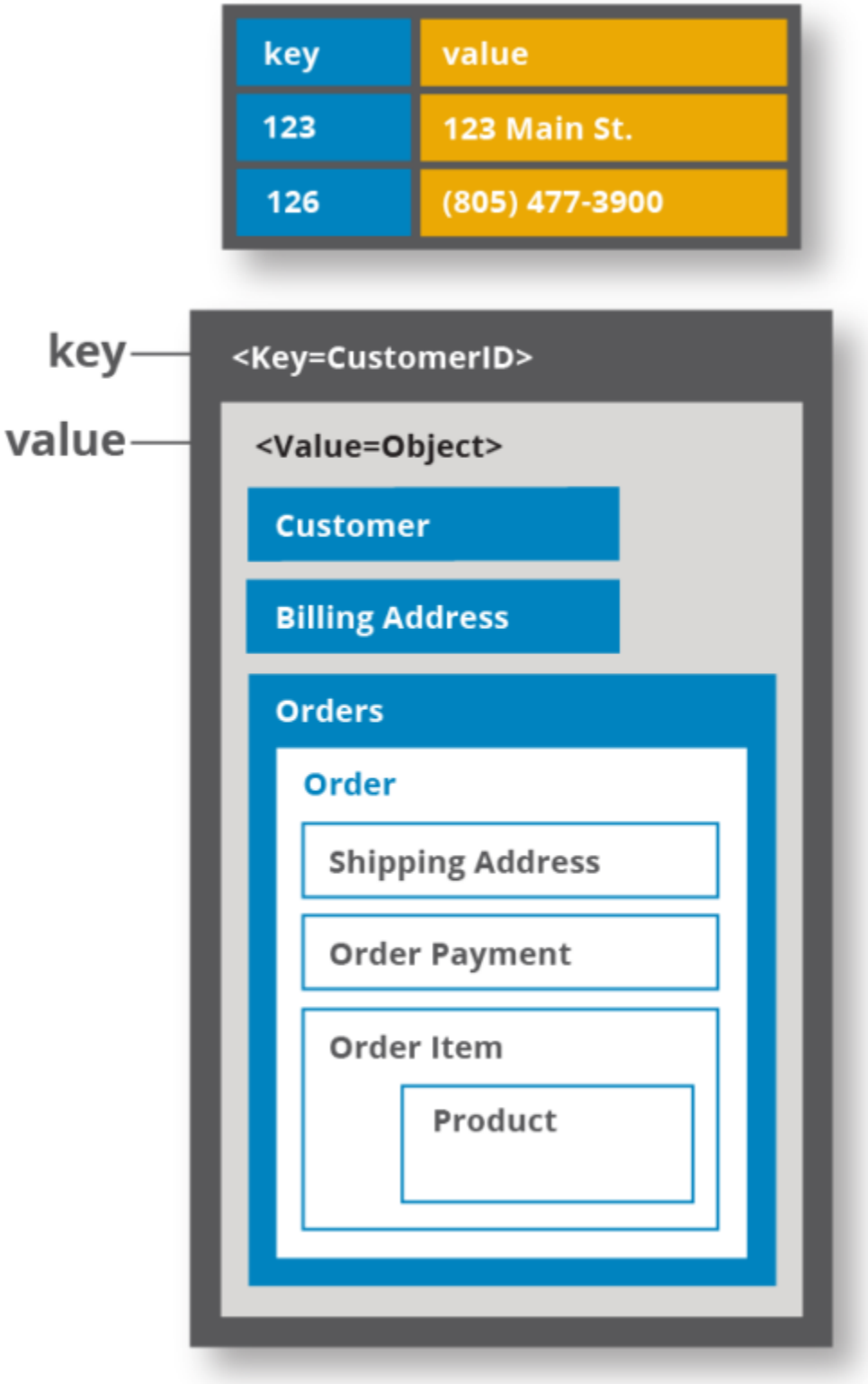
What is a key-value store?

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.



An example of a key-value store

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, including a list or another key-value pair. Some database software allows you to specify a data type for the value.

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 upfront. This creates significant opportunities for optimizations such as data compression and performance around aggregations and data access, but also introduces some inflexibility.

Key-value stores, on the other hand, are typically much more flexible and offer very fast performance for reads and writes, in part because the database is looking for a single key and is returning its associated value rather than performing complex aggregations.

Understanding K-V concepts

In 1966, a DEC PDP-7 computer was delivered to the Massachusetts General Hospital without any programming. The hospital had invested in a computer but had no way to run it. At this time, there was very little in the way of software options, operating systems, or database support. The hospital did, however, have access to the Massachusetts Institute of Technology (MIT), which was located just across the river. The MIT team started from scratch, and went on to design "MUMPS" (Massachusetts Utility Multi-Programming System) as a multi-user operational system, a database, and a language, all in one. MUMPS used a key-value store and several other features that were later incorporated into NoSQL data stores.

The use and focus of key-value structures gradually evolved into simple NoSQL key-value databases. This type of database saves data as a group of key-value pairs, which are made up of two data items that are linked. The link between the items is a "key" (such as "file name"), which acts as an identifier for an item within the data and the "value" that is the data (or content) that has been identified.

Key-value databases are often considered the simplest of the NoSQL databases. This model's simplicity makes key-value stores and databases quick, user-friendly, portable, scalable, and flexible. However, the original key-value systems were not designed to allow researchers to filter or control the data that returns from a request – they did not include a search engine. That is changing as people modify their key-value databases.

Redis introduced their key-value database in 2009. Kyle Davis, the Technical Marketing Manager at Redis Labs, recently wrote:

"The original intention of Redis (or any key-value store) was to have a particular key, or identifier, for each individual piece of data. Redis quickly stretched this concept with data types, where a single key could refer to multiple (even millions of) pieces of data. As modules came to the ecosystem, the idea of a key was stretched even further because a single piece of data could now span multiple keys (for a RediSearch index, as example). So, when asked if Redis is a key-value store, I usually respond with 'it descends from the key-value line of databases' but note that, at this point, it's hard to justify Redis as a key-value store alone."

Though many NoSQL databases continue to include key-value stores, keys can also be used in relational databases. The primary key used for relational tables uniquely identifies each record within the table. Some keys can be used to connect (or join) data that is stored in one table to the data in

other tables. Storing a primary key to a row in another table is known as a foreign key. There are primary key and foreign key errors that should be avoided.

How Key-Value Databases Work

Key-value databases do not establish a specific schema. Traditional relational databases pre-define their structures within the database, using tables that contain fields with well-defined data types. Key-value systems, on the other hand, treat data as a single collection with the key representing an arbitrary string – for example a filename, hash, or uniform resource identifier (URI). Key-value stores generally use much less memory while saving and storing the same amount of data, in turn increasing the performance for certain types of workloads.

"Pure" key-value databases do not use a query language, but they do offer a way to retrieve, save, and delete data by using the very simple commands *get, put,* and *delete. (Modified key-value databases may include full-text searches.)* Retrieving data requires a direct request method for communicating with the data file. There is no searching, nor is there a search engine. If the key is not known, there is no way to find it.

The Uses of Key-Value Databases

While relational databases handle payment transactions quite well, they struggle in dealing with high volumes of simultaneous transactions. However, NoSQL key-value databases can scale as needed and handle extremely high volumes of traffic per second, providing service for thousands of simultaneous users.

Key-value NoSQL databases come with built in redundancy, allowing them to handle lost storage nodes without problems. (Occasionally, for example, a "shopping cart" will lose items.) Key-value stores process large amounts of data and a consistent flow of read/write operations for:

- Session management: Offering users the option to save and restore sessions.
- User preferences and profile stores: Personal data on specific users.
- Product recommendations: Customized items a customer might be interested in.
- Coupons, customized ads: Adapted and viewed by customers in real-time.
- Acting as a cache for regularly viewed data which rarely gets updated.

Key-value databases are often used for session management in web applications. They do well at managing the session's information for all the new user apps on smart phones and other devices.

Key-value databases can also be used for massive multi-player online games, managing each player's session.

They are very good at managing shopping carts for online buyers – until payment time. Payment transactions and any revenue postings work better with a relational database.

As one of the simplest NoSQL databases, key-value databases can be scaled easily for purposes of big data research, while servicing multiple users, simultaneously.

Businesses selling products over the internet often struggle with the different volumes of the pre-Christmas buying season versus the rest of the year. The issue is about paying for an infrastructure scaled for the Christmas's buying peak (and paying for that infrastructure for the rest of the year) or taking the risk of not being able to handle the Christmas rush (and crashing for several hours).

Assuming a relational database handles normal year-round services, renting a Cloud service with a key-value database for the Christmas rush provides an efficient, relatively inexpensive solution.

Different Key-Value Databases

Different key-value databases use different techniques for improving on the basic key-value model. Some store all their data in RAM, while others work with a combination of SSDs (solid state drives) and RAM. Still others combine support for rotating disks and RAM.

These databases were designed to respond to the new applications that have become available for smart phones, and other devices. Organizations should avoid having all their relational databases replaced with NoSQL, especially for financial applications. Some popular key-value databases are listed below:

- Aerospike: An open-source, NoSQL database using a flash-optimized in-memory.
- Apache Cassandra: A distributed, free, open-sourced, wide column store, NoSQL database management system.
- Amazon Dynamo DB: A fully managed proprietary NoSQL database service that is offered by Amazon.
- Berkeley DB: A basic, high performance, embedded, open-source, database storage library.
- Couchbase: Designed for business critical applications and heavily modified, it provides full text searches, SQL-based querying, and analytics.
- Memcached: Speeds up websites by caching data and objects in RAM to reduce the number of times an external data source must be read. Free and open-sourced.
- Riak: Fast, flexible, and scalable, it is good for developing applications and working with other databases and applications.
- Redis: A database, message broker, and memory cache. It supports hashes, strings, lists, bitmaps, and HyperLogLog.

Generally speaking, the secret to key-value databases lies in their simplicity and the resulting speed that becomes available. Retrieving data requires a direct request (key) for the object in memory (value), and there is no query language. The data can be stored on distributed systems with no worries about where indexes are located, the volume of data, or network slowdowns. Some key-value databases are using flash storage and secondary indexes in an effort to push the limits of key-value technology.

A key-value database is both easy to build and to scale. It typically offers excellent performance and can be optimized to fit an organization’s needs. When a key-value database is modified with new applications, there is an increased chance the system will operate more slowly

What does a key-value pair mean?

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.

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. A telephone directory is a good example, where the key is the person or business name, and the value is the phone number. Stock trading data is another example of a key-value pair. In this case, you may have a key associated with values for the stock ticker, whether the trade was a buy or sell, the number of shares, or the price of the trade.

There are a few advantages that a key-value store provides 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. And key-value stores are very flexible, a valued asset in modern programming 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, and they often scale almost linearly with the number of nodes.

Key-value database use cases

The advantages listed above naturally lend themselves to several popular use cases for key-value databases.

- 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.
- 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. Hazelcast is an example of a technology that provides an in-memory key-value store for fast data retrieval.

Distributed key-value store

A distributed key-value store builds on the advantages and use cases described above by providing them at scale. A distributed key-value store is built to run on multiple computers working together, and thus allows you to work with larger data sets because more servers with more memory now hold the data. By distributing the store across multiple servers, you can increase processing performance. And if you leverage replication in your distributed key-value store, you increase its fault tolerance. Hazelcast is an example of a technology that provides a distributed key-value store for larger-scale deployments. The “IMap” data type in Hazelcast, similar to the “Map” type in Java, is a key-value store stored in memory. Unlike the Java Map type, Hazelcast IMaps are stored in memory in a distributed manner across the collective RAM in a cluster of computers, allowing you to store much more data than possible on a single computer. This gives you quick lookups with in-memory speeds while also retaining other important capabilities such as high availability and security.

Refs
https://hazelcast.com/glossary/key-value-store/
https://www.dataversity.net/understanding-key-value-databases/#