What Is a Key-Value Database?

aws.amazon.com/nosql/key-value

A key-value database is a type of non-relational database, also known as NoSQL database, that uses a simple key-value method to store data. It stores data as a collection of key-value pairs in which a key serves as a unique identifier. Both keys and values can be anything, ranging from simple objects to complex compound objects. Key-value databases (or key-value stores) are highly partitionable and allow horizontal scaling at a level that other types of databases cannot achieve.

What are the advantages of key-value databases

Traditional relational databases (SQL databases) store data in the form of tables containing rows and columns. They enforce a rigid structure on data and are not optimal for every use case. On the other hand, key-value databases are NoSQL databases. They allow flexible database schemas and improved performance at scale for certain use cases. The advantages of key-value stores include:

Scalability

As every user query requires data interaction, databases can often become a bottleneck in application performance. Several strategies to solve the issue, such as replication and sharding, add complexity to the application code. Many key-value databases provide built-in support for advanced scaling features. They scale horizontally and automatically distribute data across servers to reduce bottlenecks at a single server.

Ease of use

Key-value databases follow the object-oriented paradigm that allows developers to map real-world objects directly to software objects. Several programming languages, such as Java, also follow the same paradigm. Instead of mapping their code objects to multiple underlying tables, engineers can create key-value pairs that match their code objects. This makes key-value stores more intuitive for developers to use.

Performance

Key-value databases process constant read-write operations with low-overhead server calls. Improved latency and reduced response time give better performance at scale. They are based on simple, single-table structures rather than multiple interrelated tables. Unlike relational databases, key-value databases don't have to perform resource-intensive table joins, which makes them much faster.

What are the use cases of key-value databases

You can use key-value database systems as the primary database for your application or to handle niche requirements. We give some example key-value database use cases below.

Session management

A session-oriented application, such as a web application, starts a session when a user logs in to an application and is active until the user logs out or the session times out. During this period, the application stores all user session attributes either in the main memory or in a database. User session data may include profile information, messages, personalized data and themes, recommendations, targeted promotions, and discounts.

Each user session has a unique identifier. Session data is never queried by anything other than a primary key, so a fast key-value store is a better fit for session data. In general, key-value databases may provide smaller per-page overhead than relational databases.

Shopping cart

An e-commerce website may receive billions of orders per second during the holiday shopping season. A key-value database can handle the scaling of large amounts of data and extremely high volumes of state changes, while also servicing millions of simultaneous users through distributed processing and storage. Key-value stores also have built-in redundancy, which can handle the loss of storage nodes.

Metadata storage engine

Your key-value store can act as an underlying storage layer for higher levels of data access. For example, you can scale throughput and concurrency for media and entertainment workloads such as real-time video streaming and interactive content. You can also build out your game platform with player data, session history, and leaderboards for millions of concurrent users.

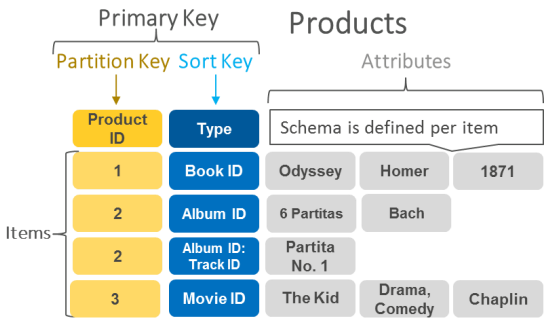
Caching

You can use a key-value database for storing data temporarily for faster retrieval. For example, social media applications can store frequently accessed data like news feed content. In-memory data caching systems also use key-value stores to accelerate application responses.

How do key-value databases work

Key-value databases work by organizing all data as a set of key-value pairs. You can think of the key as a question and the value as the answer to the question. In the example below, the primary key is a composite of two keys, Product ID and Type. The Product ID is the partition key which describes the partition in which the item will be stored. The Type is the sort key, which determines the order in which items will be stored in disk. The combination of the Partition Key and the Sort Key forms a unique primary key, which maps to a single value in the database.

In this example, the data object book has attributes like title, author, and publishing date. Every book data object has a key called BookID. You can directly link the BookID and associated book object in the key-value store. In addition, you can retrieve data by looking up the BookID in the table. Also, each item has its own schema, making key-value stores highly flexible for storing data of varying structures.



What are the features of key-value databases

Depending on the solution you choose, your key-value store can provide several additional features as listed below.

Support for complex data types

Key-value stores provide support for defined data types like integers and text. However, many of them can also support more complex objects like arrays, nested dictionaries, images, videos, and semi-structured data. By giving the database more information about your data, there is room for more storage and query performance optimization.

No need for table joins

Key-value databases don't need to perform any resource-intensive table joins. Their flexibility accommodates all the needed information in a single table. This is one of the reasons key-value stores perform so well.

Sorted keys

A key-value store can sort keys so that data is stored systematically and for implementing partitioning. For example, keys may be sorted:

- Alphabetically or numerically
- Chronologically
- By data size

Consider a key-value store that uses the customer's email address as the unique key. Email addresses can be sorted alphabetically, so all data for A-J email lists are stored on server 1, K-S on server 2, and so on.

Secondary key support

Some key-value stores allow you to define two or more different keys or secondary indexes to access the same data. For example, you can store customer data by key email address and key phone number.

Replication

Many key-value stores offer built-in replication support by automatically copying data across multiple storage nodes. This helps with auto-recovery from disasters; you still have your data in case of server failure.

Partitioning

Partitioning is how you distribute data across nodes. Many key-value databases provide default partitioning options. Some also give you the option to define input parameters for your partitions. For example, you could partition numerical keys into groups of 1000.

Advanced key-value databases also provide automatic support for distributing your key-value database across multiple geographical locations. This improves application availability and reliability because you can respond to queries close to the user's location.

ACID support

Atomicity, Consistency, Isolation, and Durability (ACID) are database properties that ensure data accuracy and reliability in all circumstances. For instance, if you are making multiple changes to your data in a sequence, atomicity requires that all changes go through in order. If one change fails, everything fails.

Advanced key-value databases provide native, server-side support for ACID. This simplifies the developer experience of making coordinated, all-or-nothing changes to multiple items both within and across tables. With transaction support, developers can extend the scale, performance, and enterprise benefits to a broader set of mission-critical workloads.

What are the limitations of key-value databases

Key-value databases do require some trade-offs, as with any kind of technology choice.

Absence of complex queries

As key-value databases don't support complex queries, developers must work around this in the code. Data operations are mainly through simple query language terms like get, put, and delete. There are limitations to how much you can filter and sort data before accessing it.

Schema mismanagement

Key-value store design does not enforce a schema on developers. Anyone can modify the schema in the database program. Development teams have to plan the data model systematically to avoid long-term problems. The lack of a tight schema also means that the application is responsible for the proper interpretation of the data it consumes, often referred to as 'schema on read'.

How can AWS support your key-value database requirements

Amazon DynamoDB is one of the most popular key-value databases designed to run high-performance applications at any scale. It's a fully managed, multi-region, multi-active database that provides features like:

- Limitless scalability, including scale-to-zero, with consistent single-digit millisecond latency.
- Serverless with no version upgrades, no maintenance windows, and no servers or software to manage.

- Designed for 99.999% availability, with DynamoDB Global Tables providing active-active replication so you can build globally distributed applications with local read performance.
- Highly secure and reliable with default encryption at rest, point-in time recovery, on-demand backup and restore, and more.
- Easy to use with integrations with many AWS services including bulk import/export from Amazon S3, Amazon Kinesis Data Streams, Amazon Cloudwatch, and more.

With this step-by-step tutorial on creating and querying a NoSQL table, you can be up and running with DynamoDB in 10 minutes. Get started with key-value databases on AWS by creating a free account today!

Introduction to Amazon DynamoDB

Amazon DynamoDB data modeling