structure where data is spread on a large cluster of machines, often thousands of machines.
- **Fast to load and query:** Columnar stores can be loaded fast. A table containing millions of rows can be loaded within a few seconds. We can start querying and analyzing immediately on the loaded data.

Choose the Correct Answers

1. Which of the following is not a NoSQL database? [a]

   (a) SQL Server (b) MongoDB

   (c) Cassandra (d) None of the mentioned

2. Point out the correct statement. [d]

   (a) Documents can contain many different key-value pairs, or key-array pairs, or even nested documents

   (b) MongoDB has official drivers for a variety of popular programming languages and development environments

   (c) When compared to relational databases, NoSQL databases are more scalable and provide superior performance

   (d) All of the mentioned

3. Which of the following is a NoSQL Database Type? [b]

   (a) SQL (b) Document databases

   (c) JSON (d) All of the mentioned

4. Which of the following is a wide-column store? [a]

   (a) Cassandra (b) Riak

   (c) MongoDB (d) Redis

5. Point out the wrong statement. [a]

   (a) Non-Relational databases require that schemas be defined before you can add data

   (b) NoSQL databases are built to allow the insertion of data without a predefined schema

   (c) NewSQL databases are built to allow the insertion of data without a predefined schema

   (d) All of the mentioned

6. "Sharding" a database across many server instances can be achieved with _____. [b]

   (a) LAN (b) SAN

   (c) MAN (d) All of the mentioned

7. Most NoSQL databases support automatic _____ meaning that you get high availability and disaster recovery. [c]

   (a) processing (b) scalability

   (c) replication (d) all of the mentioned

8. Which of the following are the simplest NoSQL databases? [a]

(a) Key-value (b) Wide-column

(c) Document (d) All of the mentioned

9. _____ stores are used to store information about networks, such as social connections. [d]

(a) Key-value (b) Wide-column

(c) Document (d) Graph

10. NoSQL databases is used mainly for handling large volumes of _____ data. [a]

(a) unstructured (b) structured

(c) semi-structured (d) all of the mentioned

Fill in the Blanks

1. _____ stores are used to store information about networks, such as social connections.
2. NoSQL databases is used mainly for handling large volumes of _____ data.
3. _____ is the simplest NoSQL databases.
4. "Sharding" a database across many server instances can be achieved with _____.
5. _____ is not a NoSQL database.
6. NoSQL can be referred to as _____.
7. Which of the following represent column in NoSQL _____.
8. The core principle of NoSQL is _____.
9. NoSQL databases are most often referred to as _____ databases
10. _____ is a online NoSQL developed by Cloudera.

ANSWERS

1. Graph
2. unstructured
3. Key-value
4. SAN
5. SQL Server
6. Not Only SQL
7. Field
8. High availability
9. Network, Distributed and Object-oriented
10. Hbase

One Mark Answers

1. How many types of mechanism works in NoSQL? Write down their name?

Ans :

There are four types of mechanisms:

i) Graph database

ii) Key value calculation

iii) Document oriented and

iv) Column view presentation.

2. Write down the NoSQL's different features.

Ans :

It can store a big amount of unstructured, structured, and semi-structured data.

It is object-oriented programming based, which is best for a web application.

3. Can we use NoSQL in an Oracle-based database.

Ans :

Yes, NoSQL is applicable in the Oracle database to record data. This database helps to find out the data records through external table functions.

4. What is a hash table? How does it work in NoSQL?

Ans :

This is like a data structure that provides an associative array of abstract data types. This table uses to function in a complex database.

5. What is the meaning of document-oriented DB?

Ans :

This is one of the features of the NoSQL database. It helps to store the data as schema-free. As a result, JavaScript object notation will be used, and scalability will be higher.

6. What is a Graph database?

Ans :

A graph database is one of the most important of all databases. It is mainly specific for storing and navigating data relationships.

7. Explain the CAP theorem in NoSQL.

Ans :

It is the most reliable three guarantees for a database. CAP theorem is expertise with skills like consistency, availability, and partition tolerance.

8. What is the main target of NoSQL?

Ans :

Create an alternate database in SQL. It helps to store textual data in a database easily that is also in a non-structured format.

9. Name a few of the companies that are using NoSQL.

Ans :

Some Companies are:Google, Amazon, Netflix and Facebook

10. What is the main principle of NoSQL is?

Ans :

The main principle of NoSQL is to make the database high availability.

UNIT II

Distribution Models: Single Server, Sharding, Master-Slave Replication, Peer-to-Peer Replication, Combining Sharding and Replication. **Consistency:** Update Consistency, Read Consistency, Relaxing Consistency, Relaxing Durability, Quorums **Version Stamps:** Business and System Transactions, Version Stamps on Multiple **Nodes Map-Reduce:** Basic Map-Reduce, Partitioning and Combining, Composing Map-Reduce Calculations

2.1 Distribution Models

2.1.1 Single Server

Q1. Explain detail about single server in distribution models?

Ans : **(Imp.)**

The primary driver of interest in NoSQL has been its ability to run databases on a large cluster. As data volumes increase, it becomes more difficult and expensive to scale up buy a bigger server to run the database on. A more appealing option is to scale outrun the database on a cluster of servers. Aggregate orientation fits well with scaling out because the aggregate is a natural unit to use for distribution

Depending on your distribution model, you can get a data store that will give you the ability to handle larger quantities of data, the ability to process a greater read or write traffic, or more availability in the face of network slowdowns or breakages. These are often important benefits, but they come at a cost. Running over a cluster introduces complexity-so it's not something to do unless the benefits are compelling.

Broadly, there are two paths to data distribution: replication and sharding. Replication takes the same data and copies it over multiple nodes. Sharding puts different data on different nodes. Replication and sharding are orthogonal techniques: You can use either or both of them. Replication comes into two forms: master-slave and peer-to-peer. We will now discuss these techniques starting at the simplest and working up to the more complex: first single-server, then master-slave replication, then sharding, and finally peer-to-peer replication.

Single Server

The first and the simplest distribution option is the one we would most often recommend no distribution at all. Run the database on a single machine that handles all the reads and writes to the data store. We prefer this option because it eliminates all the complexities that the other options introduce; it's easy for operations people to manage and easy for application developers to reason about.

Although a lot of NoSQL databases are designed around the idea of running on a cluster, it can make sense to use NoSQL with a single-server distribution model if the data model of the NoSQL store is more suited to the application. Graph databases are the obvious category here these work best in a single-server configuration. If your data usage is mostly about processing aggregates, then a single-server document or key-value store may well be worthwhile because it's easier on application developers.

2.1.2 Sharding

Q2. What is Sharding in distribution models?

Ans :

Sharding is a partitioning pattern for the NoSQL age. It's a partitioning pattern that places each partition in potentially separate servers potentially all over the world. This scale out works well for supporting people all over the world accessing different parts of the data set with performance.

Sharding Often, a busy data store is busy because different people are accessing different parts of the data set.In these circumstances we can support

horizontal scalability by putting different parts of the data onto different servers a technique that's called "sharding".

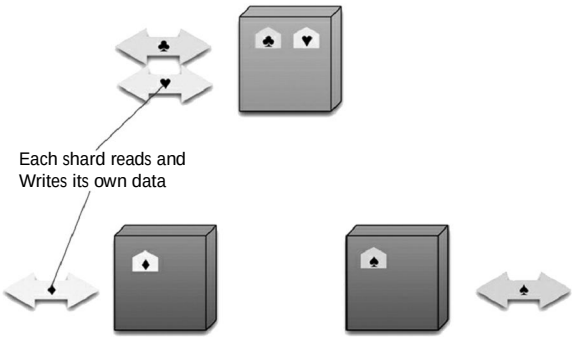


Fig: Sharding puts different data on separate nodes, each of which does its own reads and writes.

In the ideal case, we have different users all talking to different server nodes. Each user only has totalk to one server, so gets rapid responses from that server. The load is balanced out nicely between servers for example, if we have tenservers, each one only has to handle 10% of the load.

When it comes to arranging the data on the nodes, there are several factors that can help improve performance. If you know that most accesses of certain aggregates are based on a physical location, you can place the data close to where it's being accessed. If you have orders for some one who lives in Boston, you can place that data in your eastern US datacenter.

Another factor is trying to keep the load even. This means that you should try to arrange aggregates so they are evenly distributed across the nodes which all get equal amounts of the load. This may varyover time, for example if some data tends to be accessed on certain days of the weekso there maybe domain-specific rulesyou'dlike touse.

Historically most people have done shard in gas part of application logic. You might put all us to with surnames starting from A to

Don one shard and E to G on another. This complicates the programming model, as application code needs to ensure that queries are distributed across the various shards. Further more, rebalancing the sharding means changing the application code andmigrating the data. Many NoSQL databases offer auto-sharding, where the database takes on the responsibility of allocating data to shards and ensuring that data access goes to the right shard. This can make it much easier to use shard in ginan application.

Sharding is particularly valuable for performance because it can improve both read and write performance. Using replication, particularly with caching, can greatly improve read performance but does little for applications that have a lot of writes. Sharding provides a way to horizontally scale writes.

Sharding does little to improve resilience when used alone. Although the data is on different nodes,a node failure makes that shard's data unavailable just as surely as it does for a single server solution. The resilience benefits it does provide is that only the users of the data on that shard willsuffer however, it's not good to have a database with part of its data missing. With a single server it'seasier to pay the effort and cost to keep that server up and running; clusters usually try to use less reliable machines, and you're more likely to get a node failure. So in practice, sharding alone is likely to decrease resilience.

Despite the fact that sharding is made much easier with aggregates, it's still not a step to be taken lightly. Some databases are intended from the beginning to use sharding, in which case it's wise to runthem on a cluster from the very beginning of development, and certainly in production. Other databases use sharding as a deliberate step up from a single-server configuration, in which case it'sbest to start single-server and only use sharding once your load projections clearly indicate that you are runningoutofheadroom.

In any case the step from a single node to sharding is going to be tricky. We have heard tales ofteams getting into trouble because they left sharding to very late, so when they turned it on inproduction their database became essentially unavailable because the sharding support consumed all the database resources for moving the data onto new shards. The lesson here is to use sharding well.

2.1.3 Master-Slave Replication

Q3. Elaborate on Master-Slave Replication in distribution models?

Ans : **(Imp.)**

Master-slave NoSQL Data Replication. The master slave technique of NoSQL Data Replication creates a copy (master copy) of your database and maintains it as the key data source. Any updates that you may require are made to this master copy and later transferred to the slave copies.

With master-slave distribution, you replicate data across multiple nodes. One node is designated as the master, or primary. This master is the authoritative source for the data and is usually responsible for processing any updates to that data. The other nodes are slaves, or secondaries. A replication process synchronizes the slaves with the master.

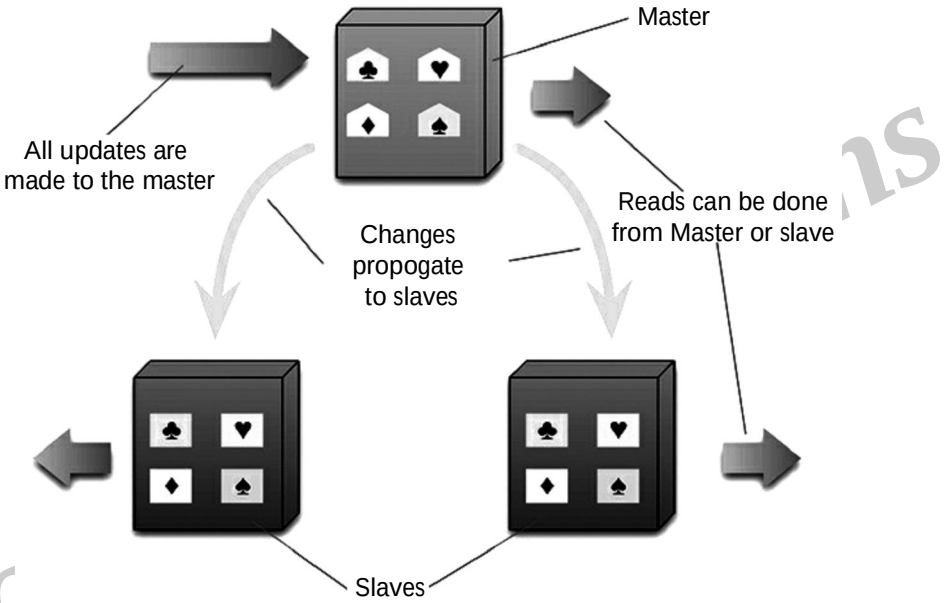


Fig.: Data is replicated from master to slaves. The master services all writes; reads may come from either master or slaves.

Master-slave replication is most helpful for scaling when you have a read-intensive dataset. You can scale horizontally to handle more read requests by adding more slave nodes and ensuring that all read requests are routed to the slaves. You are still, however, limited by the ability of the master to process updates and its ability to pass those updates on. Consequently it isn't such a good scheme for data sets with heavy write traffic, although off loading the read traffic will help a bit with handling the write load.

A second advantage of master-slave replication is read resilience: Should the master fail, the slaves can still handle read requests. Again, this is useful if most of your data access is reads. The failure of the master does eliminate the ability to handle writes until either the master is restored ora new master is appointed. However, having slaves as replicates of the master does speed up recovery after a failure of the monsters in slave can be appointed a new master very quickly.

The ability to appoint a slave to replace a failed master means that master-slave replication is useful even if you don't need to scale out. All read and write traffic can go to the master while the slave acts as a hot backup. In this case it's easiest to think of the system as a single-server store with a hot backup. You get the convenience of the single-server configuration but with greater resilience which is particularly handy if you want to be able to handle server failures gracefully.

Masters can be appointed manually or automatically. Manual appointing typically means that when you configure your cluster, you configure one node as the master. With automatic appointment, you create a cluster of nodes and they elect one of themselves to be the master. Apart from simpler configuration, automatic appointment means that the cluster can automatically appoint a new master when a master fails, reducing downtime.

In order to get read resilience, you need to ensure that the read and write paths into your application are different, so that you can handle a failure in the write path and still read. This includes such things as putting the reads and writes through separate database connections a facility that is not often supported by database interaction libraries. As with any feature, you cannot be sure you have read resilience without good tests that disable the writes and check that reads still occur.

Replication comes with some alluring benefits, but it also comes with an inevitable dark side inconsistency. You have the danger that different clients, reading different slaves, will see different values because the changes haven't all propagated to the slaves. In the worst case, that can mean that a client cannot read a write it just made. Even if you use master-slave replication just for hot backup this can be a concern, because if the master fails, any updates not passed on to the backup are lost.

2.1.4 Peer-to-Peer Replication

Q4. Explain about Peer-to-Peer Replication Distribution Models?

Ans :

The Peer-to-Peer NoSQL Data Replication works in the concept that every database copy is responsible to update its data. This can only work when every copy contains an identical format of schema and stores the same type of data. Furthermore, Database Restoration is a key requirement of this Data Replication technique.

Master-slave replication helps with read scalability but doesn't help with scalability of writes. Itprovides resilience against failure of a slave, but not of a master. Essentially, the master is still abottleneck and a single point of failure. Peer-to-peer replication attacks theseproblems by not having a master. All the replicas have equal weight, they can all accept writes, andthelossofanyofthemdoesn'tpreventaccesstothe datastore.

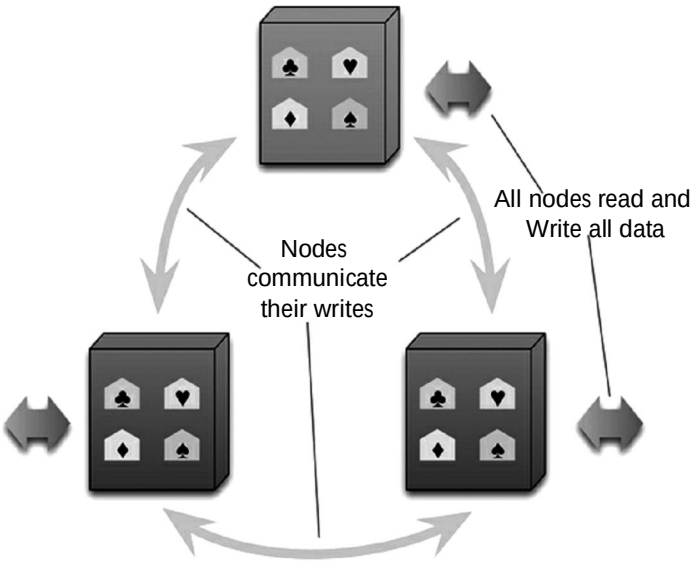


Fig.: Peer-to-peer replication has all nodes applying reads and writes to all the data

With a peer-to-peer replication cluster, you can ride over node failures without losing access to data. Further more, you can easily add nodes to improve your per formance. There's much to like here but there are complications.

The biggest complication is, again, consistency. When you can write to two different places, you run the risk that two people will attempt to update the same record at the same time a write write conflict. Inconsistencies on read lead to problems but at least they are relatively transient. Inconsistent writes are forever.

2.1.5 Combining Sharding and Replication

Q5. Explain in detail about Combining Sharding and Replication?

Ans: **(Imp.)**

Replication and sharding are strategies that can be combined. If we use both master-slave replication and sharding this means that we have multiple masters, but each data item only has a single master. Depending on your configuration, you may choose a node to be a master for some data and slaves for others, or you may dedicate nodes for master or slave duties.

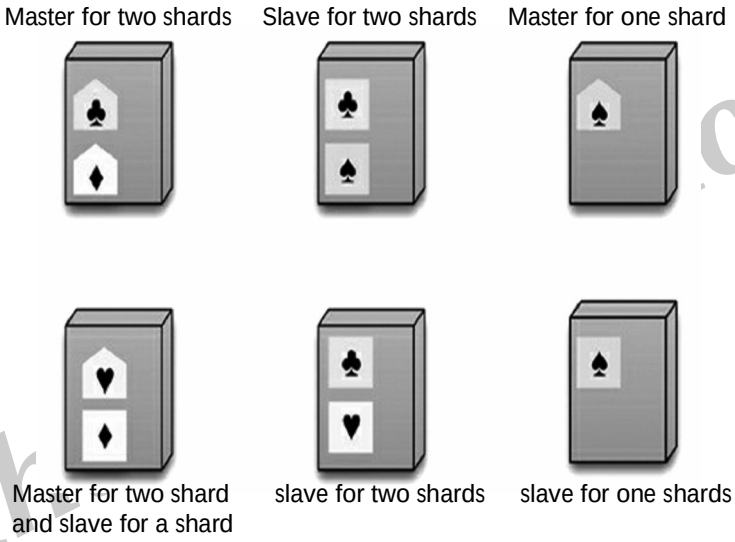


Fig.: Using master-slave replication together with sharding

Using peer-to-peer replication and sharding is a common strategy for column-family databases. In a scenario like this you might have tensor hundreds of nodes in a cluster with data sharded over them.

A good starting point for peer-to-peer replication is to have a replication factor of 3, so each shard is present on three nodes. Should a node fail, then the shards on that node will be built on the other nodes .

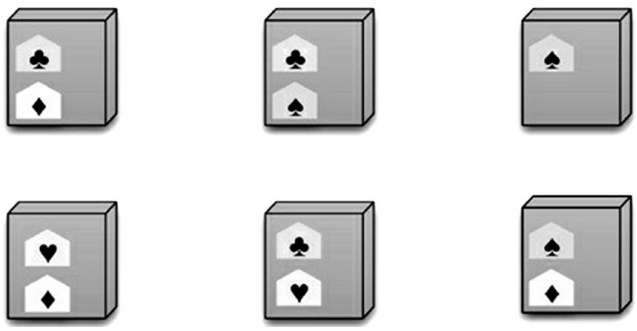


Fig.: Using peer-to-peer replication together with sharding

Key Points

There are two styles of distributing data:

- Sharding distributes different data across multiple servers, so each server acts as the single source for a subset of data.
- Replication copies data across multiple servers, so each bit of data can be found in multiple places.

A system may use either or both techniques.

Replication comes in two forms:

- Master-slave replication makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads.
- Peer-to-peer replication allows writes to any node; the nodes coordinate to synchronize their copies of the data.

Master-slave replication reduces the chance of update conflicts but peer-to-peer replication avoids loading all writes onto a single point of failure.

2.2 CONSISTENCY

2.2.1 Update Consistency

Q6. What is consistency? Explain in detail about Update Consistency.

Ans:

Meaning

Consistency can be simply defined by how the copies from the same data may vary within the same replicated database system. When the readings on a given data object are inconsistent with the last update on this data object, this is a consistency anomaly.

For many years, system architects would not compromise when it came to storing data and retrieving it. The ACID (Atomicity, Consistency, Isolation, and Durability) properties were the blue prints for every database management system. Therefore, strong consistency was not a choice. It was a requirement for all systems.

The Internet has grown to a point where billions of people have access to it, not only from a desktop but also from smartphones, smartwatches, and even other servers and services. Now-a-day's systems need to scale. The "traditional" monolithic database architecture, based on a powerful server, does not guarantees the high availability and network partition required by today's web-scale systems, as demonstrated by the CAP (Consistency, Availability, and Network Partition Tolerance) theorem. To achieve such requirements, systems cannot impose strong consistency.

Traditional relational database architectures usually have a single database instance responding to a few hundred clients. Relational databases implement the strongest consistency model, where each transaction must be immediately committed, and all clients will operate over valid data states. Reads from the same object will present the same value to all simultaneous client requests. Although strong consistency is the ideal requirement for a database, it deeply compromises horizontal-scalability. Horizontal scalability is a more affordable approach when compared to vertical scalability, for enabling higher throughput and the distribution/ replication of data across distinct database nodes. On the other hand, vertical scalability relies on a single powerful database server to store data and answer all requests. Although horizontal scaling may seem preferable, CAP theorem shows that when network partitions occur, one has to opt between availability and consistency.

To help solve this problem, NoSQL database systems have emerged. These systems have been created with a standard requirement in mind, scalability. Some NoSQL databases designers have chosen higher Availability over a more relaxed consistency strategy, an approach known as BASE (Basically Available, Soft-state and Eventually consistent). The most common NoSQL database systems can be organized into four categories, document databases, column databases, key-value stores, and graph databases.

There are also hybrid categories that mix multiple data models known has multi-model databases. In this work, our goal is to study how consistency is implemented over different non-cloud NoSQL databases. The designers of these database systems have devised different strategies to handle consistency, thus assuming variable trade offs between consistency and other quality attributes, such as availability, latency, and network partitioning tolerance we compare the consistency models

provided by five of the most popular non-cloud NoSQL database systems. One self-imposed constraint was to select at least one database of each sub-category: Key-value database (Redis); column database (Cassandra); document database (MongoDB), graph database (Neo4j), and multi-model database (Orient DB).

2.2.2 Read Consistency

Q7. What is Read Consistency?

(OR)

Explain in detail about Read Consistency

Ans :

Meaning

"Read Consistency" is one of the "Transaction Isolation" levels that describe how good concurrent transactions are isolated from each-other (i.e. in how far they can treat the database as if only they work on it) we can conclude that the probability of consistency over time is constant, resulting in 100%. That is because each write or read operation is executed on every node available before acknowledging the result to the client. Therefore, this configuration makes the Cassandra cluster strong consistent.

ONE Read Consistency Level and QUORUM

Write Consistency Level In, the consistency of a given data object eventually gets to 100%. A write operation needs three updated copies to acknowledge a successful write operation and a read operation returns the first copy the coordinator funds. The time that it is needed to reach 100% consistency is the time that the cluster needs to make all the number of copies previously set on the Replication Factor.

With Read Consistency Level ONE Cassandra will depend on the periodically Read Repair routines set by the Read Repair Chance to update all the copies of the data object and return all the time the same latest version.

QUORUM Read Consistency Level and ONE Write Consistency Level From: we can conclude that the time needed to reach full consistency of a given data object is the shortest of all configurations here (excluding the Figure 5 configuration). Three nodes are approached by the coordinator and the most updated version among them is returned. For each read operation, Cassandra cluster uses its Read Repair feature to propagate to all three nodes inside the Quorum (the three nodes), so that they all have the most updated version of the requested data among them. Because Read Repair is always triggered by a read, the cluster reaches full consistency faster on the given data object.

ONE Read Consistency Level and ONE Write Consistency Level: we have the strongest form of eventual consistency configuration in Cassandra. We need just one node with the updated data to acknowledge the write operation. For the reads, the first node the coordinator node chooses will retrieve the requested data. This may or may not be the most updated version of the data object. Eventually, the most updated version will be returned on all requests. The time needed to get to a 100% probability of consistency will depend on the Read Repair Chance and the Replication Factor. The higher the probability of the Read Repair Chance, the shorter the time to get to full consistency. The lower the Replication Factor, the shorter the time to get to full consistency. Modifying the Read Repair Chance and the Replication Factor to reach consistency faster will result in higher latencies because more copies and nodes are involved in the read and write operations for each client request.

2.2.3 Relaxing Consistency

Q8. What is Relaxing Consistency in NoSQL?

Ans :

Distributed systems can be highly available and durable. It's possible for data to be inconsistent; a query might return old or stale data. It might this phenomenon described as being eventually consistent.

Data analysis has replaced data acquisition as the bottleneck to evidence-based decision making we are drowning in it. Extracting knowledge from large, heterogeneous, and noisy datasets requires not only powerful computing resources, but the programming abstractions to use them effectively. The abstractions that emerged in the last decade blend ideas from parallel databases, distributed systems, and programming languages to create a

new class of scalable data analytics platforms that form the foundation for data science at realistic scales. In this course, you will learn the landscape of relevant systems, the principles on which they rely, their tradeoffs, and how to evaluate their utility against your requirements.

We will learn how practical systems were derived from the frontier of research in computer science and what systems are coming on the horizon. Cloud computing, SQL and NoSQL databases, Map Reduce and the ecosystem it spawned, Spark and its contemporaries, and specialized systems for graphs and arrays will be covered. we will also learn the history and context of data science, the skills, challenges, and methodologies the term implies, and how to structure a data science project.

Learning Goals

1. Describe common patterns, challenges, and approaches associated with data science projects, and what makes them different from projects in related fields.
2. Identify and use the programming models associated with scalable data manipulation, including relational algebra, MapReduce, and other data flow models.
3. Use database technology adapted for large-scale analytics, including the concepts driving parallel databases, parallel query processing, and in-database analytics
4. Evaluate key-value stores and NoSQL systems, describe their tradeoffs with comparable systems, the details of important examples in the space, and future trends.
5. "Think" in MapReduce to effectively write algorithms for systems including Hadoop and Spark. You will understand their limitations, design details, their relationship to databases, and their associated ecosystem of algorithms, extensions, and languages. write programs in Spark.
6. Describe the landscape of specialized Big Data systems for graphs, arrays, and streams.

2.2.4 Relaxing Durability

Q9. What is relaxing durability in NoSQL?

Ans:

Relaxed-durability databases trade the full durability of committed transactions for enhanced runtime performance for transactional workloads. A relaxed-durability database created with the non recovery level is similar to an in-memory database: you cannot recover data or logs if the server terminates or is shut down.

In-memory databases reside entirely in cache and do not use disk storage for data or logs, and therefore do not require disk I/O. This results in potentially better performance than a traditional disk-resident database, as well as other advantages. However, since an in-memory database exists only in cache, you cannot recover the database if the supporting host is shut down or the database fails.

With relaxed-durability databases, Adaptive Server extends the performance benefits of an in-memory database to disk-resident databases. Disk-resident databases perform writes to disk, and ensure that the transactional ACID (atomicity, consistency, integrity, and durability) properties are maintained. A traditional disk-resident database operates at full durability to guarantee transactional recovery from a server failure. Relaxed-durability databases trade the full durability of committed transactions for enhanced runtime performance for transactional workloads. A relaxed-durability database created with the non recovery level is similar to an in-memory database: you cannot recover data or logs if the server terminates or is shut down. You can also create a relaxed-durability database with the at shutdown level, where transactions are written to disk if there is a proper shutdown of the database.

Replication Server Support

Replication Server supports as the replicate database the in-memory databases and relaxed-durability databases set with durability at nonrecovery.

Minimal DML Logging and Replication

To optimize the log records that are flushed to the transaction log on disk, Adaptive Server can perform minimal to no logging when executing

some data manipulation language (DML) commands: insert, update, delete, and slow upon all types of low-durability databases, such as in-memory databases and relaxed-durability databases set with durability of at shutdown or nonrecovery.

2.2.5 Quorums

Q10. Explain in detail about Quorum?

Ans :

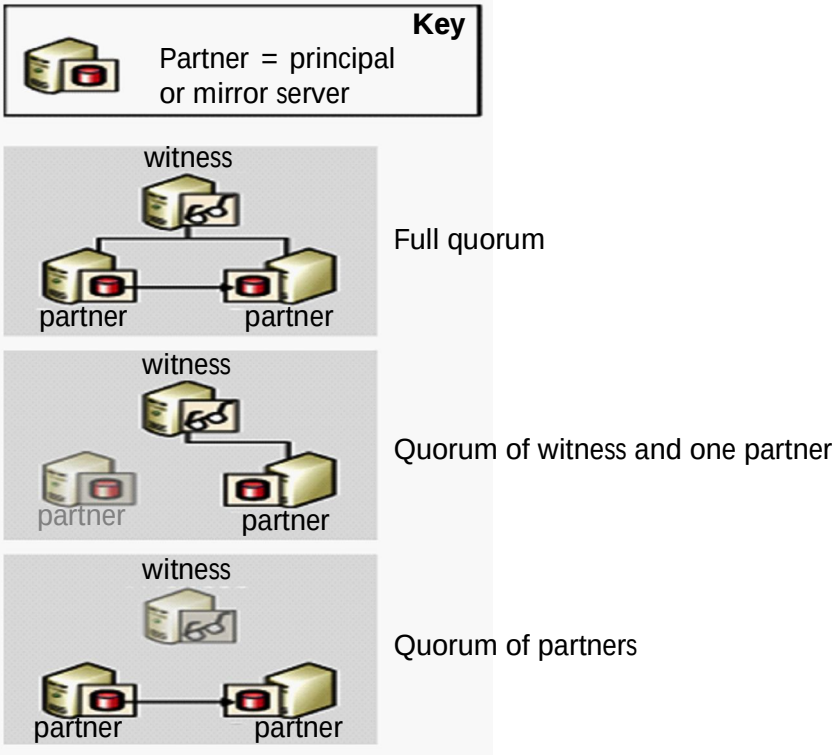
Quorum is a relationship that exists when two or more server instances in a database mirroring session are connected to each other. Typically, quorum involves three interconnected server instances. When a witness is set, quorum is required to make the database available.

Whenever a witness is set for a database mirroring session, *quorum* is required. Quorum is a relationship that exists when two or more server instances in a database mirroring session are connected to each other. Typically, quorum involves three interconnected server instances. When a witness is set, quorum is required to make the database available. Designed for high-safety mode with automatic fail over, quorum makes sure that a database is owned by only one partner at a time.

If a particular server instance becomes disconnected from a mirroring session, that instance loses quorum. If no server instances are connected, the session loses quorum and the database becomes unavailable. Three types of quorum are possible:

- A *full quorum* includes both partners and the witness.
- A *witness-to-partner quorum* consists of the witness and either partner.
- A *partner-to-partner quorum* consists of the two partners.

The following figure shows these types of quorum.



As long as the current principal server has quorum, this server owns the role of principal and continues to serve the database, unless the database owner performs a manual failover. If the principal server loses quorum, it stops serving the database. Automatic failover can occur only if the principal database has lost quorum, which guarantees that it is no longer serving the database.

A disconnected server instance saves its most recent role in the session. Typically, a disconnected server instance reconnects to the session when it restarts and regains quorum.

Quorum in High-Safety Mode Sessions

In high-safety mode, quorum allows automatic failover by providing a context in which the server instances with quorum arbitrate which partner owns the role of principal. The principal server serves the database if it has quorum. If the principal server loses quorum when the synchronized mirror server and witness retain quorum, automatic failover occurs.

The quorum scenarios for high-safety mode are as follows:

- A full quorum that consists of both partners and the witness.

  Ordinarily, all three server instances participate in a three-way quorum, called a full quorum. With a full quorum, the principal and mirror servers continue to perform their respective roles (unless manual failover occurs).

- A witness-to-partner quorum that consists of the witness and either partner.

  If the network connection between the partners is lost because one of the partners has been lost, the following cases are possible:

- The mirror server is lost, and the principal server and witness retain quorum.

  In this case, the principal sets its database to DISCONNECTED and runs with mirroring in a SUSPENDED state. (This is referred to as running exposed, because the database is currently not being mirrored.) When the mirror server rejoins the session, the server regains quorum as mirror and starts resynchronizing its copy of the database.

- The principal server is lost, and the witness and the mirror server retain quorum.

  In this case, automatic failover occurs. For more information, see Database Mirroring Operating Modes.

- All the server instances lose quorum, but subsequently the mirror and witness reconnect. The database will not be served in this case.

Rarely, the network connection between failover partners is lost while both partners remain connected to the witness. In this event, two, separate witness-to-partner quorums exist, with the witness as a liaison. The witness informs the mirror server that the principal server is still connected. Therefore, automatic failover does not occur. Instead, the mirror server retains the mirror role and waits to reconnect to the principal. If the redo queue contains log records at this point, the mirror server continues to roll forward the mirror database. On reconnecting, the mirror server will resynchronize the mirror database.

- A *partner-to-partner quorum* that consists of the two partners.

  As long as the partners retain quorum, the database continues in a SYNCHRONIZED state, and manual failover remains possible. Without the witness, automatic failover is not possible; but when the witness regains quorum, the session resumes regular operation, and automatic failover is supported again.

  The session loses quorum:

  If all the server instances become disconnected from each other, the session is said to have *lost quorum*. As server instances reconnect to each other, they regain quorum with each other.

- If the principal server reconnects with either of the other server instances, the database becomes available.

- If the principal server remains disconnected, but the mirror and witness reconnect to each other, automatic failover cannot occur

because data loss might occur. Therefore, the database remains unavailable, until the principal server rejoins the session.

- When all three server instances have reconnected, full quorum is regained, and the session resumes its regular operation.

2.3 VERSION STAMPS

2.3.1 Business and System Transactions

Q11. Explain Business and system Transactions with an example?

Ans : **(Imp.)**

A NoSQL originally referring to non-SQL or nonrelational is a database that provides a mechanism for storage and retrieval of data. In this article, we will see NoSQL transactions. There are some features of NoSQL:

- It has the feature of Horizontal Scaling.
- The main advantage of using NoSQL is that it is easy to use for developers.
- NoSQL has very flexible schemas.

Types of NoSQL Database

- **Key-Value Databases:** They contain only keys and values which is why they are called simpler types of databases.
- **Document Database:** In these databases, they contain fields and values and they always store the document similar to JSON.
- **Wide Column Stores:** These are the special NoSQL databases because they store data in form of tables, rows, and columns.
- **Graph Databases:** Here are Nodes and Edges present where nodes are used for storing information about places, things, and people also. while edges are used for storing information regarding the relationship between the nodes.

Transactions in NoSQL

Here transactional semantics are described in terms of ACID properties:

- **Atomicity:** It means the transaction can either be completed or fails completely. There is not a partial completion of the transaction in it.
- **Consistency:** The database always is in a consistent state while the transactions are at the beginning and end. The data should be consistent when the transaction begins and ends.
- **Isolation:** Isolation is the main property as it means when two transactions are being held at the same time we get the same result when the transaction is executed in sequence or the transaction is executed in parallel.
- **Durability:** When the transaction is being ended it cannot be reversed as stated by this property of transaction. It also states that any changes in transactions should be saved.

NoSQL database maintains all these properties, Here in Consistency database provides various different consistency policies. At one end application can specify the absolute consistency which actually guarantees that all reads return the most recently written value for a given designated key. wherever at the other end application has the capability to tolerate the inconsistent data and can specify weak consistency.

Isolation's main aim is to ensure that an operation is independent of other concurrent operations and even has the capability of optimistic/ pessimistic locking.

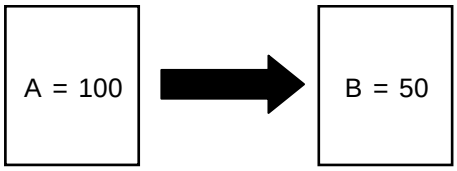
Durability mainly tells us that data is stored in a case when the system failure occurs. In durability the data is durable enough if the disk fails the data can be stored in any other case memory even the memory sometimes can be crashed but if data is durable the data can be saved anywhere and can be fetched when it is to be used.

Importance of ACID Transactions

These all acid transactions we read above have to ensure the highest possibility for reliability and integrity of data. These acid transactions ensure that our data never falls into an inconsistent state because the operation or task is only partially completed or cannot be completed. ACID properties provide correctness and consistency to our database.

Working of ACID Transactions:

We discuss the working of ACID transactions by taking the example :



We have to send 50 from S1 to S2 so we have to know here how the ACID transaction works in such an operation:

1. Atomicity

The atomicity ensures that the full operation takes place or is aborted back. It ensures that both operations take place or neither even one cannot take place.

- First operation **S**1: A – 50;
- Second operation S2: B + 50;

2. Consistency

Consistency property ensures that the integrity of transactions is maintained properly. Such as:

- Before transaction S1: A + B = 150. (A = 100, B = 50).
- After transaction: S2: A + B = 150. (A = 50, B = 100).

3. Isolation

Isolation ensures that if the two transactions are going on no other transaction can be accessed between them. It means two operations can be done at one time and no other operation or task can interfere with them.

Such as if two operations such as A – 50 and B + 50 are working no third transaction can interfere with them.

4. Durability

As we discussed above one time it completes the operation cannot be reversed after it. If all transactions are completed it cannot go back to its previous steps.

2.3.2 Version Stamps on Multiple Nodes

Q12. What are Version Stamps on Multiple Nodes in No SQL Database?

Ans : **(Imp.)**

The basic version stamp works well when you have a single authoritative source for data, such as a single server or master-slave replication. In that case the version stamp is controlled by the master. Any slaves follow the master's stamps. But this system has to be enhanced in a peer-to-peer distribution model because there's no longer a single place to set the version stamps.

If you're two nodes for some data, you run into the chance that they may give you different answers. If this happens, your reaction may vary depending on the cause of that difference. It may be that an update has only reached one node but not the other, in which case you can accept the latest (assuming you can tell which one that is). Alternatively, you may have run into an inconsistent update, in which case you need to decide how to deal with that. In this situation, a simple GUID, since these don't tell you enough about the relationships.

The simplest form of version stamp is a counter. Each time a node updates the data, it increments the counter and puts the value of the counter into the version stamp. If you have blue and green slave replicas of a single master, and the blue node answers with a version stamp of 4 and the green node with 6, you know that the green's answer is more recent.

In multiple-master cases, we need something fancier. One approach, used by distributed version control systems, is to ensure that all nodes contain a history of version stamps. That way you can see if the blue node's answer is an ancestor of the green's answer. This would either require the clients to hold onto version stamp histories, or the server nodes to keep version stamp histories and include them when asked for data. This also detects an inconsistency, which we would see if we get two version stamps and neither of them has the other in their histories. Although version control systems keep these kinds of histories, they aren't found in NoSQL databases.

A simple but problematic approach is to use timestamps. The main problem here is that it's usually difficult to ensure that all the nodes have a consistent notion of time, particularly if updates can happen rapidly. Should a node's clock get out of sync, it can cause all sorts of trouble. In addition, you can't detect write-write conflicts with timestamps, so it would only work well for the single-master case- and then a counter is usually better.

The most common approach used by peer-to-peer NoSQL systems is a special form of version stamp which we call a vector stamp. In essence, a vector stamp is a set of counters, one for each node. A vector stamp for three nodes (blue, green, black) would look something like [blue: 43, green: 54, black: 12]. Each time a node has an internal update, it updates its own counter, so an update in the green node would change the vector to [blue: 43, green: 55, black: 12]. Whenever two nodes communicate, they synchronize their vector stamps. There are several variations of exactly how this synchronization is done. We're coining the term "vector stamp" as a general term in this book; you'll also come across vector clocks and version vectors—these are specific forms of vector stamps that differ in how they synchronize.

By using this scheme, you can tell if one version stamp is newer than another because the newer stamp will have all its counters greater than or equal to those in the older stamp. So [blue: 1, green: 2, black: 5] is newer than [blue:1, green: 1, black 5] since one of its counters is greater. If both stamps have a counter greater than the other, e.g. [blue: 1, green: 2, black: 5] and [blue: 2, green: 1, black: 5], then you have a write-write conflict.

There may be missing values in the vector, in which case we use treat the missing value as 0. So [blue: 6, black: 2] would be treated as [blue: 6, green: 0, black: 2]. This allows you to easily add new nodes without invalidating the existing vector stamps.

Vector stamps are a valuable tool that spots inconsistencies, but doesn't resolve them. Any conflict resolution will depend on the domain you are working in. This is part of the consistency/latency tradeoff. You either have to live with the fact that network partitions may make your system unavailable, or you have to detect and deal with inconsistencies.

Key Points

- Version stamps help you detect concurrency conflicts. When you read data, then update it, you can check the version stamp to ensure nobody updated the data between your read and write.
- Version stamps can be implemented using counters, GUIDs, content hashes, times tamps, or a combination of these.
- With distributed systems, a vector of version stamps allows you to detect when different nodes have conflicting updates.

2.4 MAP-REDUCE

2.4.1 Basic Map-Reduce

Q13. What is Basic Map-Reduce? Explain in detail how the data is processing?

Ans: **(Imp.)**

Meaning

A MapReduce is a data processing tool which is used to process the data parallelly in a distributed form. It was developed in 2004, on the basis of paper titled as "MapReduce: Simplified Data Processing on Large Clusters," published by Google.

The MapReduce is a paradigm which has two phases, the mapper phase, and the reducer phase. In the Mapper, the input is given in the form of a key-value pair. The output of the Mapper is fed to the reducer as input. The reducer runs only after the Mapper is over. The reducer too takes input in key-value format, and the output of reducer is the final output.

Steps in Map Reduce

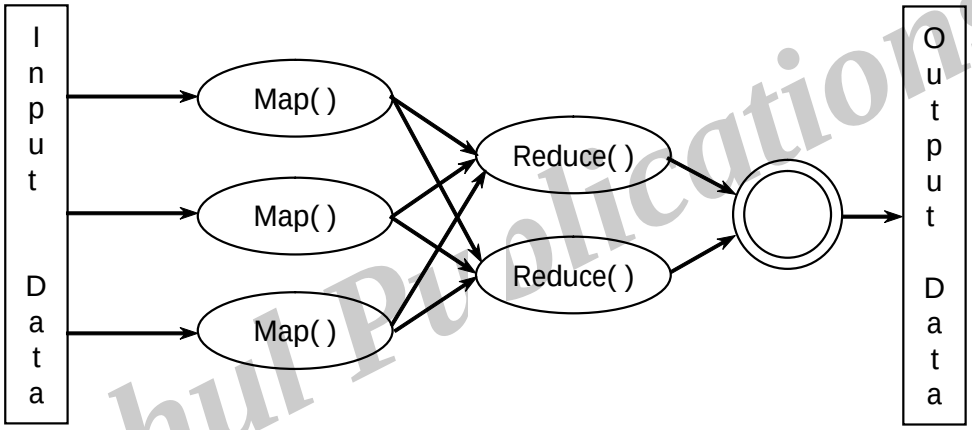
- The map takes data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
- Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these lists of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.

- An output of sort and shuffle sent to the reducer phase. The reducer performs a defined function on a list of values for unique keys, and Final output <key, value> will be stored/displayed.

MapReduce Algorithm

MapReduce paradigm is based on sending map-reduce programs to computers where the actual data resides.

- During a MapReduce job, Hadoop sends Map and Reduce tasks to appropriate servers in the cluster.
- The framework manages all the details of data-passing like issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on the nodes with data on local disks that reduces the network traffic.
- After completing a given task, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

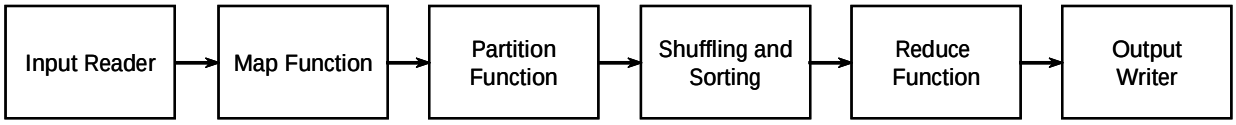


Usage

- It can be used in various application like document clustering, distributed sorting, and web link-graph reversal.
- It can be used for distributed pattern-based searching.
- We can also use MapReduce in machine learning.
- It was used by Google to regenerate Google's index of the World Wide Web.
- It can be used in multiple computing environments such as multi-cluster, multi-core, and mobile environment

Data Flow in MapReduce

MapReduce is used to compute the huge amount of data . To handle the upcoming data in a parallel and distributed form, the data has to flow from various phases.



Phases of MapReduce data flow

1. Input reader

The input reader reads the upcoming data and splits it into the data blocks of the appropriate size (64 MB to 128 MB). Each data block is associated with a Map function.

Once input reads the data, it generates the corresponding key-value pairs. The input files reside in HDFS.

2. Map function

The map function processes the upcoming key-value pairs and generated the corresponding output key-value pairs. The map input and output type may be different from each other.

3. Partition function

The partition function assigns the output of each Map function to the appropriate reducer. The available key and value provide this function. It returns the index of reducers.

4. Shuffling and Sorting

The data are shuffled between/within nodes so that it moves out from the map and get ready to process for reduce function. Sometimes, the shuffling of data can take much computation time.

The sorting operation is performed on input data for Reduce function. Here, the data is compared using comparison function and arranged in a sorted form.

5. Reduce function

The Reduce function is assigned to each unique key. These keys are already arranged in sorted order. The values associated with the keys can iterate the Reduce and generates the corresponding output.

6. Output writer

Once the data flow from all the above phases, Output writer executes. The role of Output writer is to write the Reduce output to the stable storage.

2.4.2 Partitioning and Combining

Q14. Explain about Partitioning and Combining with an example?

Ans : **(Imp.)**

A partitioner works like a condition in processing an input dataset. The partition phase takes place after the Map phase and before the Reduce phase.

The number of partitioners is equal to the number of reducers. That means a partitioner will divide the data according to the number of reducers. Therefore, the data passed from a single partitioner is processed by a single Reducer.

Partitioner

A partitioner partitions the key-value pairs of intermediate Map-outputs. It partitions the data using a user-defined condition, which works like a hash function. The total number of partitions is same as the number of Reducer tasks for the job. Let us take an example to understand how the partitioner works.

MapReduce Partitioner Implementation

For the sake of convenience, let us assume we have a small table called Employee with the following data. We will use this sample data as our input dataset to demonstrate how the partitioner works.

Id	Name	Age	Gender	Salary
1201	gopal	45	Male	50,000
1202	manisha	40	Female	50,000
1203	khalil	34	Male	30,000
1204	prasanth	30	Male	30,000
1205	kiran	20	Male	40,000
1206	laxmi	25	Female	35,000
1207	bhavya	20	Female	15,000
1208	reshma	19	Female	15,000
1209	kranthi	22	Male	22,000
1210	Satish	24	Male	25,000
1211	Krishna	25	Male	25,000
1212	Arshad	28	Male	20,000
1213	lavanya	18	Female	8,000

We have to write an application to process the input dataset to find the highest salaried employee by gender in different age groups (for example, below 20, between 21 to 30, above 30).

Input Data

The above data is saved as input.txt in the "/home/hadoop/hadoopPartitioner" directory and given as input.

1201	gopal	45	Male	50000
1202	manisha	40	Female	51000
1203	khaleel	34	Male	30000
1204	prasanth	30	Male	31000
1205	kiran	20	Male	40000
1206	laxmi	25	Female	35000
1207	bhavya	20	Female	15000
1208	reshma	19	Female	14000
1209	kranthi	22	Male	22000
1210	Satish	24	Male	25000
1211	Krishna	25	Male	26000
1212	Arshad	28	Male	20000
1213	lavanya	18	Female	8000

Based on the given input, following is the algorithmic explanation of the program.

Map Tasks

The map task accepts the key-value pairs as input while we have the text data in a text file. The input for this map task is as follows:

Input

The key would be a pattern such as "any special key + filename + line number" (example: key = @input1) and the value would be the data in that line (example: value = 1201 \t gopal \t 45 \t Male \t 50000).

Method

The operation of this map task is as follows:

- Read the **value** (record data), which comes as input value from the argument list in a string.
- Using the split function, separate the gender and store in a string variable.

```
String[] str = value.toString().split("\t", -3);
String gender=str[3];
```

- Send the gender information and the record data **value** as output key-value pair from the map task to the **partition task**.

```
context.write(new Text(gender), new Text(value));
```

Output

Finally, you will get a set of key-value pair data in three collections of different age groups. It contains the max salary from the Male collection and the max salary from the Female collection in each age group respectively.

After executing the Map, the Partitioner, and the Reduce tasks, the three collections of key-value pair data are stored in three different files as the output.

All the three tasks are treated as MapReduce jobs. The following requirements and specifications of these jobs should be specified in the Configurations:

- Job name
- Input and Output formats of keys and values
- Individual classes for Map, Reduce, and Partitioner tasks

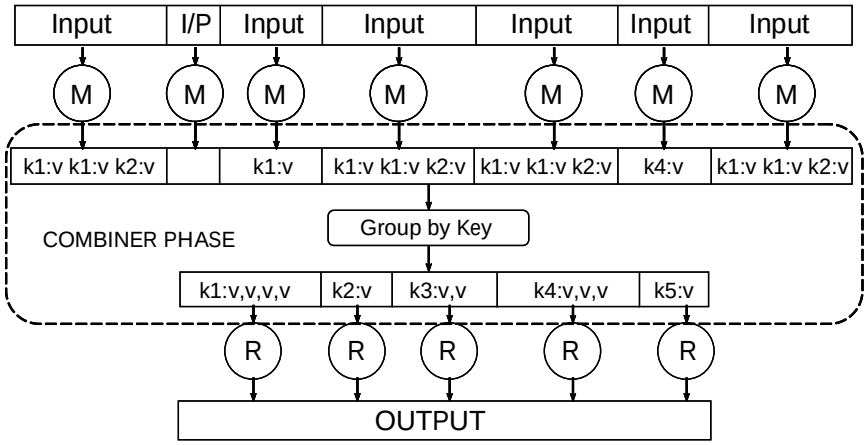
A Combiner, also known as a **semi-reducer,** is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class.

The main function of a Combiner is to summarize the map output records with the same key. The output (key-value collection) of the combiner will be sent over the network to the actual Reducer task as input.

Combiner

The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.

The following MapReduce task diagram shows the COMBINER PHASE.



Working of Combiner

Here is a brief summary on how MapReduce Combiner works:

- A combiner does not have a predefined interface and it must implement the Reducer interface's reduce() method.
- A combiner operates on each map output key. It must have the same output key-value types as the Reducer class.
- A combiner can produce summary information from a large dataset because it replaces the original Map output.
- Although, Combiner is optional yet it helps segregating data into multiple groups for Reduce phase, which makes it easier to process.

MapReduce Combiner Implementation

The following example provides a theoretical idea about combiners. Let us assume we have the following input text file named **input.txt** for MapReduce.

What do you mean by Object

What do you know about Java

What is Java Virtual Machine

How Java enabled High Performance

The important phases of the MapReduce program with Combiner are shown below.

Record Reader

This is the first phase of MapReduce where the Record Reader reads every line from the input text file as text and yields output as key-value pairs.

Input: Line by line text from the input file.

Output: Forms the key-value pairs. The following is the set of expected key-value pairs.

<1, What do you mean by Object>

<2, What do you know about Java>

<3, What is Java Virtual Machine>

<4, How Java enabled High Performance>

Map Phase

The Map phase takes input from the Record Reader, processes it, and produces the output as another set of key-value pairs.

Input

The following key-value pair is the input taken from the Record Reader.

<1, What do you mean by Object>

<2, What do you know about Java>

<3, What is Java Virtual Machine>

<4, How Java enabled High Performance>

The Map phase reads each key-value pair, divides each word from the value using StringTokenizer, treats each word as key and the count of that word as value. The following code snippet shows the Mapper class and the map function.

Output

The expected output is as follows:

<What,1> <do,1> <you,1> <mean,1> <by,1> <Object,1>

<What,1> <do,1> <you,1> <know,1> <about,1> <Java,1>

<What,1> <is,1> <Java,1> <Virtual,1> <Machine,1>

<How,1> <Java,1> <enabled,1> <High,1> <Performance,1>

The Combiner phase reads each key-value pair, combines the common words as key and values as collection. Usually, the code and operation for a Combiner is similar to that of a Reducer. Following is the code snippet for Mapper, Combiner and Reducer class declaration.

Reducer Phase

The Reducer phase takes each key-value collection pair from the Combiner phase, processes it, and passes the output as key-value pairs. Note that the Combiner functionality is same as the Reducer.

Input

The following key-value pair is the input taken from the Combiner phase.

<What,1,1,1> <do,1,1> <you,1,1> <mean,1> <by,1> <Object,1>

<know,1> <about,1> <Java,1,1,1>

<is,1> <Virtual,1> <Machine,1>

<How,1> <enabled,1> <High,1> <Performance,1>

Output

The expected output from the Reducer phase is as follows:

<What,3> <do,2> <you,2> <mean,1> <by,1> <Object,1>

<know,1> <about,1> <Java,3>

<is,1> <Virtual,1> <Machine,1>

<How,1> <enabled,1> <High,1> <Performance,1>

Record Writer

This is the last phase of MapReduce where the Record Writer writes every key-value pair from the Reducer phase and sends the output as text.

Input

Each key-value pair from the Reducer phase along with the Output format.

Output

It gives you the key-value pairs in text format. Following is the expected output.

What	3
do	2
you	2
mean	1
by	1
Object	1
know	1
about	1
Java	3
is	1
Virtual	1
Machine	1
How	1
enabled	1
High	1
Performance	1

2.4.3 Composing Map-Reduce Calculations

Q15. Explain Composing Map-Reduce Calculation with an example?

Ans : **(Imp.)**

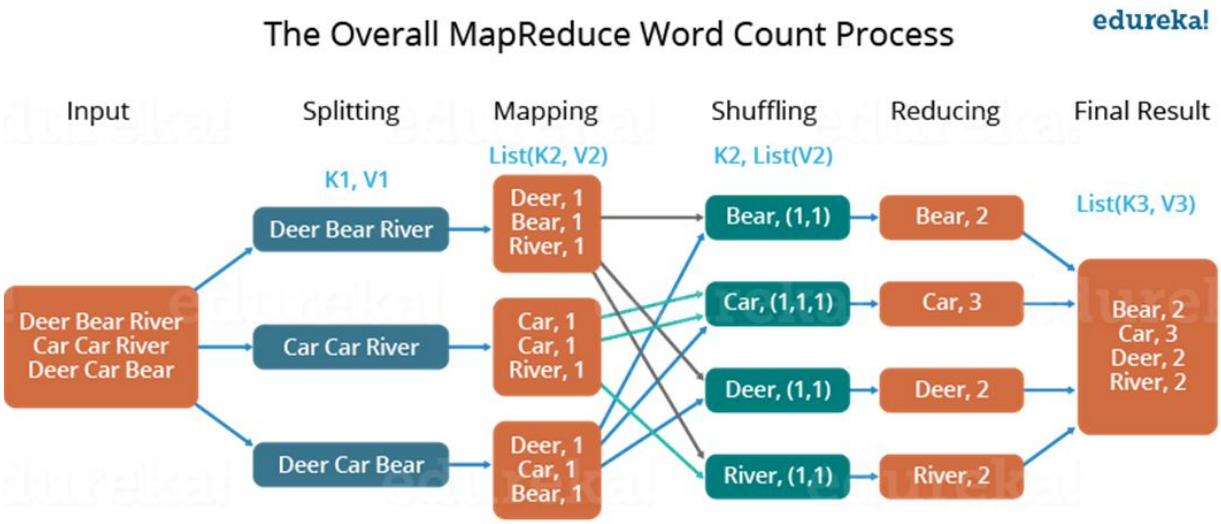
MapReduce is a programming model used for efficient processing in parallel over large data-sets in a distributed manner. The data is first split and then combined to produce the final result. The libraries for MapReduce is written in so many programming languages with various different-different optimizations

A Word Count Example of MapReduceCalculation:

How a MapReduce works by taking an example a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

Advantages of MapReduce

The two biggest advantages of MapReduce are:

1. Parallel Processing

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which helps us to process the data using different machines. As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount as shown in the figure below (2).

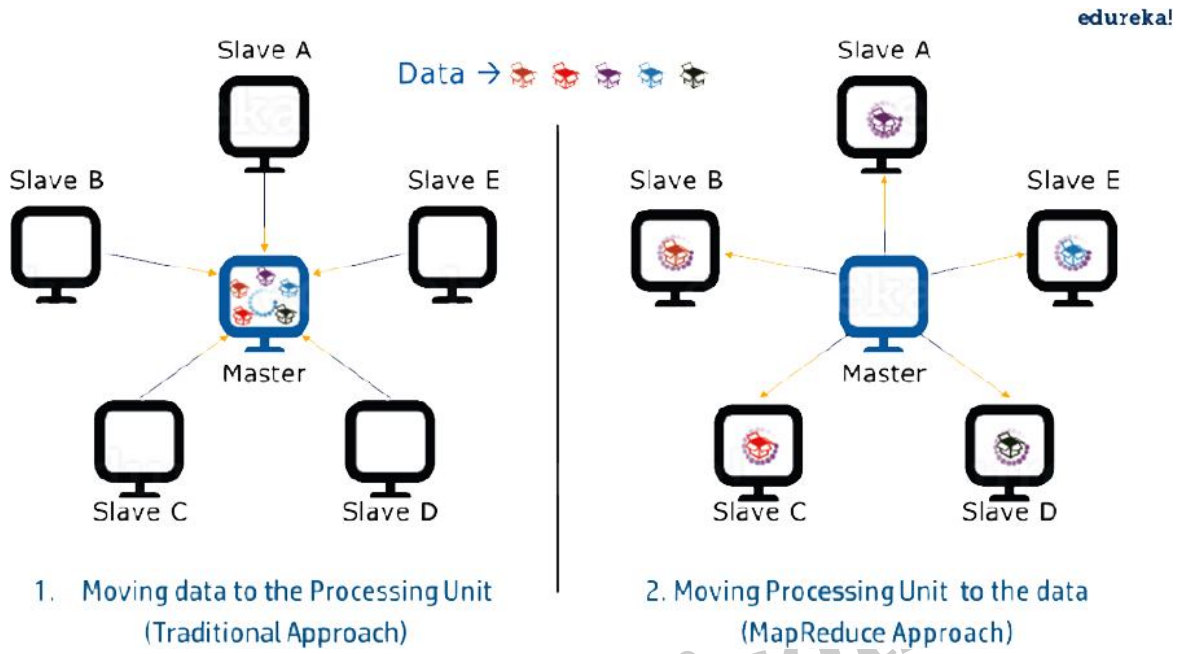


Fig.: Traditional Way Vs. MapReduce Way - MapReduce Tutorial

2. Data Locality

In Data Locality Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed the following issues:

- Moving huge data to processing is costly and deteriorates the network performance.
- Processing takes time as the data is processed by a single unit which becomes the bottleneck.
- The master node can get over-burdened and may fail.

Now, MapReduce allows us to overcome the above issues by bringing the processing unit to the data. So, as you can see in the above image that the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:

- It is very cost-effective to move processing unit to the data.
- The processing time is reduced as all the nodes are working with their part of the data in parallel.
- Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

Short Question & Answers

1. Explain about single server in distribution models?

Ans:

Single Server

The first and the simplest distribution option is the one we would most often recommendno distribution at all. Run the database on a single machine that handles all the reads and writes to the data store. We prefer this option because it eliminates all the complexities that the other options introduce; it's easy for operations people to manage and easy for application developers to reason about.

Although a lot of NoSQLdatabases are designed around the idea of running on a cluster, it can make sense to use NoSQLwith a single-server distribution model if the data model of the NoSQL store is more suited to the application. Graph databases are the obvious category herethese work best in a single-server configuration. If your data usage is mostly about processing aggregates, then a single-server document or key-value store may well be worthwhile because it's easier on application developers.

2. What are theAdvantages and Disadvantages of No SQL?

Ans:

Advantages of NoSQL

- Can be used as Primary or Analytic Data Source
- Big Data Capability
- No Single Point of Failure
- Easy Replication
- No Need for Separate Caching Layer
- It provides fast performance and horizontal scalability.
- Can handle structured, semi-structured, and unstructured data with equal effect
- Object-oriented programming which is easy to use and flexible
- NoSQL databases don't need a dedicated high-performance server
- Support Key Developer Languages and Platforms
- Simple to implement than using RDBMS
- It can serve as the primary data source for online applications.
- Handles big data which manages data velocity, variety, volume, and complexity
- Excels at distributed database and multi-data center operations
- Eliminates the need for a specific caching layer to store data
- Offers a flexible schema design which can easily be altered without downtime or service disruption

Disadvantages of NoSQL

- No standardization rules
- Limited query capabilities
- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn't work as well with relational data
- The learning curve is stiff for new developers
- Open source options so not so popular for enterprises.

3. ExplainMaster-Slave Replication in distribution models?

Ans:

Master-slave NoSQL Data Replication. The master-slave technique of NoSQL Data Replication creates a copy (master copy) of your database and

maintains it as the key data source. Any updates that you may require are made to this master copy and later transferred to the slave copies.

With master-slave distribution, you replicate data across multiple nodes. One node is designated as the master, or primary. This master is the authoritative source for the data and isusually responsible for processing any updates to that data. The other nodes are slaves, or secondaries. A replication processsynchronizestheslaveswiththe master.

4. What is consistency?

Ans:

Consistency can be simply deûned by how the copies from the same data may vary within thesame replicated database system. When the readings on a given data object are inconsistent withthe last update on this data object, this is a consistency anomaly.

For many years, system architects would not compromise when it came to storing data andretrieving it. The ACID (Atomicity, Consistency, Isolation, and Durability) properties were theblueprints for every database management system. Therefore, strong consistency was not a choice.It was a requirement for all systems.

The Internet has grown to a point where billions of people have access to it, not only froma desktop but also from smartphones, smartwatches, and even other servers and services. Nowaday's systems need to scale. The "traditional" monolithic database architecture, based on a powerful server,does not guarantees the high availability and network partition required by today's web-scale systems,as demonstrated by the CAP (Consistency, Availability, and Network Partition Tolerance) theorem. To achieve such requirements, systems cannot impose strong consistency.

5. What are the Importance of ACID Transactions?

Ans:

Importance of ACID Transactions

ACID transactions we read above have to ensure the highest possibility for reliability and integrity of data. These acid transactions ensure that our data never falls into an inconsistent state because the operation or task is only partially completed or cannot be completed. ACID properties provide correctness and consistency to our database.

Working of ACID Transactions

We discuss the working of ACID transactions by taking the example:

We have to send 50 from S1 to S2 so we have to know here how the ACID transaction works in such an operation:

1. **Atomicity:** The atomicity ensures that the full operation takes place or is aborted back. It ensures that both operations take place or neither even one cannot take place.
   - First operation **S**1: A – 50;
   - Second operation S2: B + 50;
2. **Consistency:** Consistency property ensures that the integrity of transactions is maintained properly. Such as:
   - Before transaction S1: A + B = 150. (A = 100, B = 50).
   - After transaction: S2: A + B = 150. (A = 50, B = 100).

3. **Isolation:** Isolation ensures that if the two transactions are going on no other transaction can be accessed between them. It means two operations can be done at one time and no other operation or task can interfere with them.

   Such as if two operations such as A – 50 and B + 50 are working no third transaction can interfere with them.

4. **Durability:** As we discussed above one time it completes the operation cannot be reversed after it. If all transactions are completed it cannot go back to its previous steps.

6. What are the Learning Goals of Relaxing Consistency?

Ans :

Learning Goals

1. Describe common patterns, challenges, and approaches associated with data science projects, and what makes them different from projects in related fields.
2. Identify and use the programming models associated with scalable data manipulation, including relational algebra, MapReduce, and other data flow models.
3. Use database technology adapted for large-scale analytics, including the concepts driving parallel databases, parallel query processing, and in-database analytics
4. Evaluate key-value stores and NoSQL systems, describe their tradeoffs with comparable systems, the details of important examples in the space, and future trends.
5. "Think" in MapReduce to effectively write algorithms for systems including Hadoop and Spark. You will understand their limitations, design details, their relationship to databases, and their associated ecosystem of algorithms, extensions, and languages. write programs in Spark.
6. Describe the landscape of specialized Big Data systems for graphs, arrays, and streams.

7. What is Basic Map-Reduce?

Ans:

MapReduce

A MapReduce is a data processing tool which is used to process the data parallelly in a distributed form.

The MapReduce is a paradigm which has two phases, the mapper phase, and the reducer phase. In the Mapper, the input is given in the form of a key-value pair. The output of the Mapper is fed to the reducer as input. The reducer runs only after the Mapper is over. The reducer too takes input in key-value format, and the output of reducer is the final output.

Steps in Map Reduce:

- The map takes data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
- Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these lists of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.
- An output of sort and shuffle sent to the reducer phase. The reducer performs a defined function on a list of values for unique keys, and Final output <key, value> will be stored/displayed.

8. What are the phases of data flow in MapReduce?

Ans :

Data Flow in MapReduce

MapReduce is used to compute the huge amount of data . To handle the upcoming data in a parallel and distributed form, the data has to flow from various phases.

Phases of MapReduce data flow

Input reader

The input reader reads the upcoming data and splits it into the data blocks of the appropriate

size (64 MB to 128 MB). Each data block is associated with a Map function.

Once input reads the data, it generates the corresponding key-value pairs. The input files reside in HDFS.

Map function

The map function processes the upcoming key-value pairs and generated the corresponding output key-value pairs. The map input and output type may be different from each other.

Partition function

The partition function assigns the output of each Map function to the appropriate reducer. The available key and value provide this function. It returns the index of reducers.

Shuffling and Sorting

The data are shuffled between/within nodes so that it moves out from the map and get ready to process for reduce function. Sometimes, the shuffling of data can take much computation time.

The sorting operation is performed on input data for Reduce function. Here, the data is compared using comparison function and arranged in a sorted form.

Reduce function

The Reduce function is assigned to each unique key. These keys are already arranged in sorted order. The values associated with the keys can iterate the Reduce and generates the corresponding output.

Output writer

Once the data flow from all the above phases, Output writer executes. The role of Output writer is to write the Reduce output to the stable storage.

9. What is Shuffling and Sorting in MapReduce?

Ans:

Shuffling and Sorting are two major processes operating simultaneously during the working of mapper and reducer.

The process of transferring data from Mapper to reducer is Shuffling. It is a mandatory operation for reducers to proceed their jobs further as the shuffling process serves as input for the reduce tasks.

In MapReduce, the output key-value pairs between the map and reduce phases (after the mapper) are automatically sorted before moving to the Reducer. This feature is helpful in programs where you need sorting at some stages. It also saves the programmer's overall time.

10. What are the key differences between Pig vs MapReduce?

Ans:

PIG is a data flow language, the key focus of Pig is manage the flow of data from input source to output store. As part of managing this data flow it moves data feeding it to

Process 1

Taking output and feeding it to,

Process 2

The core features are preventing execution of subsequent stages if previous stage fails, manages temporary storage of data and most importantly compresses and rearranges processing steps for faster processing. While this can be done for any kind of processing tasks Pig is written specifically for managing data flow of Map reduce type of jobs. Most if not all jobs in a Pig are map reduce jobs or data movement jobs. Pig allows for custom functions to be added which can be used for processing in Pig, some default ones are like ordering, grouping, distinct, count etc.

MapReduce on the other hand is a data processing paradigm, it is a framework for application developers to write code in so that its easily scaled to PB of tasks, this creates a separation between the developer that writes the application vs the developer that scales the application. Not all applications can be migrated to Map reduce but good few can be including complex ones like k-means to simple ones like counting unique in a dataset.

Choose the Correct Answers

1. A _____ node acts as the Slave and is responsible for executing a Task assigned to it by the Job Tracker. [c]

   (a) MapReduce (b) Mapper

   (c) TaskTracker (d) JobTracker

2. Point out the correct statement. [a]

   (a) MapReduce tries to place the data and the compute as close as possible

   (b) Map Task in MapReduce is performed using the Mapper() function

   (c) Reduce Task in MapReduce is performed using the Map() function

   (d) All of the mentioned

3. _____ part of the MapReduce is responsible for processing one or more chunks of data and producing the output results. [a]

   (a) Maptask (b) Mapper

   (c) Task execution (d) All of the mentioned

4. _____ function is responsible for consolidating the results produced by each of the Map() functions/tasks. [a]

   (a) Reduce (b) Map

   (c) Reducer (d) All of the mentioned

5. Point out the wrong statement. [d]

   (a) A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner

   (b) The MapReduce framework operates exclusively on <key, value> pairs

   (c) Applications typically implement the Mapper and Reducer interfaces to provide the map and reduce methods

   (d) None of the mentioned

6. _____ maps input key/value pairs to a set of intermediate key/value pairs. [a]

   (a) Mapper (b) Reducer

   (c) Both Mapper and Reducer (d) None of the mentioned

7. The number of maps is usually driven by the total size of _____ [a]

   (a) Inputs (b) Outputs

   (c) tasks (d) None of the mentioned

8. _____ is the default Partitioner for partitioning key space. [c]

(a) HashPar (b) Partitioner

(c) HashPartitioner (d) None of the mentioned

9. Running a _____ program involves running mapping tasks on many or all of the nodes in our cluster. [a]

(a) MapReduce (b) Map

(c) Reducer (d) All of the mentioned

10. The framework groups Reducer inputs by key in _____ stage. [a]

(a) Sort (b) Shuffle

(c) Reduce (d) None of the mentioned

Fill in the Blanks

1. _____ part of the MapReduce is responsible for processing one or more chunks of data and producing the output results.
2. _____ function is responsible for consolidating the results produced by each of the Map() functions/ tasks.
3. The number of maps is usually driven by the total size of _____
4. Running a _____ program involves running mapping tasks on many or all of the nodes in our cluster.
5. A _____ node acts as the Slave and is responsible for executing a Task assigned to it by the JobTracker.
6. The framework groups Reducer inputs by key in _____ stage.
7. The mapper implementation processes one line at a time via _____ method.
8. Applications can use the _____ to report progress and set application level status messages.
9. The right level of parallelism for maps seems to be around _____ maps per-node.
10. _____ maps input key/value pairs to a set of intermediate key/value pairs.

ANSWERS

1. Maptask
2. Reduce
3. Inputs
4. MapReduce
5. Task Tracker
6. Sort
7. Map
8. Reporter
9. 10-100
10. Mapper

One Mark Answers

1. What are the main components of MapReduce Job?

Ans:

Main Driver Class: providing job configuration parameters

Mapper Class:must extend org.apache.hadoop.mapreduce.Mapper class and performs execution.

2. What is Shuffling and Sorting in Map Reduce?

Ans:

Shuffling and Sorting are two major processes operating simultaneously during the working of mapper and reducer.

3. What is Partitioner and its usage?

Ans:

Partitioner that controls the partitioning of the intermediate map-reduce output keys using a hash function.

4. What is the difference between HDFS block and InputSplit?

Ans:

An HDFS block splits data into physical divisions while InputSplit in MapReduce splits input files logically.

5. What is JobTracker?

Ans:

JobTracker is a Hadoop service used for the processing of MapReduce jobs in the cluster.

6. Define Writable data types in MapReduce.

Ans:

Hadoop reads and writes data in a serialized form in writable interface.

7. What is Output Committer?

Ans:

Output Committer describes the commit of MapReduce task.

8. What is a "map" in Hadoop?

Ans:

In Hadoop, a map is a phase in HDFS query solving. A map reads data from an input location, and outputs a key value pair according to the input type.

9. What is a "reducer" in Hadoop?

Ans:

In Hadoop, a reducer collects the output generated by the mapper, processes it, and creates a final output of its own.

10. What are the parameters of mappers and reducers?

Ans:

The four parameters for mappers are:

- LongWritable (input)
- text (input)
- text (intermediate output)
- IntWritable (intermediate output)

The four parameters for reducers are:

- Text (intermediate output)
- IntWritable (intermediate output)
- Text (final output)
- IntWritable (final output)

UNIT III

Key-Value Databases: What Is a Key-Value Store, Key-Value Store Features, Suitable Use Cases, When Not to Use Document Databases: What Is a Document Database, Features, Suitable Use Cases, When Not to Use.

3.1 KEY-VALUE DATABASES

3.1.1 What Is a Key-Value Store

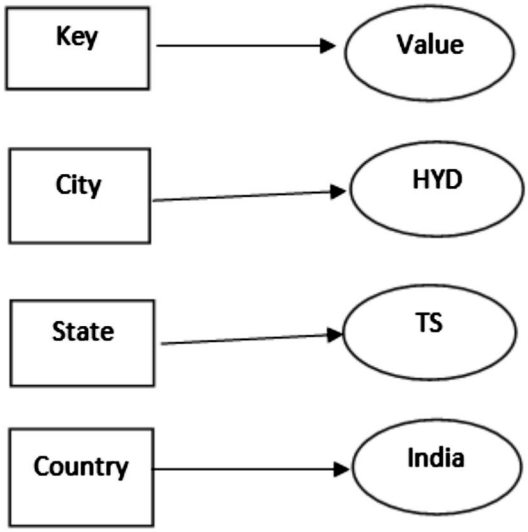
Q1. What is a Key-Value Store? Explain in detail.

(OR)

What is a Key-Value (Model)Database? Explain with advantages and disadvantages.

Ans: **(Imp.)**

A key-value data model or database is also referred to as a key-value store. It is a non-relational type of database. In this, an associative array is used as a basic database in which an individual key is linked with just one value in a collection. For the values, keys are special identifiers. Any kind of entity can be valued. The collection of key-value pairs stored on separate records is called key-value databases and they do not have an already defined structure.



Key-Value Database

A key-value database (sometimes called a key-value store) uses a simple key-value method to store data. These databases contain a simple string (the key) that is always unique and an arbitrary large data field (the value). They are easy to design and implement.

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

MAC table

Key	Value
10.94.214.172	3c:22:fb:86:c1:b1
10.94.214.173	00:0a:95:9d:68:16
10.94.214.174	3c:1b:fb:45:c4:b1

Fig. : An Example of Key-value database

As the name suggests, this type of NoSQL database implements a hash table to store unique keys along with the pointers to the corresponding data values. The values can be of scalar data types such as integers or complex structures such as JSON, lists, BLOB, and so on. A value can be stored as an integer, a string, JSON, or an array—with a key used to reference that value. It typically offers excellent performance and can be optimized to fit an organization's needs. Key-value stores have no query language but they do provide a way to add and remove key-value pairs. Values cannot be queried or searched upon. Only the key can be queried.

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

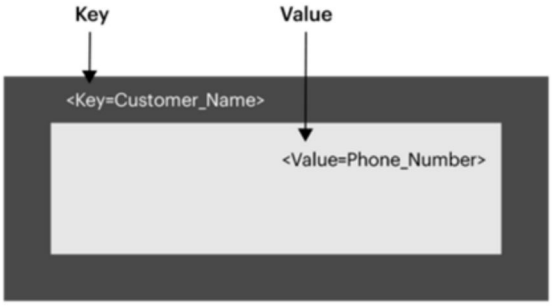


Fig. : A simple example of key-value data store.

Working of key-value databases:

A number of easy strings or even a complicated entity are referred to as a value that is associated with a key by a key-value database, which is utilized to monitor the entity. Like in many programming paradigms, a key-value database resembles a map object or array, or dictionary, however, which is put away in a tenacious manner and controlled by a DBMS.

An efficient and compact structure of the index is used by the key-value store to have the option to rapidly and dependably find value using its key. For example, Redis is a key-value store used to track lists, maps, heaps, and primitive types (which are simple data structures) in a constant database. Redis can uncover a very basic point of interaction to query and manipulate value types, just by supporting a predetermined number of value types, and when arranged, is prepared to do high throughput.

When to use a key-value database:

Here are a few situations in which you can use a key-value database: -

- Session management on a large scale.
- Using cache to accelerate application responses.
- Storing personal data on specific users.

- Product recommendations, storing personalized lists of items for individual customers.
- Managing each player's session in massive multiplayer online games.

Features:

- One of the most un-complex kinds of NoSQL data models.
- For storing, getting, and removing data, key-value databases utilize simple functions.
- Querying language is not present in key-value databases.
- Built-in redundancy makes this database more reliable.

Some Examples of Key-value Databases:

Here are some popular key-value databases which are widely used:

- **Couchbase:** It permits SQL-style querying and searching for text.
- **Amazon Dynamo DB**

  The key-value database which is mostly used is Amazon DynamoDB as it is a trusted database used by a large number of users. It can easily handle a large number of requests every day and it also provides various security options.
- **Riak:** It is the database used to develop applications.
- **Aerospike:** It is an open-source and real-time database working with billions of exchanges.
- **Berkeley DB**

  It is a high-performance and open-source database providing scalability.

  Advantages and disadvantages of key-value store.

Advantages:

- It is very easy to use. Due to the simplicity of the database, data can accept any kind, or even different kinds when required.
- Its response time is fast due to its simplicity, given that the remaining environment near it is very much constructed and improved.
- Key-value store databases are scalable vertically as well as horizontally.
- Built-in redundancy makes this database more reliable.

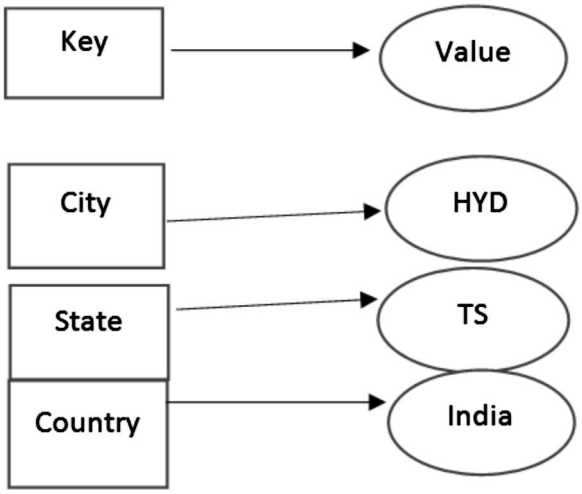
Disadvantages:

- As querying language is not present in key-value databases, transportation of queries from one database to a different database cannot be done.
- The key-value store database is not refined. You cannot query the database without a key.

Q2. What is a Key-Value Store? Explain the popular Key-Value Store databases

Ans: **(Imp.)**

A key-value data model or database is also referred to as a key-value store. It is a non-relational type of database. In this, an associative array is used as a basic database in which an individual key is linked with just one value in a collection. For the values, keys are special identifiers. Any kind of entity can be valued. The collection of key-value pairs stored on separate records is called key-value databases and they do not have an already defined structure.



Key-Value Database:

A key-value database (sometimes called a key-value store) uses a simple key-value method to store data. These databases contain a simple string (the key) that is always unique and an arbitrary large data field (the value). They are easy to design and implement.

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

MAC table

Key	Value
10.94.214.172	3c:22:fb:86:c1:b1
10.94.214.173	00:0a:95:9d:68:16
10.94.214.174	3c:1b:fb:45:c4:b1

Fig. : An Example of Key-value database

As the name suggests, this type of NoSQL database implements a hash table to store unique keys along with the pointers to the corresponding data values. The values can be of scalar data types such as integers or complex structures such as JSON, lists, BLOB, and so on. A value can be stored as an integer, a string, JSON, or an array-with a key used to reference that value. It typically offers excellent performance and can be optimized to fit an organization's needs. Key-value stores have no query language but they do provide a way to add and remove key-value pairs. Values cannot be queried or searched upon. Only the key can be queried.

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

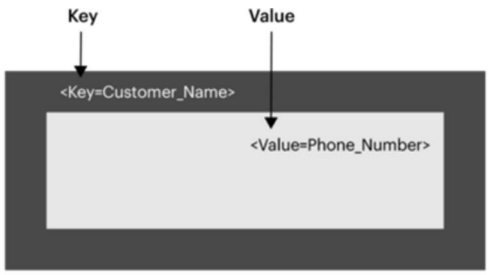


Fig.: A simple example of key-value data store.

Examples of Popular Key-Value Databases:

There are several different types of key-value database models to pick from, for example, some store data on an SSD, while others store on RAM. The truth is, some of the most popular and widely-used databases are key-value stores, and we rely on them on a daily basis in our day-to-day lives.

- **Amazon Dynamo DB**: Probably the most widely used key-value store database, in fact, it was the research into DynamoDB that really started making NoSQL really popular.
- **Aerospike :** Open-source database that is optimized for in-memory storage.
- **Berkeley DB** : Another open-source database that is a high-performance database storage library, although it's relatively basic.
- **Couchbase :** Interestingly allows for text searches and SQL-style querying.
- **Memcached :** Helps speed up websites by storing cache data in RAM, plus it's free and open-source.
- **Riak** : Made for developing apps, it works well with other databases and apps.
- **Redis** : A multi-purpose database that also acts as memory cache and message broker.

3.1.2 Key-Value Store Features

Q3. Explain about Features of Key-Value Store.

Ans:

No SQL refers to a non-SQL or nonrelational database that main purpose of it is to provide a mechanism for storage and retrieval of data. NoSQL database stores the information in JSON documents instead of columns and rows. As we know the relational database use rows and columns for storing and retrieval of data but in the case of NoSQL it uses JSON documents instead of rows and columns and that is why it is also known as nonrelational SQL or database.

A NoSQL database includes simplicity of design, simpler horizontal scaling, and has fine control over availability. The data structures used in the NoSQL database are different from those we used in the relational database. the database used in NoSQL is more advanced which makes some operations faster in No SQL.

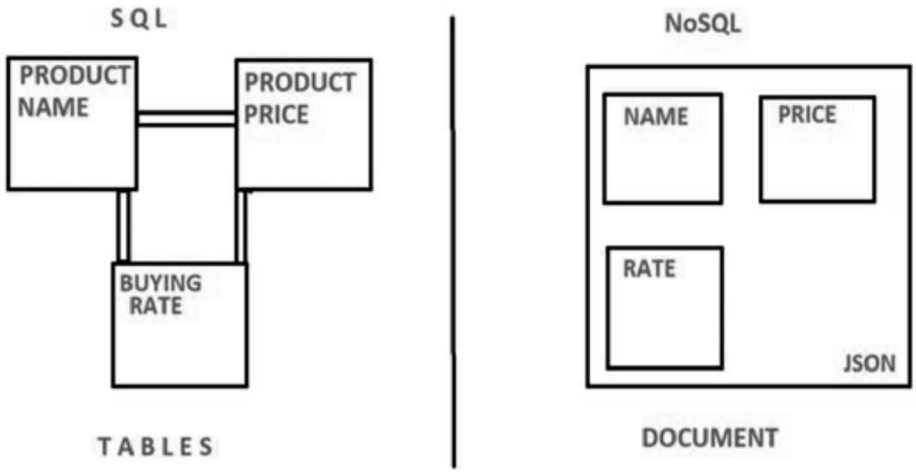


Fig.: Graphical difference between SQL and NoSQL

No SQL is Using for the following reasons

- Relationships present in NoSQL are less complex as compared to relational database systems.
- Actions performed in NoSQL are fast as compared to other databases.
- Implementation of it is less costly than others.
- Programming in it is easy to use and more flexible.
- A high level of scalability is provided by NoSQL.

Types of NoSQL

These are some of the most popular types of NoSQL as follows:

- **Document databases**

  Primary operation of it is to store the information in documents.

- **Key-Value Store**

  These groups associate the data in collections with records that are identified with unique keys for easy retrieval.

- **Wide Column database**

  They use the tabular format yet allow a wide variance in how data is named and formatted in each row and each table. It is different from relational databases because the names and format of the columns vary from row to row in the table.

- **Graph database**

  Its main aim is to use graph structures to define the relationships between data points.

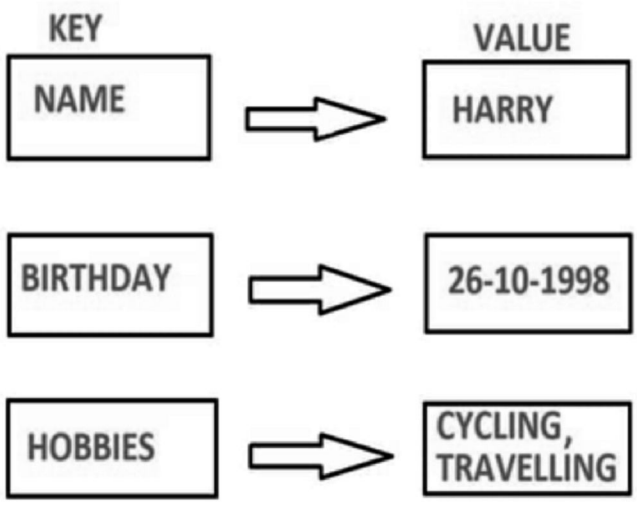


Fig.: Diagram of Key-Value Store in NoSQL

Features of Key-Value Store:

- **Consistency**

  Consistency is a feature only applicable for operations on a single key in a key-value store. There are various implementations in the key-value store for example in RIAK, the eventually consistent model of consistency is implemented.

- **Transactions**

  In it, there are no guarantees on the writes as many data stores implement transactions in different ways for example RIAK uses the concept of quorum implemented by using the W value replication factor. (RIAK is an open-source and distributed database that is generally based on a NoSQL database system.)

- **Query**

  All the key-value stores can be query by the key and that's about it. If we have requirements to query by using some of the attributes of the column, it is not possible for using the database in this condition, our application needs to read the value to recognize if the attribute meets the conditions.

- **Scaling**

  Key values stored scale by a process called sharding. Sharding means we can support scalability by putting different parts of the data onto different servers, this is called sharding.

Popular Key-Value Databases:

- **REDIS**

  Redis is one of the popular key-value databases as it is an open-source, in-memory data structure, used as a database and message broker. REDIS supports many data structures such as lists, hashes, sets, strings. REDIS has many more important features such as it has built-in replication, LUA scripting and it also supports LRU eviction.

- **AEROSPIKE**

  It is the world's leading enterprise-grade, internet-scale, key-value store database, it is popular for some of its advantages over other databases such as aerospike gives strong consistency, linear scalability, and higher performance as compared to others.

- **AMAZON DynamoDB**

  The main reason behind the popularity of this database is that it is a fully-managed database service that provides fast performance at any scale. Many AWS customers chose DynamoDB for web gaming, mobile, ed-tech, IoT, and many other applications.

3.1.3 Suitable Use Cases

Q4. What are the Suitable Use Cases of Key - Value Database

Ans: **(Imp.)**

Here are 10 enterprise use cases best addressed by Key value Database

- **Personalization**

  A personalized experience requires data, and lots of it demographic, contextual, behavioral and more. The more data available, the more personalized the experience. However, relational databases are overwhelmed by the volume of data required for personalization. In contrast, a distributed NoSQL database can scale elastically to meet the most demanding workloads and build and update visitor profiles on the fly, delivering the low latency required for real-time engagement with your customers.

- **Profile Management**

  User profile management is core to Web and mobile applications to enable online transactions, user preferences, user authentication and more. Today, Web and mobile applications support millions – or even hundreds of millions – of users. While relational databases can struggle to serve this amount of user profile data as they are limited to a single server, distributed databases can scale out across multiple servers. With NoSQL, capacity is increased simply by adding commodity servers, making it far easier and less expensive to scale.

- **Real-Time Big Data**

  The ability to extract information from operational data in real-time is critical for an agile enterprise. It increases operational efficiency, reduces costs, and increases revenue by enabling you to act immediately on current data. In the past, operational databases and analytical databases were maintained as different environments. The operational database powered applications while the analytical database was part of the business intelligence and reporting

environment. Today, NoSQL is used as both the front-end – to store and manage operational data from any source, and to feed data to Hadoop – as well as the back-end to receive, store and serve analytic results from Hadoop.

- **Content Management**

The key to effective content is the ability to select a variety of content, aggregate it and present it to the customer at the moment of interaction. NoSQL document databases, with their flexible data model, are perfect for storing any type of content – structured, semi-structured or unstructured – because NoSQL document databases don't require the data model to be defined first. Not only does it enable enterprises to easily create and produce new types of content, it also enables them to incorporate user-generated content, such as comments, images, or videos posted on social media, with the same ease and agility.

- **Catalog**

Catalogs are not only referenced by Web and mobile applications, they also enable point-of-sale terminals, self-service kiosks and more. As enterprises offer more products and services, and collect more reference data, catalogs become fragmented by application and business unit or brand.Because relational databases rely on fixed data models, it's not uncommon for multiple applications to access multiple databases, which introduces complexity and data management challenges. By contrast, a NoSQL document database, with its flexible data model, enables enterprises to more easily aggregate catalog data within a single database.

- **Customer 360° View**

Customers expect a consistent experience regardless of channel, while the enterprise wants to capitalize on upsell/cross-sell opportunities and to provide the highest level of customer service. However, as the number of products and services, channels, brands and business units increases, the fixed data model of relational databases forces enterprises to fragment customer data because different applications work with different customer data. NoSQL document databases use a flexible data model that enables multiple applications to access the same customer data as well as add new attributes without affecting other applications.

- **Mobile Applications**

With nearly two billion smartphone users, mobile applications face scalability challenges in terms of growth and volume. For instance, it is not uncommon for mobile games to reach tens of millions of users in a matter of months. With a distributed, scale-out database, mobile applications can start with a small deployment and expand as the user base grows, rather than deploying an expensive, large relational database server from the beginning.

- **Internet of Things**

Today, some 20 billion devices are connected to the Internet everything from smartphones and tablets to home appliances and systems installed in cars, hospitals and warehouses. The volume, velocity, and variety of machine-generated data are increasing with the proliferation of digital telemetry, which is semi-structured and continuous. Relational databases struggle with the three well-known challenges from big data IoT applications: scalability, throughput, and data variety. By contrast, NoSQL allows enterprises to scale concurrent data access to millions of connected devices and systems, store large volumes of data, and meet the performance requirements of mission-critical infrastructure and operations.

- **Digital Communications**

In an enterprise environment, digital communication may take the form of online interaction via direct messaging to help visitors find a product or complete the checkout process. And as with mobile text messaging, the application may need to support millions of website visitors. Relational databases are limited in responsiveness and scalability while NoSQL databases, thanks to their distributed architecture, deliver the sub-milli second responsiveness and elastic scalability that digital communication applications require.

- **Fraud Detection**

  For financial service organizations, fraud detection is essential to reducing profit loss, minimizing financial exposure and complying with regulations. When customers pay with a credit or debit card, they expect immediate confirmation. The process impacts both the enterprise and its customers. Fraud Detection relies on data – detection algorithm rules, customer information, transaction information, location, time of day and more – applied at scale and in less than a millisecond. While relational databases struggle to meet this low latency requirement, elastically scalable NoSQL databases can reliably deliver the required performance.

3.1.4 When Not to Use Document Data bases

Q5. Explain about When Not to Use Document Databases.

Ans:

When to Use NoSQL:

Given below are the use cases where you should prefer using Document databases:

- To handle a huge volume of structured, semi-structured and unstructured data.
- Where there is a need to follow modern software development practices like Agile Scrum and if you need to deliver prototypes or fast applications.
- If you prefer object-oriented programming.
- If your relational database is not capable enough to scale up to your traffic at an acceptable cost.
- If you want to have an efficient, scale-out architecture in place of an expensive and monolithic architecture.
- If you have local data transactions that need not be very durable.
- If you are going with schema-less data and want to include new fields without any ceremony.
- When your priority is easy scalability and availability.

When to Avoid :

Enlisted below are some pointers that would guide you on when to avoid Document database.

- If you are required to perform complex and dynamic querying and reporting, then you should avoid using NoSQL as it has a limited query functionality. For such requirements, you should prefer SQL only.
- NoSQL also lacks in the ability to perform dynamic operations. It can't guarantee ACID properties. In such cases like financial transactions, etc., you may go with SQL databases.
- You should also avoid NoSQL if your application needs run-time flexibility.
- If consistency is a must and if there aren't going to be any large-scale changes in terms of the data volume, then going with the SQL database is a better option.

Q6. Explain Pros & Cons SQL & NoSQL

Ans:

Pros and Cons of SQL & NoSQL

Enlisted below are the various Pros and Cons of SQL as well as NoSQL.

SQL Pros:

- It is highly suitable for relational databases.
- Has a predefined schema which is helpful in many cases.
- Normalization can be greatly used here, thus it also helps in removing redundancy and organizing data in a better way.
- Transactions in SQL databases are ACID compliant, thereby guarantees security and stability.
- Follows well-defined standards like ISI and ANSI which are accepted worldwide.
- Code-free.
- Unbeatable speed in retrieving database records with great ease.
- Uses single standardized language i.e SQL across different RDBMS.

SQL Cons:

- The process of interfacing is complex.
- As SQL is an object, it occupies space.
- Handling Big data is very costly as you will have to increase the hardware for scaling.
- When a table is dropped, the view becomes inactive.

NoSQL Pros:

- Capable of handling big data.
- As it is schema-less and table free, it offers a high level of flexibility with data models.
- It is a low-cost database and the open source NoSQL databases provide very affordable solutions to small enterprises.
- Easier and low-cost scalability. You don't need to increase the hardware for scaling. You just need to add more servers to the pool as NoSQL is schema-free and built on distributed systems.
- Detailed database modeling is not required here. Hence it saves time and effort.

NoSQL Cons:

- The benefits of NoSQL come at the cost of relaxing ACID properties. NoSQL offers only eventual consistency.
- Relatively less community support.
- Lacks standardization, unlike SQL, which in turn creates some issues during migration.
- Inter-operability is also a concern in the case of NoSQL databases.

3.2 When Not to Use Document Data Base

3.2.1 What Is a Document Database

Q7. Explain about Document Database in NoSQL

Ans: **(Imp.)**

Document Databases in NoSQL

Document Database (Data Model)

A Document Data Model is a lot different than other data models because it stores data in JSON, BSON, or XML documents. in this data model, we can move documents under one document and apart from this, any particular elements can be indexed to run queries faster. Often documents are stored and retrieved in such a way that it becomes close to the data objects which are used in many applications which means very less translations are required to use data in applications. JSON is a native language that is often used to store and query data too.

So, in the document data model, each document has a key-value pair below is an example for the same.

```
{
    "Name" : "Kumar",
    "Address" : "Near Ushodaya Colony",
    "Email" : "hotmail123@yahoo.com",
    "Contact" : "1234567890"
}
```

Working of Document Data Model:

This is a data model which works as a semi-structured data model in which the records and data associated with them are stored in a single document which means this data model is not completely unstructured. The main thing is that data here is stored in a document.

Features:

- **Document Type Model**

  As we all know data is stored in documents rather than tables or graphs, so it becomes easy to map things in many programming languages.

- **Flexible Schema**

  Overall schema is very much flexible to support this statement one must know that not all documents in a collection need to have the same fields.

- **Distributed and Resilient**

  Document data models are very much dispersed which is the reason behind horizontal scaling and distribution of data.

- **Manageable Query Language**

  These data models are the ones in which query language allows the developers to perform CRUD (Create Read Update Destroy) operations on the data model.

Examples of Document Data Models :

- Amazon DocumentDB
- MongoDB
- Cosmos DB
- ArangoDB
- Couchbase Server
- CouchDB

Applications of Document Data Model:

- **Content Management**

  These data models are very much used in creating various video streaming platforms, blogs, and similar services Because each is stored as a single document and the database here is much easier to maintain as the service evolves over time.

- **Book Database**

  These are very much useful in making book databases because as we know this data model lets us nest.

- **Catalog**

  When it comes to storing and reading catalog files these data models are very much used because it has a fast reading ability if incase Catalogs have thousands of attributes stored.

- **Analytics Platform**

  These data models are very much used in the Analytics Platform.

Q8. Define Document Database? How document databases differ from relational databases?

Ans:

Document Database (Data Model):

A Document Data Model is a lot different than other data models because it stores data in JSON, BSON, or XML documents. in this data model, we can move documents under one document and apart from this, any particular elements can be indexed to run queries faster. Often documents are stored and retrieved in such a way that it becomes close to the data objects which are used in many applications which means very less translations are required to use data in applications. JSON is a native language that is often used to store and query data too.

Document databases differ from relational databases:

Document databases are significantly different in function to traditional relational databases.

Relational databases typically store data in separate, linked tables (as defined by the programmer), allowing single objects to be spread across several tables. But, in document databases, all information for a given document or object is stored in a single instance there is no need to do object-relational mapping when loading data into the database, or when retrieving things.

Document databases are typically faster for this reason, although this is incumbent on how you use it: jump to the pros and cons for more on this.

Flexible Schemas

Rather than the tabular model, document stores have a dynamic self-describing schema, adaptable to change. No need to predefine it is the database. Values and fields can alternate through different documents; modify the design at any stage, without fundamentally disrupting its structure no need for schema migrations. Note: some document stores allow JSON schema, letting you set governing rules for managing document structures.

Better for Agile Developers

Due to the intuitive data model, document-oriented databases are faster and easier for developers. The objects in your code can be mapped to the documents, making them more intuitive to handle. Decomposition of data across tables is eliminated as a necessity, along with the need to integrate a separate ORM layer, or using costly JOINs.

Powerful Querying

Query in a flexible way, with the expressive query language and multifaceted indexing feature. This is an essential difference between relational databases and document stores. The query language has comprehensive abilities, letting you deal with data however you think is best. Full-time aggregations, ad hoc queries, and indexing are deep ways of processing, modifying, and retrieving your data. ACID transactions let you retain guarantees you are accustomed to having in SQL databases, whether this is manipulation of data in single documents or in shard multiples.

Widely Compatible

JSON documents are used in every corner. As a language-independent, human readable, and non data-intensive standard, JSON is widely used for data interchange and storage. Remember that document stores are a subset or superset of other existing data models which means that you can codify data however your application requires key-value pairs; rich objects; tables; geospatial and time-series data; and graph edges/nodes. Only a single query language is needed to work in documents, adding a consistency to your development workflow whatever data model you have chosen.

Distributed Systems

Distributed systems increase how massively scalable and resilient your data is. While relational databases have a more monolithic framework with incremental scaling-up, document databases are essentially a form of distributed systems. Each document is an independent unit, more easily distributed across servers without destroying data locality. Retain a high level of availability of applications using replication with self-healing recovery. This also allows for isolation of several workloads from one another in a cluster. And native starting allows for application transparent, elastic horizontal scale-out to facilitate workload scaling.

3.2.2 Features

Q9. What are the features of Document Data Base (Data Model)?

Ans:

Working of Document Data Model

This is a data model which works as a semi-structured data model in which the records and data associated with them are stored in a single document which means this data model is not completely unstructured. The main thing is that data here is stored in a document.

Features:

- **Document Type Model**

  As we all know data is stored in documents rather than tables or graphs, so it becomes easy to map things in many programming languages.

- **Flexible Schema**

  Overall schema is very much flexible to support this statement one must know that not all documents in a collection need to have the same fields.

- **Distributed and Resilient**

  Document data models are very much dispersed which is the reason behind horizontal scaling and distribution of data.

- **Manageable Query Language**

  These data models are the ones in which query language allows the developers to perform CRUD (Create Read Update Destroy) operations on the data model.

Examples of Document Data Models :

- Amazon DocumentDB
- MongoDB
- Cosmos DB
- ArangoDB
- Couchbase Server
- CouchDB

Advantages

- **Schema-less**

  These are very good in retaining existing data at massive volumes because there are absolutely no restrictions in the format and the structure of data storage.

- **Faster creation of document and maintenance**

  It is very simple to create a document and apart from this maintenance requires is almost nothing.

- **Open formats**

  It has a very simple build process that uses XML, JSON, and its other forms.

- **Built-in versioning**

  It has built-in versioning which means as the documents grow in size there might be a chance they can grow in complexity. Versioning decreases conflicts.

Disadvantages:

- **Weak Atomicity**

  It lacks in supporting multi-document ACID transactions. A change in the document data model involving two collections will require us to run two separate queries i.e. one for each collection. This is where it breaks atomicity requirements.

- **Consistency Check Limitations**

  One can search the collections and documents that are not connected to an author collection but doing this might create a problem in the performance of database performance.

- **Security**

  Nowadays many web applications lack security which in turn results in the leakage of sensitive data. So it becomes a point of concern, one must pay attention to web app vulnerabilities.

3.2.3 Suitable Use Cases:

Q10. Explain about Suitable Use Cases of Document Database

Ans: **(Imp.)**

Document Database:

A document database, also known as a document-oriented database or document store, is a NoSQL database that stores data as structured documents instead of rows and columns. Document databases use JavaScript Object Notation (JSON), extensive markup language (XML), binary JSON (BSON), or yet another markup language (YAML) formats to define, store, manage, and retrieve data.

Documents in a Document Database:

A document refers to a self-describing record in a document database. Here's an example of what a document looks like in a document database.

Example of a document written as a JSON object:

```
{
    "_id": "navinkumar",
    "firstName": "Navin",
    "lastName": "Kumar",
    "email": "navin@g2.com",
    "department": "DataScience"
}
```

Documents store information about objects and related metadata in field-value pairs. The values include strings, dates, arrays, objects, and numbers.

Use Cases of document databases:

Document database software systems enable organizations to access data immediately with fast queries and flexible indexing. The flexibility of using the same document model for application coding and data query makes document database systems even more lucrative for information technology (IT) companies. Here're the features that make organizations choose document databases over SQL databases.

Intuitive document data model

Document databases store data using documents instead of structures, like tables or graphs. Programming languages map these documents to objects via coding and enable you to store data together so you can access them together. This flexibility allows developers to write less code and still deliver stellar end-user performance.

Besides empowering developers to rapidly create applications, document databases eliminate the need to integrate separate object-relational mapping (ORM) layers, run expensive joins, or decompose data across tables.

Document databases using JSON documents for data storage enable you to structure data using rich objects, key-value stores, graph nodes and edges, geospatial, or time-series data. This data modeling helps you create easy-to-access, language-independent, lightweight, and human-readable documents.

Flexible schema

Document databases come with dynamic and self-describing schemas (implementation of a data model in a specific database) that offer you the flexibility to have documents with different fields in a collection. This ability to accommodate varying fields across documents eliminates the need for pre-defining schemas in a database.

When developers don't have to pre-define schemas, they can easily modify structures without causing disruptions during schema migration. Some document databases come with a schema validation feature that allows you to enforce document structure rules and optionally lock down schemas.

Horizontal scaling and resiliency

Document stores facilitate horizontal scaling or scale-out, enabling you to add nodes to share the data load. The ability to spread data across nodes without requiring queries to join nodes together makes data distribution easier.

Furthermore, document databases support replication and partitioning or sharding, both of which help you to scale database performance.

Easy querying

Document databases ease the CRUD operation execution by letting developers query through an API or query language. This ease of querying translates into easy data retrieval using field values or unique identifiers.

Working of Document DataBase

A document database software stores or fetches information in the form of a document or semi-structured database. You can manage these non-relational documents based on key-value pairs instead of a tabular schema of rows and columns.

Document databases can parse documents, regardless of the type of data they store. This data storage flexibility makes querying, adding, editing, and deleting easier for developers. However, you can still use different file format schemas to define document structures.

3.2.4 When not use

Q11. When should you not use a Document DB?

Ans:

Not Use a Document DB

We shouldn't use a document database when any one of these criteria are true: your data is structured as a hierarchy or a graph (network) of arbitrary depth, the typical access pattern emphasizes reading over writing, or. there's no requirement for ad-hoc queries.

When use a Document DB

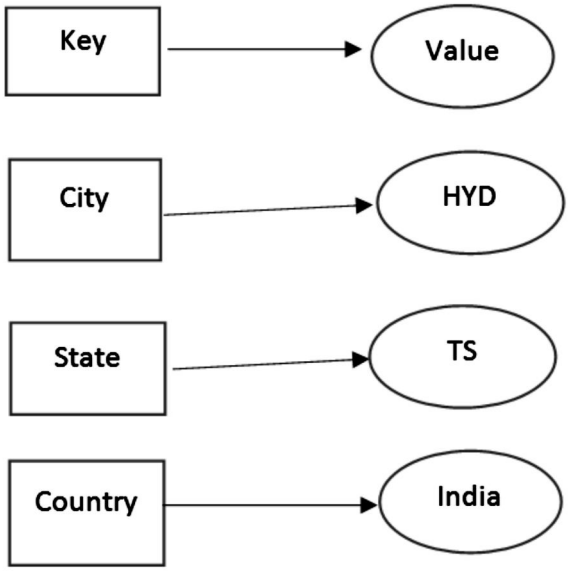
A document database is a great choice for content management applications such as blogs and video platforms. With a document database, each entity that the application tracks can be stored as a single document. The document database is more intuitive for a developer to update an application as the requirements evolve.

Short Question & Answers

1. What is a Key-Value Store?

Ans:

A key-value data model or database is also referred to as a key-value store. It is a non-relational type of database. In this, an associative array is used as a basic database in which an individual key is linked with just one value in a collection. For the values, keys are special identifiers. Any kind of entity can be valued. The collection of key-value pairs stored on separate records is called key-value databases and they do not have an already defined structure.



2. What are the advantages and disadvantages of key-value store.

Ans:

Advantages:

- It is very easy to use. Due to the simplicity of the database, data can accept any kind, or even different kinds when required.
- Its response time is fast due to its simplicity, given that the remaining environment near it is very much constructed and improved.
- Key-value store databases are scalable vertically as well as horizontally.
- Built-in redundancy makes this database more reliable.

Disadvantages:

- As querying language is not present in key-value databases, transportation of queries from one database to a different database cannot be done.
- The key-value store database is not refined. You cannot query the database without a key.

3. What are the NoSQL use cases

Ans :

- Fraud detection and identity authentication.
- Inventory and catalog management.
- Personalization, recommendations and customer experience.
- Internet of things (IoT) and sensor data.
- Financial services and payments.
- Messaging.
- Logistics and asset management.
- Content management systems.

4. State the Advantages of Document Database

Ans:

Below are some key advantages of document databases are :

Advantages

(i) Schema-less

(ii) Faster creation and care

(iii) No foreign keys

(iv) Open formats

(v) Built-in versioning

5. What are the challenges of Document database

Ans:

Some of the most common document database challenges stem from atomicity requirements, consistency, and security.

1. Security

Today, data applications need to eliminate malware infections, tackle unauthorized access, maintain integrity, and preserve confidentiality for data security purposes. Relational databases handle these security issues with data authentication, authorization, database watermarking, and audit logs, while document databases need database-level security and fine-grained control.

Common security issues for document database systems include lack of automatic data encryption audit log, copyright preservation, and certificate-based authentication.

2. Lack of consistency checking

Document database systems contain documents with varying fields. These documents may not have relations with one another. This lack of inter-relation reduces consistency checks, which cause problems during database consistency audits.

3. Lack of atomicity

Relational databases make data changes with a single query or command. With document databases, you have to run two separate queries to make changes in two data collections. The need for running separate queries violates atomicity requirements, meaning you'll have to break down a requirement further to achieve the desired outcome.

6. What is Document Database?

Ans:

Document Database:

A document database, also known as a document-oriented database or document store, is a NoSQL database that stores data as structured documents instead of rows and columns. Document databases use JavaScript Object Notation (JSON), extensive markup language (XML), binary JSON (BSON), or yet another markup language (YAML) formats to define, store, manage, and retrieve data.

Documents in a Document Database:

A document refers to a self-describing record in a document database. Here's an example of what a document looks like in a document database.

Example of a document written as a JSON object:

```
{
    "_id": "navinkumar",
    "firstName": "Navin",
    "lastName": "Kumar",
    "email": "navin@g2.com",
    "department": "DataScience"
}
```

Documents store information about objects and related metadata in field-value pairs. The values include strings, dates, arrays, objects, and numbers.

7. What are the features of Document Data Base (Data Model)?

Ans:

Features

- **Document Type Model**

  As we all know data is stored in documents rather than tables or graphs, so it becomes easy to map things in many programming languages.

- **Flexible Schema**

  Overall schema is very much flexible to support this statement one must know that not all documents in a collection need to have the same fields.

- **Distributed and Resilient**

  Document data models are very much dispersed which is the reason behind horizontal scaling and distribution of data.

- **Manageable Query Language**

  These data models are the ones in which query language allows the developers to perform CRUD (Create Read Update Destroy) operations on the data model.

  Examples of Document Data Models :

  - Amazon DocumentDB
  - MongoDB
  - Cosmos DB
  - ArangoDB
  - Couchbase Server
  - CouchDB

8. Write the Differences Between SQL And NoSQL

Ans:

Difference Between SQL And NoSQL

S.No	SQL	NoSQL
1.	SQL databases are mainly relational database (RDBMS).	NoSQL databases are mainly non-relational or distributed databases.
2.	An aged technology.	Relatively young technology.
3.	SQL databases are table based in the form of row & columns and must strictly adhere to standard schema definitions. They are a better option for applications which need multi-row transactions.	NoSQL databases can be based on documents, key-value pairs, graphs or columns and they don't have to stick to standard schema definitions.
4.	They have a well-designed pre-defined schema for structured data.	They have the dynamic schema for unstructured data. Data can be flexibly stored without having a pre-defined structure.
5.	SQL databases favors normalized schema. Costly to scale.	NoSQL databases favors de-normalized schema. Cheaper to scale when compared to relational databases.
6.	SQL databases are vertically scalable. They can be scaled by increasing the hardware capacity (CPU, RAM, SSD, etc.) on a single server.	NoSQL databases are horizontally scalable. They can be scaled by adding more servers to the infrastructure to manage large load and lessen the heap.
7.	Best fit for high transaction-based applications. Not suitable for hierarchical data storage.	You can use NoSQL for heavy transactional purpose. However, it is not the best fit for this. Suitable for hierarchical data storage and storing large data sets (E.g. Big Data).
8.	Example of SQL databases: MySQL, Oracle, MS-SQL, SQLite.	Examples of NoSQL databases: MongoDB, Apache CouchDB, Redis, HBase.

9. When to Use NoSQL?

Ans:

Given below are the use cases where you should prefer using NoSQL databases:

- To handle a huge volume of structured, semi-structured and unstructured data.
- Where there is a need to follow modern software development practices like Agile Scrum and if you need to deliver prototypes or fast applications.
- If you prefer object-oriented programming.
- If your relational database is not capable enough to scale up to your traffic at an acceptable cost.
- If you want to have an efficient, scale-out architecture in place of an expensive and monolithic architecture.
- If you have local data transactions that need not be very durable.
- If you are going with schema-less data and want to include new fields without any ceremony.
- When your priority is easy scalability and availability.

10. When to Avoid NoSQL?

Ans:

Enlisted below are some pointers that would guide you on when to avoid NoSQL.

- If you are required to perform complex and dynamic querying and reporting, then you should avoid using NoSQL as it has a limited query functionality. For such requirements, you should prefer SQL only.
- NoSQL also lacks in the ability to perform dynamic operations. It can't guarantee ACID properties. In such cases like financial transactions, etc., you may go with SQL databases.
- You should also avoid NoSQL if your application needs run-time flexibility.
- If consistency is a must and if there aren't going to be any large-scale changes in terms of the data volume, then going with the SQL database is a better option.

11 . What does "Document-oriented" vs. Key-Value mean when talking about MongoDB vs Cassandra?

Ans:

A key-value store provides the simplest possible data model and is exactly what the name suggests: it's a storage system that stores values indexed by a key. You're limited to query by key and the values are opaque, the store doesn't know anything about them. This allows very fast read and write operations (a simple disk access) and I see this model as a kind of non-volatile cache (i.e. well suited if you need fast accesses by key to long-lived data).

A document-oriented database extends the previous model and values are stored in a structured format (a document, hence the name) that the database can understand. For example, a document could be a blog post and the comments and the tags stored in a denormalized way. Since the data are transparent, the store can do more work (like indexing fields of the document) and you're not limited to query by key. As I hinted, such databases allows to fetch an entire page's data with a single query and are well suited for content oriented applications (which is why big sites like Facebook or Amazon like them).

Choose the Correct Answers

1. Which of the following is not an example of a NoSQL database? [d]

   (a) CouchDB (b) MongoDB

   (c) HBase (d) PostgreSQL

2. SQL databases are: [b]

   (a) Horizontally scalable

   (b) Vertically scalable

   (c) Either horizontally or vertically scalable

   (d) They don't scale

3. NoSQL databases are most often referred to as: [b]

   (a) Relational (b) Distributed

   (c) Object-oriented (d) Network

4. Which of the following companies developed NoSQL database Apache Cassandra? [d]

   (a) LinkedIn (b) Twitter

   (c) MySpace (d) Facebook

5. When is it best to use a NoSQL database? [c]

   (a) When providing confidentiality, integrity, and availability is crucial

   (b) When the data is predictable

   (c) When the retrieval of large quantities of data is needed

   (d) When the retrieval speed of data is not critical

6. Which of the following is not a reason NoSQL has become a popular solution for some organizations? [b]

   (a) Better scalability

   (b) Improved ability to keep data consistent

   (c) Faster access to data than relational database management systems (RDBMS)

   (d) More easily allows for data to be held across multiple servers

7. Which of the following format is supported by Mongo DB? [c]

   (a) SQL (b) XML

   (c) BSON (d) All of the mentioned

8. Which of the following is a primary classification for NoSQL architectures? [d]

(a) Document databases (b) Graph databases

(c) Key-value databases (d) All of the mentioned

9. Which is an advantage of NewSQL? [d]

(a) Less complex applications, greater consistency.

(b) Convenient standard tooling.

(c) SQL influenced extensions.

(d) All of the mentioned

10. _____ systems are scale-out file-based (HDD) systems moving to more uses of memory in the nodes. [a]

(a) NoSQL (b) NewSQL

(c) SQL (d) All of the mentioned

Fill in the Blanks

1. SQL databases are_____
2. No SQL databases are_____
3. Which of the format is supported by MongoDB_____
4. _____ stores are used to store information about network like Facebook.
5. NoSQL databases is used mainly for handling large volumes of_____data.
6. Hadoop works in _____
7. The inter process communication between different nodes in Hadoop uses_____
8. In HDFS the files can't be_____
9. Hadoop is a framework that works with a variety of related tools_____
10. _____ command is used to interact and view Job Queue information in HDFS.

ANSWERS

1. Vertically scalable
2. Distributed
3. BSON
4. Graph
5. All kinds of data
6. Master-Slave fashion
7. RPC
8. Executed
9. Map Reduce, Hive and HBase
10. Queue

One Mark Questions

1. What do you understand by 'Database'?

Ans:

Database is an organized collection of related data where the data is stored and organized to serve some specific purpose.

2. What Is Key Value Database?

Ans:

A key value database or key value store is a row-based associative array with a core data model comprised of key-value pairs.

3. What are the examples of key value databases?

Ans:

Aerospike, Amazon DynamoDB, Apache Cassandra, Couchbase and Redis.

4. What are the key value use cases?

Ans:

Data is denormalized, Relatively simple data models, Simple schema, High throughput and Low latencies.

5. What are documents?

Ans:

A document is a record in a document database. A document typically stores information about one object and any of its related metadata.

6. What are Collections?

Ans:

A collection is a group of documents. Collections typically store documents that have similar contents.

7. What are CRUD operations?

Ans:

Create: Documents can be created in the database.

Read: Documents can be read from the database.

Update: Existing documents can be updated.

Delete: Documents can be deleted from the database.

8. How much easier are documents to work with than tables?

Ans:

Working with data in documents to be easier and more intuitive than working with data in tables.

9. What are the use cases for document databases?

Ans:

- Customer data management and personalization
- Internet of Things (IoT) and time-series data
- Product catalogs and content management
- Payment processing
- Mobile apps
- Real-time analytics.

10. What are document databases good for?

Ans:

Document databases are general-purpose databases that can be used in a variety of use cases across industries.

UNIT IV

Column-Family Stores: What Is a Column-Family Data Store, Features, Suitable Use Cases,

When Not to Use Graph Databases: What Is a Graph Database, Features, Suitable Use Cases, When Not to Use

4.1 COLUMN-FAMILY STORES

4.1.1 What Is a Column-Family Data Store

Q1. Explain in detail about Column-Family Data Store.

Ans: **(Imp.)**

A column store database is a type of database that stores data using a column-oriented model. A column store database can also be referred to as a:

- Column database
- Column family database
- Column oriented database
- Wide column store database
- Wide column store
- Columnar database
- Columnar store

The Structure of a Column Store Database:

Columns store databases use a concept called a keyspace. A key space is kind of like a schema in the relational model. The keyspace contains all the column families (kind of like tables in the relational model), which contain rows, which contain columns.

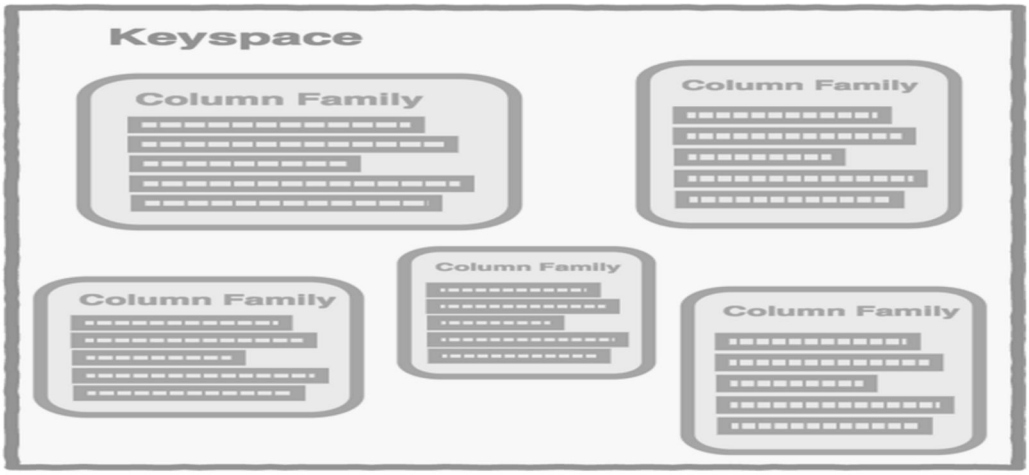


Fig. : A key space co: training column families

Here's a closer look at a column family:

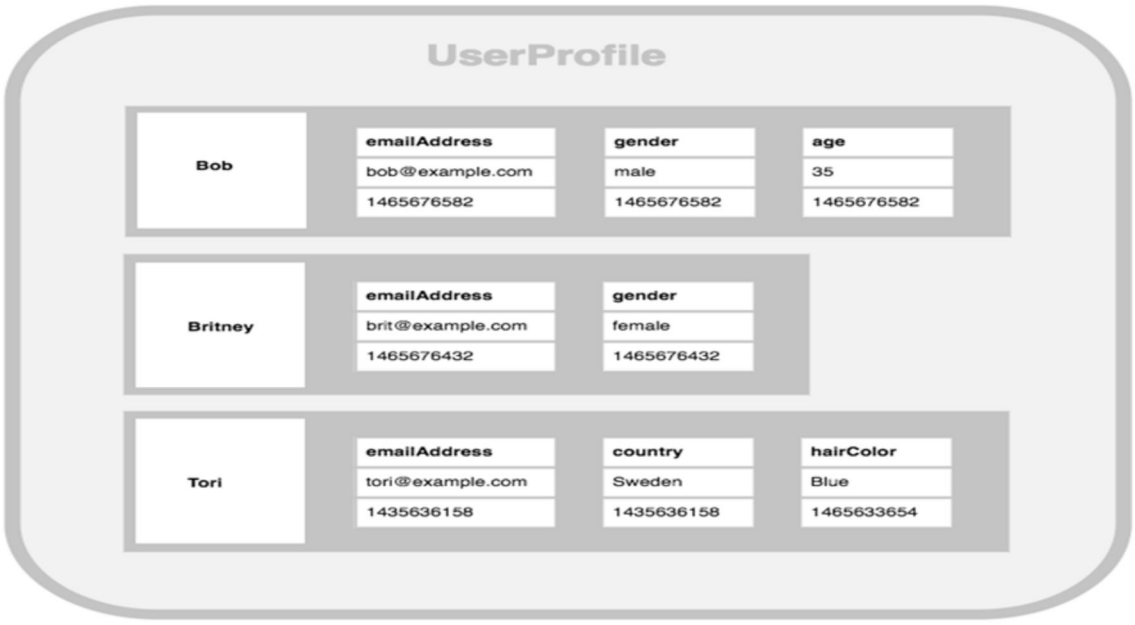
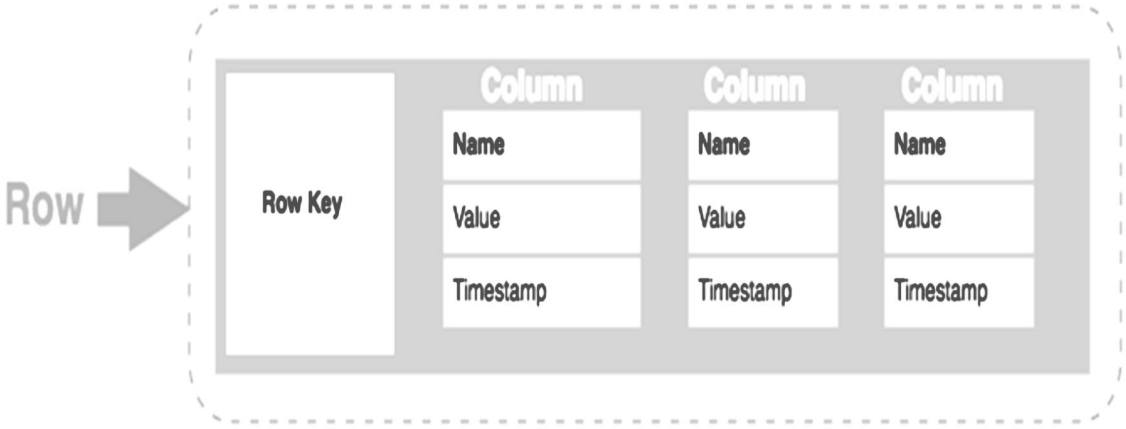


Fig. : A column family containing 3 rows. Each row contains its own set of columns.

As the above diagram shows:

- A column family consists of multiple rows.
- Each row can contain a different number of columns to the other rows. And the columns don't have to match the columns in the other rows (i.e. they can have different column names, data types, etc).
- Each column is contained to its row. It doesn't span all rows like in a relational database. Each column contains a name/value pair, along with a timestamp. Note that this example uses Unix/ Epoch time for the timestamp.

Here's how each row is constructed



Here's a breakdown of each element in the row:

- **Row Key**

  Each row has a unique key, which is a unique identifier for that row.

- **Column**

  Each column contains a name, a value, and timestamp.

- **Name**

  This is the name of the name/value pair.

- **Value**

  This is the value of the name/value pair.

- **Timestamp**

  This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

Some DBMSs expand on the column family concept to provide extra functionality/storage ability. For example, Cassandra has the concept of composite columns, which allow you to nest objects inside a column.

Benefits

Some key benefits of columnar data bases include:

- **Compression**

  Column stores are very efficient at data compression and/or partitioning.

- **Aggregation queries**

  Due to their structure, columnar databases perform particularly well with aggregation queries (such as SUM, COUNT, AVG, etc).

- **Scalability**

  Columnar databases are very scalable. They are well suited to massively parallel processing (MPP), which involves having data spread across a large cluster of machines – often thousands of machines.

- **Fast to load and query**

  Columnar stores can be loaded extremely fast. A billion-row table could be loaded within a few seconds. You can start querying and analysing almost immediately.

These are just some of the benefits that make columnar databases a popular choice for organizations dealing with big data.

- Examples of Column Store DBMSs
  - Bigtable
  - Cassandra
  - HBase
  - Vertica
  - Druid
  - Accumulo
  - Hypertable

Q2. What is column store database? Give its advantages and disadvantages.

Ans: **(Imp.)**

Meaning

A column store database is a type of database that stores data using a column-oriented model.

A columnar database is a database management system (DBMS) that stores data in columns instead of rows. The purpose of a columnar database is to efficiently write and read data to and from hard disk storage in order to speed up the time it takes to return a query. Columnar databases store data in a way that greatly improves disk I/O performance. They are particularly helpful for data analytics and data warehousing.

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.

These databases are most often utilized when there is a need for a large data model. They are very useful for data warehouses, or when there is a need for high performance or handling intensive querying.

Working of Mechanism

- 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.
- The number of columns pertaining to each row, or their name, can vary. In other words, not every column of a column family, and thus a database, has the same number of rows. In fact, even though they might share their name, each column is contained within one row and does not run across all rows.
- The column families are located in a keyspace. Each keyspace holds an entire NoSQL data store and it has a similar role or importance that a schema has for a relational database.
- However, as NoSQL datastores have no set structure, keyspaces represent a schemaless database containing the design of a data store and its own set of attributes.
- One of the most popular columnar databases available is MariaDB. It was created as a fork of MySQL intended to be robust and scalable, handle many different purposes and a large volume of queries.
- Apache Cassandra is another example of a columnar database handling heavy data loads across numerous servers, making the data highly available.
- Some of the other names on this list include Apache HBase, Hypertable and Druid specially designed for analytics. These databases support certain features of platforms such as Outbrain, Spotify and Facebook.

Types

- **Standard column family**

  This column family type is similar to a table; it contains a key-value pair where the key is the row key, and the values are stored in columns using their names as their identifiers.

- **Super column family**

  A super column represents an array of columns. Each super column has a name and a value mapping the super column out to several different columns. Related super columns are joined under a single row into super column families. Compared to a relational database, this is like a view of several different tables within a database. Imagine you had the view of the columns and values available for a single row that is a single identifier across many different tables and were able to store them all in one place: That is the super column family.

Advantages

- **Scalability**

  This is a major advantage and one of the main reasons this type of database is used to store big data. With the ability to be spread over hundreds of different machines depending on the scale of the database, it supports massively parallel processing. This means it can employ many processors to work on the same set of computations simultaneously.

- **Compression**

  Not only are they infinitely scalable, but they are also good at compressing data and thus saving storage.

- **Very responsive**

  The load time is minimal, and queries are performed fast, which is expected given that they are designed to hold big data and be practical for analytics.

Disadvantages

- **Online transactional processing**

  These databases are not very efficient with online transactional processing as much as they are for online analytical processing. This means they are not very good with updating transactions but are designed to analyze them. This is why they can be found holding data required for business analysis with a relational database storing data in the back end.

- **Incremental data loading**

  As mentioned above, typically column-oriented databases are used for analysis and are quick to retrieve data, even when processing complex queries, as it is kept close together in columns. While incremental data loads are not impossible, columnar databases do not perform them in the most efficient way. The columns first need to be scanned to identify the right rows and scanned further to locate the modified data which requires overwriting.

- **Row-specific queries**

  Like the potential downfalls mentioned above, it all boils down to the same issue, which is using the right type of database for the right purposes. With row-specific queries, you are introducing an extra step of scanning the columns to identify the rows and then locating the data to retrieve. It takes more time to get to individual records scattered in multiple columns, rather than accessing grouped records in a single column. Frequent row-specific queries might cause performance issues by slowing down a column-oriented database, which is particularly designed to help you get to required pieces of information quickly, thus defeating its purpose.

4.1.2 Features

Q3. What are the Features of Column Store Databases.

Ans: **(Imp.)**

Key benefits of column store databases:

Key benefits of column store databases include faster performance in load, search, and aggregate functions. Column store databases are scalable and can read billions of records in seconds. Column store databases are also efficient in data compression and partitioning.

Popular Column databases

Some of the popular column-oriented DBMS include Bigtable, Cassandra, HBase, Druid, Hypertable, MariaDB, Azure SQL Data Warehouse, Google BigQuery, IBM Db2, MemSQL, SQL Server, and SAP HANA.

Bigtable

Google Bigtable is a petabyte-scale, fully managed NoSQL database service for large analytical and operational workloads.

Key features include:

- Low latency, massively scalable NoSQL
- Consistent sub-10ms latency
- Replication provides higher availability, higher durability, and resilience in the face of zonal failures
- Ideal for Ad Tech, Fintech, and IoT
- Storage engine for machine learning applications
- Easy integration with open source big data tools

Cassandra

Apache Cassandra No SQL database is a highly scalable and highly available without compromising performance. Linear scalability and proven fault-tolerance on commodity hardware or cloud infrastructure make it the perfect platform for mission-critical data. Cassandra's support for replicating across multiple datacenters is best-in-class, providing lower latency for your users and the peace of mind of knowing that you can survive regional outages.

HBase

Apache HBase is an open-source, distributed, scalable, NoSQL big data store that allows billions of rows of big data access in seconds.

Key features include:

- Linear and modular scalability
- Strictly consistent reads and writes
- Automatic and configurable sharding of tables
- Automatic failover support between Region Servers
- Convenient base classes for backing Hadoop Map Reduce jobs with Apache HBase tables
- Easy to use Java API for client access
- Block cache and Bloom Filters for real-time queries
- Query predicate push down via server-side Filters
- Thrift gateway and a REST-ful Web service that supports XML, Protobuf, and binary data encoding options
- Extensible jruby-based (JIRB) shell.

4.1.3 Suitable Use Cases

Q4. What are the Suitable Use Cases of Column Store Data bases?

Ans :

Suitable Use Cases: some of the problems where column-family databases are a good fit.

(i) Event Logging

Column-family databases with their ability to store any data structures are a great choice to store event information, such as application state or errors encountered by the application. Within the enterprise, all applications can write their events to Cassandra with their own columns and the rowkey of the form appname: timestamp. Since we can scale writes, Cassandra would work ideally for an event logging system

(ii) Content Management Systems, Blogging Platforms

Using column families, you can store blog entries with tags, categories, links, and trackbacks in different columns. Comments can be either stored in the same row or moved to a different keyspace; similarly, blog users and the actual blogs can be put into different column families.

(iii) Counters

Often in web applications you need to count and categorize visitors of a page to calculate analytics. You can use the Counter Column Type during creation of a column family.

CREATE COLUMN FAMILY visit_counter

WITH default_validation_ class= Counter ColumnType

AND key_validation_class=UTF8 Type AND comparator=UTF8Type;

Once a column family is created, you can have arbitrary columns for each page visited within the web application for every user.

```
INCR visit_counter['mfowler'][home] BY 1;
INCR visit_counter['mfowler'][products] BY 1;
INCR visit_counter['mfowler'][contactus] BY 1;
```

Incrementing counters using CQL:

```
UPDATE visit_counter SET home = home + 1 WHERE KEY='mfowler'
```

4.1.4 When Not to Use

Q5. What is column store database? Explain about When Not to Use column-family databases.

Ans: **(Imp.)**

A column store database is a type of database that stores data using a column-oriented model.

A columnar database is a database management system (DBMS) that stores data in columns instead of rows. The purpose of a columnar database is to efficiently write and read data to and from hard disk storage in order to speed up the time it takes to return a query. Columnar databases store data in a way that greatly improves disk I/O performance. They are particularly helpful for data analytics and data warehousing.

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.

These databases are most often utilized when there is a need for a large data model. They are very useful for data warehouses, or when there is a need for high performance or handling intensive querying.

When Not to use column-family databases:

There are problems for which column-family databases are not the best solutions, such as systems that require ACID transactions for writes and reads. If you need the database to aggregate the data using queries (such as SUM or AVG), you have to do this on the client side using data retrieved by the client from all the rows.

Cassandra is not great for early prototypes or initial tech spikes: During the early stages, we are not sure how the query patterns may change, and as the query patterns change, we have to change the column family design. This causes friction for the product innovation team and slows down developer productivity. RDBMS impose high cost on schema change, which is traded off for a low cost of query change; in Cassandra, the cost may be higher for query change as compared to schema change.

4.2 When Not to use Graph Databases

4.2.1 What Is a Graph Database

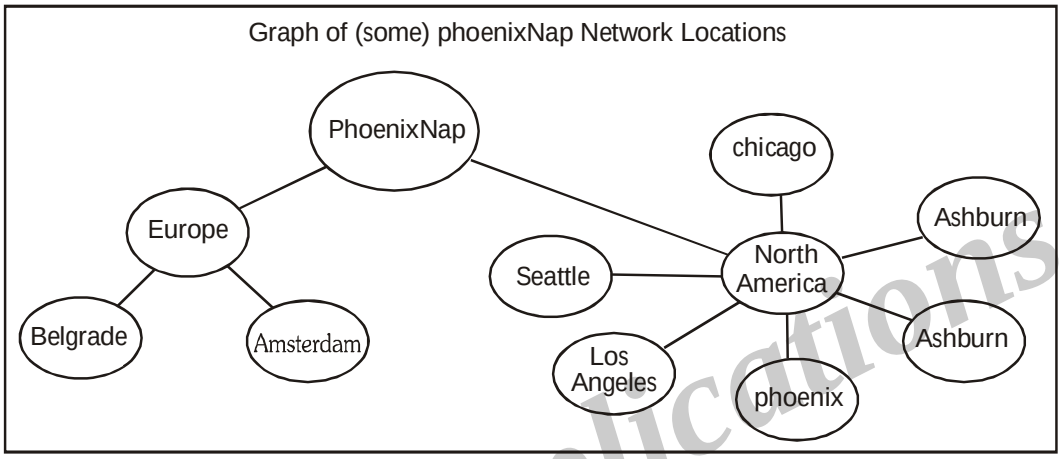
Q6. What Is a Graph Database?

Ans:

A graph database is a NoSQL-type data base system based on a topographical network structure. The idea stems from graph theory in mathematics, where graphs represent data sets using nodes, edges, and properties.

- Nodes or points are instances or entities of data which represent any object to be tracked, such as people, accounts, locations, etc.
- Edges or lines are the critical concepts in graph databases which represent relationships between nodes. The connections have a direction that is either unidirectional (one way) or bidirectional (two way).
- Properties represent descriptive information associated with nodes. In some cases, edges have properties as well.

For example, analyze some of the network locations



Nodes with descriptive properties form relationships represented by edges. Graph databases provide a conceptual view of data more closely related to the real world. Modeling complex connections becomes easier since relationships between data points are given an equal value of importance as the data itself.

Working of Graph Databases

Graph databases work by treating data and relationships between data equally. Related nodes are physically connected, and the physical connection is also treated as a piece of data.

Modeling data in this way allows querying relationships in the same manner as querying the data itself. Instead of calculating and querying the connection steps, graph databases read the relationship from storage directly.

Graph databases are more closely related to other NoSQL data modeling techniques in terms of agility, performance, and flexibility. Like other NoSQL databases, graphs do not have schemas, which makes the model flexible and easy to alter along the way.

The future of graph databases

Graph databases and graph techniques have been evolving as compute power and big data have increased over the past decade. In fact, it's become increasingly clear that they will become the standard tool for analyzing a brave new world of complex data relationships. As businesses and organizations continue pushing the capabilities of big data and analysis, the ability to derive insights in increasingly complex ways makes graph databases a must-have for today's needs and tomorrow's successes.

Oracle makes it easy to adopt graph technologies. Oracle Database and Oracle Autonomous Database include a graph database and graph analytics engine so users can discover more insights in their data by using the power of graph algorithms, pattern matching queries, and visualization. Graphs are part of

Oracle's converged database, which supports multimodel, multiworkload, and multi-tenant requirements–all in a single database engine.

Although all graph databases claim they are high-performance, Oracle's graph offerings are performant both in query performance and algorithms, as well as tightly integrated with Oracle database. This makes it easy for developers to add graph analytics to existing applications and make use of the scalability, consistency, recovery, access control, and security that the database provides by default.

4.2.2 Features

Q7. What are the features of Graph Data bases?

Ans: **(Imp.)**

Graph databases became more popular with the rise of big data and social media analytics. Many multi-model databases support graph modeling. However, there are numerous graph native databases available as well.

The Following features of Graph Databases:

1. Janus Graph

Janus Graph is a distributed, open-source and scalable graph database system with a wide range of integration options catered to big data analytics. Some of the main features of JanusGraph include:

- Support for ACID transactions with the ability to bear thousands of concurrent users.
- Multiple options for storing the graph data, such as Cassandra or HBase.
- Complex search available by default as well as optional support for Elasticsearch.
- Full integration with Apache Spark for advanced data analytics.
- JanusGraph uses the graph transversal query language Gremlin, which is Turing complete.

2. Neo4j

Neo4j (Network Exploration and Optimization 4 Java) is a graph database written in Java with native graph storage and processing. The main features of Neo4j are:

- The database is scalable through data partitioning into pieces known as shards.
- High availability provided through continuous backups and rolling upgrades.
- Multiple instances of databases are separable while remaining on one dedicated server, providing a high level of security.
- Neo4j uses the Cypher graph query language, which is programmer friendly.

3. DGraph

DGraph (Distributed graph) is an open-source distributed graph database system designed with scalability in mind. Some exciting features of DGraph include:

- Horizontal scalability for running in production with ACID transactions.
- DGraph is an open-source system with support for many open standards.
- The query language is GraphQL, which is designed for APIs.

4. DataStax Enterprise Graph

The DataStax Enterprise Graph is a distributed graph database based on Cassandra and optimized for enterprises. Features include:

- Data Stax provides continuous availability for enterprise needs.
- The database integrates with offline Apache Spark seamlessly.
- Real-time search and analytics are fully integrated.
- Scalability available through multiple data centers.
- It supports Gremlin as well as CQL for querying.

Q8. State Advantages and Disadvantages of graph database.

Ans :

Every database type comes with strengths and weaknesses. The most important aspect is to know the differences as well as available options for specific problems. Graph databases are a growing technology with different objectives than other database types.

Advantages

Some advantages of graph databases include:

- The structures are agile and flexible.
- The representation of relationships between entities is explicit.
- Queries output real-time results. The speed depends on the number of relationships.

Disadvantages

The general disadvantages of graph databases are:

- There is no standardized query language. The language depends on the platform used.
- Graphs are inappropriate for transactional-based systems.
- The user-base is small, making it hard to find support when running into a problem.

4.2.3 Suitable Use Cases

Q9. What are the Use Cases of Graph Data bases? Explain its advantages and disadvantages?

Ans: **(Imp.)**

Graph Database Use Cases

There are many notable examples where graph databases outperform other database modeling techniques, some of which include:

- **Real-Time Recommendation Engines**

  Real-time product and ecommerce recommendations provide a better user experience while maximizing profitability. Notable cases include Netflix, eBay, and Walmart.

- **Master Data Management**

  Linking all company data to one location for a single point of reference provides data consistency and accuracy. Master data management is crucial for large-scale global companies.

- **GDPR and regulation compliances**

  Graphs make tracking of data movement and security easier to manage. The databases reduce the potential of data breaches and provide better consistency when removing data, improving the overall trust with sensitive information.

- **Digital asset management**

  The amount of digital content is massive and constantly increasing. Graph databases provide a scalable and straightforward database model to keep track of digital assets, such as documents, evaluations, contracts, etc.

- **Context-aware services**

  Graphs help provide services related to actual-world characteristics. Whether it is natural disaster warnings, traffic updates, or product recommendations for a given location, graph databases offer a logical solution to real-life circumstances.

- **Fraud detection**

  Finding suspicious patterns and uncovering fraudulent payment transactions is done in real-time using graph databases. Targeting and isolating parts of graphs provide quicker detection of deceptive behavior.

- **Semantic search**

  Natural language processing is ambiguous. Semantic searches help provide meaning behind keywords for more relevant results, which is easier to map using graph databases.

- **Network management**

  Networks are linked graphs in their essence. Graphs reduce the time needed to alert a network administrator about problems in a network.

- **Routing**

  Information travels through a network by finding optimal paths makes graph databases the perfect choice for routing.

4.2.4 When Not to Use

Q10. Explain when not use Graph Databases.

Ans:

If you need to run frequent table scans and searches for data that fits defined categories, a graph database wouldn't be very helpful. Graph databases are well equipped to traverse relationships when you have a specific starting point or at least a set of points to start with (nodes with the same label). They are not suited for traversing the whole graph often. While it's possible to run such queries, other storage solutions may be more optimized for such bulk scans.

If the majority of the queries in our example include searches by property values over the entire network, then a graph database wouldn't be the right fit.

2. When you Need Key/Value Storage

Very often, databases are used to lookup information stored in key/value pairs. When you have a known key and need to retrieve the data associated with it, a graph database is not particularly useful.

For example, if the sole purpose of your database is storing a user's personal information and retrieving it by name or ID, then refrain from using a graph. But if there were other entities involved (visited locations for example), and a large number of connections is required to map them to users, then a graph database could bring performance benefits. A good rule of thumb is, if most of your queries return a single node via a simple identifier (key), then just skip graph databases.

3. When you Need to Store Large Chunks of Information

If the entities in your model have very large attributes like BLOBs, CLOBs, long texts… then graph databases aren't the best solution. While you can store those objects as nodes and link them to other nodes to utilize the power of traversing relationships, sometimes it just makes more sense to store them directly with the entities they are connected to.

When to Use Graph Database:

- Graph databases should be used for heavily interconnected data.
- It should be used when amount of data is larger and relationships are present.
- It can be used to represent the cohesive picture of the data.

Short Question & Answers

1. What is a Column-Family Data Store?

Ans:

A column store database is a type of database that stores data using a column-oriented model.

The Structure of a Column Store Database:

Columns store databases use a concept called a keyspace. A keyspace is kind of like a schema in the relational model. The keyspace contains all the column families (kind of like tables in the relational model), which contain rows, which contain columns.

A column family consists of multiple rows:

- Each row can contain a different number of columns to the other rows. And the columns don't have to match the columns in the other rows (i.e. they can have different column names, data types, etc).
- Each column is contained to its row. It doesn't span all rows like in a relational database. Each column contains a name/value pair, along with a timestamp. Note that this example uses Unix/Epoch time for the timestamp.

Here's a breakdown of each element in the row:

- **Row Key**

  Each row has a unique key, which is a unique identifier for that row.

- **Column**

  Each column contains a name, a value, and timestamp.

- **Name**

  This is the name of the name/value pair.

- **Value**

  This is the value of the name/value pair.

- **Timestamp**

  This provides the date and time that the data was inserted. This can be used to determine the most recent version of data.

2. What are the advantages and disadvantages of column-oriented data bases?

Ans:

Advantages

- **Scalability**

  This is a major advantage and one of the main reasons this type of database is used to store big data. With the ability to be spread over hundreds of different machines depending on the scale of the database.

- **Compression**

  Not only are they infinitely scalable, but they are also good at compressing data and thus saving storage.

- **Very responsive**

  The load time is minimal, and queries are performed fast, which is expected given that they are designed to hold big data and be practical for analytics.

Disadvantages

- **Online transactional processing**

  These databases are not very efficient with online transactional processing as much as they are for online analytical processing. This means they are not very good with updating transactions but are designed to analyze them

- **Incremental data loading**

  As mentioned above, typically column-oriented databases are used for analysis and are quick to retrieve data, even when processing complex queries, as it is kept close together in columns.

- **Row-specific queries**

  Like the potential downfalls mentioned above, it all boils down to the same issue, which is using the right type of database for the right purposes. With row-specific queries, you are introducing an extra step of scanning the columns to identify the rows and then locating the data to retrieve.

3. What is a Graph Database?

Ans:

A graph database is a NoSQL-type data base system based on a topographical network structure. The idea stems from graph theory in mathematics, where graphs represent data sets using nodes, edges, and properties.

- Nodes or points are instances or entities of data which represent any object to be tracked, such as people, accounts, locations, etc.
- Edges or lines are the critical concepts in graph databases which represent relationships between nodes. The connections have a direction that is either unidirectional (one way) or bidirectional (two way).
- Properties represent descriptive information associated with nodes. In some cases, edges have properties as well.

4. How Graph and Graph Databases Work? Give the examples of Graph Databases

Ans:

Graph databases provide graph models They allow users to perform traversal queries since data is connected. Graph algorithms are also applied to find patterns, paths and other relationships this enabling more analysis of the data. The algorithms help to explore the neighboring nodes, clustering of vertices analyze relationships and patterns. Countless joins are not required in this kind of database.

Example of Graph Database

- Recommendation engines in E commerce use graph databases to provide customers with accurate recommendations, updates about new products thus increasing sales and satisfying the customer's desires.
- Social media companies use graph databases to find the "friends of friends" or products that the user's friends like and send suggestions accordingly to user.
- To detect fraud Graph databases, play a major role. Users can create graph from the transactions between entities and store other important information. Once created, running a simple query will help to identify the fraud.

5. What are the Advantages and Disadvantages of Graph Database?

Ans:

Advantages

- Potential advantage of Graph Database is establishing the relationships with external sources as well
- No joins are required since relationships is already specified.
- Query is dependent on concrete relationships and not on the amount of data.
- It is flexible and agile.
- it is easy to manage the data in terms of graph.

Disadvantages

- Often for complex relationships speed becomes slower in searching.
- The query language is platform dependent.
- They are inappropriate for transactional data
- It has smaller user base.

6. What are the types of Graph Databases?

Ans:

Types

- **Property Graphs**

  These graphs are used for querying and analyzing data by modelling the relationships among the data. It comprises of vertices that has information about the particular subject and edges that denote the relationship. The vertices and edges have additional attributes called properties.

- **RDF Graphs**

  It stands for Resource Description Framework. It focuses more on data integration. They are used to represent complex data with well-defined semantics. It is represented by three elements: two vertices, an edge that reflect the subject, predicate and object of a sentence. Every vertex and edge is represented by URI (Uniform Resource Identifier).

7. What are key benefits of Column Store Databases?

Ans:

Benefits

Some key benefits of columnar databases include:

- Compression. Column stores are very efficient at data compression and/or partitioning.
- Aggregation queries. Due to their structure, columnar databases perform particularly well with aggregation queries (such as SUM, COUNT, AVG, etc).
- Scalability. Columnar databases are very scalable. They are well suited to massively parallel processing (MPP), which involves having data spread across a large cluster of machines – often thousands of machines.
- Fast to load and query. Columnar stores can be loaded extremely fast. A billion-row table could be loaded within a few seconds. You can start querying and analysing almost immediately.

These are just some of the benefits that make columnar databases a popular choice for organisations dealing with big data.

8. What are key features of Column Store Databases

Ans:

Features

- Low latency, massively scalable NoSQL
- Consistent sub-10ms latency
- Replication provides higher availability, higher durability, and resilience in the face of zonal failures
- Ideal for Ad Tech, Fintech, and IoT
- Storage engine for machine learning applications
- Easy integration with open source big data tools

9. What are differences between Graph and Relational Databases?

Ans:

The following table outlines the critical differences between graph and relational databases:

Type	Graph	Relational
Format	Nodes and edges with properties	Tables with rows and columns
Relationships	Represented with edges between nodes	Created using foreign keys between tables
Flexibility	Flexible	Rigid
Complex queries	Quick and responsive	Requires complex joins
Use-case	Systems with highly connected relationships	Transaction focused systems with more straightforward relationships

10. What are the graph database use cases with an examples?

Ans:

Graph Database Use Case Examples

There are many notable examples where graph databases outperform other database modeling techniques, some of which include:

- **Real-Time Recommendation Engines**

  Real-time product and ecommerce recommendations provide a better user experience while maximizing profitability. Notable cases include Netflix, eBay, and Walmart.

- **Master Data Management**

  Linking all company data to one location for a single point of reference provides data consistency and accuracy. Master data management is crucial for large-scale global companies.

- **GDPR and regulation compliances**

  Graphs make tracking of data movement and security easier to manage. The databases reduce the potential of data breaches and provide better consistency when removing data, improving the overall trust with sensitive information.

- **Digital asset management**

  The amount of digital content is massive and constantly increasing. Graph databases provide a scalable and straightforward database model to keep track of digital assets, such as documents, evaluations, contracts, etc.

- **Context-aware services**

  Graphs help provide services related to actual-world characteristics. Whether it is natural disaster warnings, traffic updates, or product recommendations for a given location, graph databases offer a logical solution to real-life circumstances.

- **Fraud detection**

  Finding suspicious patterns and uncovering fraudulent payment transactions is done in real-time using graph databases. Targeting and isolating parts of graphs provide quicker detection of deceptive behavior.

- **Semantic search**

  Natural language processing is ambiguous. Semantic searches help provide meaning behind keywords for more relevant results, which is easier to map using graph databases.

- **Network management**

  Networks are linked graphs in their essence. Graphs reduce the time needed to alert a network administrator about problems in a network.

- **Routing**

  Information travels through a network by finding optimal paths makes graph databases the perfect choice for routing.

Choose the Correct Answers

1. The full form of 'CRUD' is _____. [d]
   (a) Create-Read-Update-Define (b) Create-Read-Update-Deliver
   (c) Create-Run-Update-Delete (d) Create-Read-Update-Delete

2. _____ distributes different data across multiple servers. [b]
   (a) None of the options (b) Bucketing
   (b) Sharding (d) Partitioning

3. NoSQL databases are designed to expand _____. [d]
   (a) with increase of load (b) vertically
   (c) hardware wise (d) horizontally

4. The key-value pair data storages include all, except _____. [b]
   (a) in-memory (b) Network Attached Storage
   (b) Disk Storage (c) Cache Storage

5. Cassandra has properties of both _____ and _____. [c]
   (a) Kudu and Hbase (b) Neo4j / Amazon Dynamo
   (c) Google Bigtable / Amazon Dynamo (d) MongoDB / Google BigTable

6. A Riak Convergent Replicated Data Type (CRDT) includes _____. [c]
   (a) JSON/Maps/Sets (b) JSON/Maps
   (c) Maps/Sets/Counters (d) Sets

7. The RDBMS 'table' equivalent terminology in Riak is _____. [c]
   (a) data store (b) data segment
   (c) bucket (d) key-value

8. In column-oriented stores, data is stored on a _____ basis. [b]
   (a) Column (b) Column Family
   (c) Row Key (d) Row

9. JSON documents are built up of _____. [c]
   (a) arrays (b) objects
   (c) All the options (d) values

10. Graph databases are generally built for use with _____. [b]
    (a) Data Warehouse Batch Processing (b) OLTP
    (c) OLAP (d) All the options

Fill in the Blanks

1. In column-oriented stores, data is stored on a _____basis.
2. Cypher query language is associated with _____.
3. _____ is a Columnar database that runs on a Hadoop cluster.
4. _____ in Key-Value Databases are similar to 'Tables' in RDBMS.
5. Cassandra was developed by _____.
6. NoSQL can handle _____.
7. Hash Table Design is similar to _____.
8. Example(s) of Columnar Database is/are _____.
9. In the Master-Slave Replication model, the slave node services _____.
10. The Property Graph Model is similar to _____.

ANSWERS

1. Column Family
2. Neo4j
3. Apache HBase
4. Buckets
5. Facebook
6. Unstructured and Semi-structured data
7. Key Value datastore
8. Cassandra and HBase
9. Read operations
10. Entity Relationship Diagram

One Mark Answers

1. Give few examples of Column-Family Stores.

Ans :

Cassandra, HBase, Hypertable, and Amazon SimpleDB

2. What Is a Column-Family Data Store?

Ans :

Column-family databases store data in column families as rows that have many columns associated with a row key.

3. What are Columnar Databases?

Ans :

A column data store is also known as a column-oriented DBMS. Column store DBMS store data in columns rather than rows.

4. What is Keyspace?

Ans :

Column store DBMS uses a keyspace that is like a database schema in RDBMS. The keyspace contains all the column families.

5. What are the Key benefits of column store databases?

Ans :

Key benefits of column store databases include faster performance in load, search, and aggregate functions.

6. What is HBase?

Ans :

Apache HBase is an open-source, distributed, scalable, NoSQL big data store that allows billions of rows of big data access in seconds.

7. What is Bigtable?

Ans :

Google Bigtable is a petabyte-scale, fully managed NoSQL database service for large analytical and operational workloads.

8. What is a graph database?

Ans :

A graph database is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data.

9. How does a graph database store the data?

Ans :

A graph database uses a graph data structure to store data. This means that data is stored as nodes and edges.

10. What are the advantages of using graph data structures to represent relationships between entities?

Ans :

First, they can be used to easily represent complex relationships. Second, they offer a high degree of flexibility.

Lab Practicals

1. Installation of NoSQL Databases: Redis,MongoDB, Cassandra, Neo4j on Windows

Ans :

(i) How to Install Redis on Windows

You can run Redis on Windows 10 using Windows Subsystem for Linux(a.k.a WSL2). WSL2 is a compatibility layer for running Linux binary executables natively on Windows 10 and Windows Server 2019. WSL2 lets developers run a GNU/Linux environment (that includes command-line tools, utilities, and applications) directly on Windows.

Follow these instructions to run a Redis database on Microsoft Windows 10.

Step 1: Turn on Windows Subsystem for Linux

In Windows 10, Microsoft replaced Command Prompt with PowerShell as the default shell. Open PowerShell as Administrator and run this command to enable Windows Subsystem for Linux (WSL):

Enable-WindowsOptionalFeature-Online-FeatureName Microsoft-Windows-Subsystem-Linux

Reboot Windows after making the change — note that you only need to do this once.

Step 2: Launch Microsoft Windows Store

start ms-windows-store:

Then search for Ubuntu, or your preferred distribution of Linux, and download the latest version.

Step 3: Install Redis server

Installing Redis is simple and straightforward. The following example works with Ubuntu (you'll need to wait for initialization and create a login upon first use):

sudo apt-add-repository ppa:redislabs/redis

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install redis-server

Please note that the sudo command might or mightn't be required based on the user configuration of your system.

Step 4: Restart the Redis server

Restart the Redis server as follows:

sudo service redis-server restart

Step 5: Verify if your Redis server is running

Use the redis-cli command to test connectivity to the Redis database.

$ redis-cli

127.0.0.1:6379> set user:1 "Jane"

127.0.0.1:6379> get user:1

"Jane"

Please note: By default, Redis has 0-15 indexes for databases, you can change that number databases NUMBER in redis.conf.

Step 6: Stop the Redis Server

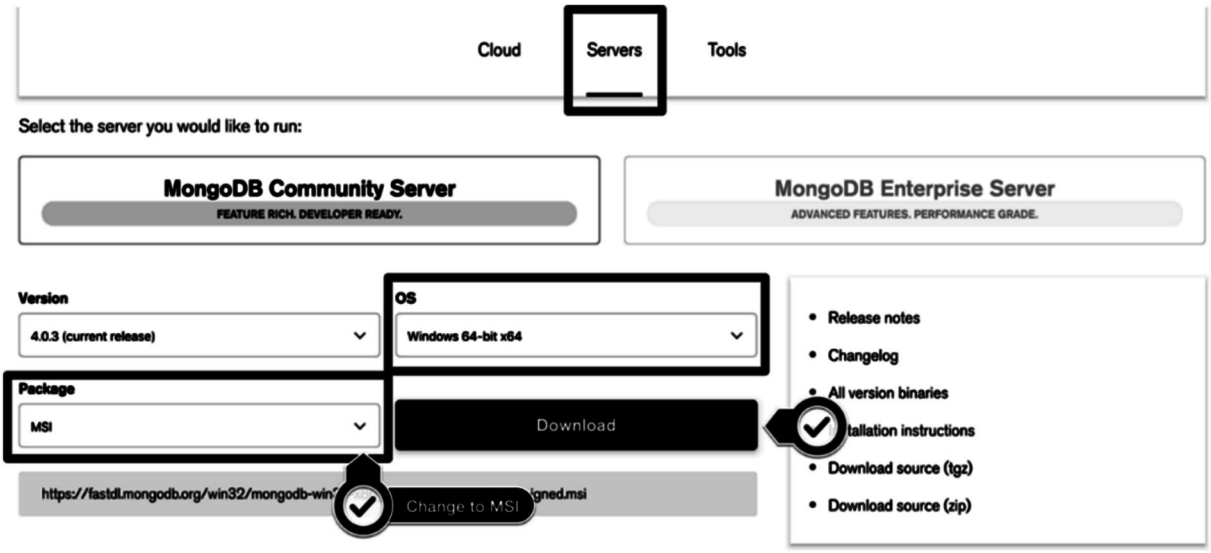
sudo service redis-server stop

(ii) Installation MongoDB

How to Download & Install MongoDB on Windows

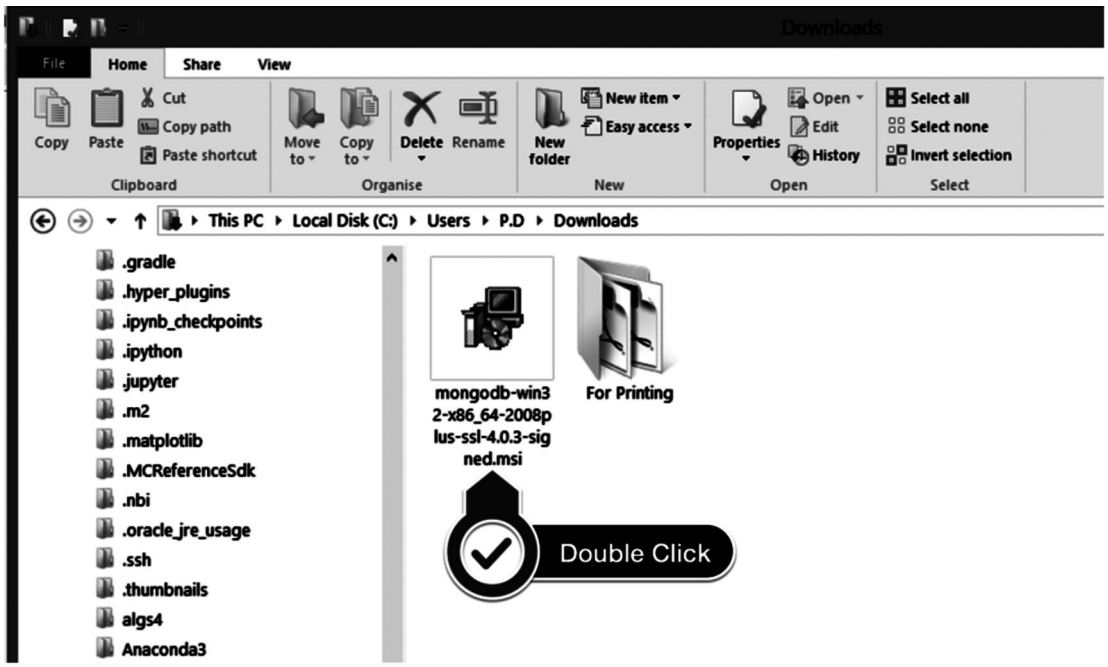
Step 1: Download the MongoDB MSI Installer Package

Head over here and download the current version of MongoDB. Make sure you select MSI as the package you want to download.



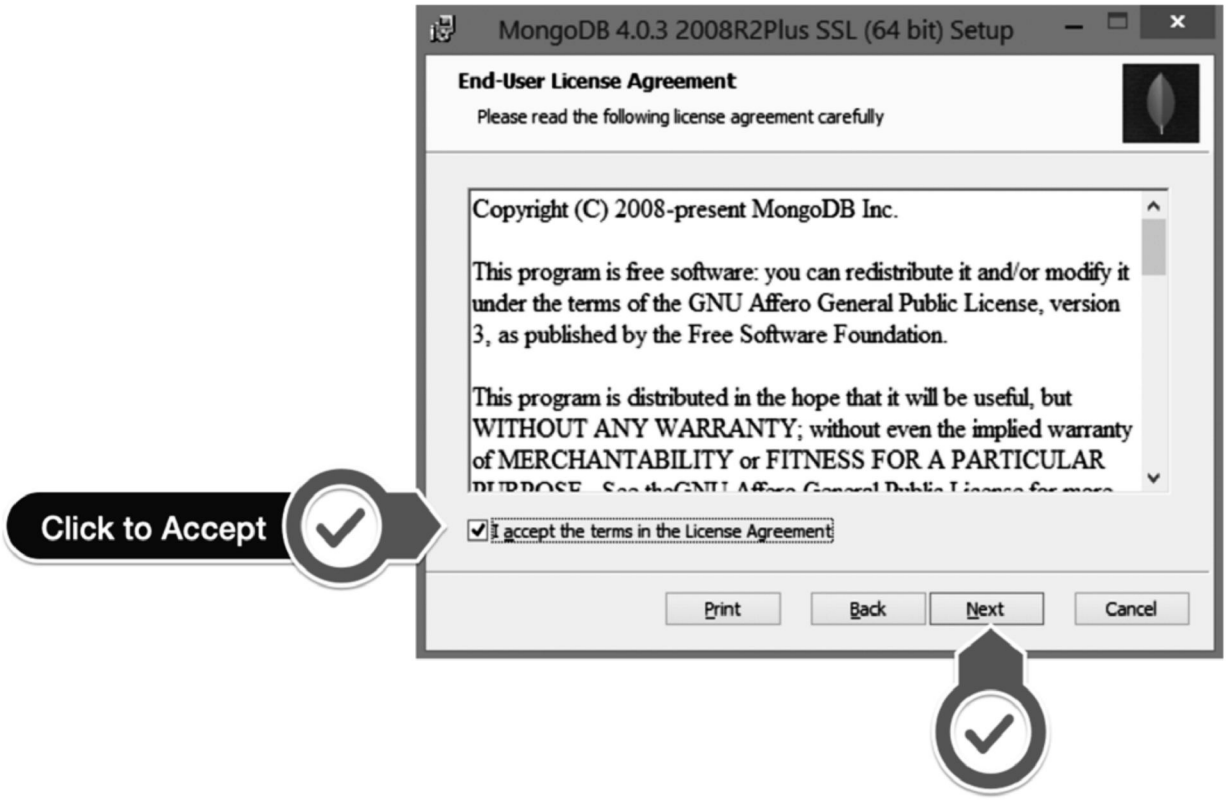
Step 2: Install MongoDB with the Installation Wizard

Make sure you are logged in as a user with Admin privileges. Then navigate to your downloads folder and double click on the .msi package you just downloaded. This will launch the installation wizard.

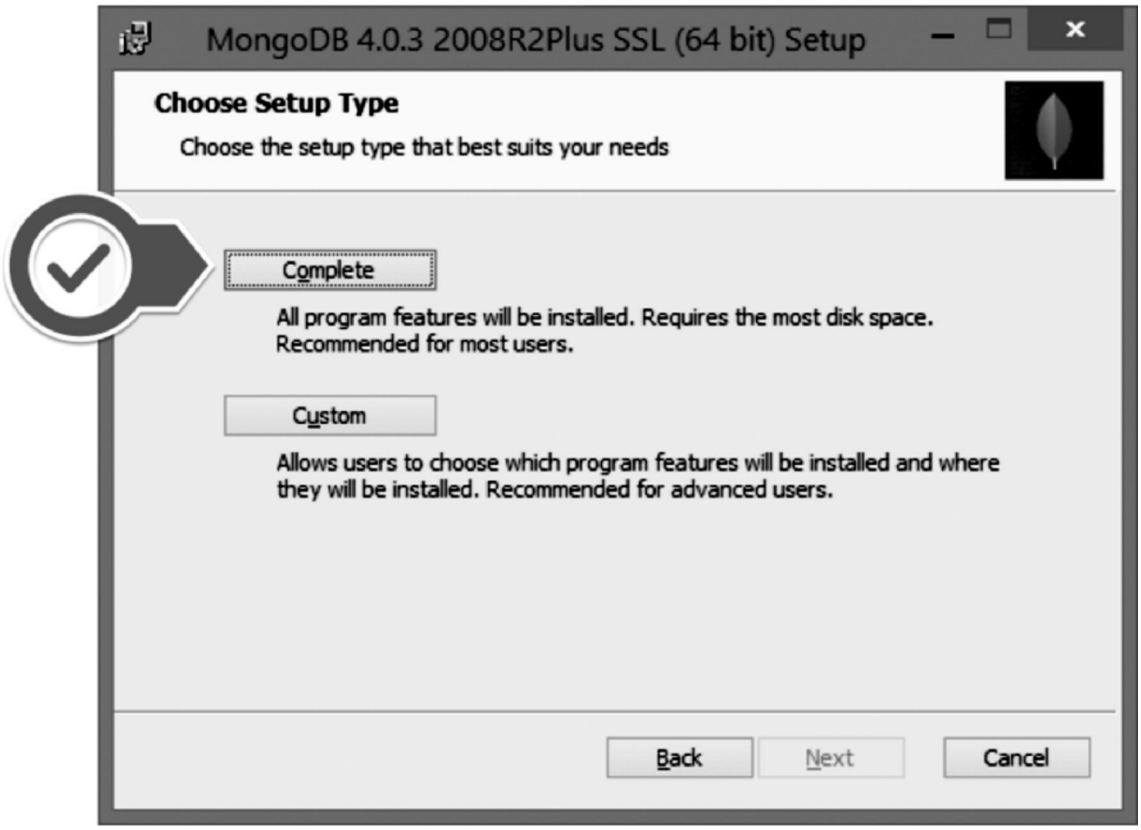


B. Click Next to start installation

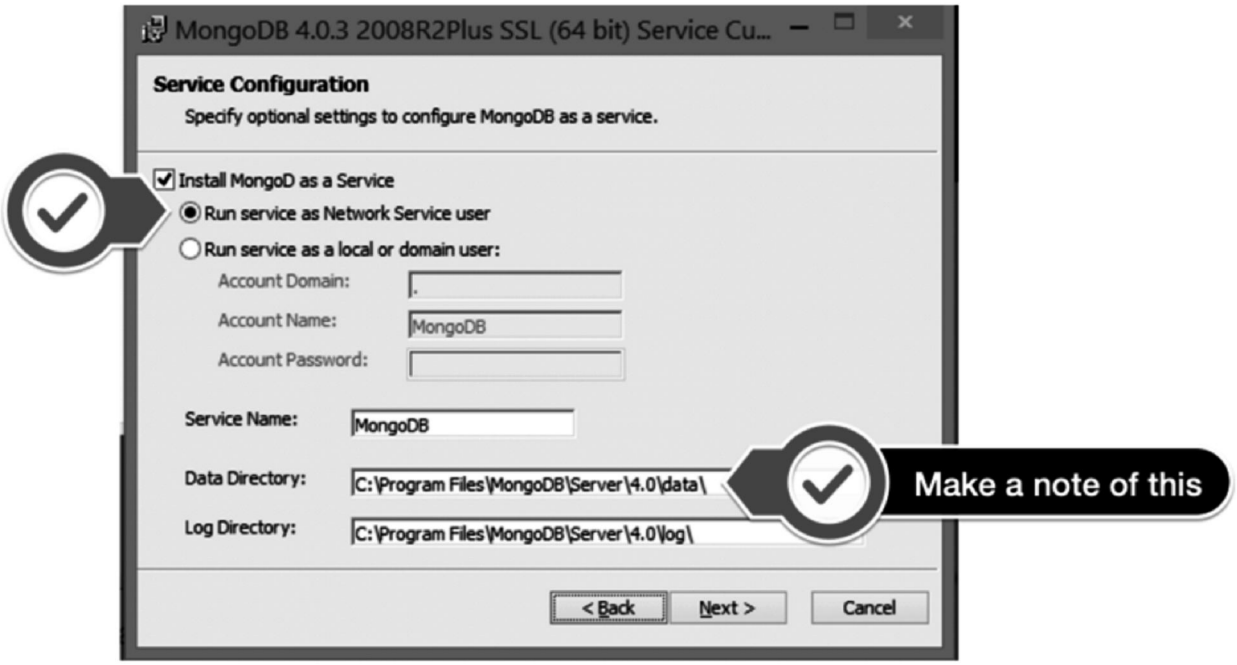
C. Accept the licence agreement then click Next.



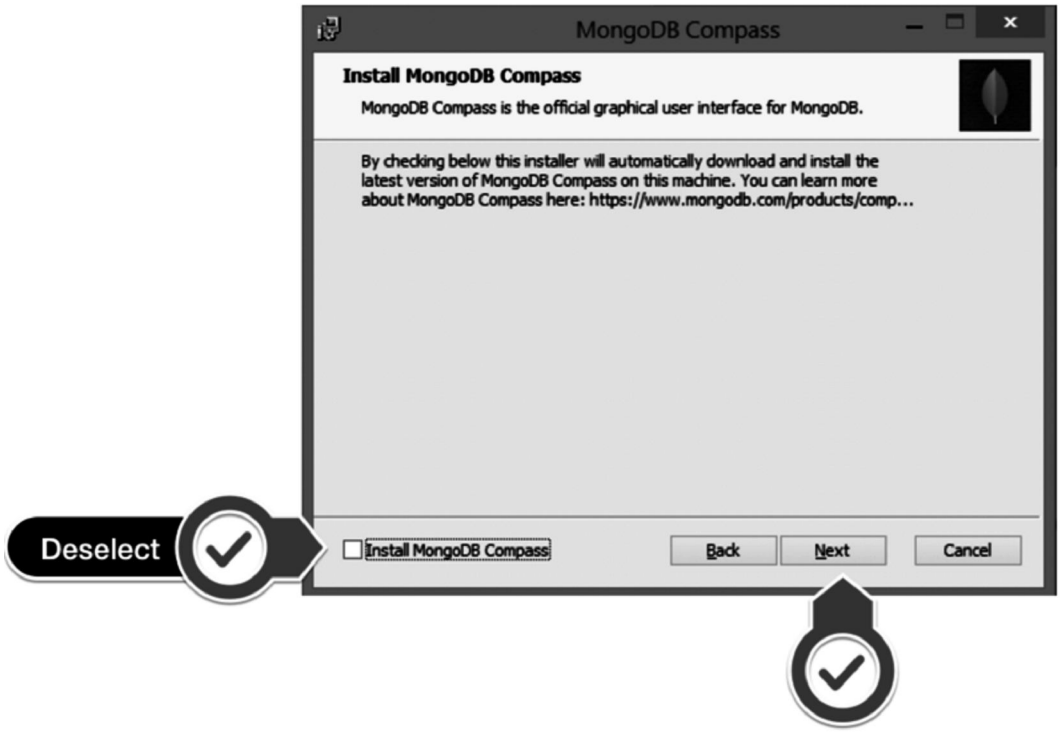
D. Select the Complete setup.



E Select "Run service as Network Service user" and make a note of the data directory, we'll need this later.

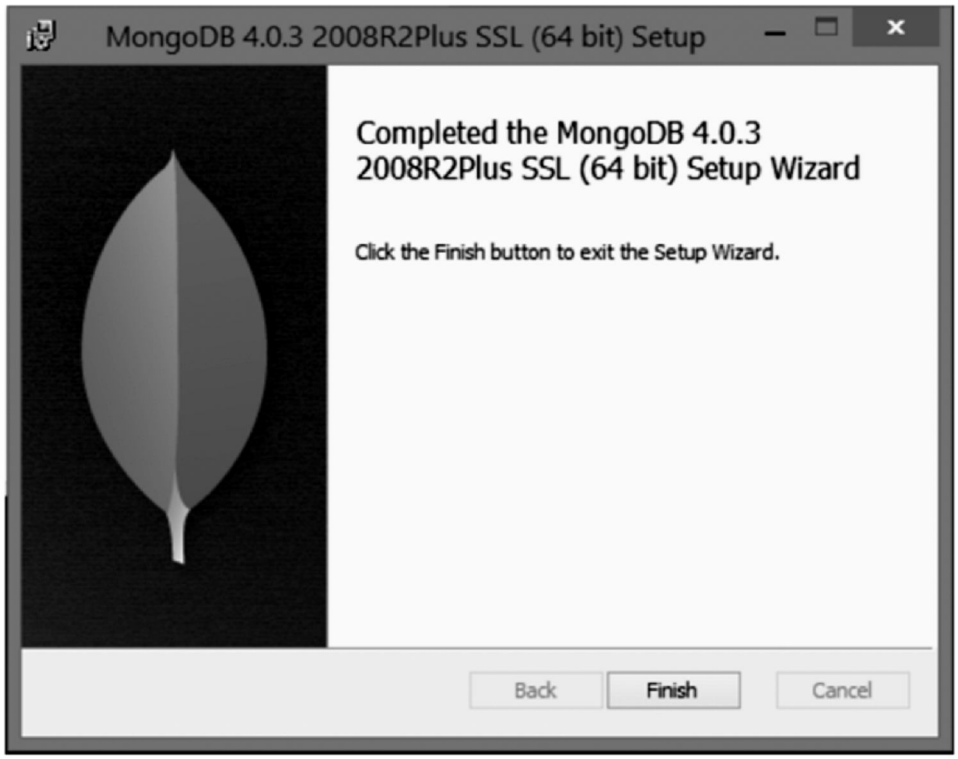


F. We won't need Mongo Compass, so deselect it and click Next.



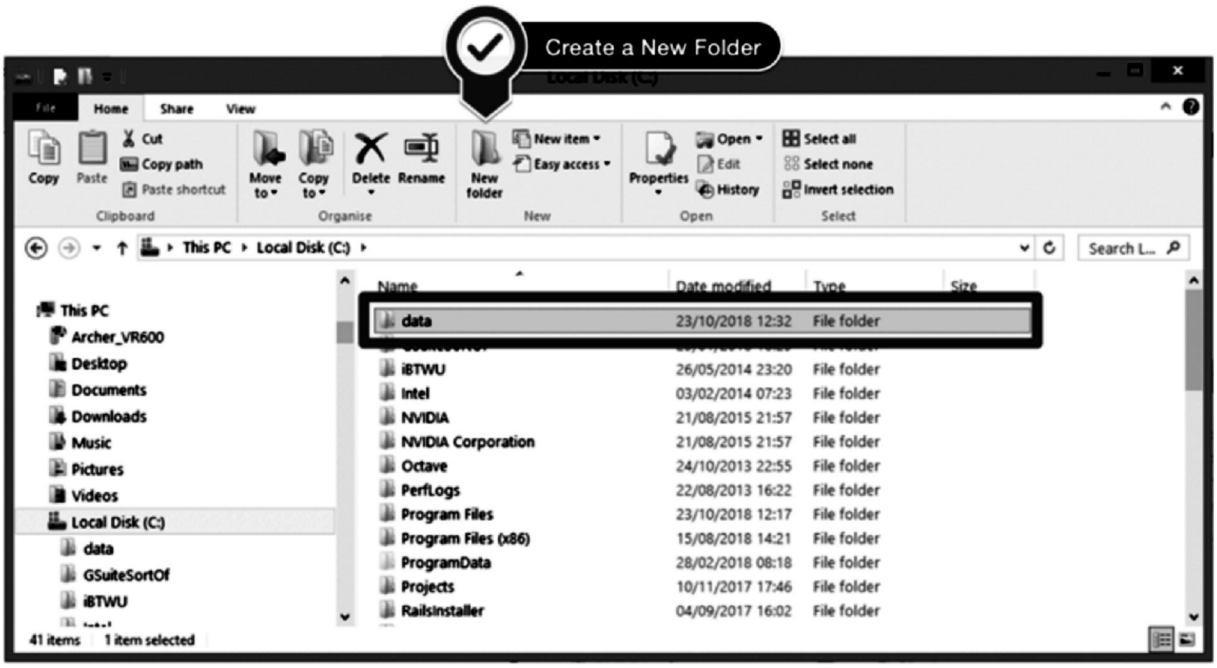
G. Click Install to begin installation.

F. Hit Finish to complete installation.



Step 3— Create the Data Folders to Store our Databases

A. Navigate to the C Drive on your computer using Explorer and create a new folder called data here.



B. Inside the data folder you just created, create another folder called db.

Step 4: Setup Alias Shortcuts for Mongo and Mongod

Once installation is complete, we'll need to set up MongoDB on the local system.

A. Open up your Hyper terminal running Git Bash.

B. Change directory to your home directory with the following command: cd ~

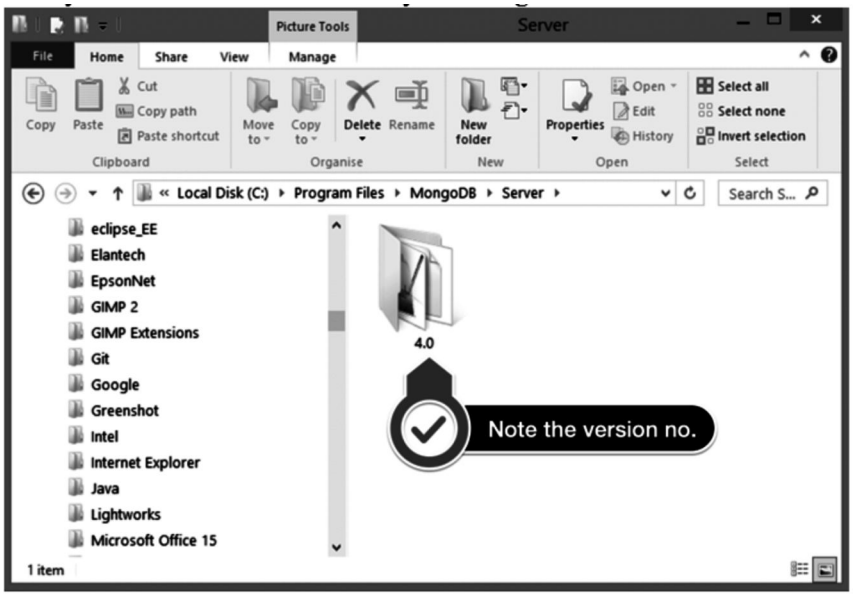
C. Here, we're going to create a file called .bash_profile using the following command: touch .bash_profile

D. Open the newly created .bash_profile with vim using the following command: vim .bash_profile

E. In vim, hit the I key on the keyboard to enter insert mode.

F. In your explorer go to C '! Program Files '! MongoDB '! Server

Now you should see the version of your MongoDB.



G. Paste in the following code into vim, make sure your replace the 4.0 with your version that you see in explorer

alias mongod="/c/Program\ files/MongoDB/Server/4.0/bin/mongod.exe"

alias mongo="/c/Program\ Files/MongoDB/Server/4.0/bin/mongo.exe"

F. Hit the Escape key on your keyboard to exit the insert mode. Then type :wq! to save and exit Vim.

Step 5: Verify That Setup was Successful

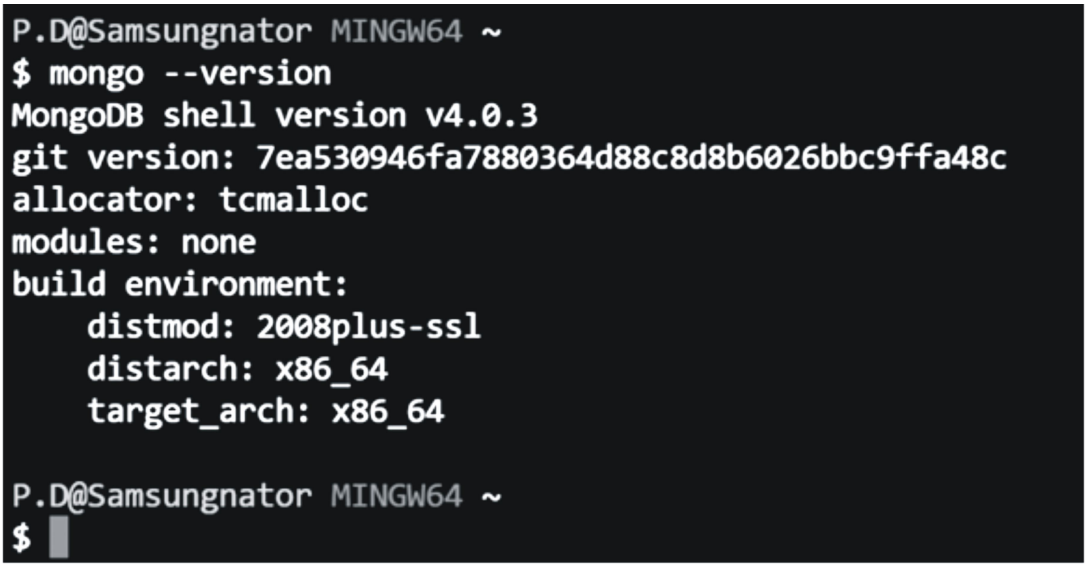
A. Close down the current Hyper terminal and quit the application.

B. Re-launch Hyper.

C. Type the following commands into the Hyper terminal:

mongo —version

Once you've hit enter, you should see something like this:



This means that you have successfully installed and setup MongoDB on your local system!

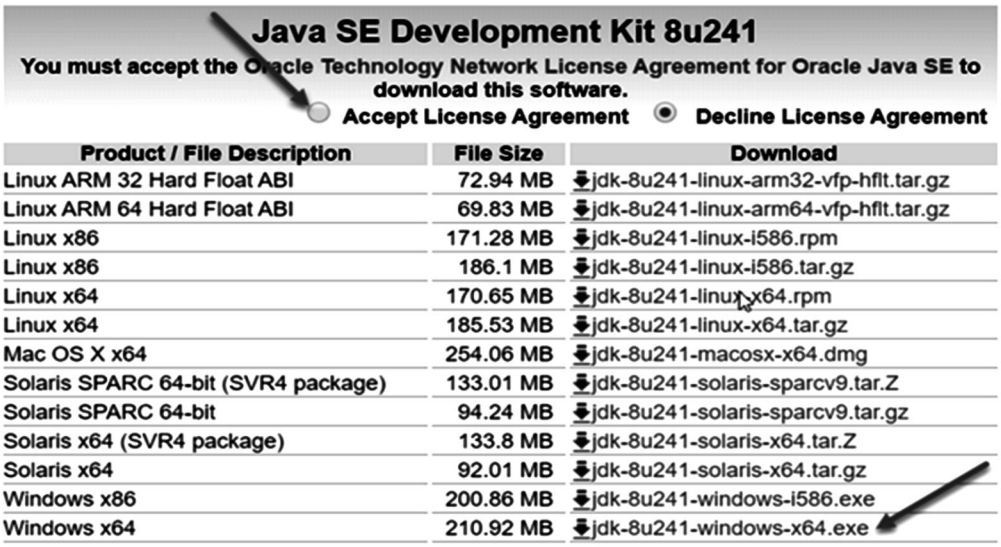
(iii) Cassandra installation

The software required for the Cassandra installation are:

1. JDK 8 (Java SE Development Kit 8)
2. Python 2.7or latest versions
3. Apache Cassandra.

JDK 8 Installation

1. Download the latest JDK 8 here. When this article was written, the latest version was JDK 8u241.



Java SE Development Kit 8u241

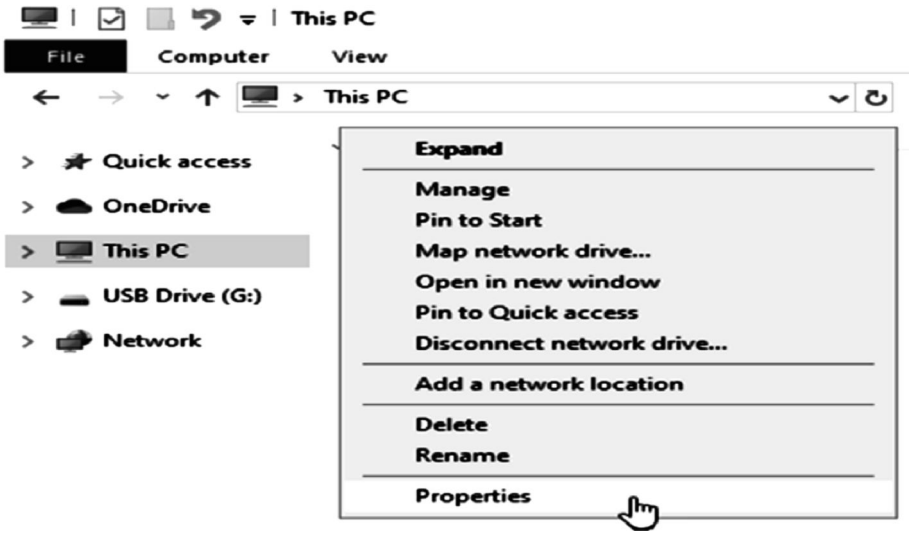
You must accept the Oracle Technology Network License Agreement for Oracle Java SE to download this software.

Accept License Agreement **Decline License Agreement**

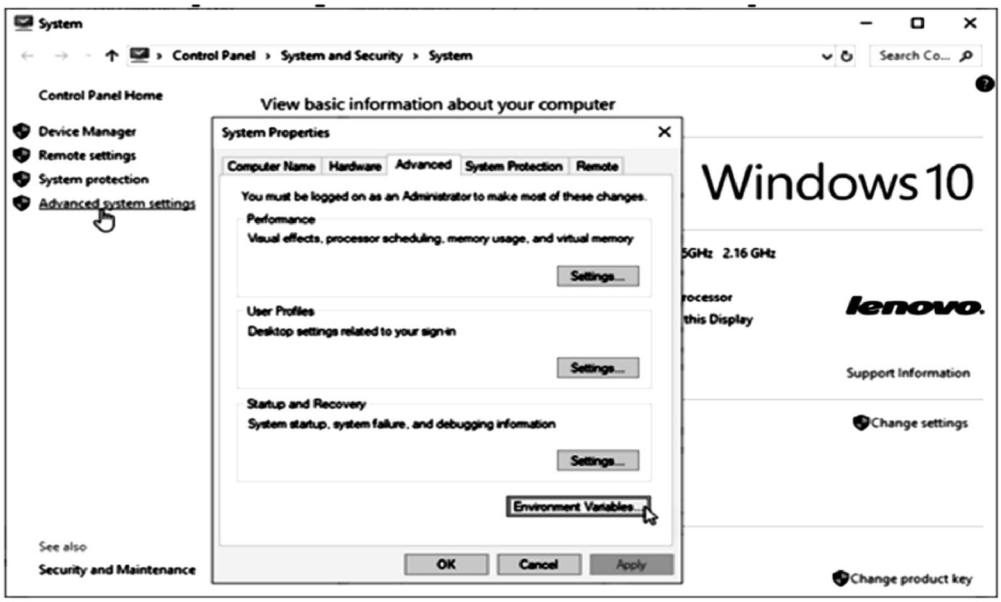
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	72.94 MB	jdk-8u241-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	69.83 MB	jdk-8u241-linux-arm64-vfp-hflt.tar.gz
Linux x86	171.28 MB	jdk-8u241-linux-i586.rpm
Linux x86	186.1 MB	jdk-8u241-linux-i586.tar.gz
Linux x64	170.65 MB	jdk-8u241-linux-x64.rpm
Linux x64	185.53 MB	jdk-8u241-linux-x64.tar.gz
Mac OS X x64	254.06 MB	jdk-8u241-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	133.01 MB	jdk-8u241-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	94.24 MB	jdk-8u241-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	133.8 MB	jdk-8u241-solaris-x64.tar.Z
Solaris x64	92.01 MB	jdk-8u241-solaris-x64.tar.gz
Windows x86	200.86 MB	jdk-8u241-windows-i586.exe
Windows x64	210.92 MB	jdk-8u241-windows-x64.exe

- Click Accept License Agreement
- Select jdk-8u241 according to the operating system used. The picture shows the JDK for Windows x64 selected.
- Perform Sign in by entering a username and password if you already have the Oracle account, or click Create Account to create a new one.

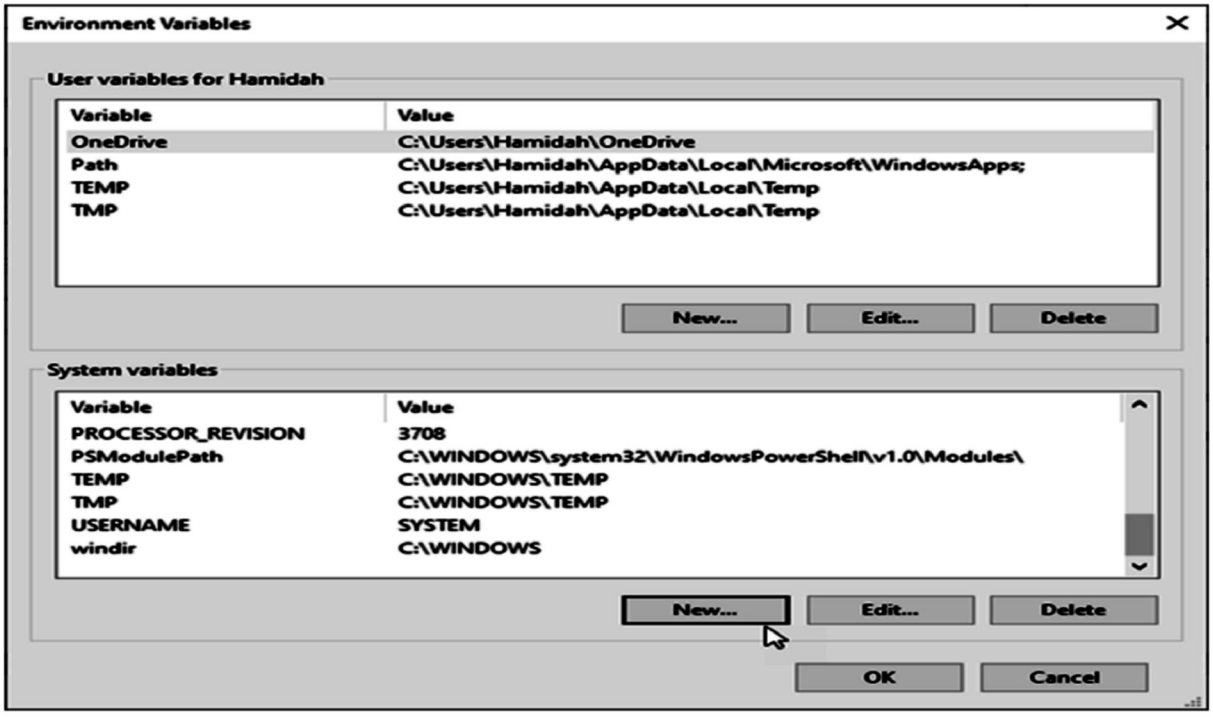
2. Double-click the downloaded jdk-8u241-XXXXX.exe file to install. Follow the instructions as-is.
3. Determine the JAVA_HOME variable in Windows.



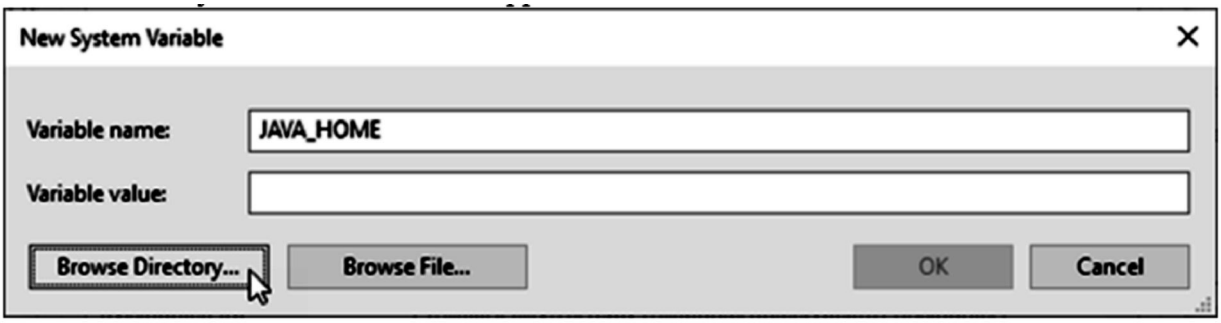
- In File Explorer, right-click This PC, then select Properties.



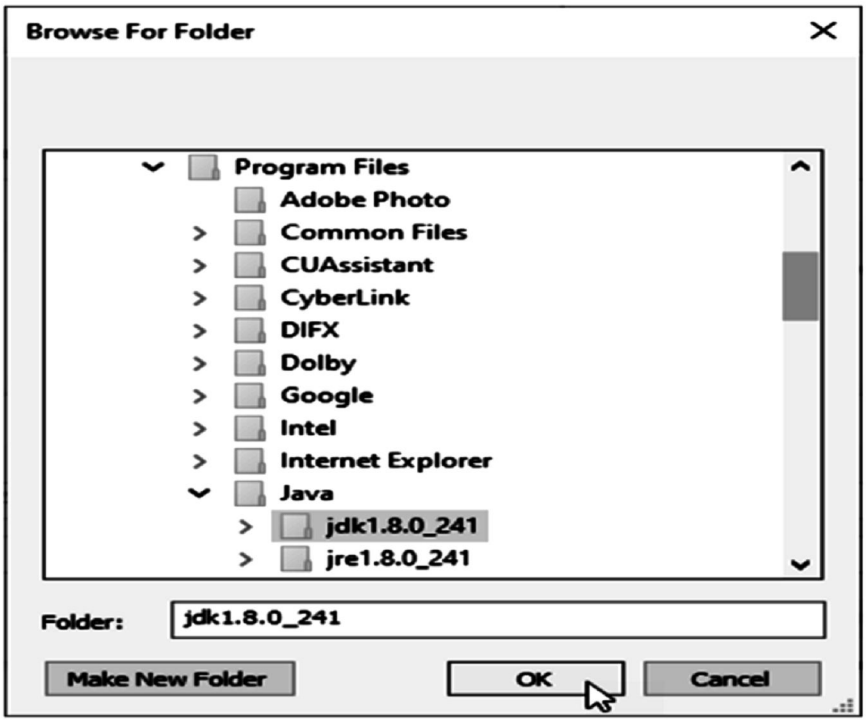
- On the Systems screen, click the Advanced system settings button.
- On the System Properties screen, click the Environment Variables button.
- The Environmental Variables screen appears.



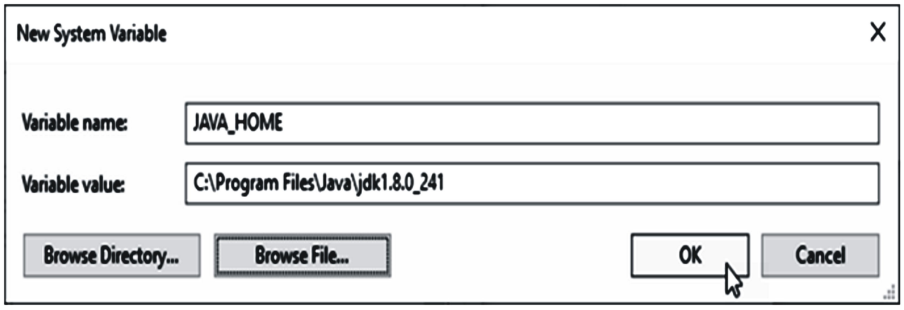
- Under System variables, click the New button.
- The New System Variable screen appears.



- In the Variable name text box, type JAVA_HOME.
- Click the Browse Directory… button.
- The Browse For Folder screen appears.



- Navigate to the C:\Program Files\Java\jdk1.8.0_241 folder.
- Click the OK button.



- The New System Variable screen in the Variable value text box already contains text following the previously selected folder.
- Then click the OK button.
- On the Environmental Variables screen, click the OK button.
- On the System Properties screen, click the OK button.

Python 2.7 Installation

1. Download the latest Python 2.7 or latest versions

(iv) Neo4j Installation

Windows installation

This section describes how to install Neo4j on Windows. Check System Requirements to see if your setup is suitable.

Windows console application

1. If it is not already installed, get OpenJDK 8 or Oracle Java 8.
2. Download the latest release from Neo4j Download Center.

   Select the appropriate ZIP distribution.
3. Make sure to download Neo4j from Neo4j Download Center and always check that the SHA hash of the downloaded file is correct:
   (a) To find the correct SHA hash, go to Neo4j Download Center and click on SHA-256 which will be located below your downloaded file.
   (b) Using the appropriate commands for your platform, display the SHA-256 hash for the file that you downloaded.
   (c) Ensure that the two are identical.
4. Right-click the downloaded file, click Extract All.
5. Place the extracted files in a permanent home on your server, for example D:\neo4j\. The top level directory is referred to as NEO4J_HOME.
   (a) To run Neo4j as a console application, use: <NEO4J_HOME>\bin\neo4j console.
   (b) To install Neo4j as a service use: <NEO4J_HOME>\bin\neo4j install-service.
   (c) For additional commands and to learn about the Windows PowerShell module included in the Zip file, see Windows PowerShell module.
6. Visit http://localhost:7474 in your web browser.
7. Connect using the username 'neo4j' with default password 'neo4j'. You'll then be prompted to change the password.
8. Stop the server by typing Ctrl-C in the console.

Windows service

Neo4j can also be run as a Windows service. Install the service with bin\neo4j install-service, and start it with bin\neo4j start.

The available commands for bin\neo4j are: help, start, stop, restart, status, install-service, uninstall-service, and update-service.

When installing a new release of Neo4j, you must first run bin\neo4j uninstall-service on any previously installed versions.

Java options

When Neo4j is installed as a service, Java options are stored in the service configuration. Changes to these options after the service is installed will not take effect until the service configuration is updated.

For example, changing the setting dbms.memory.heap.max_size in neo4j.conf will not take effect until the service is updated and restarted. To update the service, run bin\neo4j update-service. Then restart the service to run it with the new configuration.

The same applies to the path to where Java is installed on the system. If the path changes, for example when upgrading to a new version of Java, it is necessary to run the update-service command and restart the service. Then the new Java location will be used by the service.

Example 1. Update service example

1. Install service
   bin\neo4j install-service
2. Change memory configuration
3. echo dbms.memory.heap.initial_size=8g >> conf\neo4j.conf
   echo dbms.memory.heap.max_size=16g >> conf\neo4j.conf
4. Update service
   bin\neo4j update-service
5. Restart service
   bin\neo4j restart

Windows PowerShell module

The Neo4j PowerShell module allows administrators to:

- Install, start and stop Neo4j Windows® Services.
- Start tools, such as Neo4j Admin and Cypher Shell.
- The PowerShell module is installed as part of the ZIP file distributions of Neo4j.

System requirements

- Requires PowerShell v2.0 or above.
- Supported on either 32 or 64 bit operating systems.

Managing Neo4j on Windows

On Windows, it is sometimes necessary to Unblock a downloaded ZIP file before you can import its contents as a module. If you right-click on the ZIP file and choose "Properties" you will get a dialog which includes an "Unblock" button, which will enable you to import the module.

Running scripts has to be enabled on the system. This can, for example, be achieved by executing the following from an elevated PowerShell prompt.

2. Practice CRUD (Create, Read, Update, and Delete) operations on the four databases: Redis, MongoDB, Cassandra, Neo4j

Ans :

(i) Crud Operation using Redis

Step#1 : Create a Runner class to test all the methods

Now create a Runner class and test all methods that we defined in EmployeeDaoImpl.java class.

```
importjava.util.Map;
importorg.springframework.beans.factory.annotation.Autowired;
importorg.springframework.boot.CommandLineRunner;
importorg.springframework.stereotype.Component;
importcom.dev.springboot.redis.dao.IEmployeeDao;
importcom.dev.springboot.redis.model.Employee;
```

```
@Component
public classRedisOpertionsRunnerimplementsCommandLineRunner {
    @Autowired
    private IEmployeeDaoempDao;
    @Override
    public voidrun(String... args) throws Exception {
          //saving one employee
empDao.saveEmployee(newEmployee(500, "Emp0", 2150.0));
          //saving multiple employees
empDao.saveAllEmployees(
Map.of( 501, new Employee(501, "Emp1", 2396.0),
          502, newEmployee(502, "Emp2", 2499.5),
          503, newEmployee(503, "Emp4", 2324.75)
      )
);
       //modifying employee with empId 503
  empDao.updateEmployee(newEmployee(503, "Emp3", 2325.25));
       //deleting employee with empID 500
  empDao.deleteEmployee(500);
    //retrieving all employees
  empDao.getAllEmployees().forEach((k,v)->System.out.println(k +" : "+v));
    //retrieving employee with empID 501
System.out.println("Emp details for 501 : "+empDao.getOneEmployee(501));
  }
}
```

Step#2 : Start Redis Server

In order to test the implemented method, we need to start the Redis Server.

Step#3 : Run Spring Boot Project

Right Click on the project, select Run As -> Spring Boot Project.

Step#4 : Check the output

Below is the final output after including all operations. However, we can test individual methods separately by commenting other methods.

```
503 : Employee(empId=503, empName=Emp3, empSalary=2325.25)
501 : Employee(empId=501, empName=Emp1, empSalary=2396.0)
502 : Employee(empId=502, empName=Emp2, empSalary=2499.5)
Emp details for 501 : Employee(empId=501, empName=Emp1, empSalary=2396.0)
```

(ii) Crud Operation using MongoDB

Crud Operations

INSERT INDIVIDUAL DOCUMENTS

We'll create two new documents. One for information about Mickey Mouse and one for Charlie Brown .

```
        Our import additions
        import com.mongodb.ErrorCategory;
import com.mongodb.MongoWriteException;
import com.mongodb.client.MongoCollection;
import org.bson.Document;
ADDING DOCUMENTS
          //Insert a document into the "characters" collection.
MongoCollection collection = database.getCollection("characters");
          Document mickeyMouse = new Document();
          Document charlieBrown = new Document();
mickeyMouse.append("_id", 1)
.append("characterName", "Mickey Mouse")
.append("creator", new Document("firstName", "Walt").append("lastName", "Disney"))
.append("pet", "Goofy");
charlieBrown.append("_id", 2)
.append("characterName", "Charlie Brown")
.append("creator", new Document("firstName", "Charles").append("lastName", "Shultz"))
.append("pet", "Snoopy");
             try {
collection.insertOne(mickeyMouse);
collection.insertOne(charlieBrown);
System.out.println("Successfully inserted documents. \n");
          } catch (MongoWriteExceptionmwe) {
             if (mwe.getError().getCategory().equals(ErrorCategory.DUPLICATE_KEY)) {
System.out.println("Document with that id already exists");
            }
          }
```

We expect the following output:

Successful database connection established.

Successfully inserted documents.

We can go to the Mongo Shell by running mongo cartoon from a command prompt we will open a Mongo Shell and be using the cartoon database. We know that our data is in the characters collection so if we do a find() on our collection, and pretty print it with db.characters.find().pretty() we will get back our two documents.

```
{
      "_id" : 1,
      "characterName" : "Mickey Mouse",
      "creator" : {
            "firstName" : "Walt",
            "lastName" : "Disney"
      },
```

```
    "pet" : "Goofy"
}
{
    "_id" : 2,
    "characterName" : "Charlie Brown",
    "creator" : {
        "firstName" : "Charles",
        "lastName" : "Shultz"
    },
    "pet" : "Snoopy"
}
```

Maintenance and Multiple Inserts

This next step we're going to do a few things. Since we are inserting our documents with specific _id values, we can't run it again and try to again insert Mickey Mouse into our collection as we'll generate a Duplicate Key error. So, we'll do a bit of maintenance at the beginning of our code to delete the current characters collection each time we run our program and yet keep the data in the database at the end.

Let's get some output about our collection size and also generate multiple documents using the insertMany() method of the driver. We'll start with inserting documents starting at an _id of 3 (since we already have a 1 and 2) and generate enough so we have 50 documents in our collection.

Our import additions

```
import java.util.ArrayList;
import java.util.List;
```

Multiple inserts

```
        // Delete the collection and start fresh - add before the initial inserts
collection.drop();
        // Basic data on collection
System.out.println("Collection size: " + collection.count() + " documents. \n");
// Create and insert multiple documents
        List documents = new ArrayList();
        for (int i = 3; i< 51; i++) {
documents.add(new Document ("_id", i)
.append("characterName", "")
.append("creator", "")
.append("pet", "")
            );
        }
collection.insertMany(documents);
        // Basic data on collection
System.out.println("Collection size: " + collection.count() + " documents. \n");
```

Very nice. Now we get some output with data about our growing collection!

```
Successful database connection established.
```

Successfully inserted documents.

Collection size: 2 documents.

Collection size: 50 documents.

Updating Documents

We should probably how to update a document as well. After all, we have 50 documents in the collection but only two of them have any useful data. We use the updateOne() method to update a single document along with the $set operator. We'll print out the document with the _id of 3, update it, and print it out again to show the change. We can find our document to print using a eq Query Filter, or equals filter.

Our import additions

```
import com.mongodb.client.model.Filters;
```

Updates, find with filter

```
        // Update a document
        // print the third document before update.
        Document third = collection.find(Filters.eq("_id", 3)).first();
System.out.println(third.toJson());
collection.updateOne(new Document("_id", 3),
                new Document("$set", new Document("characterName", "Dilbert")
.append("creator", new Document("firstName", "Scott").append("lastName", "Adams"))
.append("pet", "Dogbert"))
        );
System.out.println("\nUpdated third document:");
        Document dilbert = collection.find(Filters.eq("_id", 3)).first();
System.out.println(dilbert.toJson());
```

Nice work! Our output now is:

```
Successful database connection established.
Successfully inserted documents.
Collection size: 2 documents.
Collection size: 50 documents.
Original third document:
{ "_id" : 3, "characterName" : "", "creator" : "", "pet" : "" }
Updated third document:
{ "_id" : 3, "characterName" : "Dilbert", "creator" : { "firstName" : "Scott", "lastName" : "Adams" }, "pet" :
"Dogbert" }
```

Whew! Almost there. Let's print out the entirety of our collection. We do this with a MongoCursor and iterate over all of the documents.

Our import additions

```
import com.mongodb.client.MongoCursor;
```

Find all with cursor

```
        // Find and print ALL documents in the collection
System.out.println("Print the documents.");
MongoCursor cursor = collection.find().iterator();
```

```
        try {
            while (cursor.hasNext()) {
System.out.println(cursor.next().toJson());
            }
        } finally {
cursor.close();
        }
```

With all of the new output, and most of it empty, perhaps we should clean up our collection a bit. We currently have 50 documents in there, but only three of them have any pertinent data. Let's get rid of all documents that don't have data using the deleteMany() function. We're going to need another filter here to delete documents whose _id is greater than or equal to 4. Fortunately there is a built in filter, gte that will do just that!

Our final code for this tutorial then is available here as a gist.

```
        //Delete data
System.out.println("\nDelete documents with an id greater than or equal to 4.");
collection.deleteMany(Filters.gte("_id", 4));
        // Find and print ALL documents in the collection
System.out.println("\nPrint all documents.");
MongoCursor cursor2 = collection.find().iterator();
        try {
            while (cursor2.hasNext()) {
System.out.println(cursor2.next().toJson());
            }
        } finally {
            cursor2.close();
        }
```

As we would expect, our collection is now left with only three documents, one each for Mickey Mouse, Charlie Brown, and Dilbert.

(iii) CRUD Operation using Cassandra

Cassandra CRUD Operation stands for Create, Update, Read and Delete or Drop. These operations are used to manipulate data in Cassandra. Apart from this, CRUD operations in Cassandra, a user can also verify the command or the data.

(a) Create Operation

A user can insert data into the table using Cassandra CRUD operation. The data is stored in the columns of a row in the table. Using INSERT command with proper what, a user can perform this operation.

A Syntax of Create Operation-

INSERT INTO <table name>

(<column1>,<column2>....)

VALUES(<value1>,<value2>...)

USING<option>

create a table data to illustrate the operation. Example consist of a table with information about students in college. The following table will give the details about the students.

Table.1 Cassandra Crud Operation – Create Operation

EN	NAME	BRANCH	PHONE	CITY
001	Ayush	Electrical Engineering	9999999999	Boston
002	Aarav	Computer Engineering	8888888888	New York City
003	Kabir	Applied Physics	7777777777	Philadelphia

EXAMPLE 1:

Creating a table and inserting the data into a table:

INPUT:

cqlsh:keyspace1> INSERT INTO student(en, name, branch, phone, city)

VALUES(001, 'Ayush', 'Electrical Engineering', 9999999999, 'Boston');

cqlsh:keyspace1> INSERT INTO student(en, name, branch, phone, city)

VALUES(002, 'Aarav', 'Computer Engineering', 8888888888, 'New York City');

cqlsh:keyspace1> INSERT INTO student(en, name, branch, phone, city)

VALUES(003, 'Kabir', 'Applied Physics', 7777777777, 'Philadelphia');

Table.2 Cassandra Crud Operation – OUTPUT After Verification (READ operation)

EN	NAME	BRANCH	PHONE	CITY
001	Ayush	Electrical Engineering	9999999999	Boston
002	Aarav	Computer Engineering	8888888888	New York City
003	Kabir	Applied Physics	7777777777	Philadelphia

(b) Update Operation

The second operation in the Cassandra CRUD operation is the UPDATE operation. A user can use UPDATE command for the operation. This operation uses three keywords while updating the table.

Where: This keyword will specify the location where data is to be updated.

Set: This keyword will specify the updated value.

Must: This keyword includes the columns composing the primary key.

A Syntax of Update Operation-

UPDATE <table name>

SET <column name> = <new value>

<column name> = <value>...

WHERE <condition>

EXAMPLE 2:

Let's change few details in the table 'student'. In this example, we will update Aarav's city from 'New York City' to 'San Fransisco'.

INPUT:

cqlsh:keyspace1> UPDATE student SET city='San Fransisco'WHERE en=002;

Table.3 Cassandra Crud Operation – OUTPUT After Verification

EN	NAME	BRANCH	PHONE	CITY
001	Ayush	Electrical Engineering	9999999999	Boston
002	Aarav	Computer Engineering	8888888888	San Fransisco
003	Kabir	Applied Physics	7777777777	Philadelphia

(c) Read Operation

This is the third Cassandra CRUD Operation – Read Operation. A user has a choice to read either the whole table or a single column. To read data from a table, a user can use SELECT clause. This command is also used for verifying the table after every operation.

SYNTAX to read the whole table-

SELECT * FROM <table name>;

EXAMPLE 3:

To read the whole table 'student'.

INPUT:

cqlsh:keyspace1> SELECT * FROM student;

Table.4 Cassandra Crud Operation – OUTPUT After Verification

EN	NAME	BRANCH	PHONE CITY	
001	Ayush	Electrical Engineering	9999999999	Boston
002	Aarav	Computer Engineering	8888888888	San Fransisco
003	Kabir	Applied Physics	7777777777	Philadelphia

SYNTAX to read selected columns-

EXAMPLE 4:

To read columns of name and city from table 'student'.

INPUT:

cqlsh:keyspace1> SELECT name, city FROM student;

Table.5 Cassandra Crud Operation – OUTPUT After Verification

NAME	CITY
Ayush	Boston
Aarav	San Fransisco
Kabir	Philadelphia

(d) Delete Operation

Delete operation is the last Cassandra CRUD Operation, allows a user to delete data from a table. The user can use DELETE command for this operation.

A Syntax of Delete Operation-

DELETE <identifier> FROM <table name> WHERE <condition>;

EXAMPLE 5:

In the 'student' table let us delete the 'phone' or phone number from 003 row.

cqlsh:keyspace1> DELETE phone FROM student WHERE en=003;

Table.6 Cassandra Crud Operation – OUTPUT After Verification

EN	NAME	BRANCH	PHONE	CITY
001	Ayush	Electrical Engineering	9999999999	Boston
002	Aarav	Computer Engineering	8888888888	San Fransisco
003	Kabir	Applied Physics	null	Philadelphia

SYNTAX for deleting the entire row-

DELETE FROM <identifier> WHERE <condition>;

EXAMPLE 6:

In the 'student' table, let us delete the entire third row.

cqlsh:keyspace1> DELETE FROM student WHERE en=003;

Table.7 Cassandra Crud Operation – OUTPUT After Verification

EN	NAME	BRANCH	PHONE	CITY
001	Ayush	Electrical Engineering	9999999999	Boston
002	Aarav	Computer Engineering	8888888888	San Fransisco

So, this was all about Cassandra CRUD operations.

(iv) CRUD Operation using Neo4j

1. Introduction

CRUD operations in Neo4j.We are going to use SET, Remove, Delete and Merge(Create and MATCH) clause.

2. Merge Command

Merge commend is used to either create a node with labels or properties if the node does not exist in the database or select the nodes with particular conditions if the nodes are already presents in the database.

Using Merge command with labels and properties.

Example

Bash

$ Merge(Jeson:Man)returnJeson, labels(Jeson)

Since it does not exists in the database, it will create a new node and a new label Man and return the node and labels of that specific node.

Bash

$ Merge(Jack:Man{Name:"Jack",Gender:"male"})return Jack

When running this merge command, it will firstly look at if there exists a node jack with corresponding key-value pairs, if it does not exits, it will create a new node and return the node.

Bash

$ Merge(Jack:Man{Name:"Jack",Age:"30"})return Jack

It will seperately create a node with the same name as before but differnt properties.

Then, if we run the following comand

Bash

$ Merge(Man{Name:"Jack"})returnMan.Name, Man.Gender, Man.Age

It will find all the nodes with name Jack and label Man, and the properties related to jack

Using Merge command to create Node with existing node properties.

Bash

MATCH (n) WHERE EXISTS(n.Gender)

Merge (gender:Gender{Gender:n.Gender})

return gender

We could see that we have created all gender with matching all the nodes with Gender properties.

Merge command with on create clause.

When creating a new node, we would like to set the properties for this node at the same time, ON CREATE clause will allows the user to set properties at creation time.

Bash

Merge (Yuki:Person{Name:"Yuki"})

ON CREATE SET Yuki.Gender="Female"

RETURN Yuki.Name,Yuki.Gender

Merge command with on match clause.

When traversing from the graph database to find some nodes with some conditions, we need to set some properties to those node at the same time for updating the information regarding to this node.

Bash

Merge (person:Person)

ON MATCH SET person.Is_Person= TRUE

RETURN person.Name, person.Is_Person

Using merge command we could find all the node with label Person and add a new property Is_Person to be true

Merge command for relationships

Bash

Merge (mika:Person{Name:"mika"})

ON CREATE SET mika.Gender="Male"

RETURN mika.Name,mika.Gender

Suppose Yuki has a boyfriend mika. we want to create relationships between them.We could run the following commands

Bash

```
MATCH (girl:Person),(boy:Person)
WHERE girl.Name="Yuki" AND boy.Name="mika"
MERGE (girl)-[relationship:Has_A_Boy_Friend]->(boy)
RETURN girl,boy
```

Smiley face

3. SET Command

SET Clause is to set and remove properties of nodes. If Yuki goes to Tokyo University, we could run the following command. We could use set clause to add school properties to Yuki, and remove the Is_Person property we have set before.

Bash

```
MATCH (Yuki:Person)
WHERE Yuki.Name="Yuki"
SET Yuki.School="Tokyo University"
SET Yuki.Is_Person= NULL
RETURN Yuki
```

We could also use SET Clause to set multiple labels for some nodes,e.g,add two labels to Yuki which are Student and Gril.

Bash

```
MATCH (Yuki:Person)
WHERE Yuki.Name="Yuki"
SET Yuki:Student:Gril
RETURN Yuki
```

4. DELETE Command

DELETE command is to delete specific nodes or relationships between nodes. Here we will learn some of the basic usage of delete clause in CQL.

DELETE ALL NODES

Bash

```
MATCH (node)
DETACH DELETE node
```

DETACH is to firstly delete the relationships between the nodes that we are going to delete.

DELETE GROUPS OF NODES WITH CONDITIONS

It is quite similar but still notice that when deleting a node, we should delete the relationships first.

Bash

```
MATCH (mika:Person)
WHERE mika.Name="mika"
DETACH DELETE mika
```

3. Usage of Where Clause equivalent in MongoDB

Ans :

Where Clause in MongoDB

To query the document on the basis of some condition, you can use following operations.

Operation	Syntax	Example
Equality	{<key>:{$eg;<value>}}	db.mycol.find({"by":"tutorials point"}).pretty()
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()
Values in an array	{<key>:{$in:[<value1>, <value2>,……<valueN>]}}	db.mycol.find({"name":{$in:["Raj", "Ram", db.mycol.find({"name":{$in:["Raj", "Ram", "Raghu"]}}).pretty()
Values not in an array	{<key>:{$nin:<value>}}	db.mycol.find({"name":{$nin:["Ramu", "Raghav"]}}).pretty()

Example

Consider the following documents in the players collection:

```
db.players.insertMany([
{ _id: 12378, name: "Steve", username: "KumarNaveen", first_login: "2017-01-01" },
{ _id: 2, name: "adi", username: "sahasra", first_login: "2001-02-02" }
])
```

The following example uses $where and the hex_md5() JavaScript function to compare the value of the name field to an MD5 hash and returns any matching document.

```
db.players.find( { $where: function() {
return (hex_md5(this.name) == "9b53e667f30cd329dca1ec9e6a83e994")
} } );
```

The operation returns the following result:

```
{
"_id" : 2,
"name" :"adi",
"username" :"sahasra",
"first_login" :"2001-02-02"
}
```

4. Usage of operations in MongoDB – AND in MongoDB, OR in MongoDB, Limit Records and Sort Records. Usage of operations in MongoDB – Indexing, Aggregation and Map Reduce.

Ans :

➢ **AND in MongoDB**

Syntax

To query documents based on the AND condition, you need to use $and keyword. Following is the basic syntax of AND -

```
>db.mycol.find({ $and: [ {<key1>:<value1>}, { <key2>:<value2>} ] })
```

Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```
>db.mycol.find({$and:[{"by":"tutorials point"},{"title":"MongoDB Overview"}]}).pretty()
{
   "_id":ObjectId("5dd4e2cc0821d3b44607534c"),
   "title":"MongoDB Overview",
   "description":"MongoDB is no SQL database",
   "by":"tutorials point",
   "url":"http://www.rahulpublication.com",
   "tags":[
      "mongodb",
      "database",
      "NoSQL"
   ],
   "likes":100
}
>
```

For the above given example, equivalent where clause will be ' where by = 'tutorials point' AND title = 'MongoDB Overview' '. You can pass any number of key, value pairs in find clause.

➢ **OR in MongoDB**

Syntax

To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR -

```
>db.mycol.find(
   {
      $or: [
```

```
        {key1: value1}, {key2:value2}
      ]
   }
).pretty()
```

Example

Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

```
>db.mycol.find({$or:[{"by":"tutorials point"},{"title":"MongoDB Overview"}]}).pretty()
{
"_id":ObjectId(7df78ad8902c),
"title":"MongoDB Overview",
"description":"MongoDB is no sql database",
"by":"tutorials point",
"url":"http://www.rahulpublication.com",
"tags":["mongodb","database","NoSQL"],
"likes":"100"
}
>
```

- **Limit Records and Sort Records.**

The Limit() Method

To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.

Syntax

The basic syntax of limit() method is as follows -

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

Example

Consider the collection myycol has the following data.

```
{_id :ObjectId("507f191e810c19729de860e1"), title:"MongoDB Overview"},
{_id :ObjectId("507f191e810c19729de860e2"), title:"NoSQL Overview"},
{_id :ObjectId("507f191e810c19729de860e3"), title:"Tutorials Point Overview"}
```

Following example will display only two documents while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

If you don't specify the number argument in limit() method then it will display all documents from the collection.

The sort() Method

To sort documents in MongoDB, you need to use sort() method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax

The basic syntax of sort() method is as follows -

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

Example

Consider the collection myycol has the following data.

```
{_id :ObjectId("507f191e810c19729de860e1"), title:"MongoDB Overview"}
{_id :ObjectId("507f191e810c19729de860e2"), title:"NoSQL Overview"}
{_id :ObjectId("507f191e810c19729de860e3"), title:"Tutorials Point Overview"}
```

Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Rahul Publisher Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

Note:

if you don't specify the sorting preference, then sort() method will display the documents in ascending order.

- **Usage of operations in MongoDB – Indexing**

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require MongoDB to process a large volume of data.

Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

The createIndex() Method

To create an index, you need to use createIndex() method of MongoDB.

Syntax

The basic syntax of createIndex() method is as follows().

```
>db.COLLECTION_NAME.createIndex({KEY:1})
```

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

Example

```
>db.mycol.createIndex({"title":1})
```

```
{
    "createdCollectionAutomatically":false,
    "numIndexesBefore":1,
    "numIndexesAfter":2,
    "ok":1
}
>
```

In createIndex() method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.createIndex({"title":1,"description":-1})
>
```

Aggregation:

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count(*) and with group by is an equivalent of MongoDB aggregation.

The aggregate() Method

For the aggregation in MongoDB, you should use aggregate() method.

Syntax

Basic syntax of aggregate() method is as follows -

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example

In the collection you have the following data -

```
{
    _id:ObjectId(7df78ad8902c)
    title:'MongoDB Overview',
    description:'MongoDB is no sql database',
by_user:‘navin',
    url:'http://www.navin@123.com',
    tags:['mongodb','database','NoSQL'],
    likes:100
},
{
    _id:ObjectId(7df78ad8902d)
title:'NoSQL Overview',
    description:'No sql database is very fast',
    by_user:'tutorials point',
```

```
        url:'http://www.navin@123.com',
    tags:['mongodb','database','NoSQL'],
        likes:10
    },
    {
    _id:ObjectId(7df78ad8902e)
    title:'Neo4j Overview',
    description:'Neo4j is no sql database',
    by_user:'Neo4j',
        url:'http://www.neo4j.com',
        tags:['neo4j','database','NoSQL'],
    likes:750
    },
```

Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method -

```
>db.mycol.aggregate([{$group :{_id :"$by_user",num_tutorial:{$sum :1}}}])
{"_id":"rahulpublication","num_tutorial":2}
{"_id":"Neo4j","num_tutorial":1}
>
```

MapReduce:

Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations. MapReduce is generally used for processing large data sets.

MapReduce Command

Following is the syntax of the basic mapReduce command -

```
>db.collection.mapReduce(
function(){emit(key,value);},//map function
function(key,values){returnreduceFunction},{//reduce function
out: collection,
    query: document,
    sort: document,
    limit: number
}
)
```

The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

In the above syntax -

- map is a javascript function that maps a value with a key and emits a key-value pair
- reduce is a javascript function that reduces or groups all the documents having the same key
- out specifies the location of the map-reduce query result
- query specifies the optional selection criteria for selecting documents
- sort specifies the optional sort criteria
- limit specifies the optional maximum number of documents to be returned

Using MapReduce

Consider the following document structure storing user posts. The document stores user_name of the user and the status of post.

```
{
"post_text":"rahulpublication is an awesome website for tutorials",
"user_name":"mark",
"status":"active"
}
```

Now, we will use a mapReduce function on our posts collection to select all the active posts, group them on the basis of user_name and then count the number of posts by each user using the following code -

```
>db.posts.mapReduce(
function(){ emit(this.user_id,1);},
function(key, values){returnArray.sum(values)},{
query:{status:"active"},
out:"post_total"
}
)
```

The above mapReduce query outputs the following result -

```
{
    "result" : "post_total",
    "timeMillis" : 9,
    "counts" : {
        "input" : 4,
        "emit" : 4,
        "reduce" : 2,
        "output" : 2
},
"ok" : 1,
}
```

The result shows that a total of 4 documents matched the query (status:"active"), the map function emitted 4 documents with key-value pairs and finally the reduce function grouped mapped documents having the same keys into 2.

To see the result of this mapReduce query, use the find operator -

```
>db.posts.mapReduce(
function(){ emit(this.user_id,1);},
function(key, values){returnArray.sum(values)},{
query:{status:"active"},
out:"post_total"
}
).find()
```

The above query gives the following result which indicates that both users tom and mark have two posts in active states -

```
{"_id":"tom","value":2}
{"_id":"mark","value":2}
```

In a similar manner, MapReduce queries can be used to construct large complex aggregation queries. The use of custom Javascript functions make use of MapReduce which is very flexible and powerful.

5. Write a program to count the number of occurrences of a word using MapReduce.

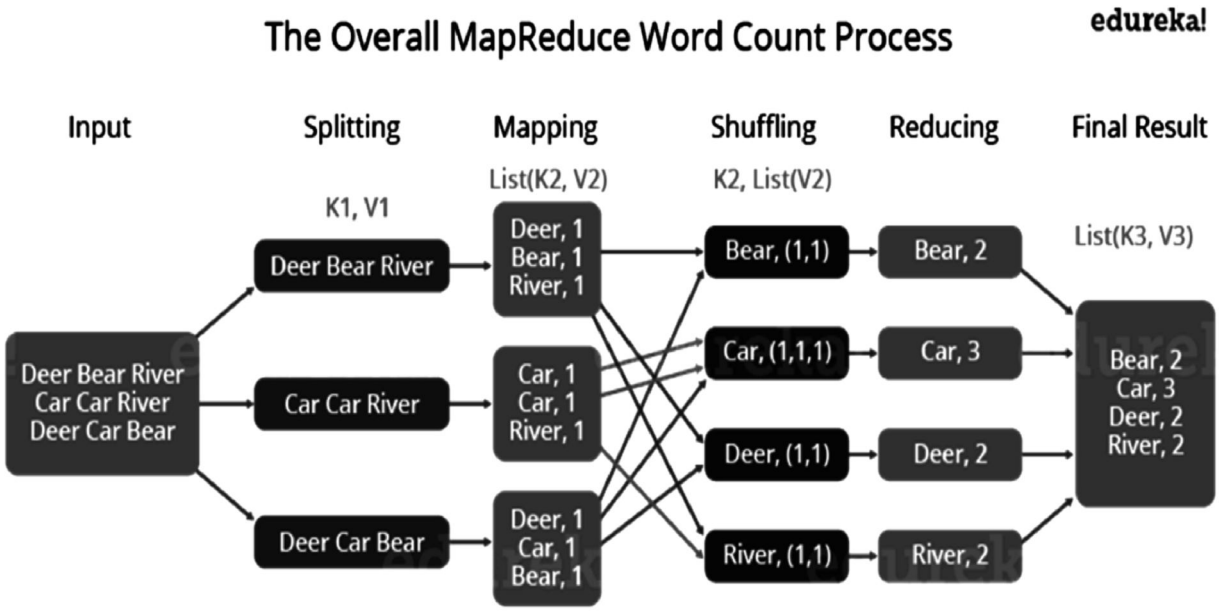
Ans :

MapReduce: A Word Count Example of MapReduce

Let us understand, how a MapReduce works by taking an example where I have a text file called example.txt whose contents are as follows:

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

Now, suppose, we have to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



- First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.
- Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Dear, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.
- So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- Finally, all the output key/value pairs are then collected and written in the output file.

Explanation of MapReduce Program

The entire MapReduce program can be fundamentally divided into three parts:

- Mapper Phase Code
- Reducer Phase Code
- Driver Code

We will understand the code for each of these three parts sequentially.

Mapper code:

```
publicstaticclassMap extendsMapper&lt;LongWritable,Text,Text,IntWritable&gt; {
    publicvoidmap(LongWritable key, Text value, Context context)
        throwsIOException,InterruptedException {
            String line = value.toString();
    StringTokenizer tokenizer = newStringTokenizer(line);
     while(tokenizer.hasMoreTokens()) {
          value.set(tokenizer.nextToken());
          context.write(value, newIntWritable(1));
     }
```

- We have created a class Map that extends the class Mapper which is already defined in
- the MapReduce Framework.
- We define the data types of input and output key/value pair after the class declaration using angle brackets.
- Both the input and output of the Mapper is a key/value pair.
- Input:

- The key is nothing but the offset of each line in the text file: LongWritable
- The value is each individual line (as shown in the figure at the right): Text
- Output:
- The key is the tokenized words: Text
- We have the hardcoded value in our case which is 1: IntWritable
- Example – Dear 1, Bear 1, etc.
- We have written a java code where we have tokenized each word and assigned them a hardcoded value equal to 1.

Reducer Code:

```
publicstaticclassReduce extendsReducer&lt;Text,IntWritable,Text,IntWritable&gt; {
    publicvoidreduce(Text key, Iterable&lt;IntWritable&gt; values,Context context)
            throwsIOException,InterruptedException {
                    intsum=0;
        for(IntWritable x: values)
                    {
                sum+=x.get();
                    }
        context.write(key, newIntWritable(sum));
                    }
                }
```

- We have created a class Reduce which extends class Reducer like that of Mapper.
- We define the data types of input and output key/value pair after the class declaration using angle brackets as done for Mapper.
- Both the input and the output of the Reducer is a key-value pair.

Input:

- The key nothing but those unique words which have been generated after the sorting and shuffling phase: Text
- The value is a list of integers corresponding to each key: IntWritable
- Example – Bear, [1, 1], etc.

Output:

- The key is all the unique words present in the input text file: Text
- The value is the number of occurrences of each of the unique words: IntWritable
- Example – Bear, 2; Car, 3, etc.
- We have aggregated the values present in each of the list corresponding to each key and produced the final answer.
- In general, a single reducer is created for each of the unique words, but, you can specify the number of reducer in mapred-site.xml.

Driver Code:

```
Configuration conf= newConfiguration();
Job job = newJob(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = newPath(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, newPath(args[0]));
FileOutputFormat.setOutputPath(job, newPath(args[1]));
```

- In the driver class, we set the configuration of our MapReduce job to run in Hadoop.
- We specify the name of the job, the data type of input/output of the mapper and reducer.
- We also specify the names of the mapper and reducer classes.
- The path of the input and output folder is also specified.
- The method setInputFormatClass () is used for specifying how a Mapper will read the input data or what will be the unit of work. Here, we have chosen TextInputFormat so that a single line is read by the mapper at a time from the input text file.
- The main () method is the entry point for the driver. In this method, we instantiate a new Configuration object for the job.

FACULTY OF SCIENCE

B.Sc. III Year V Semester (CBCS) Examination

Model Paper - I

NO SQL DATA BASES

PAPER - V(B) : (DATA SCIENCE)

Time: 3 Hours | Max. Marks: 80

SECTION – A (8 × 4 = 32 Marks)

[Short Answer Type]

Note: Answer any **EIGHT** questions. All questions carry equal marks.

1. What is a NoSQL data model? How is data stored in NoSQL? **(Unit-I, SQA-1)**
2. What are the Advantages and Disadvantages of Graph Data Model. **(Unit-I, SQA-3)**
3. What are the Features of NoSQL Databases? **(Unit-I, SQA-7)**
4. Explain about single server in distribution models? **(Unit-II, SQA-1)**
5. Explain Master-Slave Replication in distribution models? **(Unit-II, SQA-3)**
6. What are the Importance of ACID Transactions? **(Unit-II, SQA-5)**
7. What are the NoSQL use cases. **(Unit-III, SQA-3)**
8. What is Document Database? **(Unit-III, SQA-6)**
9. Write the Differences Between SQL And NoSQL. **(Unit-III, SQA-8)**
10. What Is a Column-Family Data Store? **(Unit-IV, SQA-1)**
11. What are the Advantages and Disadvantages of Graph Database? **(Unit-IV, SQA-5)**
12. What are differences between graph and relational databases? **(Unit-IV, SQA-9)**

SECTION – B (4 × 12 = 48 Marks)

[Essay Answer Type]

Note: Attempt **ALL** questions. All questions carry equal marks.

13. (a) What is a relational database? And Explain the importance of it. **(Unit-I, Q.No. 1)**

(OR)

(b) Explain Graph Based Databases in NoSQL with its applications. **(Unit-I, Q.No. 12)**

14. (a) What is Sharding in distribution models? **(Unit-II, Q.No. 2)**

(OR)

(b) Explain Composing Map-Reduce Calculation with an example? **(Unit-II, Q.No. 15)**

15. (a) What is a Key-Value Store? Explain the popular Key-Value Store databases. **(Unit-III, Q.No. 2)**

(OR)

(b) What are the features of Document Data Base (Data Model)? **(Unit-III, Q.No. 9)**

16. (a) Explain in detail about Column-Family Data Store? **(Unit-IV, Q.No. 1)**

(OR)

(b) What are the features of Graph Databases? **(Unit-IV, Q.No. 7)**

FACULTY OF SCIENCE

B.Sc. III Year V Semester (CBCS) Examination

Model Paper - II

NO SQL DATA BASES

PAPER - V(B) : (DATA SCIENCE)

Time: 3 Hours Max. Marks: 80

SECTION – A (8 × 4 = 32 Marks)

[Short Answer Type]

Note: Answer any **EIGHT** questions. All questions carry equal marks.

1. What is Column-Family Stores? **(Unit-I, SQA-8)**
2. What are Applications of NoSQL Databases? **(Unit-I, SQA-6)**
3. What is Graph Based Data Model in NoSQL? **(Unit-I, SQA-9)**
4. What is Basic Map-Reduce? **(Unit-II, SQA-7)**
5. What are the phases of data flow in MapReduce? **(Unit-II, SQA-8)**
6. What are the Advantages and Disadvantages of No SQL? **(Unit-II, SQA-2)**
7. Compare the Advantages and Disadvantages of Document Database. **(Unit-III, SQA-4)**
8. What are the challenges of Document database? **(Unit-III, SQA-5)**
9. What does "Document-oriented" vs. Key-Value mean when talking about MongoDB vs Cassandra? **(Unit-III, SQA-11)**
10. How Graph and Graph Databases Work? Give the examples of Graph Databases. **(Unit-IV, SQA-4)**
11. What are key benefits of Column Store Databases? **(Unit-IV, SQA-7)**
12. What are the graph database use cases with an examples? **(Unit-IV, SQA-10)**

SECTION – B (4 × 12 = 48 Marks)

[Essay Answer Type]

Note: Attempt **ALL** questions. All questions carry equal marks.

13. (a) Explain about the Applications and Integration of Databases? **(Unit-I, Q.No. 5)**

(OR)

(b) What is Aggregate-Oriented Databases in NoSQL? **(Unit-I, Q.No. 10)**

14. (a) Explain detail about single server in distribution models? **(Unit-II, Q.No. 1)**

(OR)

(b) What is quorum? Explore in detail. **(Unit-II, Q.No. 10)**

15. (a) Explain about Features of Key-Value Store. **(Unit-III, Q.No. 3)**

(OR)

(b) Define Document Database? How document databases differ from relational databases? **(Unit-III, Q.No. 8)**

16. (a) What are the Features of Column Store Databases. **(Unit-IV, Q.No. 3)**

(OR)

(b) What are the features of Graph Databases? **(Unit-IV, Q.No. 7)**

FACULTY OF SCIENCE

B.Sc. III Year V Semester (CBCS) Examination

Model Paper - III

NO SQL DATA BASES

PAPER - V(B) : (DATA SCIENCE)

Time: 3 Hours Max. Marks: 80

SECTION – A (8 × 4 = 32 Marks)

[Short Answer Type]

Note: Answer any **EIGHT** questions. All questions carry equal marks.

1. What is a relational database? **(Unit-I, SQA-5)**
2. What are the Applications of Graph Data Model? **(Unit-I, SQA-4)**
3. What are the Advantages and Disadvantages of Graph Data Model? **(Unit-I, SQA-3)**
4. What is consistency? **(Unit-II, SQA-4)**
5. What are the Learning Goals of Relaxing Consistency? **(Unit-II, SQA-6)**
6. What are the key differences between Pig vs MapReduce? **(Unit-II, SQA-10)**
7. What is a Key-Value Store? **(Unit-III, SQA-1)**
8. What are the advantages and disadvantages of key-value store. **(Unit-III, SQA-2)**
9. When to avoid NoSQL **(Unit-III, SQA-10)**
10. What Is a Graph Database? **(Unit-IV, SQA-3)**
11. What are the types of Graph Databases? **(Unit-IV, SQA-6)**
12. What are key features of Column Store Databases. **(Unit-IV, SQA-8)**

SECTION – B (4 × 12 = 48 Marks)

[Essay Answer Type]

Note: Attempt **ALL** questions. All questions carry equal marks.

13. (a) What is Aggregate Data Model? Give with an example. **(Unit-I, Q.No. 8)**

(OR)

(b) What is Column-Family Stores? Explain in detail? **(Unit-I, Q.No. 9)**

14. (a) What is relaxing durability in NoSQL? **(Unit-II, Q.No. 8)**

(OR)

(b) Explain Business and system Transactions with an example? **(Unit-II, Q.No. 11)**

15. (a) What is a Key-Value (Model)Database? Explain with advantages and disadvantages. **(Unit-III, Q.No. 1)**

(OR)

(b) Explain about Document Database in NoSQL. **(Unit-III, Q.No. 7)**

16. (a) What Is a Graph Database? Explain in detail. **(Unit-IV, Q.No. 6)**

(OR)

(b) What are the Use Cases of Graph Databases? Explain its advantages and disadvantages? **(Unit-IV, Q.No. 9)**