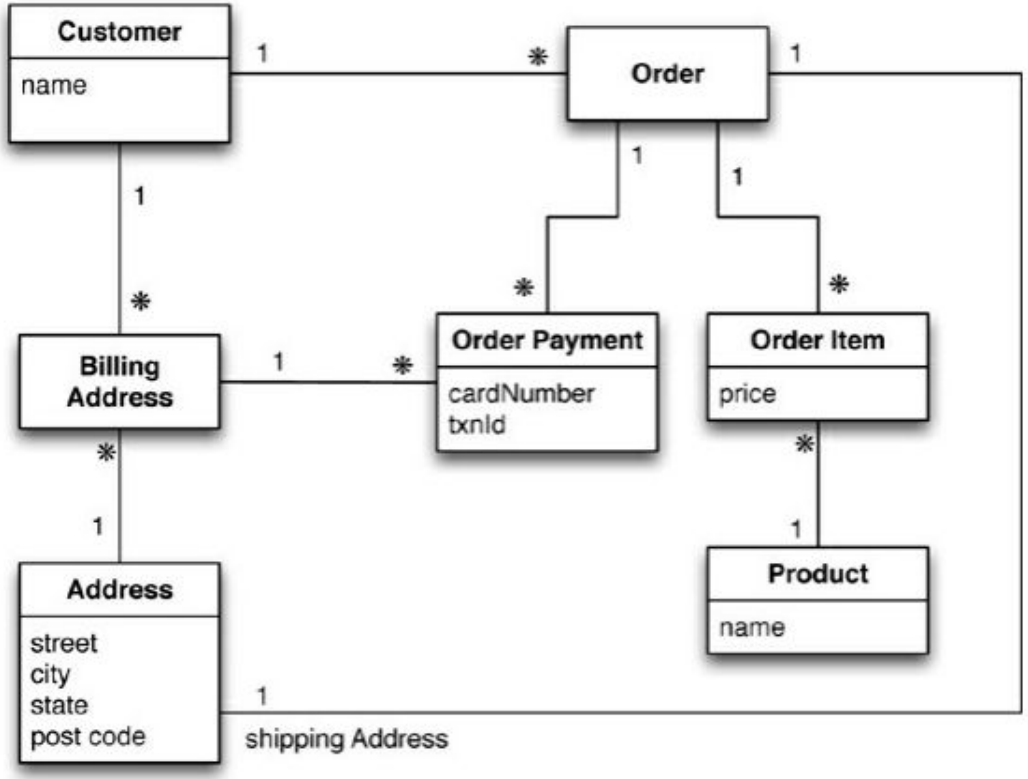
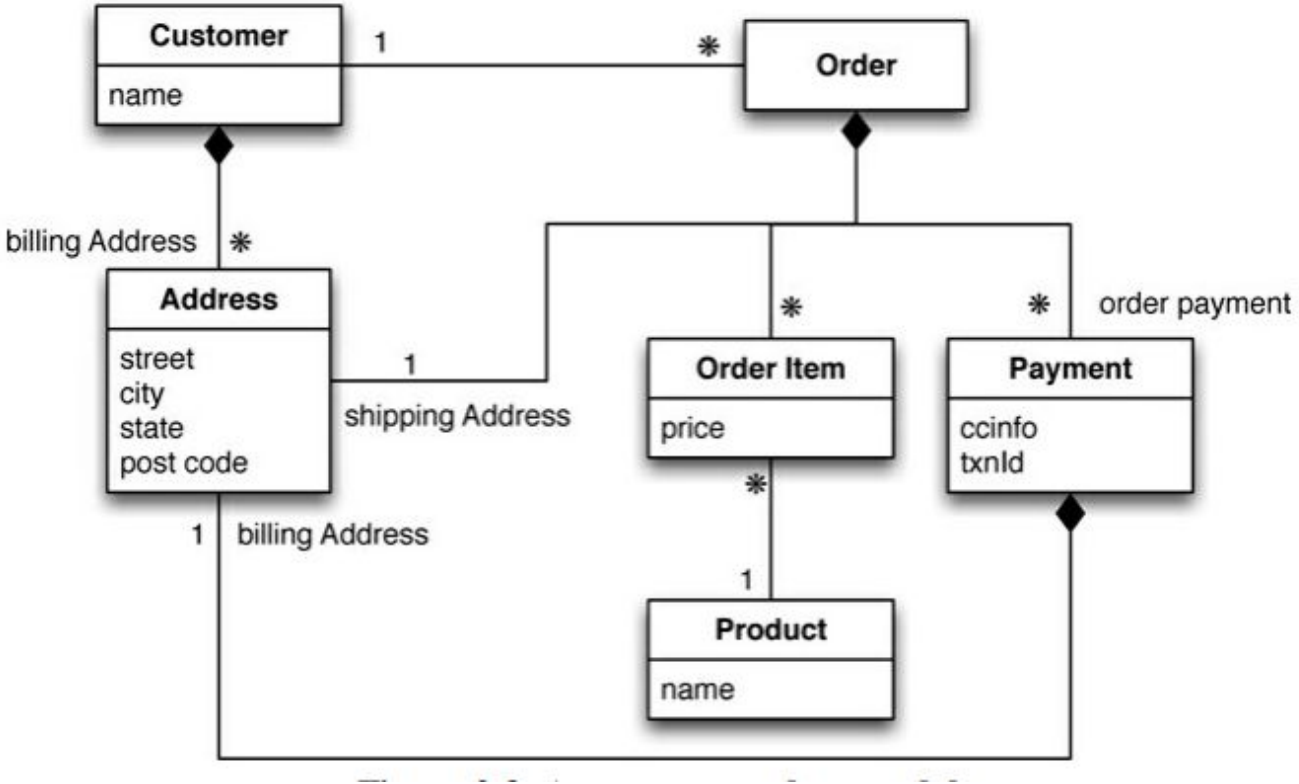
Unit - 2

Relational data model



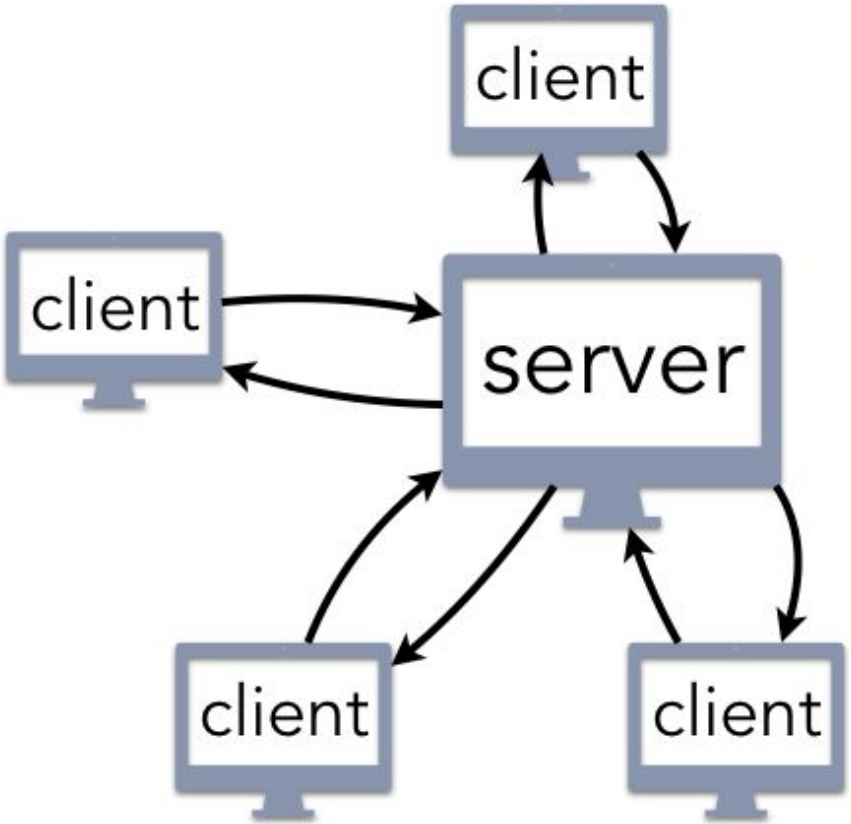
Aggregate data model



Distribution Models

- Single server
- Sharding
- Master-Slave Replication
- Peer-to-Peer Replication
- Combining Sharding & Replication

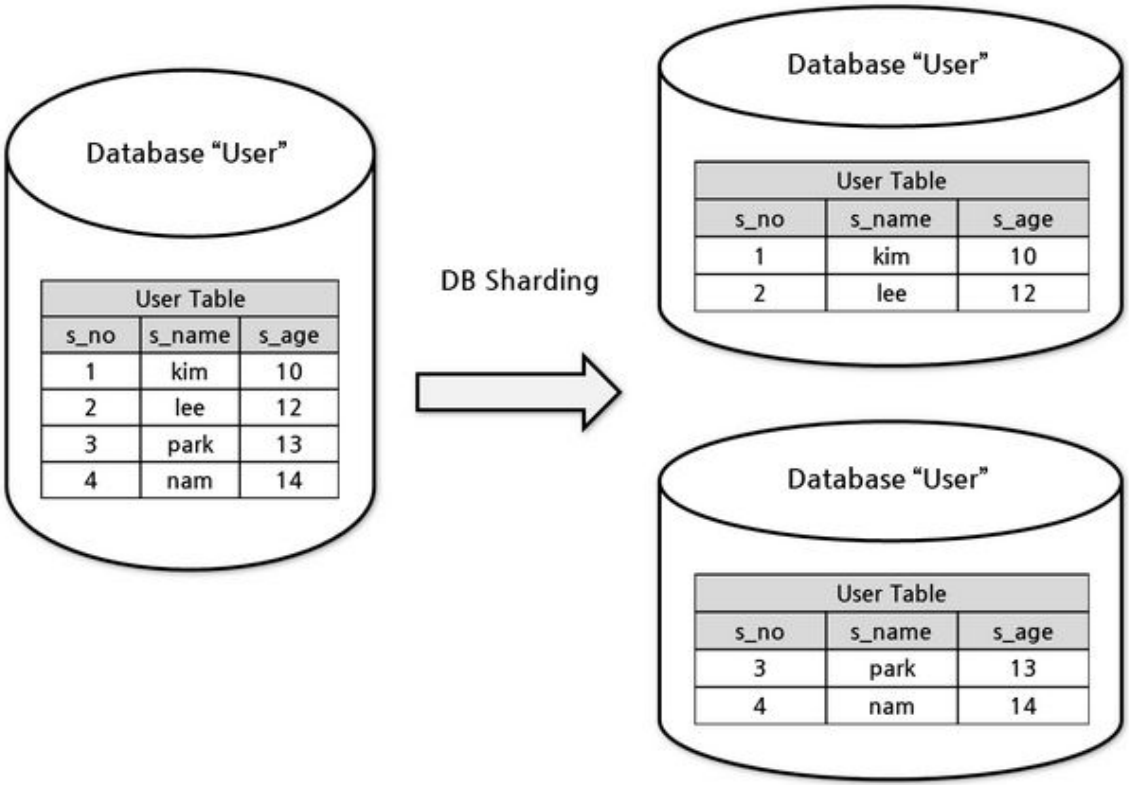
Single Server



Single Server

- No distribution at all
- Single server handles all the read & write requests
- Graph databases work better on a single server architecture

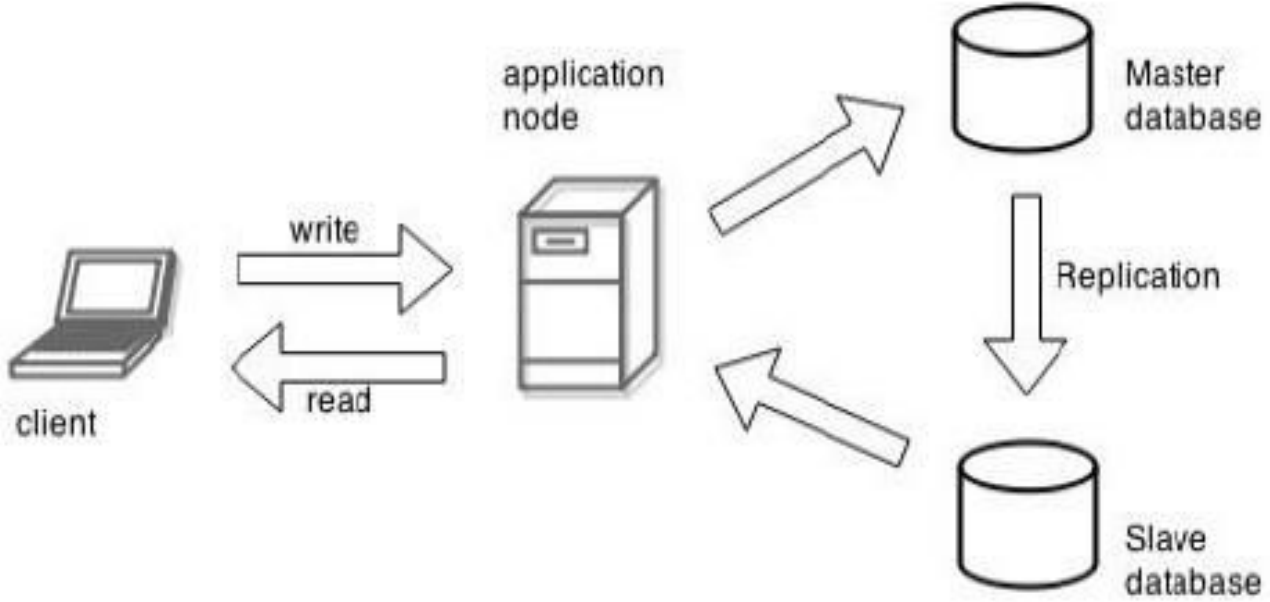
Sharding


Database “User”
User Table
s_no s_name s_age
1 kim 10
2 lee 12
3 park 13
4 nam 14
DB Sharding
Database “User”
User Table
s_no s_name s_age
1 kim 10
2 lee 12
Database “User”
User Table
s_no s_name s_age
3 park 13
4 nam 14

Sharding

- Different users working with different data
- Related data needs to be clumped together
- Another factor is load balancing
- Auto-sharding
- Node failures

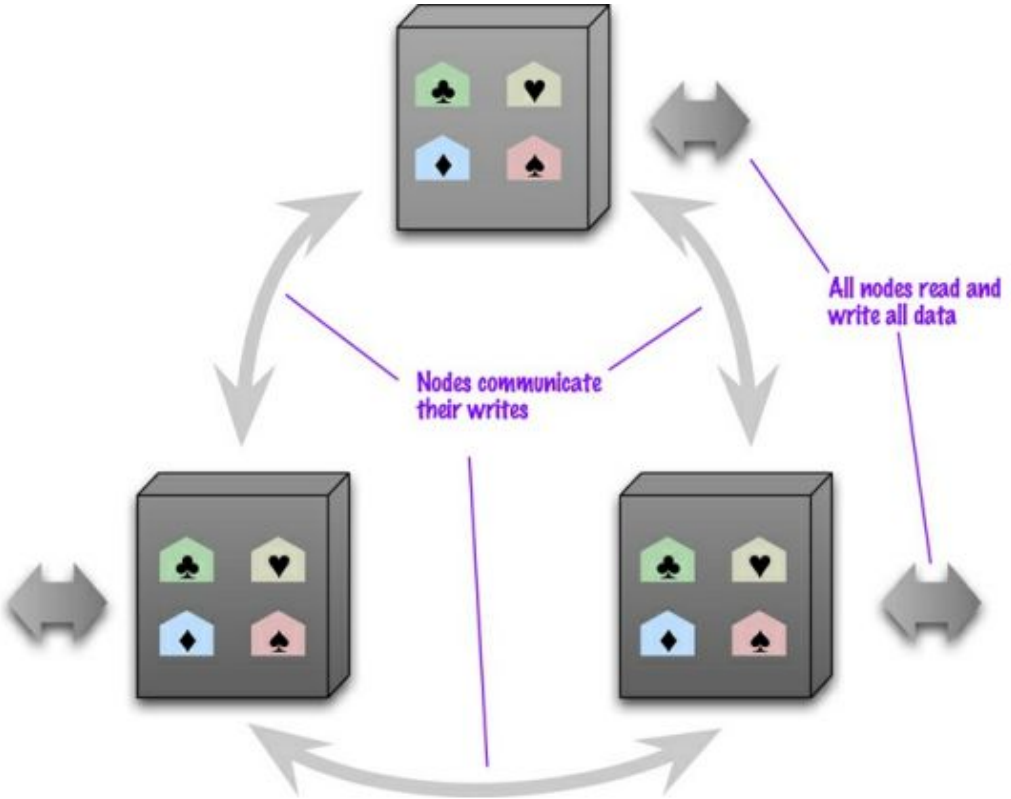
Master slave replication



Master Slave Replication

- A single master and multiple slaves
- Master handles the writes and slaves handles the reads
- Read resilience
- Master node failure

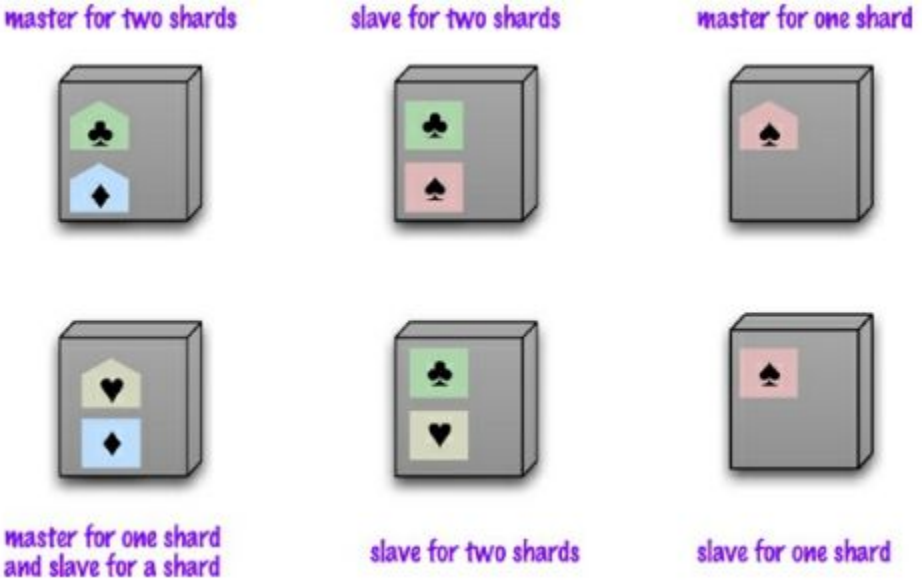
Peer - Peer replication



Peer to peer Replication

- No master at all
- All nodes can accept reads & writes
- Provides the write resilience as well
- Consistency problems
  - Write-write conflict

Combining replication & sharding



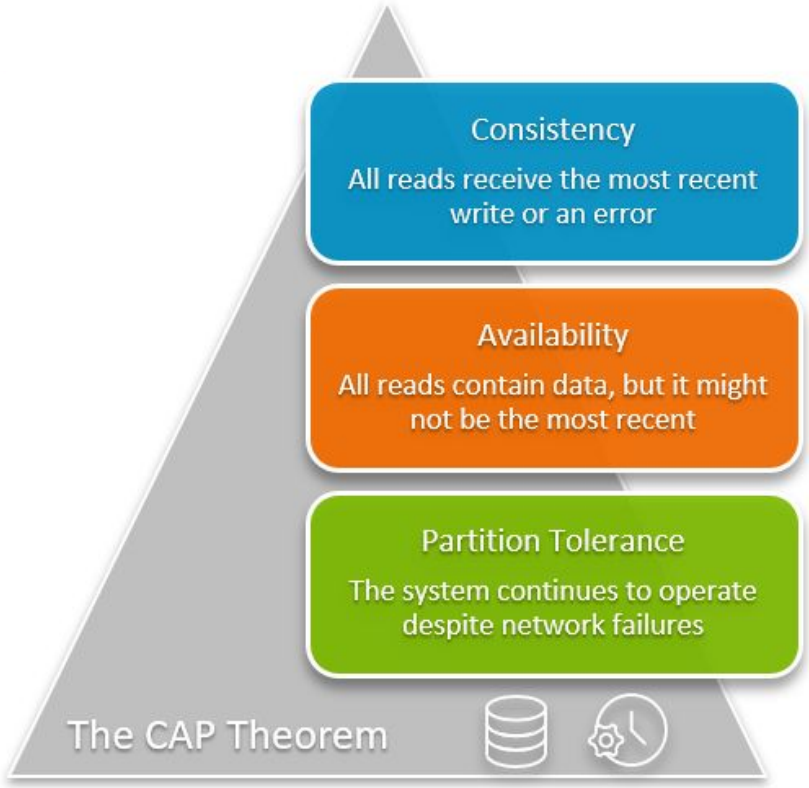
Combining Sharding & Replication

- Multiple masters but each data item has only single master
- A single node can be master for one item and slave for other

Key Points

- There are two styles of distributing data:
- Sharding distributes different data across multiple servers, so each server acts as the single source for a subset of data.
- Replication copies data across multiple servers, so each bit of data can be found in multiple places. A system may use either or both techniques.
- Replication comes in two forms:
- Master-slave replication makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads.
- Peer-to-peer replication allows writes to any node; the nodes coordinate to synchronize their copies of the data. Master-slave replication reduces the chance of update conflicts but peer-to-peer replication avoids loading all writes onto a single point of failure.

CAP Theorem

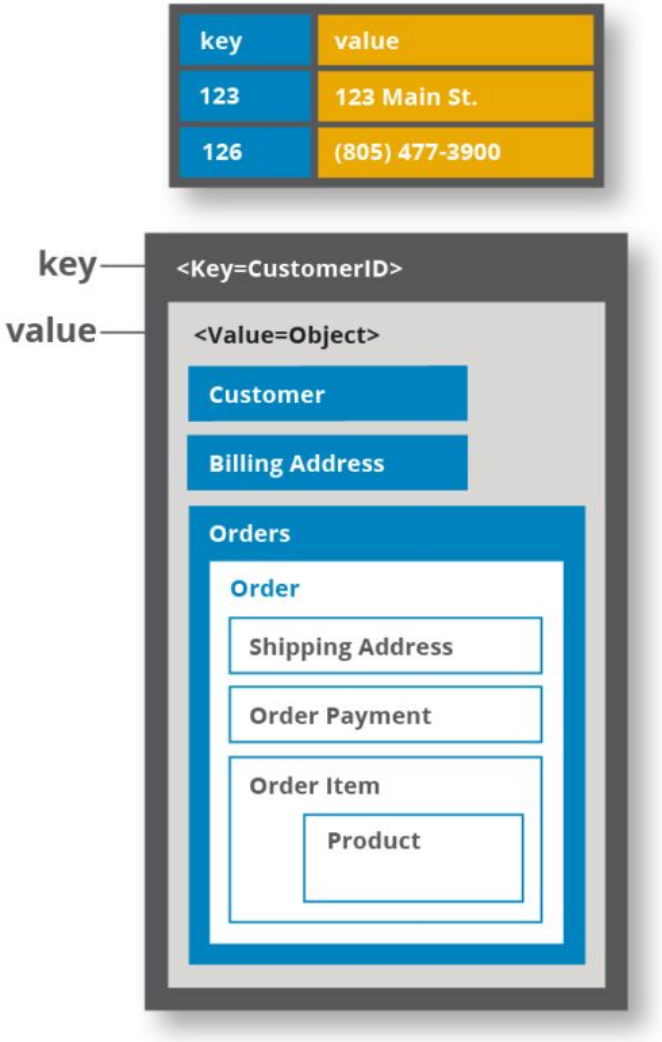

Consistency
All reads receive the most recent write or an error
Availability
All reads contain data, but it might not be the most recent
Partition Tolerance
The system continues to operate despite network failures
The CAP Theorem

Key-Value Store

A key-value store, or key-value database, is a type of data storage software program that stores data as a set of unique identifiers, each of which have an associated value. This data pairing is known as a “key-value pair.” The unique identifier is the “key” for an item of data, and a value is either the data being identified or the location of that data.

get, put, delete

Riak, Redis, Memchaced DB, BerkleyDB, HamsterDB, Amazon DynamoDB, Project Voldemart



Riak

Oracle	Riak
database instance	Riak cluster
table	bucket
row	key-value
rowid	key

Riak

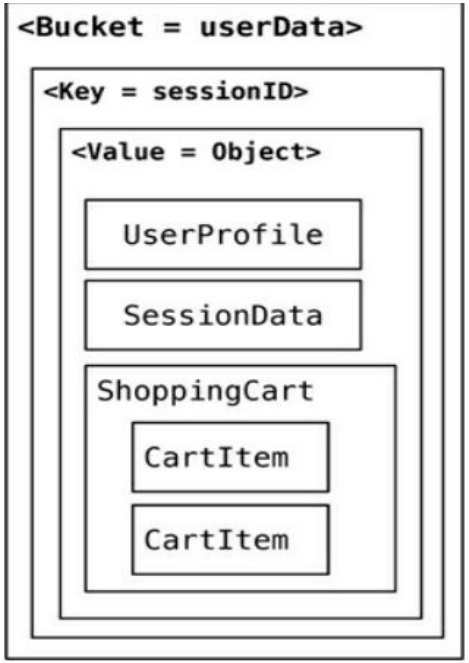


Figure 8.1. Storing all the data in a single bucket

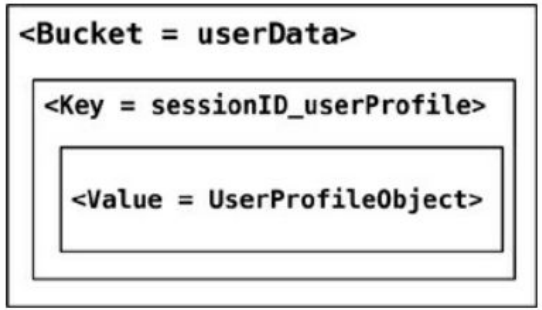


Figure 8.2. Change the key design to segment the data in a single bucket.

Key-Value store features

Consistency

It is applicable for operations which involves single key
Ex: put, get, delete

- Optimal writes are expensive
- In distributed data stores like Riak, eventually consistency model is implemented
- Newest write wins
- Oldest writes loose

Consistency

```
Bucket bucket = connection
    .createBucket(bucketName)
    .withRetrier(attempts(3))
    .allowSiblings(siblingsAllowed)
    .nVal(numberOfReplicasOfTheData)
    .w(numberOfNodesToRespondToWrite)
    .r(numberOfNodesToRespondToRead)
    .execute();
```

If we need data in every node to be consistent, we can increase the `numberOfNodesToRespondToWrite` set by `w` to be the same as `nVal`. Of course doing that will decrease the write performance of the cluster. To improve on write or read conflicts, we can change the `allowSiblings` flag during bucket creation: If it is set to false, we let the last write to win and not create siblings.

Key-Value store features

Transactions

- In NoSQL transactions are implemented differently for different types
- Riak uses quorum
- The write said to be successful if it is executed on at least w nodes
- Suppose replication factor is 10 and w value is 7 which means 3 nodes can be down for write operations

Key-Value store features

Query Features

- All key value databases can be queried by using the key
- It is not possible to read the part of the value directly, we need to get the value and parse it
- What if we do not know the key?
- Riak-Search to search for the values

Suitable use cases(when to use)

- Storing session information
- Userprofiles, preferences
- Shopping cart data

When not to use

- Relationships among the data
- Multi Operation Transactions
- Query by data
- Operation by sets