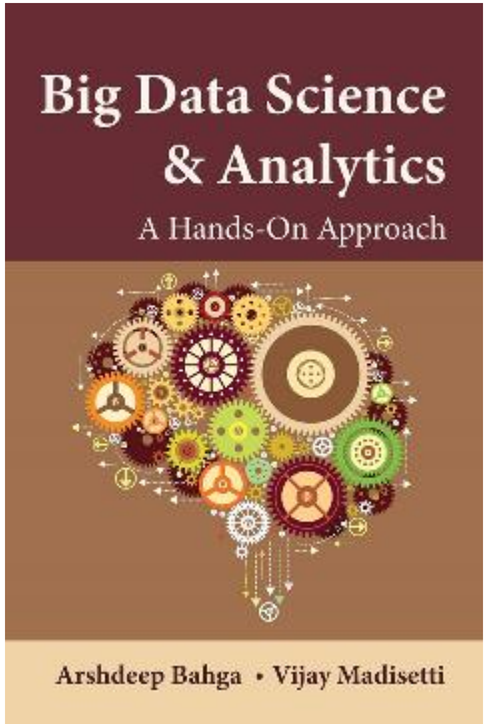
NoSQL Databases

Prof. Gheith Abandah

Reference



- Chapter 4: **NoSQL**

- Arshdeep Bahga and Vijay Madisetti, **Big Data Science and Analytics: A Hands-On Approach**, 2019.
  - Web site: http://www.hands-on-books-series.com/

Outline

- Introduction
- Key-value databases
- Document databases
- Column family databases
- Graph databases
- Summary

Introduction

- **Non-relational databases** (**NoSQL**) are becoming **popular** with the increasing use of cloud computing services.
- They have better **horizontal scaling** capability and **improved performance** for big data at the cost of having **less rigorous consistency** models.
- Optimized for **fast retrieval** and **appending** operations on records.

Outline

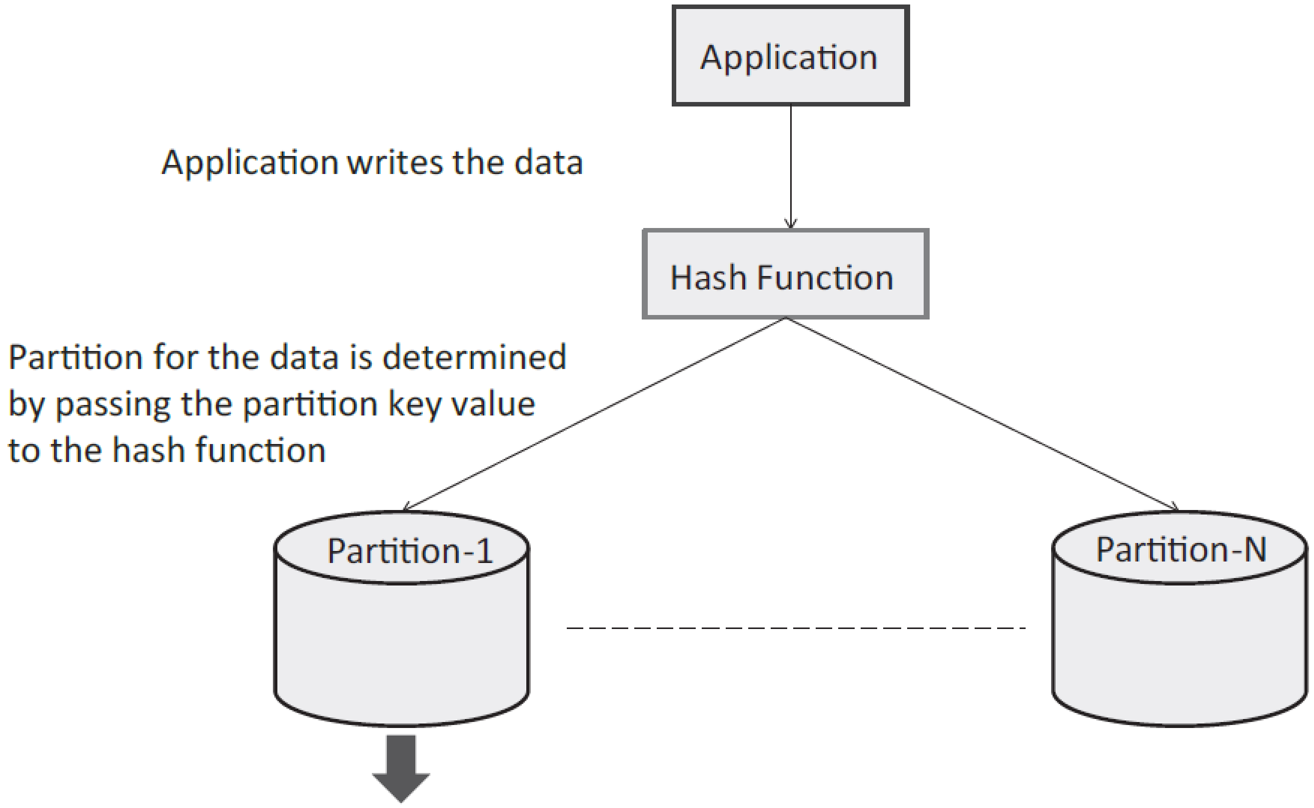
- Introduction
- Key-value databases
- Document databases
- Column family databases
- Graph databases
- Summary

Key-Value Databases

- These databases store data in the form of **key-value pairs**.
- The database uses **unique keys** to determine where the values should be stored.
- **Hashing** is used for determining the partitions for the keys.
- The values can be virtually of **any type**.
- Unlike relational databases, there are **no** constraints of **fixed schemas** and columns.
- Some key-value databases support **tables**, **buckets** or **collections** to create separate namespaces for the keys.

Amazon DynamoDB

- **Scalable**, **reliable** and **high-performance** key-value DB.
- A DynamoDB **table** is a collection of **items** and each item is a collection of **attributes** (k, v).
- The **primary key** consists of:
  - **Partition key** that hashes the partition
  - Optional **sort key** within the partition



Key	value
161305173	{name: `Ivor Merritt', address: `Ap #527-9960 Vel St.', city: `Lauw',zip: `5624',country: `Peru'}
162307206	{name: `Cade Nguyen', address: `486, 6221 Et St.', city: `Barnstaple',zip: `10903',country: `Ukraine'}

Main operations

- Put Item
- Query
- Scan

```
item = table.put_item(data={
  ‘customerID’:row[0],
  ‘name’:row[1],
  ‘address’: row[2], …
  },overwrite=True)

result = table.query_2(
  customerID__eq = ’1623072020799’)

result = table.scan(
  country__eq =’India’)
```

Outline

- Introduction
- Key-value databases
- Document databases
- Column family databases
- Graph databases
- Summary

Document Databases

- Store **semi-structured data** in the form of **documents** which are encoded in different standards such as **JSON** and **XML**.
- **One collection's** documents have **similar fields**.
- They allow efficiently **querying** the documents based on the **attribute values** in the documents.
- **All data** that needs to be retrieved together is **stored in one document**.

ID	Document
56fd4f59849f6367af489537	{ "title" : "Motorola Moto G (3rd Generation)", "features" : ["Advanced water resistance", "13 MP camera", "5in HD display", "Quad core processing power", "5MP rear camera", "Great 24hr battery", "4G LTE Speed"], "specifications" : { "Color" : "Black", "Size" : "16 GB", "Dimensions" : "0.2 x 2.9 x 5.6 inches", "Weight" : "5.4 ounces" }, "price" : 219.99 }
56fd504d849f6367af489538	{ "title" : "Canon EOS Rebel T5", "features" : [

MongoDB

- Is a **powerful**, **flexible** and **highly scalable** document database system.
- Is designed for **web applications** and **serving database** for data analytics applications.
- A document includes a **JSON-like** set of key-value pairs.
- **Documents** are grouped together to form **collections**. Collections can have documents with different sets of key-value pairs.
- Collections are organized into **databases**, and there can be multiple databases running on a single **MongoDB instance**.

MongoDB Python Command Examples

```
# Insert an item
collection.insert_one(item)

# Retrieve all items
results = db.collection.find()
for item in results:
    print(item)

# Find an item
results = collection.find({"title" : "Motorola Moto G"})
for item in results:
    print(item)
```

Outline

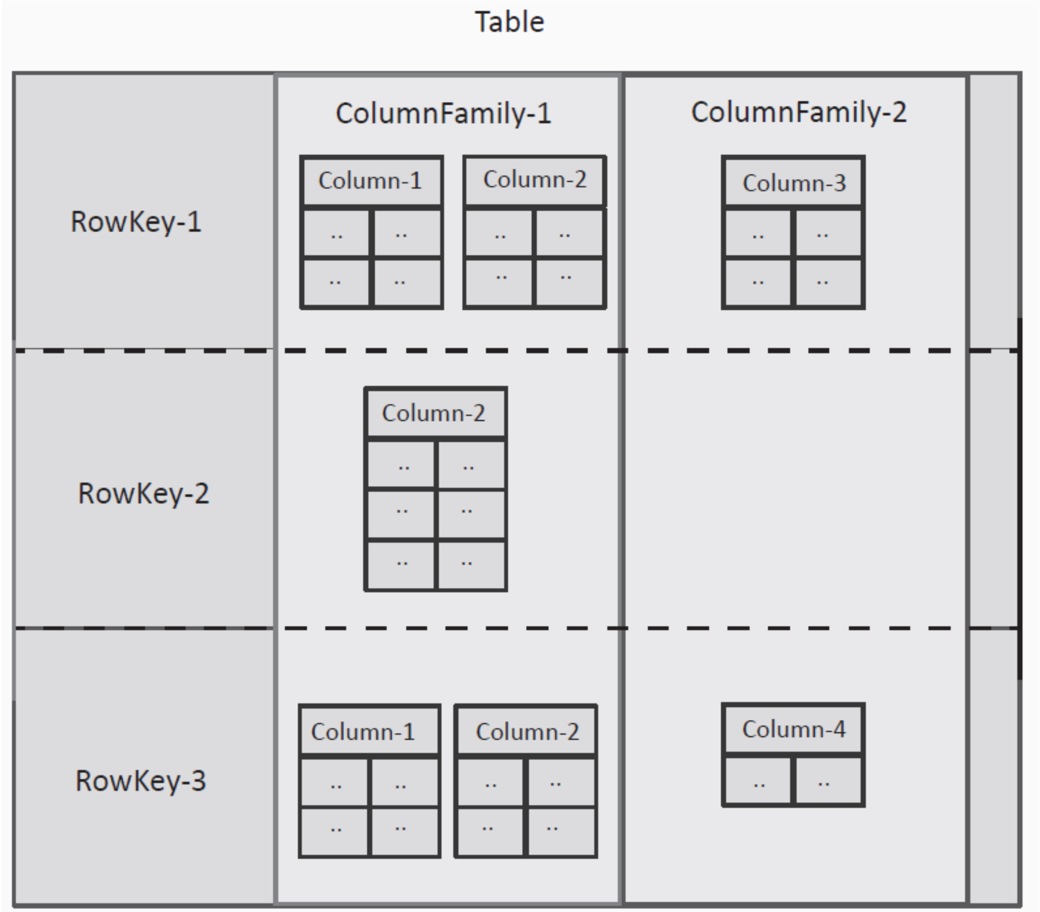
- Introduction
- Key-value databases
- Document databases
- Column family databases
- Graph databases
- Summary

Column Family Databases

- Support **high-throughput reads** and **writes** and have **distributed** and **highly available** architectures.
- In column family databases the basic unit of data storage is a **column**, which has a name and a **value**.
- A collection of columns make up a row which is identified by a **row-key**.
- Columns are grouped together into **columns families**.
- The number of **columns** can **vary** across different rows.
- All **information** related to an **entity** can be retrieved by reading a **single row**.

HBase

- **Scalable**, **distributed**, column-family database that provides **structured data storage** for **large tables**.
- A table consists of **rows** indexed by the **row key**.
- Each row includes **multiple column families**.
- Each column includes **multiple cells** which are **timestamped**.
- HBase tables are **indexed** by the **row key**, **column key** and **timestamp**.

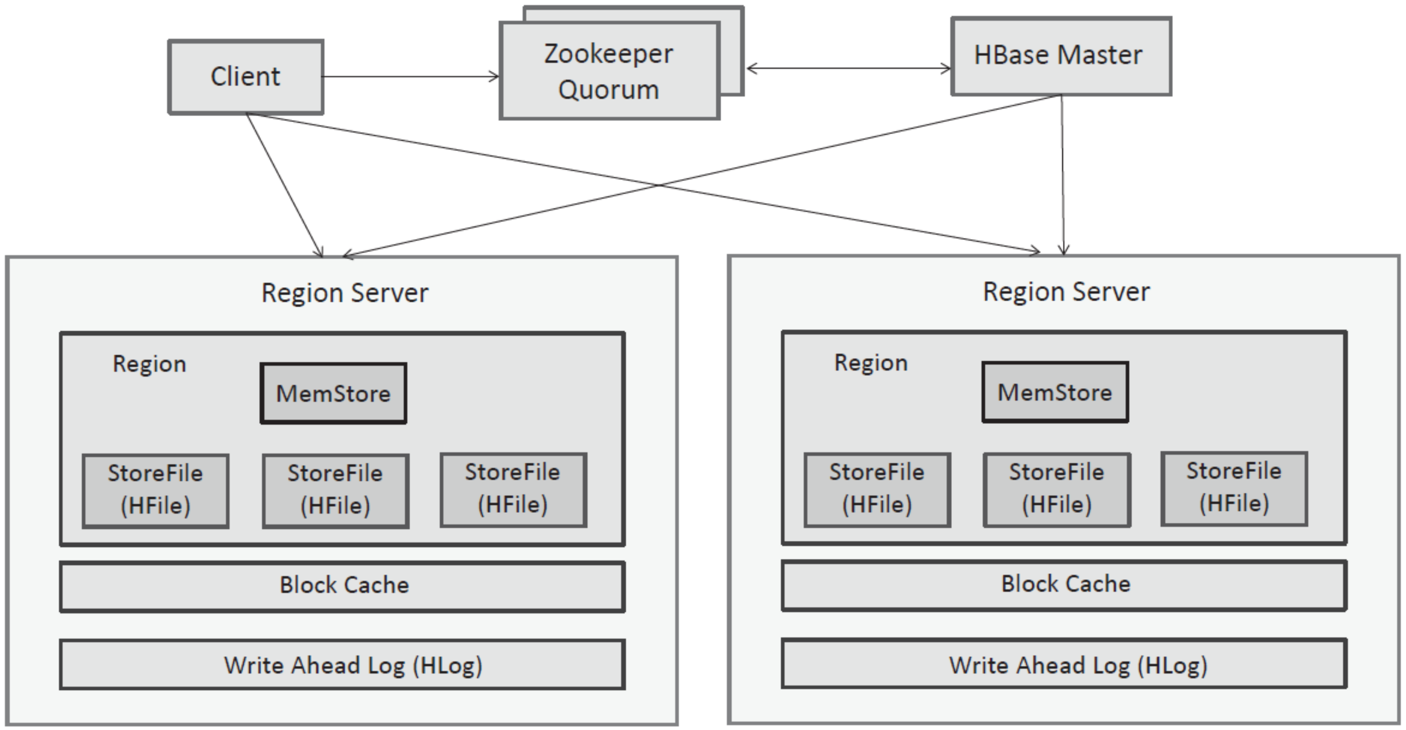


HBase

- **Columns families** are declared at the time of creation of the table and **cannot be changed** later.
- **Columns** can be **added dynamically**.
- **HBase** is:
  - **Sparse**: not all row/column entries are present
  - **Distributed**: tables are partitioned based on row keys into regions.
  - **Persistent**: Not temporary
  - **Multi-dimensional**: A key includes Table, RowKey, ColumnFamily, Column, TimeStamp
  - **Sorted Map**: Rows are sorted by the row key. Columns in a column family are sorted by the column key.

HBase Architecture

- **Multiple region servers/regions**
- The **Master** is responsible for maintaining the meta-data and assignment of regions to servers.
- The **Zookeeper** coordinates the distributed state.
- **HFiles** and **HLogs** are persistent and **MemStore** and **Block Cache** improve performance.



HBase Operations

- **Put**: adds a new entry.
- **Get**: returns values for a given row key.
- **Scan**: returns values for a range of row keys.
- **Delete**: adds a special marker called Tombstone to an entry. Entries marked with Tombstones are removed during the compaction process.

```
# Put
table.put(‘row-1’,
    ‘details:name’: ‘Cloud Book’)

# Get
row = table.row(‘row-1’)
print(row[‘details:name’])

# Scan
for key, data in table.scan():
    print(key, data)

# Delete row
row = table.delete(‘row-1’)
```

Outline

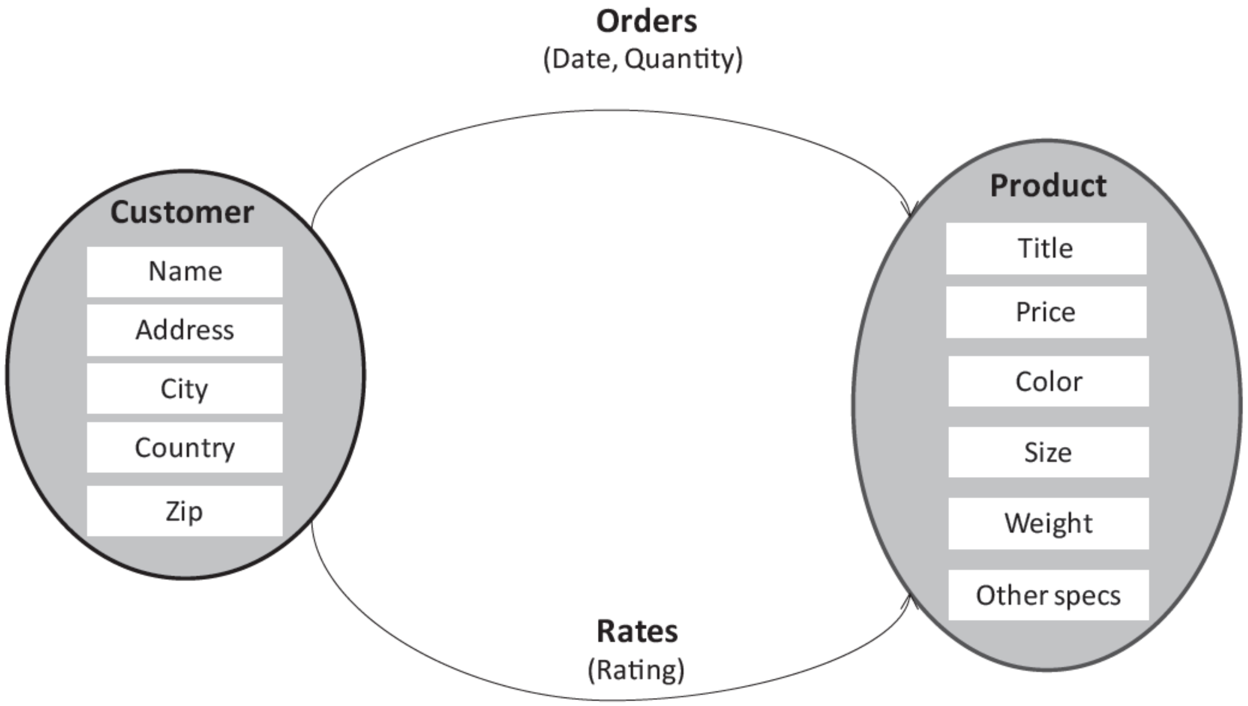
- Introduction
- Key-value databases
- Document databases
- Column family databases
- Graph databases
- Summary

Graph Databases

- Designed for **storing data** that has **graph structure** with nodes and edges.
- **Nodes** represent the entities in the data model and have attributes.
- **Edges** are the relationships between the entities.
- **Examples**: Author → Book, Ali ↔ Wafa, A – path – B
- Useful in **social media**, **financial**, **networking**, **enterprise applications**.
- **Model relationships** in the form of **links** between the nodes; no need for join operations.

Neo4j

- Provides support for Atomicity, Consistency, Isolation, Durability (**ACID**).
- For create, read, update and delete (**CRUD**) operations, Neo4j provides a query language called **Cypher**.
  - **Create** a node or a relationship
  - **Query** for a node or a label



Outline

- Introduction
- Key-value databases
- Document databases
- Column family databases
- Graph databases
- Summary

Summary

	Key-Value DB	Document DB	Column Family DB	Graph DB
Data model	Key-value pairs uniquely identified by keys	Documents (having key-value pairs) uniquely identified by document IDs	Columns having names and values, grouped into column families	Graphs comprising of nodes and relationships
Querying	Query items by key, Database specific APIs	Query documents by document-ID, Database specific APIs	Query rows by key, Database specific APIs	Graph query language such as Cypher, Database specific APIs
Use	Applications involving frequent small reads and writes with simple data models	Applications involving data in the form of documents encoded in formats such as JSON or XML, documents can have varying number of attributes	Applications involving large volumes of data, high throughput reads and writes, high availability requirements	Applications involving data on entities and relationships between the entities, spatial data
Examples	DynamoDB, Cassandra	MongoDB, CouchDB	HBase, Google BigTable	Neo4j, AllegroGraph