Key-Value Database in System Design

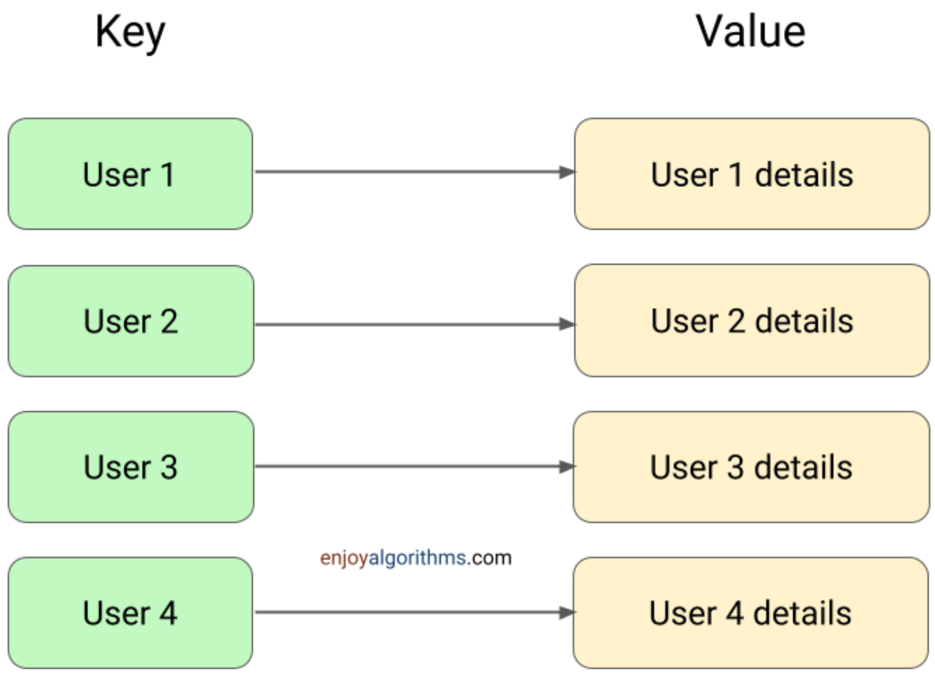
A key-value database is a NoSQL (non-relational) database that stores data in the form of key-value pairs. These key-value pairs are similar to the way data is stored in a map or dictionary, where each key is associated with a single value. Here key serves as a primary key and values can be simple data types like strings or numbers, or complex data structures like JSON.

So there are two important points to keep in mind:

- There is no connection between the values stored in a key-value database.
- Keys are unique to ensure that there is no ambiguity when searching for a specific value.

Key-value databases are generally more flexible and offer faster read and write performance compared to relational databases, which may require more complex aggregations to retrieve data. Instead, a key-value database simply looks up the value associated with a specific key.

There are several different implementations of key-value databases available on the market. Each one of them has some unique features and capabilities. Here are some examples: Aerospike, Apache Cassandra, Amazon DynamoDB, Berkeley DB, Memcached, Riak, Redis, etc.

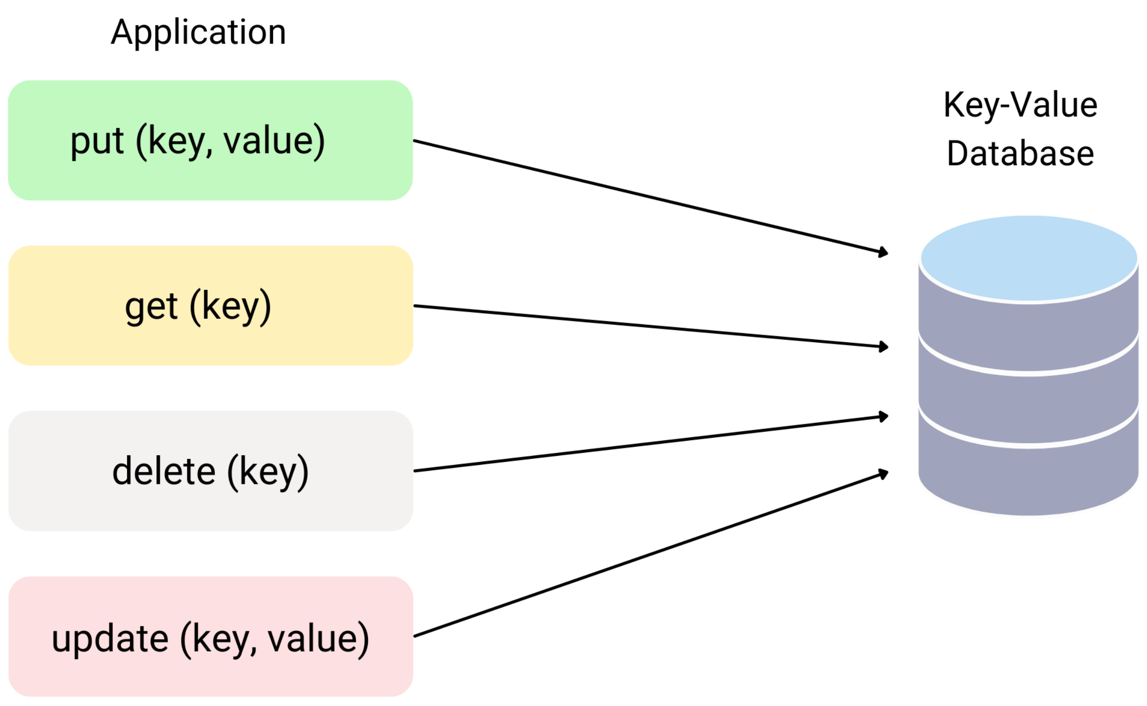


Key-value database operations

A key-value database helps users to store, access, and update data using simple commands like get, put, and delete. This simplicity makes key-value stores fast, easy to use, portable, and flexible. They are also highly partitionable and can scale horizontally, which means they can handle large amounts of data and traffic without slowing down.

Some common operations that can be performed on a key-value databases:

- get(key): Retrieve the value associated with a key.
- put(key, value): Insert a value into the database associated with a key.
- delete(key): Remove the value associated with a key.
- update(key, value): Update the value associated with a key.



Some key-value databases also provide some of these advanced operations:

- Scan: Retrieving a range of keys or all the keys in the database.
- Range queries: Retrieving key-value pairs within a specified range of keys.
- Atomic operations: Certain operations are executed atomically without interference from concurrent operations. For example: Atomic increments or decrements of numeric values.
- Batch operations: Performing multiple put, get, or delete operations together as a single atomic operation. This will help in optimizing performance when working with multiple key-value pairs.
- Secondary indexing: Associating additional keys with specific values. These secondary keys can be used for efficient retrieval of data based on criteria other than the primary key.
- TTL (Time-to-Live): Setting an expiration time for a key-value pair. Once the TTL expires, the key-value pair is automatically deleted. This can be useful for implementing cache invalidation mechanisms.

Characteristics of key-value databases

- **High Availability**: Key-value databases use the idea of replication to maintain multiple copies of data across different nodes. This ensures that users have always access to data even if some nodes become unavailable or fail.

- **Partition Tolerance:** Key-value databases are designed to be partition-tolerant i.e. they can continue to operate even in the presence of network partitions or communication failures.
- **Eventual Consistency:** Most of the key-value databases prioritize availability and partition tolerance over strong consistency. So after a write operation, it may take some time for the data to be replicated and become consistent across all nodes.
- **Scalability**: Key-value databases can handle large amounts of data by distributing read and write operations across multiple servers.
- **Performance:** Key-value databases use in-memory storage and optimized data structures. So they are highly optimized for fast data retrieval based on the key, which can provide low latency and high throughput for read and write operations.
- **Schemaless:** Each key-value pair is independent and the structure of the values can vary across different records. So it does not require a predefined schema or data model.
- **Simplicity:** The simple data model of a key-value database helps us in easy development and integration with various languages and frameworks.
- **Limited query capabilities:** Key-value databases lack complex querying capabilities compared to relational databases. The idea is simple: They are designed for simple key-based lookups for retrieving values. But some key-value databases extend their query capability by providing features like secondary indexes or range queries.

When to use a key-value database?

Key-value databases offer significant speed and scalability benefits compared to traditional databases. So are used in applications that require high-speed data access.

- Session management: We can use it to store session attributes in online applications. For example, it can manage the sessions of individual players in a multiplayer online game.
- In-memory data caching: We can use it as an effective cache mechanism for frequently accessed but rarely updated data. This can help to accelerate application responses.
- User preference storage and personalization: We can use it to provide product recommendations by storing personalized lists of items for individual customers.
- Implementing blockchain-based solutions: It is often used in implementing blockchain-based ideas. In this case, the key is a hash value and the value is the corresponding block.
- Real-time random data access: Key-value databases offer fast in-memory access. This makes them useful in situations where an application needs to handle many small continuous reads and writes.
- Storing basic information: We can easily store some basic data based on key-value pairs in key-value database. For example, we can use it to store URLs as keys and web pages as values.
- Content delivery networks: We can use key-value database to store and serve large web objects (such as videos, images, and audio) in a high availability environment like content delivery network.

Key-value store Vs Cache

As we have seen above, a key-value database can be used as a cache to provide real-time information by quickly returning a value for a specific key. It can store a pool of frequently accessed data to improve performance. But there are some differences between a cache and a key-value store:

- While a key-value database can be used to persist data, caches are used in conjunction with a database to increase read performance.
- Caches are not designed to enhance write or update performance, whereas key-value databases are very effective for these operations.
- While key-value databases are designed to be resilient to server

failure, caches are typically stored in RAM and do not offer transactional guarantees if the server crashes.

Are key-value databases similar to tables in RDBMS?

Key-value databases and tables in RDBMS have some similarities, but they are different in terms of their data models and structures. Here are some of the key differences:

- In an RDBMS, data is organized into structured tables of rows and columns. It requires a predefined schema that specifies the structure of tables like column names, data types, relationships, etc. But key-value databases do not have a fixed schema. Here each key-value pair can have a different structure.
- RDBMS typically use SQL, which allows complex queries like joins, aggregations, and filtering across multiple tables. On the other side, key-value databases often provide a simpler interface with basic operations like get, put, and delete based on the key.
- RDBMS can establish relationships between tables using primary and foreign keys. But key-value databases do not have built-in mechanisms for managing relationships between data. So it is up to the application developer to handle any relationships.
- RDBMS provide ACID properties and ensure data consistency and integrity. On the other side, key-value databases often prioritize availability and partition tolerance over strong consistency.

If you have any queries or feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!