**What is a key-value store?**

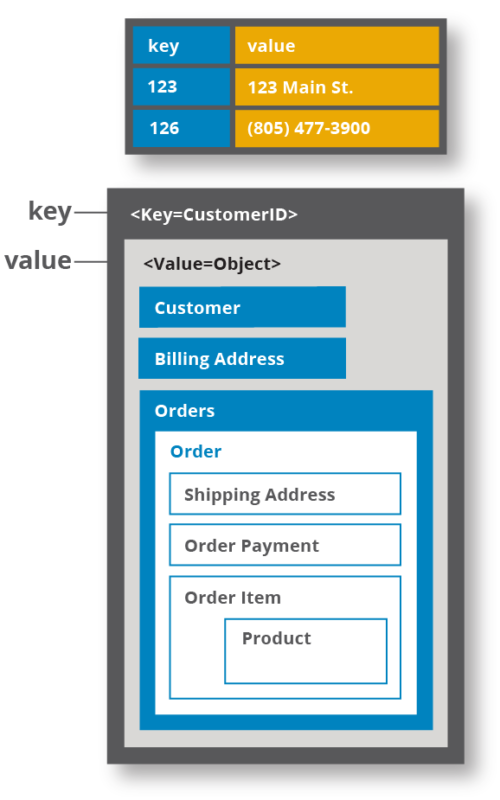
Over the years, database systems have evolved from relational databases storing data in rows and columns to [NoSQL](https://www.mongodb.com/scale/types-of-nosql-database-management-systems) distributed databases allowing other structures and solutions. Key-value pair stores are not a new concept and were already with us for the last few decades.

A key-value database, sometimes called key-value store, 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.

Data is written (inserted, updated, and deleted) and queried based on the key to store/retrieve its value.

In its simplest form, a key-value store is like a dictionary object as it exists in most programming languages, but which is stored in a persistent way and managed by a Database Management System (DBMS).





An example of a key-value store

**Data pairing Key and Value**

A key-value pair is two pieces of data associated with each other. The key is a unique identifier that points to its associated value, and a value is either the data being identified or a pointer to that data.

The key could be anything, depending on restrictions imposed by the database software, but it needs to be unique in the database so there is no ambiguity when searching for the key and its value.

The value could be anything from a number or simple string to a complex object, including a list or another key-value pair. Some database software allows you to specify a data type for the value.

A key-value pair is the fundamental data structure of a key-value store or key-value database, but key-value pairs have existed outside of software for much longer. One simple example is a telephone directory, which is linked with a person or business name.

## **Comparation with traditional relational database and Key-value store advantages**

In traditional relational database design, data is stored in tables composed of rows and columns. The database developer specifies many attributes of the data to be stored in the table in advance. This creates significant opportunities for optimizations such as data compression and performance around aggregations and data access, but also introduces some inflexibility.

On the other hand, Key-value stores provides a few advantages over traditional row-column-based databases. Thanks to the simple data format that gives it its name, a key-value store can be very fast for read and write operations, in part because the database is looking for a single key and is returning its associated value rather than performing complex operations.

And key-value stores are very flexible, a really important characteristic, as we generate more data without traditional structures.

Also, key-value stores do not require placeholders such as “null” for optional values, so they may have smaller storage requirements.

# F**eatures of a key-value database**

key-value databases are defined so simply, but can be extended and optimized in numerous ways. The common features:

* Retrieving a value (if there is one) stored and associated with a given key
* Deleting the value (if there is one) stored and associated with a given key
* Setting, updating, and replacing the value (if there is one) associated with a given key

These are the minimum features for a key-value store.

## **Advantages of key-value databases**

They are easy to design and implement.

A key-value approach allows defining efficient and compact data structure to access data in a simple form of a key-value fetch/update/remove.

The compact structure and fast indexing makes this database concept a win for specific application workloads.

Key-value databases are highly divisible and scalable, they allow a level of scalability that other types of databases cannot reach.

**Cons**

However, modern applications will probably require more than just a key-value retrieval.

## **When to use a key-value database / Key-value database use cases**

The advantages listed give themselves to several popular use cases for key-value databases.

Key-value stores are used for use cases where applications will require values to be retrieved fast via keys.

There are several use-cases where choosing a key value store approach is an optimal solution:

* Real time random data access
  + e.g., user session management in an online application on a large scale.
  + Managing each player’s session in massive multiplayer online games.
  + Storing personal data on specific users.
  + Web applications may store user session details and preference in a key-value store. All the information is accessible via user key, and key-value stores lend themselves to fast reads and writes.
  + Real-time recommendations and advertising are often powered by key-value stores because the stores can quickly access and present new recommendations or ads as a web visitor moves throughout a site.
  + Shopping Cart: Key-value databases can handle the scaling of large amounts of data and extremely high volumes of state changes, while serving millions of users simultaneously through distributed processing and storage.
* Caching mechanism for frequently accessed data or configuration based on keys.

On the technical side, key-value stores are commonly used for in-memory data caching to speed up applications by minimizing reads and writes to slower disk-based systems.

* Applications that needs to handle lots of small continuous reads and writes. Key-value databases offer fast in-memory access.

## **Key-value database vs cache**

Databases supporting key-value stores persist the data to a disk serving the database files, while a key-value cache implementation will mostly keep the data loaded [in memory](https://www.mongodb.com/databases/in-memory-database). In case of a server fault or restart, the data needs to be preloaded into the cache as it was not persistent.

**Most important Key-value Databases**

The two most important Key-value Databases in the actuality are

* Redis
* Amazon DynamoDB

Redis (which means Remote Dictionary Serve) is an open-source NoSQL database known to be a fast in-memory key-value data store.

**Redis CRUD Op (& in our project)**

(CREATE, READ, UPDATE, DELETE )

How to build a Shopping cart app using NodeJS and Redis: <https://developer.redis.com/howtos/shoppingcart/>

* How the data is stored & How the data is modified: <https://redis.io/commands/set>
* How the data is accessed:
  + <https://redis.io/commands/get>
  + <https://redis.io/commands/hgetall>
* How the data is deleted: <https://redis.io/commands/del>

Rollback: <https://redis.io/topics/transactions>

**Why Redis does not support roll backs?**

If you have a relational databases background, the fact that Redis commands can fail during a transaction, but still Redis will execute the rest of the transaction instead of rolling back, may look odd to you.

However there are good opinions for this behavior:

* Redis commands can fail only if called with a wrong syntax (and the problem is not detectable during the command queueing), or against keys holding the wrong data type: this means that in practical terms a failing command is the result of a programming errors, and a kind of error that is very likely to be detected during development, and not in production.
* Redis is internally simplified and faster because it does not need the ability to roll back.

An argument against Redis point of view is that bugs happen, however it should be noted that in general the roll back does not save you from programming errors. For instance if a query increments a key by 2 instead of 1, or increments the wrong key, there is no way for a rollback mechanism to help. Given that no one can save the programmer from his or her errors, and that the kind of errors required for a Redis command to fail are unlikely to enter in production, we selected the simpler and faster approach of not supporting roll backs on errors.

All commands list: <https://redis.io/commands>

**HSET**: Set the string value of a hash field.

The cart data is stored in a hash

**HDEL**: Delete one or more hash fields.

The product can be removed from the cart

The cart can be cleared using

**DEL**: All carts can be deleted when reset data is requested

**HGETALL**: Get all the fields and values in a hash.

**Estrutura/Puntos Importantes**

* Explain Key value Model
* Key points and real uses
* Exply types supported (5)
* Redis
* Comparison to SQL
* Demo
* When it is useful?

Difference between Redis and Memcached is that Redis supports some data structures as values.

Links:

What is a key value store: <https://redis.com/nosql/key-value-databases/>

Data types: <https://redis.io/topics/data-types>

Data types intro: <https://redis.io/topics/data-types-intro>

Integrated modules: <https://redis.com/redis-enterprise/technology/integrated-modules/#rejson>

Rollbacks Redis: <https://redis.io/topics/transactions>

redis persistence: <https://redis.io/topics/persistence>