MODULE-3

NoSQL Big Data Management, Mongo DB and Cassandra

NoSQL Big Data Management

INTRODUCTION

- 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, *JSON files, BigTable, HBase, MongoDB,Cassandra, and CouchDB.*

- 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 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.

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. ***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 .
5. ***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).

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).

NOSQL DATA STORE

- 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 SQL Databases

- **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.

- **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 column_name(s) FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name ;**
- **Select KitKatSales From TransactionsTbl INNER JOIN ACVMSalesTbl ON TransactionsTb1.KitKatSales=TransactionsTb1.KitKatSales;**
- The statement selects those records in a column named KitKatSales which match the values in two tables: one TransactionsTbl and other ACVMSalesTbl.

- 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. 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	CHOICE
1	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.

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.

Big Data *NoSQL or Not-Only SQL*

- NoSQL DB does not require specialized RDBMS like storage and hardware for processing.
- Storage can be 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

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

(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

- **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.

NoSQL Data store	Description
Apache's HBase	HDFS compatible, open-source and non-relational data store written in Java; A column-family based NoSQL data store, data store providing BigTable-like capabilities ; scalability, strong consistency, versioning, configuring and maintaining data store characteristics
Apache's MongoDB	HDFS compatible; master-slave distribution ; document-oriented data store with JSON-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 ; open source; NoSQL; scalable, non-relational, column- family based, fault-tolerant and tunable consistency used by Facebook and Instagram

NoSQL Data store	Description
Apache's CouchDB	A project of Apache which is also widely used database for the web. CouchDB consists of Document Store. It uses the JSON 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.
- **C->Consistency ,A-> Availability ,P-> Partition**
- **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.
- 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.

- **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.

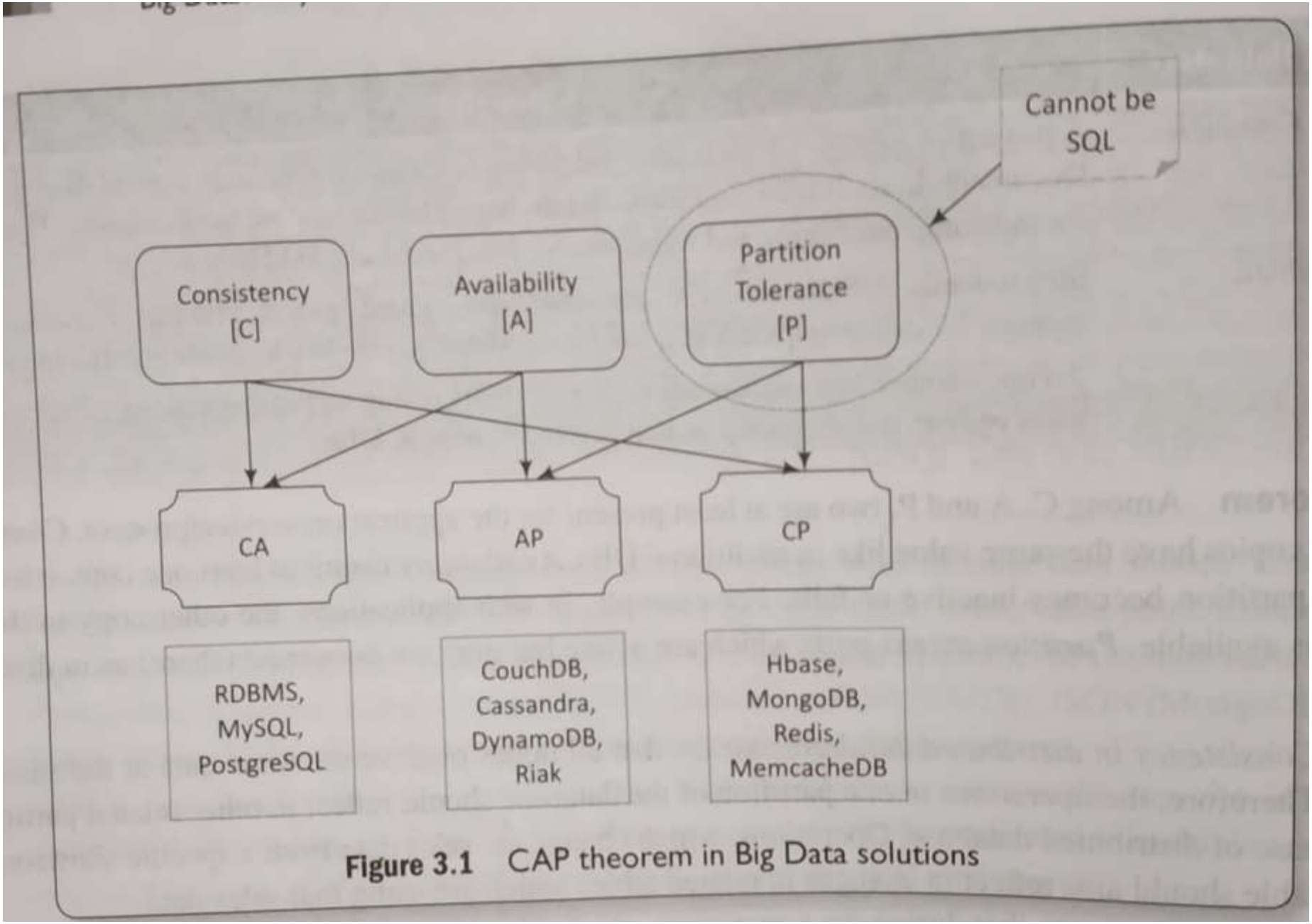


Figure 3.1 CAP theorem in Big Data solutions

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.

- 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.
- 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.

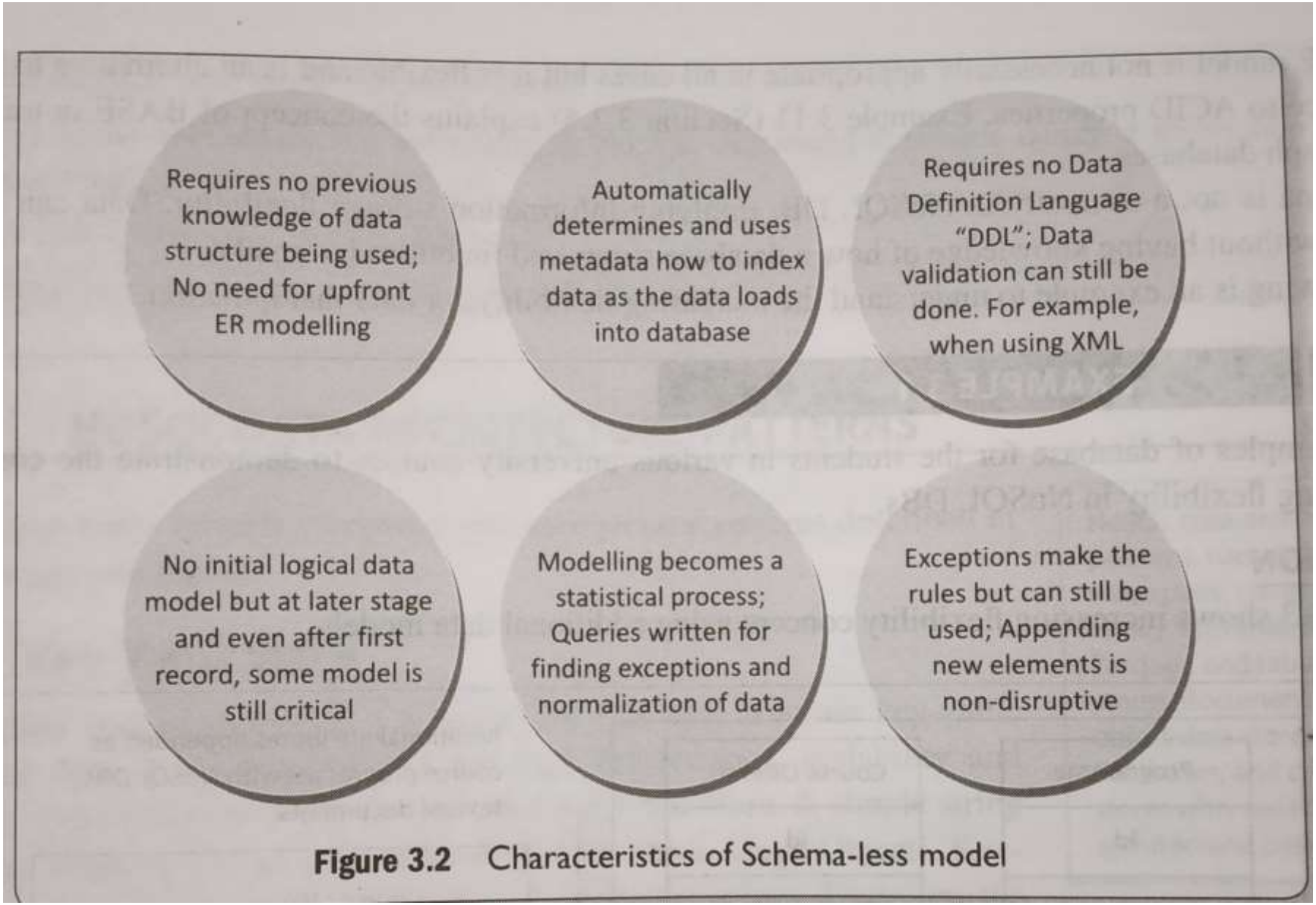


Figure 3.2 Characteristics of Schema-less model

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**.

- **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.
- **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.

- **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.

- 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.
- Use examples of database for the students in various university courses to demonstrate the concept of increasing flexibility in NoSQL DBs.
- Figure 3.3 shows increasing flexibility concept using additional data models

Figure 3.3 shows increasing flexibility

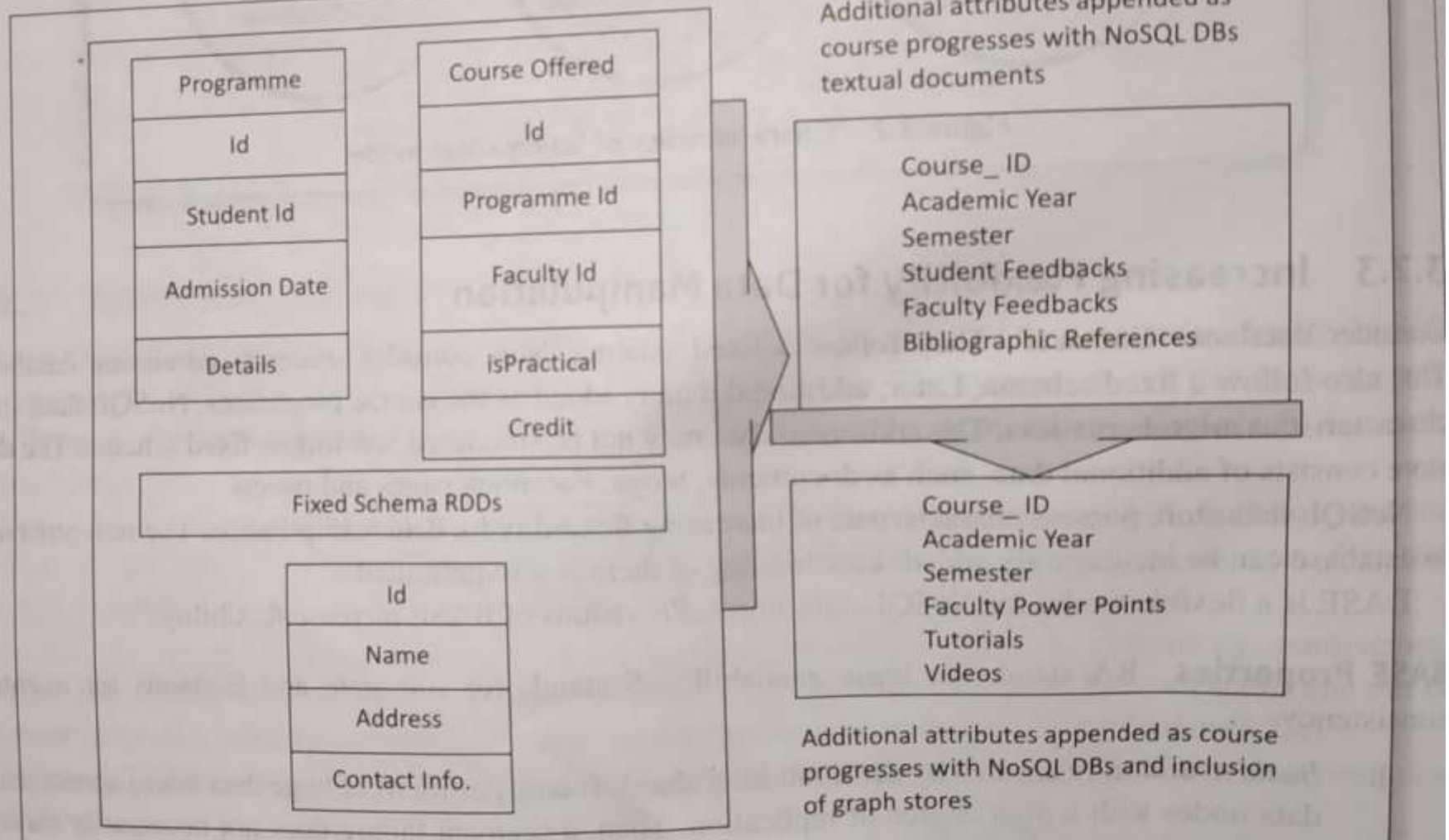


Figure 3.3 Increasing flexibility in NoSQL DB of students

NOSQL DATA ARCHITECTURE PATTERNS

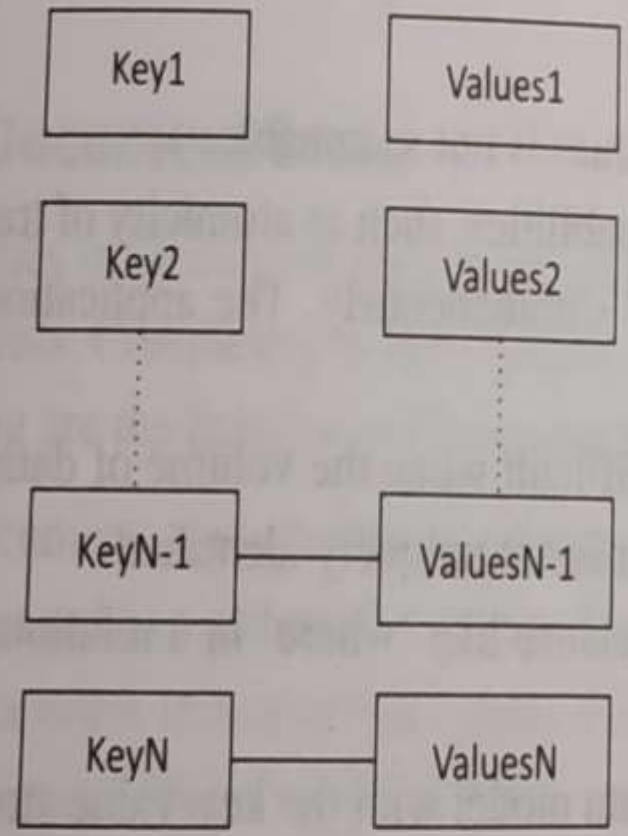
- NoSQL data stores broadly categorize into architectural patterns described in the following subsections:

1. Key-Value Store
2. Document Store
3. Tabular Data
4. Object Data Store
5. Graph Database

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.

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 (key1, 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 gives a comparison between Traditional Relational data model with 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

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, in JSON formats data model. 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,

(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.

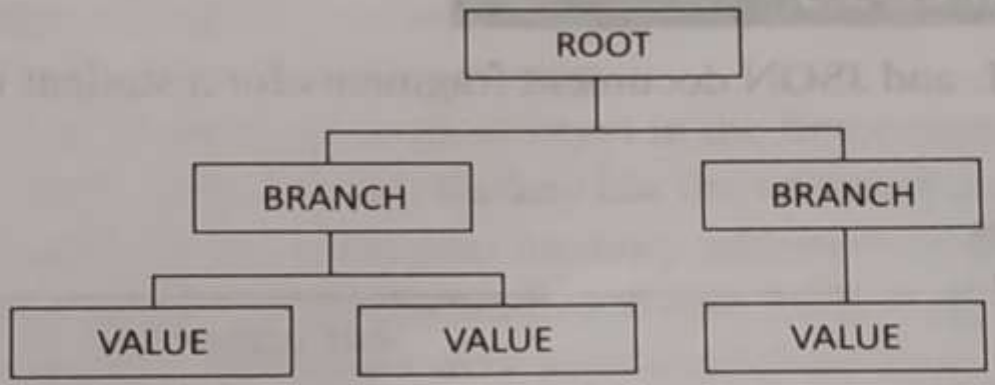
- ***Document JSON Format-MongoDB Database*** MongoDB Document database provides a rich query language and constructs, such as database indexes allowing easier handling of Big Data.

Example of Document in Document Store:

```
{
    "id” : “1001"
    “Student Name":
    {
          “First“ : “Ashish",
           “Middle” : "Kumar”,
            “Last “: “Rai"
    }
    "Category“ :  "Student",
     "Class” : B.Tech.”,
      “semester “: “VII”,
       “Branch” : “computer engineering”,
       "Mobile“ : "12345"
}
```

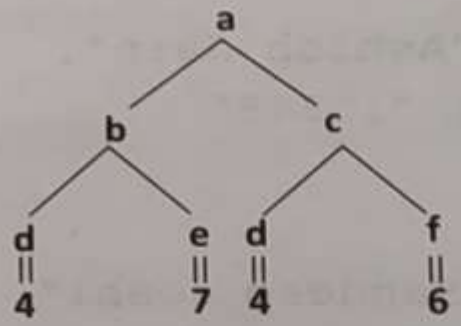
- The document store allows querying the data based on the contents as well.
- For example, it is possible to search the document where student's first name is "Ashish“.
- Document store can also provide the search value's exact location.
- The search is by using the document path.
- A type of key accesses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




```
<a>
   <b>
        <d> 4 </d>
        <e> 7 </e>
   </b>
   <c>
        <d> 4 </d>
        <f> 6 </f>
   </c>
</a>
```

(a) XML document fragment



(b) Tree representation of fragment

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.

- 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 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</name>
        <rollNo>12346</rollNo>
    </student>
</students>
```

(b) XML equivalent

- **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.

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.

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.

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

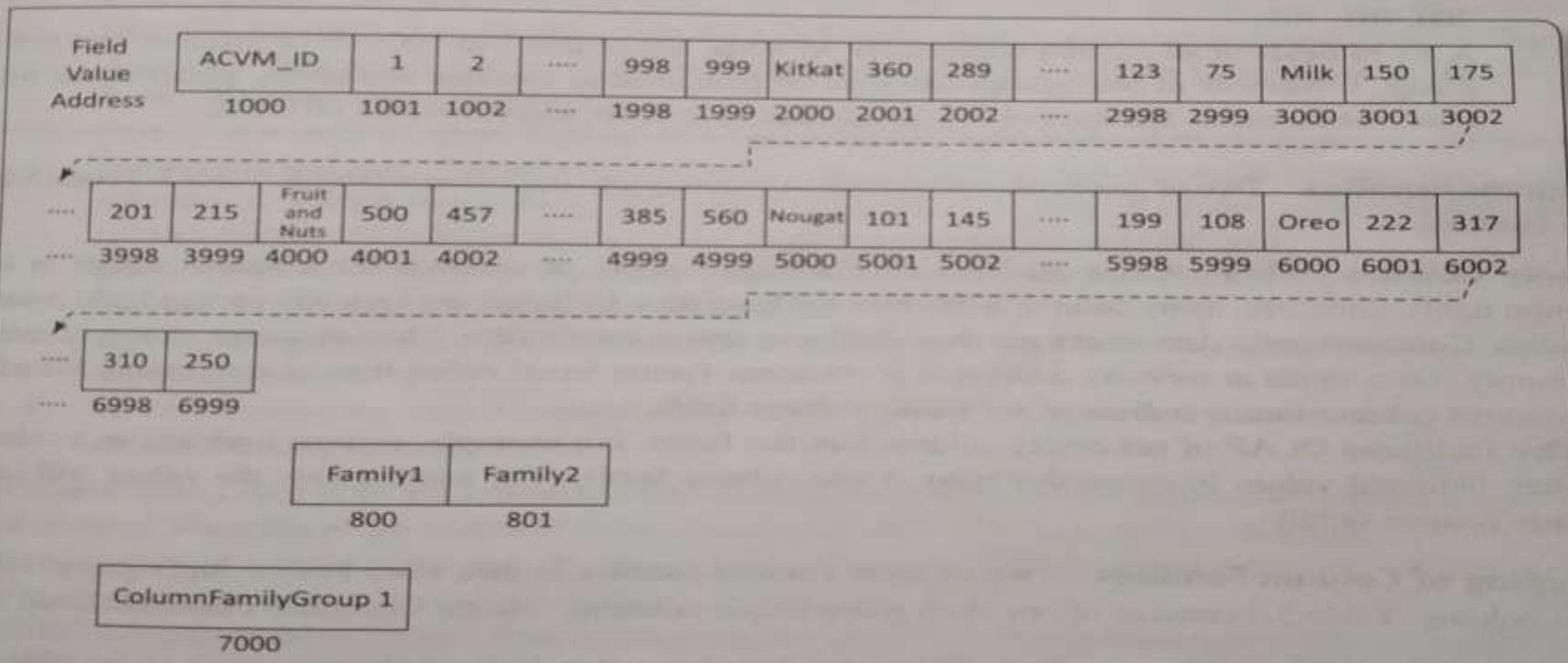


Figure 3.6 Fields in columnar storage and addresses in memory.

- **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.
- **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.

Characteristics of Columnar Family Data Store

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.

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.

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 .
- ***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

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:
- 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.

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.

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;

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.
- 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.

- 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

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	
	...	...	...	...	
	...	...	...	...	

Parquet File Formats.

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'
	..	…	…	…
	..	…	…	…

- 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 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.

- 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.

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.

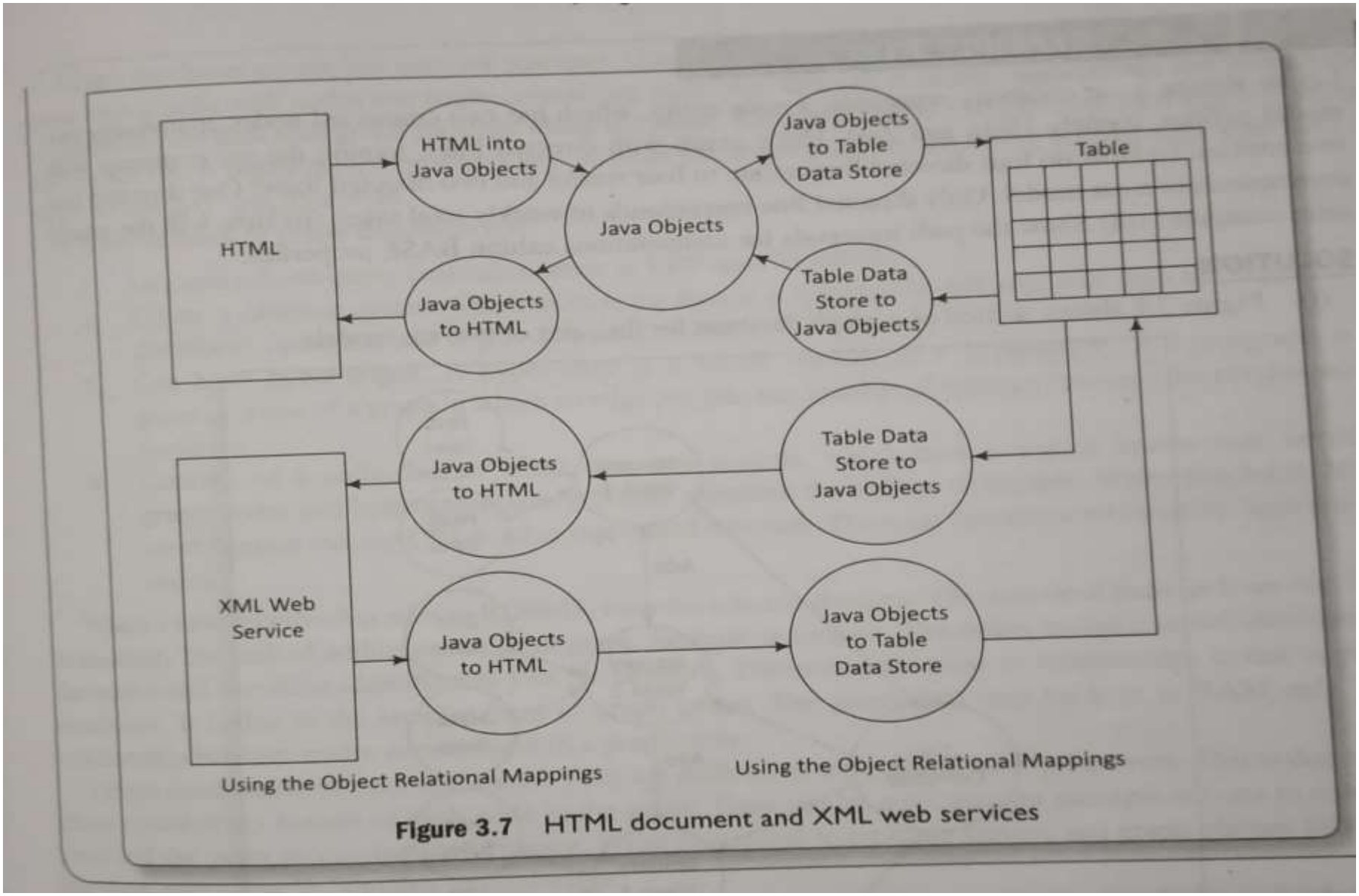


Figure 3.7 HTML document and XML web services

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 e1, e2, … and V is set of vertices, v1, v2, …., vn.
- 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.

TION

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

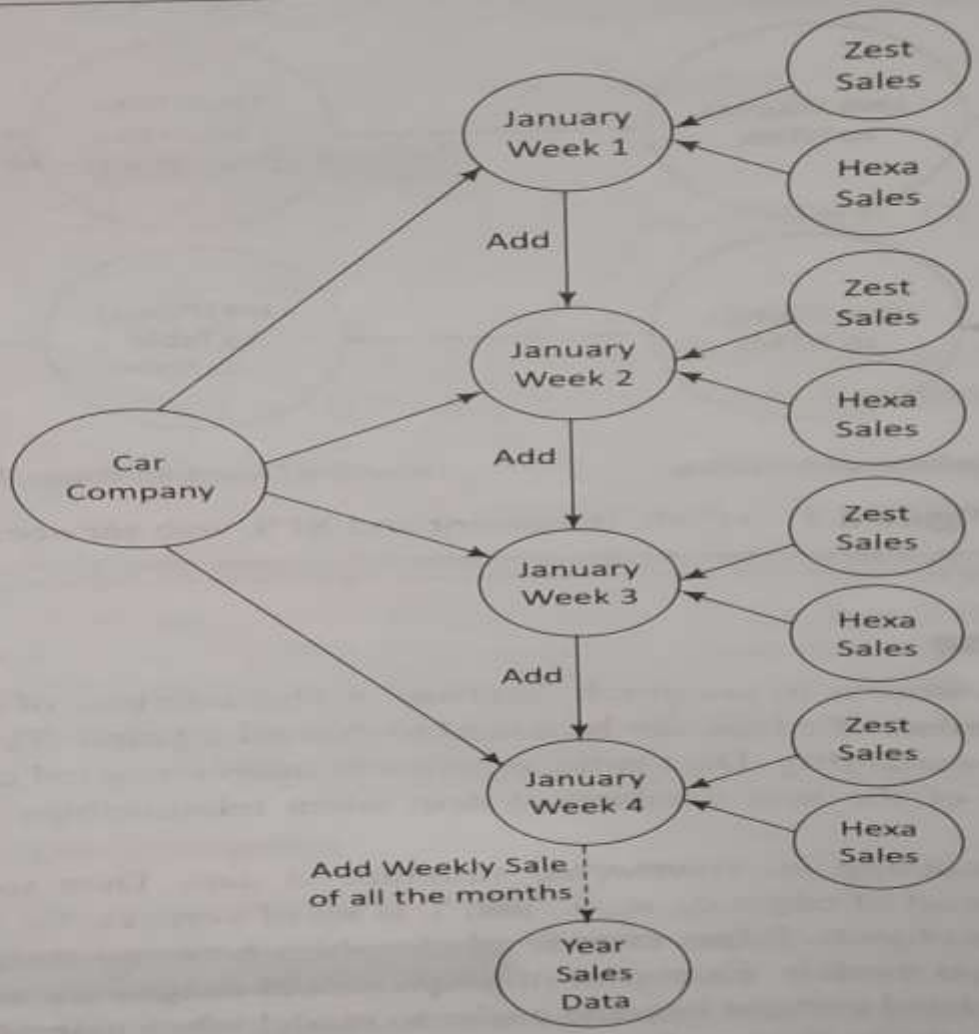


Figure 3.8 Section of the graph database for car-model sales

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

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.

NOSQL TO 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
- NoSQL data processing scales horizontally as well vertically

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, MongoDB or 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 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.

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.

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.

- 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.

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:

1. **Single Server Model**
2. **Sharding Very Large Databases**
3. **Master-Slave Distribution Model**
4. **Peer-to-Peer Distribution Model**

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.

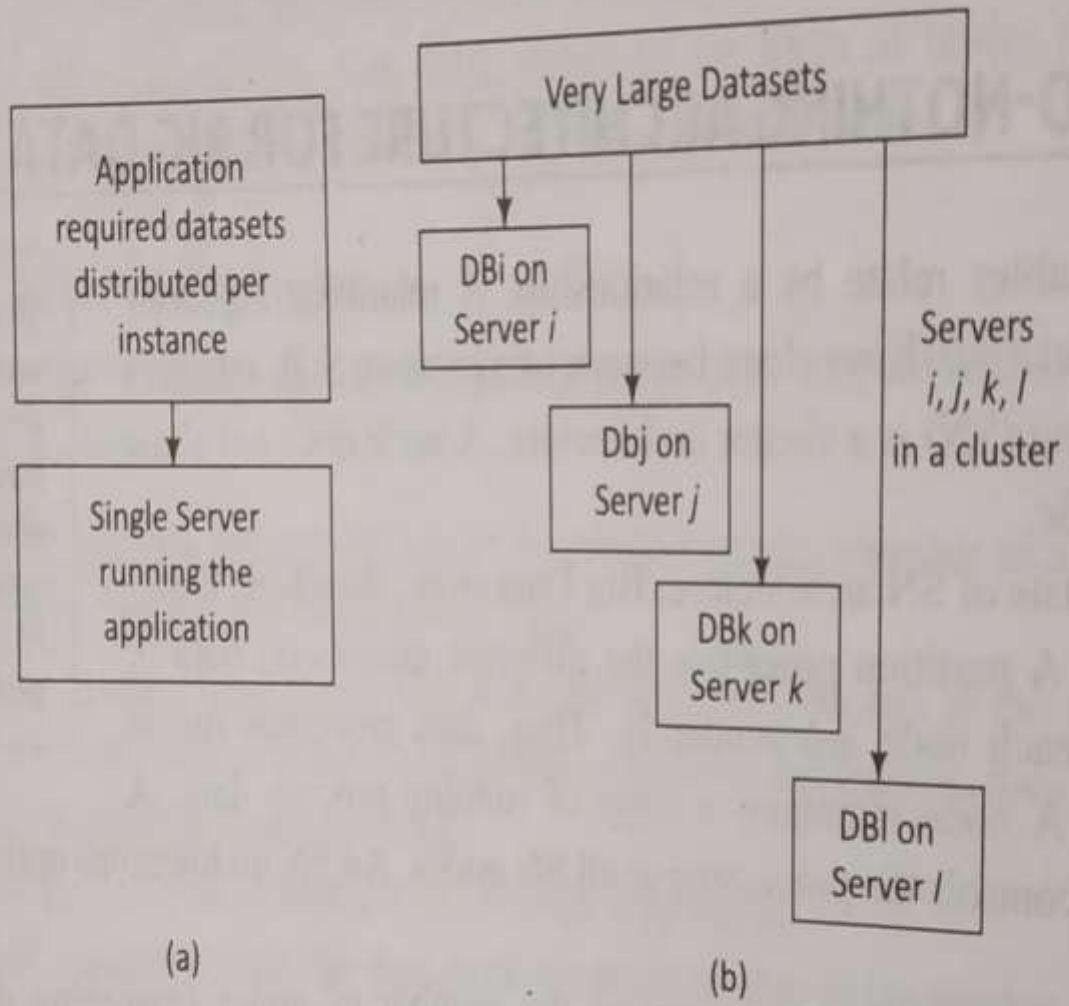


Figure 3.9 (a) Single server model (b) Shards distributed on four servers in a cluster.

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.

- 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.

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.***

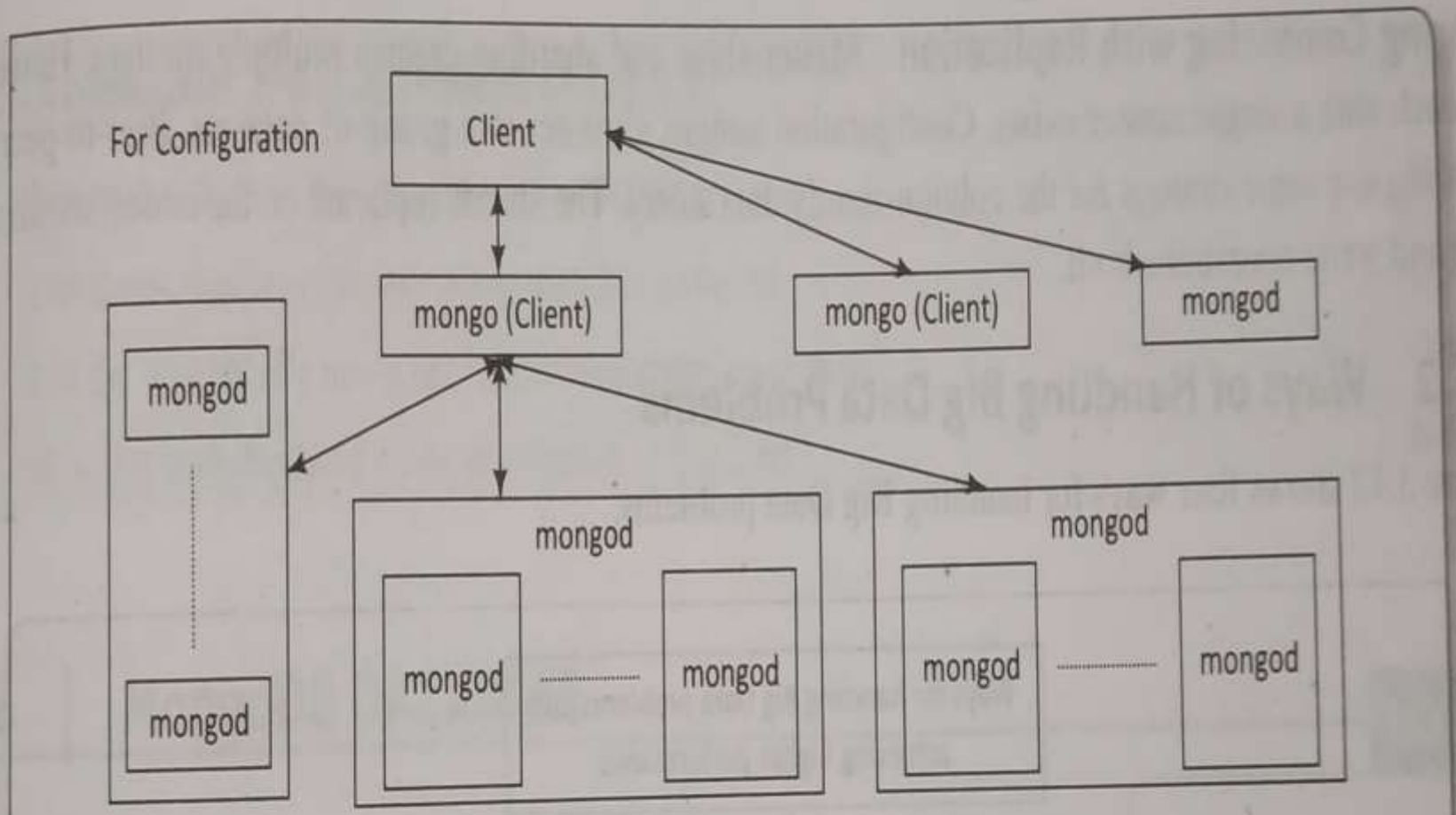


Figure 3.10 Master-slave distribution model. Mongo is a client and mongod is the server

- **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.

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.

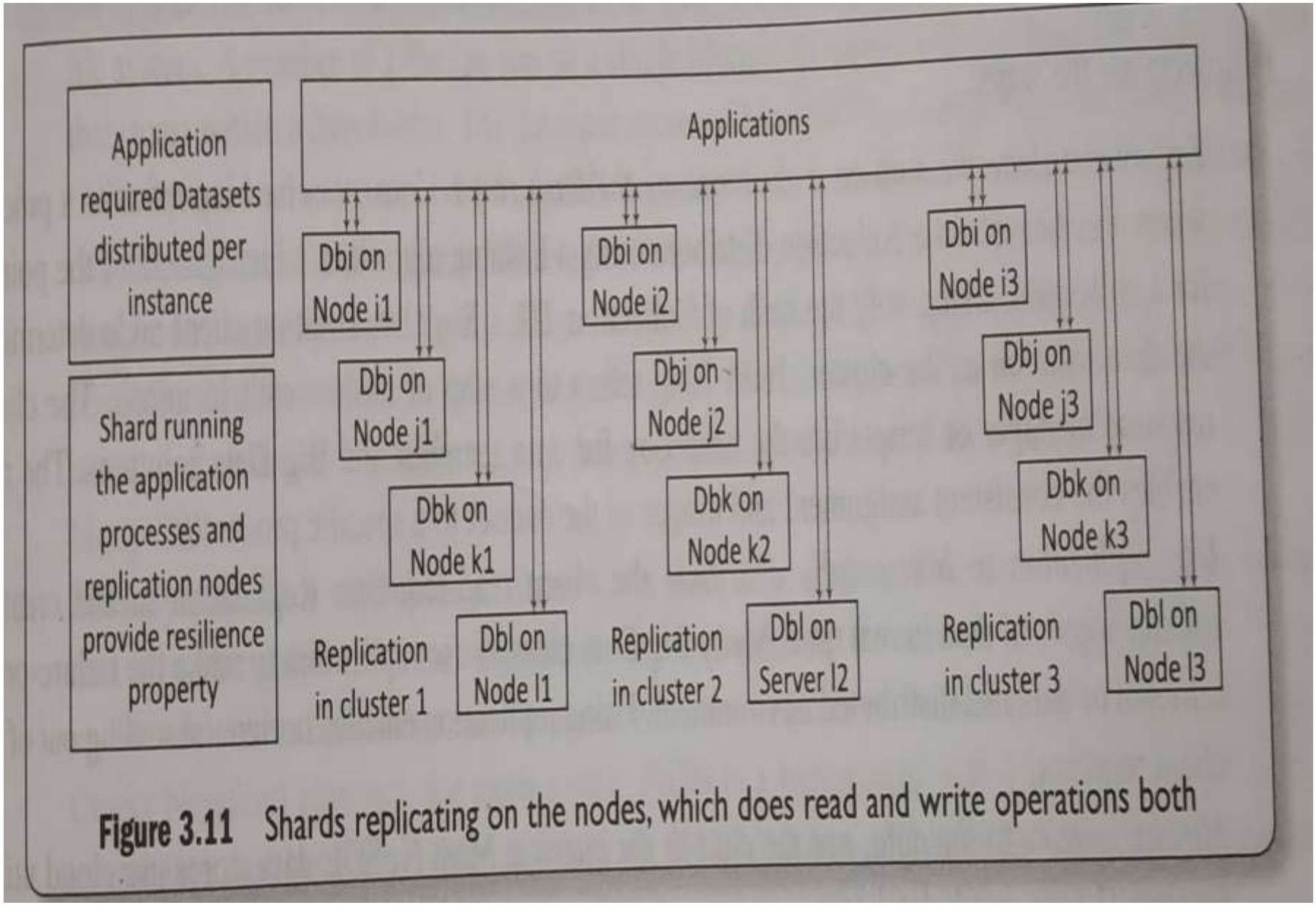


Figure 3.11 Shards replicating on the nodes, which does read and write operations both

Ways of Handling Big Data Problems

- Figure 3.12 shows four ways for handling Big Data problems.

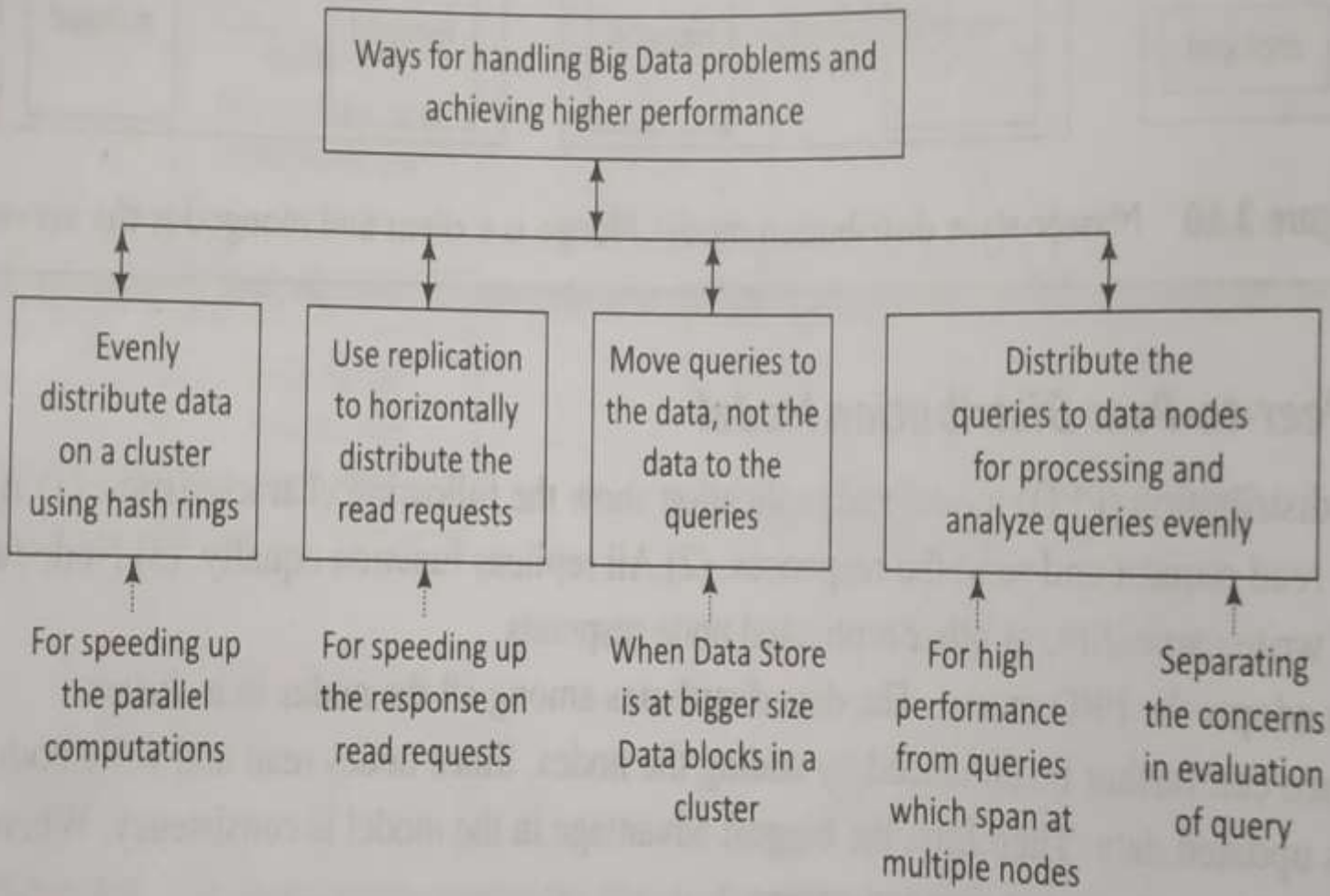


Figure 3.12 Four ways for handling big data problems

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).

- **Mongo Databases –**refer TEXT BOOK pdf uploaded in Microsoft teams, page from 217 to 226.
- **Cassandra Databases-** refer TEXT BOOK pdf uploaded in Microsoft teams, page from 227 to 237.