DATABASE MANAGEMENT SYSTEMS

Authors – Dan Sullivan, NoSQL for Mere Mortals.

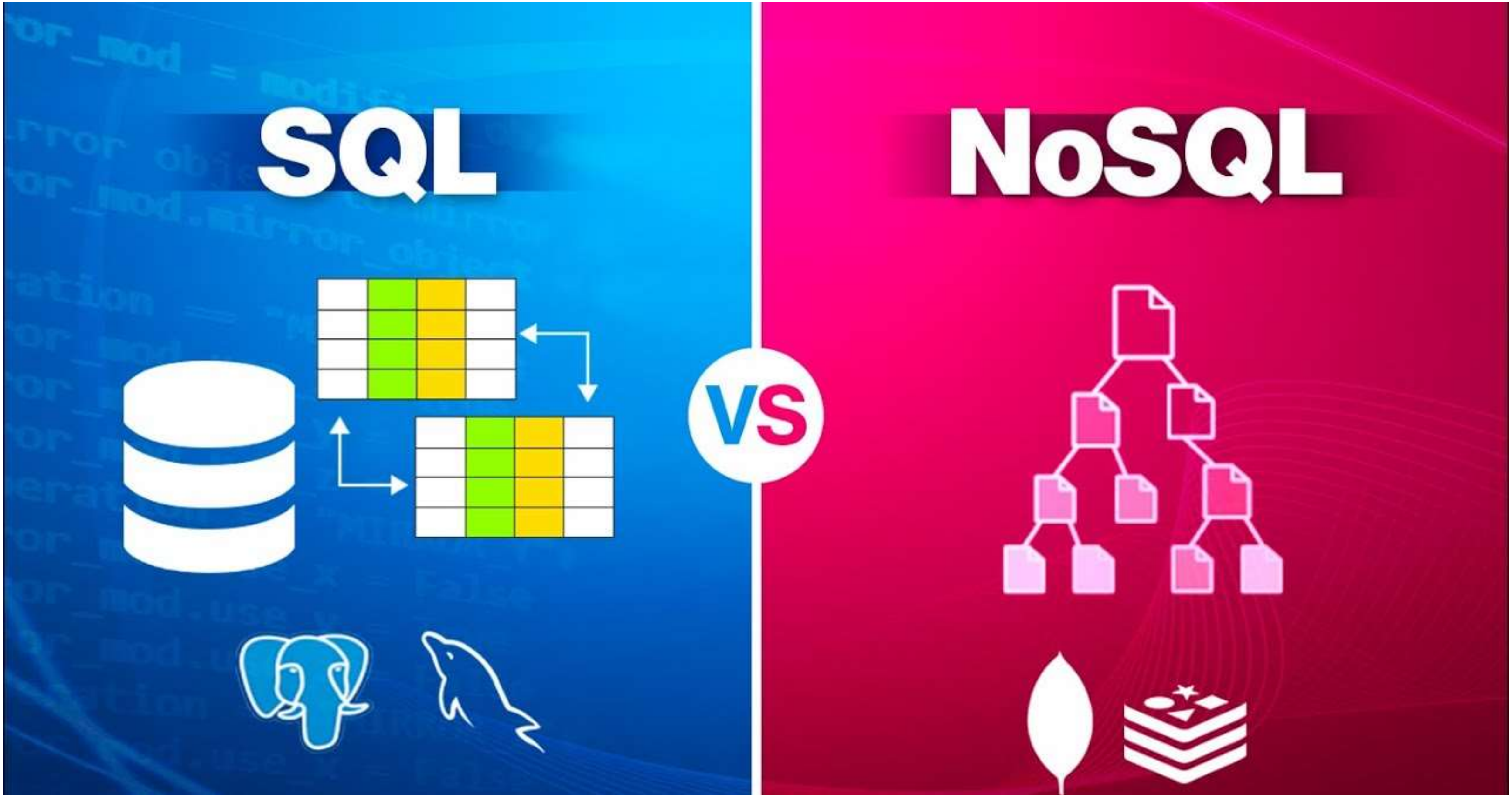
NO

SQL

UNIT-V

Motivations for Not Just/No SQL (No SQL) Databases, The CAP theorem, ACID and BASE, Types of NoSQL databases: Key –Value Pair Databases, Document databases, Column Family Databases, Graph Databases. Introduction to

Key-Value Databases, Key-Value terminology and Designing for the Key-Value Databases


SQL
VS
NoSQL

Introduction to NoSQL:

- NoSQL stands for Not Only SQL
- NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data.
- Unlike traditional relational databases that use tables with pre-defined schemas(table structures) to store data(i.e., not dependent on relational model/tables).
- NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data including **key- value, document, columnar, and graph formats.**
- Ex : MongoDB , Cassandra.

- NoSQL Database is used to refer a non – SQL or non relational database.

- It provides a mechanism for storage and retrieval of data other than tabular relations model used in relational databases. It is generally used to store big data and real-time web applications.

- The main drawback of relational database is, it cannot handle huge amount of data, and to overcome this issue we use NoSQL for fast performance and scalability of data handling capacity.

- Advantages:
    1. Supports SQL
    2. Fast performance
    3. Horizontal scalability

COMMON COMPARISONS BETWEEN

MYSQL & NOSQL

	MySQL	NoSQL
Nature	Relational Database	Non-Relational Database
Design	Based on the concept of tables	Based on the concept of documents
Scalable	Tough to scale due to its relational nature	Easily scalable big data compared to relational
Model	Detailed database model is needed before creation	No need of a detailed database model
Community	Vast community available	Community is growing rapidly, but still smaller compared to MySQL
Standardization	SQL is standard language	Lacks standard query language
Schema	The Schema is rigid	The Schema is dynamic
Flexibility	Not very flexible in terms of design	Very flexible in terms of design
Insertions	Inserting new columns or fields affect the design	No effect on the design with the insertion of new columns or fields

- Data structures used by NoSQL databases are sometimes also viewed as more complex to design and flexible to use than relational database tables.

- One simple example of a NoSQL database is a *document database*. In a document database, data is stored in documents rather than tables.

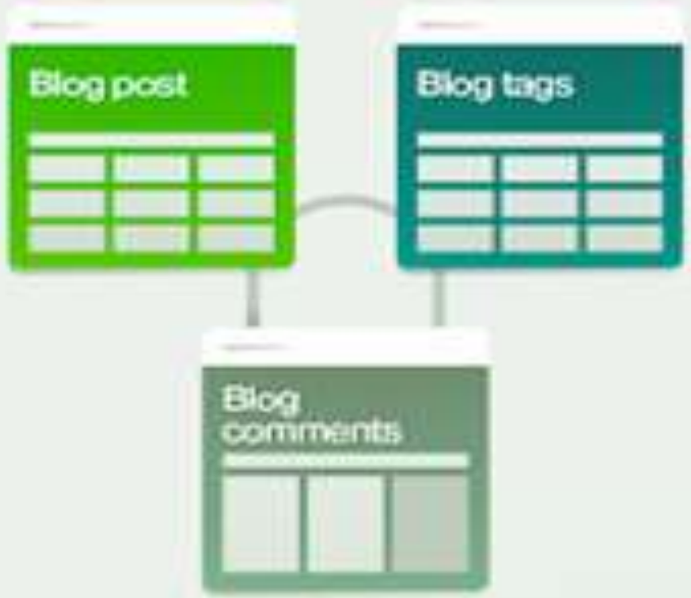
- Each document can contain a different set of fields, making it easy to accommodate changing data requirements
- Example::

"Take, for instance, a database that holds data regarding employees.". In a relational database, this information might be stored in tables, with one table for employee information and another table for department information. In a document database, each employee would be stored as a separate document, with all of their information contained within the document.

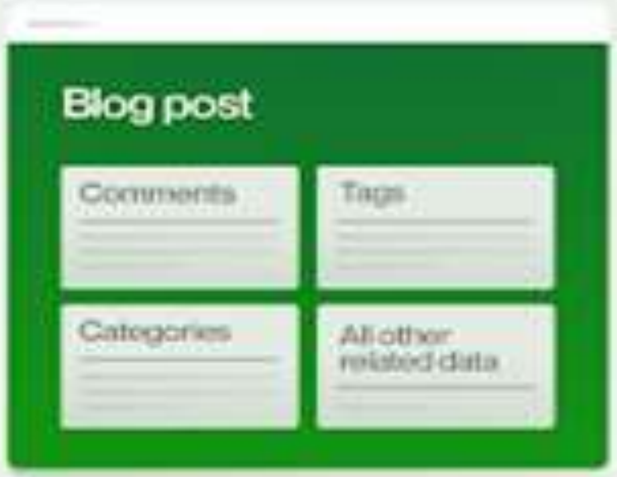
Relational databases

VS

Non-relational databases



A relational database organizes structured data fields into defined columns.



A non-relational database does not incorporate the table model. Instead, data ca be stored in a single document file.

KEY FEATURES OF NoSQL:

1.Dynamic schema: NoSQL databases do not have a fixed schema and can accommodate changing data structures without the need for migrations or schema alterations.

2.Horizontal scalability: NoSQL databases are designed to scale out by adding more nodes to a database cluster, making them well-suited for handling large amounts of data and high levels of traffic.

3.Document-based: Some NoSQL databases, such as MongoDB, use a document-based data model, where data is stored in a semi-structured format, such as JSON or BSON.

4.Key-value-based: Other NoSQL databases, such as Redis, use a key-value data model, where data is stored as a collection of key-value pairs.

5.Column-based: Some NoSQL databases, such as Cassandra, use a column-based data model, where data is organized into columns instead of rows.

6.Distributed and high availability: NoSQL databases are often designed to be highly available and to automatically handle node failures and data replication across multiple nodes in a database cluster.

7.Flexibility: NoSQL databases allow developers to store and retrieve data in a flexible and dynamic manner, with support for multiple data types and changing data structures.

9.Performance: NoSQL databases are optimized for high performance and can handle a high volume of reads and writes, making them suitable for big data and real-time applications.

Advantages of NoSQL:

There are many advantages of working with NoSQL databases such as MongoDB and Cassandra. The main advantages are high scalability and high availability.

1.Flexibility: NoSQL databases are designed to handle unstructured or semi-structured data, which means that they can accommodate dynamic changes to the data model. This makes NoSQL databases a good fit for applications that need to handle changing data requirements.

2.High availability: The auto, replication feature in NoSQL databases makes it highly available because in case of any failure data replicates itself to the previous consistent state.

3.Scalability: NoSQL databases are highly scalable, which means that they can handle large amounts of data and traffic with ease. This makes them a good fit for applications that need to handle large amounts of data or traffic.

4.Performance: NoSQL databases are designed to handle large amounts of data and traffic, which means that they can offer improved performance compared to traditional relational databases.

5.Cost-effectiveness: NoSQL databases are often more cost-effective than traditional relational databases, as they are typically less complex and do not require expensive hardware or software.

6.Agility: Ideal for agile development.

Disadvantages of NoSQL:

1.Lack of standardization: There are many different types of NoSQL databases, each with its own unique strengths and weaknesses. This lack of standardization can make it difficult to choose the right database for a specific application.

2.Lack of ACID compliance: NoSQL databases are not fully ACID-compliant, which means that they do not guarantee the consistency, integrity, and durability of data. This can be a drawback for applications that require strong data consistency guarantees.

3.Narrow focus: NoSQL databases have a very narrow focus as it is mainly designed for storage but it provides very little functionality. Relational databases are a better choice in the field of Transaction Management than NoSQL.

4.Open-source: NoSQL is an open-source database. There is no reliable standard for NoSQL yet. In other words, two database systems are likely to be unequal.

5.Lack of support for complex queries: NoSQL databases are not designed to handle complex queries, which means that they are not a good fit for applications that require complex data analysis or reporting.
6.Lack of maturity: NoSQL databases are relatively new and lack the maturity of traditional relational databases. This can make them less reliable and less secure than traditional databases.
7.Management challenge: The purpose of big data tools is to make the management of a large amount of data as simple as possible. But it is not so easy. Data management in NoSQL is much more complex than in a relational database. NoSQL, in particular, has a reputation for being challenging to install and even more hectic to manage on a daily basis.

8.GUI is not available: GUI mode tools to access the database are not flexibly available in the market.

9.Backup: Backup is a great weak point for some NoSQL databases like MongoDB. MongoDB has no approach for the backup of data in a consistent manner.

10.Large document size: Some database systems like MongoDB and CouchDB store data in JSON format. This means that documents are quite large (Big Data, network bandwidth, speed), and having descriptive key names actually hurts since they increase the document size.

Types of NoSQL database: Types of NoSQL databases and the name of the database system that falls in that category are:

1. **Graph Databases**: Examples – Amazon Neptune, Neo4j

2. **Key value store:** Examples – Memcached, Redis, Coherence

3. **Column:** Examples – Hbase, Big Table, Accumulo

4. **Document-based:** Examples – MongoDB, CouchDB, Cloudant

Further be explained in detail..

As we got familiar to the introduction part , We shall move on to the first topic of our syllabus..

TOPIC - 1

Motivations for Not Just/No SQL(No SQL) Databases

Motivations for Not Just/No SQL (No SQL) Databases

- ➢ The data management professionals and software designers are got motivated of real world problems related to huge data management in the case of web applications serving many users.

- ➢ It was difficult to manage that much amount of data with relational databases, which motivated the professionals to develop NoSQL.

- ➢ Four characteristics of data management systems that are particularly important for large-scale data management tasks are
  - Scalability
  - Cost
  - Flexibility
  - Availability

Scalability:

- Scalability is the ability to efficiently meet the needs for varying workloads.

- **Vertical Scalability/Scaling Up:** When the load increases on the server, more power (CPU, RAM, Storage) is added i.e., replacing the existing server with more load bearable server according to the requirement. SQL/Relational databases works on vertical scalability.
- **Horizontal Scalability/Scaling Out:** New nodes or machines are added to existing server network in the case of high load and can be shut down if load is in control. NoSQL databases works on Scaling out.
- Note:

  SQL : Vertical Scalability/Scaling Up

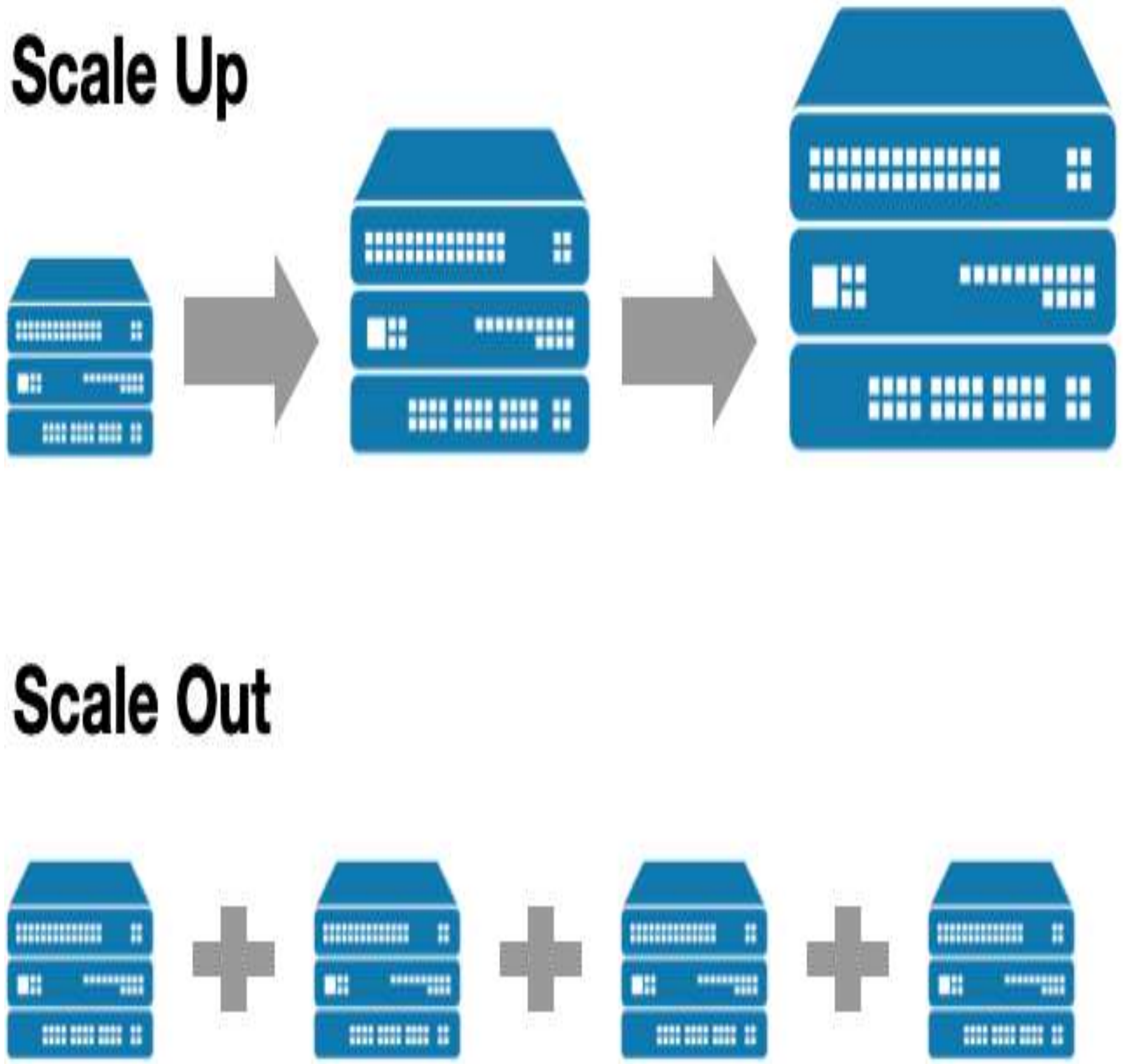
  NoSQL : Horizontal Scalability / Scaling Out

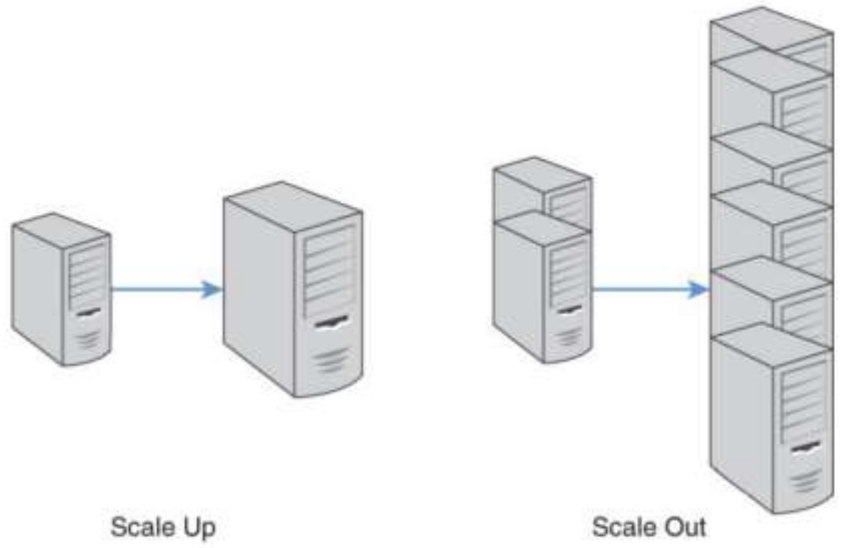
Horizontal Scalability is more efficient than vertical.

- NoSQL databases are designed to utilize servers available in a cluster with *minimal intervention by database administrators*. As new servers are added or removed, the NoSQL database management system *adjusts to use the new set* of available servers.

- Scaling up by replacing a server requires migrating the database management to a new server.
- Scaling out by adding resources would not require a migration, but would likely require some downtime to add hardware to the database server.

- When you work with relational databases(SQL), it is often challenging to scale out. Additional database software may be needed to manage multiple servers working as a single database system. Additional database components can add complexity and cost to operations.

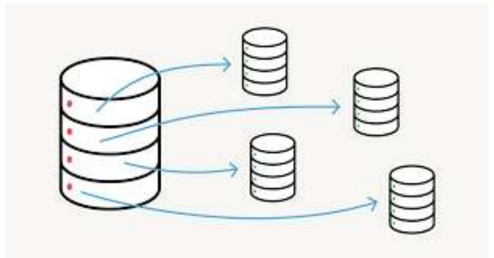

Scale Up
Scale Out


Scale Up
Scale Out

Advantages of Scalability:

1.Distribution of Data : NoSQL databases distribute data across multiple nodes in a cluster, allowing for parallel processing and reducing the load on individual machines. This distribution can be based on various strategies such as sharding, replication, or partitioning.

2. Partitioning/Sharding: Many NoSQL databases support partitioning or sharding, where data is divided into smaller subsets called partitions or shards. Each shard can then be stored and processed independently on different nodes in the cluster. This allows for better utilization of resources and improved performance.



3.Load Balancing: NoSQL databases typically incorporate mechanisms for load balancing, ensuring that requests are evenly distributed across nodes in the cluster. This prevents any single node from becoming a bottleneck and helps maintain consistent performance as the system scales.

4.Replication: Replication is commonly used in NoSQL databases to improve fault tolerance and availability. By replicating data across multiple nodes, the system can continue to function even if some nodes fail. Replication also enables read scaling, allowing multiple nodes to serve read requests simultaneously.

Cost:

- The cost of database licenses is an obvious consideration for any business or organization

- Commercial software vendors employ a variety of licencing models based on the size of the server running the RDBMS.

- Web applications may have increase or decrease of requirement of the size of the server based on the trend of users, no.of concurrent users on the database, or by the number of named users allowed to use the software.
- Should the users pay for maximum no. of users or average no. of uses?

- Users of open source software avoid these issues.

- The software is free to use on as many servers of whatever size needed because open source developers do not typically charge fees to run their software

- Fortunately for NoSQL database users, the major NoSQL databases are available as open source.

Flexibility:

- In RDBMS, a database designer expect to know at the start of a project all the tables and columns that will be needed to support an application.

- For example, an e-commerce application should contain different tables for different items such as laptop table contain processor, ram, speed etc. as attributes, a fridge table has capacity, power consumption etc.. as attributes.
- Not every attribute be used by every table/product as different products have different attributes.
- Unlike relational databases, some NoSQL databases do not require a fixed table structure. For example, in a document database, a program could dynamically add new attributes as needed without having to have a database designer alter the database design

Availability:

- NoSQL databases are designed to take advantage of multiple, low-cost servers. When one server fails or is taken out of service for maintenance, the other servers in the cluster can take on the entire workload.

- Backup servers keep replicated copies of data from the primary server in case the primary server fails.

- If that happens, the backup can take on the workload that the primary server had been processing. This can be an inefficient configuration because a server is kept in reserve in the event of a failure but otherwise is not helping to process the workload.

- As NoSQL databases run with multiple servers and there will be no interruption to data availability in the case of a server fails, remaining servers distribute the load among themselves.

- NoSQL databases are unlikely to displace relational databases the way RDBMSs displaced flat file, hierarchical, and network databases.

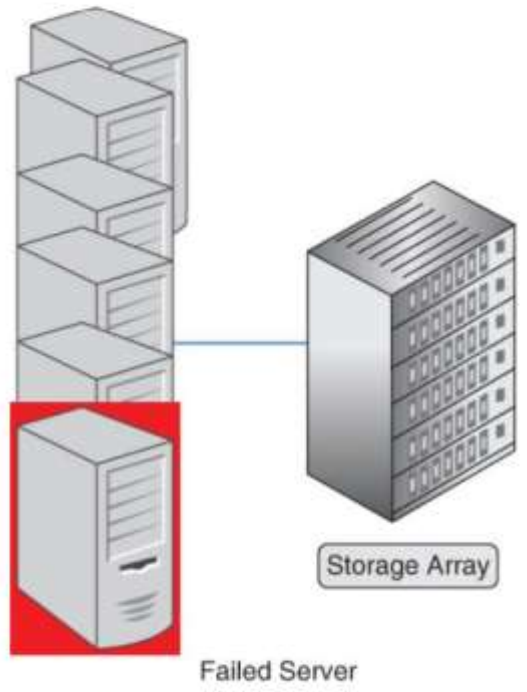


Figure 1.12 *High-availability NoSQL clusters run multiple servers. If one fails, the others can continue to support applications.*

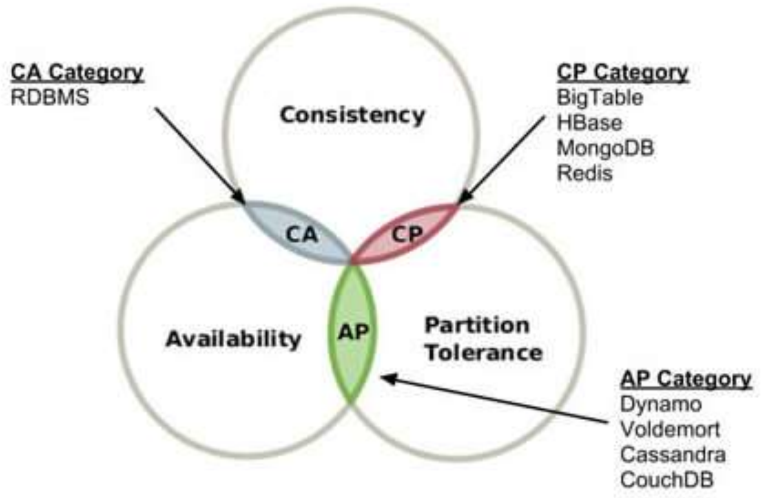
Topic 1 completed

TOPIC - 2

Consistency, Availability, and Partitioning: The CAP Theorem

CAP

Theorem



- The CAP theorem, also known as Brewer's theorem.

- **Statement:** CAP theorem states that distributed databases cannot have consistency (C), availability (A), and partition protection (P) all at the same time with data replication.

- *Terminology:*

1. **Consistency (C)**: All nodes in the distributed system have the same data at the same time. In other words, when a data update is performed, all subsequent reads will reflect that update.

2. **Availability (A)**: Every request received by a non-failing node in the system must result in a response, regardless of the state of the system. This means that the system continues to function even if some nodes fail.

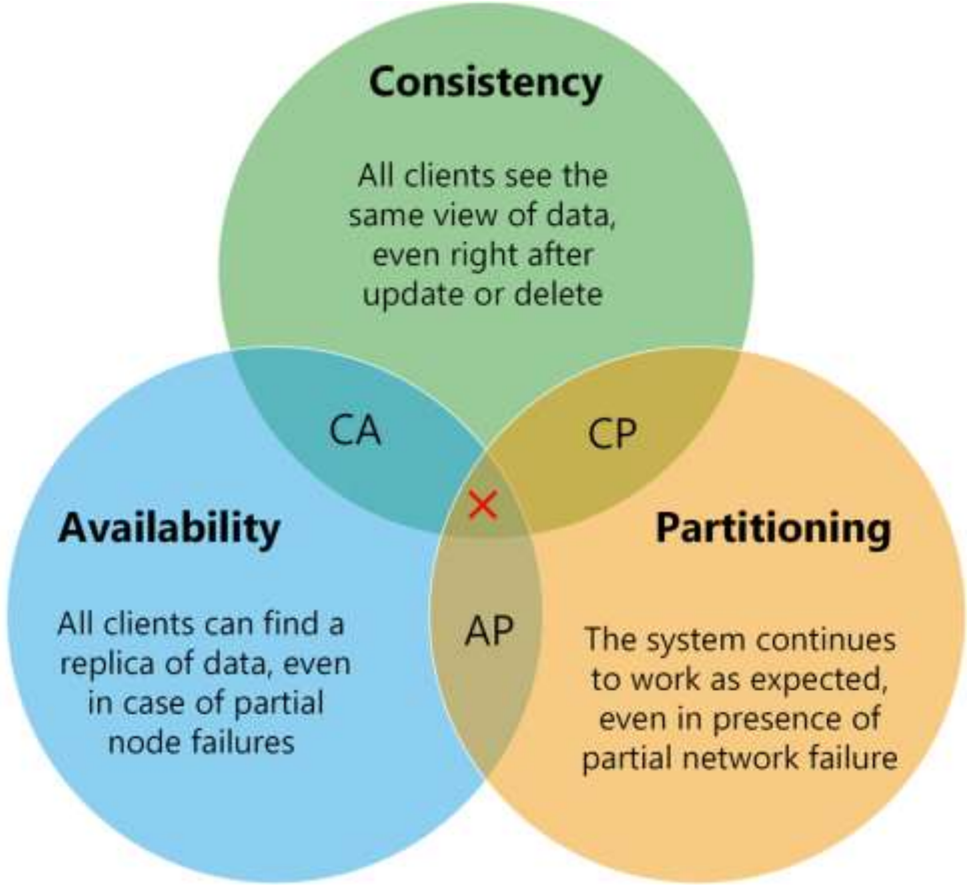
3.Partition tolerance (P): The system continues to operate despite network partitions (communication failures) that may occur between nodes, which may cause messages to be lost or delayed.

The following are the possible three combinations:

1.CA(consistent, available) → (SQL)
2.CP(consistent, partition tolerant) } No SQL
3.AP(available, partition tolerant) }

But CAP theorem forces us to choose only two combinations at a time i.e., CP,AP in document type databases

Let's see the reason 🤔


Consistency
All clients see the same view of data, even right after update or delete
CA
CP
×
Availability
All clients can find a replica of data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure

- If database servers running the same distributed database are partitioned by a network failure, then you could continue to allow both to respond to queries and preserve availability but at the risk of them becoming inconsistent as data copying/sharding may take time and between these transactions, it is inconsistent.

- Alternatively, you could disable one so that only one of the servers responds to queries. This would avoid returning inconsistent data to users querying different servers but we cannot say that it always be available because there is a chance of failure of that server.

- So, it is difficult to maintain both consistency and availability at a time in document database.

AP (Available and Partition Tolerance): - read and write conflicts

Sacrifices consistency to ensure both availability and partition tolerance. This means that the system remains available and can tolerate network partitions, but consistency might be relaxed, leading to eventual consistency.

Examples include systems like DynamoDB or Cassandra.

Explanation:

- ➢ When there are multiple servers, the database can be available even if one server fails. ✅
- ➢ But data consistency may not be maintained at the time of server failure.☹
- ➢ The Server 1 may fail without updating the most recent data into server 2(a copy server)
- ➢ Then the recent data may not be available in Server 2. But database will not become unavailable.

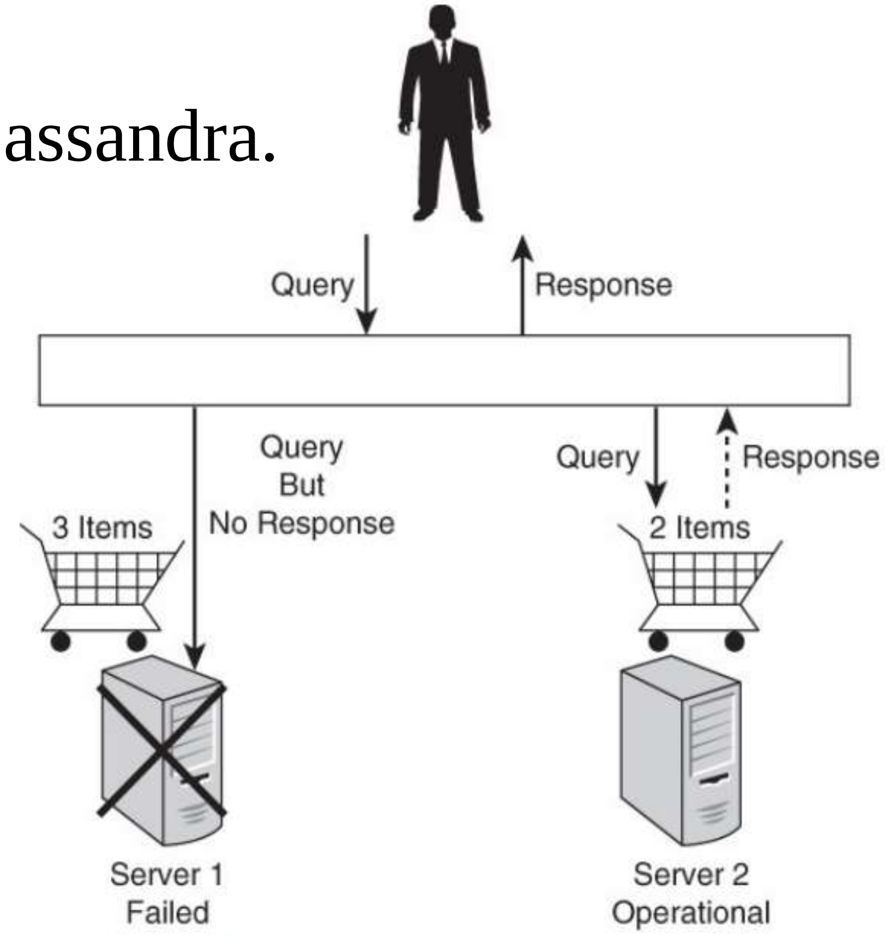


Figure 2.8 *Data can be available but not consistent.*

CP (Consistency and Partition Tolerance):

Sacrifice availability to ensure both consistency and partition tolerance. This means that the system remains consistent and can tolerate network partitions but may not always be available.

Examples include systems like MongoDB with strong consistency guarantees.

Explanation:

- To maintain the data consistency i.e., same data at every data node, it can make unavailable of data to users until each node becomes consistent.
- Until write program updates data on each and every server, all read requests are blocked
- Hence, not be available always

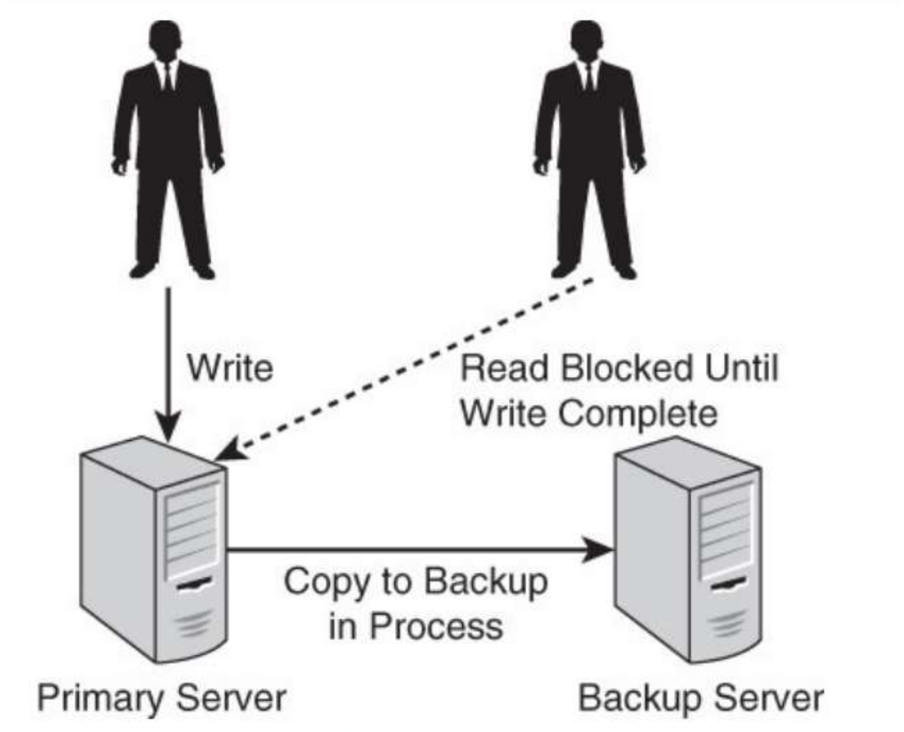


Figure 2.9 *Data can be consistent but not available.*

Choosing the Right Trade-off:
The choice between CP and AP depends on the specific needs of your application. Here's a guideline:

•**Choose CP if:** Data consistency is critical, and occasional unavailability during network partitions is acceptable (e.g., financial transactions).

•**Choose AP if:** System availability is essential, and some temporary inconsistency during network issues is tolerable (e.g., social media platforms).

Topic 2 Completed

TOPIC - 3
ACID and BASE

ACID:

- Atomicity
- Consistency
- Isolation
- Durability

BASE:

- Basically Available
- Soft State
- Eventually Consistent

- ➢ We have already seen ACID properties in transaction management.
- ➢ Let’s have a glimpse of ACID properties:

- **Atomicity**: Guarantees that all operations within a transaction are completed successfully, or none at all, ensuring transactions are indivisible.

- **Consistency**: Ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants.

- **Isolation**: Enables transactions to operate independently of and transparently to each other, as if they are the only ones in the system.

- **Durability**: Ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.

- Relational database management systems are designed to support ACID transactions i.e., for SQL.

- NoSQL databases typically support BASE transactions, although some NoSQL databases also provide some level of support for ACID transactions.

BASE: Basically Available, Soft state, Eventually consistent(for NoSQL)

- **1.Basically Available(BA):** This means that there can be a partial failure in some parts of the distributed system and the rest of the system continues to function.

- NoSQL databases often keep multiple copies of data on different servers. This allows the database to respond to queries even if one of the servers has failed but with degraded functionality.

- ➢ Imagine an e-commerce platform. During peak sales, the system might experience slower response times or temporary limitations on adding items to carts, but it should still be available for browsing and essential functions.

- ➢ **2.Soft State(S):** It refers to the fact that data may eventuall with more recent data.

- ➢ Data across different nodes (copies) in the system might not be identical at all times. This allows for faster read operations and improved scalability but introduces potential inconsistency.

- ➢ Consider a social media platform where user feeds might take a few seconds to update with the latest posts after a friend adds a new one. This temporary inconsistency prioritizes responsiveness.

3.Eventual Consistency:

- This means that there may be times when the database is in an inconsistent state.

- There is, however, a possibility that the multiple copies may not be consistent for a short period of time. This can occur when a user or program updates one copy of the data and other copies continue to have the old version of the data.

- Eventually, the replication mechanism in the NoSQL database will update all copies, but in the meantime, the copies are inconsistent.

- The time it takes to update all copies depends on several factors, such as the load on the system and the speed of the network.

Example:

- Consider a database that maintains three copies of data. A user updates her address in one server.
- The NoSQL database management system automatically updates the other two copies.
- One of the other copies is on a server in the same local area network, so the update happens quickly.
- The other server is in a data center thousands of miles away, so there is a time delay in updating the third copy.
- A user querying the third server while the update is in progress might get the user’s old address while someone querying the first server gets the new address.

- Types of Eventual Consistency:

- Casual consistency
- Read-your-writes consistency (Every read output will be updated value, no inconsistency)
- Session consistency
- Monotonic read consistency
- Monotonic write consistency

- Casual Consistency :Casual consistency ensures that the database reflects the order in which operations were updated. (Order of updates = Order of operations)
- For example, if Alice changes a customer’s outstanding balance to $1,000 and one minute later Bob changes it to $2,000, all copies of the customer’s outstanding balance will be updated to $1,000 before they are updated to $2,000.

- Read-Your-Writes Consistency: Read-your-writes consistency means that once you have updated a record, all of your reads of that record will return the updated value.

- You would never retrieve a value inconsistent with the value you had written.

- Let’s say Alice updates a customer’s outstanding balance to $1,500. The update is written to one server and the replication process begins updating other copies.

- During the replication process, Alice queries the customer’s balance. She is guaranteed to see $1,500 when the database supports read-your-writes consistency.

- Session Consistency: Session consistency ensures read-your-writes consistency during a session.

- You can think of a session as a conversation between a client and a server or a user and the database. As long as the conversation continues, the database “remembers” all writes you have done during the conversation.

- If the session ends and you start another session with the same server, there is no guarantee it will “remember” the writes you made in the previous session.

- A session may end if you log off an application using the database or if you do not issue commands to the database for so long that the database assumes you no longer need the session and abandons it.

- Monotonic Read Consistency: Monotonic read consistency ensures that if you issue a query and see a result, you will never see an earlier version of the value.

- Let’s assume Alice is yet again updating a customer’s outstanding balance. The outstanding balance is currently $1,500. She updates it to $2,500.

- Bob queries the database for the customer’s balance and sees that it is $2,500.

- If Bob issues the query again, he will see the balance is $2,500 even if all the servers with copies of that customer’s outstanding balance have not updated to the latest value.

- Monotonic write consistency: It is a consistency model that ensures that write operations issued by a single client are applied in the same order they were issued.

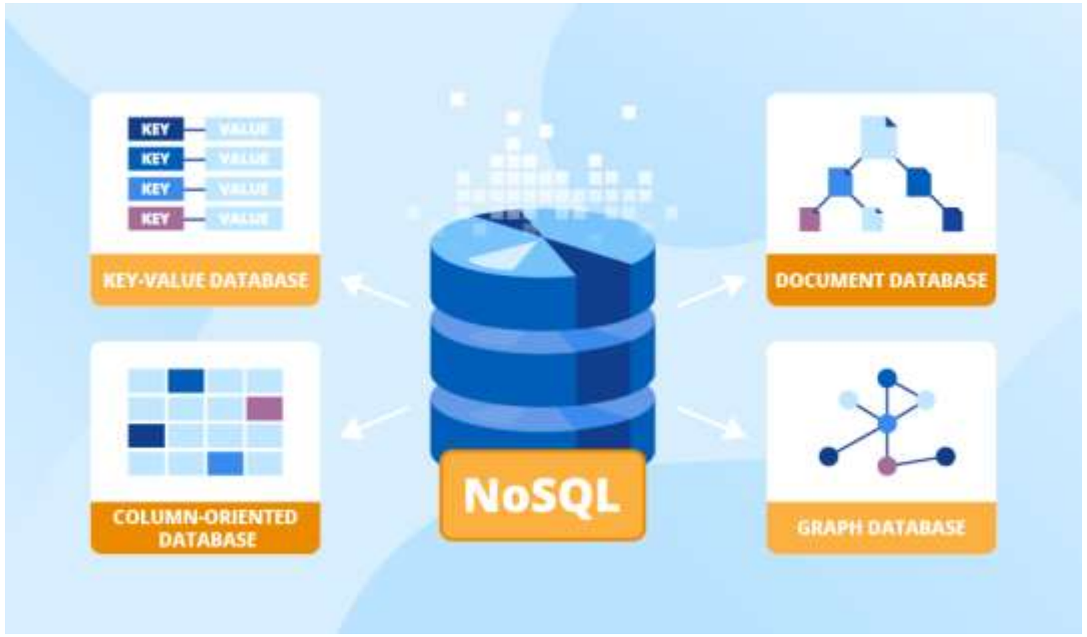
- This means that if a client makes two write operations, W_1 followed by W_2 the system guarantees that W_1 will be applied before W_2, regardless of the replicas involved or potential system failures.

- This model is crucial for maintaining logical order in updates to a distributed system, preventing scenarios where a later update is overwritten by an earlier one due to delays or network partitioning.
- Monotonic write consistency helps ensure that the system's state progresses in a predictable and orderly manner, reflecting the sequence of operations as intended by the client.

Topic 3 Completed

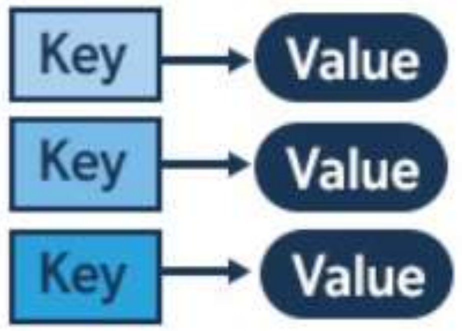
TOPIC - 4
Types of NoSQL databases:

1. Key –Value Pair Databases
2. Document databases
3. Column Family Databases
4. Graph Databases



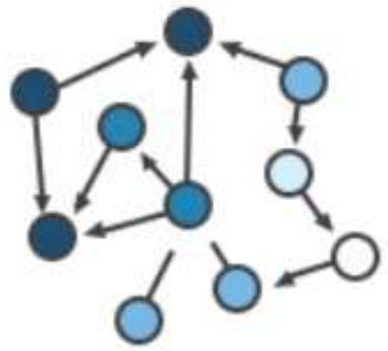
NoSQL

Key-Value



Column-Family

Graph



Document

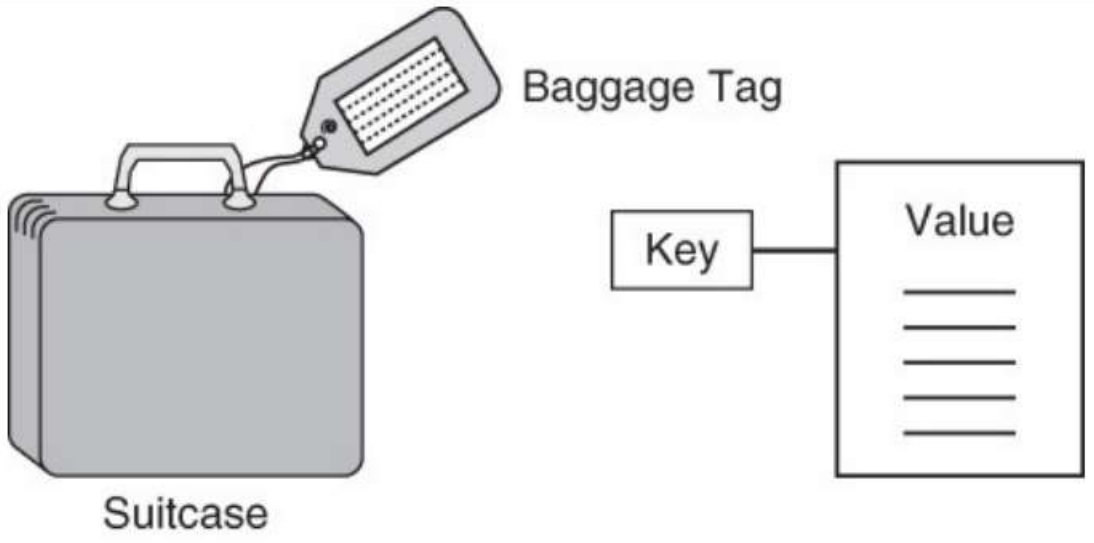
Types of non-relational databases

Types	Performance	Scalability	Flexibility	Complexity
Key-value store	High	High	High	None
Column store	High	High	Moderate	Low
Document	High	High	High	Low
Graph database	Variable	Variable	High	High

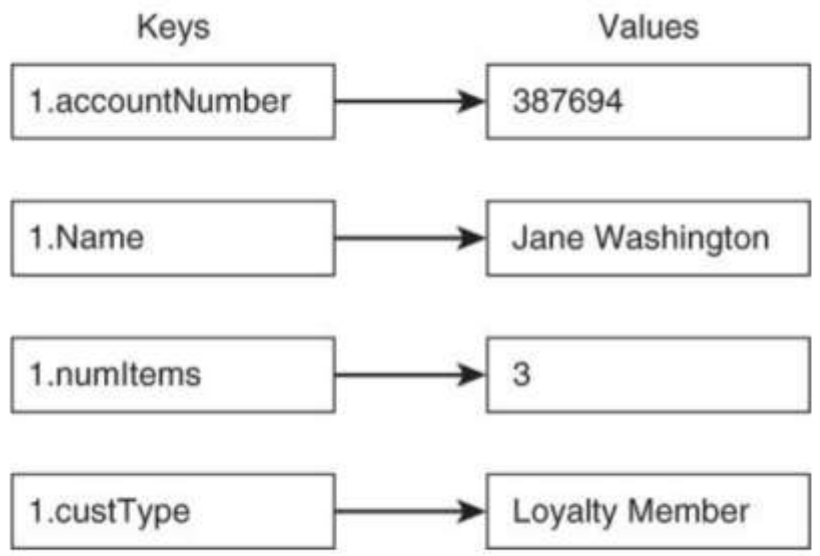
1.Key-Value Pair Databases

- Key-value pair databases are the simplest form of NoSQL databases. These databases are modeled on two components: keys and values

- Every data element in the database is stored in key-value pairs. The data can be retrieved by using a unique key allotted to each element in the database. The values can be simple data types like strings and numbers or complex objects.



- A key-value store is like a relational database with only two columns which is the key and the value.
- Key features of the key-value store:
  - Simplicity.
  - Scalability.
  - Speed.
- Keys are identifiers associated with values. Values are data stored along with keys.
- Values can be as simple as a string, such as a name, or a number, such as the number of items in a customer’s shopping cart. You can store more complex values, such as images or binary objects too.



.11 *Key-value databases are modeled on a simple, two-part data structure consisting of an identifier and a data value.*

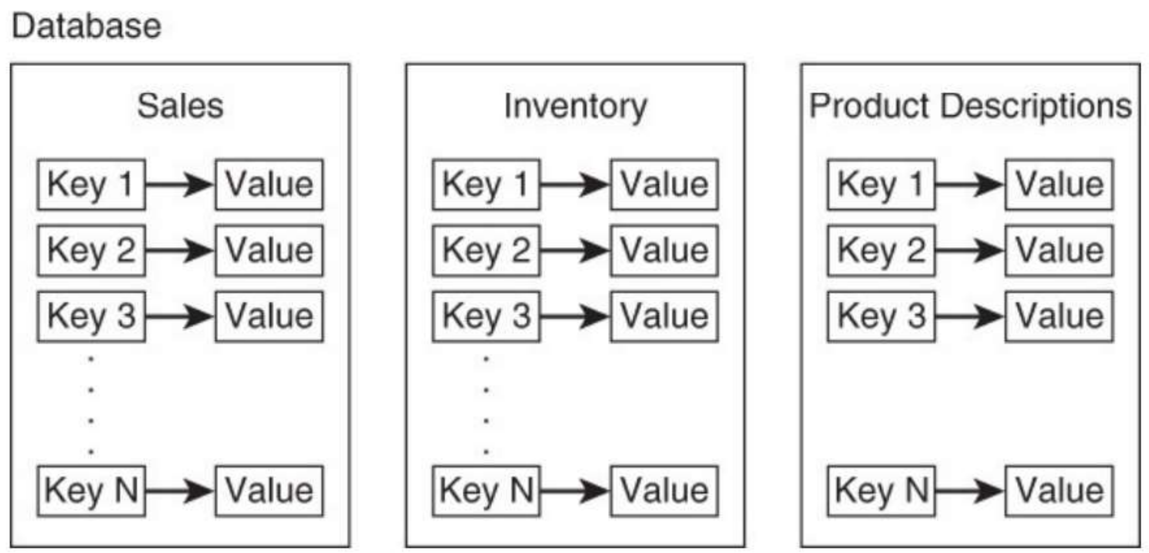


Figure 2.12 *Key-value databases may support separate namespaces within a single database.*

- cust1.accountNumber
- cust1.name
- cust1.address
- cust1.numItems
- cust1.custType
- cust2.accountNumber
- cust2.name
- cust2.address
- cust2.numItems
- cust2.custType

d so on. Similarly, the keys for the warehouse data would be

- wrhs1.number
- wrhs1.address
- wrhs2.number
- wrhs2.address

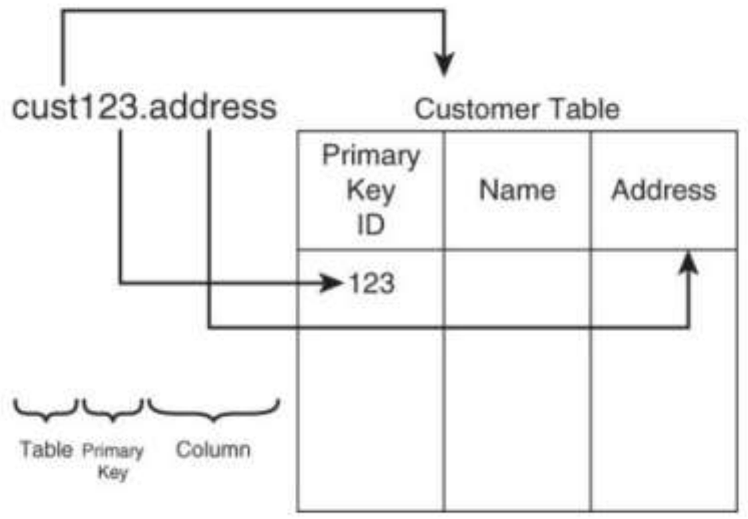


Figure 2.13 *The key-naming convention outlined above maps to patterns seen in relational database tables.*

← Difference b/w rdbms and key-value database

2.Document-based databases

- A document database stores values as documents. In this case, documents are semi-structured entities, typically in a standard format such as JavaScript Object Notation (JSON) or Extensible Markup Language (XML).

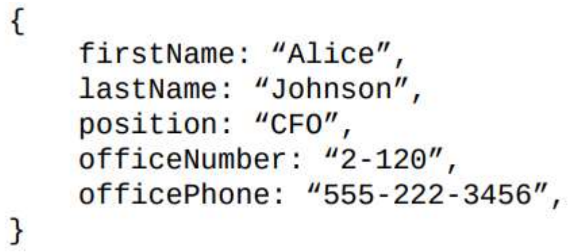
- Documents can be stored and retrieved in a form that is much closer to the data objects used in applications which means less translation is required to use these data in the applications.

- In the Document database, the particular elements can be accessed by using the index value that is assigned for faster querying.

- ➢ Collections are the group of documents that store documents that have similar contents. Not all the documents are in any collection as they require a similar schema because document databases have a flexible schema.
- ➢ Key features of documents database:

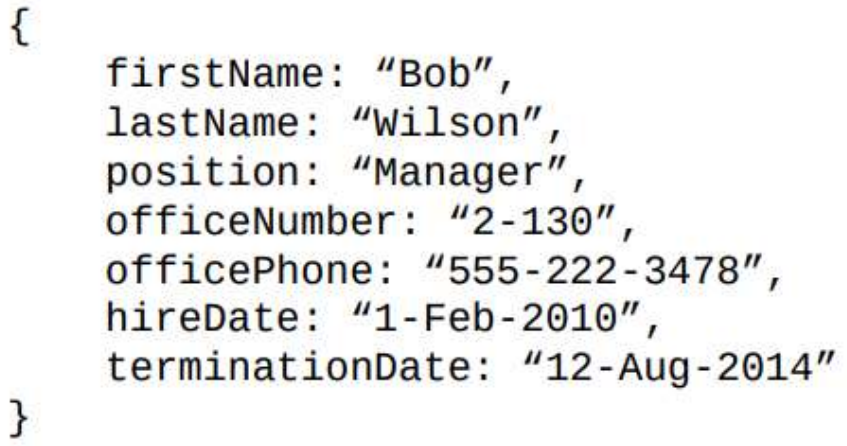
•<u>Flexible schema</u>: Documents in the database has a flexible schema. It means the documents in the database need not be the same schema.
•<u>Faster creation and maintenance</u>: the creation of documents is easy and minimal maintenance is required once we create the document.
•<u>No foreign keys</u>: There is no dynamic relationship between two documents so documents can be independent of one another. So, there is no requirement for a foreign key in a document database.
•<u>Open formats</u>: To build a document we use XML, JSON, and others.

- Documents Instead of storing each attribute of an entity with a separate key, document databases store multiple attributes in a single document. Here is a simple example of a document in JSON format:



```
{
    firstName: “Alice”,
    lastName: “Johnson”,
    position: “CFO”,
    officeNumber: “2-120”,
    officePhone: “555-222-3456”,
}
```

- The lack of a fixed schema gives developers more flexibility with document databases than they have with relational databases. For example, employees can have different attributes than the ones listed above.

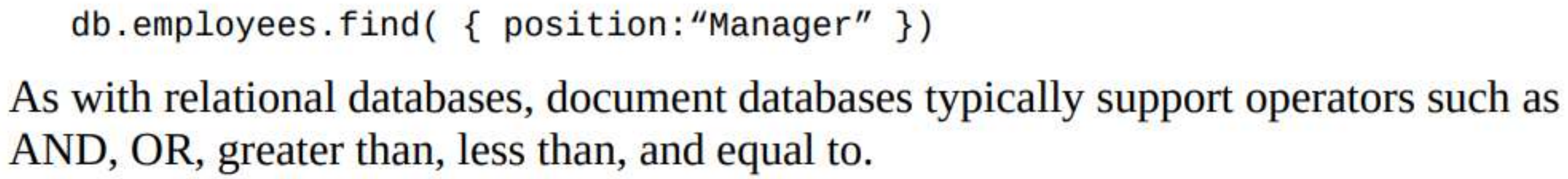


```
{
    firstName: “Bob”,
    lastName: “Wilson”,
    position: “Manager”,
    officeNumber: “2-130”,
    officePhone: “555-222-3478”,
    hireDate: “1-Feb-2010”,
    terminationDate: “12-Aug-2014”
}
```

- We can’t store JSON,XML documents in key-valued databases because key-value databases have few restrictions on the type of data stored as a value, you could store a JSON document as a value. The only way to retrieve such a document is by its key, however.

- Document databases provide application programming interfaces (APIs) or query languages that enable you to retrieve documents based on attribute values.

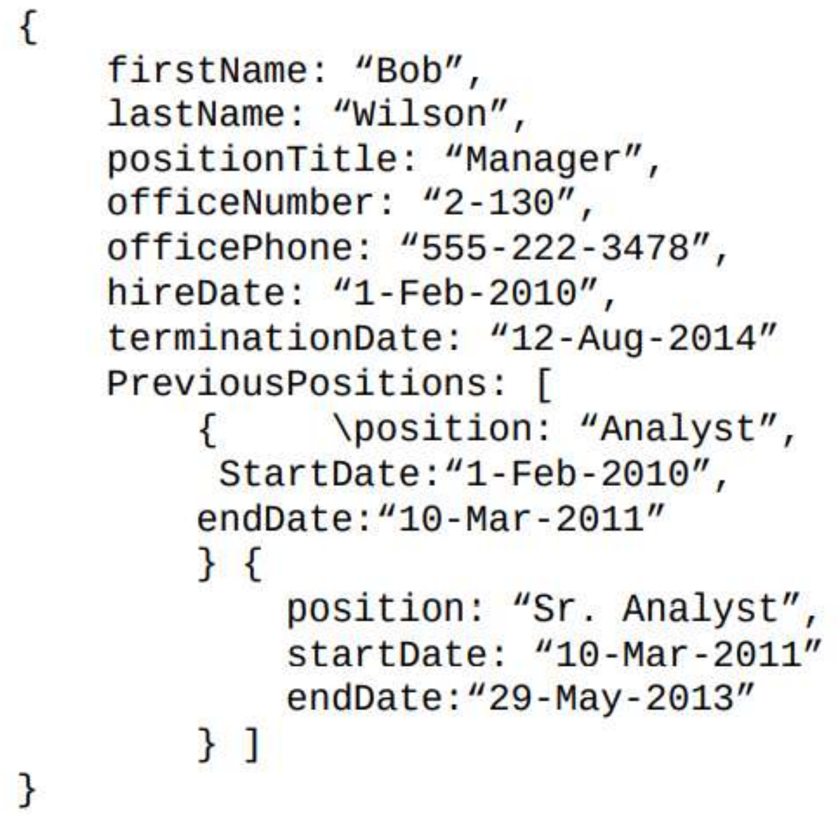
- For example, if you have a database with a collection of employee documents called “employees,” you could use a statement such as the following to return the set of all employees with the position Manager:



```
db.employees.find( { position:“Manager” })
```

As with relational databases, document databases typically support operators such as AND, OR, greater than, less than, and equal to.

- The difference between document based and relational based is that documents can have embedded documents and lists of multiple values within a document.
- For example, the employee documents might include a list of previous positions an employee held within the company. For example:



```
{
    firstName: “Bob”,
    lastName: “Wilson”,
    positionTitle: “Manager”,
    officeNumber: “2-130”,
    officePhone: “555-222-3478”,
    hireDate: “1-Feb-2010”,
    terminationDate: “12-Aug-2014”
    PreviousPositions: [
        {     \position: “Analyst”,
         StartDate:“1-Feb-2010”,
        endDate:“10-Mar-2011”
        } {
            position: “Sr. Analyst”,
            startDate: “10-Mar-2011”
            endDate:“29-May-2013”
        } ]
}
```

- Embedding documents or lists of values in a document eliminates the need for joining documents the way you join tables in a relational database.

- If you stored a list of document identifiers in a document and want to look up attributes in the documents associated with those identifiers, then you would have to implement that operation in your program.

3.Column Family Databases

- Column family databases are perhaps the most complex of the NoSQL database types, atleast in terms of the basic building block structures.
- Column family databases share some terms with relational databases, such as rows and columns, but you must be careful to understand important differences between these structures.
- **Columns and Column Families**

A column is a basic unit of storage in a column family database. A column is a name and a value

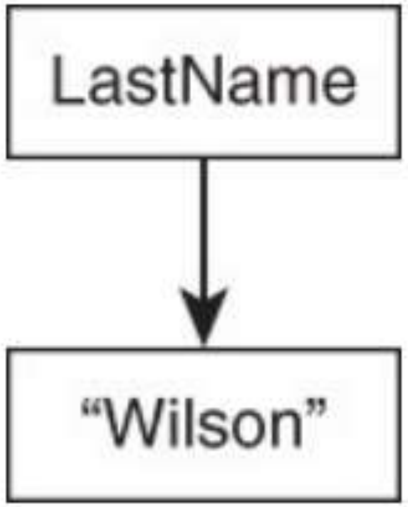


Figure 2.14 *A column consists of a name and a value. In this example, the column is named lastName and has a value of “Wilson.”*

A set of columns makes up a row. Rows can have the same columns, or they can have different columns, as shown in Figure 2.15.

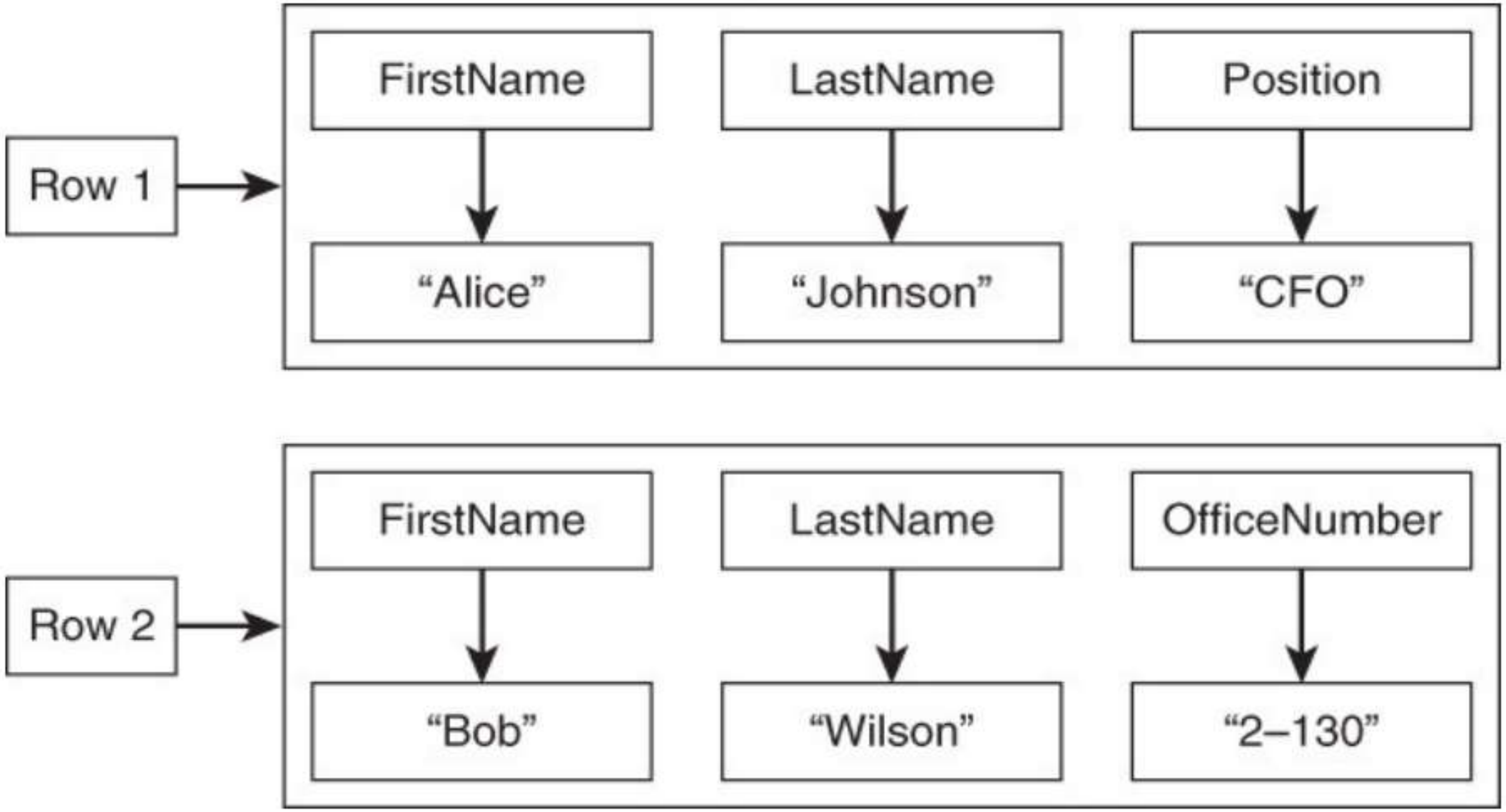


Figure 2.15 *A row consists of one or more columns. Different rows can have different columns.*

- When there are large numbers of columns, it can help to group them into collections of related columns.

- For example, first and last name are often used together, and office numbers and office phone numbers are frequently needed together.

- These can be grouped in collections called column families.

- As in document databases, column family databases do not require a predefined fixed schema. Developers can add columns as needed.

- Also, rows can have different sets of columns and super columns. Column family databases are designed for rows with many columns. It is not unusual for column family databases to support millions of columns.

Differences Between Column Family and Relational Databases

- They both make use of rows and columns, for example. There are important differences in terms of data models and implementation details.

- One thing missing from column family databases is support for joining tables.

- In relational databases, data about an object can be stored in multiple tables.

- For example, a customer might have name, address, and contact information in one table; a list of past orders in another table; and a payment history in another.

- If you needed to reference data from all three tables at once, you would need to perform a join operation between tables.

- Column family databases are typically denormalized, or structured so that all relevant information about an object is in a single, possibly very wide, row.

- Query languages for column family databases may look similar to SQL. The query language can support SQL-like terms such as SELECT, INSERT, UPDATE, and DELETE as well as column family–specific operations, such as CREATE COLUMN FAMILY.

4.Graph Databases

- Instead of modeling data using columns and rows, a graph database uses structures called nodes and relationships (in more formal discussions, they are called vertices and edges).

- A node is an object that has an identifier and a set of attributes. A relationship is a link between two nodes that contain attributes about that relation.

- There are many ways to use graph databases. Nodes can represent people, and relationships can represent their friendships in social networks.

- A node could be a city, and a relationship between cities could be used to store information about the distance and travel time between cities.

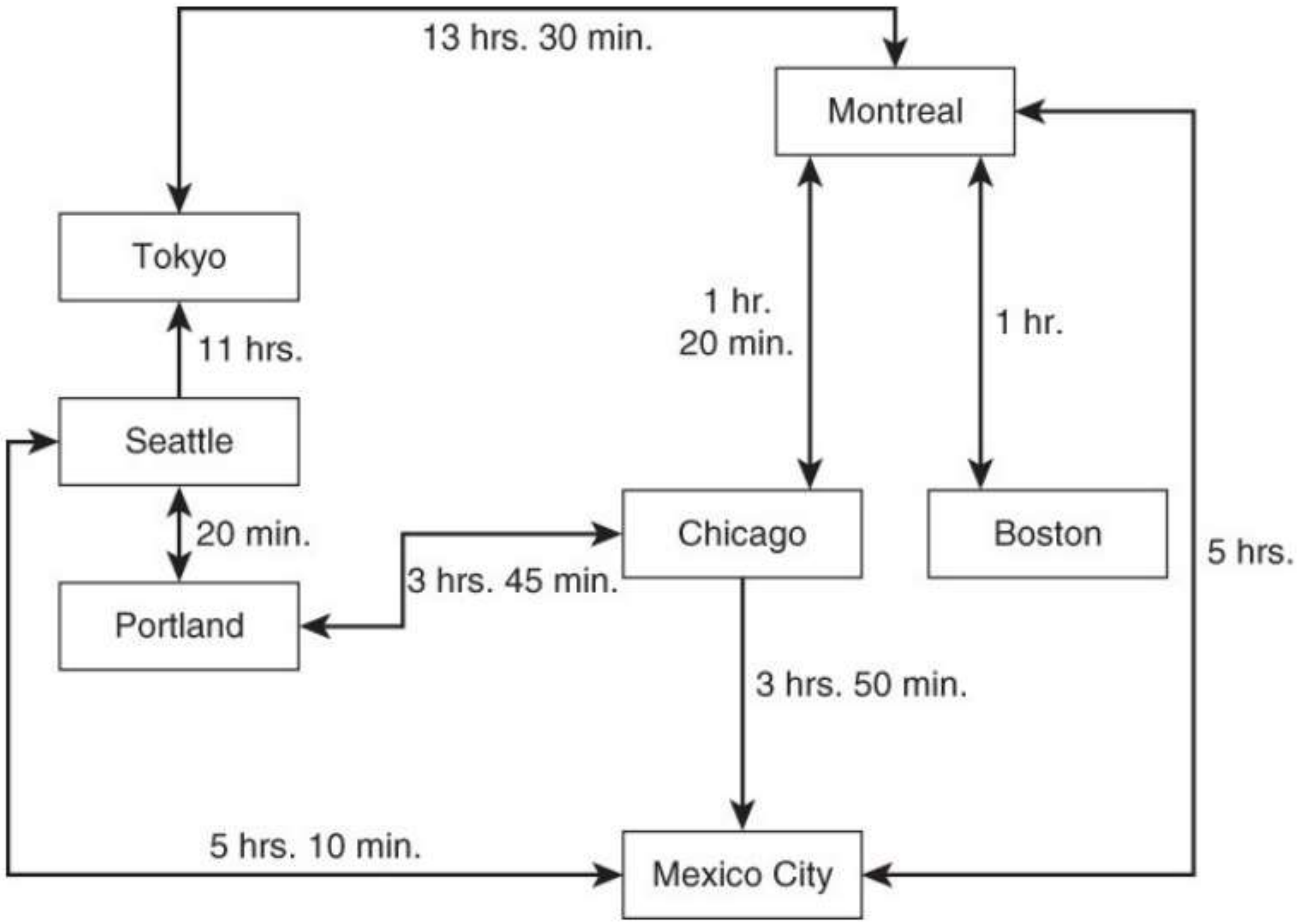


Figure 2.16 *Properties of relationships or nodes store attributes about relations between linked nodes. In this case, attributes include flying times between cities.*

- Both the nodes and relationships can have complex structures. For example, each city can have a list of airports along with demographic and geographic data about the city.

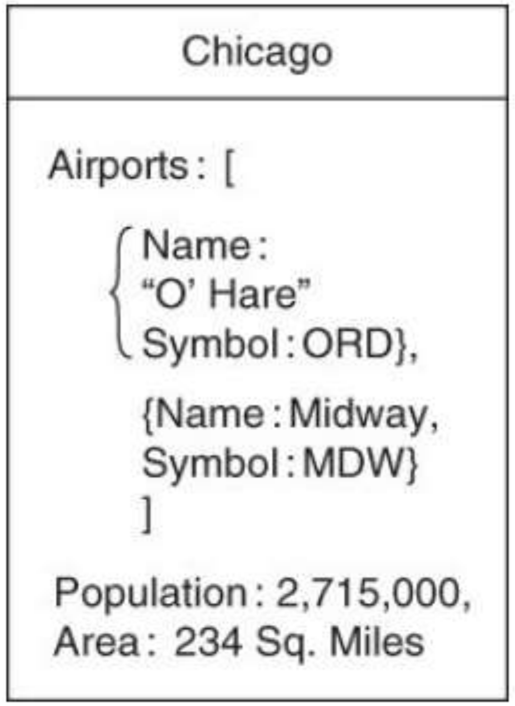
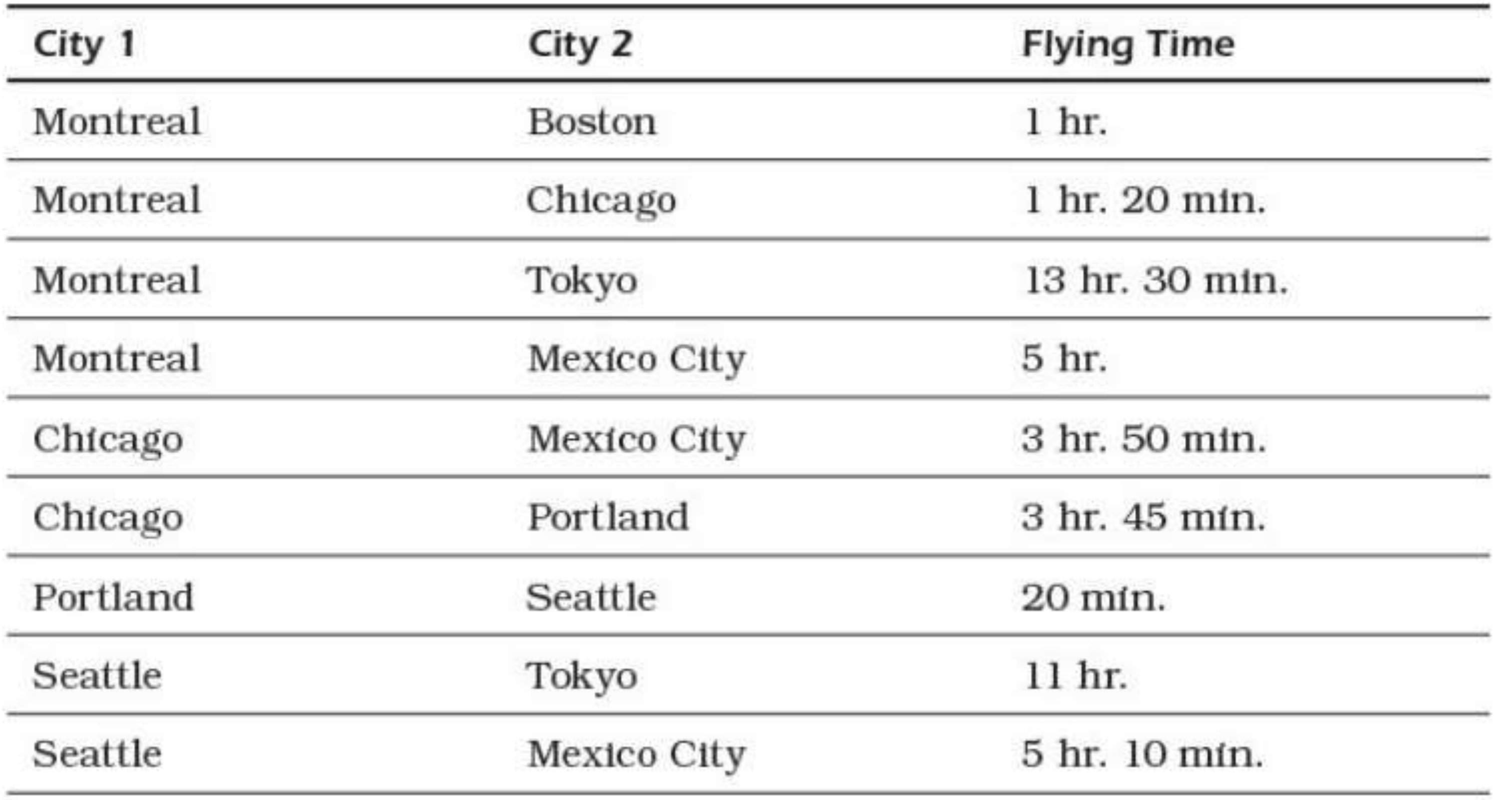


Figure 2.17 *Nodes can also have attributes to describe the node. In this case, attributes include information about the airports in the city along with population and geographic area.*

Differences Between Graph and Relational Databases

- Graph databases are designed to model adjacency between objects. Every node in the database contains pointers to adjacent objects in the database. This allows for fast operations that require following paths through a graph.



City 1	City 2	Flying Time
Montreal	Boston	1 hr.
Montreal	Chicago	1 hr. 20 min.
Montreal	Tokyo	13 hr. 30 min.
Montreal	Mexico City	5 hr.
Chicago	Mexico City	3 hr. 50 min.
Chicago	Portland	3 hr. 45 min.
Portland	Seattle	20 min.
Seattle	Tokyo	11 hr.
Seattle	Mexico City	5 hr. 10 min.

Table 2.1 *Flight Times Between Cities Modeled as a Relational Table*

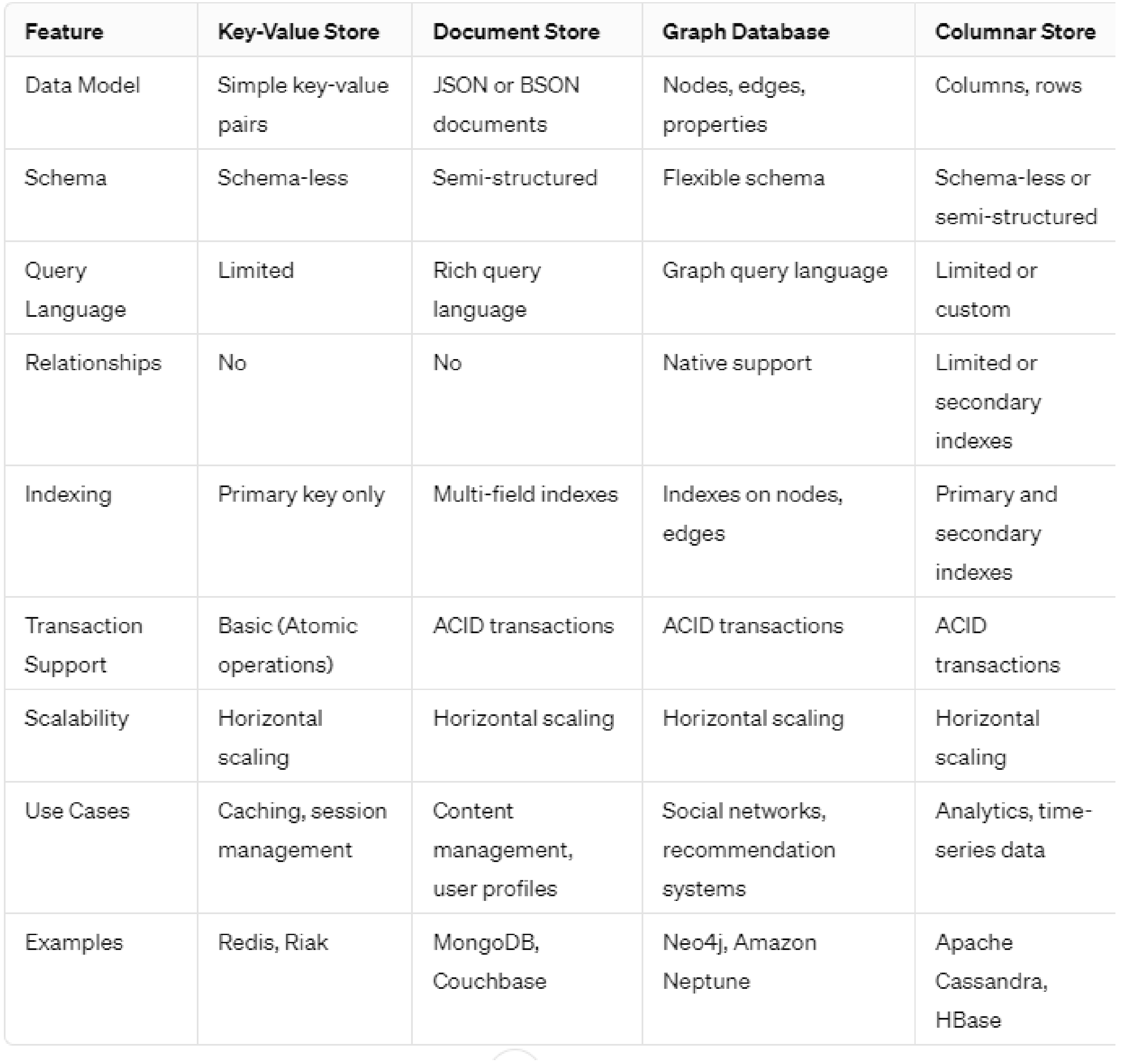
- Querying is more difficult. You would have to write multiple SQL statements or use specialized recursive statements if they are provided (for example, Oracle’s CONNECT BY clause in SELECT statements) to find paths using the table representation of the data.

- Graph databases allow for more efficient querying when paths through graphs are involved.

- Many application areas are efficiently modeled as graphs and, in those cases, a graph database may streamline application development and minimize the amount of code you would have to write

Topic 4 Completed

➢ **Main differences between four databases (previous question)****

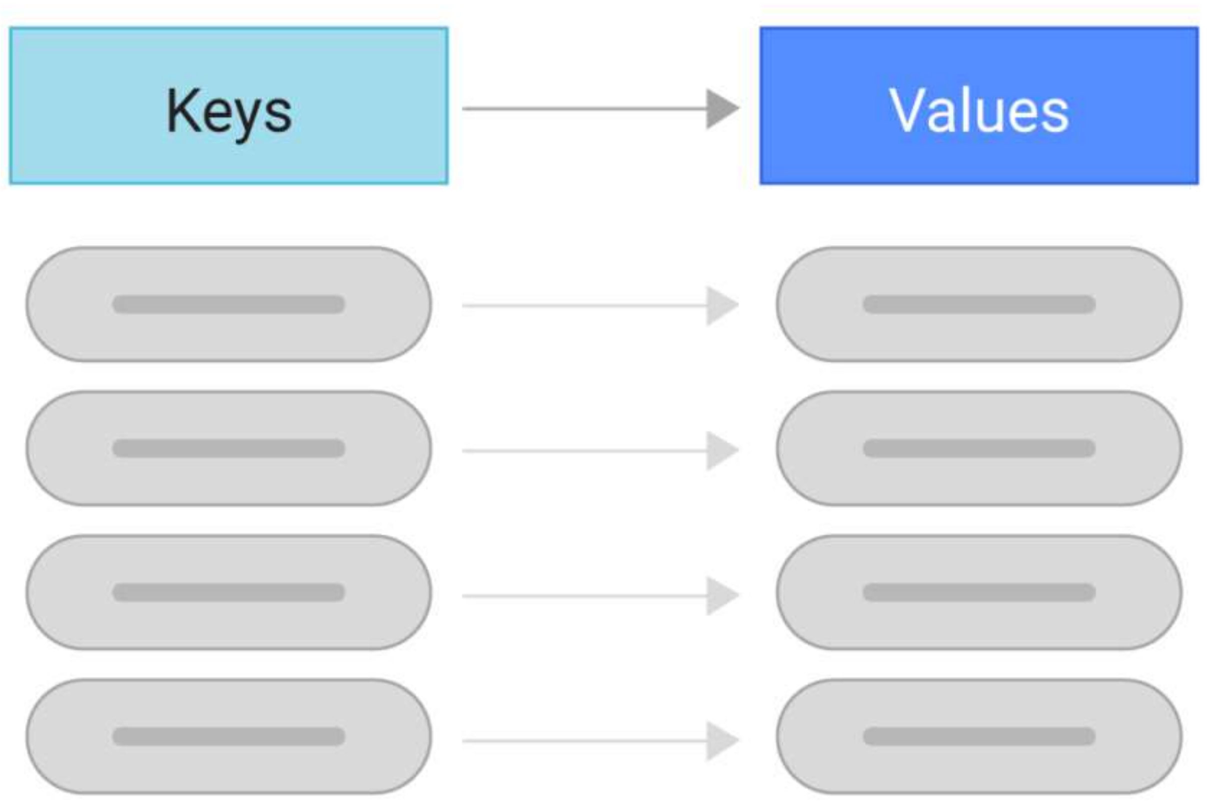


Feature	Key-Value Store	Document Store	Graph Database	Columnar Store
Data Model	Simple key-value pairs	JSON or BSON documents	Nodes, edges, properties	Columns, rows
Schema	Schema-less	Semi-structured	Flexible schema	Schema-less or semi-structured
Query Language	Limited	Rich query language	Graph query language	Limited or custom
Relationships	No	No	Native support	Limited or secondary indexes
Indexing	Primary key only	Multi-field indexes	Indexes on nodes, edges	Primary and secondary indexes
Transaction Support	Basic (Atomic operations)	ACID transactions	ACID transactions	ACID transactions
Scalability	Horizontal scaling	Horizontal scaling	Horizontal scaling	Horizontal scaling
Use Cases	Caching, session management	Content management, user profiles	Social networks, recommendation systems	Analytics, time-series data
Examples	Redis, Riak	MongoDB, Couchbase	Neo4j, Amazon Neptune	Apache Cassandra, HBase

Once refer applications of all four databases

TOPIC - 5

Introduction to Key-Value Databases



Topics Covered :

1. From Arrays to Key-Value Databases
2. Essential Features of Key-Value Databases
3. Properties of Keys
4. Characteristics of Values

- Key-value databases are the simplest of the NoSQL databases and are a good place to start a detailed examination of NoSQL database options.

- As the name implies, the design of this type of data store is based on storing data with identifiers known as keys.

- This chapter introduces key-value data structures by starting with an even simpler data structure: the array.

- Normal array has more constraints on its indexes and the values to be stored in it.

From Arrays to Key-Value Databases

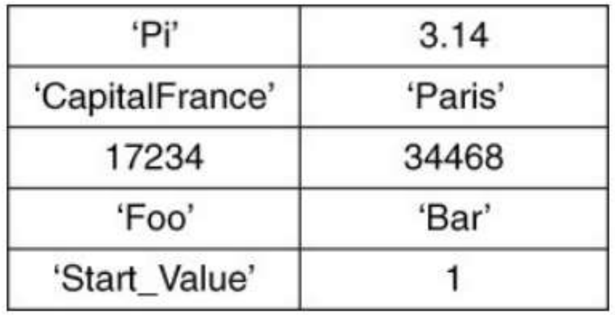
- **<u>Associative Arrays: Taking Off the Training Wheels:</u>**
- An associative array is a data structure, like an array, but is not restricted to using integers as indexes or limiting values to the same type. You could, for example, have commands such as the following:

```
exampleArray[1] = 'Goodbye world.'
exampleArray[2] = 'This is a test.'
exampleArray[5] = 'Key-value database'
exampleArray[9] = 'Elements can be set in any order.'
```

Normal Array

```
exampleAssociativeArray['Pi'] = 3.1415
exampleAssociativeArray['CapitalFrance'] = 'Paris'
exampleAssociativeArray['ToDoList'] = { 'Alice' : 'run
  reports; meeting with Bob', 'Bob' : 'order inventory;
  meeting with Alice' }
exampleAssociativeArray[17234] = 34468
```

Associative Array



'Pi'	3.14
'CapitalFrance'	'Paris'
17234	34468
'Foo'	'Bar'
'Start_Value'	1

Figure 3.2 *An associative array shares some characteristics of arrays but has fewer constraints on keys and values.*

- Unlike normal arrays, associate arrays can have arbitrary values of identifiers as index values.

- Depending on the programming language or database, you may be able to use keys with even more complex data structures, such as a list of values.

- Identifiers/Indexes = Keys
  Values assigned = Values

- **Note :**
  1.Associative arrays go by a number of different names, including dictionary, map, hash map, hash table, and symbol table.
  2.Values stored in the associative array can vary.

Cache memory – key value database

- Many key – value data stores keep persistent copies of data on long term storage such as hard drives or flash drives. Some data stores only keep data in memory.

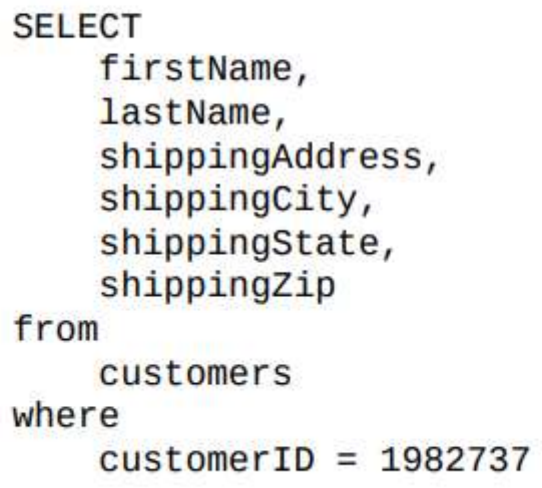
- Data can be retrieved faster if it is in memory than from a database or disk.

- The first time the program fetches the data, it will need to read from the disk but after that the results can be saved in memory.

- An in-memory cache is an associative array. The values retrieved from the relational database could be stored in the cache by creating a key for each value stored.

- One way to create a unique key for each piece of data for each customer is to concatenate a unique identifier with the name of the data item in the following example.

- A SQL query such as the following retrieves name and shipping address information from a relational table called customers:

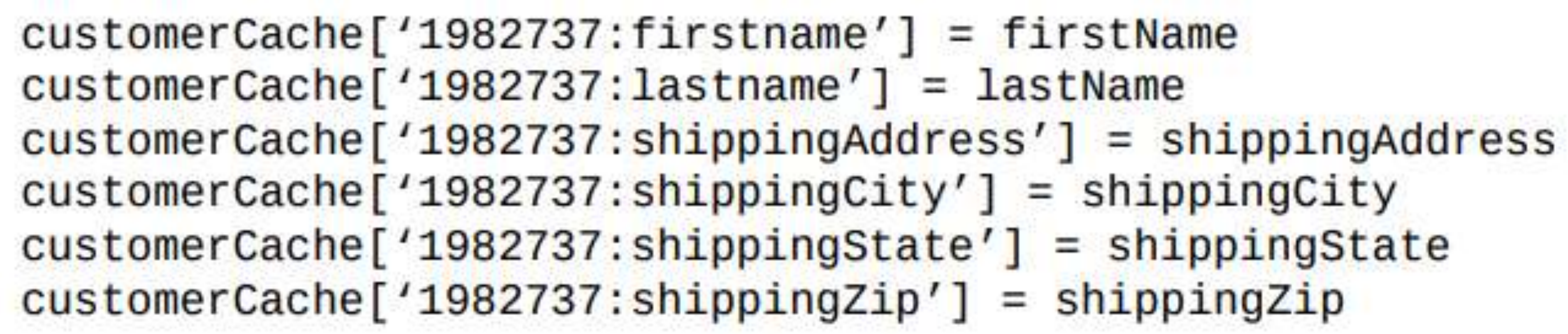


```
SELECT
    firstName,
    lastName,
    shippingAddress,
    shippingCity,
    shippingState,
    shippingZip
from
    customers
where
    customerID = 1982737
```

- Only the information for the customer with the customerID of 1982737 is retrieved.

- Because the customerID is part of the key, the cache can store data about as many customers as needed without creating separate program variables for each set of customer data.

- ➢ The following stores the data retrieved from the database in an in-memory cache.



```
customerCache['1982737:firstname'] = firstName
customerCache['1982737:lastname'] = lastName
customerCache['1982737:shippingAddress'] = shippingAddress
customerCache['1982737:shippingCity'] = shippingCity
customerCache['1982737:shippingState'] = shippingState
customerCache['1982737:shippingZip'] = shippingZip
```

- ➢ No need of creating a new key value to be concatenated as customerID is already a key value. We can directly concatenate it.

- ➢ Programs that access customer data will typically check the cache first for data and if it is not found in the cache, the program will then query the database. Here is sample pseudocode for a getCustomer function:

```
define getCustomer(p_customerID):
    begin
        if exists(customerCache['1982737:firstName]),
            return(
                   customerCache[p_customerID
                     +':lastname'],
                   customerCache[p_customerID
                     +':shippingAddress'],
                   customerCache[p_customerID
                     +':shippingCity'],
                   customerCache['p_customerID
                     +':shippingState'],
                   customerCache[p_customerID
                     +":shippingZip']
                   );
    else
            return(addQueryResultsToCache(p_customerID,
                   'SELECT
                        firstName,
                        lastName,
                        shippingAddress,
                        shippingCity,
                        shippingState,
                        shippingZip
                    FROM
                         customers
                WHERE
                        customerID = p_customerID')
     end;
 );
```

- The pseudocoded function takes one parameter, p_customerID, which is a unique identifier of the customer.
- The if statement checks if there exists a key in the cache that consists of the customer identifier passed in as a parameter and the character string 'firstName'.
- If the customer's first name is not in the cache, the function executes another function called addQueryResultsToCache. This function takes a key, and SQL query returns the data associated with that key.
- The function also stores a copy of the returned data in the cache so it is available next time the getCustomer function is called.

❖ **Caution:** Like arrays in programming languages, when the server is shut down or the cache terminates, the data in memory is lost. The next time the application starts, it will have to reload the cache with data by executing statements like the SQL statement in the getCustomer function.

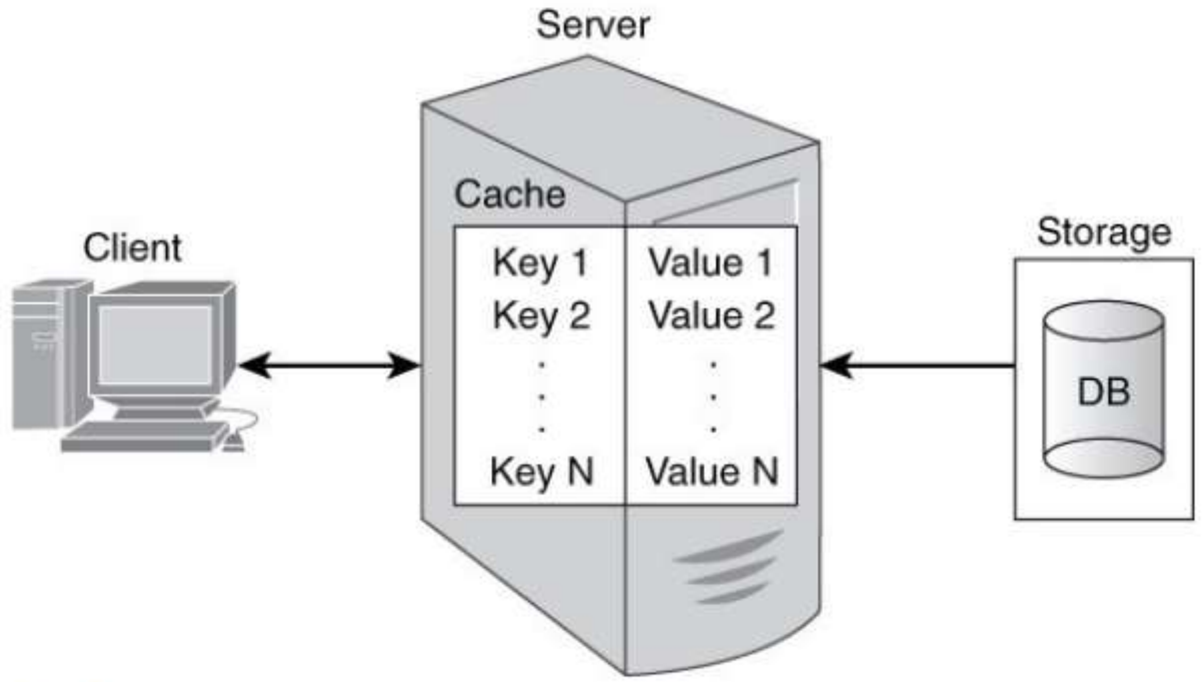


Figure 3.3 *Caches are associative arrays used by application programs to improve data access performance.*

In-Memory and On-Disk Key-Value Database:

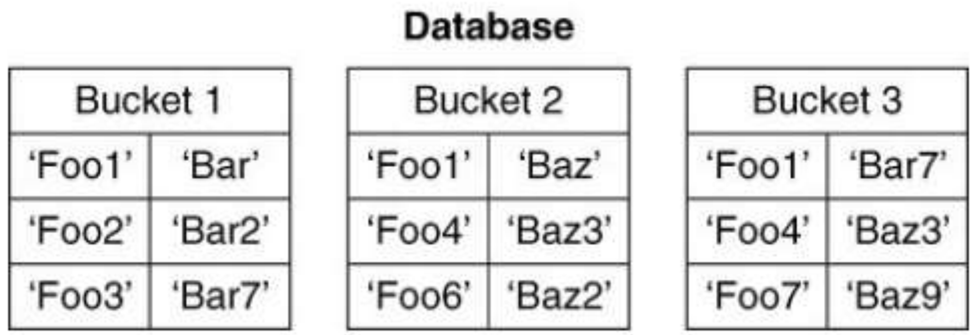
- ➢ Key-value databases impose a minimal set of constraints on how you arrange your data. There is no need for tables if you do not want to think in terms of groups of related attributes.

❖ **Note**

The one design requirement of a key-value database is that each value has a unique identifier in the form of the key. Keys must be unique within the namespace defined by the key-value database. The namespace can be called a bucket, a database, or some other term indicating a collection of key-value pairs (see Figure 3.4).

- ➢ Topic related to namespace will be discussed later..

- ➢ Caches are helpful for improving the performance of applications that perform many database queries. Key-value data stores are even more useful when they store data persistently on disk, flash devices, or other long-term storage. They offer the fast performance benefits of caches plus the persistent storage of databases


Database

Bucket 1		Bucket 2		Bucket 3	
'Foo1'	'Bar'	'Foo1'	'Baz'	'Foo1'	'Bar7'
'Foo2'	'Bar2'	'Foo4'	'Baz3'	'Foo4'	'Baz3'
'Foo3'	'Bar7'	'Foo6'	'Baz2'	'Foo7'	'Baz9'

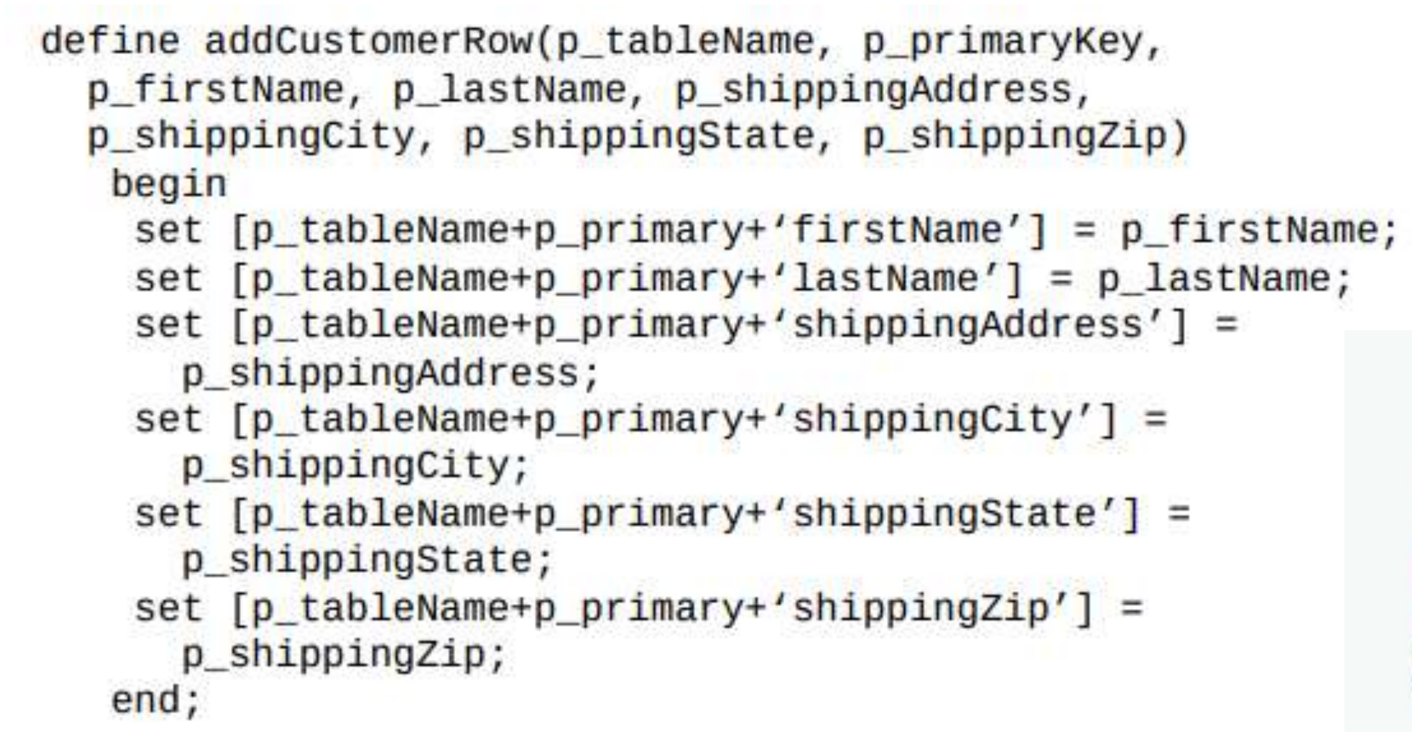
Figure 3.4 *Keys of a key-value database must be unique within a namespace.*

- A developer could use a key-naming convention that uses a table name, primary key value, and an attribute name to create a key to store the value of an attribute, as shown in the following example:

```
customer:1982737:firstName
customer:1982737:lastName
customer:1982737:shippingAddress
customer:1982737:shippingCity
customer:1982737:shippingState
customer:1982737:shippingZip
```

- Table_name : primary_key_value : attribute_name.

- The developer can create a set of functions that emulate the operations performed on a table, such as creating, reading, updating, or deleting a row. One example of pseudocode for a create function is



```
define addCustomerRow(p_tableName, p_primaryKey,
  p_firstName, p_lastName, p_shippingAddress,
  p_shippingCity, p_shippingState, p_shippingZip)
   begin
    set [p_tableName+p_primary+‘firstName’] = p_firstName;
    set [p_tableName+p_primary+‘lastName’] = p_lastName;
    set [p_tableName+p_primary+‘shippingAddress’] =
      p_shippingAddress;
    set [p_tableName+p_primary+‘shippingCity’] =
      p_shippingCity;
    set [p_tableName+p_primary+‘shippingState’] =
      p_shippingState;
    set [p_tableName+p_primary+‘shippingZip’] =
      p_shippingZip;
   end;
```

❖ Essential Features of Key-Value Databases :

➢ A variety of key-value databases is available to developers, and they all share three essential features:

1.Simplicity → use simple data structures, no need of schema,joins,datatype may vary

2.Speed →As we use this in cache memory, speed of retrieving information increases

3.Scalability

3.Scalability:

➢ Scalability is the capability to add or remove servers from a cluster of servers as needed to accommodate the load on the system.

➢ Key-value databases take different approaches to scaling read and write operations. Let's consider two options:

- Master-slave replication
- Masterless replication

MASTER_SLAVE IMPLEMENTATION

- Applications like news sites or live sports scores see many more reads than writes.
- To handle this, a master-slave architecture can be used.
- The master server accepts both reads and writes, while slave servers only handle reads.
- Updated data is replicated from the master to slaves.

Advantages:

1. Simplicity: Slaves only communicate with the master, reducing complexity.
2. No write coordination: The master handles all writes, eliminating conflicts.

Disadvantages:

1. Single point of failure: If the master fails, writes are blocked.

Solution:

1. Servers can detect failures using heartbeat messages.
2. Upon master failure, slaves can elect a new master to resume write operations.

Overall, the master-slave model offers efficient read scaling but requires a solution for master failure to ensure write availability

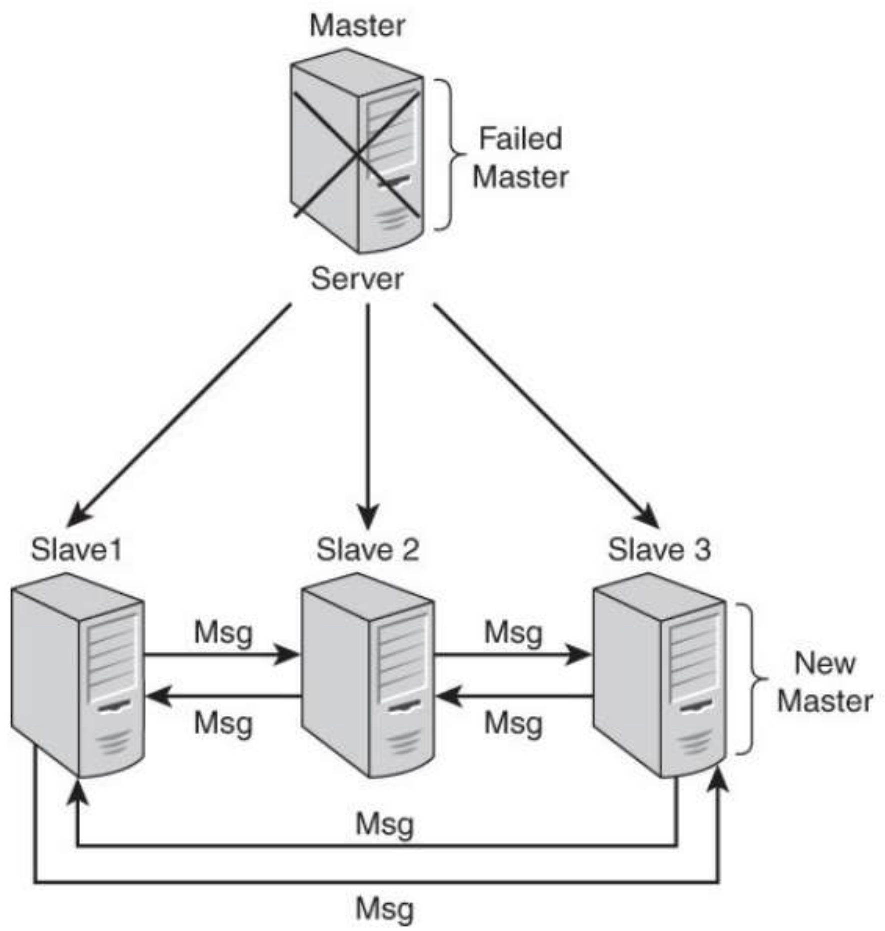


Figure 3.8 *Once a failed master server is detected, the slaves initiate a protocol to elect a new master.*

Masterless Replication Model:

•All servers can accept reads and writes, improving write scalability.

•Challenge: Preventing conflicts when multiple servers try to update the same data (e.g., selling the same concert ticket).

Scaling Reads in Masterless Replication:

•No single master copy exists for updates.

•Servers collaborate in a ring structure (logical concept, not physical network layout) for replication.

Example:

•Eight servers configured in a ring.

•Each server keeps four replicas of data.

•Upon a write, a server replicates the change to its two neighbors and the server two links ahead.

Overall, the masterless model avoids a single point of failure for writes but requires solutions for data consistency across multiple servers.

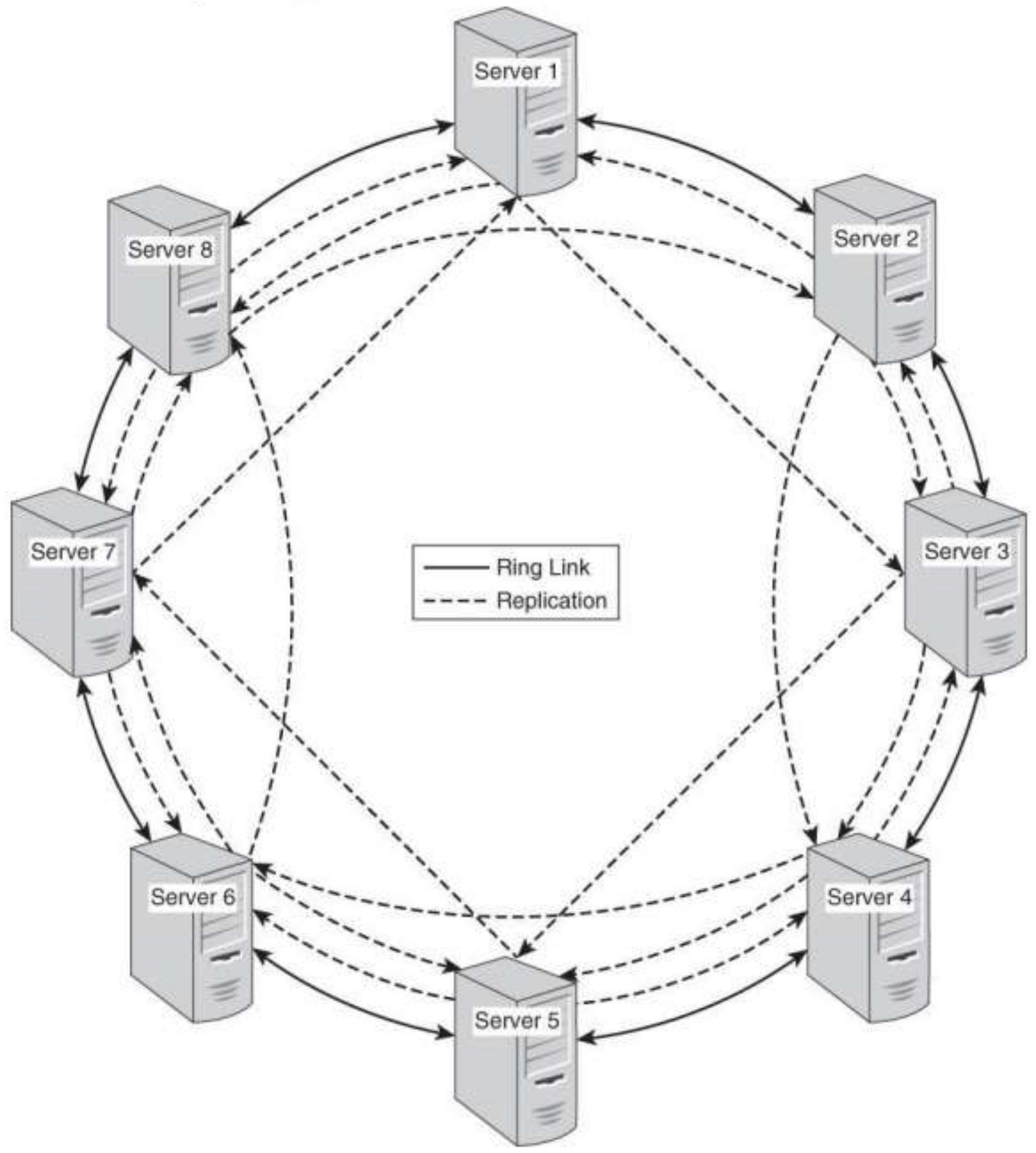


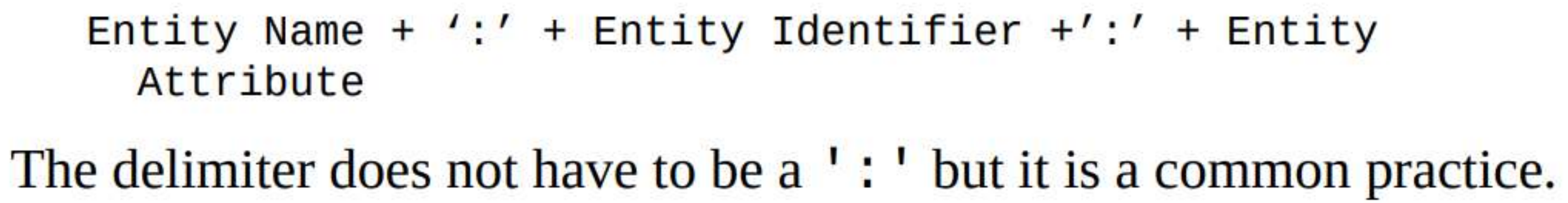
Figure 3.11 *An eight-server cluster in a ring configuration with a replication factor of 4.*

PROPERTIES OF KEYS:

HOW TO CREATE KEYS?

- Keys are used to identify, index, or otherwise reference a value in a key-value database. The one essential property of a key is that it must be unique within a namespace.

- Counters and sequences are functions that return a new unique number each time the function is called. Database application designers use these routinely to make keys for rows of data stored in a table. Each generated number is a unique identifier used by a row in a table.



```
Entity Name + ':' + Entity Identifier +':' + Entity
  Attribute
```

The delimiter does not have to be a `':'` but it is a common practice.

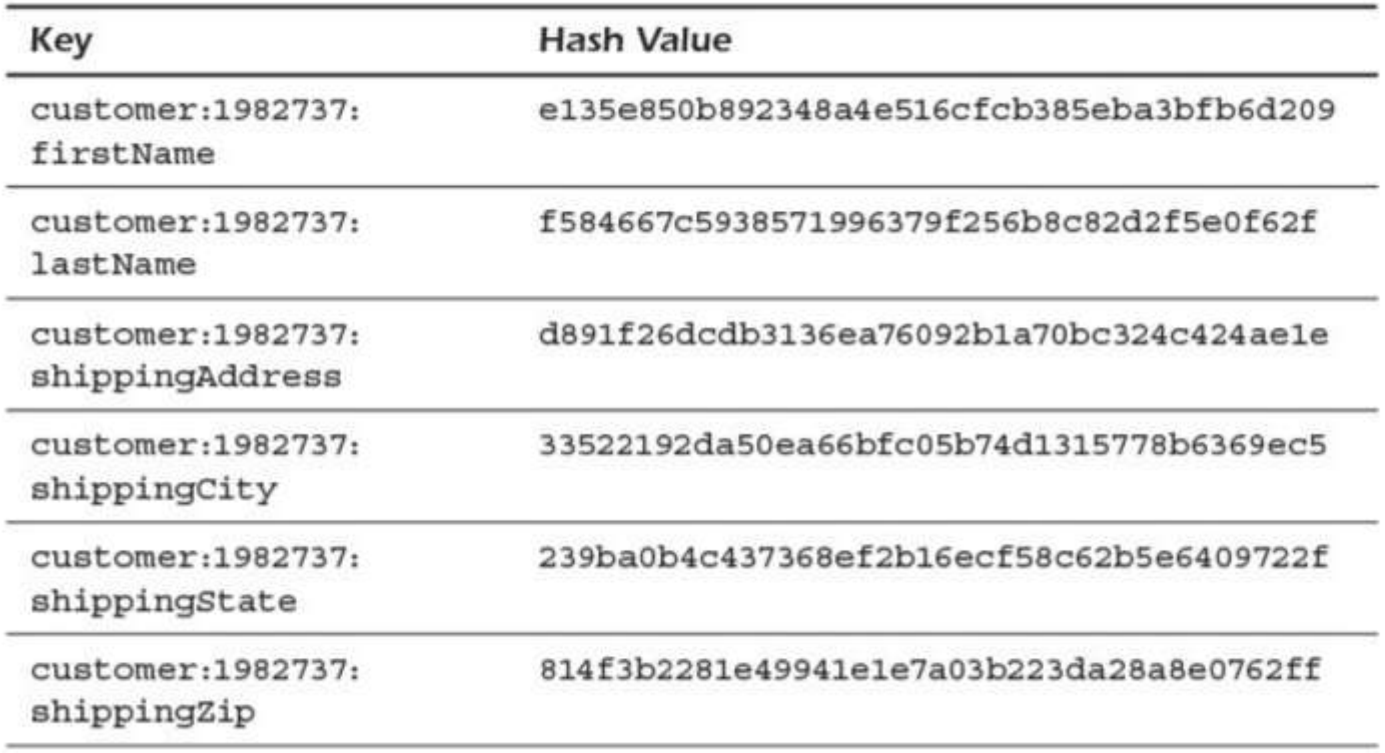
Using Keys to Locate Values:

•Keys can be various data types (integers, strings, even lists) for flexibility.

•A hash function is used to convert these keys into unique numerical or string identifiers.

•These hash values act as "pointers" to locate the actual data associated with the original key.

•Hash functions are not perfect and can sometimes generate collisions (same hash value for different keys). (Dealing with collisions is covered in a separate chapter, not discussed here)



Key	Hash Value
customer:1982737: firstName	e135e850b892348a4e516cfcb385eba3bfb6d209
customer:1982737: lastName	f584667c5938571996379f256b8c82d2f5e0f62f
customer:1982737: shippingAddress	d891f26dcdb3136ea76092b1a70bc324c424ae1e
customer:1982737: shippingCity	33522192da50ea66bfc05b74d1315778b6369ec5
customer:1982737: shippingState	239ba0b4c437368ef2b16ecf58c62b5e6409722f
customer:1982737: shippingZip	814f3b2281e49941e1e7a03b223da28a8e0762ff

Keys Help Avoid Write Problems

•Hash function outputs (numbers) are used to determine which server stores the data for a specific key.
•Goal: Distribute writes evenly across all servers in the cluster.

Distributing Writes with Hash Function:

1.Modulo Operation: The hash value is divided by the number of servers (e.g., dividing by 8 for 8 servers).
2.Remainder as Server ID: The remainder after the division becomes the server ID responsible for storing the data.

- •This ensures each server receives a roughly equal portion of writes.
- •Any hash value will have a remainder between 0 and (number of servers - 1).

Example (Using 8 Servers):

•Hash value 32 divided by 8 has a remainder of 0, so server 0 stores the data.
•Hash value 41 divided by 8 has a remainder of 1, so server 1 stores the data.

Benefit: This approach leverages the hash function to distribute writes more intelligently than a simple round-robin method.

Real-World Example (Concert Tickets):
•The challenge: Prevent multiple servers from selling the same ticket (seat, venue, city, date).
•Solution: Construct a key that combines all these details (e.g., A73:CivCen:PDX:0715).
•This key ensures all requests for the same specific ticket are directed to the same server using the modulo operation.
•This eliminates the risk of servers accidentally selling duplicate tickets.
Conclusion:
•Hash functions play a crucial role in distributing data efficiently across servers in key-value databases.
•By combining keys and modulo operations, these systems can ensure data consistency and prevent conflicts during writes.

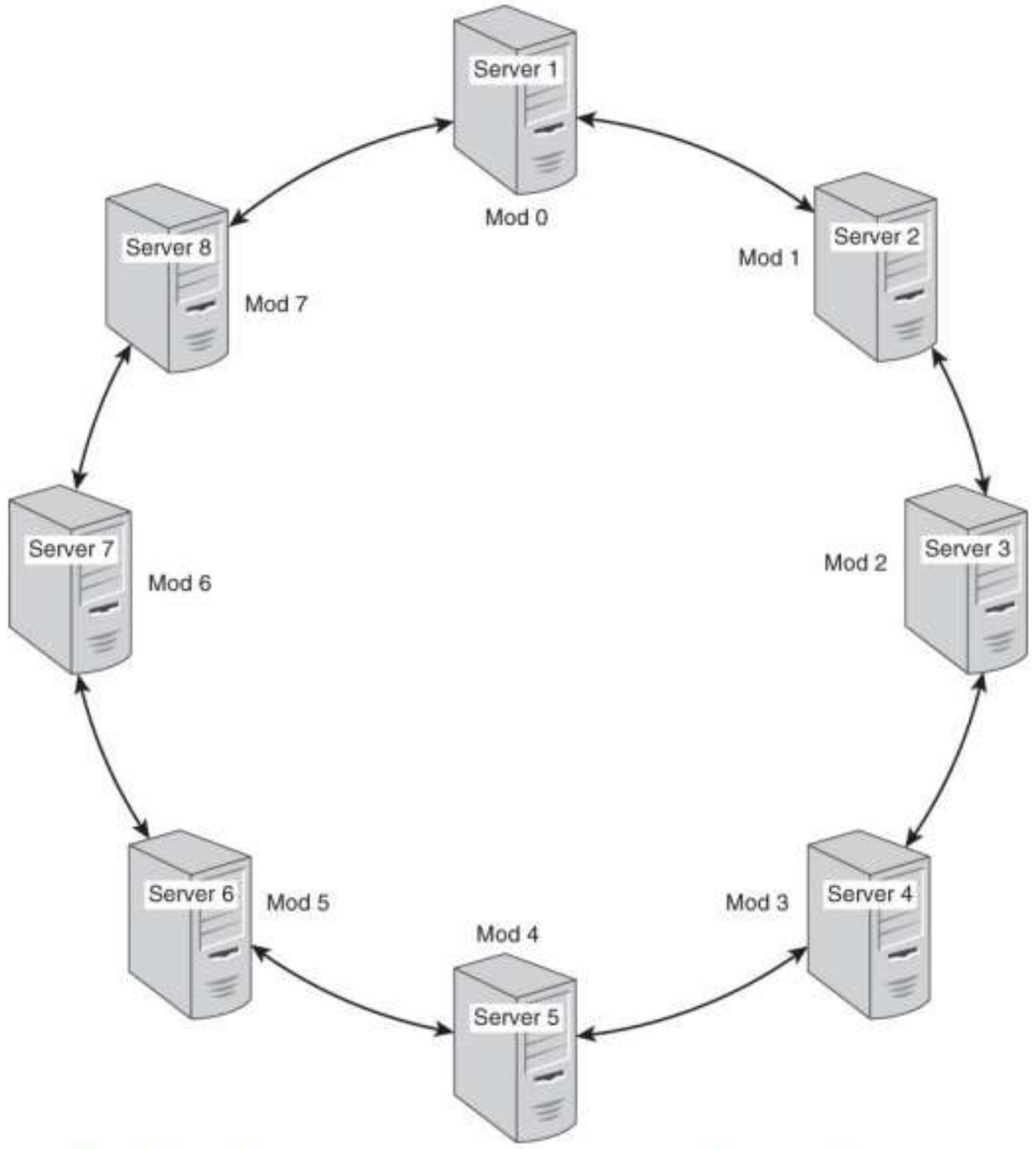


Figure 3.12 *An eight-server cluster in a ring configuration with modulo number assigned.*

CHARACTERISTICS OF VALUES:

- To declare values, we need not specify the type of values we are going to assign to the key , unlike strongly typed programming languages.
- We can store the values in any type as our wish(whatever we do it’s our wish).
- But be aware of type of database you use, because different types have different limitations on size of the value declare.
- Some may allow ACID transactions but limit you to smaller keys and values.
- Another key-value data store might allow for large values but limit keys to numbers or strings

Limitations on Searching for Values:

- ➢ You can retrieve a value by key, you can set a value by key, and you can delete values by key.

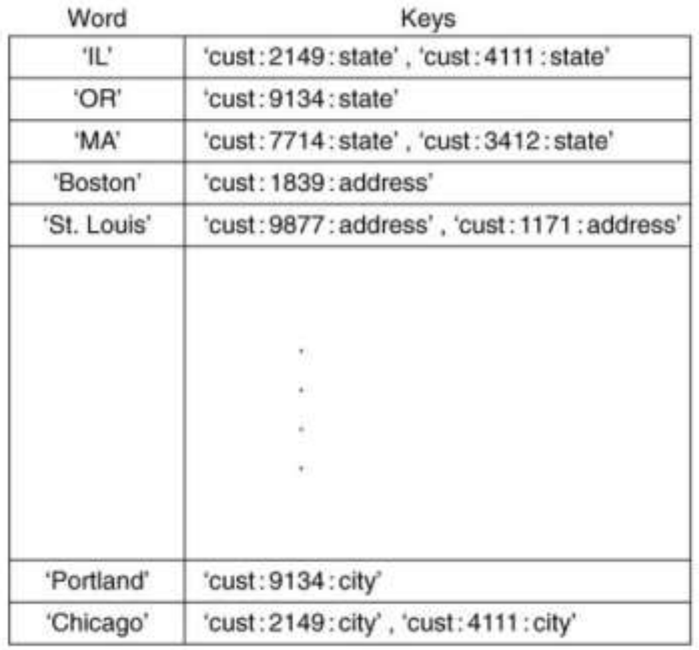
- ➢ Key-value databases do not support query languages for searching over values. There are two ways to address this limitation.

- ➢ You, as an application developer, could implement the required search operations in your application. For example, you could generate a series of keys, query for the value of each key, and test the returned value for the pattern you seek.

- ➢ This method enables you to search value strings, but it is inefficient. If you need to search large ranges of data, you might retrieve and test many values that do not have the city you are looking for.

- Some key-value databases incorporate search functionality directly into the database.

- A built-in search system would index the string values stored in the database and create an index for rapid retrieval. Rather than search all values for a string, the search system keeps a list of words with the keys of each key-value pair in which that word appears



Word	Keys
'IL'	'cust:2149:state' , 'cust:4111:state'
'OR'	'cust:9134:state'
'MA'	'cust:7714:state' , 'cust:3412:state'
'Boston'	'cust:1839:address'
'St. Louis'	'cust:9877:address' , 'cust:1171:address'
	. . . .
'Portland'	'cust:9134:city'
'Chicago'	'cust:2149:city' , 'cust:4111:city'

ure 3.13 *A search index helps efficiently retrieve data when selecting by criteria based on values.*

Review Questions From Text Book

1. How are associative arrays different from arrays?
2. How can you use a cache to improve relational database performance?
3. What is a namespace?
4. Describe a way of constructing keys that captures some information about entities and attribute types.
5. Name three common features of key-value databases.
6. What is a hash function? Include important characteristics of hash functions in your definition.
7. How can hash functions help distribute writes over multiple servers?
8. What is one type of practical limitation on values stored in key-value databases?
9. How does the lack of a query language affect application developers using key-value databases?
10. How can a search system help improve the performance of applications that use key-value databases?

TOPIC - 6
Key-Value Terminology

Topics Covered :
Key-Value Database Data Modeling Terms
Key-Value Architecture Terms
Key-Value Implementation Terms

1.Key-Value Database Data Modeling Terms :

•**Data models:** Abstractions that organize data in databases. They focus on the "what" (information) rather than the "how" (storage).

•**Data structures:** Well-defined ways to store data using underlying hardware. They focus on the "how" (implementation details).

•Data structures offer a higher level of organization and machine-level operations on addresses. Data models serve a similar purpose. They provide a level of organization and abstraction above data structures

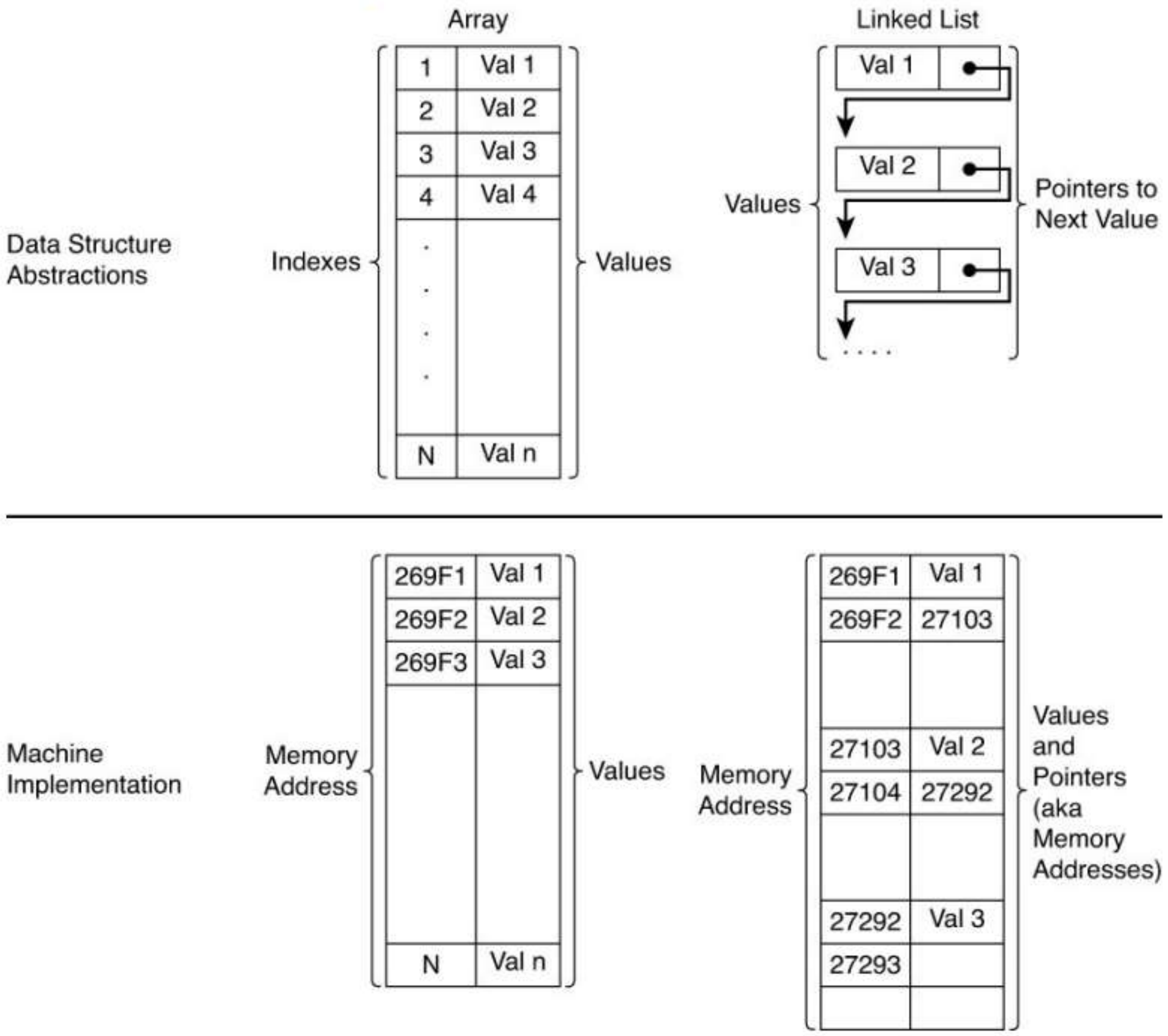


Figure 4.1 *Data structures provide higher-level organizations than available at the machine level.*

➢ Key

➢ A key is a reference to a value. It is analogous to an address

➢ A key can take on different forms depending on the key-value database used. At a minimum, a key is specified as a string of characters, such as "Cust9876" or "Patient:A384J:Allergies“

➢ The supported key data types in Redis version 2.8.13 include

- Strings
- Lists
- Sets
- Sorted sets \
- Hashes
- Bit arrays

➢

- Lists are ordered collections of strings.

- Sets are collections of unique items in no particular order. Sorted sets, as the name implies, are collections of unique items in a particular order.

- Hashes are data structures that have key-value characteristics: They map from one string to another.

- Bit arrays are binary integer arrays in which each individual bit can be manipulated using various bit array operations

➢Value

- A value is an object, typically a set of bytes, that has been associated with a key. Values can be integers, floating-point numbers, strings of characters, binary large objects (BLOBs), semi-structured constructs such as JSON objects, images, audio, and just about any other data type you can represent as a series of bytes.

➢Namespace

- A namespace is a collection of key-value pairs. You can think of a namespace as a set, a collection, a list of key-value pairs without duplicates, or a bucket for holding key-value pairs.

- A namespace could be an entire key-value database. The essential characteristic of a namespace is it is a collection of key-value pairs that has no duplicate keys. It is permissible to have duplicate values in a namespace.

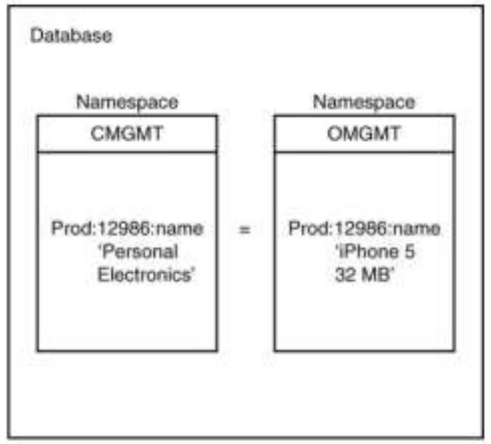


Figure 4.3 *Namespaces enable duplicate keys to exist without causing conflicts by maintaining separate collections of keys.*

➢ Partition:

- ➢ Just as it is helpful to organize data into subunits

- ➢ A partitioned cluster is a group of servers in which servers or instances of key-value database software running on servers are assigned to manage subsets of a database.

- ➢ . Let’s consider a simple example of a two-server cluster. Each server is running key-value database software. Ideally, each server should handle 50% of the workload

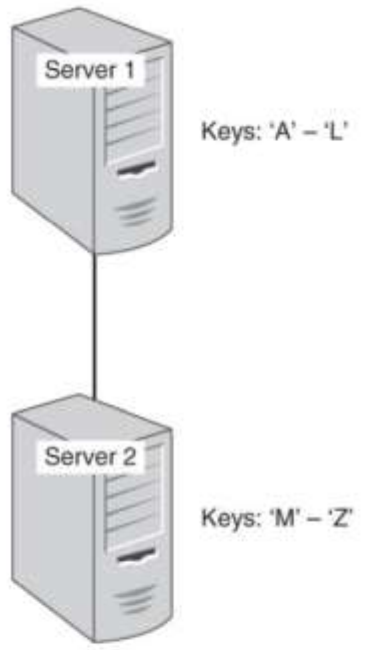


Figure 4.4 *Servers in a cluster are assigned subsets of data to manage.*

- Partition schemes should be chosen to distribute the workload as evenly as possible across the cluster.

- The “Partition Key” section describes a widely used method to help ensure a fairly even distribution of data and, therefore, workload

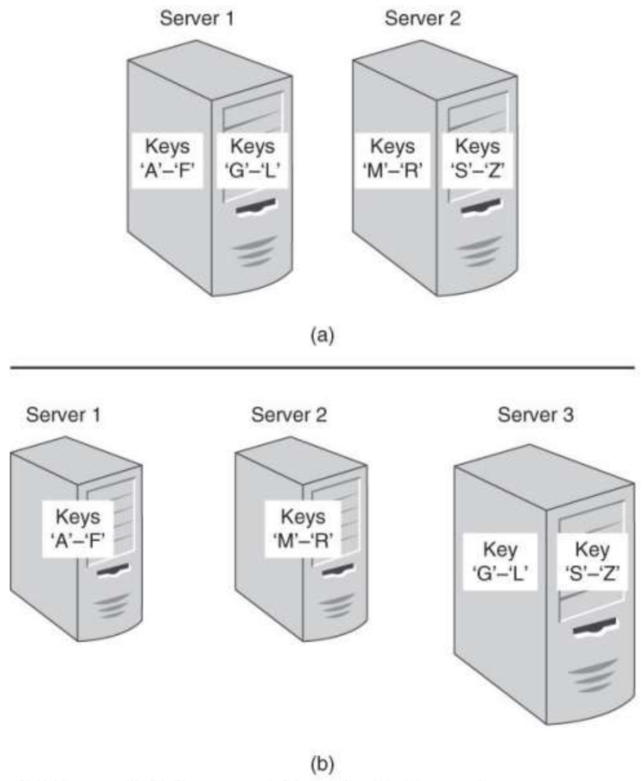


Figure 4.5 *When multiple instances of key-value database software run on servers in a cluster, servers can be added to the cluster and instances reallocated to balance the workload.*

➢ Partition Key

➢ A partition key is a key used to determine which partition should hold a data value

The Role of Partition Keys:

•**Distribution Strategy:** The partition key acts as a guide to determine where a particular data value (associated with a key) should be stored within the database.

•**Balancing Workload:** The primary goal is to distribute data (workload) across partitions as evenly as possible. This ensures efficient data retrieval and avoids overloading specific partitions with requests.

Challenges and Solutions:

•**Uneven Distribution:** Sometimes, natural key properties might not inherently promote balanced distribution. For instance, if most keys start with the letter "A," all partitions starting with "A" would be overloaded.

•**Hash Functions as a Rescue:** In such cases, hash functions come into play. They take a variable-length key (like a name) and convert it into a fixed-size value (like a number). This number, derived from the hash function, acts as a new "partition key." Ideally, a good hash function distributes these new values evenly across the available partitions.

Benefits of Effective Partitioning:

•**Faster Retrieval:** Knowing the partition key helps locate the specific partition where the desired data resides, leading to faster retrieval times.
•**Balanced Load:** By distributing data evenly across partitions, the overall workload on the system is balanced, ensuring optimal performance

Key-Value Architecture Terms

- ➢ The architecture of a key-value database is a set of characteristics about the servers, networking components, and related software that allows multiple servers to coordinate their work. Three key-value architectures:
  - Clusters
  - Rings
  - Replication

➢ 1.CLUSTERS

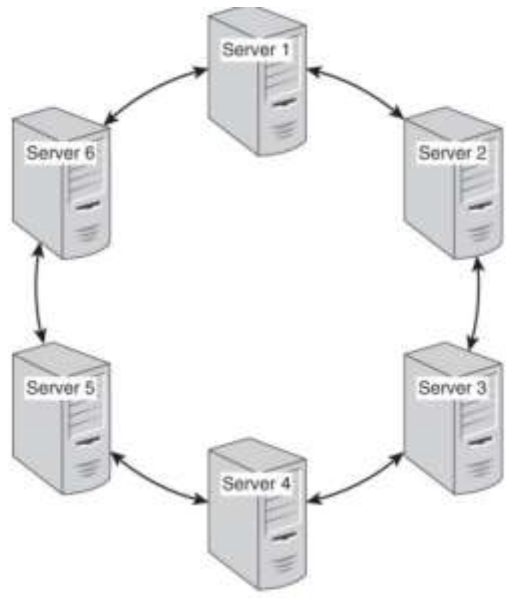
A cluster is a group of connected computers that work together as a single system. There are two main types of clusters: loosely coupled and tightly coupled.

•**Loosely coupled clusters** are made up of independent servers that can perform most tasks on their own. They communicate with each other occasionally to share information and check on each other's health. If a server fails, the other servers can take over its work.
multiple servers.

•**Tightly coupled clusters** have a high degree of communication between servers. This is necessary for them to work together on complex tasks. They may also have a master node that controls the other nodes in the cluster. If the master node fails, the other nodes can elect a new one.

Both types of clusters use replication to ensure that data is not lost if a server fails. Replication means that copies of the data are stored on multiple servers. There are two main ways to organize replication:

1.Master – slave replication
2. Masterless replication



igure 4.7 *A ring architecture of key-value databases links adjacent nodes in the cluster.*

2.RING

- A ring architecture is a way of organizing data in a key-value database cluster. The data is divided into partitions, and each server in the cluster is responsible for a certain range of partitions.

- A hash function is used to map a key (such as a customer ID) to a number between 0 and a maximum value (e.g., 95).

- The servers are arranged in a circular fashion, like a ring.

- Each server is responsible for a range of partition keys based on the hash function.

- Each server is also linked to two other servers in the ring: one to its left and one to its right.

- This ring architecture has several advantages:

- **Simplicity:** It is a simple way to organize data and assign responsibility to servers.
- **High availability:** If a server fails, its neighbors can take over its work. This means that the database is still available to read and write data, even if some servers are down.
- **Scalability:** It is easy to add or remove servers from the cluster. When a new server is added, it takes over some of the responsibility from its neighbors.
- One common way to replicate data in a ring architecture is to write copies of data to the two servers that are linked to the server that originally stores the data. This ensures that there are multiple copies of the data available, even if one server fails.

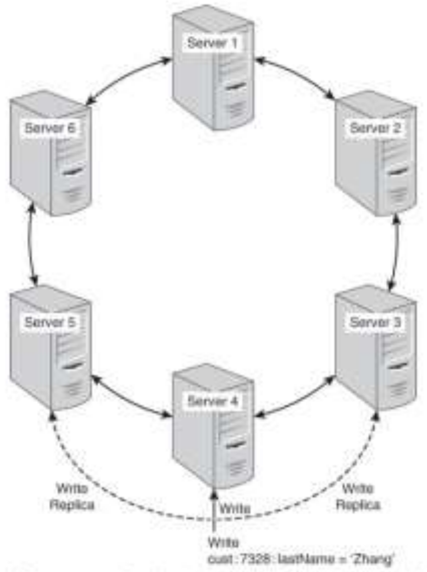
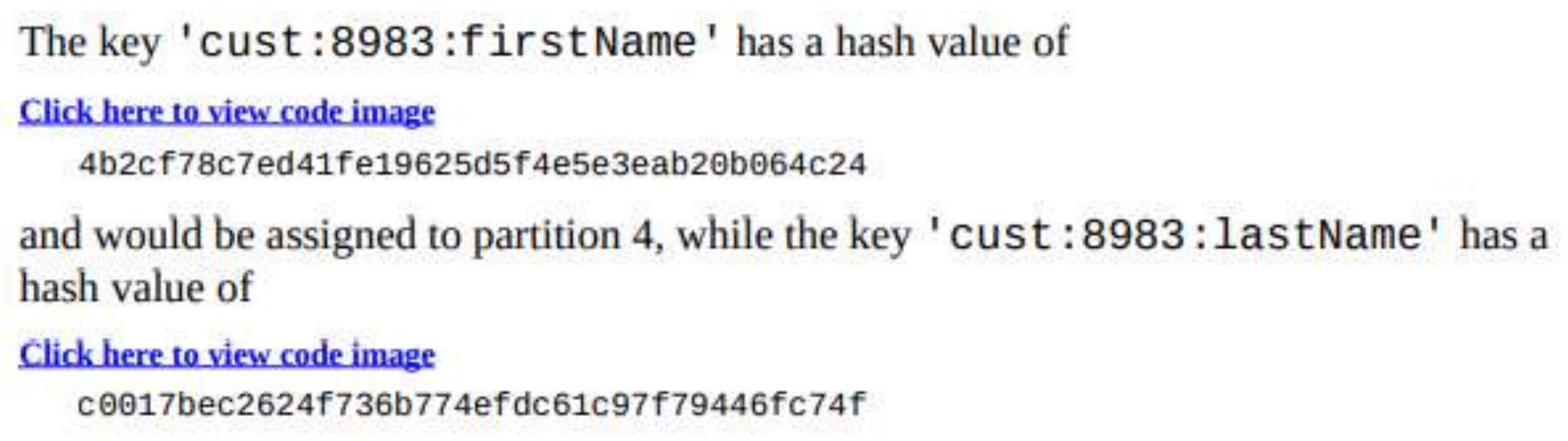


Figure 4.8 *One way to replicate data is to write copies of data to adjacent nodes in the cluster ring.*

Key-Value Implementation Terms
1.Hash function

- A hash function is like a special algorithm that takes an input of any size (text, numbers, etc.) and squeezes it into a fixed-size output string.

- This even distribution ensures no single server gets overloaded, keeping things balanced. It's like having different colored boxes for different types of documents, so no box gets overflowing with just one kind.



The key `'cust:8983:firstName'` has a hash value of

Click here to view code image

```
    4b2cf78c7ed41fe19625d5f4e5e3eab20b064c24
```

and would be assigned to partition 4, while the key `'cust:8983:lastName'` has a hash value of

Click here to view code image

```
    c0017bec2624f736b774efdc61c97f79446fc74f
```

2.Collision

- Collision in hash functions happens when two different inputs result in the same output. Ideally, a hash function should be collision resistant, meaning it's difficult to find such inputs.

- However, collisions are possible. When this happens, we need a collision resolution strategy to handle these cases.

- Imagine a hash table with limited space for each output. If a collision occurs, one of the colliding values might get lost.
- A common solution is to use linked lists. Each slot in the hash table can hold a list instead of just a single value. If a collision happens, both colliding values are added to the linked list at that slot.
- This way, even if collisions occur, we can still store all the data using linked lists. It's like having multiple rooms within each box (slot) in our earlier example to fit overflowing documents (data)

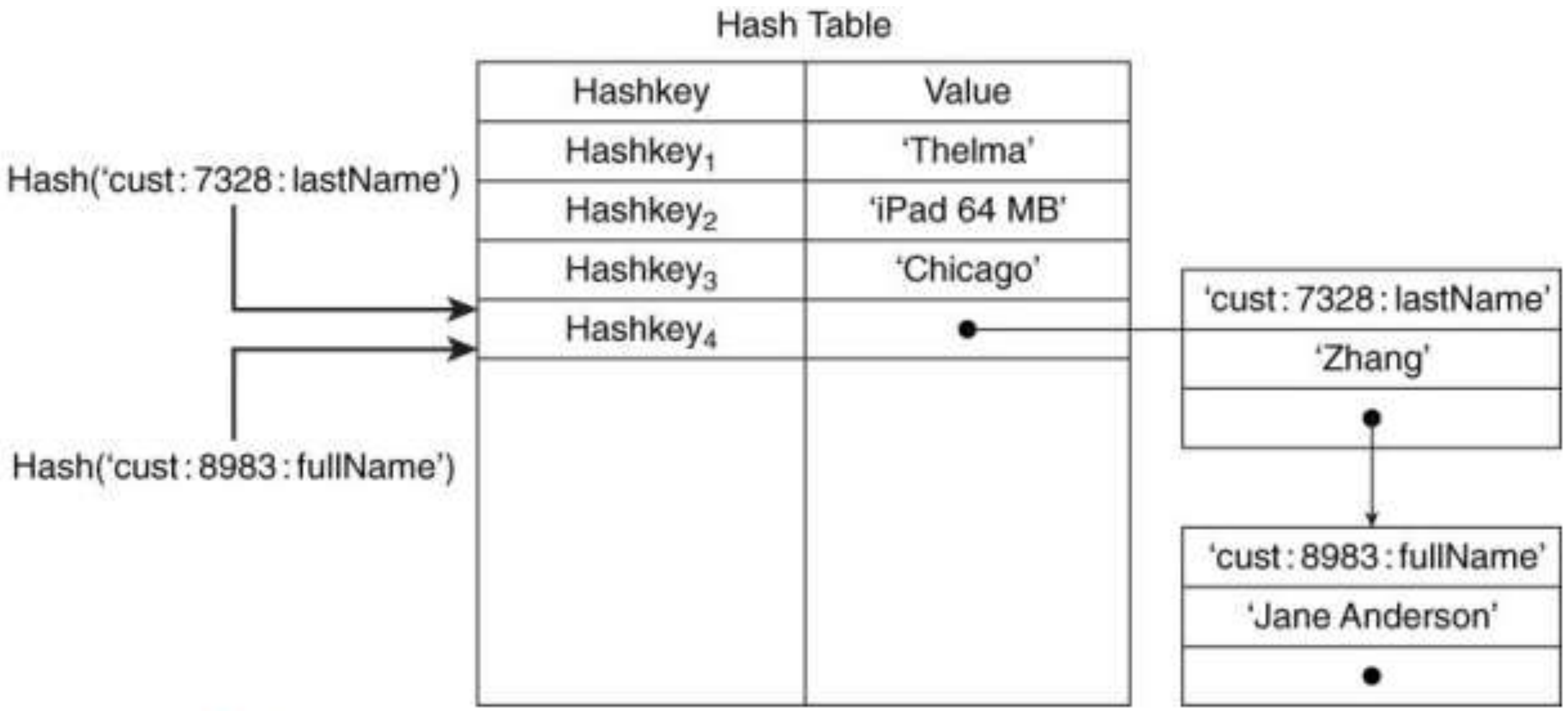


Figure 4.9 *Collisions with hash functions are managed using collision resolution strategies, such as maintaining linked lists of values.*

3.Compression

- Compression techniques can optimize both memory usage and storage space. Here's what to consider for compression algorithms in key-value stores:

- **Speed:** The compression and decompression operations should be fast

- **Compression ratio:** Ideally, the compressed data should be much smaller than the original data.

- However, there's often a trade-off between speed and compression ratio:
- **Faster algorithms:** Like Snappy, they compress data quickly but may result in a larger compressed size (20% to 100% bigger than other algorithms).
- **Slower algorithms:** These might compress data to a smaller size, but the compression and decompression processes would be slower.

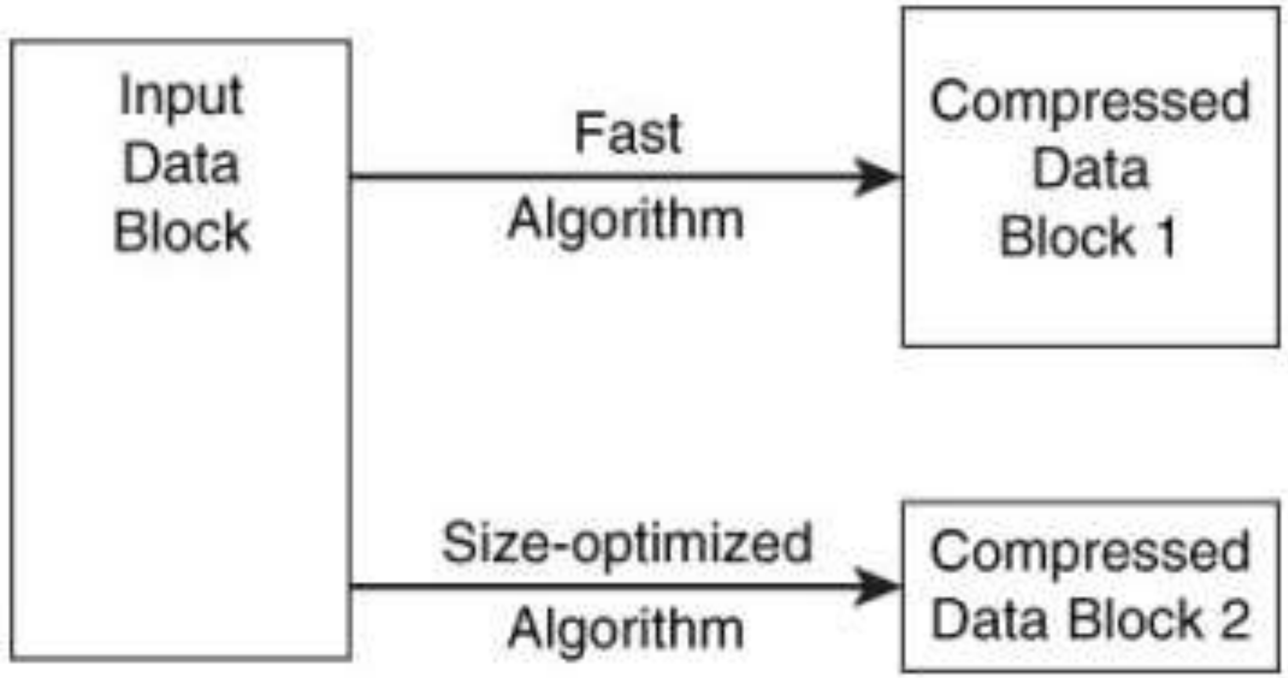


Figure 4.10 *Compression algorithms may be designed to optimize for speed or datc size.*

TOPIC - 7

Designing for the Key-Value Databases

Topics Covered In This Chapter

1.Key Design and Partitioning

- **Keys Should Follow a Naming Convention**
- Use meaningful and unambiguous naming components, such as 'cust' for customer or 'inv' for inventory.
- Use range-based components when you would like to retrieve ranges of values. Ranges include dates or integer counters.
- Use a common delimiter when appending components to make a key. The ':' is a commonly used delimiter, but any character that will not otherwise appear in the key will work.
- Keep keys as short as possible without sacrificing the other characteristics mentioned in this list.

1. Consistent Key Pattern:

- Define a key pattern that includes:
  - Entity or object type (e.g., 'customer')
  - Unique identifier (e.g., '198277')
  - Attribute name (e.g., 'fname')
  - Common delimiter (e.g., ':’)

2. Generic Set and Get Functions:

- Create a pair of generic functions:
  - getCustAttr(p_id, p_attrName) retrieves a value based on a key constructed using the pattern.
  - setCustAttr(p_id, p_attrName, p_value) sets a value for a key constructed using the pattern.

3. Benefits:

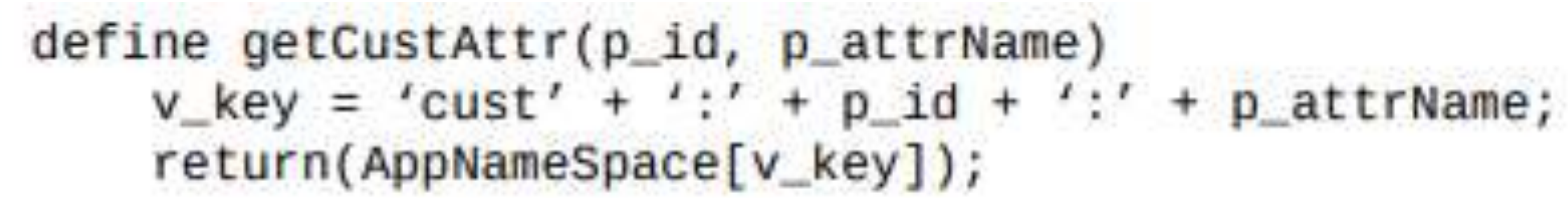
- **Reduced Code:** Less code needed for numerous read and write operations.
- **Improved Readability:** Code becomes clearer and more maintainable.
- **Minimized Low-Level Operations:** Reduces repetitive string manipulation.

4. Considerations:

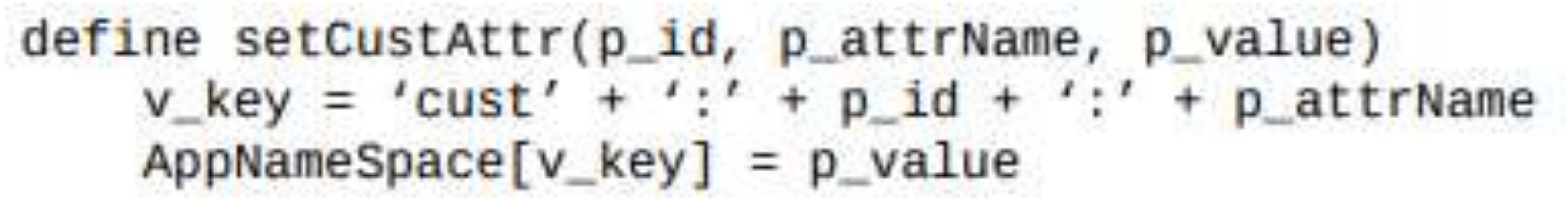
•**Error Handling:** Production applications should include error checking and handling for write operations, ensuring data integrity.

•**Namespace Conventions:** Use consistent naming conventions for namespaces as well, enhancing organization.

Remember: Well-designed keys streamline code, boost maintainability, and promote efficient data management in key-value stores.



```
define getCustAttr(p_id, p_attrName)
    v_key = ‘cust’ + ‘:’ + p_id + ‘:’ + p_attrName;
    return(AppNameSpace[v_key]);
```



```
define setCustAttr(p_id, p_attrName, p_value)
    v_key = ‘cust’ + ‘:’ + p_id + ‘:’ + p_attrName
    AppNameSpace[v_key] = p_value
```

Dealing with Ranges of Values in Key-Value Stores

- ➢ Traditional key-value stores aren't ideal for retrieving groups of values based on a range. Here's how to tackle this challenge:

1. Encode Ranges in Keys:

- ➢ Include a value in the key that indicates a range.
- ➢ Example: Use a date prefix ("cust061514") to represent customers who bought something on June 15, 2014.

2. Key Structure Example:

- ➢ Key prefix: "cust061514" (indicates purchases on June 15, 2014)
- ➢ Suffix with a counter and "custId": "cust061514:1:custId", "cust061514:2:custId", etc.

3. Benefits:

- ➢ Easier to write functions to retrieve ranges of values based on the prefix.

Partitioning with Keys in Key-Value Stores

- ➢ Partitioning is a way to organize data in a key-value store across multiple servers (nodes) in a cluster. This improves performance and scalability. Here's how keys are involved in partitioning:

1. Partitioning Methods:

- ➢ **Hashing:** A common method that scatters key-value pairs evenly across nodes using a hash function.

- ➢ **Range Partitioning:** Groups keys with similar values together and assigns them to the same node.

2. Range Partitioning :

- ➢ Requires a defined sorting order for keys (e.g., customer number).
- ➢ Groups consecutive values (ranges) and sends them to the same node.
- ➢ Needs a table to map key ranges to specific nodes (like Table 5.1).

3. Considerations for Range Partitioning:

- Carefully plan for future data growth, as changes might require reassigning keys and data migration between nodes.
- May not be suitable for all scenarios compared to hashing.

Range of Values	Assigned Node
cust:00001-cust:00999	Server 1
cust:01000-cust:01999	Server 2
cust:02000-cust:02999	Server 3
cust:04000-cust:04999	Server 4

Table 5.1 *Sample Range Partition Table*

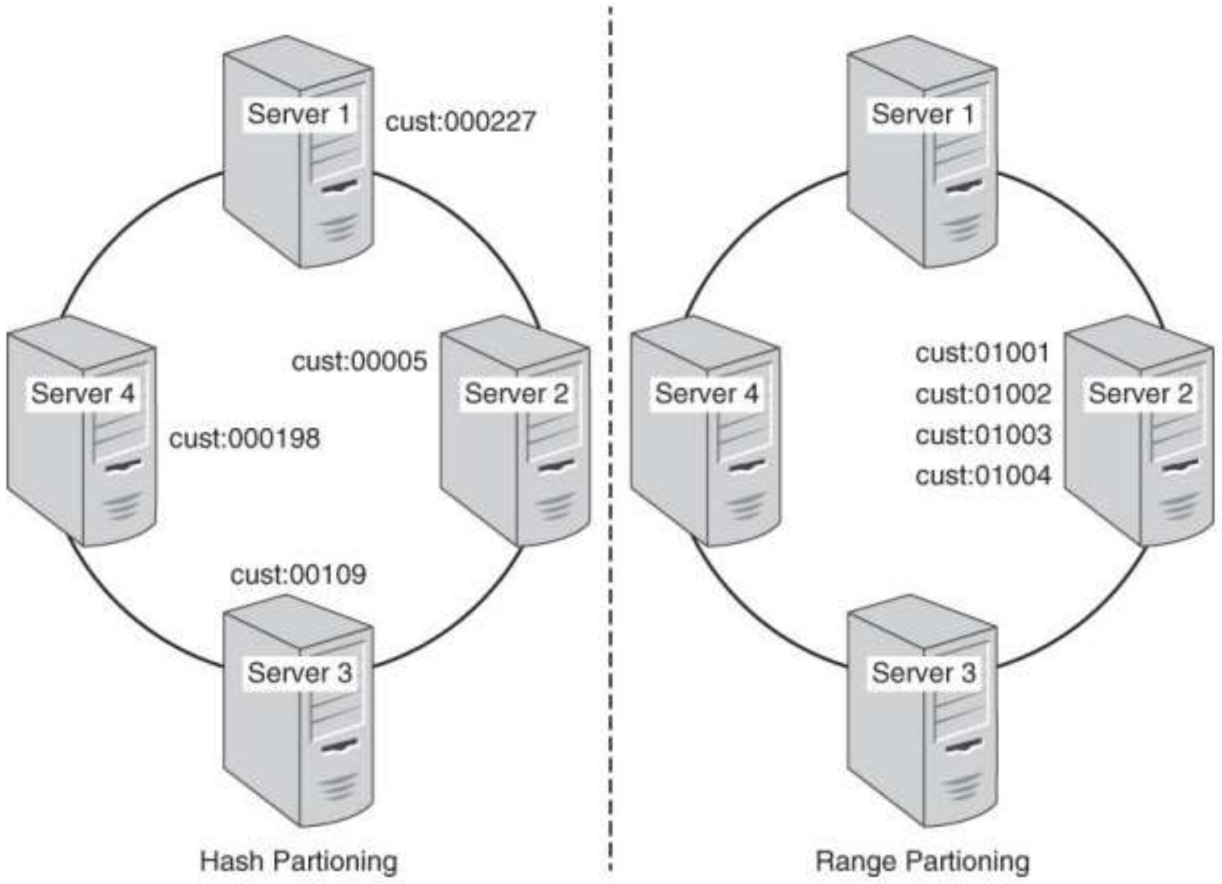


Figure 5.1 *Different hashing schemes will lead to different key-to-node assignments.*

2.Designing Structured Values

- Designing structured datatypes reduce the latency i.e., as we define a structure in programming language to make some datatypes as a group , we are grouping some data here such that when we need them we can get that data iin a single function call itself.

- **Ex:** If we need cust_id, cust_name, address everytime at once then we use structured datatypes.

- Fetching a value from the key-value database can take a long time, at least compared with primitive operations.

- The reason is that retrieving a value can require reading from a disk. This means that the read operation must wait for the read/write heads to get into position.

- The latency, or time you have to wait for the disk read to complete, is significantly longer than the time needed to perform other operations in the function

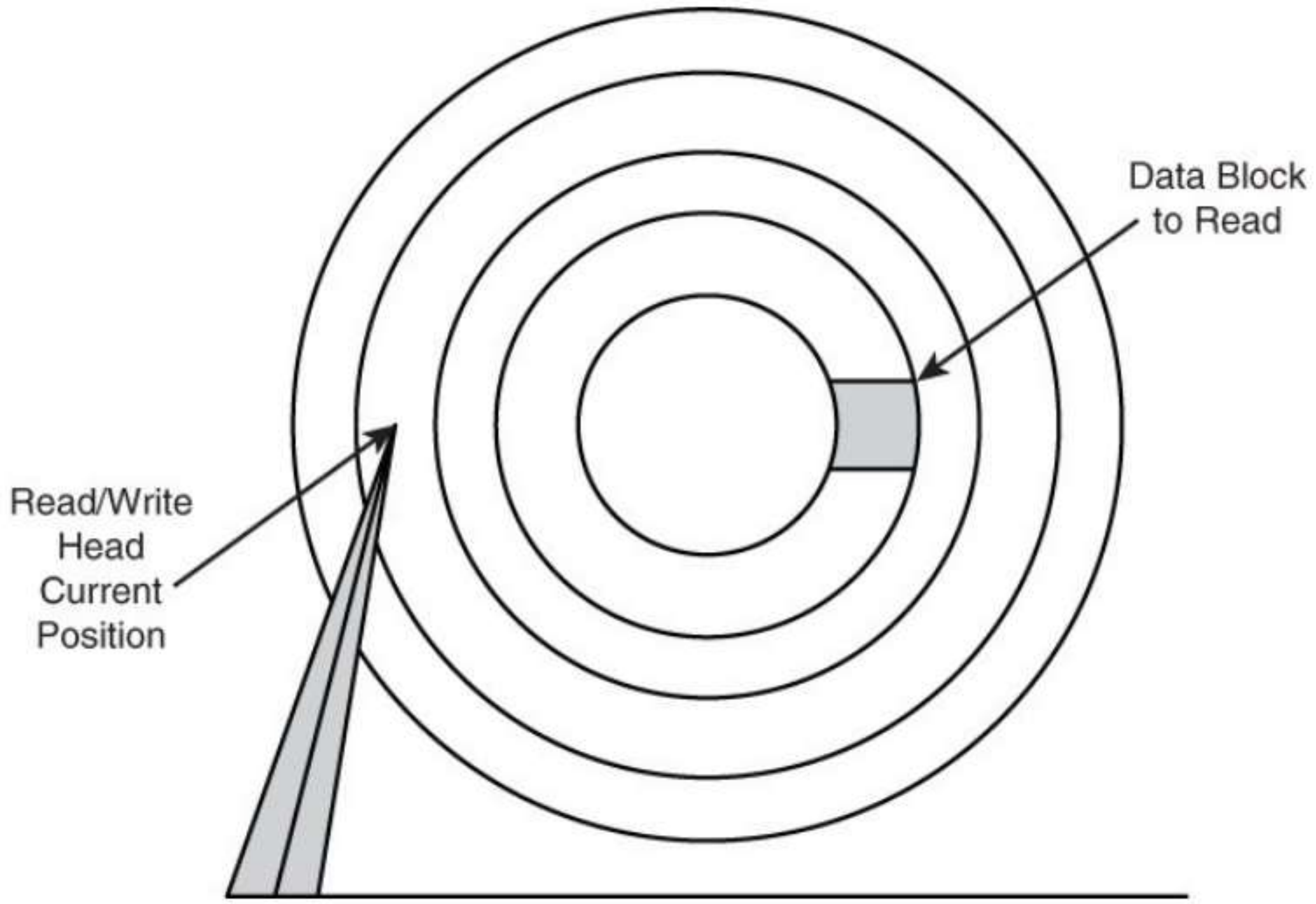


Figure 5.2 *Reading a value from disk requires the read/write heads to move to the proper track and the platter to rotate to the proper block. This can lead to long latencies.*

Solution

- One way to improve the speed of fetching values from the key-value database is to store frequently used values in memory.

- This works well in many cases but is limited by the amount of room in memory allocated to caching keys and values.

- Another approach is to store commonly used attribute values together

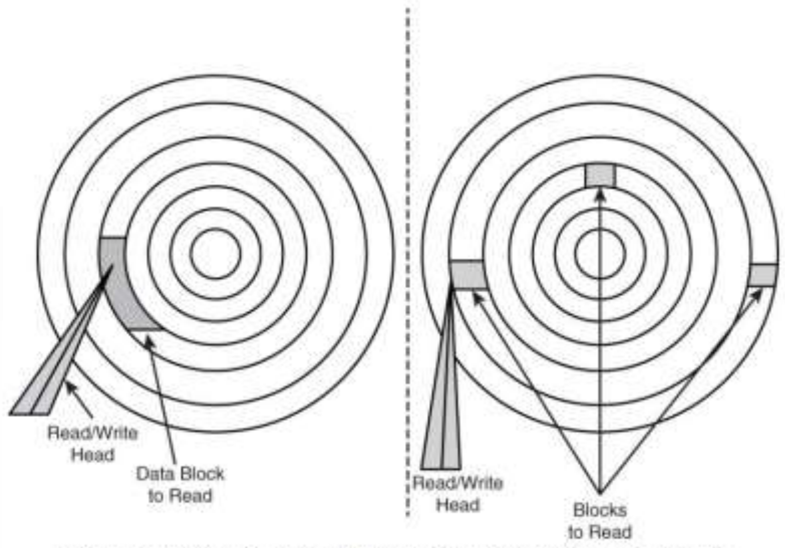


Figure 5.3 *Reading a single block of data is faster than reading multiple blocks referenced by multiple keys.*

- Key-value databases usually store the entire list together in a data block so there is no need to hash multiple keys and retrieve multiple data blocks.

- An exception to this rule occurs if the data value is larger than the disk data block size. This can occur if you store a large image or other sizeable object as a value

- **3.Limitations of Key-Value Databases.**

- The only way to look up values is by key

- Some key-value databases do not support range queries.

- There is no standard query language comparable to SQL for relational databases.

➢ 4.Design Patterns for Key-Value Databases

➢ Types of design patterns of key-value databases

- Time to Live (TTL) keys
- Emulating tables
- Aggregates
- Atomic aggregates
- Enumerable keys
- Indexes

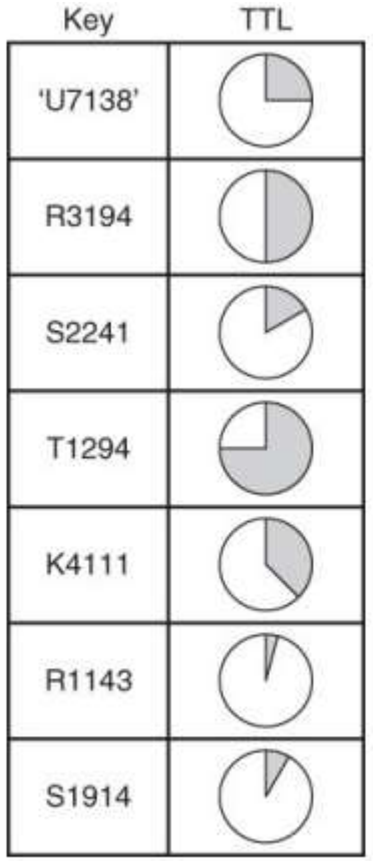


Figure 5.5 *Time to Live keys are useful for allowing users to reserve a product or resource for a limited time while other operations, such as making a payment, complete.*

1.Time to Live (TTL) in Key-Value Stores

•TTL is a parameter that specifies how long a piece of data remains valid.

•It's often used for temporary data or data with limited usefulness after a certain time.

Benefits of TTL in Key-Value Stores:

•**Caching:** Helps manage limited memory resources by automatically removing data after it expires.

•**Resource Management:** Ensures keys holding resources (e.g., shopping cart seats) are not held indefinitely.

•**Prevents Stale Data:** Removes outdated data that might lead to incorrect results.

Example:
•An e-commerce ticketing system holds seats for customers while they process payment.
•Keys with customer IDs and seat IDs can have a 5-minute TTL.
•This ensures seats aren't unavailable for too long if a customer abandons their cart.

2.Emulating Tables in Key-Value Stores

Emulate – to make a copy

•While most key-value stores don't support tables directly, emulation is possible using specific key naming conventions.

•This approach is useful for managing related sets of attributes that are frequently accessed together.

•It's not a perfect replacement for relational tables:

- Lacks features like SQL queries for complex filtering or range searches.
- Not scalable for emulating many tables or complex data relationships.

3.Aggregates

- Aggregation is a pattern that supports different attributes for different subtypes of an entity**.**

Relational Database Approach:

•Two main strategies:

- **Single Table:** Include all attributes for all subtypes, even unused ones for some subtypes (can become unwieldy).
- **Multiple Tables:** One table for common attributes and separate tables for each subtype (requires joins for complete data)

Key-Value Store Approach: Aggregation

•A single entity type is used for all subtypes.

•Values are stored as lists of attribute-value pairs specific to each subtype.

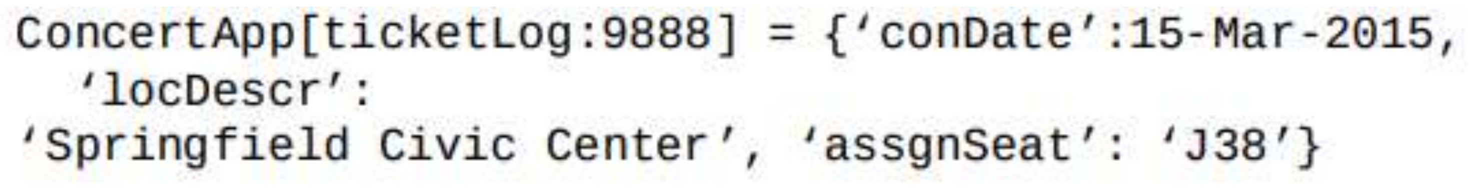
•A "type" indicator within the list specifies the subtype

4.Atomic Aggregates

- Atomic aggregates contain all values that must be updated together or not at all.
- A technique to achieve data consistency when transactions are unavailable.
- Stores multiple related values as a single unit using one key-value pair assignment.
- Ensures all values are updated successfully or none at all (similar to atomic transactions)
- Improves data consistency in key-value stores without full transactions.
- Reduces risk of inconsistencies caused by partial writes.

5.Enumerable Keys

- Enumerable keys are keys that use counters or sequences to generate new keys. This on its own would not be too useful; however, when combined with other attributes, this can be helpful when working with groups of keys.



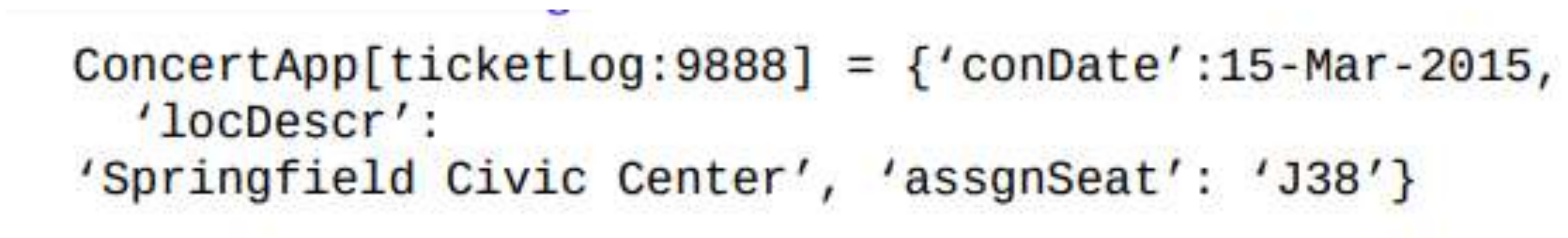
```
ConcertApp[ticketLog:9888] = {'conDate':15-Mar-2015,
  'locDescr':
'Springfield Civic Center', 'assgnSeat': 'J38'}
```

- The key is a combination of the entity name 'ticketLog' and a counter.

- The counter starts at 1 and increases by one each time a ticket is sold. This is suitable for recording information, but it does not help if you want to work with a range of logged values.

6.Indexes

- Inverted indexes are sets of key-value pairs that allow for looking up keys or values by other attribute values of the same entity.



```
ConcertApp[ticketLog:9888] = {‘conDate’:15-Mar-2015,
  ‘locDescr’:
‘Springfield Civic Center’, ‘assgnSeat’: ‘J38’}
```

- Concert ticket log stores details like date, location, and assigned seat for each ticket (identified by a unique key).

- Finding all seats assigned at a specific location (e.g., Springfield Civic Center) is not straightforward.

- Inverted indexes enhance the functionality of key-value stores by enabling efficient lookups based on attributes other than the primary key. This improves data searchability without relying on built-in search capabilities that might not be available in all key-value stores.

Course Code : 20CT1105 **R-2020** Reg. No:

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)
Madhurawada, Visakhapatnam
Affiliated to JNT University – K, Kakinada
B.Tech IV Semester Regular & Supplementary Examinations, June 2023
Database Management Systems
(Common to CSE (DS), CSE (AI & ML) & IT)

Date: 26-06-2023 **Time: 3 Hours** **Max. Marks: 70**

1. Answer ONE Question from each UNIT
2. All parts of a Question must be answered at one place to get valued.
3. All questions carry equal marks.

UNIT-I

1. a) Explain in detail about the three-tier schema architecture of DBMS. 7 Marks
   b) Define Entity, Entity set, Attribute with respect to ER model.List different types of attributes along with their symbols. 7 Marks

2. a) With a neat diagram describe the overall system architecture of DBMS. 7 Marks
   b) Develop an ER diagram for keeping track of information about a company database taking into account at least five entities. 7 Marks

UNIT-II

3. a) Consider the following schema: Suppliers (sid, sname, address), Parts (pid, pname, color) and Catalog (sid, pid, cost) Write the relational algebraic queries for the following: 7 Marks
   i)Find the sids of suppliers who supply some red or green part
   ii) Find the sids of suppliers who supply every red or green part
   iii) Find the pids of parts supplied by at least two different suppliers
   b) What are the basic relational algebra operators that query languages have? Explain with examples. 7 Marks

4. a) Discuss the Entity Integrity and Referential Integrity constraints. Why is each considered important? 7 Marks
   b) What is a view? How views are implemented? 7 Marks

UNIT-III

5. a) Illustrate with suitable example 3NF and BCNF. 7 Marks
   b) Consider the relation R(A,B,C,D) with set of functional dependencies 7 Marks
   $F = \{A \rightarrow B, BC \rightarrow D, A \rightarrow C\}$. Answer the following;
   (i) Identify the candidate key(s) for R.
   (ii) (ii) Identify the best normal form that R satisfies (1NF, 2NF, 3NF, or BCNF).
   (iii) If R is not in BCNF, decompose it into a set of BCNF relations that preserve the dependencies.

6. a) Illustrate with suitable example partial, full and transitive functional dependencies. 7 Marks
   b) For the relation R(V,W,X,Y,Z) with the functional dependencies $F=\{X \rightarrow YV, Y \rightarrow Z, Z \rightarrow Y, VW \rightarrow X\}$ determine any two candidate keys of R. Also specify the highest normal form of R. 7 Marks

Course Code : 20CT1105 **R-2020** Reg. No:

UNIT-IV

7. a) List the ACID properties of a transaction. Explain the usefulness of each. 7 Marks
   b) What is conflict serializable schedule? Illustrate with suitable example. 7 Marks

8. Explain in detail about concurrency control techniques (Lock Based and Timestamp based protocols) with an example. 14 Marks

UNIT-V

9. a) Write the essential features of key-value pair databases? 7 Marks
   b) Explain about motivations for using no-sql databases? 7 Marks

10. a) What is a document database? Write basic operations on document databases. 7 Marks
    b) Explain about the features of column-family databases? 7 Marks

Course Code :20CT1105/ 2020 **R-2020** Reg. No:

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)
Madhurawada, Visakhapatnam
Affiliated to JNT University – K, Kakinada
B.Tech IV Semester Supplementary Examinations, Dec. 2022 /January 2023
Database Management Systems
[Common to CSE (AI&ML), CSE (Data Science) & IT]

Date: 05-01-2023 **Time: 3 Hours** **Max. Marks: 70**

1. Answer ONE Question from each UNIT
2. All parts of a Question must be answered at one place to get valued.
3. All questions carry equal marks.

UNIT-I

1. a) Explain different types of database users and write the functions of DBA. 7 Marks
   b) Distinguish strong entity set with weak entity set. Illustrate with an ER diagram. 7 Marks
2. a) Differentiate between File processing system Vs Database Management System. 7 Marks
   b) Write short notes on the following: 7 Marks
   i) DDL ii) DML iii) TCL

UNIT-II

3. a) Differentiate between Tuple Relational Calculus and Domain Relational Calculus. 7 Marks
   b) Explain join operations in relational algebra. 7 Marks
4. a) Discuss about the following keys: 7 Marks
   i) Primary Key ii) Candidate Key iii) Foreign Key iv) Super Key
   b) What is a view? How will you create, update and delete views? 7 Marks

UNIT-III

5. a) What is Normalization? Discuss its merits and demerits and also explain 1NF and 2 NF with suitable examples. 7 Marks
   b) List out the Problems related to decompositions. Explain about Loss less and Lossy decompositions with an example. 7 Marks
6. a) Illustrate Multivalued dependencies and fourth normal form with suitable example. 7 Marks
   b) Define Armstrong's axioms for FD's. 7 Marks

UNIT-IV

7. a) Explain the concept of serial schedule and serializable schedule. 7 Marks
   b) Discuss Static and Dynamic hashing techniques. 7 Marks
8. a) Explain two phase locking protocol with an example. 7 Marks
   b) Distinguish between B Trees and B+ Trees. 7 Marks

UNIT-V

9. a) What is NoSQL? Discuss its advantages over traditional RDBMS. 7 Marks
   b) What is CAP theorem? Discuss its properties. 7 Marks
10. Discuss the types of NoSQL databases. 14 Marks

Course Code : 20CT1105/ 2020 **R-2020** Reg. No :

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)
Madhurawada, Visakhapatnam
Affiliated to JNT University – K, Kakinada
B.Tech IV Semester Regular Examinations, July/August 2022
Database Management Systems
[Common to CSE, CSE (AI&ML), CSE (DS) & IT]

Date: 01-08-2022 **Time: 3 Hours** **Max. Marks: 70**

1. Answer ONE Question from each UNIT
2. All parts of a Question must be answered at one place to get valued.
3. All questions carry equal marks.

UNIT-I

1. a) Draw the Structure of DBMS and explain the components of DBMS system structure. 7 Marks
   b) Write the concept of Specialization, Generalization and Aggregations. Illustrate with an ER diagrams. 7 Marks
2. a) Explain different levels of data abstraction in a DBMS. And discuss about Data Independence. 7 Marks
   b) What are the elements of ER diagram? Illustrate with an example. 7 Marks

UNIT-II

3. a) What is a View? Give its importance in databases and give syntax for Creating, Dropping and Updating a View. 7 Marks
   b) State about SELECT and PROJECT operations in Relational algebra? Give one example for each. 7 Marks
4. a) Write short notes of Joins used in databases. 7 Marks
   b) Illustrate different set operations in Relational algebra with an example. 7 Marks

UNIT-III

5. a) Define normalization. Explain 1NF, 2NF Normal forms with suitable examples? 7 Marks
   b) Explain Lossless and Lossy decomposition with an example. 7 Marks
6. a) Write short notes on the following: 7 Marks
   i) Multivalued dependencies ii) 4NF
   b) What are the problems caused by redundancy? Explain each one with an example. 7 Marks

UNIT-IV

7. a) What is a transaction? Explain its desirable properties. 7 Marks
   b) Discuss two phase locking protocol with three variations of S2PL, C2PL and R2PL. 7 Marks
8. a) What is view serializability? Explain with suitable example. 7 Marks
   b) Write in detail about Hash based Indexing and Tree based Indexing. 7 Marks

UNIT-V

9. a) What are the main differences between the four types of NoSql databases (KeyValue Store, Column-Oriented Store, Document-Oriented, Graph Database)? 7 Marks
   b) What are the applications of a document databases? Explain. 7 Marks
10. a) Explain BASE terminology in a context of NoSQL. 7 Marks
    b) What is CAP theorem how it is applicable to NoSQL systems? 7 Marks

Good luck
on
semester
exams!