Course code : **CSE3009**
Course title : **No SQL Data Bases**
Module : **3**
Topic : **1**

Key –Value Data Stores

Objectives

This session will give the knowledge about

- From array to key –value databases
- Essential features of key – value Databases

Array to key –value databases

After scalar variables, like integers and characters, the array is one of the simplest.

An array is an ordered list of values. Each value in the array is associated with an integer index. The values are all the same type.

- exampleArray[0] = ‘Hello world.’
- exampleArray[1] = ‘Goodbye world.’
- exampleArray[2] = ‘This is a test.’
- exampleArray[3] = 3.1415

You might see the two following limitations when working with arrays:

- The index can only be an integer.
- The values must all have the same type.

Associative Arrays

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. For example:

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

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

Associative Arrays

Associative arrays generalize the idea of an ordered list indexed by an identifier to include arbitrary values for identifiers and values.

Values stored in the associative array can vary.

Associative arrays go by a number of different names, including dictionary, map, hash map, hash table, and symbol table.

Associative arrays are the basic structure underlying the concept of key-value databases.

Caches

Key-value databases build on the concept of an associative array, but there are important differences.

Many key-value data stores keep persistent copies of data on long-term storage, such as hard drives or flash devices.

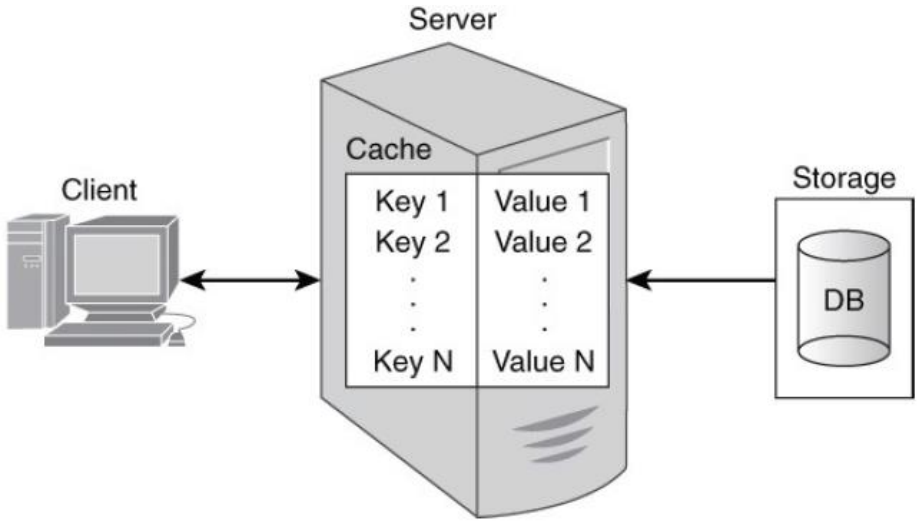
Some key-value data stores only keep data in memory. These are typically used so programs can access data faster than if they had to retrieve data from disk drives

Caches

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

The program could run faster if it retrieved data from memory rather than from the database.

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.



Caches

If the program is relatively simple and only needs to track one customer at a time, then the application programmer could use character-string variables to store the customer name and address information.

When a program must track many customers and other entities at the same time, then using a cache makes more sense.

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-Memory and On-Disk Key-Value Database

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.

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.

In-Memory and On-Disk Key-Value Database

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

Database

Bucket 1	
'Foo1'	'Bar'
'Foo2'	'Bar2'
'Foo3'	'Bar7'

Bucket 2	
'Foo1'	'Baz'
'Foo4'	'Baz3'
'Foo6'	'Baz2'

Bucket 3	
'Foo1'	'Bar7'
'Foo4'	'Baz3'
'Foo7'	'Baz9'

Key-value stores

A key-value store is a simple database that when presented with a simple string (the key) returns an arbitrary large BLOB of data (the value).

Key-value stores have no query language; they provide a way to add and remove key-value pairs into/from a database.

A key-value store is like a dictionary.

The dictionary is a simple key-value store where word entries represent keys and definitions represent values.

Key-value stores

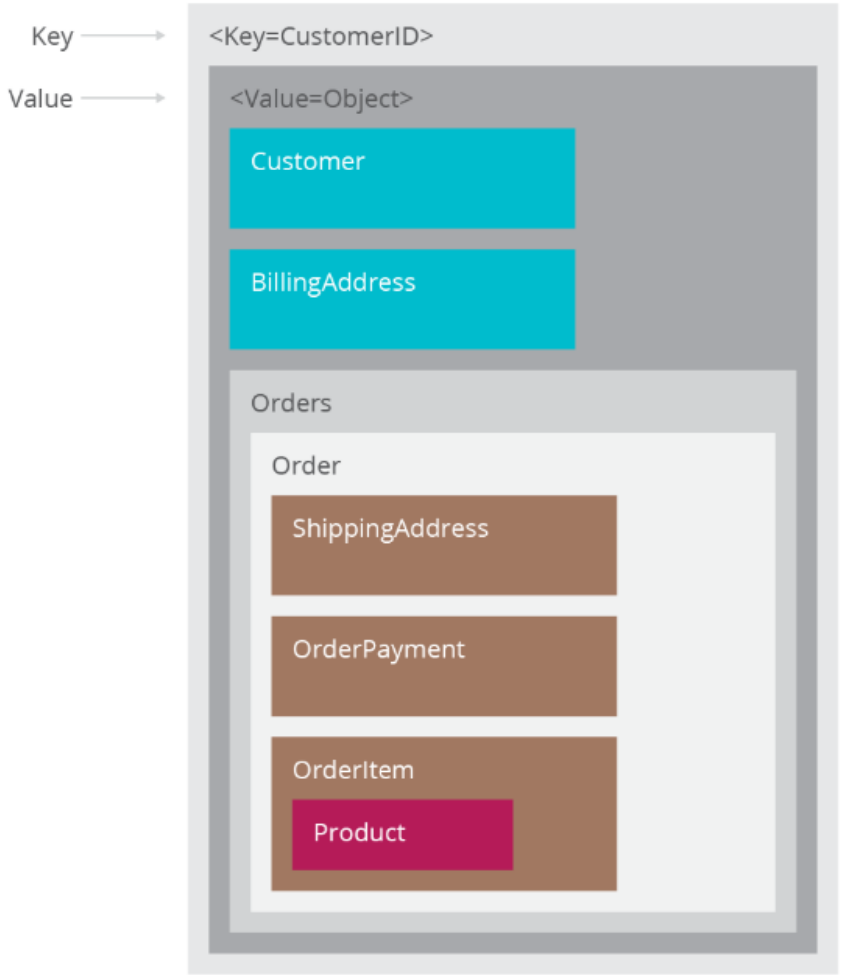
The key value type basically, uses a hash table in which there exists a unique key and a pointer to a particular item of data.

A bucket is a logical group of keys – but they don't physically group the data. There can be identical keys in different buckets.

The key can be synthetic or auto-generated while the value can be String, JSON, BLOB (basic large object) etc.

To read a value you need to know both the key and the bucket because the real key is a hash (Bucket + Key).

Key-value stores – Data types



Key-value stores – Data types

The key in a key-value store is flexible and can be represented by many formats:

- Logical path names to images or files
- Artificially generated strings created from a hash of the value
- REST web service calls
- SQL queries

Values, like keys, are also flexible and can be any BLOB of data, such as images, web pages, documents, or videos.

Sample items in a key-value store

	Key	Value
Image name →	image-12345.jpg	Binary image file
Web page URL →	http://www.example.com/my-web-page.html	HTML of a web page
File path name →	N:/folder/subfolder/myfile.pdf	PDF document
MD5 hash →	9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
REST web service call →	view-person?person-id=12345&format=xml	<Person><id>12345</id .</Person>
SQL query →	SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id .</Person>

Key-value stores - Operations

In addition to the put, get, and delete API, a key-value store has two rules: distinct keys and no queries on values:

1. Distinct keys - You can never have two rows with the same key-value. This means that all the keys in any given key-value store are unique.

2. No queries on values - You can't perform queries on the values of the table.

Key-value stores - Operations

A key-value store prohibits where clause type of operation, as you can't select a key-value pair using the value.

The key-value store resolves the issues of indexing and retrieval in large datasets by transferring the association of the key with the value to the application layer, allowing the key-value store to retain a simple and flexible structure.

Essential Features of Key-Value Databases

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

- Simplicity
- Speed
- Scalability

These characteristics sound like an ideal combination that should be embraced by every database, but as you will see, there are limitations that come along with these valued features.

Simplicity

Key-value databases use a bare-minimum data structure.

In key-value databases, you work with a simple data model. The syntax for manipulating data is simple.

Typically, you specify a namespace, which could be a database name, a bucket name, or some other type of collection name, and a key to indicate you want to perform an operation on a key-value pair.

When you specify only the namespace name and the key, the key-value database will return the associated value. When you want to update the value associated with a key, you specify the namespace, key, and new value.

Simplicity

Key-value databases are flexible and forgiving. If you make a mistake and assign the wrong type of data, for example, a real number instead of an integer, the database usually does not complain.

This feature is especially useful when the data type changes or you need to support two or more data types for the same attribute.

If you need to have both numbers as strings for customer identifiers, you can do that with code such as the following:

- shoppingCart[cart:1298:customerID] = 1982737
- shoppingCart[cart:3985:customerID] = ‘Johnson, Louise’

Speed

Major database vendors create tools to help developers and database administrators identify slow-running queries.

Key-value databases are known for their speed. With a simple associative array data structure and design features to optimize performance, key-value databases can deliver high-throughput, data-intensive operations.

One way to keep database operations running fast is to keep data in memory. Reading and writing data to RAM is much faster than writing to a disk.

Speed

Key-value databases can have the advantages of fast write operations to RAM and the persistence of disk-based storage by using both.

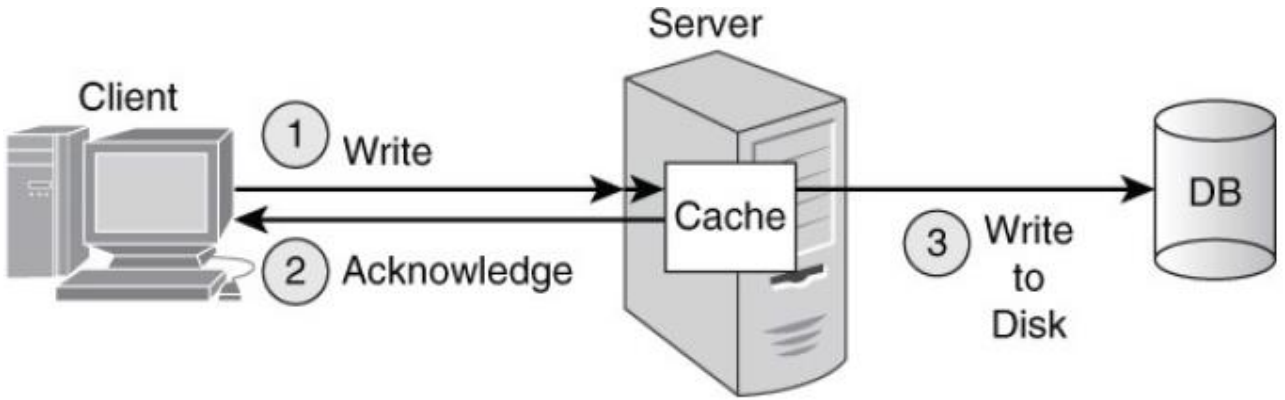
When a program changes the value associated with a key, the key-value database can update the entry in RAM and then send a message to the program that the updated value has been saved.

The program can then continue with other operations. While the program is doing something else, the key-value database can write the recently updated value to disk.

Speed

The new value is saved to disk unless there is a power loss or some other failure between the time the application updates the value and the key-value database stores the value on disk.

Write operations can return control to the calling application faster by first writing inserts and updates to RAM and then updating disk storage.



Speed

Similarly, read operations can be faster if data is stored in memory. This is the motivation for using a cache, as described earlier.

Because the size of the database can exceed the size of RAM, key-value stores have to find ways of managing the data in memory.

When the key-value database uses all the memory allocated to it, the database will need to free some of the allocated memory before storing copies of additional data.

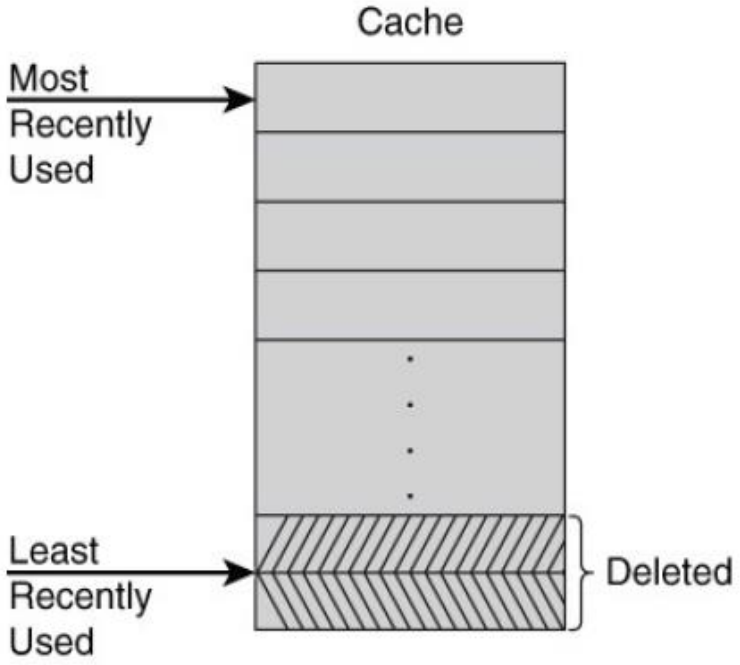
There are multiple algorithms for this, but a commonly used method is known as least recently used (LRU).

Speed

The idea behind the LRU algorithm is that if data has not been used in a while, it is less likely to be used than data that has been read or written more recently.

This intuition makes sense for many application areas of key-value databases.

Least recently used algorithms delete data that has not been read or written as recently as other data.



Scalability

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

- Master-slave replication
- Masterless replication

Scaling with Master-Slave Replication

One way to keep up with a growing demand for read operations is to add servers that can respond to queries. A master-slave replication model works well in this case.

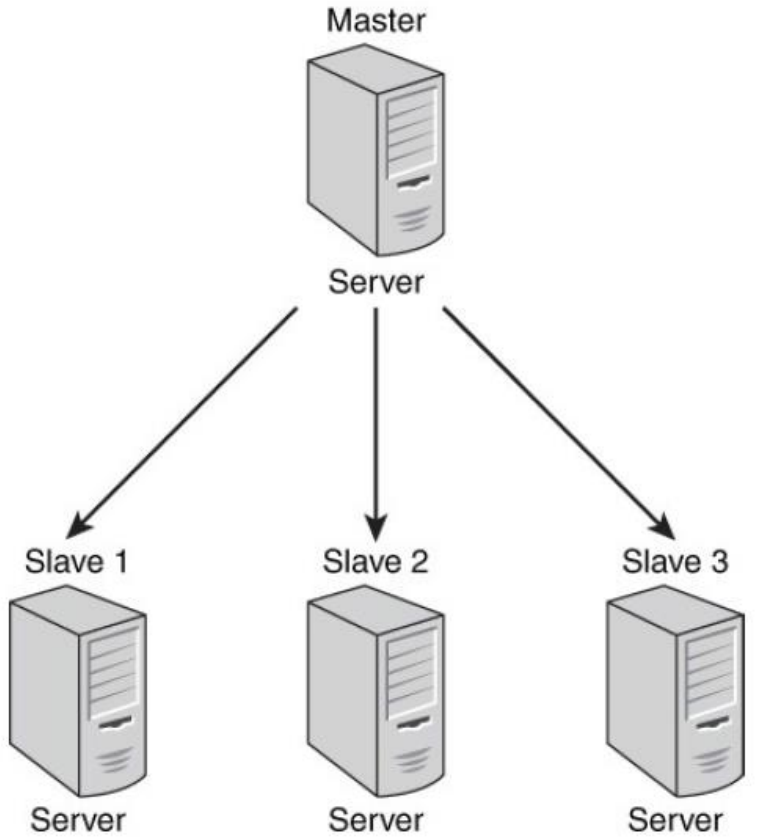
Scalability: Master-slave architectures

Master-slave architectures have a simple communication pattern during normal operations.

The master is a server in the cluster that accepts write and read requests.

It is responsible for maintaining the master record of all writes and replicating, or copying, updated data to all other servers in the cluster.

These other servers only respond to read requests.



Scalability: Master-slave architectures

An advantage of master-slave models is simplicity. Except for the master, each node in the cluster only needs to communicate with one other server: the master.

The master accepts all writes, so there is no need to coordinate write operations or resolve conflicts between multiple servers accepting writes.

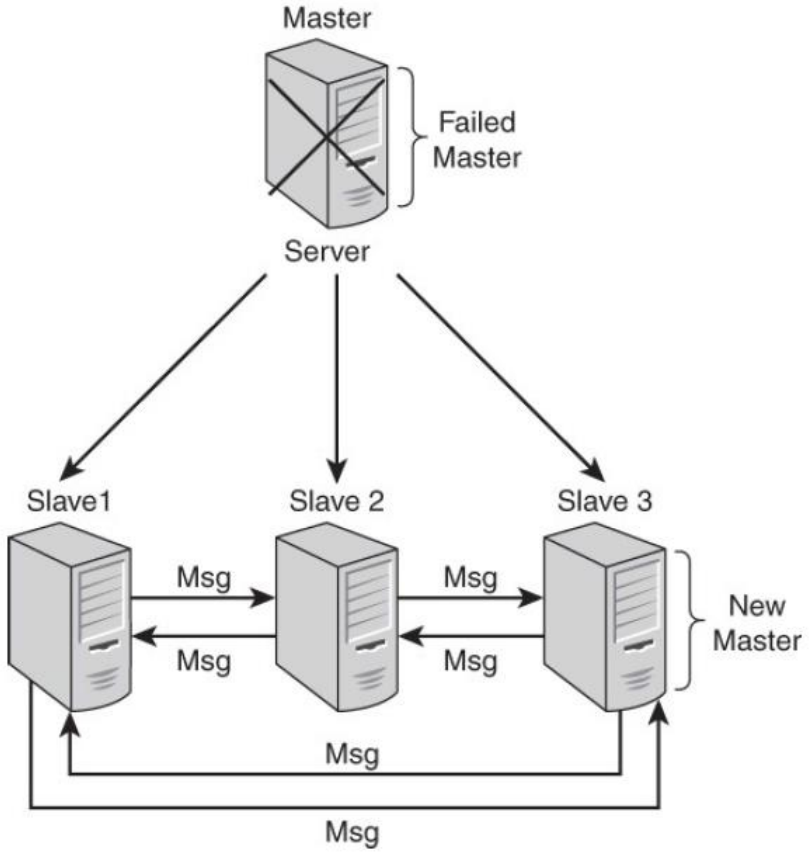
A disadvantage of the master-slave replication model is that if the master fails, the cluster cannot accept writes.

This can adversely impact the availability of the cluster The master server is known as a single point of failure.

Scalability: Master-slave architectures

In the case of master-slave configurations, if a number of slave servers do not receive a message from the master within some period of time, the slaves may determine the master has failed.

At that point, the slaves initiate a protocol to promote one of the slaves to master



Scalability: Masterless architectures

Scaling with Masterless Replication

The master-slave replication model with a single server accepting writes does not work well when there are a large number of writes.

A better option for this application is a masterless replication model in which all nodes accept reads and writes.

In a masterless replication model, there is not a single server that has the master copy of updated data, so no single server can copy its data to all other servers. Instead, servers in a masterless replication model work in groups to help their neighbors.

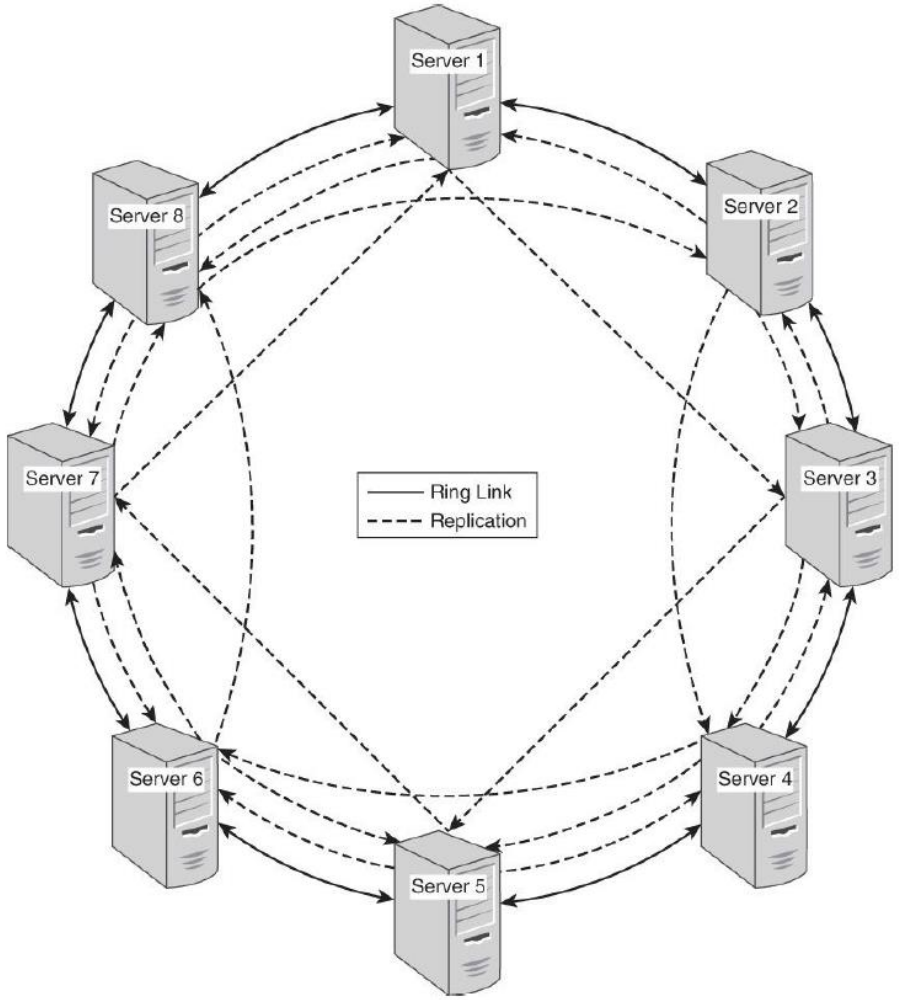
Scalability: Masterless architectures

An eight-server cluster in a ring configuration with a replication factor of 4. Database administrators can configure a key-value database to keep a particular number of replicas.

In this scenario, the administrator has decided that fou replicas are sufficient.

Each time there is a write operation to one of the servers, it replicates that change to the three other servers holding its replica.

In this scenario, each server replicates to its two neighbors and to the server two links ahead.



Summary

This session will give the knowledge about

- From array to key –value databases
- Essential features of key – value Databases
- Reference: NoSQL for Mere Mortals by Dan Sullivan