Course code : **CSE3009**
Course title : **No SQL Data Bases**
Module : **3**
Topic : **3**

Key-Value Database
Data Modeling Terms

Objectives

This session will give the knowledge about

- Key-Value Database Data Modeling Terms
- Key-Value Architecture Terms
- Key-Value Implementation Terms

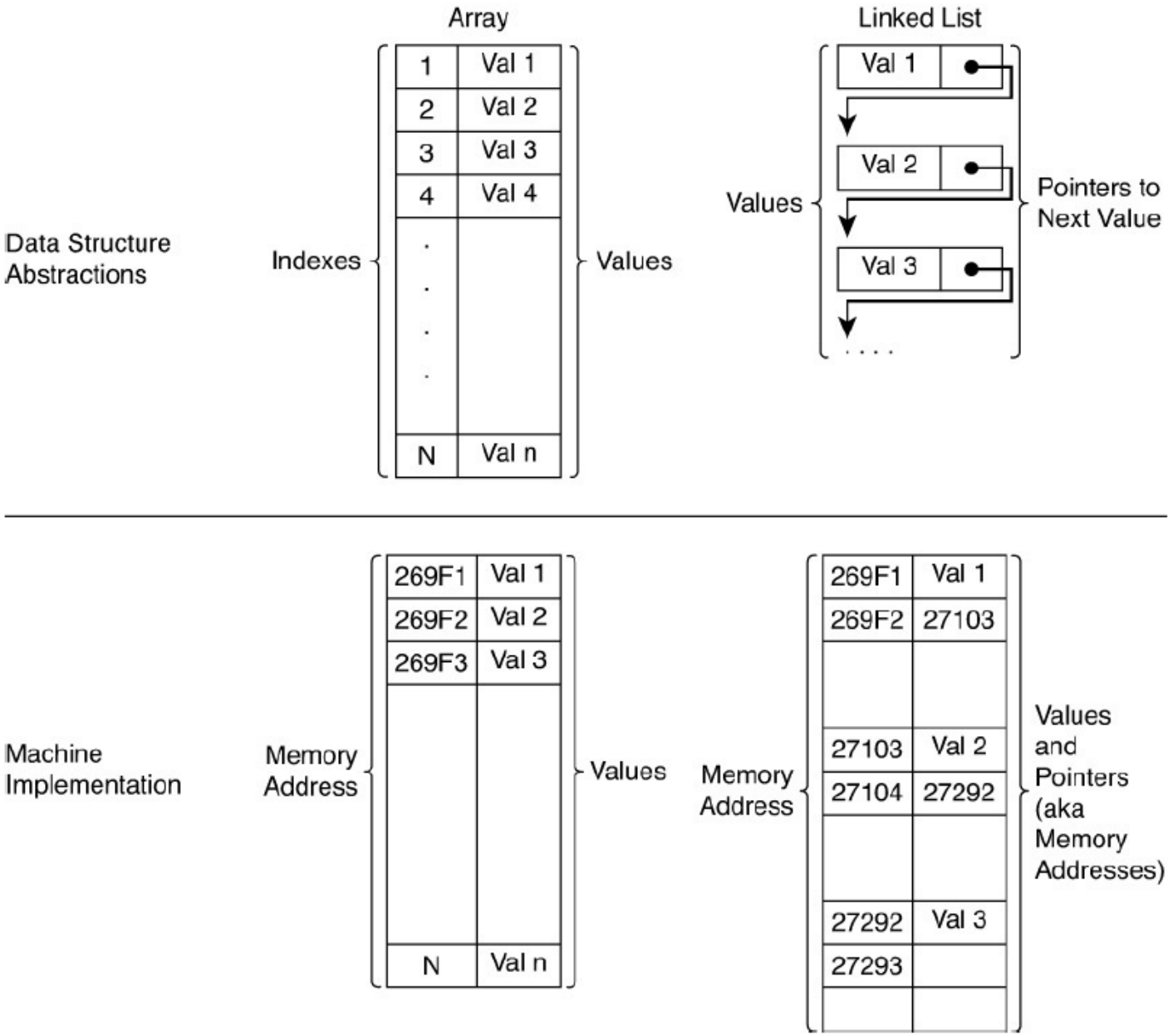
Data Modeling Terms

Data models are abstractions that helps in organize the information conveyed by the data in databases. They are different from data structures.

Data structures are well-defined data storage structures that are implemented using elements of underlying hardware, particularly RAM and persistent data storage, such as hard drives and flash devices.

For example, an integer variable in a programming language may be implemented as a set of four contiguous bytes, or 32 bits. An array of 100 integers can be implemented as a contiguous set of 4-byte memory addresses.

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


Array
Linked List
Data Structure Abstractions
Indexes
Values
1
Val 1
2
Val 2
3
Val 3
4
Val 4
N
Val n
Values
Pointers to Next Value
Val 1
Val 2
Val 3
Machine Implementation
Memory Address
Values
269F1
Val 1
269F2
Val 2
269F3
Val 3
N
Val n
Memory Address
Values and Pointers (aka Memory Addresses)
269F1
Val 1
269F2
27103
27103
Val 2
27104
27292
27292
Val 3
27293

Data Modeling Terms

Data models provide a layer of abstraction above data structures that allows database application developers to focus more on the information that must be managed and less on implementation issues.

Data models typically organize multiple kinds of related information.

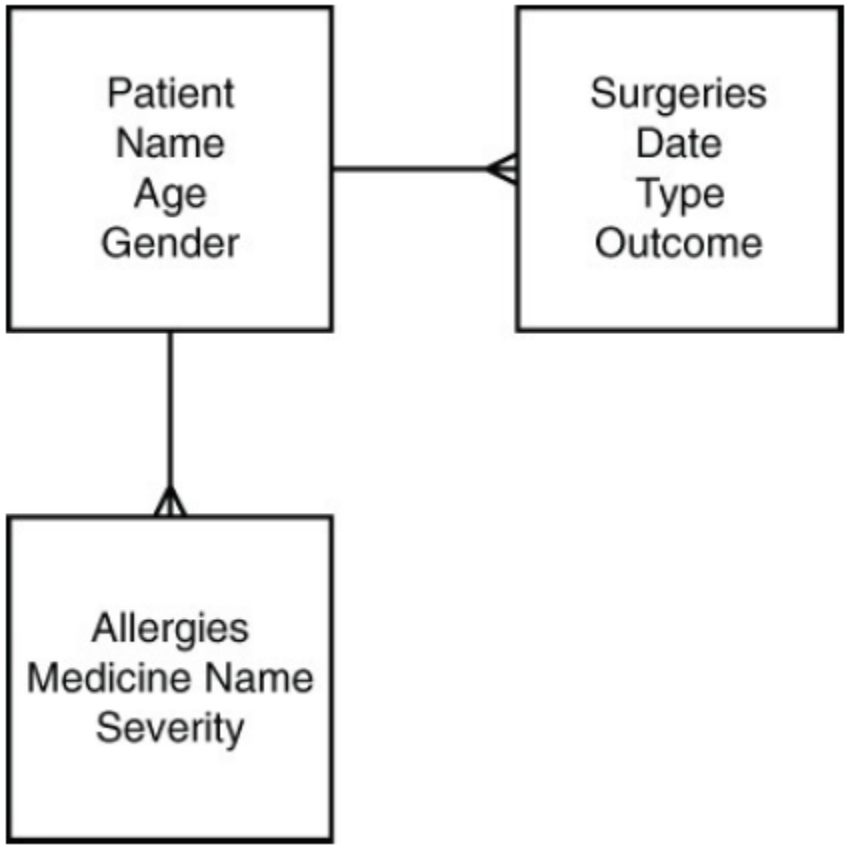
The elements of data models vary with the type of database. Relational databases are organized around tables.

Tables are used to store information about entities, such as customers, patients, orders, and surgeries. Entities have attributes that capture information about particular entities.

Data Modeling Terms

A customer management data model could model information about customers' names, addresses, orders, and payment histories.

Clinical databases could include information such as patients' names, ages, genders, current prescriptions, past surgeries, allergies, and other medically relevant details.



Key

Keys in key-value databases are similarly not values but are ways of finding and manipulating values.

The address 1232 NE River St. is a reference to a building located in a particular place. Among other things, it enables postal workers and delivery services to find a particular building and drop off or pick up letters and packages.

The string “1232 NE River St.” is obviously not a building, but it is a way to find the corresponding building.

Key

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".

In Redis more complex data structures are used as keys. The supported key data types in Redis version 2.8.13 include

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

Key

Redis developers use the term data structures server instead of key-value data store.

- 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.

Key - Limitations

Keep in mind that strings should not be too long. Long keys will use more memory and key-value databases tend to be memory-intensive systems already. At the same time, avoid keys that are too short. Short keys are more likely to lead to conflicts in key names. For example, the key

CMP:1897:Name

could refer to the name of a marketing campaign or the name of a component in a product. A better option would be

CAMPN:1897:Name

to refer to a marketing campaign and

COMPT:1897:Name

to refer to a component in a product.

Value

The definition of value with respect to key-value databases is so vague that it is almost not useful.

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.

Value

It is important to understand that different implementations of key-value databases have different restrictions on values.

Most key-value databases will have a limit on the size of a value.

Redis, for example, can have a string value up to 512MB in length.

FoundationDB (foundationdb.com), a key-value database known for its support of ACID transactions, limits the size of values to 100,000 bytes.

Key-value implementations will vary in the types of operations supported on values. At the very least, a key-value database will support getting and setting values.

Namespace

A namespace is a collection of key-value pairs.

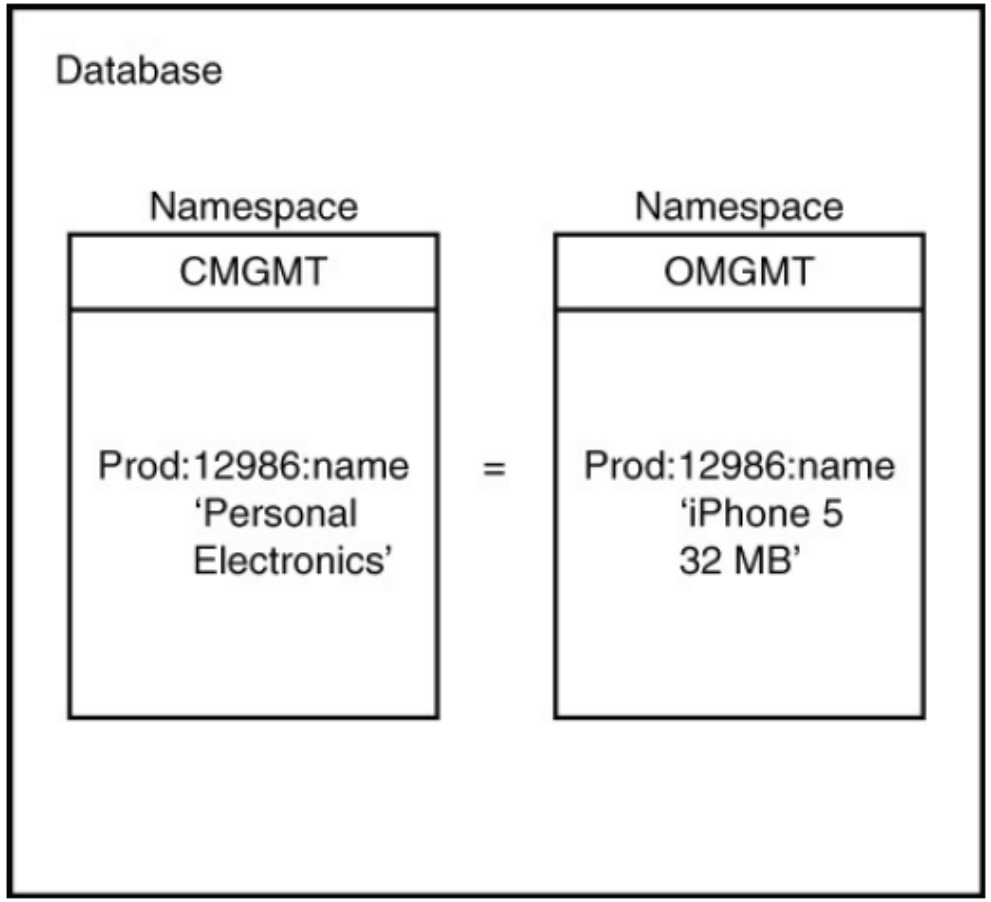
You can think of a namespace as a set, a collection, a list of key-value pairs without duplicates, or a bucket for holding key-value pairs.

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. Namespaces are helpful when multiple applications use a key-value database.

Namespace

Namespaces enable duplicate keys to exist without causing conflicts by maintaining separate collections of keys



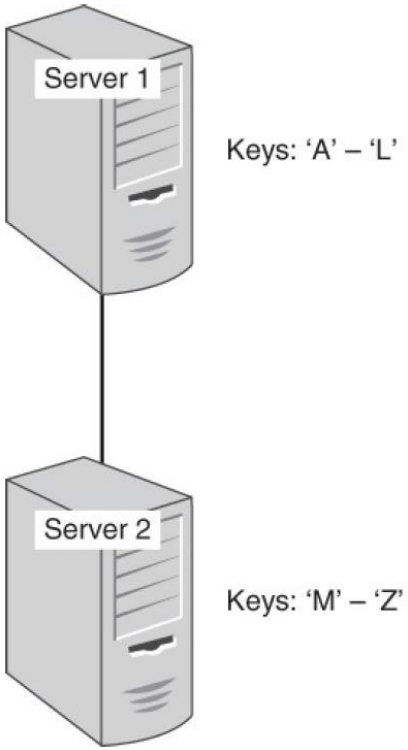
Partition

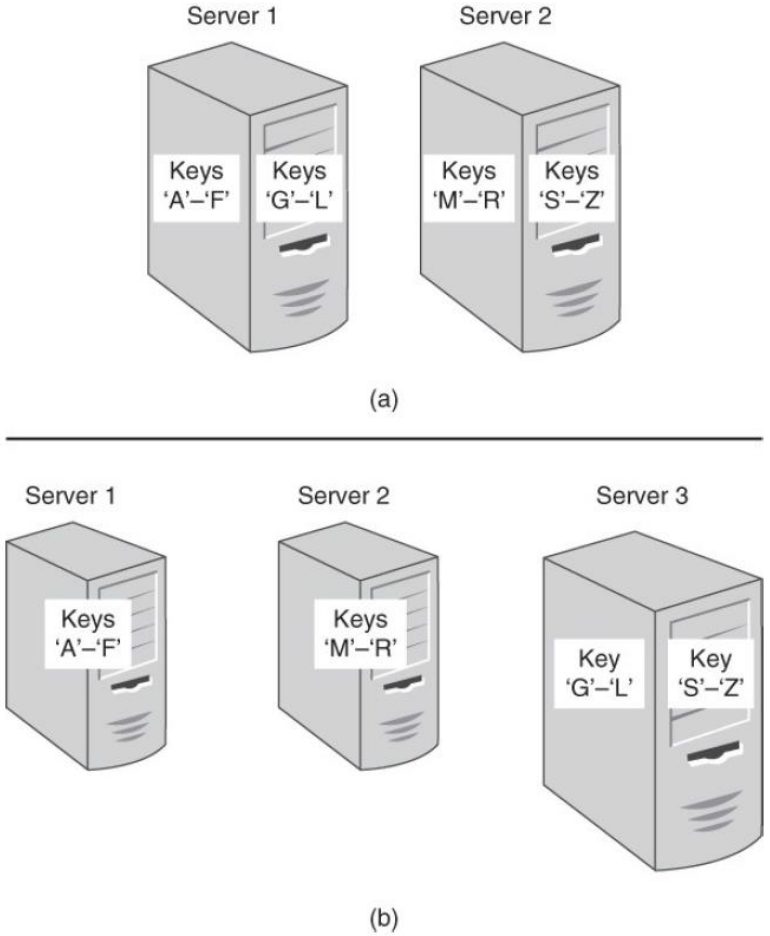
A partitioned cluster is a group of servers in which servers or instances of key-value database software running on servers are assigned to manage subsets of a database.

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. There are several ways to handle this.

You could simply decide that all keys starting with the letters A through L are handled by Server 1 and all keys starting with M through Z are managed by Server 2. In this case, you are partitioning data based on the first letter of the key (see Figure 1).

Partition


Server 1
Keys: 'A' – 'L'
Server 2
Keys: 'M' – 'Z'


Server 1
Server 2
Keys 'A'–'F'
Keys 'G'–'L'
Keys 'M'–'R'
Keys 'S'–'Z'
(a)
Server 1
Server 2
Server 3
Keys 'A'–'F'
Keys 'M'–'R'
Key 'G'–'L'
Key 'S'–'Z'
(b)

Partition

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

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 (Figure-2).

Note that a server may support more than one partition. This can happen if servers are running virtual machines and each virtual machine supports a single partition.

Alternatively, key-value databases may run multiple instances of partition software on each server. This allows for a number of partitions larger than the number of servers.

Partition Key

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

In the previous example, the first letter of a key name is used to determine which partition manages it.

Other simple strategies are partitioning by numeric value and string value.

Any key in a key-value database is used as a partition key; good partition keys are ones that distribute workloads evenly.

Schemaless

Schemaless is a term that describes the logical model of a database. In the case of key value databases, you are not required to define all the keys and types of values you will use prior to adding them to the database.

Schemaless data models allow you to make changes as needed without changing a schema that catalogs all keys and value types (see Figure).

Schemaless data models allow for multiple types of representations of the same data to exist simultaneously.

Key-Value Database	
Keys	Values
cust : 8983 : firstName	'Jane'
cust : 8983 : lastName	'Anderson'
cust : 8983 : fullName	'Jane Anderson'

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 terms frequently appear when discussing key-value architectures:

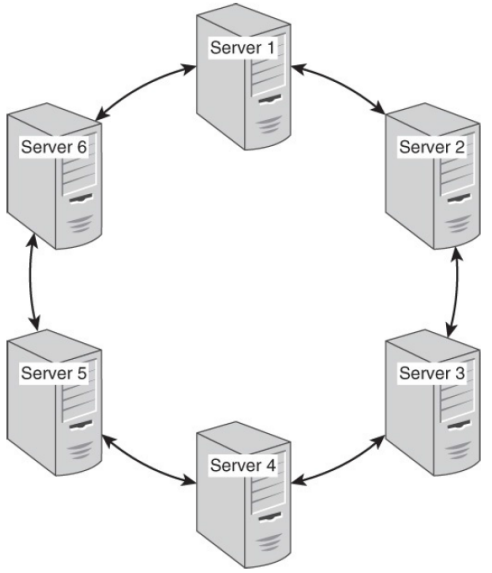
- Clusters
- Rings
- Replication

Clusters

Clusters are sets to connected computers that coordinate their operations (see Figure). Clusters may be loosely or tightly coupled.

Loosely coupled clusters consist of fairly independent servers that complete many functions on their own with minimal coordination with other servers in the cluster.

Tightly coupled clusters tend to have high levels of communication between servers. This is needed to support more coordinated operations, or calculations, on the cluster. Key-value clusters tend to be loosely coupled.



A ring architecture of key-value databases links adjacent nodes in the cluster

Ring

A ring is a logical structure for organizing partitions. A ring is a circular pattern in which each server or instance of key-value database software running on a server is linked to two adjacent servers or instances. Each server or instance is responsible for managing a range of data based on a partition key.

Consider a simple hashlike function that maps a partition key from a string; for example, 'cust:8983:firstName' to a number between 0 and 95.

Now assume that you have an eight-node cluster and the servers are labeled Server 1, Server 2, Server 3, and so on.

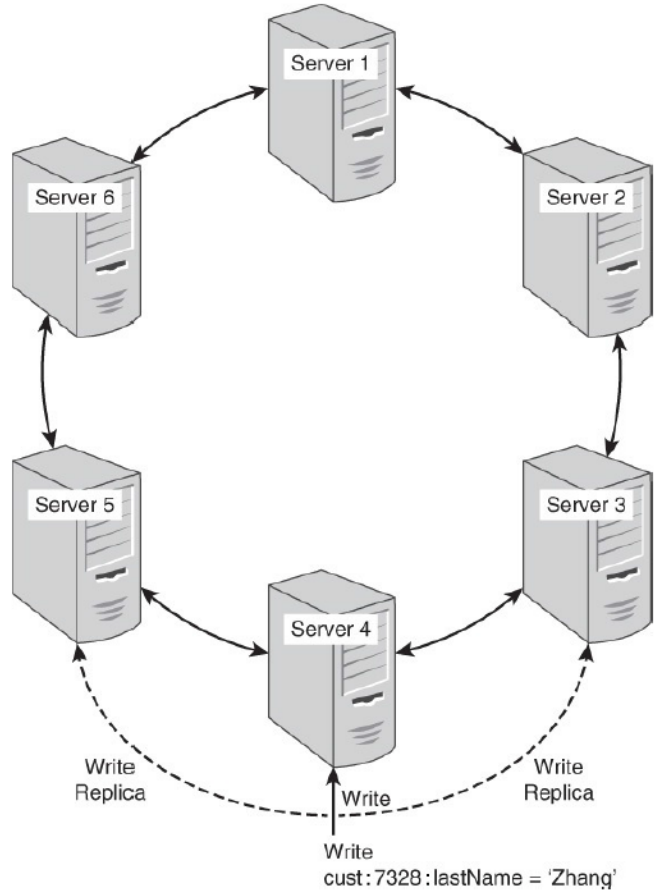
With eight servers and 96 possible hash like values, you could map the partitions to servers, as shown in next Table

Ring

In this model, Server 2 is linked to Server 1 and Server 3; Server 3 is linked to Server 2 and Server 4; and so on. Server 1 is linked to Server 8 and Server 2. Refer to Figure to see a graphical depiction of a ring architecture.

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

Server Name	Partition Range
Server 1	0–11
Server 2	12–23
Server 3	24–35
Server 4	36–47
Server 5	48–59
Server 6	60–71
Server 7	72–83
Server 8	84–95



Replication

Replication is the process of saving multiple copies of data in your cluster. This provides for high availability as described previously.

One parameter you will want to consider is the number of replicas to maintain. The more replicas you have, the less likely you will lose data; however, you might have lower performance with a large number of replicas.

If your data is easily regenerated and reloaded into your key-value database, you might want to use a small number of replicas.

If you have little tolerance for losing data, a higher replica number is recommended.

Key-Value Implementation Terms

The terms discussed in this last set of key-value vocabulary deal with topics you generally do not work with directly.

These terms cover operations that happen behind the scenes of application programs but are nonetheless crucial to the functioning of a key-value database.

- Hash Function
- Collision
- Compression

Hash Function

Hash functions are algorithms that map from an input—for example, a string of characters to an output string. The size of the input can vary, but the size of the output is always the same.

For example, a simple string like 'Hello world' maps to “2aae6c35c94fcfb415dbe95f408b9ce91ee846ed”

Hash functions are generally designed to distribute inputs evenly over the set of all possible outputs. The output space can be quite large.

Hash Function

For example, the SHA-1 has 2^160 possible output values. This is especially useful when hashing keys. No matter how similar your keys are, they are evenly distributed across the range of possible output values. The ranges of output values can be assigned to partitions and you can be reasonably assured that each partition will receive approximately the same amount of data.

For example, assume you have a cluster of 16 nodes and each node is responsible for one partition. You can use the first digit output by the SHA-1 function to determine which partition should receive the data.

As you might recall, the SHA-1 function outputs a hexadecimal, or base-16, number. The hexadecimal digits are 0–9 and a–f for a total of 16 digits.

Collision

A collision occurs when two distinct inputs to a hash function produce the same output.

When it is difficult to find two inputs that map to the same hash function output, the hash function is known as collision resistant.

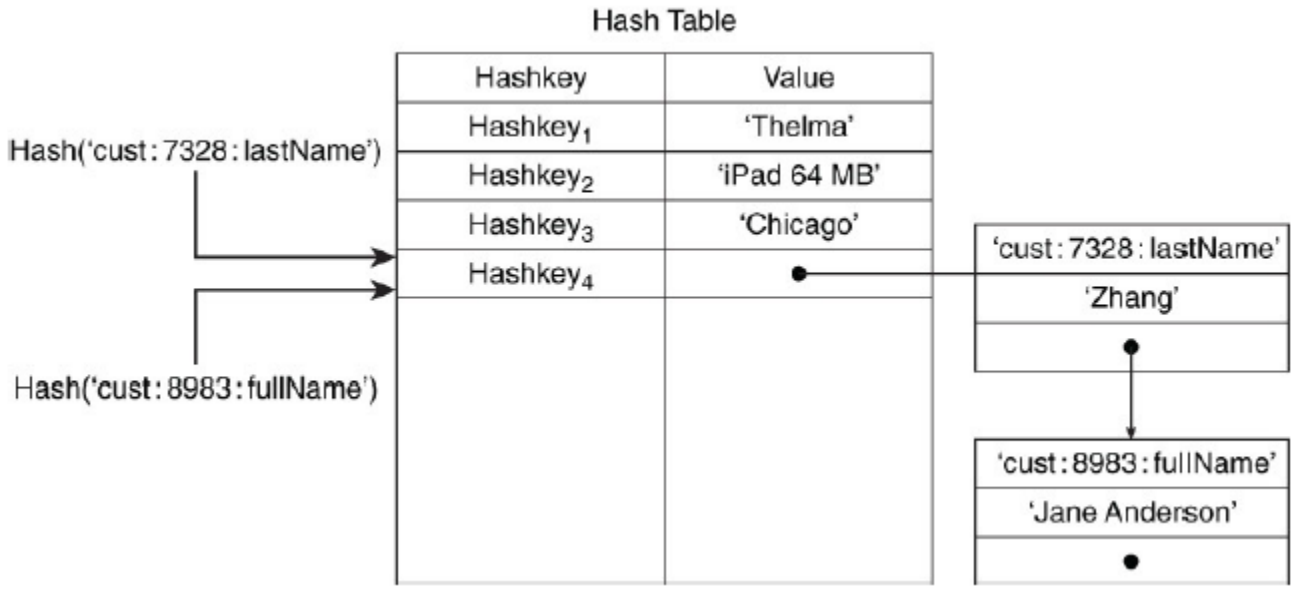
If a hash table is not collision resistant or if you encounter one of those rare cases in which two inputs map to the same output, you will need a collision resolution strategy.

A simple method to deal with this is to implement a list in each cell of a hash table. Most entries will include a single value, but if there are collisions, the hash table cell will maintain a list of keys and values, as shown in Figure.

Collision

This is a logical representation of a generic solution to the collision problem; actual implementations may vary.

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



Compression

Key-value databases are memory intensive. Large numbers of large values can quickly consume substantial amounts of memory.

One way to optimize memory and persistent storage is to use compression techniques.

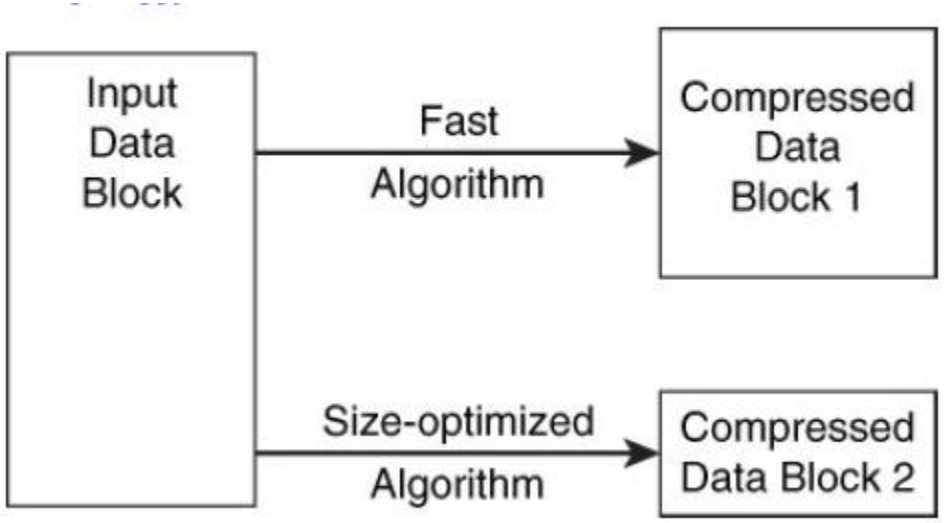
A compression algorithm for key-value stores should perform compression and decompression operations as fast as possible.

This often entails a trade-off between the speed of compression/decompression and the size of the compressed data. Faster compression algorithms can lead to larger compressed data than other, slower algorithms (see Figure).

Compression

For example, the Snappy compression algorithm compresses 250MB per second and decompresses 500MB per second on a Core i7, 64-bit mode processor but produces compressed data that is 20% to 100% larger than the same data compressed by other algorithms.

Compression algorithms may be designed to optimize for speed or data size.



Summary

This session will give the knowledge about

- Key-Value Database Data Modeling Terms
- Key-Value Architecture Terms
- Key-Value Implementation Terms
- Reference: NoSQL for Mere Mortals by Dan Sullivan