Module 3
Chapter 3 Text Book 1
NoSQL Big Data Management, MongoDB and Cassandra

Syllabus: Introduction, NoSQL Data Store, NoSQL Data Architecture Patterns, NoSQL to Manage Big Data, Shared-Nothing Architecture for Big Data Tasks, MongoDB, Databases, Cassandra Databases. Text book 1: Chapter 3: 3.1-3.7

Course Outcome 3: Determine the concepts of NoSQL using MongoDB and Cassandra for Big Data.

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

1. **Increased reliability and fault tolerance:** The important advantage of distributed computing system 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:

- Issues in troubleshooting in a larger networking infrastructure
- Additional software requirements and
- Security risks for data and resources.

3.2 NOSQL DATA STORE

Features of SQL

- SQL is a programming language based on relational algebra.
- It is a declarative language and it defines the data schema. SQL creates databases and RDBMS.
- 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.
- **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.
- **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.
- **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
- 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.
- The traditional RDBMS has a problem when storing the records beyond a certain physical swage limit This is because RDBMS does not support horizontal scalability

3.2.1 NoSQL

Features of NoSQL

- For example, consider **sharding** a big table in a DBMS into two. Assume writing (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.**
- 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 (1) does not follow SQL compliant formats, ("Not only SQL." use SQL compliant formats with variety of other querying and access methods.
- 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 Not Only SQL is a Class of non-relational data storage systems, flexible data models and multiple schema.
- NoSQL DB does not require specialized RDBMS like storage and hardware for processing. Storage can be on a cloud.
- 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, S3 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 S3)], 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).

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

Big Data NoSQL Solutions

- Apache Hbase
- Apache MongoDB
- Apache Casandra
- Apache CouchDB
- Oracle NoSQL
- Riak

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.
- 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.
- **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.
- **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 (Consistency. Availability and Partition 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.

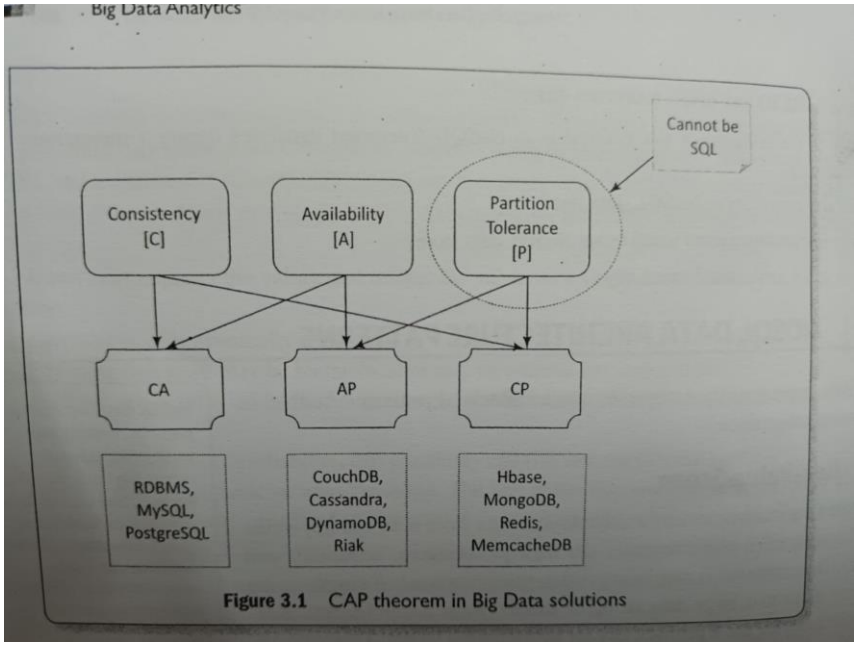


Figure 3.1 CAP theorem in Big Data solutions

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.
- **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

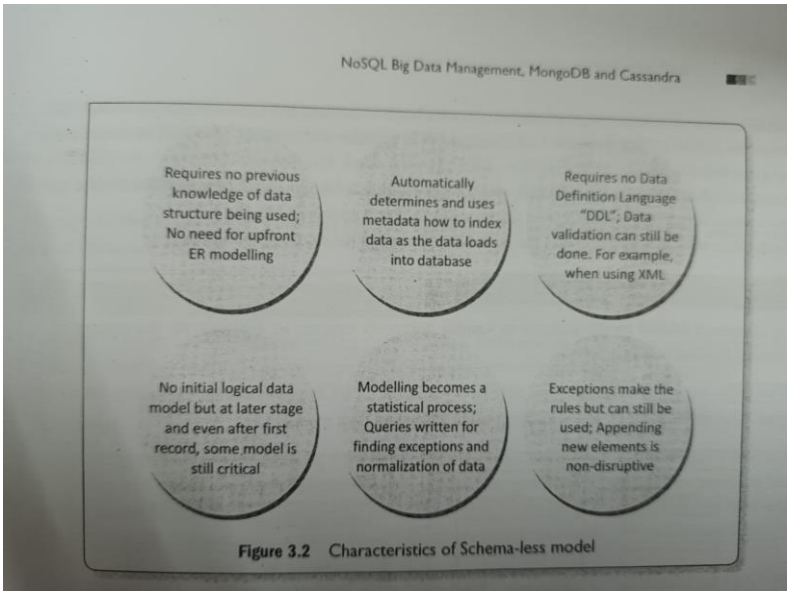


Figure 3.2 Characteristics of Schema-less model

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**

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-like adherence to ACID properties

EXAMPLE 3.1

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

SOLUTION

Figure 3.3 shows increasing flexibility concept using additional data models.

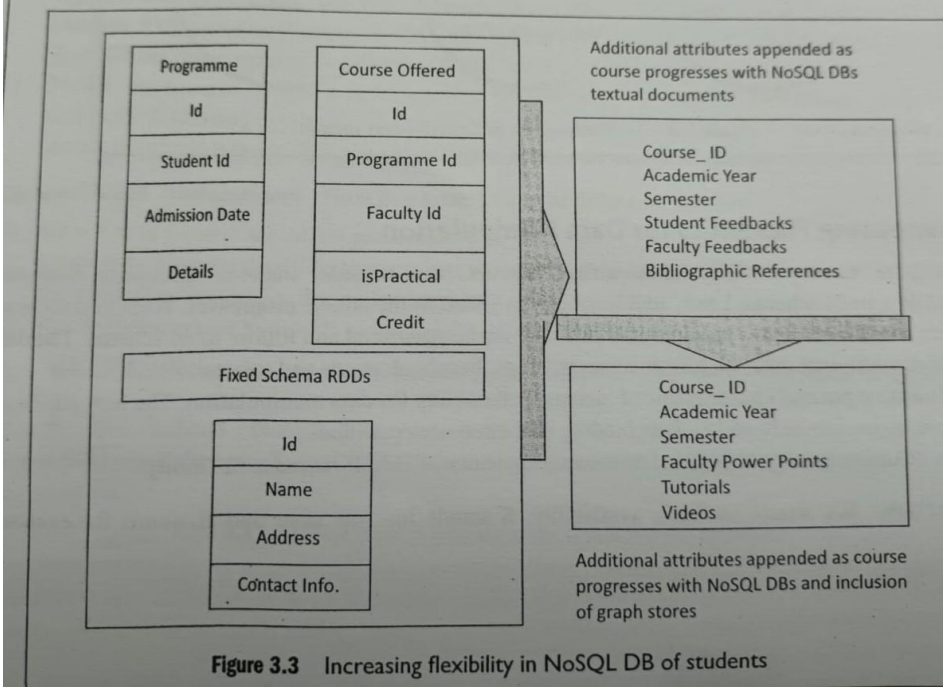


Figure 3.3 Increasing flexibility in NoSQL DB of students

3.3 NOSQL DATA ARCHITECTURE PATTERNS

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

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

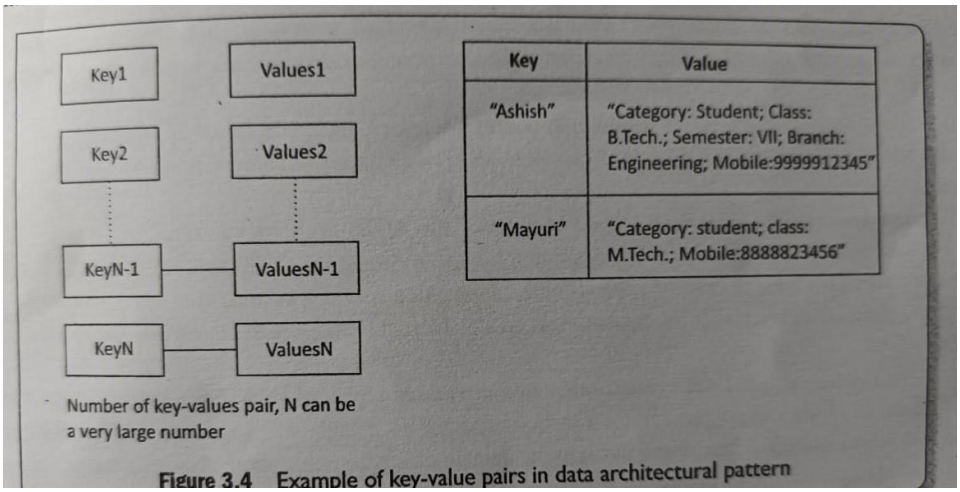


Figure 3.4 Example of key-value pairs in data architectural pattern

- **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. item. Values can be of any data

2. A query just requests the values and returns the values as a single item,

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 quries 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 man 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 alread present.

(iii) Multi-get (key1, key2, .., keyN), returns the list of values associated with the list of key

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

Limitations of key-value store architectural pattern are:

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

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

(3) Maintaining unique values as keys may become more difficult when the volume of data increase One cannot retrieve a single result when a key-value pair is not uniquely identified.

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

(5) Riak is open source data store.It is a key value data store system.

(6) Data auto-distributes and replicates in Riak.

(7) It is thus, fault tolerant and reliable.

(8) Some other widely used key-value pairs 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.

5.3.2 Document Store

- **Characteristics of Document** Data Store are high performance and flexibility. Scalability varies, depends on Cored 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, in JSON formats data model Ex: XML document object model (DOM), or machine-readable data as one BLOB. Hierarchical information stores in a single unit called **document tree**.

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, document exchange and (v) document search.

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

- **Real-life Datasets** describe a very large real-life dataset for Big Data analytics.
- An application later analyses the structures in csv, json or other, and **creates data stores in new forms.**
- Runs in next step ETL, analytics or other functions.
- This feature is called **late binding** (schema-on-read, or schema-on-need).
- **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 (Java 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.

EXAMPLE 3.3

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, CS101, ""Theory of Computations"", 7.8.
1, CS102,1, ""Computer Architecture"", 7.8.
.....
2, CS204, ""Object Oriented Programming"", 7.2.

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

1, Theory, EL101, ""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 $(4 \times 9) = 36$ for preetiGradeSheet.csv and $(5 \times 5) = 25$ 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, *studentGradeSheets.json* 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,
masterfile: "Students_Grades",
instancetype: "single",
mandatory: true,
"description": "Uniquely identifies student grade master file Object
Students_Grades "
```

- 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.
- The database stores and retrieves documents, such as XML, JSON, BSON (Binary-encoded Script Object Notation (for objects)).
- Some of the popular document data stores are CouchDB, MongoDB, Terrastore, OrientDB and RavenDB.
- **CouchDB uses** 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

  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).

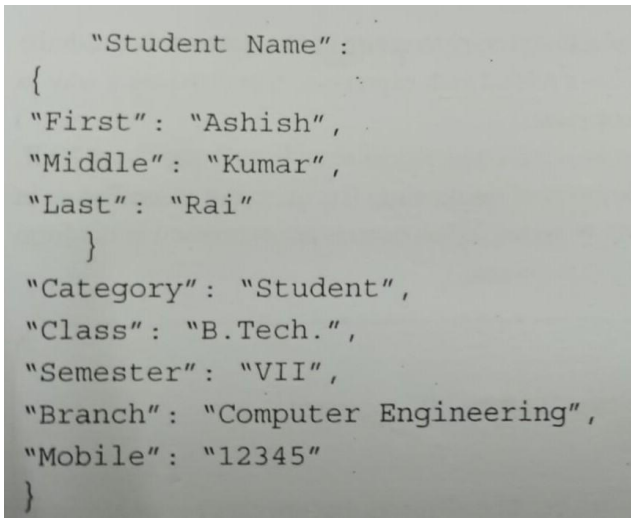
  3. CouchDB is a **multi-master application**. Write does not require field locking when controlling the concurrency during multi-master application.

  4. CouchDB querying language is JavaScript.

  5. CouchDB queries the **indices using a web browser**. CouchDB accesses the documents using HTTP API. 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"



- 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.

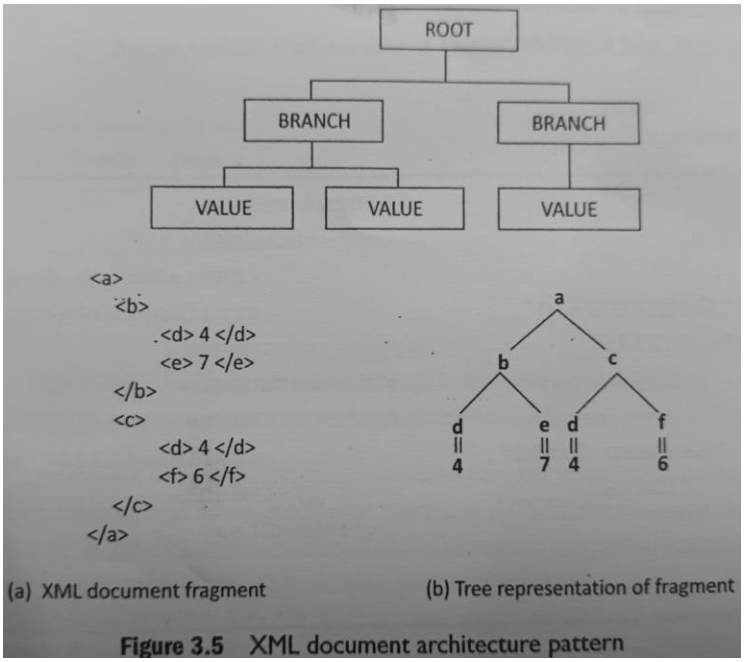


Figure 3.5 XML document architecture pattern

- 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 of XPath expressions.
- 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 in JSON 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:

```
{
   "students": [
   {
      "name": "Ashish Jain",
      "rollNo": "12345"
   },
   {
      "name": "Sandeep Joshi",
      "rollNo": "12346"
   }
   ]
}
```

(a) JSON

```
<students>
    <student>
        <name>Ashish Jain</name>
        <rollNo>12345</rollNo>
    </student>
    <student>
        <name>Sandeep Joshi</nam
        <rollNo>12346</rollNo>
    </student>
</students>
```

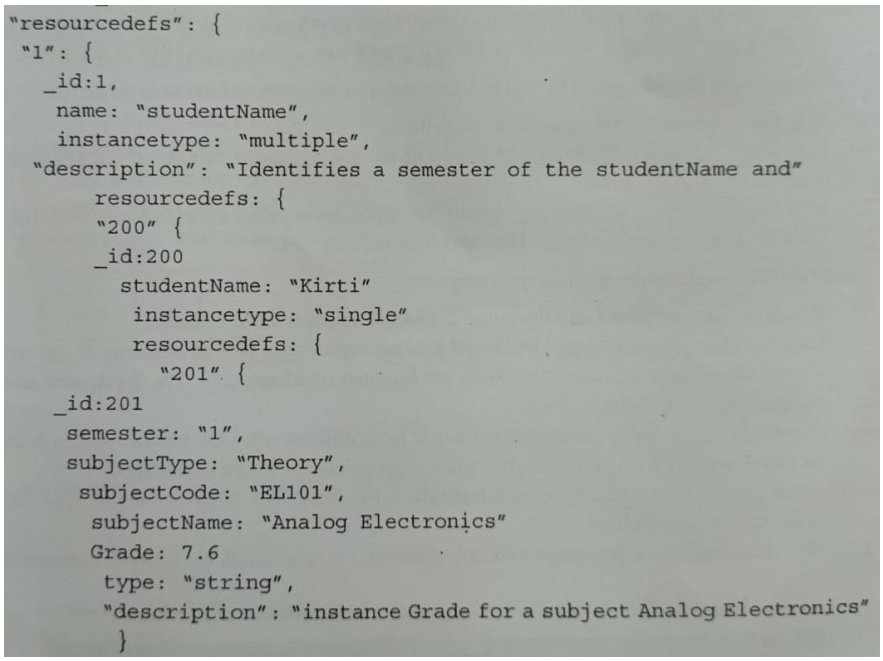
(b) XML equivalent

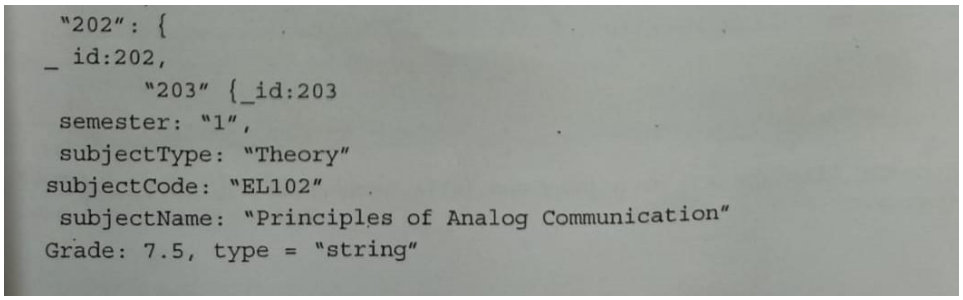
- **When compared with XML, JSON has the following advantages:**
- XML is easier to understand but XML is more verbose than JSON.
- XML is used to describe structured data and does not include arrays, whereas JSON includes arrays.
- JSON has basically key-value pairs and is easier to parse from JavaScript.
- 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 Tabular Data

- 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.
- Relational DB store is in-memory row-based data, 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 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.
- In-memory column-based DB store a column as a consecutive memory or disk entry. This strategy makes the analytics processing fast.

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.

- **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.
- A column-family head is also called a **super-column head**.
- **Examples** of columnar family data stores are HBase, BigTable, HyperTable and Cassandra

EXAMPLE 3.6

Consider Example 1.6(i). Assume in-memory columnar storage. Data for a large number of ACVMs with n 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

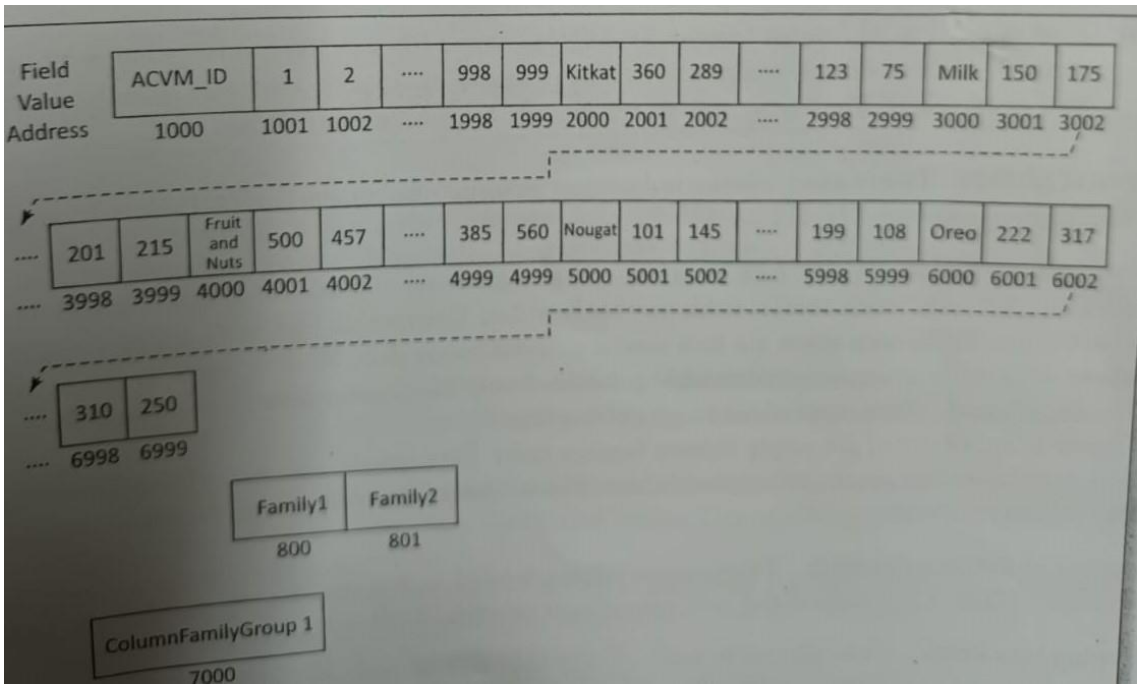
Assume the following columnar storage at memory:

(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:

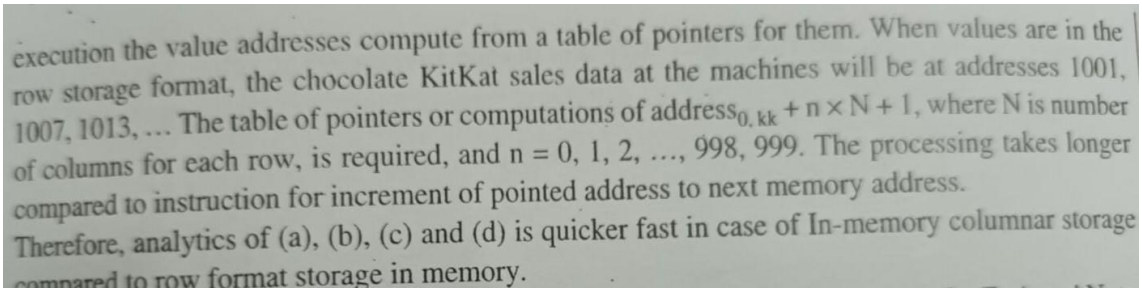
Table 3.3 Each day's sales of chocolates on 999 ACVMs

		Nestle Chocolate Flavours Group				
		Popular Flavours Family		Costly Flavours Family		
	ACVM_ID	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

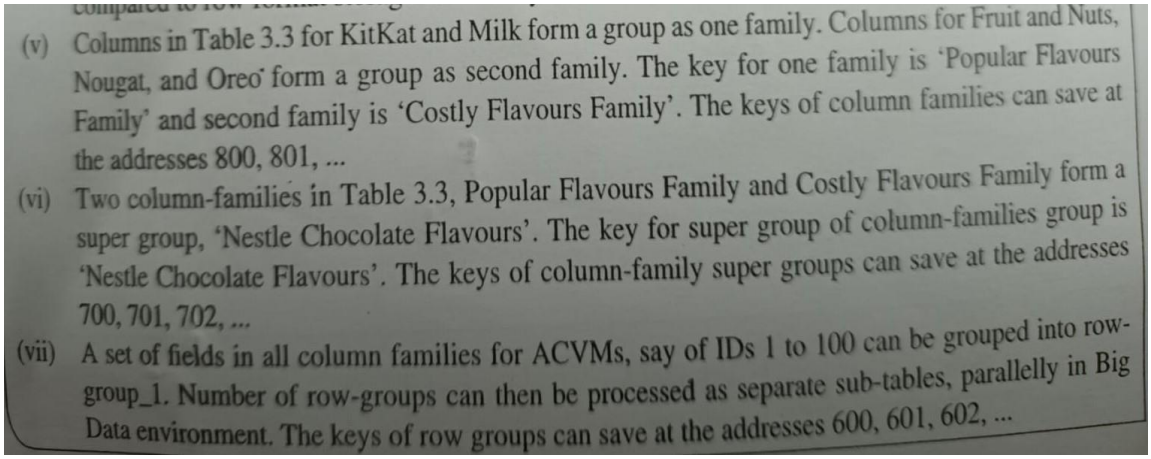


(iii) An analytics application computes the results, such as (a) total sales of each flavour, KitKat, Milk, Fruit and Nuts, Nougat and Oreo, each day, (b) Each ACVM requirement for refilling at each ACVM each day, (c) ID of maximum sales of chocolates each day, (d) cluster of ACVMs showing highest sales,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

(iv) Consider first address of sales data for KitKat, $address_{0, kk}$ = 1001 of machine ID, ACVMId at $address_0$ =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$_{0, kk} + n \times N + 1$, 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_1. 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.
- **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.
- Grouping of Column Families Two or more column-families in data store form a super group, called super column. Ex: '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.
- 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:**
- **Scalability.**
- The database uses row IDs and column names to locate a column and values at the column fields. The back-end system can distribute queries over a large number of processing nodes without **performing any Join operations**.
- Scalability means addition of number of rows as the number of ACVMs increase. 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.

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.

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 100s 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.

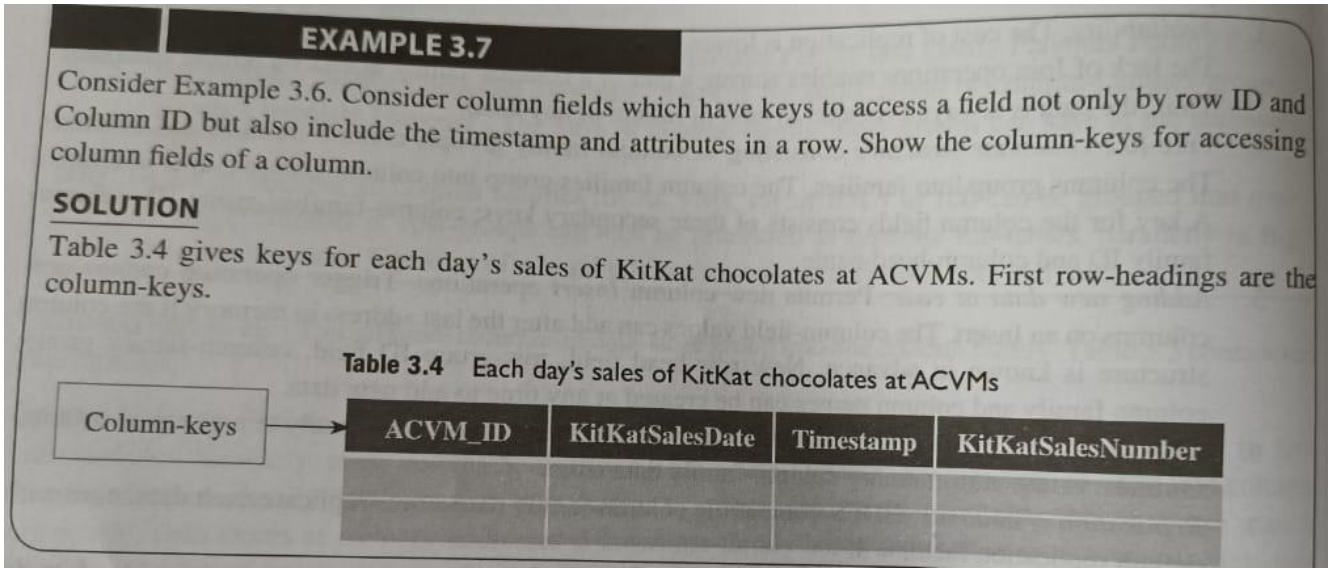
5. Handles million of operations per second.

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.



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 →	ACVM_ID	KitKatSalesDate	Timestamp	KitKatSalesNumber

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).
- **RCFile** (Record Columnar File) is a data placement structure that determines how to store relational tables on computer clusters. It is designed for systems using the MapReduce framework.
- The RCFile structure includes a data storage format, data compression approach, and optimization techniques for data reading.
- It is able to meet all the four requirements of data placement: (1) fast data loading, (2) fast query processing, (3) highly efficient storage space utilization, and (4) a strong adaptivity to dynamic data access patterns.
- **Data storage format**
- For example, a table in a database consists of 4 columns (c1 to c4):

c1	c2	c3	c4
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44
51	52	53	54

To serialize the table, RC File partitions this table first horizontally and then vertically, instead of only partitioning the table horizontally like the row-oriented DBMS (row-store). The horizontal partitioning will first partition the table into multiple row groups based on the row-group size, which is a user-specified value determining the size of each row group.

Row Group 1

c1	**c2**	**c3**	**c4**
11	**12**	**13**	**14**
21	**22**	**23**	**24**
31	**32**	**33**	**34**

Row Group 2

c1	**c2**	**c3**	**c4**
41	**42**	**43**	**44**
51	**52**	**53**	**54**

- Then, in every row group, RC File partitions the data vertically like column-store. Thus, the table will be serialized as:

  Row Group 1 Row Group 2

  11, 21, 31; 41, 51;

  12, 22, 32; 42, 52;

  13, 23, 33; 43, 53;

  14, 24, 34; 44, 54;

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.

Row-group_1 for IDs 1 to 2					
1	360	150	500	101	222
2	289	175	457	145	317

Row-group_m for IDs 998 to 999					
998	123	201	385	199	310
999	75	215	560	108	250

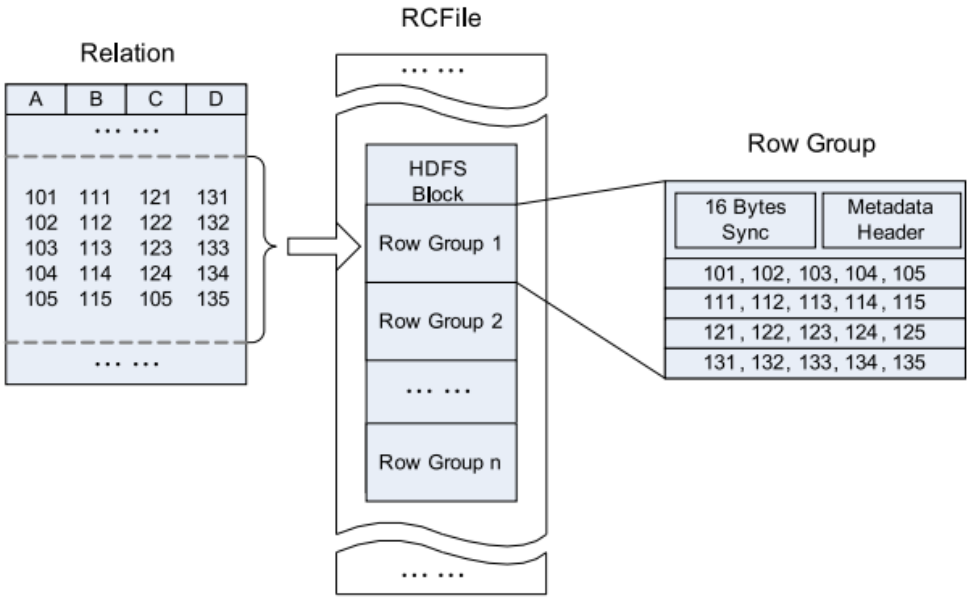
Make a file in RC format.

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.

Row group_1		Row group_m	
1, 2;		998, 999;	ACVM_ID
360, 289;		123, 75;	KitKat
....		...	Milk
....		...	Fruit and Nuts
		...	Nougat
222, 317;		310, 250;	Oreo

RC file for row group_1 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;



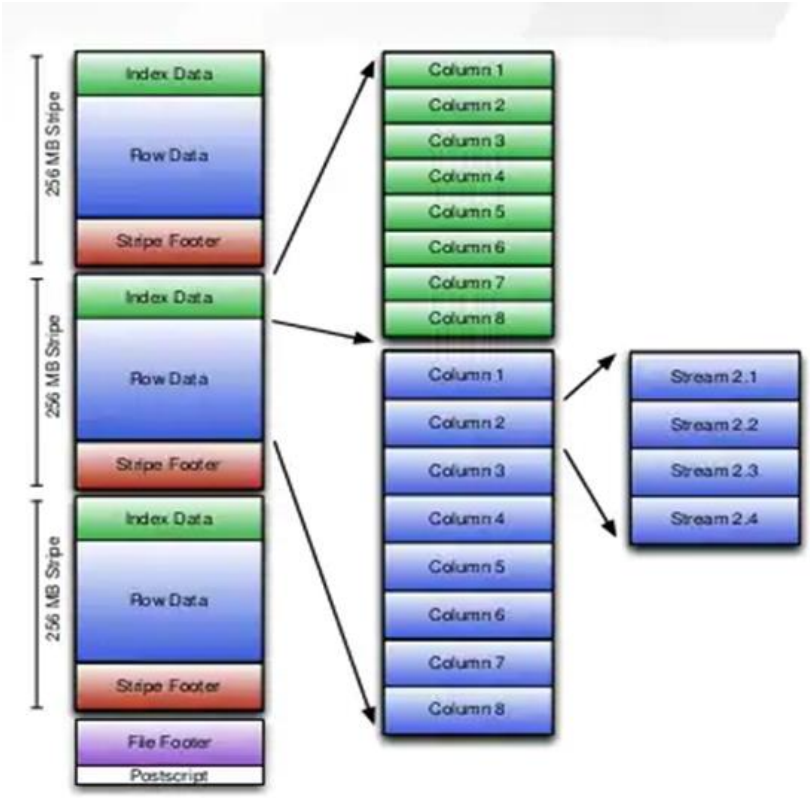
3.3.3.4 ORC File Format

- An ORC (Optimized Row Columnar) file consists of row-group data called stripes.
- ORC enables concurrent reads of the same file using separate Record Readers.

- Metadata store uses Protocol Buffers for addition and removal of fields.
- ORC is an intelligent Big Data file format for HDFS and Hive. 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.
- 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

Stripe_ID	Index Column 1				Index Column 2
	Index column 1 key 1				Index column 2 key 1
	Contents-Column name	Contents Minimum value	Contents Maximum value	Count (number) of content-column fields	
	…	…	…	…	
	…	…	…	…	
	Index column 1 key 2				Index column 2 key 2
	Column-name	Minimum value	Maximum value	Count of number of column fields	
	…	…	…	…	
	…	…	…	…	



- 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.

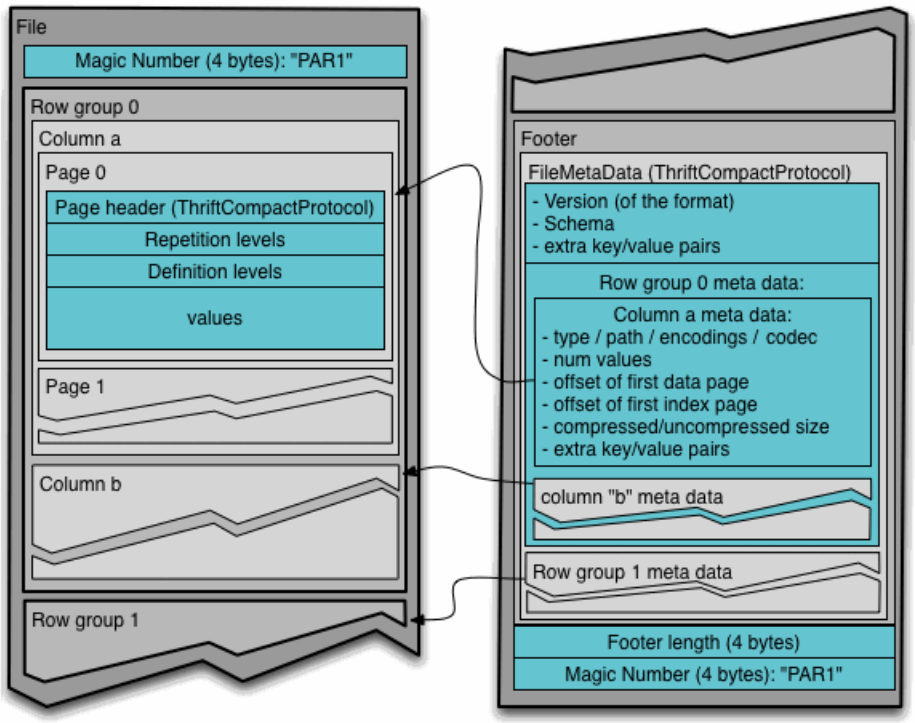
3.3.3.5 Parquet File Formats

- 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. 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 compulsorily consists of metadata, though the file need not consists of data
- The Parquet file consists of row groups. A -group 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.

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

Row-group_ID	Column Chunk 1 key			
	Page 1 key	Page 2 key	‘’’’	Page m key
	..	…	…	…
	..	…	…	…
	Column Chunk 2 key			
	Page 1key	Page 2 key	‘’’’	Page key m’
	..	…	…	…
	..	…	…	…



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 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.
- Eleven **Functions Supporting APIs**. 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)

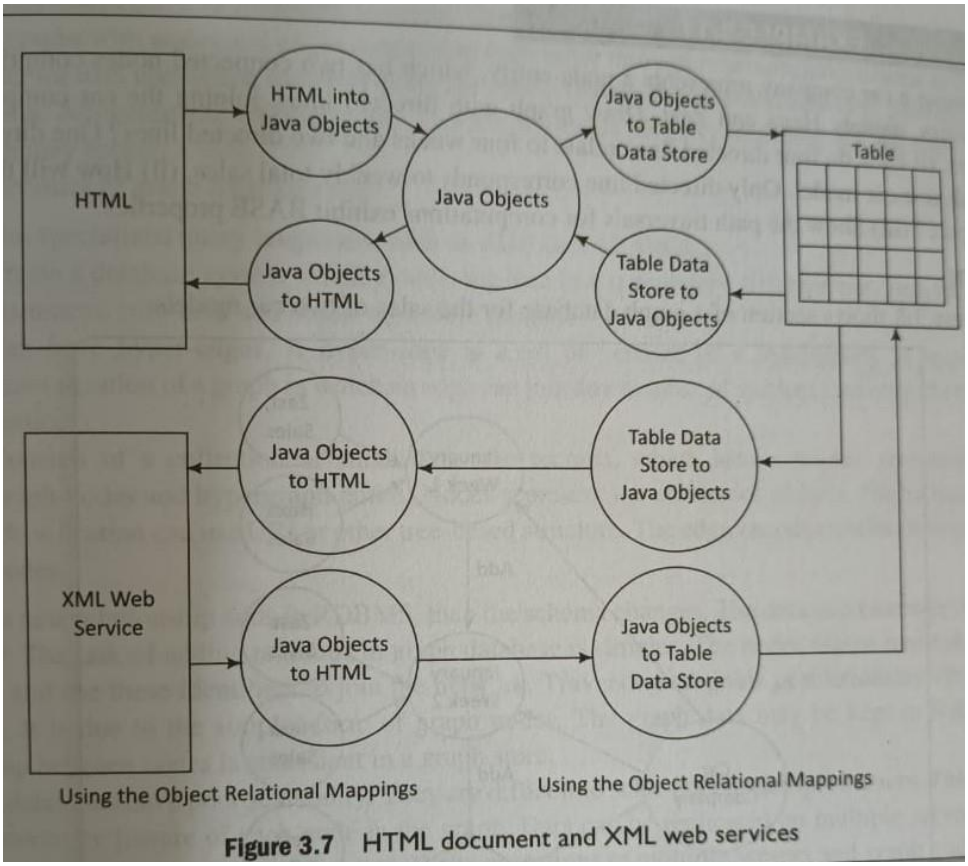
persistent object life cycle, (x) adding modules and (xi) locking and caching strategy.

- Amazon S3 (Simple Storage Service) S3 refers to Amazon web service on the cloud named S3.
- The S3 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.
- S3 assigns an ID number for each stored object.
- S3 is scalable storage infrastructure, same as used in Amazon e-commerce service. S3 may store trillions of objects.

3.3.4.1 Object Relational Mapping

Object–relational mapping (**ORM**, **O/RM**, and **O/R mapping tool**) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages.

- 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



- **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.
- 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, e2, and V is set of vertices, V_1, V_2, ..., V_1.
- **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.
- 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.
- 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.

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

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

OLUTION

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

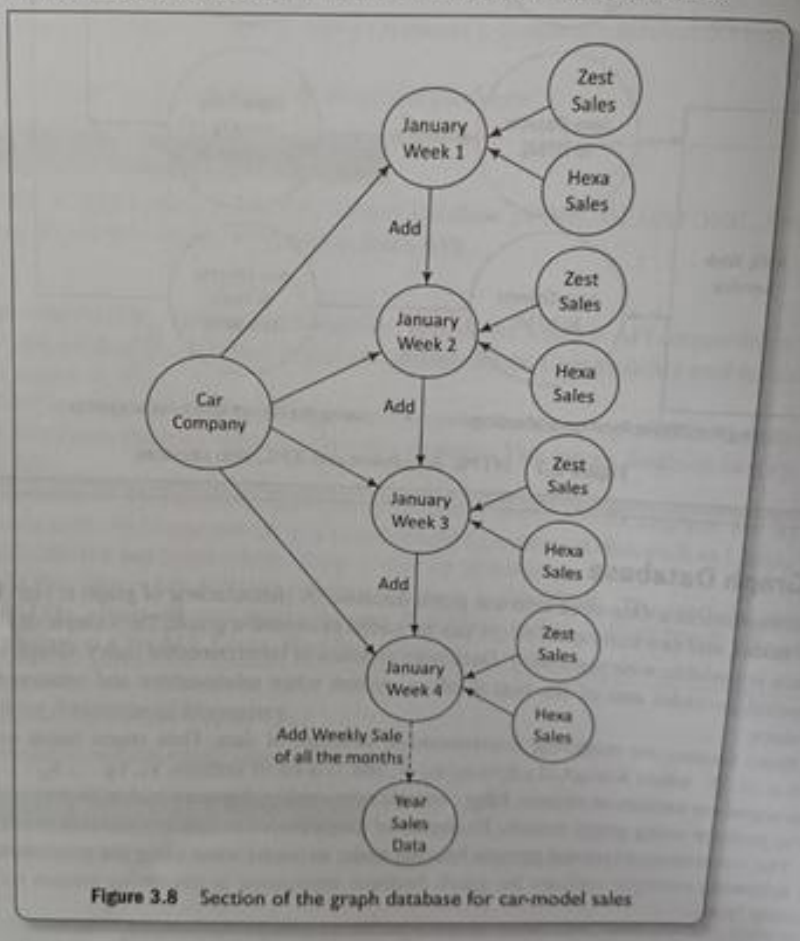


Figure 3.8 Section of the graph database for car-model sales

(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.

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).

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.

3.4 NoSQL TO MANAGE BIG DATA

3.4.1 Using NoSQL to Manage Big Data

- NoSQL
- **Limits** the support for **Join queries**, supports **sparse matrix** like columnar-family,

- 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.**
- NoSQL data processing scales horizontally

3.4.1.1 NoSQL Solutions for Big Data

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 of RDBMS.

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.

Table 3.7 gives a comparison.

Feature	NoSQL	SQL
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 of SQL	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

3.5 SHARED-NOTHING ARCHITECTURE FOR BIG DATA TASK

- The columns of tables relate by relationship. A relational algebraic equation specifies the relation. Keys share between two or more SQL tables in RDBMS.
- Shared Nothing is a **cluster architecture**. A node **do not share data** with any other node.
- Big store consists SN architecture. Big Data store, therefore, easily partitions into shards.
- A partition **processes the different quires** on data of the different users at each node independently. Thus, data processes run in parallel at the nodes.
- A node maintains a copy running-process data. A coordination protocol controls the processing at all SN nodes. An SN architecture optimizes massive parallel data processing.
- Data of different data stores partition among number nodes.
- Processing may require every node to maintain its own copy of the application data using coordination protocol. Examples using the partitioning and processing are Hadoop, Flink and spark.
- **Features of SN Architecture are:**

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 DB).
4. No network Contention.

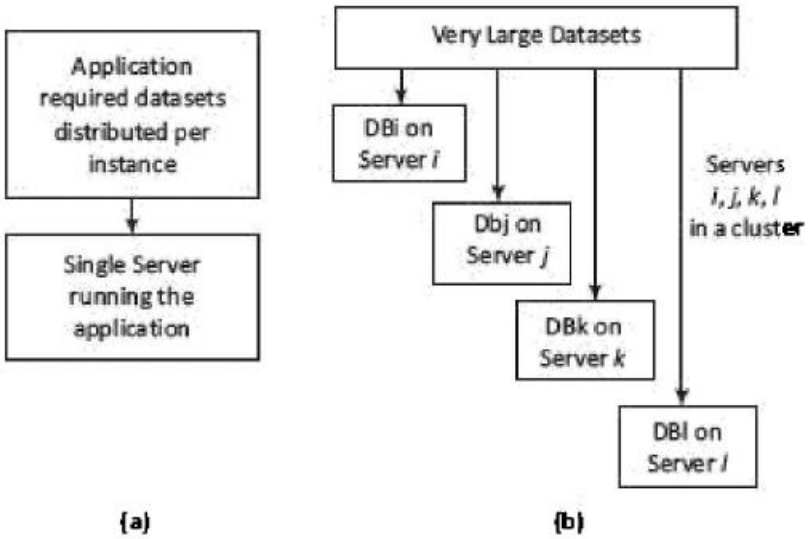
3.5.1 Choosing the Distribution Models

- 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.
- These data stores also use the SSD model. An application executes the data sequentially on a single server



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 I different servers at the cluster. DBi, DBj, DBk, and DBl, are four shards.
- 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. 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.

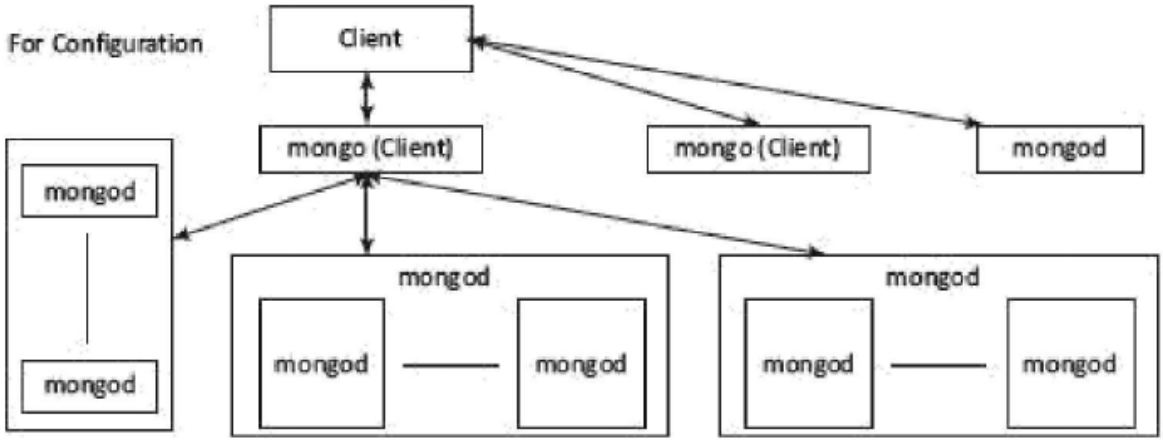


Figure 3.10 Master-slave distribution model. Mongo is a client and mongod is the server

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.

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.

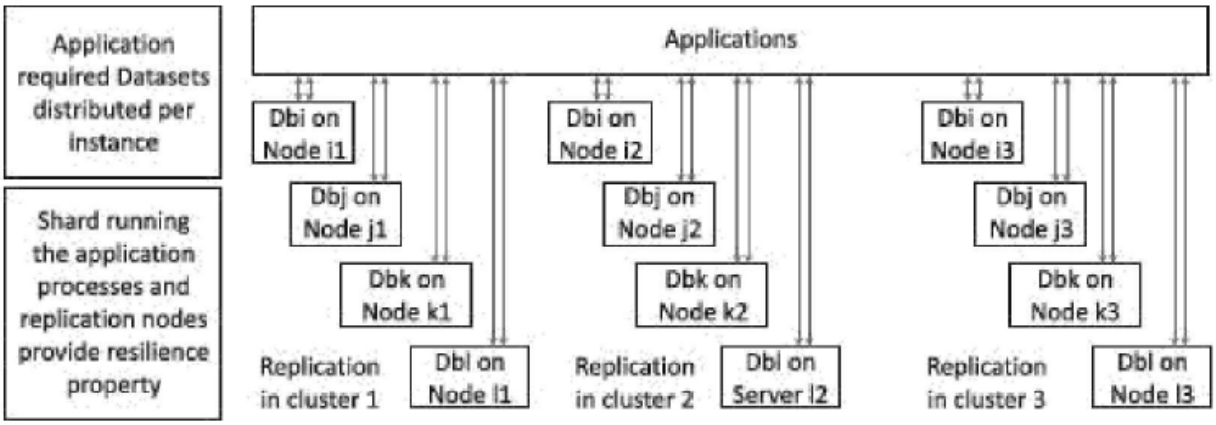


Figure 3.11 Shards replicating on the nodes, which does read and write operations both

3.5.1.5 Choosing Master-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.

3.5.2 Ways of Handling Big Data Problems

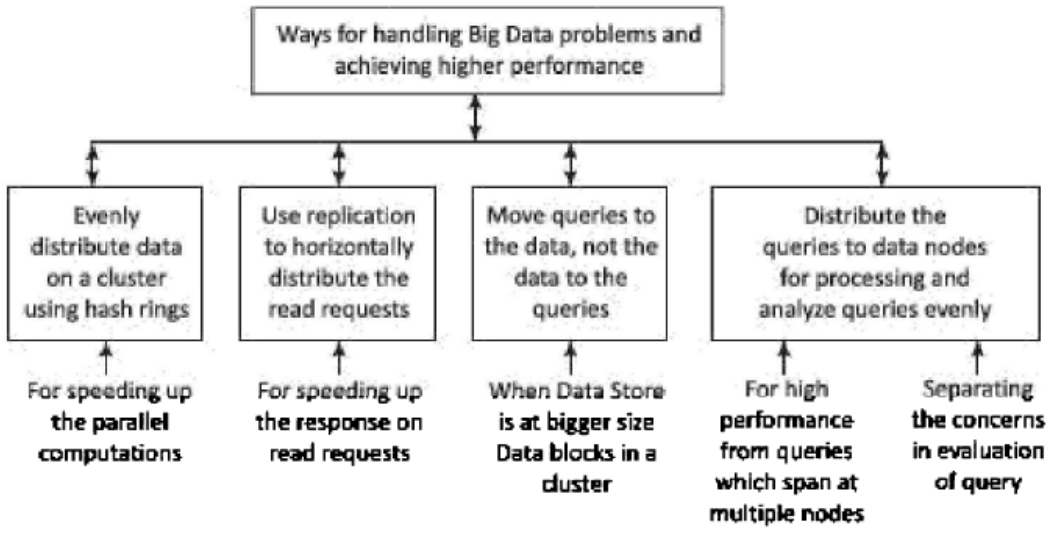


Figure 3.12 Four ways for handling big data problems

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).

3.6 MongoDB Database

- MongoDB is an open source DBMS. MongoDB programs 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 and (xi) fault tolerant. Document data store in JSON-like documents. The data store uses the dynamic schemas.
- **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 MongoDB is 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.

name	quantity	size	status	tags	rating
journal	25	14x21,cm	A	brown, lined	9
notebook	50	8.5x11,in	A	college-ruled,perforated	8
paper	100	8.5x11,in	D	watercolor	10
planner	75	22.85x30,cm	D	2019	10
postcard	45	10x,cm	D	double-sided,white	2

- **Example**

```
{ "name": "notebook",
"qty": 50,
 "rating": [ { "score": 8 }, { "score": 9 } ],
 "size": { "height": 11, "width": 8.5, "unit": "in" },
"status": "A",
"tags": [ "college-ruled", "perforated"]}
```

3. **Document model is well defined**. Structure of document is clear, Document is the unit of storing data in a MongoDB database.

- Documents are analogous to the records of RDBMS table.
- Insert, update and delete operations can be performed on a collection. Document use JSON (JavaScript 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. MongoDB is a document data store in which one collection holds different 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.

6. Storing of documents on disk is in BSON serialization format. BSON is a binary representation of JSON document. The mongo JavaScript shell and MongoDB language drivers perform translation ad 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 that's nearly as powerful as SQL.

9. No complex Joins.

10. Distributed DB makes high availability and provides horizontal scalability.

11. Indexes on any field in a collection of documents. User can create indexes on any field in a document. Indices support query and operations. By default Mongodb create indices on 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 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 of RDBMS and MongoDB databases

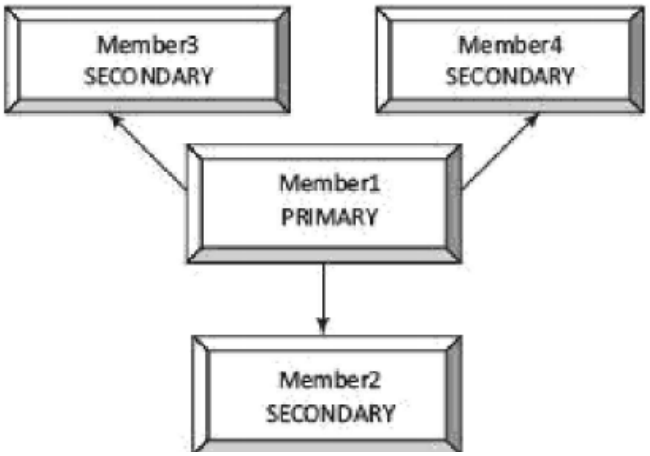
RDBMS	MongoDB
Database	Data store
Table	Collection
Column	Key
Value	Value
Records / Rows / Tuple	Document / Object
Joins	Embedded Documents
Index	Index
Primary key	Primary key (_id) is default key provided by MongoDB itself

- **Replication**
  - Replication ensures **high availability** in Big Data. Presence of multiple copies increases **on different database servers**.
  - This makes DBs **fault-tolerant a**gainst any database server failure. Multiple copies of data certainly help in **localizing** the data and ensure availability of data in a distributed system environment.
  - MongoDB replicates with the help of a **replica set**.
  - A replica set in MongoDB is a group of mongod (MongoDb server) processes that store the same dataset.
  - 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 MongoDB Client 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.



- **Auto-sharding**
  - Sharding is a method for **distributing data** across multiple in a distributed application environment.
  - A **single machine** may not be adequate to store the data. When the data size increases, do **not provide retrieval** operation.
  - **Vertical scaling** by increasing the 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.

Table 3.10 Data types which MongoDB documents 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	ObjectIds (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 ObjectId 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.

Table 3.11 Comparison of features MongoDB with 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

Table 3.12 MongoDB querying 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.<database name>. find ()	Views all documents in a collection
db.<database name>.update ()	Updates a document
db.<database name>.remove ()	Deletes a document

Sample usages of commands

- To create database:

$use lego

- To see existing database

$db

- To get list of all the databases

$show dbs

- To drop database : By default deletes test database

$use lego
$db.dropDatabase()

- **To create a collection:** 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.

```
db.lego.insert
(
        {
                "ProductCategory": "Airplane",
                "ProductId": 10725,
                "ProductName": "Lost Temple"
        }
)
```

- To add array in a collection

```
db.lego.insert
(
        [
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 10725,
                        "ProductName": "Lost Temple"
                },
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 31047,
                        "ProductName": "Propeller Plane"
                },
                {
                        "ProductCategory": "Airplane",
                        "ProductId": 31049,
                        "ProductName": "Twin Spin Helicopter"
                }
        ]
)
```

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.

- To Update a document:

  db.<database name>.update()

- To delete a document :

  db.<database name>.remove()

  db.lego.remove("ProductID":10555))