# Key value Databases


pm_jat @ daiict


# Summarize: "Spark DataFrame API and Spark SQL"

- DataFrame and Relational Table similarities
  - Both are "Collection of Rows" and each row has schema.
  - Both can be queried using SQL
  - Underlying "Query Optimizer"
- However DataFrame is different as follows:
  - DataFrame is in memory
  - DataFrame is distributed
  - Requires registering as Table with Spark-SQL engine before SQL queries can be executed on them
  - Under the hood is Spark and all transformations and query processing happens in parallel on a computing cluster


# Summarize: "Spark DataFrame API and Spark SQL"

- DataFrames are created by
  - Reading data from a file in different formats, "spark-sql-table", or other data sources
  - From already in memory RDD


# Summarize: "Spark DataFrame API and Spark SQL"

- Spark-SQL Table and Relational Table similarities
  - Both are "Collection of Rows" with schema (all the rows homgenous in terms of schema)
  - Both can be queried using SQL
  - Underlying "Query Optimizer"
- However Spark_SQL Table is different as follows:
  - Spark-SQL table is distributed
  - Under the hood is Spark-SQL Engine and query processing happens in parallel on a computing cluster
  - Spark-SQL table can be "in memory"


# Recall: Notion of "Key Value"

- Key value is an efficient mechanism of searching
  - As you know Hashing or binary search trees are efficient search algorithms
- However search can only be performed on key
- In Key value arrangement "Key" is search key where as data object is value.
- Primary operations on Key value are "**Put**" and "**Get**".
- Many No SQL database explores this idea of key value.


# Key Value databases

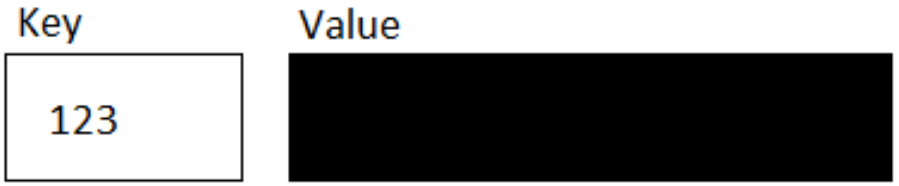
- Key Value databases are based on “Key-Value” strategy.
- These are aggregation oriented databases, and have
  - “Aggregation” as value, and
  - ID of aggregation as Key (every aggregation is supposed to have a key before it can be saved in the database)
- If “Map” is called as “In Memory Database” Key value database is basically a persistent version of Maps.
- Data are distributed on computing nodes based on Key.
- DynamoDB, Redis, Riak are popular key-value databases.

# Key-Value Databases

- Database is collection of Key-Value pairs
- Following mapping [text 1] should help to understand

| RDB | DynamoDB | Riak |
|---|---|---|
| Database | | Riak Cluster |
| Table | Table | Bucket |
| Row | Key-Value | Key-Value |
| RowID/Key | Key | Key |

# Key Value databases - Data Model




- For Key-value databases, value is black chunk of bytes
- That means Key-Value Database Management systems may not do any processing on Values.
- CRUD (Create, Retrieve, Update, and Delete) operations are done **based on Key only**.
- Value part holds a Data object. The representation of data object can be in any format. String, Tuple, CLOB, BLOB, XML, JSON.
- Have No Schema. You can put anything in value part.

# Key and Value

Key | “orderNo”: 1234

Value

```
"customer": {
  "id": 1,
  "name": "Mayank",
  "email": "mayank@gmail.com",
  "billingAddress": {"city": "Bangalore"}
},
"orderItems":[
  {
    "productId":27,
    "price": 325,
    "cat":"book",
    "productName": "NoSQL Distilled"
  },
  {
    "productId":19,
    "price": 550,
    "cat":"computer accessories",
    "productName": "Pen Drive : 128 GB"
  }
],
"shippingAddress":[{"city":"Ahmedabad"}]
```


# Key Value databases - Operations

- A client can either:
  - Get the value for a key (READ)
  - Put a value for a key (Put is ADD/UPDATE both)
  - Delete a key from the data store.
- The value is a blob that the data store just stores, without caring or knowing what's inside;
  - it's the responsibility of the application to understand what was stored.
- Since key-value systems always use primary-key access, they generally have greater performance and can be easily scaled.


# KV database – Pros and Cons

- Pros
  - Efficient queries (very predictable performance).
  - Easy to distribute across a cluster.
  - No impedance (object-relational) miss-match
- Cons
  - No complex query filters (where clause is very limited)
  - All joins must be done in code
  - No foreign key constraints
  - No triggers


# Suitable Use Cases

- Text NoSQL Distilled [1] points out following use cases of Key Value databases
- Common pattern in all following examples is that only operation we perform on database is GET and PUT on Key
- Since everything is stored as a single value, get and put is very fast

## (1) Storing Session Information

Generally, every web session is unique and is assigned a unique sessionid. since everything about the session can be stored by a single PUT request or retrieved using GET.

# Suitable Use Cases

## (2) User Profiles or User Preferences

Almost every user has a unique **userId**, **username**, and some other attribute, as well as preferences such as language, color, timezone, which products the user has access to, and so on.

## (3) Shopping Cart Data

E-commerce websites have shopping carts tied to the user. As we want the shopping carts to be available all the time, across browsers, machines, and sessions, all the shopping information can be put into the value where the key is the **userid**.

- A Key Value store would be best suited for these kinds of applications.


# Popular KV Databases

- Text [1] points out following popular key-value databases
  - Riak,
  - Redis,
  - Amazon DynamoDB,
  - Project Voldemort (an open-source implementation of Amazon DynamoDB).
  - Memcached DB,
  - Berkeley DB,
  - HamsterDB,


# KV Database Case Studies

- We shall look into DynamoDB
  - DynamoDB Architecture
  - Programming Abstraction with some examples
  - Various Techniques used in “implementing DynamoDB”


# References

[1] Chapter 8 from book “NoSQL Distilled”