Module 2: Key Value Database

Text chapters 3, 4, & 5

University of Illinois
at Springfield

What is Key-Value Database

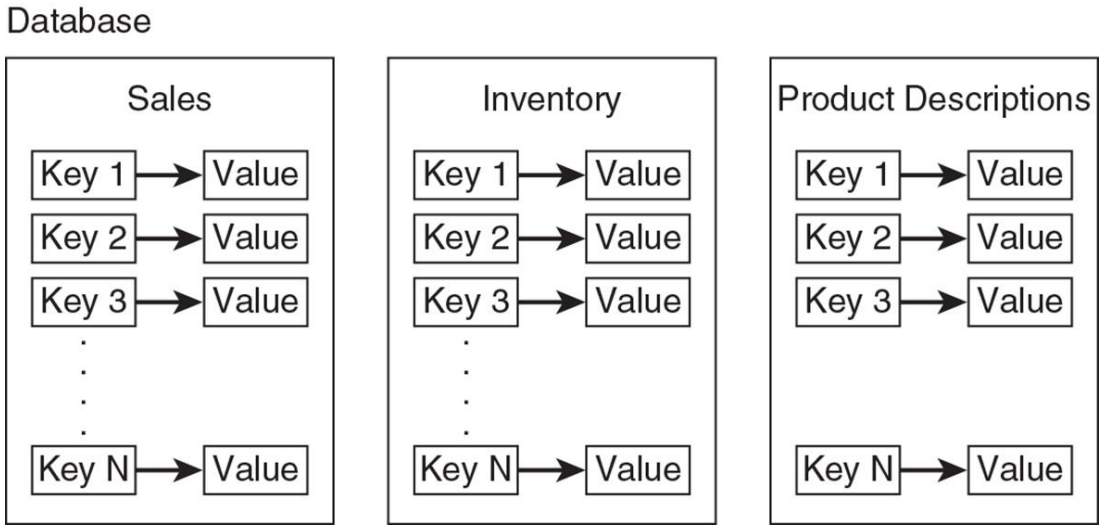
- Key-value databases are the simplest of the NoSQL databases.
- They can only store pairs of keys and values, as well as retrieve values when a key is known.
- While not adequate for complex applications, this simplicity makes K-V DB systems attractive in certain circumstances. For example resource-efficient key-value stores are often applied in embedded systems or as high performance in-process databases.

What is Key-Value Database

- **A key-value data store is a more complex variation on the array data structure.**

1	True
2	True
3	False
4	True
5	False
6	False
7	False
8	True
9	False
10	True

Figure 3.1 An array is an ordered list of elements. All elements are of the same type. The value of each element of the array is read and set by reference to its index.



- **We extend the concept of an array by relaxing constraints on the simple data structure and adding persistent data storage features to create a range of other useful data structures, including associative arrays, caches, and persistent key-value databases.**

- **Key-value databases build on the concept of an associative array, but there are important differences.**
  - **Many key-value data stores keep persistent copies of data on long-term storage, such as hard drives or flash devices.**
  - **Some key-value data stores keep only data in memory.**
    - These are typically used so programs can access data faster than if they had to retrieve data from disk drives.

Associative Array

- An associative array is a data structure, like an array, but is not restricted to using integers as indexes or limiting values to the same type.
- Associative arrays generalize the idea of an ordered list indexed by an identifier to include arbitrary values for identifiers and values.
  - In addition, note that values stored in the associative array can vary.

'Pi'	3.14
'CapitalFrance'	'Paris'
17234	34468
'Foo'	'Bar'
'Start_Value'	1

Figure 3.2 An associative array shares some characteristics of arrays but has fewer constraints on keys and values.

Caches

- Caches are helpful for improving the performance of applications that perform many database queries.
- Key-value data stores are even more useful when they store data persistently on disk, flash devices, or other long-term storage.
  - They offer the fast performance benefits of caches plus the persistent storage of databases.

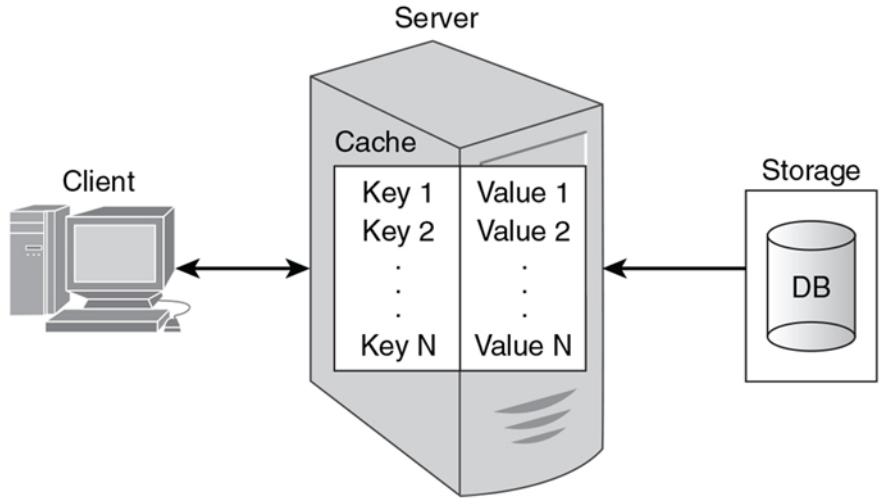


Figure 3.3 Caches are associative arrays used by application programs to improve data access performance.

Why use Key-Value Database

- **Key-value databases impose a minimal set of constraints on how you arrange your data.**
  - **There is no need for tables if you do not want to think in terms of groups of related attributes.**

- **Key-value databases are designed for speed.**
  - **They read and write data inefficiently. Key-value databases do not support complex query languages, such as SQL. To retrieve a value, one must know the key.**

Key design is important

- It is important to design keys in ways that are easily managed.
- In the most simple case, a key could be based on a sequence of values.
  - This is sometimes used in relational database design. Such keys are not necessarily related to the purpose of information stored.

Key design example

- A set of keys could be 198721, 198722, 198723, and so on.
  - As long as a sequence number is not used to store more than one value, this scheme will work.
- A prefix can be added to a key to indicate the type of values stored.
  - For example, cust198721, cust198722, and cust198723 could be used to indicate that the keys refer to values about customers.
  - Similarly, the keys ord198721, ord198722, and ord198723 could refer to values stored about orders.

Key design example

- **A more complex naming pattern could be used to emulate relational table like organizations.**
- **Designers could use a key-naming convention that uses a table name, primary key value, and an attribute name to create a key to store the value of an attribute:**

  customer:1982737:firstName

  customer:1982737:lastName

  customer:1982737:shippingAddress

  customer:1982737:shippingCity

  customer:1982737:shippingState

  customer:1982737:shippingZip

Key design example

- **Values can be atomic structures, such as integers and floating-point numbers or they can be more complex structures, ranging from strings to documents.**
- **Key-value databases typically treat values as atomic units, so it is not possible, for example, to query for all values that have a particular string in a value.**
  - **The exceptions to this statement are key-value databases that provide text-search capabilities. Text search is a useful feature and should be used when available,**
    - but this feature is not a standard part of key-value databases at this time.

Data Model Vs Data Structure

- It is important to distinguish Data Models from Data Structures. Data Structures and Data Models are not synonymous.
- Data Structures provide a high-level abstraction of data storage
- Data models provide a layer of abstraction above data structures that enables database application developers to focus more on the information that must be managed and less on implementation issues.

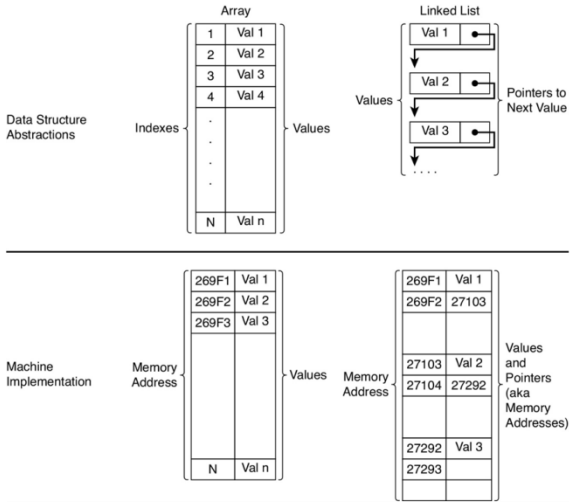


Figure 4.1 Data structures provide higher-level organizations than available at the machine level.

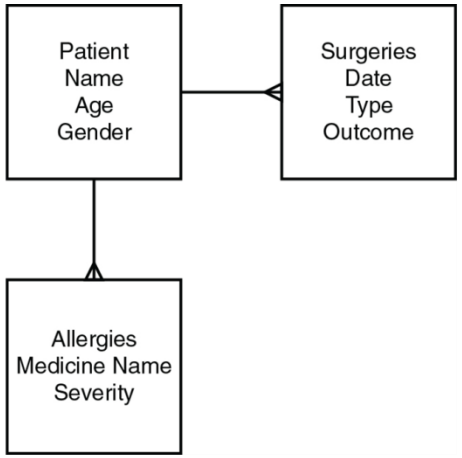


Figure 4.2 Data models provide a layer of abstraction above data structures that allows database application developers to focus more on the information that must be managed and less on implementation issues.

Key-Value Data Model Terms: KEY

- **Key: A *key* is a reference to a value.**
  - **It is analogous to an address.**
  - **A key can take on different forms depending on the key-value database used.**
  - **At a minimum, a key is specified as a string of characters.**
  - **Strings used as keys should not be too long.**
    - Long keys will use more memory, and key-value databases tend to be memory-intensive systems already.
    - At the same time, avoid keys that are too short. Short keys are more likely to lead to conflicts in key names.

Key-Value Data Model Terms: VALUE

- **Value: A *value* is an object, typically a set of bytes, that has been associated with a key.**
- **Values can be**
  - **integers,**
  - **floating-point numbers,**
  - **strings of characters,**
  - **binary large objects (BLOBs),**
  - **semi-structured constructs such as JSON objects,**
  - **images, audio, and just about any other data type you can represent as a series of bytes.**
- **Most key-value databases will have a limit on the size of a value.**

Key-Value Data Model Terms: NAMESPACE

- **Namespace: A *namespace* is a collection of key-value pairs.**
- **A namespace is a set; duplicate keys are not allowed.**
  - **It is possible to have the same value assigned to multiple keys, but keys can be used only once in a namespace.**
- **A namespace could be an entire key-value database. The essential characteristic of a namespace is that it is a collection of key-value pairs that has no duplicate keys.**

Key-Value Data Model Terms: PARTITION

- **Partition: A partition is a scheme for dividing large databases into subsets that are easily managed on multiple servers.**
  - **In general database terms, a group of servers working together to implement a database is known as a partitioned cluster.**

- **Partition key: A *partition key* is a key used to determine which partition should hold a data value.**
  - **In key-value databases, all keys are used to determine where the associated value should be stored.**

Key-Value Data Model Terms: SCHEMALESS

- **Schemaless: A term that refers to the fact that key-value databases do not require a separate specification of a database model.**
  - **Relational databases require a specification of tables, columns, and constraints. This is not the case for key-value databases.**
- **The schema of a key-value database is implicit in the way developers use the key-value database.**
  - **There is no distinct definition of what keys and values are allowed.**
  - **This provides for substantial flexibility.**
    - Some NoSQL vendors use the term *flexible schema* to distinguish NoSQL implicit schemas from relational, explicit schemas.

Key-Value Data Model Architecture Terms

- **Clusters: *Clusters* are sets to connected computers that coordinate their operations.**
- **Clusters may be loosely or tightly coupled.**
  - **Loosely coupled clusters consist of fairly independent servers that complete many functions on their own with minimal coordination with other servers in the cluster.**
  - **Tightly coupled clusters tend to have high levels of communication between servers. This is needed to support more coordinated operations, or calculations, on the cluster.**
- **Key-value clusters tend to be loosely coupled.**

Key-Value Data Model Architecture Terms

- **Ring: A *ring* is a logical structure for organizing partitions.**
  - **It is a circular pattern in which each server or instance of key-value database software running on a server is linked to two adjacent servers or instances.**
  - **Each server or instance is responsible for managing a range of data based on a partition key.**

Key-Value Data Model Architecture Terms

- **Replication: *Replication* is the process of saving multiple copies of data in your cluster.**
  - **The more replicas you have, the less likely you will lose data**
    - however, you might have lower performance with a large number of replicas

Key-Value Data Model Implementation Terms

- **Hash functions: *Hash functions* are algorithms that map from an input**
  - **Hash functions are generally designed to distribute inputs evenly over the set of all possible outputs.**
    - The output space can be quite large. For example, the SHA-1 has 2^{160} possible output values.
  - **This is especially useful when hashing keys. The ranges of output values can be assigned to partitions, and you can be reasonably assured that each partition will receive approximately the same amount of data.**

Key-Value Data Model Implementation Terms

- **Collision: A *collision* occurs when two distinct inputs to a hash function produce the same output.**
  - **When it is difficult to find two inputs that map to the same hash function output, the hash function is known as collision resistant.**
  - **If a hash table is not collision resistant, or if you encounter one of those rare cases in which two inputs map to the same output, you will need a collision resolution strategy, such as maintaining a linked list of values.**

Key-Value Data Model Implementation Terms

- **Compression: Because key-value databases can be memory intensive, compression is used to optimize memory and persistent storage.**
- **A compression algorithm for key-value stores should perform compression and decompression operations as fast as possible.**
  - **This often entails a trade-off between the speed of compression/decompression and the size of the compressed data.**

Key-Value Database Design

- Key-value databases are relatively simple when compared to other types of databases.
- Designing for them is, however, challenging. The following slides describe important topics in key-value database design.

Key-Value Database Design

- Key design has an impact on database performance.
- Keys should have some logical structure to make code readable and extensible, but they should also be designed with storage efficiency in mind.
- Developers and designers should use meaningful and unambiguous naming components, such as ‘cust’ for customer or ‘inv’ for inventory.

Key-Value Database Design

- **Use range-based components when you would like to retrieve ranges of values.**
  - **Ranges include dates or integer counters.**
- **Use a common delimiter when appending components to make a key.**
  - **The colon (:) is a commonly used delimiter, but any character that will not otherwise appear in the key will work.**
- **Keep keys as short as possible without sacrificing the other characteristics mentioned in this list.**

Key-Value Database Design

- **Designers and developers also need to consider how to structure values.**
- **Data that is frequently used together should be stored together.**
  - **This reduces latency, or the time required to fetch or update data from the database.**
  - **One way to store commonly used attribute values together is with structured documents, such as a JSON structure.**
  - **In the case of a customer management database, you could store a list with both the customer's name and address together.**
    - This is a more complex value structure than using several different keys but has significant advantages in some cases. By storing a customer name and address together, you might reduce the number of disk seeks that must be performed to read all the needed data.

Key-Value Database Design

- Using structured data types, such as lists and sets, can improve the overall efficiency of some applications by minimizing the time required to retrieve data.
  - It is important to also consider how increasing the size of a value can adversely impact read and write operations.
- As with other areas of software engineering and data modeling, design patterns can help improve the quality of applications and the clarity of code.

Key-Value Database Design Considerations

- **A well-designed key pattern helps minimize the amount of code a developer must write to create functions that access and set values.**
  - **One criteria for assessing a naming convention is to consider how much code is required to generate keys in a generic set or get function.**

Key-Value Database Design

- The example pseudo-code in this chapter does not contain error checking.
  - This was probably done to keep the purpose of the code clear.
- Any code running in a production environment should have error checking.
- In some organizations, custom applications should have code to log specific events.
  - The examples in the chapter do not include logging code, either.

Key-Value Database Considerations

- **Key-value databases do not support query languages.**
  - **It is difficult to get or set a range of values unless the key naming convention is designed to support it.**
  - **If developers anticipate having to perform bulk operations on a range of key-value pairs, then the key naming convention should be designed to allow automatic generation of keys in a range.**

Key-Value Database Considerations

- **Key design is restricted by the key-value database you use.**
  - **Some key-value databases require that all keys must be strings. Others, such as Redis, allow for other data types, including lists and sets.**
- **Common partitioning methods are based on hashes and ranges.**
  - ***Hashing* is a common method of partitioning that evenly distributes keys and values across all nodes.**
  - ***Range partitioning* works by grouping contiguous values and sending them to the same node in a cluster.**

Key-Value Database Limitations

- It is important to emphasize the limitations of key-value databases; these include the following:
  - The only way to look up values is by key.
  - Some key-value databases do not support range queries.
  - There is no standard query language comparable to SQL for relational databases.