MODULE 3

NoSQL Big Data Management, MongoDB and Cassandra

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

LO 3.1 Get conceptual understanding of NoSQL data stores, Big Data solutions, schema-less models, and increased flexibility for data manipulation

LO 3.2 Get knowledge of NoSQL data architecture patterns namely, key-value pairs, tabular, column family, BigTable, Record Columnar (RC), Optimized Row Columnar (ORC) and Parquet, document, object and graph data stores, and the variations in architectural patterns

LO 3.3 Get conceptual understanding of NoSQL data store management, applications and handling problems in Big Data

LO 3.4 Solve Big Data analytics using shared-nothing architecture, choosing a distribution model among master-slave and peer-to-peer models, and get the knowledge of four ways by which the NoSQL handles the Big Data problems

LO 3.5 Apply the MongoDB databases and query commands

LO 3.6 Use the Cassandra databases, data model, clients, and integrate them with Hadoop

RECALL FROM PREVIOUS CHAPTERS

Big Data use new tools for processing and analysis of large volume of data. Big Data sources are Hadoop or Spark compatible file system, structured, unstructured or NoSQL data Store (Table 1.1). Big Data distributed computing uses shared-nothing paradigm, *no in-between data sharing and inter-processor communication.* (Table 1.2)

Chapter 1 introduced NoSQL. NoSQL data stores can store semi-structured or unstructured data. NoSQL stands for No-SQL or Not Only SQL. NoSQL databases can coexists with SQL databases. NoSQL data applications do not integrate with SQL databases applications. NoSQL databases store Big Data. Examples of NoSQL data stores are key-value pairs, hash key, *)SON* files, BigTable, HBase, MongoDB,Cassandra, and CouchDB(Section 1.6.2.1).

This chapter focuses on providing detailed concepts of NoSQL data architectural patterns, management of Big Data, data distribution models, handling of Big Data problems using NoSQL, MongoDB for document and Cassandra for columnar stores.

3.1 ! INTRODUCTION

Big Data uses distributed systems. A distributed system consists of multiple data nodes at clusters of machines and distributed software components. The tasks execute in parallel with data at nodes in clusters. The computing nodes communicate with the applications through a network.

Following are the features of distributed-computing architecture (Chapter 2):

1. *Increased reliability and fault tolerance:* The important advantage of distributed computing system is reliability. If a segment of machines in a cluster fails then the rest of the machines continue work. When the datasets replicate at number of data nodes, the fault tolerance increases further. The dataset in remaining segments continue the same computations as being done at failed segment machines.

2. *Flexibility* makes it very easy to install, implement and debug new services in a distributed environment.
3. *Sharding* is storing the different parts of data onto different sets of data nodes, clusters or servers. For example, university students huge database, on sharding divides in databases, called shards. Each shard may correspond to a database for an individual course and year. Each shard stores at different nodes or servers.
4. *Speed:* Computing power increases in a distributed computing system as shards run parallelly on individual data nodes in clusters independently (no data sharing between shards).
5. *Scalability:* Consider sharding of a large database into a number of shards, distributed for computing in different systems. When the database expands further, then adding more machines and increasing the number of shards provides horizontal scalability. Increased computing power and running number of algorithms on the same machines provides vertical scalability (Section 1.3.1).
6. *Resources sharing:* Shared resources of memory, machines and network architecture reduce the cost.
7. *Open system* makes the service accessible to all nodes.
8. *Performance:* The collection of processors in the system provides higher performance than a centralized computer, due to lesser cost of communication among machines (Cost means time taken up in communication).

The demerits of distributed computing are: (i) issues in troubleshooting in a larger networking infrastructure, (ii) additional software requirements and (iii) security risks for data and resources.

Big Data solutions require a scalable distributed computing model with shared-nothing architecture. A solution is Big Data store in HDFS files. NoSQL data also store Big Data, and facilitate random read/write accesses. The accesses are sequential in HDFS data.

HBase is a NoSQL solution (Section 2.6.3). Examples of other solutions are

MongoDB and Cassandra. MongoDB and Cassandra DBMSs create HDFS compatible distributed data stores and include their specific query processing languages.

Following are selected key terms used in database systems.

Class refers to a template of program codes that is extendable. Class creates instances, called objects. A class consists of initial values for member fields, called *state* (of variables), and implementations of member functions and methods called behavior. An implementation means program codes along with values of arguments in the functions and methods (Java Class uses methods, C++ functions.) An abstract class consists of at least one abstract member or method.

Object is an instance of a class in Java, C++, and other object-oriented languages. Object can be an instance of another object (for example, in JavaScript).

Tupple is an ordered set of data which constitutes a record. For example, one row record in a table. A row in a relational database has column fields or attributes. Example of a tupple is (JLRWSale,Week 1, 138, Week 2, 232, ..., week 52, 186) in an RDBMS table. Here, JLRWSalemeans Jaguar Land Rover Weekly Sale. (JLRWSale,Week 1, 138) is also a tupple, and gives JLR week 1 sales= 138. (Week 2, 232, ..., week 52, 186) means week 2 sales = 232 abd 52 sales = 186 JLRs.

Transaction means execution of instructions in two interrelated entities, such as a query and the database.

Database transactional model refers to a model for transactions, such as the one following the ACID
(Section 3.2) or BASE properties (Section 3.2.3).

My SQL refers to a widely used open-source database, which excels as a content management server.

Oracle refers to a widely used object-relational DBMS, written in the C++ language that provides applications integration with service-oriented architectures and has high reliability. Oracle has also released the NoSQL database system.

DB2 refers to a family of database server products from IBM with built-in support to handle advanced Big Data analytics.

Sybase refers to database server based on relational model for businesses, primarily on UNIX. Sybase was the first enterprise-level DBMS in Linux.

MS SQL server refers to a Microsoft-developed RDBMS for enterprise-level databases that supports both SQL and NoSQL architectures.

PostgreSQL refers to an enterprise-level, object-relational DBMS. PostgreSQL uses procedural languages like Perl and Python, in addition to SQL.

This chapter describes NoSQL data architecture patterns, NoSQL for increasing the flexibility in data store architecture, NoSQL usages in Big Data management, and the solutions, such as MongoDB and Cassandra. Section 3.2 describes NoSQL data stores for Big Data usages, schema-less models, and increasing the flexibility for data manipulation. Section 3.3 describes NoSQL data-architecture patterns: Key-value stores, graph stores, column family stores, tabular stores, document stores, object data stores, and variations of NoSQL architectural patterns. Section 3.4 describes NoSQL for managing Big Data, solutions for Big Data, and types of Big Data problems. Section 3.5 describes use of shared-nothing architecture, choosing a distribution model, master-slave versus peer-to-peer, and four ways by which NoSQL handles Big Data problems. Sections 3.6 describes MongoDB and query commands used in the applications. Section 3.7 describes Cassandra databases, data-model, clients and integration with Hadoop and applications.

3.2 NOSQL DATA STORE

LO 3.1

NoSQL d~t;i stora Big Dm so'llJ.lllions, sohem1~l'ess mod~ls, an([! i ncrt!:3Silillgi fileld bi lity ifor cara man ipll.!lla:tion

SQL is a programming language based on relational algebra. It is a declarative language and it defines the data schema . SQL creates databases and RDBMSs. RDBMS uses tabular data store with relational algebra, precisely defined operators with relations as the operands. Relations are a set of tuples. Tuples are named attributes. A tuple identifies uniquely by keys called candidate keys.

Transactions on SQL databases exhibit ACID properties. ACID stands for atomicity, consistency, isolation and durability.

ACID Properties in SQL Transactions

Following are the meanings of these characteristics during the transactions.

Atomicity of transaction means all operations in the transaction must complete, and if interrupted, then must be undone (rolled back). For example, if a customer withdraws an amount then the bank in first operation enters the withdrawn amount in the table and in the next operation modifies the balance with new amount available. Atomicity means both should be completed, else undone if interrupted in between.

Consistency in transactions means that a transaction must maintain the integrity constraint, and follow the consistency principle. For example, the difference of sum of deposited amounts and withdrawn amounts in a bank account must equal the last balance. All three data need to be consistent.

Isolation of transactions means two transactions of the database must be isolated from each other and done separately.

Durability means a transaction must persist once completed.

Triggers, Views and Schedules in SQLDatabases

Trigger is a special stored procedure. Trigger executes when a specific action(s) occurs within a database, such as change in table data or actions such as UPDATE, INSERT and DELETE. For example, a Trigger store procedure inserts new columns in the columnar family data store.

View refers to a logical construct, used in query statements. A View saves a division of complex query instructions and that reduces the query complexity. Viewing of a division is similar to a view of a table. View does not save like data at the table. Query statement when uses references to a view, the statement executes the View. Query (processing) planner combines the information in View definition with the remaining actions on the query. A query planner plans how to break a query into sub-queries for obtaining the required answer. View, hides the query complexity by dividing the query into smaller, more manageable pieces.

Schedule refers to a chronological sequence of instructions which execute concurrently. When a transaction is in the schedule then all instructions of the transaction are included in the schedule. Scheduled order of instructions is maintained during the transaction. Scheduling enables execution of multiple transactions in allotted time intervals.

Join in SQL Databases

SQL databases facilitate combining rows from two or more tables, based on the related columns in them. *Combining* action uses *Join* function during a database transaction. *Join* refers to a clause which combines. Combining the products (AND operations) follows next the selection process. A Join operation does pairing of two tuples obtained from different relational expressions. Joins, if and only if a given Join condition satisfies. Number of Join operations specify using relational algebraic expressions. SQL provides JOIN clause, which retrieves and joins the related data stored across multiple tables with a single command,Join. For example, consider an SQL statement:

```
Select KitKatSales From TransactionsTbl INNER JOIN
ACVMSalesTbl ON TransactionsTbl.KitKatSales =
TransactionsTbl.KitKatSales;
```

The statement selects those records in a column named KitKatSales which match the values in two tables: one TransactionsTbl and other ACVMSalesTbl.

Relational databases and RDBMS developed using SQL have issues of scalability and distributed design. This is because all tuples need to be on the same data node. The database has an issue of indexing over distributed nodes. They do not model the hierarchical, object-oriented, semi-structured or graph databases.

Database Tables have relationships between them which are represented by related fields. RDBMS allows the Join operations on the related columns. The traditional RDBMS has a problem when storing the records beyond a certain physical storage limit. This is because RDBMS does not support horizontal scalability (Section 1.3.1).

For example, consider sharding a big table in a DBMS into two. Assume writing first 0.1 million records (1 to 100000) in one table and from 100001 in another table. Sharding a database means breaking up into many, much smaller databases that share nothing, and can distribute across multiple servers. Handling of the Joins and managing data in the other related tables are cumbersome processes, when using the sharding.

The problem continues when data has no defined number of fields and

formats. For example, the data associated with the choice of chocolate flavours of the users of ACVM in Example 1.6(i). Some users provide a single choice, while some users provide two choices, and a few others want to fill three best flavours of their choice.

User Id	Choitt
	Dairy Milk
2	Dairy Milk. KitKat
3	KitKat. Snicker, Munch

Defining a field becomes tough when a field in the database offers choice between two or many. This makes RDBMS unsuitable for data management in Big Data environments as well as data in their real forms.

SQL compliant format means that database tables constructed using SQL and they enable processing of the queries written using SQL. 'NoSQL' term conveys two different meanings: (i) does not follow SQL compliant formats, (ii)"Not only SQL" use SQL compliant formats with variety of other querying and access methods.

3.2.1 NoSQL

A new category of data stores is NoSQL (means Not Only SQL) data stores. NoSQL is an altogether new approach of thinking about databases, such as schema flexibility, simple relationships, dynamic schemas, auto sharding, replication, integrated caching, horizontal scalability of shards, distributable tuples, semi-structures data and flexibility in approach.

Issues with NoSQL data stores are lack of standardization in approaches, processing difficulties for complex queries, dependence on eventually consistent results in place of consistency in all states.

3.2.1.1 Big Data NoSQL or Not-Only SQL

NoSQl or No1t Olilly SQ:L is a Olass mfllllcn□-felational data storage SY.5.~rms, fie:wibl~ da:tta lillod'els and litrumltipile sch~rma

NoSQL DB does not require specialized RDBMS like storage and hardware for processing. Storage can be on a cloud. Section 1.6.2.1 introduced NoSQL data storage system. NoSQL records are in non-relational data store systems. They use flexible data models. The records use multiple schemas.

NoSQL data stores are considered as semi-structured data. Big Data Store uses NoSQL. Figure 1.7 showed co-existence of data store at server or cloud with SQL, RDBMS with NoSQL and Big Data at Hadoop, Spark, Mesos, 53 or compatible Clusters. However, no integration takes place with applications that are based on SQL. NoSQL data store characteristics are as follows:

1. NoSQL is a class of non-relational data storage system with flexible data model. Examples of NoSQL data-architecture patterns of datasets are key-value pairs, name/value pairs, Column family
   Big-data store, Tabular data store, Cassandra (used in Facebook/ Apache), HBase, hash table [Dynamo (Amazon 53)], unordered keys using JSON (CouchDB), *JSON* (PNUTS), *JSON* (MongoDB), Graph Store, Object Store, ordered keys and semi-structured data storage systems.
2. NoSQL not necessarily has a fixed schema, such as table; do not use the concept of Joins (in distributed data storage systems); Data written at one node can be replicated to multiple nodes. Data store is thus fault• tolerant. The store can be partitioned into unshared shards.

Features in NoSQL Transactions NoSQL transactions have following features:

(i) Relax one or more of the ACID properties.

(ii) Characterize by two out of three properties (consistency, availability and partitions) of CAP theorem, two are at least present for the application/ service/ process.

(iii) Can be characterized by BASE properties (Section 3.2.3).

Big Data NoSQL solutions use standalone-server, master-slave and peer-to• peer distribution models.

Big Data NoSQL Solutions NoSQL DBs are needed for Big Data solutions. They play an important role in handling Big Data challenges. Table 3.1 gives the examples of widely used NoSQL data stores.

Table 3.1 NoSQL data stores and their characteristic features

NoSQL Data store	Description

Apache's HBase	HDFS compatible, open-source and non-relational data store written inJava; A column-family based NoSQL data store, data store providing BigTable-like capabilities (Sections 2.6 and 3.3.3.2); scalability, strong consistency, versioning, configuring and maintaining data store characteristics
Apache's MongoDB	HDFS compatible; master-slave distribution model (Section 3.5.1.3); document-oriented data store withJSON-like documents and dynamic schemas; open-source, NoSQL, scalable and non-relational database; used by Websites Craigslist, eBay, Foursquare at the backend
Apache's Cassandra	HDFS compatible DBs; decentralized distribution peer-to-peer model (Section 3.5.1.4); open source; NoSQL; scalable, non-relational, column-family based, fault-tolerant and tuneable consistency (Section 3.7) used by Facebook and Instagram
Apache's CouchDB	A project of Apache which is also widely used database for the web. CouchDBconsists of Document Store. It uses the *)SON* data exchange format to store its documents,JavaScript for indexing, combining and transforming documents, and HTTP APis
Oracle NoSQL	Step towards NoSQL data store; distributed key-value data store; provides transactional semantics for data manipulation, horizontal scalability, simple administration and monitoring
Riak	An open-source key-value store; high availability (using replication concept), fault tolerance, operational simplicity, scalability and written in Erlang

CAP Theorem Among C, A and P, two are at least present for the application/service/process. *Consistency* means all copies have the same value like in traditional DBs. *Availability* means at least one copy is available in case a partition becomes inactive or fails. For example, in web applications, the other copy in the other partition is available. *Partition* means parts which are active but may not cooperate (share) as in distributed DBs.

1. *Consistency in distributed databases* means that all nodes observe the same data at the same time. Therefore, the operations in one partition of the database should reflect in other related partitions in case of distributed database. Operations, which change the sales data from a specific showroom in a table should also reflect in changes in related tables which are using that sales data.

2. *Availability* means that during the transactions, the field values must be available in other partitions of the database so that each request receives a response on success as well as failure. (Failure causes the response to request from the replicate of data). Distributed databases require transparency between one another. Network failure may lead to data unavailability in a certain partition in case of no replication. Replication ensures availability.
3. *Partition* means division of a large database into different databases without affecting the operations on them by adopting specified procedures.

Partition tolerance: Refers to continuation of operations as a whole even in case of message loss, node failure or node not reachable.

Brewer's CAP (c.onsistency, Availability and £.artition Tolerance) theorem demonstrates that any distributed system cannot guarantee C, A and P together.

1. Consistency- All nodes observe the same data at the same time.
2. Availability- Each request receives a response on success/failure.
3. Partition Tolerance-The system continues to operate as a whole even in case of message loss, node failure or node not reachable.

Partition tolerance cannot be overlooked for achieving reliability in a distributed database system. Thus, in case of any network failure, a choice can be:

- Database must answer, and that answer would be old or wrong data (AP).
- Database should not answer, unless it receives the latest copy of the data (CP).

The CAP theorem implies that for a network partition system, the choice of consistency and availability are mutually exclusive. CA means consistency and availability, AP means availability and partition tolerance and CP means consistency and partition tolerance. Figure 3.1 shows the CAP theorem usage in Big Data Solutions.

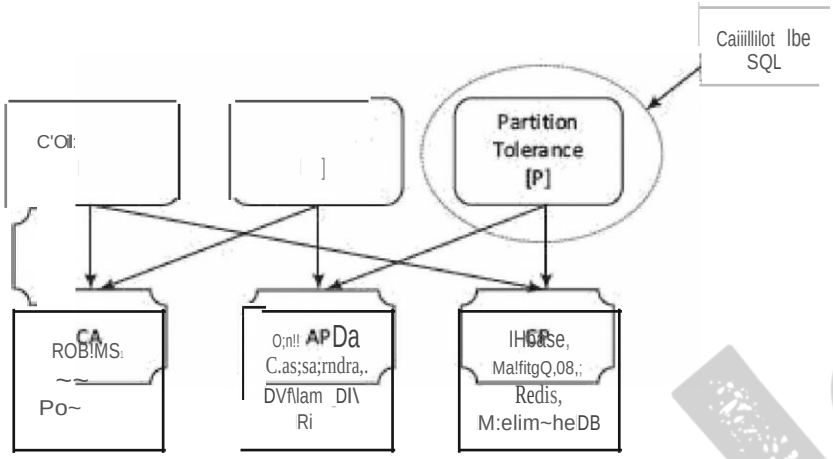


Figure 3.1 CAP theorem in Big Data solutions

3.2.2 Schema-less Models

Schema of a database system refers to designing of a structure for datasets and data structures for storing into the database. NoSQL data not necessarily have a fixed table schema. The systems do not use the concept of Join (between distributed datasets). A cluster-based highly distributed node manages a single large data store with a NoSQL DB.

Data written at one node replicates to multiple nodes. Therefore, these are identical, fault-tolerant and partitioned into shards. Distributed databases can store and process a set of information on more than one computing nodes.

NoSQL data model offers relaxation in one or more of the ACID properties (Atomicity, consistence, isolation and durability) of the database. Distribution follows CAP theorem. CAP theorem states that out of the three properties, two must at least be present for the application/service/process. (Section 3.2.1).

Figure 3.2 shows characteristics of Schema-less model for data stores. ER stands for entity-relation modelling.

Relations in a database build the connections between various tables of data. For example, a table of subjects offered in an academic programme can be connected to a table of programmes offered in the academic institution. NoSQL

data stores use non-mathematical relations but store this information as an aggregate called metadata.

Metadata refers to data describing and specifying an object or objects. Metadata is a record with all the information about a particular dataset and the inter-linkages. Metadata helps in selecting an object, specifications of the data and, usages that design where and when. Metadata specifies access permissions, attributes of the objects and enables additions of an attribute layer to the objects. Files, tables, documents and images are also the objects.

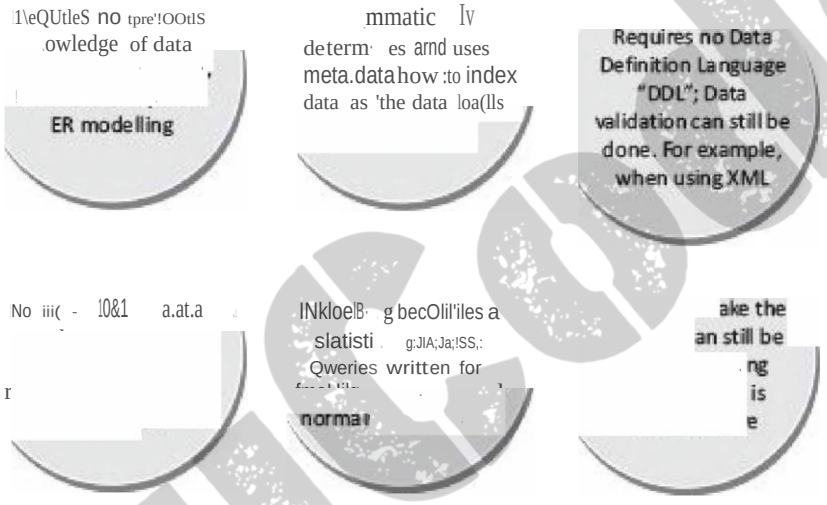


Figure 3.2 Characteristics of Schema-less model

3.2.3 Increasing Flexibility for Data Manipulation

Consider database 'Contacts'. They follow a fixed schema. Now consider students' admission database. That also follow a fixed schema. Later, additional data is added as the course progresses. NoSQL data store characteristics are schema-less. The additional data may not be structured and follow fixed schema. The data store consists of additional data, such as documents, blogs, Facebook pages and tweets.

NoSQL data store possess characteristic of increasing flexibility for data manipulation. The new attributes to database can be increasingly added. Late binding of them is also permitted.

BASE is a flexible model for NoSQL data stores. Provisions of BASE increase flexibility.

BASE Properties BA stands for basic availability, S stands for soft state and E stands for eventual consistency.

1. *Basic availability* ensures by distribution of shards (many partitions of huge data store) across many data nodes with a high degree of replication. Then, a segment failure does not necessarily mean a complete data store unavailability.
2. *Soft state* ensures processing even in the presence of inconsistencies but achieving consistency eventually. A program suitably takes into account the inconsistency found during processing. NoSQL database design does not consider the need of consistency all along the processing time.
3. *Eventual consistency* means consistency requirement in NoSQL databases meeting at some point of time in future. Data converges eventually to a consistent state with no time-frame specification for achieving that. ACID rules require consistency all along the processing on completion of each transaction. BASE does not have that requirement and has the flexibility.

BASE model is not necessarily appropriate in all cases but it is flexible and is an alternative to SQL-likeadherence to ACID properties. Example 3.11 (Section 3.3.5) explains the concept of BASE in transactions using graph databases.

Schema is not a necessity in NoSQL DB, implying information storage flexibility. Data can store and retrieve without having knowledge of how a database stores and functions internally.

Following is an example to understand the increasing flexibility for data manipulation.

EXAMPLE 3.1

Use examples of database for the students in various university courses to demonstrate the concept of increasing flexibility in NoSQL DBs.

SOLUTION

Figure 3.3 shows increasing flexibility concept using additional data models.

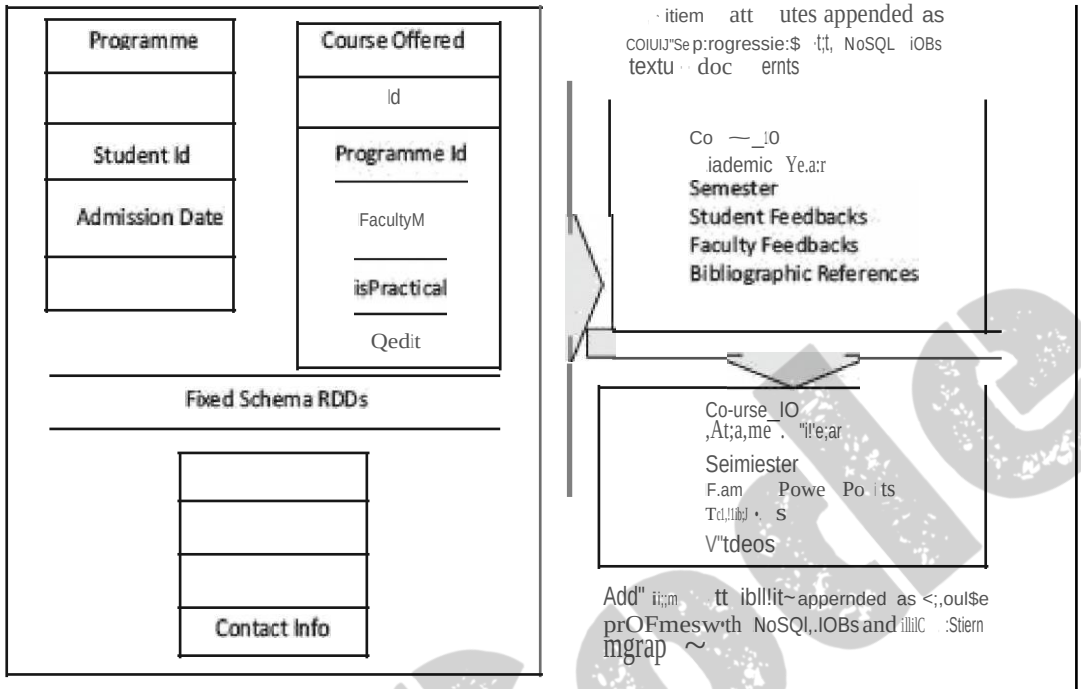


Figure 3.3 Increasing flexibility in NoSQL DB of students

Self-Assessment Exercise linked to LO 3.1

1. Explain when will you use the following: MongoDB, Cassandra, CouchDB, Oracle NoSQL and Riak.
2. How does CAP theorem hold in NoSQL databases?
3. How do ACID and BASE properties differ?
4. Why is the consistency not enforcable in NoSQL distributed databases during a transaction processing?
5. List characteristics of NoSQL data store.
6. Why is metadata a must when using NoSQL data store?

3.3 | NOSQL DATA ARCHITECTURE PATTERNS

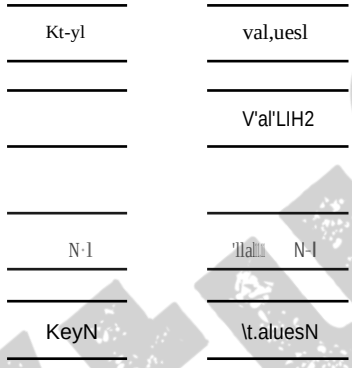
NoSQL data stores broadly categorize into architectural patterns described in the following subsections:

LO 3.2

NoSQL d\3:ta- archltectur@ paittemnis. namely lt@y• value palrs, oolum n farmif[Y. mgiTabl'@. RC, OR[, Pair,c1u et am~ tabular da~ rtores, dC1r].11m1ent: stores. object dab stores. gra;ph d~t.abais~s. and d~~ mires wiith Va'rii.rliions mi :.:rrd11t@di:ral pattems

3.3.1 Key-Value Store

The simplest way to implement a schema-less data store is to use key-value pairs. The data store characteristics are high performance, scalability and flexibility. Data retrieval is fast in key-value pairs data store. A simple string called, key maps to a large data string or BLOB (Basic Large Object). Key-value store accesses use a primary key for accessing the values. Therefore, the store can be easily scaled up for very large data. The concept is similar to a hash table where a unique key points to a particular item(s) of data. Figure 3.4 shows key-value pairs architectural pattern and example of students' database as key-value pairs.



Key	Value
"Ashish"	"Category: Student; Class: B.Tech.; Semester: VII; Branch: Engineering; Mobile:9999912345"
"Mayuri"	"Category: student; class: M.Tech.; Mobile:8888823456"

Figure 3.4 Example of key-value pairs in data architectural pattern

Advantages of a key-value store are as follows:

1. Data Store can store any data type in a value field. The key-value system stores the information as a BLOB of data (such as text, hypertext, images, video and audio) and return the same BLOB when the data is retrieved. Storage is like an English dictionary. Query for a word retrieves the meanings, usages, different forms as a single item in the dictionary. Similarly, querying for key retrieves the values.

2. A query just requests the values and returns the values as a single item. Values can be of any data type.
3. Key-value store is eventually consistent.
4. Key-value data store may be hierarchical or may be ordered key-value store.
5. Returned values on queries can be used to convert into lists, table• columns, data-frame fields and columns.
6. Have (i) scalability, (ii) reliability, (iii) portability and (iv) low operational cost.
7. The key can be synthetic or auto-generated. The key is flexible and can be represented in many formats: (i) Artificially generated strings created from a hash of a value, (ii) Logical path names to images or files, (iii) REST web-service calls (request response cycles), and (iv) SQL queries.

The key-value store provides client to read and write values using a key as follows:

(i) Get (key), returns the value associated with the key.

(ii) Put (key, value), associates the value with the key and updates a value if this key is already present.

(iii) Multi-get (keyl, key2, .. ' keyN), returns the list of values associated with the list of keys.

(iv) Delete (key), removes a key and its value from the data store.

Limitations of key-value store architectural pattern are:

(i) No indexes are maintained on values, thus a subset of values is not searchable.

(ii) Key-value store does not provide traditional database capabilities, such as atomicity of transactions, or consistency when multiple transactions are executed simultaneously. The application needs to implement such capabilities.

(iii) Maintaining unique values as keys may become more difficult when the

volume of data increases. One cannot retrieve a single result when a key• value pair is not uniquely identified.

(iv) Queries cannot be performed on individual values. No clause like 'where' in a relational database usable that filters a result set.

Table 3.2 gives a comparison between traditional relational data model with the key-value store model.

Table 3.2 Traditional relational data model vs. the key-value store model

Traditional relational model	Key-value store model
Result set based on row values	Queries return a single item
Values of rows for large datasets are indexed	No indexes on values
Same data type values in columns	Any data type values

Typical uses of key-value store are: (i) Image store, (ii) Document or file store, (iii) Lookup table, and
(iv) Query-cache.

Riak is open-source Erlang language data store. It is a key-value data store system. Data auto-distributes and replicates in Riak. It is thus, fault tolerant and reliable. Some other widely used key-value pairs in NoSQL DBs are Amazon's DynamoDB,Redis (often referred as Data Structure server), Memcached and its flavours, Berkeley DB, upscaledb (used for embedded databases), project Voldemort and Couchbase.

Concept of Hash Key The following example explains the hash and key-value pairs associated with a hash in traditional data.

EXAMPLE 3.2

Consider an example. Assume that student name is key, *k*. Each student grade sheet entry has a number of values or set of (secondary) key-value pairs. For example, semester grade point average (SGPAs) values and cumulative grade point average (CGPA) value. How will the hash function be used?

SOLUTION

A hash function generates an index, Ik for k. Ik should ideally be unique and should uniquely map to k. Ik is a number with few digits only, compared to a number of characters (0-255 bytes) in the main key k used as input for the hash function. Assume that total 20 numbers of entries are present between slots indices between 00 to 99. Student name may consist of several characters, but index will be just two digits.

Hash table refers to using associated key-value pairs. A set of pairs retrieve by using a hash key. The hash key is a computed index using hash function for a column. The analytics may use the hash table. The table contains hash keys in the table-columns. The entries (values) across an array of slots (also called buckets). The buckets correspond to the key for the pairs at column. The values are in the associated rows of that column.

3.3.2 Document Store

Characteristics of Document Data Store are high performance and flexibility. Scalability varies, depends on stored contents. Complexity is low compared to tabular, object and graph data stores.

Following are the features in Document Store:

1. Document stores unstructured data.
2. Storage has similarity with object store.
3. Data stores in nested hierarchies. For example, inJSON formats data model [Example 3.3(ii)], XML document object model (DOM), or machine-readable data as one BLOB. Hierarchical information stores in a single unit called *document tree*. Logical data stores together in a unit.
4. Querying is easy. For example, using section number, sub-section number and figure caption and table headings to retrieve document partitions.
5. No object relational mapping enables easy search by following paths from the root of document tree.
6. Transactions on the document store exhibit ACID properties.

Typical uses of a document store are: (i) office documents, (ii) inventory store, (iii) forms data, (iv) document exchange and (v) document search.

The demerits in Document Store are incompatibility with SQL and complexity for implementation. Examples of Document Data Stores are CouchDB and MongoDB.

Real-life Datasets Section 10.3 will describe a very large real-life dataset for Big Data analytics as an examples. An application later analyses the structures in csv, json or other, and creates data stores in new forms (Sections 10.3.2 to 10.3.4). Runs in next step ETL, analytics or other functions. (Sections 10.4 to 10.6). This feature is called late binding (schema-on-read, or schema-on-need).

CSV and JSON File Formats CSV data store is a format for records [Example 1.9 and Example 3.3(i)]. CSV does not represent object-oriented databases or hierarchical data records. *JSON* and XML represent semistructured data, object• oriented records and hierarchical data records. *JSON* (lava Script Object Notation) refers to a language format for semistructured data. JSON represents object-oriented and hierarchical data records, object, and resource arrays in JavaScript.

The following example explains the CSV and *JSON* object concept and aspects of CSV andJSON file formats.

EXAMPLE3.3

Assume Preeti gave examination in Semester 1 in 1995 in four subjects. She gave examination in five subjects in Semester 2 and so on in each subsequent semester. Another student, Kirti gave examination in Semester 1 in 2016 in three subjects, out of which one was theory and two were practical subjects. Presume the subject names and grades awarded to them.

(i) Write two CSV files for cumulative grade-sheets for both the students. Point the difficulty during processing of data in these two files.

(ii) Write a file in *JSON* format with each student grade-sheet as an object instance. How does the object-oriented and hierarchical data record in *JSON* make processing easier?

SOLUTION

(i) Two CSV file for cumulative grade-sheets are as follows:

CSV file for Preeti consists of the following nine lines each with four

columns:

Semester, Subject Code, Subject Name, Grade

1, CSlOl, ""Theory of Computations?", 7.8.

1, CS102,1, ""Computer Architecture?", 7.8.

2, CS204, ""Object Oriented Programming?", 7.2.

2, CS205, ""Data Analytics?", 8.1.

The CSV file for Kirti consist of following five lines each with five columns: Semester, Subject Type, Subject Code, Subject Name, Grade

1, Theory, EllOl, ""Analog Electronics?", 7.6.

1, Theory, El102,1, ""Principles of Analog Communication?", 7.5.

1, Theory, El103, ""Digital Electronics?", 7.8.

1, Practical, CS104, ""Analog ICs"", 7.2

1, Practical, CS105, ""Digital ICs"", 8.4

A column head is a key. Number of key-value pairs are (4 x 9) = 36 for preetiGradeSheet.csv and (5 x 5) = 25 for kirtiGradeSheet.csv. Therefore, when processing student records, merger of both files into a single file will need a program to extract the key-value pairs separately, and then prepare a single file.

(ii) JSON gives an advantage of creating a single file with multiple instances and inheritances of an object. Consider a single JSON file, *studentGradeSheetsjson* for cumulative grade-sheets of many students. *Student_Grades* object is top in the hierarchy. Each *student_name* object is next in the hierarchy with object consisting of student name, each with number of instances of subject codes, subject types, subject titles and grades awarded. Each student name object-instance extends in student grades object-instances.

Part of the file construct can be as follows:

```
    0:  {
            _id: 0,
m.asterfile: ~students_Grades~,
in:stancetype:  ·~single'•,
mandatory: true,
..,description":  "Uni.queLy identifies student grade master file Object
Students  Gradea  ~
 ~resourcedefs.,,.:  {
  ~1/,I:   {
    _id:1,
     name: ~studentNamen,
     instancetype: ~multiple~,
   ~descriptionn:  ~Identifies  a semester  of the studentName andn
        re:aourcedefe:  {
        -i20  0....  {
         id:200
          studentName: ~Kirti-
           instancetype:  ~single'•
           resourcedefs:
             '"201....  {
      id: 201 semester:  ~1.,,,,
      subjectType:  ~Theory'••
      subjectCode:  ~EL101~,
        subjectName: ~Analog Blectronicen
        Grade: 7.6
         type: '-'string",
         ''descriptLon" : ~ instance Grade for a subject Analog Blectronics~
          }
    ·~2 02,. :  {
   _ id:202,
          '-'203....  {_id:203
    semester:  v.1n,
    aubject'I'ype: ~Theory•
   subjectCode:  ~B1102n
    subjectName:  ~Principles  of Analog  Communication.,,.
   Grade:  7.5, type=  ~string"


                          }
            {.-.
                              }
```

XML (extensible Markup Language) is an extensible, simple and scalable language. Its self-describing format describes structure and contents in an easy to understand format. XML is widely used. The document model consists of root element and their sub-elements. XML document model has a hierarchical structure. XML document model has features of object-oriented records. XML format finds wide uses in data store and data exchanges over the network. An XML document is semi-structured.

Document store appears quite similar to a key-value store and an object store. They are complex in implementation and are SQL incompatible. They have no object-relational layer for mapping and thus enable agile development of text analytics. No sharding of data takes place into the tables. Although the values stored as documents, follows structure and encoding of the managed data

The database stores and retrieves documents, such as XML, *JSON,* BSON (Binary-encoded Script Object Notation (for objects)). The documents are self• describing, hierarchical tree-structured consisting of maps, collections and scalar values. The documents stored are similar to each other but do not have to be the same. Some of the popular document data stores are CouchDB,MongoDB, Terrastore, OrientDB and RavenDB.

Certain NoSQL DBs enable ACID rule-based transactions also. Examples of document data stores are MongoDB,Apache Couchbase and MarkLogic.

CouchDBuses the JSON store data, HTTP APis for connectivity, JavaScript for the query language and MapReduce for processing.

Document JSON Format CouchDB Database Apache CouchDB is an open• source database. Its features are:

1. CouchDB provides mapping functions during querying, combining and filtering of information.
2. CouchDB deploys JSON Data Store model for documents. Each document maintains separate data and metadata (schema).
3. CouchDBis a multi-master application. Write does not require field locking when controlling the concurrency during multi-master application.
4. CouchDBquerying language is JavaScript. Java script is a language which

documents use to transform.

5. CouchDB queries the indices using a web browser. CouchDB accesses the documents using HTTP APL HTTP methods are Get, Put and Delete (Section 3.3.1).
6. CouchDB data replication is the distribution model that results in fault tolerance and reliability.

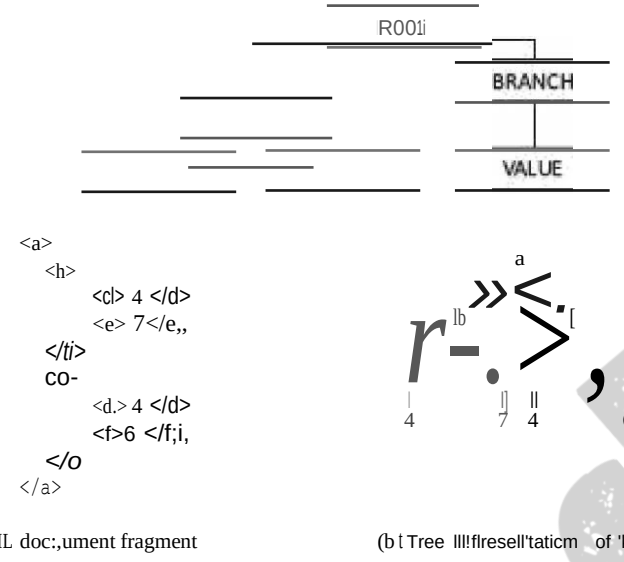
Document JSON Format-MongoDB Database MongoDB Document database provides a rich query language and constructs, such as database indexes allowing easier handling of Big Data.

Example of Document in Document Store:

```
      "'id.,,.:    "'1001"
      "'Student      Name":
{
"'First":  "'Ashish",
•Middle•: "'Kumar•,
•La.st•   :   •Ra.i'"
     }
"'Category":  "'Student",
"'Class•:   •s.Tech.•,
•semester•: ~v1r•,
~Branchg: •computer  Bngineeringg,
"'Mobile":  "'12345"
}
```

The document store allows querying the data based on the contents as well. For example, it is possible to search the document where student's first name is "Ashish", Document store can also provide the search value's exact location. The search is by using the document path. A type of key accesses the leaf values in the tree structure. Since the document stores are schema-less, adding fields to documents (XML or *JSON)* becomes a simple task.

Document Architecture Pattern and Discovering Hierarchical Structure Following is example of an XML document in which a hierarchical structure discovers later. Figure 3.5 shows an XML document architecture pattern in a document fragment and document tree structure.



(at XML doc:,ument fragment (b t Tree lll!flresell'taticm of 'k-agment

Figure 3.5 XML document architecture pattern

The document store follows a tree-like structure (similar to directory structure in file system). Beneath the root element there are multiple branches. Each branch has a related path expression that provides a way to navigate from the root to any given branch, sub-branch or value.

XQuery and XPath are query languages for finding and extracting elements and attributes from XML documents. The query commands use sub-trees and attributes of documents. The querying is similar as in SQL for databases. XPath treats XML document as a tree of nodes. XPath queries are expressed in the form ofXPath expressions. Following is an example ofXPath expressions:

EXAMPLE 3.4

Give examples of XPath expressions. Let outermost element of the XML document is *a*.

SOLUTION

An XPath expression */a/b/* c selects c elements that are children of *b* elements that are children of element *a* that forms the outermost element of the XML document.

An XPath expression */ a/b[c=5]* selects elements *b* and c that are children of

a and value of c element is *5.*

An XPath expression / a[b/ c]/ d selects elements c and d where c is child of *b* and *b* and *d* are children of *a.*

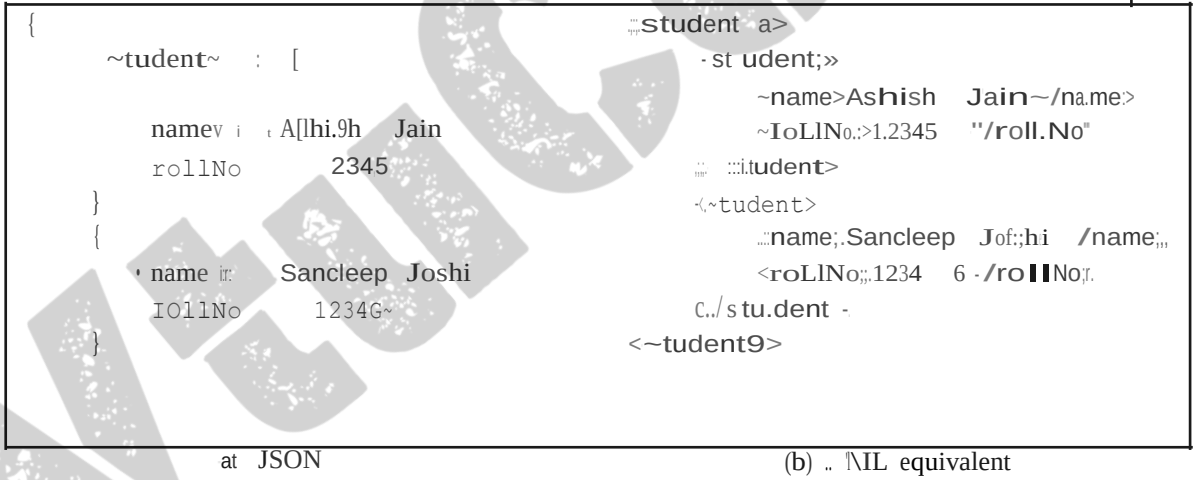
XML and JSON both are designed to form a simple and standard way of describing different kinds of hierarchical data structures. They are popularly used for storing and exchanging data. The following example explains the concept of Document Store inJSON and XML for hierarchical records.

EXAMPLE 3.5

Give the structures of XML and JSON document fragments for a student record.

SOLUTION

Following are the structures:



at JSON

(b) IL equivalent

When compared with XML, JSON has the following advantages:

- XML is easier to understand but XML is more verbose than JSON.
- XML is used to describe structured data and does not include arrays, whereas JSON includes arrays.
- *JSON* has basically key-value pairs and is easier to parse fromJavaScript.

- The concise syntax of *JSON* for defining lists of elements makes it preferable for serialization of text format objects.

Document Collection A collection can be used in many ways for managing a large document store. Three uses of a document collection are:

1. Group the documents together, similar to a directory structure in a file system. (A directory consists of grouping of file folders.)
2. Enables navigating through document hierarchies, logically grouping similar documents and storing business rules such as permissions, indexes and triggers (special procedure on some actions in a database).
3. A collection can contain other collections as well.

3.3.3 TabularData

Tabular data stores use rows and columns. Row-head field may be used as a key which access and retrieves multiple values from the successive columns in that row. The OLTP is fast on in-memory row-format data.

Oracle DBs provide both options: columnar and row format storages. Generally, relational DB store is *in-memory row-based data,* in which a key in the first column of the row is at a memory address, and values in successive columns at successive memory addresses. That makes OLTP easier. All fields of a row are accessed at a time together during OLTP. Different rows are stored in different addresses in the memory or disk. In-memory row-based DB stores a row as a consecutive memory or disk entry. This strategy makes data searching and accessing faster during *transactions* processing.

In-memory column-based data has the keys (row-head keys) in the first column of each row at successive memory addresses. The next column of each row after the key has the values at successive memory addresses. The values in the third column of each row are at the next memory addresses in succession, and so on up to N columns. The N can be a very large number. The column-based data makes the OLAP easier. All fields of a column access together. All fields of a set of columns may also be accessed together during OLAP. Different rows are stored in different addresses in the memory or disk, but each row values are now not at successive addresses. In-memory column-based DB store a column as

a consecutive memory or disk entry. This strategy makes the analytics processing fast.

Following subsections describe NoSQL format data stores based on tabular formats.

3.3.3.1 Column Family Store

Columnar Data Store A way to implement a schema is the divisions into columns. Storage of each column, successive values is at the successive memory addresses. Analytics processing (AP) In-memory uses columnar storage in memory. A pair of row-head and column-head is a key-pair. The pair accesses a field in the table.

All values in successive fields in a column consisting of multiple rows save at consecutive memory addresses. This enables fast accesses during in-memory analytics, which includes CPU accesses and analyses using memory addresses in which values are cached from the disk before processing. The OLAP (on-line AP) is also fast on in-memory column-format data. An application uses a combination of row head and a column head as a key for access to the value saved at the field.

Column-Family Data Store Column-family data-store has a group of columns as a column family. A combination of row-head, column-family head and table• column head can also be a key to access a field in a column of the table during querying. Combination of row head, column families head, column-family head and column head for values in column fields can also be a key to access fields of a column. A column-family head is also called a super-column head.

Examples of columnar family data stores are HBase, BigTable, HyperTable and Cassandra. The following example explains a column-family data store and why OLAP is fast in-memory column data store in memory:

EXAMPLE 3.6

Consider Example 1.6(i). Assume in-memory columnar storage. Data for a large number of ACVMs with an ACVM_ID each, store in column 1. Data for each day sales at each ACVM for KitKat, Milk, Fruit and Nuts, Nougat and Oreo store in Columns 2 to 6. Each row has six cells (ID -five sales data).

(i) How do the column key values store in memory?

(ii) How do the values store in the memory in columnar storage format?

(iii) How does analytics of each day's sales help?

(iv) Why do in-memory columnar storage result in fast computations during analytics?

(v) How are a column family and column-family head (key) specified?

(vi) How do a column-families group specify?

(vii) How do row groups form? What is the advantage of division into sub• groups?

SOLUTION

Assume the following columnar storage at memory:

(i) Column and row keys

Addresses 1000, 2000, 3000, ..., 6000 save the column keys. Address 1000 stores string 'ACVM_ID'. Then the adresses 1001, 1002, ..., 1999 store the row keys, means ACVM_IDs.

Chocolate name of five flavours store at addresses 2000, 3000, 4000, 5000, and 6000.

(ii) Column field values

Column 1 ACVM_IDs store at address 1001 to 1999 for 999 ACVMs. Sales in a day for KitKat, Milk, Fruit and Nuts, Nougat and Oreo store at addresses 2001 to 2999, 3001 to 3999, 4001 to 4999, 5001 to 5999, and 6001 to 6999.

Table 3.3 gives sample values in the columns for a day's sales data. The table also gives the keys for row groups, rows (ACVM_IDs), a column family group, two column families and five column heads for 5 flavours of chocolates. The table gives a row group of just 100 rows, just for the sale of assumption.

Figure 3.6 shows fields in columnar storage and addresses in memory. The figure shows ACVM_IDs as well as each day's sales of each flavour of chocolate at 999 ACVMs. Following are the addresses assigned to the

values in fields of Table 3.3:

Table 3.3 Each day's sales of chocolates on 999 ACVMs

	ACVM_ID	Nestle Chocolate Flavours Group				
		Popular Flavours Family		Costly Flavours Family		
		KitKat	Milk	Fruit and Nuts	Nougat	Oreo
Row-group_1 for IDs 1 to 100	1	360	150	500	101	222
	2	289	175	457	145	317
	....	....	....		....	....
Row-group_m for IDs 901 to 999	....	....	....		....	....
	998	123	201	385	199	310
	999	75	215	560	108	250

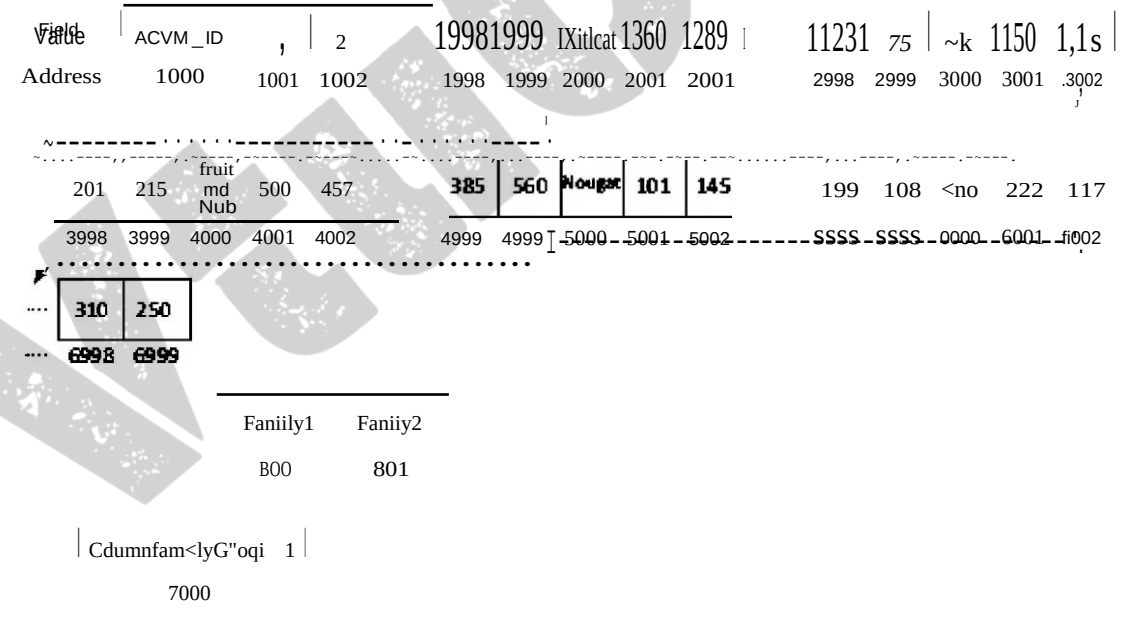


Figure 3.6 Fields in columnar storage and addresses in memory.

(iii) An analytics application computes the results, such as (a) total sales of each flavour, KitKat, Milk, Fruit and Nuts, Nougat and Oreo, each day, (b) Each ACVM requirement for refilling at each ACVM each day, (c) ID of maximum sales of chocolates each day, (d) cluster of ACVMs showing highest sales, classification of ACVMs as low, moderate and high sales.

(iv) Consider first address of sales data for KitKat, address., kk = 1001 of machine ID, ACVMid at address, =1000. Total sales at all 999 ACVMs in a day requires the sum of values between address 1001 to 1999. Increment to the next address is fast when compared to the case when during the execution the value addresses compute from a table of pointers for them. When values are in the row storage format, the chocolate KitKat sales data at the machines will be at addresses 1001, 1007, 1013, ... The table of pointers or computations of address., kk + n x N + 1, where N is number of columns for each row, is required, and n = 0, 1, 2, ..., 998, 999. The processing takes longer compared to instruction for increment of pointed address to next memory address.

Therefore, analytics of (a), (b), (c) and (d) is quicker fast in case of In• memory columnar storage compared to row format storage in memory.

(v) Columns in Table 3.3 for KitKat and Milk form a group as one family. Columns for Fruit and Nuts, Nougat, and Oreo form a group as second family. The key for one family is 'Popular Flavours Family' and second family is 'Costly Flavours Family'. The keys of column families can save at the addresses 800, 801, ...

(vi) Two column-families in Table 3.3, Popular Flavours Family and Costly Flavours Family form a super group, 'Nestle Chocolate Flavours'. The key for super group of column-families group is 'Nestle Chocolate Flavours'. The keys of column-family super groups can save at the addresses 700, 701, 702, ...

(vii) A set of fields in all column families for ACVMs, say of IDs 1 to 100 can

be grouped into row-group_l. Number of row-groups can then be processed as separate sub-tables, parallelly in Big Data environment. The keys of row groups can save at the addresses 600, 601, 602, ...

Columns Families Two of more columns in data-store group into one column family. Table 3.3 considered two families.

Sparse Column Fields A row may associate a large number of columns but contains values in few column fields. Similarly, many column fields may not have data. Columns are logically grouped into column families. Column-family data stores are then similar to sparse matrix data. Most elements of sparse matrix are empty. Data stores at memory addresses is columnar-family based rather than as row based. Metadata provide the column-family indices of not empty column fields.

That facilitates OLAP of not empty column families faster. For example, assume hash key in a column heading field and values in successive rows at one column family. For another key, the values will be in another column family.

Grouping of Column Families Two or more column-families in data store form a super group, called super column. Table 3.3 consists of one such group (super column), 'Nestle Chocolate Flavours Group'.

Grouping into Rows When number of rows are very large then horizontal partitioning of the table is a necessity. Each partition forms one row-group. For example, a group of 1 million rows per partition. A row group thus has all column data store in the memory for in-memory analytics. Practically, row groups are chosen such that memory required for the group is above, say 10 MB and below the maximum size which can cached and buffered in memory, say 1 GB for in-memory analytics.

Data caching, buffering in memory and storing back at disk takes time. So frequent disk accesses remain controlled. Therefore, minimum row-group size of 10 MB is practical (Table 3.3 considered a row group of just 100 rows for the purpose of explaining the addressing and use of keys in a columnar-family data store).

Characteristics of Columnar Family Data Store Columnar family data store imbibes characteristics of very high performance and scalability, moderate level

of flexibility and lower complexity when compared to the object and graph databases. Advantages of column stores are:

1. *Scalability:* The database uses row IDs and column names to locate a column and values at the column fields. The interface for the fields is simple. The back-end system can distribute queries over a large number of processing nodes without performing any Join operations. The retrieval of data from the distributed node can be least complicated by an intelligent plan of row IDs and columns, thereby increasing performance. Scalability means addition of number of rows as the number of ACVMs increase in Example 1.6(i). Number of processing instructions is proportional to the number of ACVMs due to scalable operations.
2. *Partitionability:* For example, large data of ACVMs can be partitioned into datasets of size, say
   1 MB in the number of row-groups. Values in columns of each row-group, process in-memory at a partition. Values in columns of each row-group independently parallelly process in-memory at the partitioned nodes.
3. *Availability:* The cost of replication is lower since the system scales on distributed nodes efficiently. The lack of Join operations enables storing a part of a column- family matrix on remote computers. Thus, the data is always available in case of failure of any node.
4. *Tree-like columnar structure* consisting of column-family groups, column families and columns. The columns group into families. The column families group into column groups (super columns). A key for the column fields consists of three secondary keys: column-families group ID, column• family ID and column-head name.
5. *Adding new data at ease:* Permits new column *Insert* operations. Trigger operation creates new columns on an Insert. The column-field values can add after the last address in memory if the column structure is known in advance. New row-head field, row-group ID field, column-family group, column family and column names can be created at any time to add new data.

6. Querying *all the field values* in a column in a family, all columns in the family or a group of column-families, is fast in in-memory column-family data store.
7. *Replication of* columns: HDFS-compatible column-family data stores replicate each data store with default replication factor= 3.
8. *No optimization for* Join: Column-family data stores are similar to sparse matrix data. The data do not optimize for Join operations.

Column-family data store in a format in which store set of column family field values which are not empty (null or zero). Metadata of the matrix consists of hash keys that reference each set distinctly.

Typical uses of column store are: (i) web crawling, (ii) large sparsely populated tables and (iii) system that has high variance.

HDFS is highly reliable for very long running queries. However, IO operations are slow. Columnar storage is a solution for faster IOs. Columnar storage in memory stores the data actually required for the IOs. Only columns needing the access load during execution. Also, a columnar-object data store can be compressed or encoded. The encoding is according to the data type. Also, the executions of different columns or column partitions can be in parallel at the cluster data-nodes.

3.3.3.2 BigTable Data Store

Examples of widely used column-family data store are Google's BigTable, HBase and Cassandra. Keys for *row key*, *column key*, *timestamp* and *attribute* uniquely identify the values in the fields (Refer Example 2.4)

Following are features of a BigTable:

1. Massively scalable NoSQL. BigTable scales up to lOOs of petabytes.
2. Integrates easily with Hadoop and Hadoop compatible systems.
3. Compatibility with MapReduce, HBase APIs which are open-source Big Data platforms.
4. Key for a field uses not only row_ID and Column_ID (for example, ACVM_ID and KitKat in Example 3.6) but also timestamp and attributes. Values are ordered bytes. Therefore, multiple versions of values may be

present in the BigTable.

5. Handles million of operations per second.
6. Handle large workloads with low latency and high throughput
7. Consistent low latency and high throughput
8. APIs include security and permissions
9. BigTable, being Google's cloud service, has global availability and its service is seamless.

The following example explains the use of rowID, ColumID and Column attributes in BigTable formats.

EXAMPLE 3.7

Consider Example 3.6. Consider column fields which have keys to access a field not only by row ID and Column ID but also include the timestamp and attributes in a row. Show the column-keys for accessing column fields of a column.

SOLUTION

Table 3.4 gives keys for each day's sales of KitKat chocolates at ACVMs. First row-headings are the column-keys.

Table 3.4 Each day's sales of KitKat chocolates at ACVMs

Column-keys

3.3.3.3 RC File Format

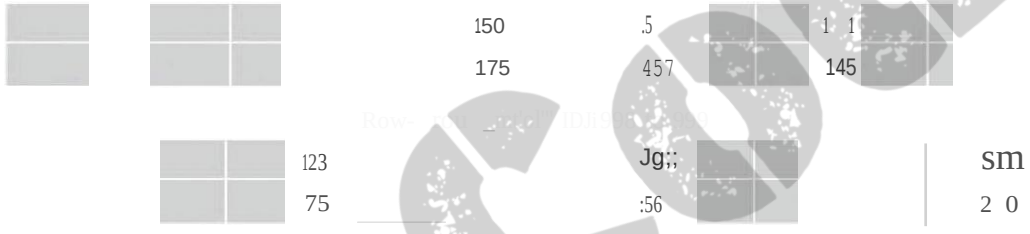
Hive uses Record Columnar (RC) file-format records for querying. RC is the best choice for intermediate tables for fast column-family store in HDFS with Hive. Serializability of RC table column data is the advantage. RC file is DeSerializable into column data. A table such as that shown in Example 3.6 can be partitioned into row groups. Values at each column of a row group store as the RC record.

The RC file records store data of a column in the row group *(Serializability* means query or transaction executable by series of instructions such that execution ensures correct results).

The following example explains the use of row groups in the RC file format for column of a row group:

EXAMPLE 3.8

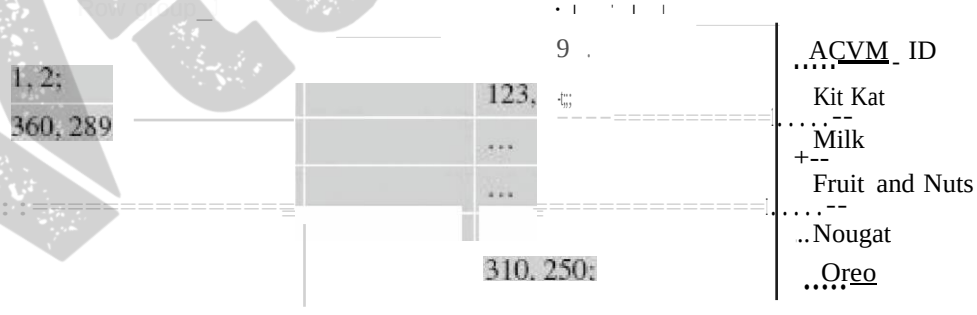
Consider Example 3.6. Practically, row groups have millions of rows and in• memory between 10 MB and 1 GB. Assume two row groups of just two rows each. Consider the following values given in Table 3.3.



Make a file in RC format.

SOLUTION

The values in each column are the records in file for each row group. Each row-group data is like a column of records which stores in the RC file.



RC file for row group_l will consists of records 1, 2; 360, 289; ..., 222, 317; on serialization of column records. RC file for row group _m will consists of

998, 999; 123, 75; ..., 310, 250;

3.3.3.4 ORC *File Format*

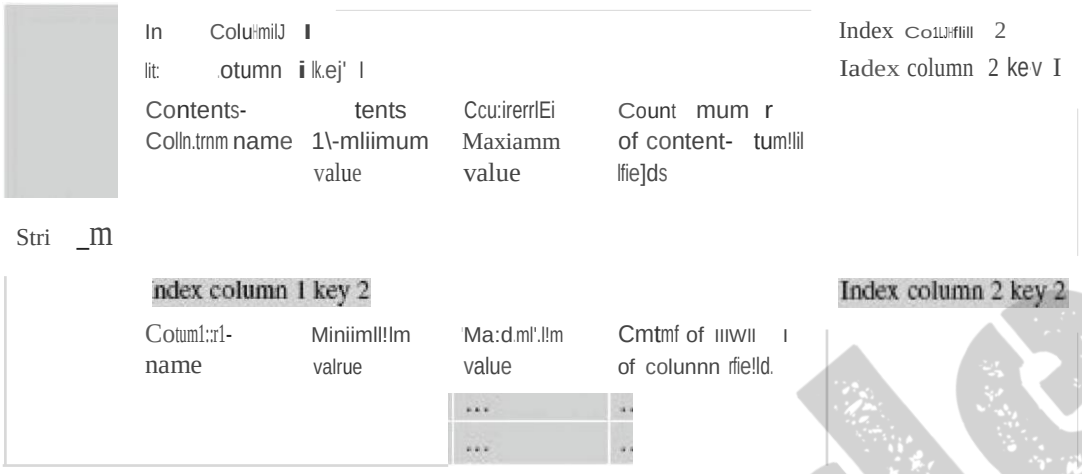
An ORC (Optimized Row Columnar) file consists of row-group data called stripes. ORC enables concurrent reads of the same file using separate RecordReaders. Metadata store uses Protocol Buffers for addition and removal of fields.1

ORC is an intelligent Big Data file format for HDFS and Hive.[2] An ORC file stores a collections of rows as a row-group. Each row-group data store in columnar format. This enables parallel processing of multiple row-groups in an HDFS cluster.

An ORC file consists of a stripe the size of the file is by default 256 MB. Stripe consists of indexing (mapping) data in 8 columns, row-group columns data (contents) and stripe footer (metadata). An ORC has two sets of columns data instead of one column data in RC. One column is for each map or list size and other values which enable a query to decide skipping or reading of the mapped columns. A mapped column has contents required by the query. The columnar layout in each ORC file thus, optimizes for compression and enables skipping of data in columns. This reduces read and decompression load.

Lightweight indexing is an ORC feature. Those blocks of rows which do not match a query skip as they do not map on using indices data at metadata. Each index includes the aggregated values of minimum, maximum, sum and count using aggregation functions on the content columns. Therefore, contents• column key for accessing the contents from a column consists of combination of row-group key, column mapping key, min, max, count (number) of column fields of the contents column. Table 3.5 gives the keys used to access or skip a contents column during querying. The keys are Stripe_ID, Index-column key, and contents-column name, min, max and count.

Table 3.5 Keys to access or skip a content column in ORC file format



Consider Example 3.6. ORC key to access during a query consist of not only column head 'KitKat' (Table 3.3) but also column minimum and maximum sales on an ACVM, count of number of fields in values 'KitKat'. Analytics operations frequently need these values. Ready availability of these values from the index data itself improves the throughput in Big Data HDFS environment. These values do not need to compute again and again using aggregation functions, such as min, max and count.

An ORC thus, optimizes for reading serially the column fields in HDFS environment. The throughput increases due to skipping and reading of the required fields at contents-column key. Reading less number of ORC file content-columns reduces the workload on the NameNode.

3.3.3.5 Parquet File Formats

Parquet is nested hierarchical columnar-storage concept. Nesting sequence is the table, row group, column chunk and chunk page. Apache Parquet file is columnar-family store file. Apache Spark SQL executes user defined functions (UDFs) which query the Parquet file columns (Section 5.2.1.3). A programmer writes the codes for an UDF and creates the processing function for big long queries.

A Parquet file uses an HDFS block. The block stores the file for processing queries on Big Data. The file compulsorily consists of metadata, though the file need not consist of data.

The Parquet file consists of row groups. A row-group columns data process in-

memory after data cache and buffer at the memory from the disk. Each row group has a number of columns. A row group has Ncol columns, and row group consists of Ncol column chunks. This means each column chunk consists of values saved in each column of each row group.

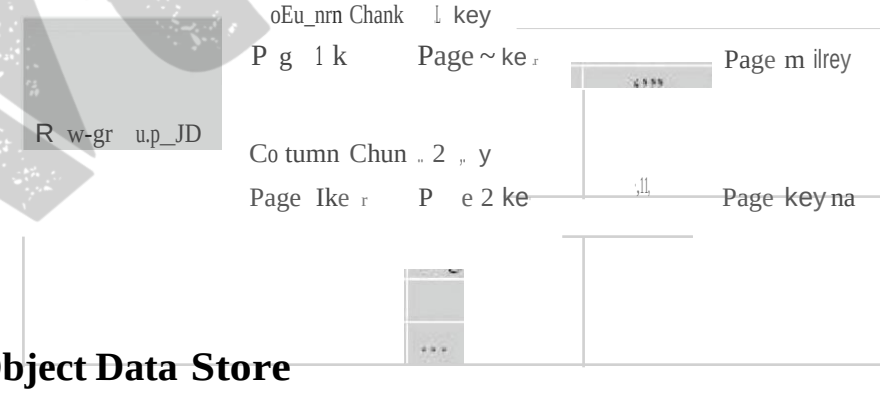
A column chunk can be divided into pages and thus, consists of one or more pages. The column chunk consists of a number of interleaved pages, Npg. A page is a conceptualized unit which can be compressed or encoded together at an instance. The unit is minimum portion of a chunk which is read at an instance for in-memory analytics.

An ORC array <int> has two columns, one for array size and the other for contents. Parquet format file does not consist of extra column per nesting level. Similarly, ORC has two columns, one is for each Map, List size, min, max and the second is for the contents. Parquet format file does not consist of extra column per nesting level, just one column per leaf in the schema.

[Parquet in English means 'a floor covering made of small rectangular wooden blocks (tiles) fitted together in a pattern. Similarly, Parquet objects have pages as the tiles. Pages build a column chunk. Column chunks build a row group. Row groups build the table. A page is like a tile consisting of column fields. The values read or write at an instance or used for encoding or compression. The values are not read separately from a page.]

Table 3.6 gives the keys used to access or skip the contents page. Three keys are: (i) row-group _ID, (ii) column-chunk key and (iii) page key.

Table 3.6 Combination of keys for content page in the Parquet file format



3.3.4 Object Data Store

An object store refers to a repository which stores the:

1. Objects (such as files, images, documents, folders, and business reports)
2. System metadata which provides information such as filename, creation_date, last_modified, language_used (such as Java, C, C#, C++, Smalltalk, Python), access permissions, supported query languages)
3. Custom metadata which provides information, such as subject, category, sharing permissions.

Metadata enables the gathering of metrics of objects, searches, finds the contents and specifies the objects in an object data-store tree. Metadata finds the relationships among the objects, maps the object relations and trends. Object Store metadata interfaces with the Big Data. API first mines the metadata to enable mining of the trends and analytics. The metadata defines classes and properties of the objects. Each Object Store may consist of a database. Document content can be stored in either the object store database storage area or in a file storage area. A single file domain may contain multiple Object Stores.

Data definition and manipulation, DB schema design, database browsing, DB administration, application compilation and debugging use a programming language.

Eleven Functions SupportingAPis An Object data store consists of functions supporting APis for: (i) scalability, (ii) indexing, (iii) large collections, (iv) querying language, processing and optimization (s), (v) Transactions, (vi) data replication for high availability, data distribution model, data integration (such as with relational database, XML, custom code), (vii) schema evolution, (viii) persistency, (ix) persistent object life cycle, (x) adding modules and (xi) locking and caching strategy.

Object Store may support versioning for collaboration. Object Store can be created using IBM 'Content Platform Engine'. Creation needs installing and configuring the engine (Engine is software which drives forward.). Console of the engine makes creation of process easy. Amazon 53 and Microsoft Azure BLOB support the Object Store.

Amazon S3 (Simple Storage Service) 53 refers to Amazon web service on the cloud named 53. The 53 provides the Object Store. The Object Store differs from the block and file-based cloud storage. Objects along with their metadata store

for each object store as the files. 53 assigns an ID number for each stored object. The service has two storage classes: Standard and infrequent access. Interfaces for 53 service are REST, SOAP and Bit Torrent. 53 uses include web hosting, image hosting and storage for backup systems. 53 is scalable storage infrastructure, same as used in Amazon e-commerce service. 53 may store trillions of objects.

The following example lists Object Store development platforms:

EXAMPLE 3.9

List the functions of Minio, Riak, VERSANT Object Database (VOD), GEMSTONE, Amazon 53 and Microsoft Azure BLOB that support using Object Store APis.

SOLUTION

1. An open-source multi-clouds object storage server is Minic, which is API compatible with Amazon 53 API and number of widely used public and private clouds. Compatibility enables data export to 53 and usages of APis.
2. Riak CS (Cloud Storage) is object storage management software on top of Riak. It models on open-source distributed-database which is Amazon• compliant. This means database exports to 53 and use 53 APis.
3. VOD consists of 11 functions supporting APis listed above. VOD enables use by multiple concurrent users. VOD supports cross-platform operating systems (OSs), such as Linux, Windows NT, AIX, HP-UX and Solaris (both 32 and 64 bits for all platforms).
4. GEMSTONE Object DB APis development language is SmallTalk. The platform supports in-memory DBs, object-oriented processing and distributed caches. GEMSTONE provides cross platform support, OSs AIX, Linux, MacOS and Solaris.

3.3.4.1 Object Relational Mapping

The following example explains object relational mapping.

EXAMPLE 3.10

How does an HTML object and XML based web service relate with tabular data stores?

SOLUTION

Figure 3. 7 shows the object relational mapping of HTML document and XML web services store with a tabular data store.

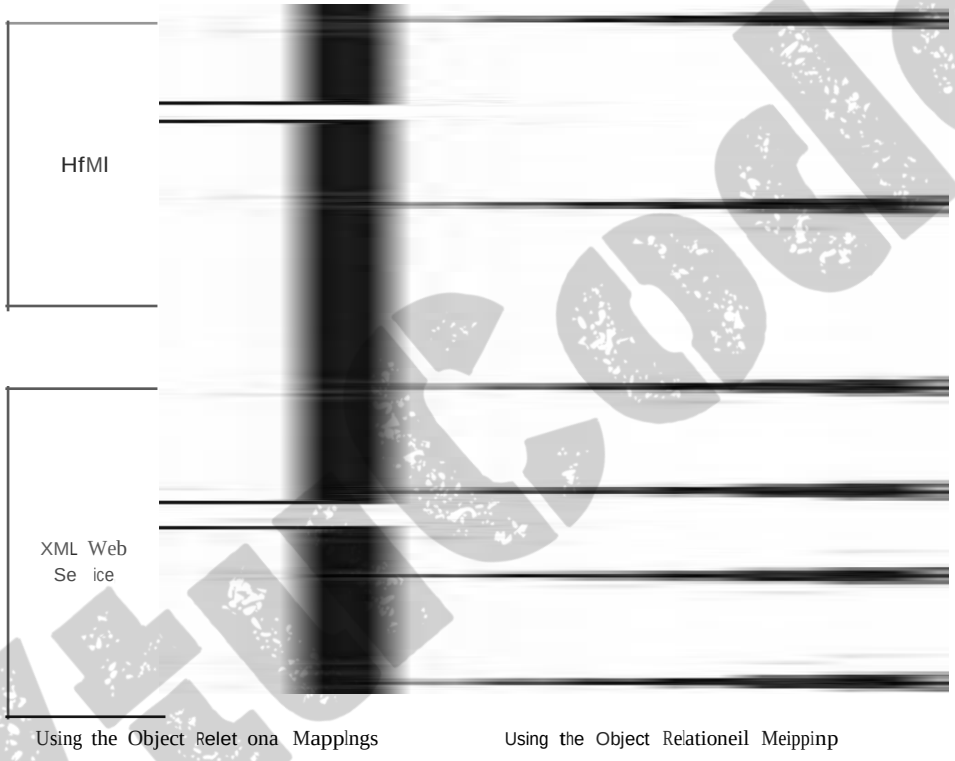


Figure 3.7 HTML document and XML web services

3.3.5 Graph Database

One way to implement a data store is to use graph database. A characteristic of graph is high flexibility. Any number of nodes and any number of edges can be added to expand a graph. The complexity is high and the performance is variable with scalability. Data store as series of interconnected nodes. Graph with data nodes interconnected provides one of the best database system when

relationships and relationship types have critical values.

Data Store focuses on modeling *interconnected* structure of data. Data stores based on graph theory relation G = (E, V), where E is set of edges e_1, e_2, ••• and V is set of vertices, v_1, v_2, •••, v_n.

Nodes represent entities or objects. Edges encode relationships between nodes. Some operations become simpler to perform using graph models. Examples of graph model usages are social networks of connected people. The connections to related persons become easier to model when using the graph model.

The following example explains the graph database application in describing entities relationships and relationship types.

EXAMPLE 3.11

Let us assume a car company represents a node entity, which has two connected nodes comprising two
model entities, namely Hexa and Zest. Draw graph with directed lines, joining the car company with two entities. (i) How do four directed lines relate to four weeks and two directed lines? One directed line corresponds to a car model. Only directed line corresponds to weekly total sales. (ii) How will the yearly sales compute? (iii) Show the path traversals for computations exhibit BASE properties.

SOLUTION

(i) Figure 3.8 shows section of a graph database for the sales of two car models.

Figure 3.8 Section of the graph database for car-model sales

(ii) The yearly sales compute by path traversals from nodes for weekly sales to yearly sales data.

(iv) The path traversals exhibit BASE properties because during the intermediate paths, consistency is not maintained. Eventually when all the path traversals complete, the data becomes consistent.

Graph databases enable fast network searches. Graph uses linked datasets, such as social media data. Data store uses graphs with nodes and edges connecting each other through relations, associations and properties.

Querying for data uses graph traversal along the paths. Traversal may use

single-step, path expressions or full recursion. A relationship represents key. A node possesses property including ID. An edge may have a label which may specify a role.

Characteristics of graph databases are:

1. Use specialized query languages, such as RDF uses SPARQL
2. Create a database system which models the data in a completely different way than the key-values, document, columnar and object data store models.
3. Can have hyper-edges. A hyper-edge is a set of vertices of a hypergraph. A hypergraph is a generalization of a graph in which an edge can join any number of vertices (not only the neighbouring vertices).
4. Consists of a collection of small data size records, which have complex interactions between graph-nodes and hypergraph nodes. Nodes represent the entities or objects. Nodes use Joins. Node identification can use URI or other tree-based structure. The edge encodes a relationship between the nodes.

When a new relationship adds in RDBMS, then the schema changes. The data need transfer from one field to another. The task of adding relations in graph database is simpler. The nodes assign internal identifiers to the nodes and use these identifiers to join the network. Traversing the joins or relationships is fast in graph databases. It is due to the simpler form of graph nodes. The graph data may be kept in RAM only. The relationship between nodes is consistent in a graph store.

Graph databases have poor scalability. They are difficult to scale out on multiple servers. This is due to the close connectivity feature of each node in the graph. Data can be replicated on multiple servers to enhance read and the query processing performance. Write operations to multiple servers and graph queries that span multiple nodes, can be complex to implement.

Typical uses of graph databases are: (i) link analysis, (ii) friend of friend queries, (iii) Rules and inference, (iv) rule induction and (v) Pattern matching. Link analysis is needed to perform searches and look for patterns and relationships in situations, such as social networking, telephone, or email

records (Sections 9.4 and 9.5). Rules and inference are used to run queries on complex structures such as class libraries, taxonomies and rule-based systems.

Examples of graph DBs are Neo4J, AllegroGraph, HyperGraph, Infinite Graph, Titan and FlockDB.Neo4J graph database enable easy usages by Java developers. Neo4J can be designed fully ACID rules compliant. Design consists of adding additional path traversal in between the transactions such that data consistency is maintained and the transactions exhibit ACID properties.

Spark provides a simple and expressive programming model that includes supports to a wide range of applications, including graph computation. Chapter 8 describes Graph Databases.

3.3.6 Variations of NoSQL Architectural Patterns

Six data architectures are SQL-table,key-value pairs, in-memory column-family, document, graph and object. Selected architecture may need variations due to business requirements. Business requirements are ease of using an architecture and long-term competitive advantage. The following example explains the requirements for the database of students of a University that offers multiple courses in their various academic programmes for several years:

EXAMPLE 3.12

List the selection requirements for the database of University students in successive years. The University runs various Under Graduate and Post Graduate programmes. Students are registered to Multiple courses in a programme.

SOLUTION

Following are the selection requirements:

1. Scalability: Since the University archives the data for several years, data store should be scalable.
2. Search ability: Search of required information needs to be fast.
3. Quarrying ability: All applications need to query the data. Query retrieves the required data among the Big Data of several years.
4. Security: Database needs security and fault tolerance.

5. Affordability: Open source is a requirement.
6. Interoperability: Needs ease in search from different platforms. Search from any computer operating system, such as Windows, Mac, Linux, Android and iOS should be feasible.
7. Importability: Database needs to import data from other platforms, such as import of slides, video lectures, tutorials, e-books, webinars should be facilitated in store.
8. Transformability: Queries may be written in one language and may require transformation to another language, such as HTML.

Analysis of the above requirements suggests the document architecture pattern will be more suitable.

Kelly-McCreary, co-founder of 'NoSQL Now' suggested that when selecting a NoSQL-pattern, the pattern may need change and require variation to another pattern(s). Some reasons for this are:

1. Focus changing from performance to scalability
2. Changing from modifiability to agility
3. Greater emphasis on Big Data, affording capacity, availability of support, ability for searching and monitoring the actions

Steps for selecting a NoSQL data architectural pattern can be as follows:

1. Select an architecture
2. Perform a use-case driven difficulty analysis for each of the six architectural patterns. Difficulties may be low, medium or high in the following processes: (i) ingestion, (ii) validation of structure and its fields, (iii) updating process using batch or record by record approach, (iv) searching process using full text or by changing the sorting order, and (v) export the reports or application results in HTML, XML or *JSON.*
3. Estimate the total efforts for each architecture for all business requirements.

Process the choice of architecture using trade-off. For example, between the

MongoDBdocument data store and Cassandra column-family data store.

Self-Assessment Exercise linked to LO 3.2

1. Compare traditional relational model and key-value pairs model.
2. When will you use the document data store?
3. Why is metadata must in a NoSQL Data Store?
4. How do interactions among graph nodes and hypergraph nodes differentiate?
5. List and compare the features of BigTable, RC, ORC and Parquet data stores.
6. What are the characteristics of the object data store model?
7. Data architecture pattern can be selected from among the six architectures, namely relational SQL table, CLAP-suitable in-memory column, key-value pairs, column-family, document and graph DBs. Explain with an example, how and when each of these is used.

3.4 | NOSQL TO MANAGE BIG DATA

LO 1.6

NoSQL dab stm@ m:aft'agemeirrrtt. applicatiHs and hadling probleffilsiliil Bi~ Data

The following subsections describe how to use a NoSQL data store to manage Big Data.

3.4.1 Using NoSQLto Manage Big Data

NoSQL (i) limits the support for Join queries, supports sparse matrix like columnar-family, (ii) characteristics of easy creation and high processing speed, scalability and storability of much higher magnitude of data (terabytes and petabytes).

NoSQL sacrifices the support of ACID properties, and instead supports CAP and BASE properties (Sections 3.2.1.1 and 3.2.3). NoSQL data processing scales horizontally as well vertically.

3.4.1.1 NoSQL Solutions for Big Data

Big Data solution needs scalable storage of terabytes and petabytes, dropping of support for database Joins, and storing data differently on several distributed servers (data nodes) together as a cluster. A solution, such as CouchDB, DynamoDB,MongoDBor Cassandra follow CAP theorem (with compromising the consistency factor) to make transactions faster and easier to scale. A solution must also be partitioning tolerant.

Characteristics of Big Data NoSQL solution are:

1. *High and easy scalability:* NoSQL data stores are designed to expand horizontally. Horizontal scaling means that scaling out by adding more machines as data nodes (servers) into the pool of resources (processing, memory, network connections). The design scales out using multi-utility cloud services.
2. *Support to replication:* Multiple copies of data store across multiple nodes of a cluster. This ensures high availability, partition, reliability and fault tolerance.
3. *Distributable:* Big Data solutions permit sharding and distributing of shards on multiple clusters which enhances performance and throughput.
4. *Usages of NoSQL servers* which are less expensive. NoSQL data stores require less management efforts. It supports many features like automatic repair, easier data distribution and simpler data models that makes database administrator (DBA) and tuning requirements less stringent.
5. *Usages of open-source* tools: NoSQL data stores are cheap and open source. Database implementation is easy and typically uses cheap servers to manage the exploding data and transaction while RDBMS databases are expensive and use big servers and storage systems. So, cost per gigabyte data store and processing of that data can be many times less than the cost ofRDBMS.
6. *Support to schema-less data model:* NoSQL data store is schema less, so data can be inserted in a NoSQL data store without any predefined schema. So, the format or data model can be changed any time, without disruption of

application. Managing the changes is a difficult problem in SQL.

7. *Support to integrated caching:* NoSQL data store support the caching in system memory. That increases output performance. SQL database needs a separate infrastructure for that.
8. *No inflexibility* unlike the SQL/RDBMS, NoSQL DBs are flexible (not rigid) and have no structured way of storing and manipulating data. SQL stores in the form of tables consisting of rows and columns. NoSQL data stores have flexibility in following ACID rules.

3.4.1.2 Types of Big Data Problems

Big Data problems arise due to limitations of NoSQL and other DBs. The following types of problems are faced using Big Data solutions.

1. Big Data need the scalable storage and use of distributed servers together as a cluster. Therefore, the solutions must drop support for the database Joins
2. NoSQL database is open source and that is its greatest strength but at the same time its greatest weakness also because there are not many defined standards for NoSQL data stores. Hence, no two NoSQL data stores are equal. For example:

(i) No stored procedures in MongoDB(NoSQL data store)

(ii) GUI mode tools to access the data store are not available in the market

(iii) Lack of standardization

(iv) NoSQL data stores sacrifice ACID compliancy for flexibility and processing speed.

A comparison of NoSQL with SQL/RDBMS shows that NoSQL data model are schema-less, no pre-defined schema, multiple data architecture patterns, complex to implement vertical scalability, variable consistency and very week adherence to ACID rules. Table 3.7 gives a comparison.

Table 3.7 Comparison of NoSQL with SQL/RDBMS

Features	NoSQL Data store	SQL/RDBMS

Model	Schema-less model	Relational
Schema	Dynamic schema	Predefined
Types of data architecture patterns	Key/value based, column-family based, document based, graph based, object based	Table based
Scalable	Horizontally scalable	Vertically scalable
Use ofSQL	No	Yes
Dataset size preference	Prefers large datasets	Large dataset not preferred
Consistency	Variable	Strong
Vendor support	Open source	Strong
ACID properties	May not support, instead follows Brewer's CAP theorem or BASE properties	Strictly follows

Self-Assessment Exercise linked to LO 3.3

1. Why does Big Data need scalable storage and uses distributed servers together as a cluster?
2. Why does Big Data solution possess CAP or BASE and may drop support for ACID properties?
3. Why does a Big Data solution drop support for the database Joins?
4. Compare NoSQL data stores with SQL databases.

3.5 | SHARED-NOTHING ARCHITECTURE FOR BIG DATA TASKS

The columns of two tables relate by a relationship. A relational algebraic equation specifies the relation. Keys share between two or more SQL tables in RDBMS. Shared nothing (SN) is a cluster architecture. A node does not share data with any other node.

Big Data store consists of SN architecture. Big Data store, therefore, easily partitions into shards. A partition processes the different queries on data of the different users at each node independently. Thus, data processes run in parallel at the nodes. A node maintains a copy of running-process data. A coordination protocol controls the processing at all SN nodes. An SN architecture optimizes massive parallel data processing.

LO 3.4

9h 21red-1r1otlriii ng arclmit"ed:,u re. choosing a distri buti,on m odel, rn :;:ist@~• $lalVif! versl!Jls pee:r-to-Jleer. and knO'Wledge of ifm!u wa:y5 lby 'llrrtlch NoSQL lhandlestliile. Bill] Daita pro1Dlerlfs

Data of different data stores partition among the number of nodes (assigning different computers to deal with different users or queries). Processing may require every node to maintain its own copy of the application's data, using a coordination protocol. Examples are using the partitioning and processing are Hadoop, Flink and Spark.

The features of SN architecture are as follows:

1. *Independence:* Each node with no memory sharing; thus possesses computational self-sufficiency
2. *Self-Healing:* A link failure causes creation of another link
3. *Each node functioning as a shard:* Each node stores a shard (a partition of large DBs)
4. No network contention.

3.5.1 Choosing the Distribution Models

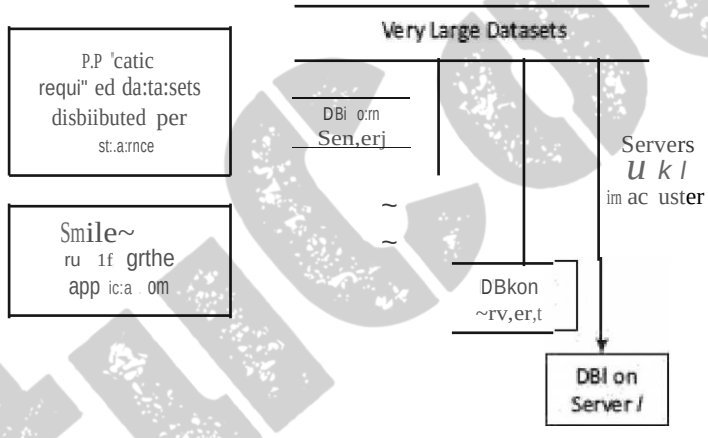
Big Data requires distribution on multiple data nodes at clusters. Distributed software components give advantage of parallel processing; thus providing horizontal scalability. Distribution gives (i) ability to handle large-sized data, and (ii) processing of many read and write operations simultaneously in an application. A resource manager manages, allocates, and schedules the resources of each processor, memory and network connection. Distribution increases the availability when a network slows or link fails. Four models for distribution of the data store are given below:

3.5.1.1 Single Server Model

Simplest distribution option for NoSQL data store and access is Single Server Distribution (SSD) of an application. A graph database processes the relationships between nodes at a server. The SSD model suits well for graph DBs. Aggregates of datasets may be key-value, column-family or BigTable data stores which require sequential processing. These data stores also use the SSD model. An application executes the data sequentially on a single server. Figure 3.9(a) shows the SSD model. Process and datasets distribute to a single server which runs the application.

3.5.1.2 Sharding Very Large Databases

Figure 3.9(b) shows sharding of very large datasets into four divisions, each running the application on four i,j, *k* and *l* different servers at the cluster. DBi, DBj, DBk and DBl are four shards.



[a]

Figure 3.9 (a) Single server model (b) Shards distributed on four servers in a cluster.

The application programming model in SN architecture is such that an application process runs on multiple shards in parallel. Sharding provides horizontal scalability. A data store may add an auto-sharding feature. The performance improves in the SN. However, in case of a link failure with the application, the application can migrate the shard DB to another node.

3.5.1.3 Master-Slave Distribution Model

A node serves as a master or primary node and the other nodes are slave nodes. Master directs the slaves. Slave nodes data replicate on multiple slave servers in

Master Slave Distribution (MSD) model. When a process updates the master, it updates the slaves also. A process uses the slaves for read operations. Processing performance improves when process runs large datasets distributed onto the slave nodes. Figure 3.10 shows an example of MongoDB. MongoDB database server is *mongod* and the client is *mongo.*

Master-Slave Replication Processing performance decreases due to replication in MSD distribution model. Resilience for read operations is high, which means if in case data is not available from a slave node, then it becomes available from the replicated nodes. Master uses the distinct write and read paths.

Complexity Cluster-based processing has greater complexity than the other architectures. Consistency can also be affected in case of problem of significant time taken for updating.

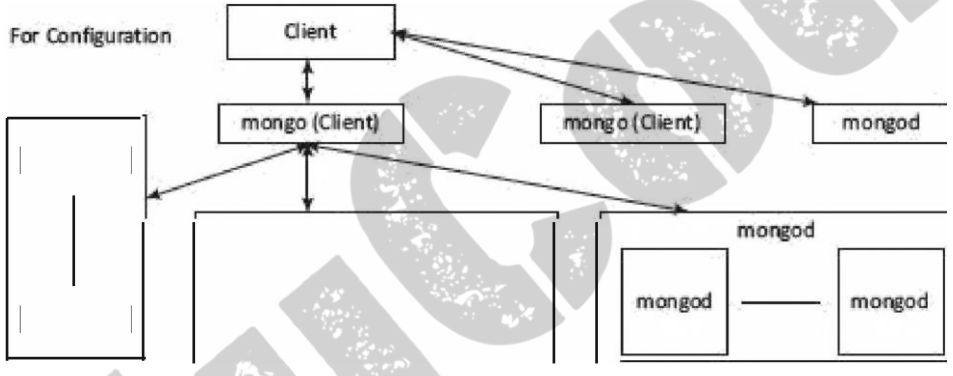


Figure 3.10 Master-slave distribution model. Mongo is a client and mongod is the server

3.5.1.4 Peer-to-Peer Distribution Model

Peer-to-Peer distribution (PPD) model and replication show the following characteristics: (1) All replication nodes accept read request and send the responses. (2) All replicas function equally. (3) Node failures do not cause loss of write capability, as other replicated node responds.

Cassandra adopts the PPD model. The data distributes among all the nodes in a cluster.

Performance can further be enhanced by adding the nodes. Since nodes read and write both, a replicated node also has updated data. Therefore, the biggest advantage in the model is consistency. When a write is on different nodes, then

write inconsistency occurs.

Figure 3.11 shows the PPD model.

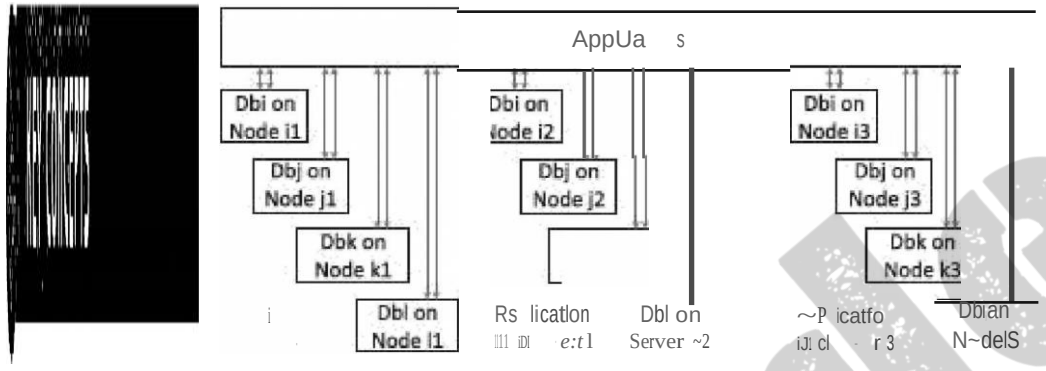


Figure3.11 Shards replicating on the nodes, which does read and write operations both

3.5.1.5 ChoosingMaster-Slave versus *Peer-to-Peer*

Master-slave replication provides greater scalability for read operations. Replication provides resilience during the read. Master does not provide resilience for writes. Peer-to-peer replication provides resilience for read and writes both.

Sharing Combining with Replication Master-slave and sharding creates multiple masters. However, for each data a single master exists. Configuration assigns a master to a group of datasets. Peer-to-peer and sharding use same strategy for the column-family data stores. The shards replicate on the nodes, which does read and write operations both.

3.5.2 Ways of HandlingBig Data Problems

Figure 3.12 shows four ways for handling Big Data problems.

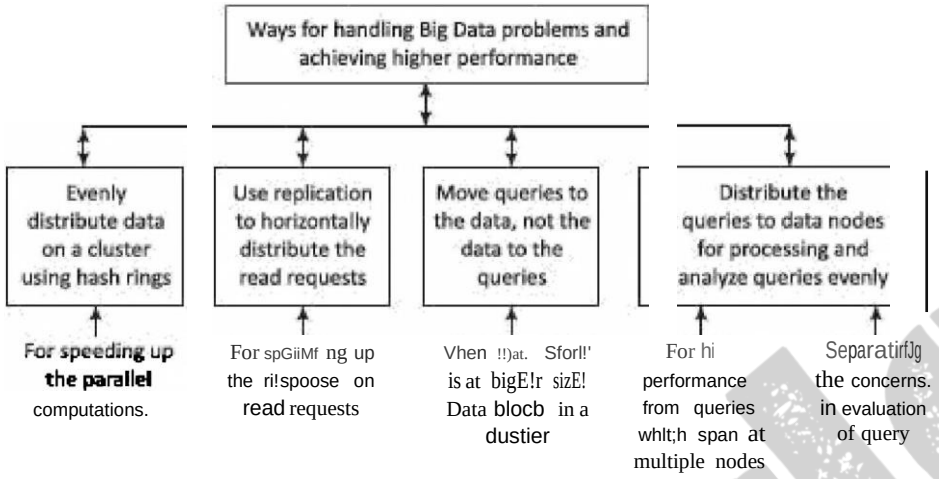


Figure 3.12 Four ways for handling big data problems

Following are the ways:

1. *Evenly distribute the data on a cluster using the hash rings:* Consistent hashing refers to a process where the datasets in a collection distribute using a hashing algorithm which generates the pointer for a collection. Using only the hash of Collection_ID, a Big Data solution client node determines the data location in the cluster. Hash Ring refers to a map of hashes with locations. The client, resource manager or scripts use the hash ring for data searches and Big Data solutions. The ring enables the consistent assignment and usages of the dataset to a specific processor.
2. *Use replication to horizontally distribute* the client read-requests: Replication means creating backup copies of data in real time. Many Big Data clusters use replication to make the failure-proof retrieval of data in a distributed environment. Using replication enables horizontal scaling out of the client requests.
3. *Moving queries to the data, not the data to the queries:* Most NoSQL data stores use cloud utility services (Large graph databases may use enterprise servers). Moving client node queries to the data is efficient as well as a requirement in Big Data solutions.
4. *Queries distribution to multiple nodes:* Client queries for the DBs analyze at

the analyzers, which evenly distribute the queries to data nodes/ replica nodes. High performance query processing requires usages of multiple nodes. The query execution takes place separately from the query evaluation (The evaluation means interpreting the query and generating a plan for its execution sequence).

Self-Assessment Exercise linked to LO 3.4

1. List pros and cons of distribution using sharding.
2. List characteristics of master-slave distribution model.
3. List the benefits of peer-to-peer nodes data distribution model.
4. How is a hash ring used in the distribution of Big Data?

3.6 ¦ MONGODB DATABASE

Learning Outcomes

cmgoliJJB am'bases am:d queryrn rTlmtillilds

MongoDBis an open source DBMS. MongoDBprograms *create* and *manage* databases. MongoDB manages the collection and document data store. MongoDB functions do querying and accessing the required information. The functions include viewing, querying, changing, visualizing and running the transactions. Changing includes updating, inserting, appending or deleting.

MongoDB is (i) non-relational, (ii) NoSQL, (iii) distributed, (iv) open source, (v) document based,
(vi) cross-platform, (vii) Scalable, (viii) flexible data model, (ix) Indexed, (x) multi-master (Section 3.5.1.3), and (xi) fault tolerant. Document data store in)SON-like documents. The data store uses the dynamic schemas.

The typical MongoDB applications are content management and delivery systems, mobile applications, user data management, gaming, e-commerce, analytics, archiving and logging.

Features Following are features of MongoDB:

1. *MongoDB data store* is a physical container for collections. Each DB gets its own set of files on the file system. A number of DBs can run on a single MongoDB server. DB is default DB in MongoDB that stores within a data folder. The database server of MongoDBis *mongod* and the client is *mongo.*
2. *Collection* stores a number of MongoDB documents. It is analogous to a table of RDBMS. A collection exists within a single DB to achieve a single purpose. Collections may store documents that do not have the same fields. Thus, documents of the collection are schema-less. Thus, it is possible to store documents of varying structures in a collection. Practically, in an RDBMS, it is required to define a column and its data type, but does not need them while working with the MongoDB.
3. Document *model* is well defined. Structure of document is clear, Document is the unit of storing data in a MongoDBdatabase. Documents are analogous to the records of RDBMS table. Insert, update and delete operations can be performed on a collection. Document useJSON (lavascript Object Notation) approach for storing data. JSON is a lightweight, self-describing format used to interchange data between various applications. JSON data basically has key-value pairs. Documents have dynamic schema.
4. MongoDBis a document data store in which one collection holds different documents. Data store in the form of JSON-style documents. Number of fields, content and size of the document can differ from one document to another.
5. *Storing of data* is flexible, and data store consists of JSON-like documents. This implies that the fields can vary from document to document and data structure can be changed over time; JSON has a standard structure, and scalable way of describing hierarchical data (Example 3.3(ii)).
6. *Storing of documents* on disk is in BSON serialization format. BSON is a binary representation of *JSON* documents. The mongo JavaScript shell and MongoDB language drivers perform translation between BSON and language-specific document representation.
7. *Querying, indexing, and real time aggregation* allows accessing and analyzing

the data efficiently.

8. *Deep query-ability-Supports* dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
9. No complex Joins.
10. *Distributed DB* makes availability high, and provides horizontal scalability.
11. *Indexes on any field* in a collection of documents: Users can create indexes on any field in a document. Indices support queries and operations. By default, MongoDB creates an index on the _id field of every collection.
12. *Atomic operations on a single document* can be performed even though support of multi-document transactions is not present. The operations are alternate to ACID transaction requirement of a relational DB.
13. *Fast-in-place updates:* The DB does not have to allocate new memory location and write a full new copy of the object in case of data updates. This results into high performance for frequent update use cases. For example, incrementing a counter operation does not fetch the document from the server. Here, the increment operation can simply be set.
14. *No configurable cache:* MongoDB uses all free memory on the system automatically by way of memory-mapped files (The operating systems use the similar approach with their file system caches). The most recently used data is kept in RAM. If indexes are created for queries and the working dataset fits in RAM, MongoDB serves all queries from memory.
15. *Conversion/mapping* of application objects to data store objects not needed

Dynamic Schema Dynamic schema implies that documents in the same collection do not need to have the same set of fields or structure. Also, the similar fields in a document may contain different types of data. Table 3.8 gives the comparison with RDBMS.

Table 3.8 Comparison ofRDBMS and MongoDB databases

RDBMS	MongoDB
Database	Data store

Table	Collection
Column	Key
Value	Value
Records / Rows / Tuple	Document/ Object
Joins	Embedded Documents
Index	Index
Primary key	Primary key (_id) is default key provided by MongoDB itself

Any relational DB has a typical schema design that shows the number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

Replication Replication ensures high availability in Big Data. Presence of multiple copies increases on different database servers. This makes DBs fault• tolerant against any database server failure. Multiple copies of data certainly help in localizing the data and ensure availability of data in a distributed system environment.

MongoDBreplicates with the help of a replica set. A replica set in MongoDBis a group of mongod (MongoDb server) processes that store the same dataset. Replica sets provide redundancy but high availability. A replica set usually has minimum three nodes. Any one out of them is called primary. The primary node receives all the write operations. All the other nodes are termed as secondary. The data replicates from primary to secondary nodes. A new primary node can be chosen among the secondary nodes at the time of automatic failover or maintenance. The failed node when recovered can join the replica set as secondary node again. Replica set starts a mongod instance by specifying -replSet option before running these commands from mongo (MongoDb Client). Table 3.9 gives the commands used for replication *(Recoverability* means even on occurrences of failures; the transactions ensure consistency).

Table 3.9 MongoDBClient commands related to replica set

Commands	Description
rs.initiate()	To initiate a new replica set

rs.conf ()	To check the replica set configuration
rs.status()	To check the status of a replica set
rs.add()	To add members to a replica set

Figure 3.13 shows a replicated dataset after creating three secondary members from a primary member.

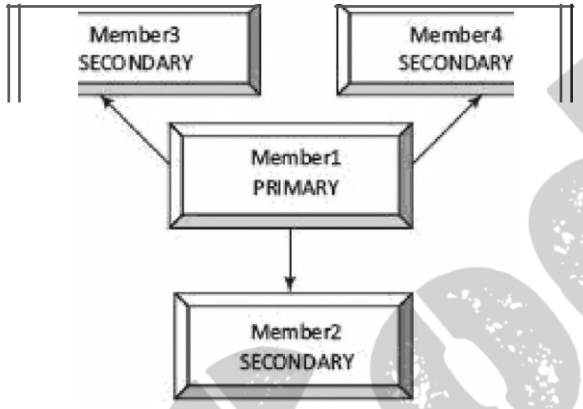


Figure 3.13 Replicated set on creating secondary members

Auto-sharding Sharding is a method for distributing data across multiple machines in a distributed application environment. MongoDBuses sharding to provide services to Big Data applications.

A single machine may not be adequate to store the data. When the data size increases, do not provide data retrieval operation. Vertical scaling by increasing the resources of a single machine is quite expensive. Thus, horizontal scaling of the data can be achieved using sharding mechanism where more database servers can be added to support data growth and the demands of more read and write operations.

Sharding automatically balances the data and load across various servers. Sharding provides additional write capability by distributing the write load over a number of mongod (MongoDBServer) instances.
(Figure 3.10) Basically, it splits the dataset and distributes them across multiple DBs, called shards on the different servers. Each shard is an independent DB. The whole collection of shards forms a single logical DB. If a DB has a 1 terabyte dataset distributed amongst 20 shards, then each shard contains only 50 Giga Byte of data.

A shard stores lesser data than the actual data and handles lesser number of operations in a single instance. For example, to insert data into a collection, the application needs to access only the shard that contains the specified collection. A cluster can thus easily increase its capacity horizontally.

Data Types Table 3.10 gives data types which MongoDBdocuments support.

Table 3.10 Data types which MongoDBdocuments support

Type	Description
Double	Represents a float value.
String	UTF-8 format string.
Object	Represents an embedded document.
Array	Sets or lists of values.
Binary data	String of arbitrary bytes to store images, binaries.
Object id	Objectlds (MongoDB document identifier, equivalent to a primary key) are: small, likely unique, fast to generate, and ordered. The value consists of 12-bytes, where the first four bytes are for timestamp that reflects the instance when Objectld creates.
Boolean	Represents logical true or false value.
Date	BSON Date is a 64-bit integer that represents the number of milliseconds since the Unix epoch (Jan 1, 1970).
Null	Represents a null value. A value which is missing or unknown is Null.
Regular Expression	RegExp maps directly to a JavaScript RegExp
32-bit integer	Numbers without decimal points save and return as 32-bit integers.
Timestamp	A special timestamp type for internal MongoDB use and is not associated with the regular date type. Timestamp values are a 64-bit value, where first 32 bits are time, t (seconds since the Unix epoch), and next 32 bits are an incrementing ordinal for operations within a given second.

64-bit integer	Number without a decimal point save and return as 64-bit integer.
Min key	MinKey compare less than all other possible BSON element values, respectively, and exist primarily for internal use.
Max key	MaxKey compares greater than all other possible BSON element values, respectively, and exist primarily for internal use.

Rich Queries and Other DB Functionalities MongoDB offers a rich set of features and functionality compared to those offered in simple key-value stores. They can be comparable to those offered by any RDBMS. MongoDB has a complete query language, highly-functional secondary indexes (including text search and geospatial), and a powerful aggregation framework for data analysis. MongoDBprovides functionalities and features for more diverse data types than a relational DB, and at scale. Table 3.11 gives a comparison of features.

Table 3.11 Comparison of features MongoDBwith respect to RDBMS

Features	RDBMS	MongoDB
Rich Data Model	No	Yes
Dynamic Schema	No	Yes
Typed Data	Yes	Yes
Data Locality	No	Yes
Field Updates	Yes	Yes
Complex Transactions	Yes	No
Auditing	Yes	Yes
Horizontal Scaling	No	Yes

The ability to derive a document-based data model is also a distinct advantage of MongoDB. The method of storing data in the form of BSON (Binary JSON) helps to store the data in a very rich way while can hold arrays and other documents.

MongDB Query Language and Database Commands Table 3.12 gives MongoDB commands for querying the DBs.

Table 3.12 MongoDBquerying commands

Command	Functionality
Mongo	Starts MongoDB; (*mongo is MongoDB client). The default database in MongoDB is test.
db.help()	Runs help. This displays the list of all the commands.
db.stats()	Gets statistics about MongoDB server.
Use <database name)	Creates database
Db	Outputs the names of existing database, if created earlier
Dbs	Gets list of all the databases
db.dropDatabase ()	Drops a database
db.database name.insert()	Creates a collection using insert ()
db.sdatabase name>. find()	Views all documents in a collection
db.edatabase name=.update ()	Updates a document
db.sdatabase name> .remove ()	Deletes a document

Following explains the sample usages of the commands:

To Create database Command use - use command creates a database; For example, Command use lego creates a database named *leqo.* (A sample database is created to demonstrate subsequent queries. The Lego is an international toy brand). Default database in MongoDBis *test.*

To see the existence of database Command db - db command shows that *lego* database is created.

To get list of all the databases Command show dbs - This command shows

the names of all the databases.

To drop database Command db. dropDatabase () - This command drops a database. Run use lego command before the db. dropDatabase () command to drop *lego* Database. If no database is selected, the default database *test* will be dropped.

To create a collection Command insert () -To create a collection, the easiest way is to insert a record (a document consisting of keys (Field names) and Values) into a collection. A new collection will be created, if the collection does not exist. The following statements demonstrate the creation of a collection with three fields (ProductCategory, Productld and ProductName) in the *lego:*

db. lego. insert

```
"'ProductCategory": 11,Ai:rplane",
~Product!dn: 10725,
HProductName.,..: ,..,Lost TempleP
```

To add array in a collection Command insert() - Insert command can also be used to insert multiple documents into a collection at one time.

```
db. lego. insert
(
                    'ProductCatego:ry"  :  'Airplanep,
                    "Product Id"":  1072 5,
                    '"ProductName  :  "Loo t  Temple"
L
{
                    .,,ProductCategory"  :  ,,,Airplane
                    .,,ProductidP:  31047,
                    ProductName  :  "Propeller  Plane""
L
{
                    ~ProductCatego:ry":  ~Airplane
                    'Productld":  31049,
                    'Produc tName' :  '"Twin Spin  Hel i copt e
```

To view all documents in a collection Command db. <database name>. find () - Find command is equivalent to select query of RDBMS. Thus, "select * from lego" can be written as db. lego. find () in MongoDB. MongoDBcreated unique objecteld ("_id") on its own. This is the primary key of the collection. Command db. <database name>. find() .pretty() gives a prettier look.

To update a document Command db. <database name>. update () - Update command is used to change the field value. By default, multi attribute is false. If {multi: true} is not written then it will update only the first document.

To delete a document Command db. <database name>. remove () - Remove command is used to delete the document. The query db. <database name>. remove (("ProdctID": 10725)) removes the document whose productld is 10725.

Self-Assessment Exercise linked to LO 3.5

1. Compare MongoDBand RDBMS?
2. Give example that demonstrates the uses of various data types of MongoDB.

3. List the functions ofMongoDB query language and database commands.

4. How will you consider MongoDB as complete query language, which imbibes highly-functional secondary indices (including text search and geospatial), and provides a powerful aggregation framework for data analysis?

3.7 | CASSANDRA DATABASES

LO 3.6

Cassandra databases, data-models and clients, and integration with Hadoop

Cassandra was developed by Facebook and released by Apache. Cassandra was named after Trojan mythological prophet Cassandra, who had classical allusions to a curse on oracle. Later on, IBM also released the enhancement of Cassandra, as open source version. The open source version includes an IBM Data Engine which processes No SQL data store. The engine has improved throughput when workload of read-operations is intensive.

Cassandra is basically a column family database that stores and handles massive data of any format including structured, semi-structured and unstructured data.

Apache Cassandra DBMS contains a set of programs. They *create* and *manage* databases. Cassandra provides functions (commands) for querying the data and accessing the required information. Functions do the viewing, querying and changing (update, insert or append or delete), visualizing and perform transactions on the DB.

Apache Cassandra has the distributed design of Dynamo. Cassandra is written in Java. Big organizations, such as Facebook, IBM, Twitter, Cisco, Rackspace, eBay, Twitter and Netflix have adopted Cassandra.

Characteristics of Cassandra are (i) open source, (ii) scalable (iii) non• relational (v) NoSQL (iv) Distributed (vi) column based, (vii) decentralized, (viii) fault tolerant and (ix) tuneable consistency.

Features of Cassandra are as follows:

1. Maximizes the number of writes - writes are not very costly (time consuming)
2. Maximizes data duplication
3. Does not support Joins, group by, OR clause and aggregations
4. Uses Classes consisting of ordered keys and semi-structured data storage systems
5. Is fast and easily scalable with write operations spread across the cluster. The cluster does not have a master-node, so any read and write can be handled by any node in the cluster.
6. Is a distributed DBMS designed for handling a high volume of structured data across multiple cloud servers
7. Has peer-to-peer distribution in the system across its nodes, and the data is distributed among all the nodes in a cluster (Section 3.5.1.4).

Data Replication Cassandra stores data on multiple nodes (data replication) and thus has no single point of failure, and ensures availability, a requirement in CAP theorem. Data replication uses a replication strategy. Replication factor determines the total number of replicas placed on different nodes. Cassandra returns the most recent value of the data to the client. If it has detected that some of the nodes responded with a stale value, Cassandra performs a read repair in the background to update the stale values.

Components at Cassandra Table 3.13 gives the components at Cassandra and their description.

Table 3.13 Components of cassandra

Component	Description
Node	Place where data stores for processing
Data Center	Collection of many related nodes
Cluster	Collection of many data centers
Commit log	Used for crash recovery; each write operation written to commit log

Mem-table	Memory resident data structure, after data written in commit log, data write in mem-table temporarily
SSTable	When mem-table reaches a certain threshold, data flush into an SSTable disk file
Bloom filter	Fast and memory-efficient, probabilistic-data structure to find whether an element is present in a set, Bloom filters are accessed after every query.

Scalability Cassandra provides linear scalability which increases the throughput and decreases the response time on increase in the number of nodes at cluster.

Transaction Support Supports ACID properties (Atomicity, Consistency, Isolation, and Durability).

Replication Option Specifies any of the two replica placement strategy names. The strategy names are Simple Strategy or Network Topology Strategy. The replica placement strategies are:

1. Simple Strategy: Specifies simply a replication factor for the cluster.
2. Network Topology Strategy: Allows setting the replication factor for each data center independently.

Data Types Table 3.14 gives the data types built into Cassandra, their usage and descriptions

Table 3.14 Data types built into Cassandra, their usage and description

CQL Type	Description
ascii	US-ASCII character string
bigint	64-bit signed long integer
blob	Arbitrary bytes (no validation), BLOB expressed in hexadecimal
boolean	True or false
counter	Distributed counter value (64-bit long)
decimal	Variable-precision decimal integer, float
double	64-bit IEEE-754 *double precession* floating point integer, float

float	32-bit IEEE-754 *single precession* floating point integer, float
inet	IP address string in 1Pv4 or 1Pv6 format, used by the python-cql driver and CQL native protocols
int	32-bit signed integer
list	A collection of one or more ordered elements
map	AJSON-style array of literals: {literal: literal, literal: literal … }
set	A collection of one or more elements
text	UTF-8 encoded string
timestamp	Date plus time, encoded as 8 bytes since epoch integers, strings
varchar	UTF-8 encoded string
varint	Arbitrary-precision integer

Cassadra Data Model Cassandra Data model is based on Google's BigTable (Section 3.3.3.2). Each value maps with two strings (row key, column key) and timestamp, similar to HBase (Example 2.4). The database can be considered as a sparse distributed multi-dimensional sorted map. Google file system splits the table into multiple tablets (segments of the table) along a row. Each tablet, called METAl tablet, maximum size is 200 MB, above which a compression algorithm used. METAO is the master-server. Querying by METAO server retrieves a METAl tablet. During execution of the application, caching of locations of tablets reduces the number of queries.

Cassandra Data Model consists of four main components: (i) Cluster: Made up of multiple nodes and keyspaces, (ii) Keyspace: a namespace to group multiple column families, especially one per partition,
(iii) Column: consists of a column name, value and timestamp and (iv) Column• family: multiple columns with row key reference. Cassandra does keyspace management using partitioning of keys into ranges and assigning different key• ranges to specific nodes.

Following Commands prints a description (typically a series of DDL statements) of a schema element or the cluster:

DESCRIBE CLUSTER

DESCRIBE SCHEMA

DESCRIBE KEYSPACES

DESCRIBE KEYSPACE <keyspace name>

DESCRIBE TABLES

DESCRIBE TABLE <table name>

DESCRIBE INDEX <index name>

DESCRIBE MATERIALIZED VIEW <view name>

DESCRIBE TYPES

DESCRIBE TYPE <type name>

DESCRIBE FUNCTIONS

DESCRIBE FUNCTION <function name>

DESCRIBE AGGREGATES

DESCRIBE AGGREGATE <aggregate function name>

Consistency Command CONSISTENCY shows the current consistency level. CONSISTENCY <LEVEL> sets a new consistency level. Valid consistency levels are ANY, ONE, TWO, THREE,QUORUM, LOCAL_ONE, LOCAL_QUORUM, EACH_QUORUM, SERIAL AND LOCAL_SERIAL. Following are their meanings:

1. ALL: Highly consistent. A write must be written to commitlog and memtable on all replica nodes in the cluster.
2. EACH_QUORUM: A write must be written to commitlog and memtable on quorum of replica nodes in all data centers.
3. LOCAL_QUORUM: A write must be written to commitlog and memtable on quorum of replica nodes in the same center.
4. ONE: A write must be written to commitlog and memtable of at least one replica node.
5. TWO, THREE: Same as One but at least two and three replica nodes, respectively.

6. LOCAL_ONE: A write must be written for at least one replica node in the local data center.
7. ANY: A write must be written to at least one node.
8. SERIAL: Linearizable consistency to prevent unconditional update.
9. LOCAL_SERIAL: Same as Serial but restricted to the local data center.

Keyspaces A keyspace (or key space) in a NoSQL data store is an object that contains all column families of a design as a bundle. Keyspace is the outermost grouping of the data in the data store. It is similar to relational database. Generally, there is one keyspace per application. Keyspace in Cassandra is a namespace that defines data replication on nodes. A cluster contains one keyspace per node.

Create Keyspace Command CREATE KEYSPACE <Keyspace Name> WITH replication = {'class': '<Strategy name>', 'replication factor': '<No. of replicas>'} AND durable_ writes= '<TRUE/FALSE>';

CREATE KEYSPACE statement has attributes replication with option class and replication factor, and durable_write.

Default value of *durable_writes* properties of a table is set to true. That commands the Cassandra to use Commit Log for updates on the current Keyspace true or false. The option is not compulsory.

1. ALTER KEYSPACE command changes (alter) properties, such as the number of replicas and the durable_writes of a keyspace: ALTER KEYSPACE <Keyspace Name> WITH replication = ['class': '<Strategy name>', 'replication factor': '<No. of replicas>'};
2. DESCRIBE KEYSPACE command displays the existing keyspaces.
3. DROP KEYSPACE command drops a keyspace:
4. Re-executing the drop command to drop the same keyspace will result in configuration exception.
5. Use KEYSPACE command connects the client session with a keyspace.

Cassandra Query Language (CQL) Table 3.15 gives the CQL commands and their functionalities.

Table 3.15 CQL commands and their functionalities

Command	Functionality
CQLSH	A command line shell for interacting with Cassandra through CQL
HELP	Runs help. This displays the list of all the commands
CONSISTENCY	Shows the current consistency level
EXIT	Terminate the CQL shell
SHOW HOST	Displays the host
SHOW VERSION	Displays the details of current cqlsh session such as host, Cassandra version, or data type assumptions
CREATE KEYSPACE <Keyspace Name>	Creates keyspace with a name
DESCRIBE KEYSPACE <Keyspace Name>	Displays the keyspace with a name
ALTER KEYSPACE <Keyspace Name>	Modifies keyspace with a name
DROP KEYSPACE =Keyspace Name>	Deletes keyspace with a name
CREATE (TABLE | COLUMNFAMILY)	Creates a table or column family
COLLECTIONS	Lists the Collections

The following example provides the sample usages of the commands.

EXAMPLE 3.13

Give the examples of usages of various CQL commands.

SOLUTION

(1) Create Table Command: CREATE TABLE command creates a table in the current keyspace:

```
CREATE (TABLE COLUMNFAMILY) <tablename>
('<column-definition>', '<column-definition>')
(WITH <option> AND <option>);
```

Primary key is a column used to uniquely identify a row. Therefore, defining a primary key is compulsory while creating a table. A primary key is made of one or more columns of a table.

Example: Create a table *Productinfo* in the keyspace *lego,* with primary key field *Productld.*

```
Use lego;

Create table Productinfo(Productid int primary
key, ProductType text);
```

(2) Describe Tables Command: DESCRIBE TABLE Command displays all the tables in the current keyspace:

```
DESCRIBE TABLE <TABLE NAME>;
```

Example: Display the details of a table *Productinfo:*

```
DESCRIBE TABLE Productinfo;
```

(3) Alter Tables Command:

```
ALTER TABLE Command ALTER (TABLE COLUMNFAMILY)
<tablename> (ADD | DROP) <column name>
```

The above command adds a column in the table or to delete a column of the table:

Example: Add a column dateOfManufacturing in the table *Productinfo:*

```
ALTER TABLE Product Info add dateOfManufacturing
times tamp;
```

* timestamp is a datatype used for date fields.

(4) Cassandra CURD Operations: (CURD-Create, Update, Read and Delete data into tables) :

(a) Insert Command:

INSERT command creates data in a table:

```
INSERT INTO <tablename> (<columnl name>, <column2 name>....) VALUES (<valuel>, <value2>....) USING <option>
```

(b) Update Command:

UPDATE command updates data in a table. The following keywords are used while updating data in a table:

Where - This clause is used to select the row to be updated.

Set - Set the value using this keyword.

Must - Includes all the columns composing the primary key.

If a given row is unavailable, then UPDATE creates a new row.

```
UPDATE <tablename> SET <column name>= <new value> <column name>= <value>.... WHERE <condition>
```

[A WHERE clause can be used only on the columns that are a part of primary key or have a secondary index on them.]

(c) Select Command

SELECT command reads the data from a table. The command can read a whole table, a single column, or a particular cell:

```
SELECT <column name(s)> FROM <Table Name>
```

To select all records:

```
SELECT* FROM <Table Name>
```

To select records that fulfils required condition:

```
SELECT <columnl, column2,..> FROM <Table Name> where <Condition>
```

Example: Select Product Type, Product Id, Product Name, and Product Cost of Product whose Productld is 31047:

```
SELECT Product Type, Product Id, Product Name, and Product Cost
```

```
from Productinfo where Productid  31047;
```

(d) Delete Command

DELETE command deletes data from a table:

```
DELETE FROM <identifier> WHERE <condition>;
```

Example: Delete row from a table where Product id is 31047:

```
DELETE FROM Productinfo WHERE Productid = 31047;
```

(5) Creating a Table with List

CREATE Table command is used for creating a table with a list.

The following query creates a table with two columns, one is the primary key and the other has multiple items (List):

```
CREATE TABLE data (<column name>, <data type>
PRIMARY KEY, <column name list<data type>);
```

Example : Create a sample table *Contactlnfo* with three columns: *Sno, name* and *Emailid.* To store multiple Email Ids, use a list:

```
create table Contactinfo (Sno int Primary key,
Name text, emailid list <text>);
```

(6) Insert Command for inserting data into a list

INSERT Command also inserts data into a list. To insert data into the elements in a list, enter all the values separated by a comma within square braces []:

```
INSERT INTO <table name> (columnl, column2,)
VALUES (valuel, value2, [list valuel, list value2,
…J )
```

Example: Insert data of three persons into the *Contactlnfo* Table:

```
Insert into Contactinfo (Sno, Name, Email Id)
values
(1,                    'Rahul', ['rahul@gmail.com',
'rahul@yahoo.com']);
```

```
Insert into Contactinfo (Sno, Name, Email Id)
values (1, 'Geetika', ['geetika@gmail.com',
'geetika@yahoo.com']);
Insert into Contactinfo (Sno, Name, Email Id)
values (1, 'Deepika', ['deepika@gmail.com',
'deepika@yahoo.com']);
```

(7) Update Command for updating Data into a List

UPDATE command also updates data into a list:

```
UPDATE <table Name> SET <New data> where
<condition>.
```

Example : Add one more email Id to the *emailld* list in *ContactInfo* table :

```
UPDATE Contactinfo SET emailid emailid +
['preeti@ymail.com'] where SNo=1.
```

CassandraClient A relational database client connects to DB server using drivers. Java JDBC driver API enables storing and retrieving data. Cassandra has peer-to-peer distribution architecture. Several instances require the clients. The driver enables the use of different languages for connecting to DBs. Cassandra does not include the drivers.

A client-generation layer enables the database interactions. AVRO project provides the client generation layer. Third party sources provide Cassandra clients in Java, Ruby, C#, Python, Perl, PHP, C++, Scala and other languages. The Cassandra client can be included in the applications.

CassandraHadoop Support Cassandra 2.1 has Hadoop 2 support. The setup and configuration overlays a Hadoop cluster on the Cassandra nodes. A server is configured for the NameNode andJobTracker. Each Cassandra node then installs the TaskTracker and Data Node.

The nodes in the Cassandra cluster can read data from the data in the Data Node in HDFS as well as from Cassandra. A client application sends the MapReduce input to Job Tracker/Resource Manager. RM/JobTracker sends a MapReduce request of job to the Task Trackers/Node Managers and clients such

as MapReduce and Pig. The Reducer output writes to Cassandra. The client gets the results from Cassandra.

Self-Assessment Exercise linked to LO 3.6

1. List the differences between Cassandra, Google BigTable and HBase data models.
2. Compare Cassandra and RDBMS.
3. List the data types used in Cassandra.
4. How are the Cassandra query language and database commands used?
5. List the components in Casandra and their uses.
6. Write the syntax to create keyspace in Cassandra. State when the ALTER keyspace is used.
7. How are the Cassandra CQL collections used?