Key-Value Data Store

Key-Value data stores are one of the simplest NoSQL data stores available. The data is stored using a unique key that is used to identify each data entry and then the value associated with that entry.

The Key can be a single value, or a combination of values joined together to make a unique identifier. In many systems, this is also usually used to distribute the data over multiple servers using the key as a guide on where the data will be located and how it will be ordered in its location.

The value in a key-value store can be anything form a singular value, a series of values or even a just a chunk of binary data. Each item in the store can have different data and can even represent a different type of entity.

The data store will have a unique key for every entry. This can be a singular value, or a combination of values used to define a composite key.

Each key will be associated with a stored value. The value can be any type of data from a single integer to a complex data type or binary data.

In most cases the schema is unique for each item, but some key-value vendors have added the ability to define a schema based upon the entity type.

Data is retrieved using the key only.

Key-value stores to not allow filtering based upon the value data.

When retrieving data you get the whole entry, not just selected fields.

Key

Value 1 | Value 2

Product ID	Type
Book ID	1
Movie ID	2
Movie ID	2
Album ID	3

Value

Different Schema Per Item

Lord of the Rings	JRR Tolkien	19.99	
E.T	Spielberg		
Star Wars	George Lucas	1977	
Load	Metallica	1996	5 x Platinum

The key value/s are commonly used for horizontal scaling to determine the distribution location and order of entries.

For example, you might put all the types together in one server and then order them by Product ID.

Each value represents a single object and is stored in the same collection regardless of entity type. Each entry can have a unique schema and manages no relationships to other records within the database. Most entries are designed to be single simple entities which contain all their data in a single location.

Document Based Database(MongoDb)

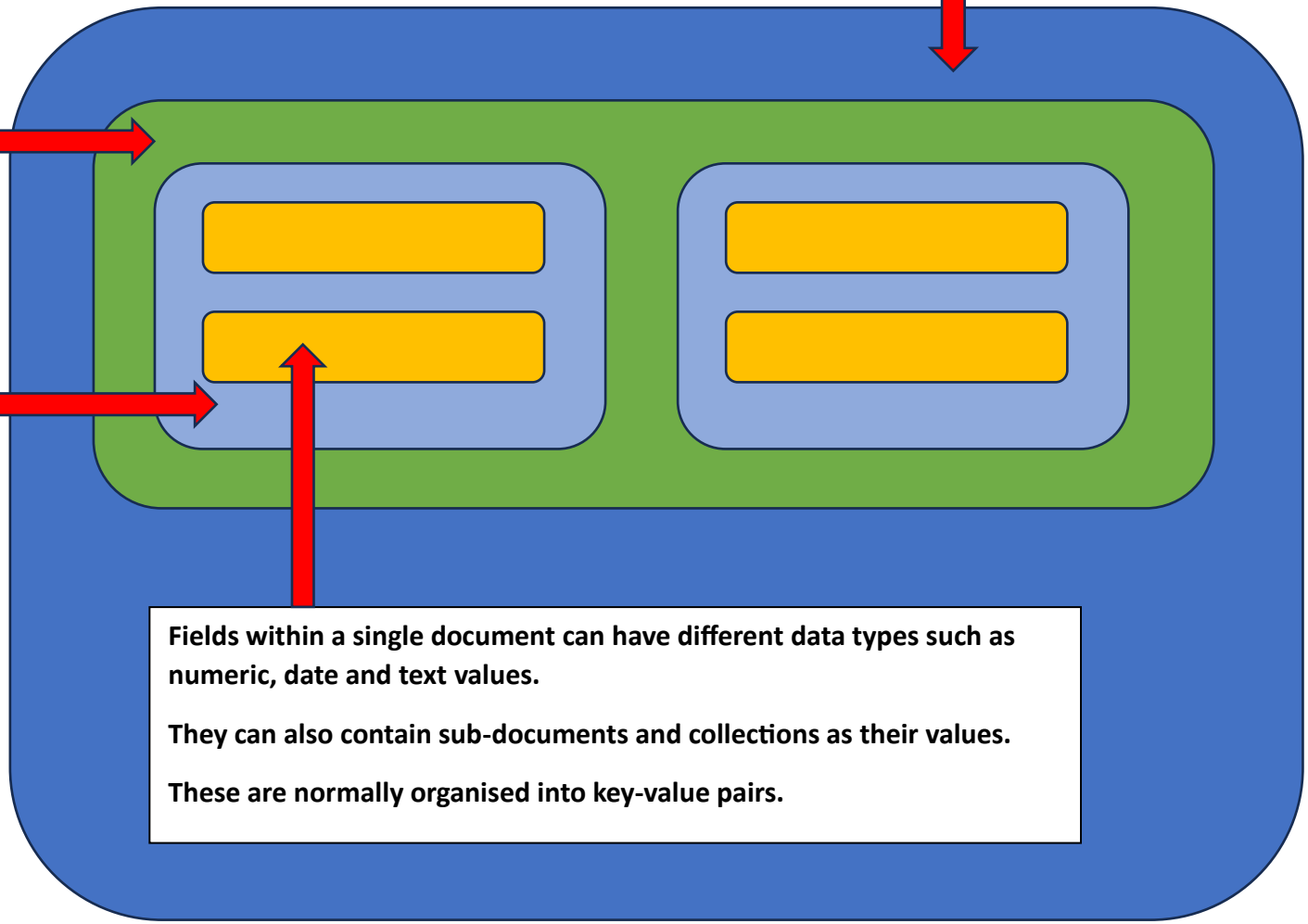
Instead of creating a series of separate tables that interact with and reference each other to organise data, document-based tables put all their data within a single, nested structure. This causes all the data to be located in the one structure for fast access and retrieval. This however does cause data duplication in some structures depending on the data used. In Mongo DB these are stored in BSON(**Binary JavaScript Object Notation**) format, which used a similar structure to JSON but is more compact when stored and has additional versatility in the types of data it can store.

The database will have a unique identifying name and forms the outer layer of the structure.

A collection will sit inside the database. It will have a unique name and will contain multiple documents.

Documents that represent related data. They will have a unique identifier as a 'Primary Key', named '_id' which is normally generated when they are created.

Document data is presented to the user in a JSON styled format.



Fields within a single document can have different data types such as numeric, date and text values.

They can also contain sub-documents and collections as their values.

These are normally organised into key-value pairs.

Example of a Document in NoSQL

Below is an example of how a single Document in a Document Based data store might be structured. As you can see, the document uses a JSON structure which relies on a combination key : value pairs and nesting brackets to define the document, its fields and sub fields.

_id field to act as an identifier when looking for this particular document.

Key : Value pairs to hold each field in the document. The first value in each set holds the field name while the 2nd value holds the associated field value/s.

Sub-fields nested in a set of brackets to indicate a field with multiple sub-values. In this example, instead of the field (address) holding a single value, it contains an object which is indicated by the curl braces. This object then holds a subset of fields and values associated with the main key.

Square brackets to show a field that has multiple related values in the form of a list/array. The field/key hobbies contains a list of multiple values that are all types of hobbies.

Wide Column Database (Cassandra Db, Apache HBase,..)

A wide column database is a NoSQL database system that initially looks like a traditional SQL database. Visually it looks like the data is stored in a table structure similar to modern relational databases. When viewed it does look like a regular table with rows and columns that each store values. The main difference is that at a logical level, instead of having multiple tables with relationships to connect related entities, this data type looks more like a singular giant table.

Visually it looks like a giant table with rows and columns which holds data where these intersect.

Each column name indicates the data that will be stored in the row.

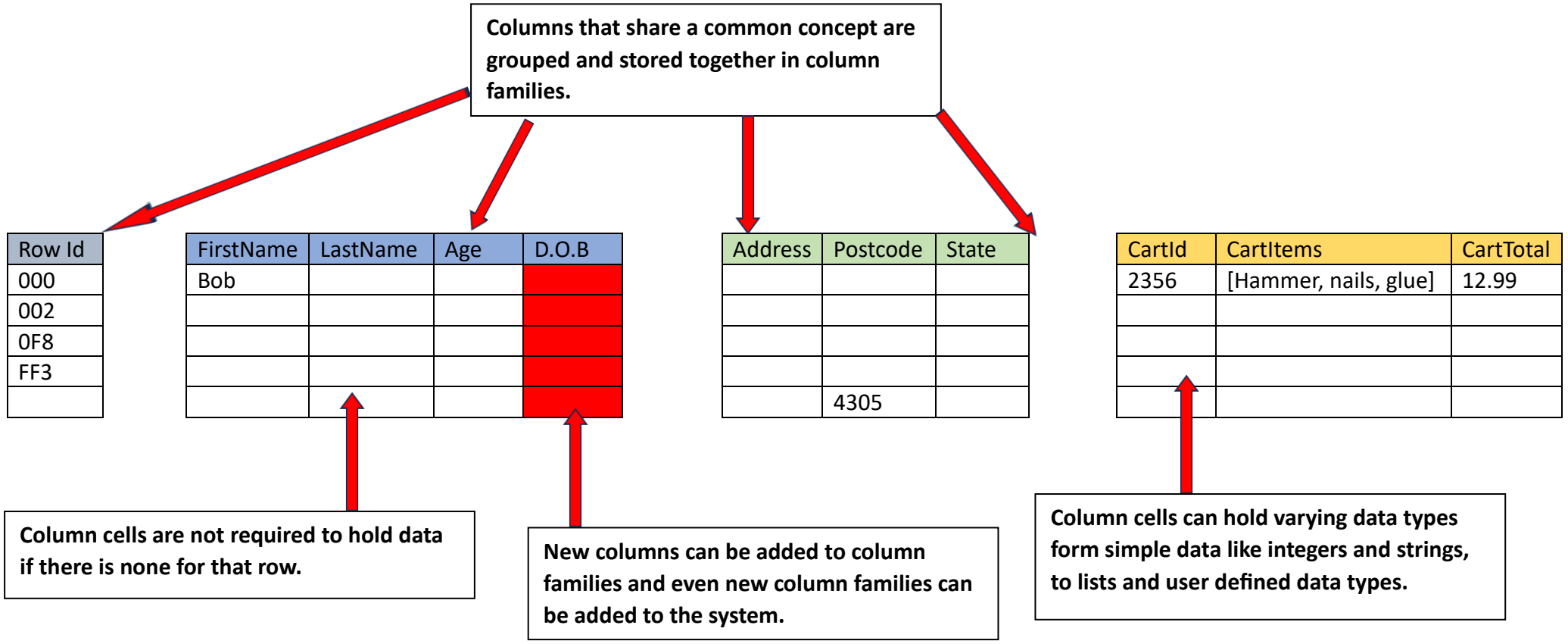
Row Id	FirstName	LastName	Age	Address	Postcode	State	CartId	CartItems	CartTotal
000	Bob								
002									
0F8									
FF3									
					4305				

Each row has an Id that is used to identify the row, similar to a primary key. This can be anything unique hat identifies the row, it does not simply have to be numeric like a SQL database.

Each row holds data associated to the Row Id and column name.

As you can see, on the surface it looks similar to a standard SQL database and has a lot of the same features.

The main difference is in how the data is stored. Instead of storing the data for the table together, organised by the rows like a traditional database. Wide column databases store the data based upon column families. Each set of related columns are grouped together and stored in separate files/locations in the system. This means that the system can be distributed over multiple servers more easily than SQL databases.



Whenever you read or write to the database it simple checks each column family set for the row associated with your row ID and retrieves the data associated with the provided Id. You can retrieve the entire row, specified column families or individual rows when you retrieve data.

NOTE: *There is another alternative called a column-store database which stores every column separately. This works on similar principles but is slower when trying to retrieve related data.*