Unit 4
Data Storage and Security in Cloud

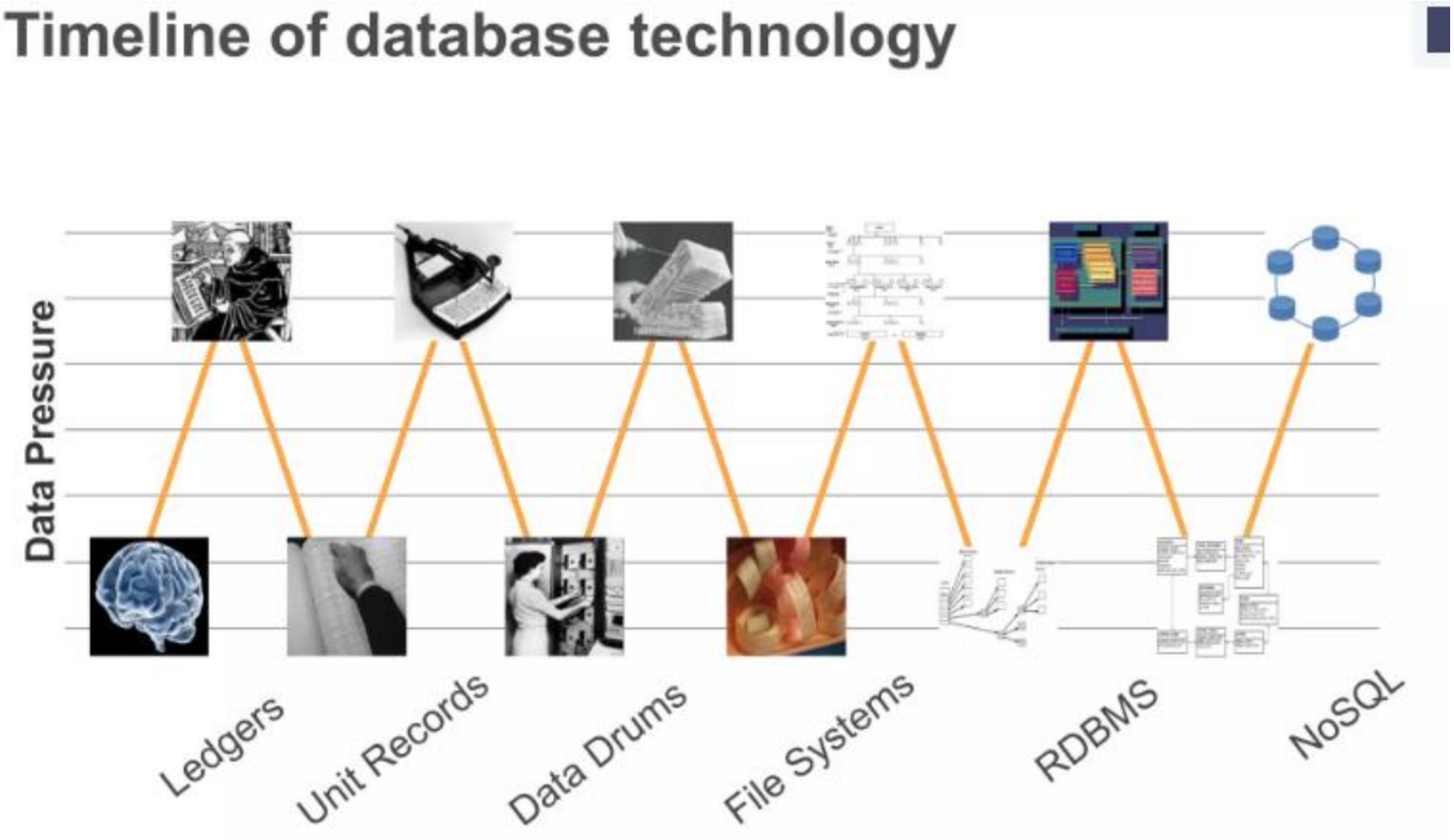
Cloud file systems: GFS and HDFS, BigTable, HBase and Dynamo Cloud data stores: Datastore and Simple DB, Cloud Storage-Overview, Cloud Storage Providers.

Securing the Cloud- General Security Advantages of Cloud-Based Solutions, Introducing Business Continuity and Disaster Recovery. Disaster Recovery- Understanding the Threats.

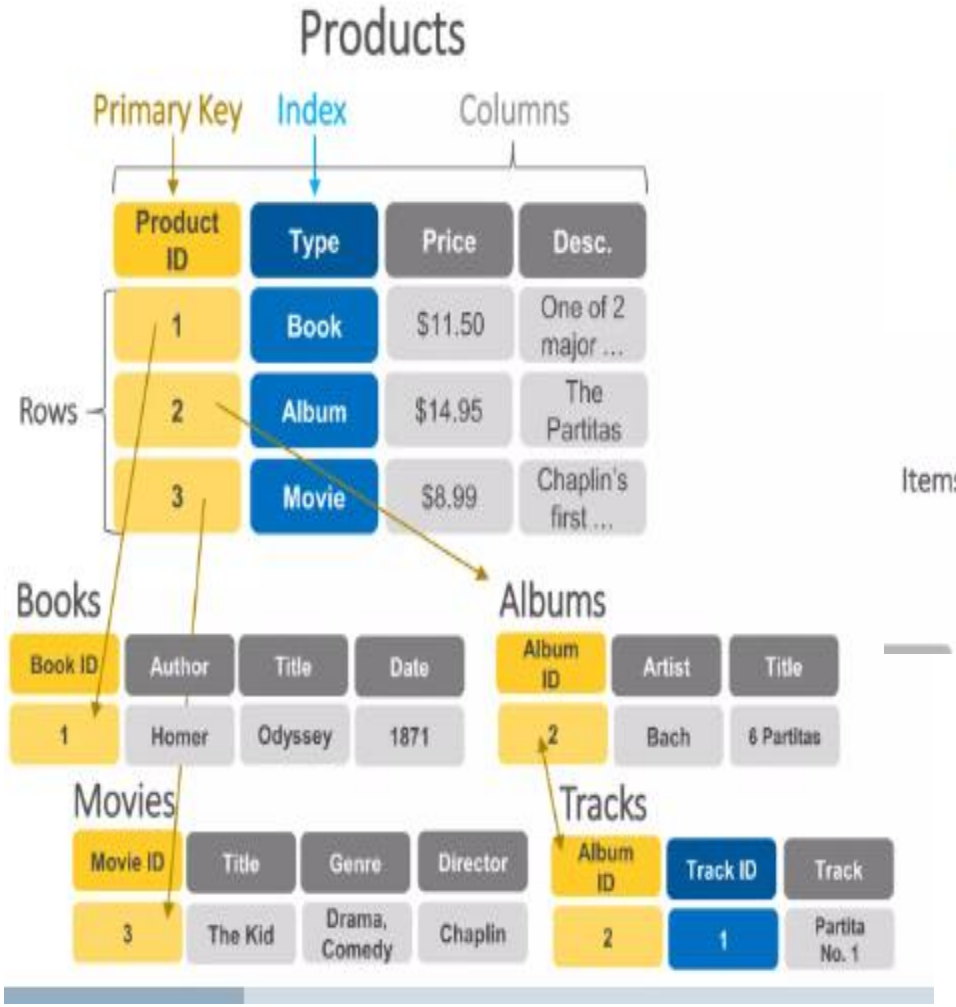
Unit 4
Data Storage and Security in Cloud

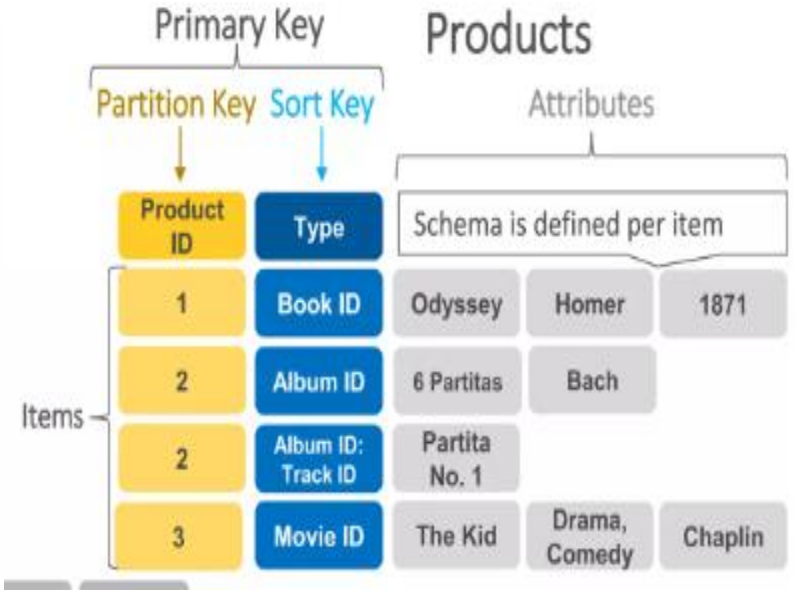
Apache HBase

Introduction

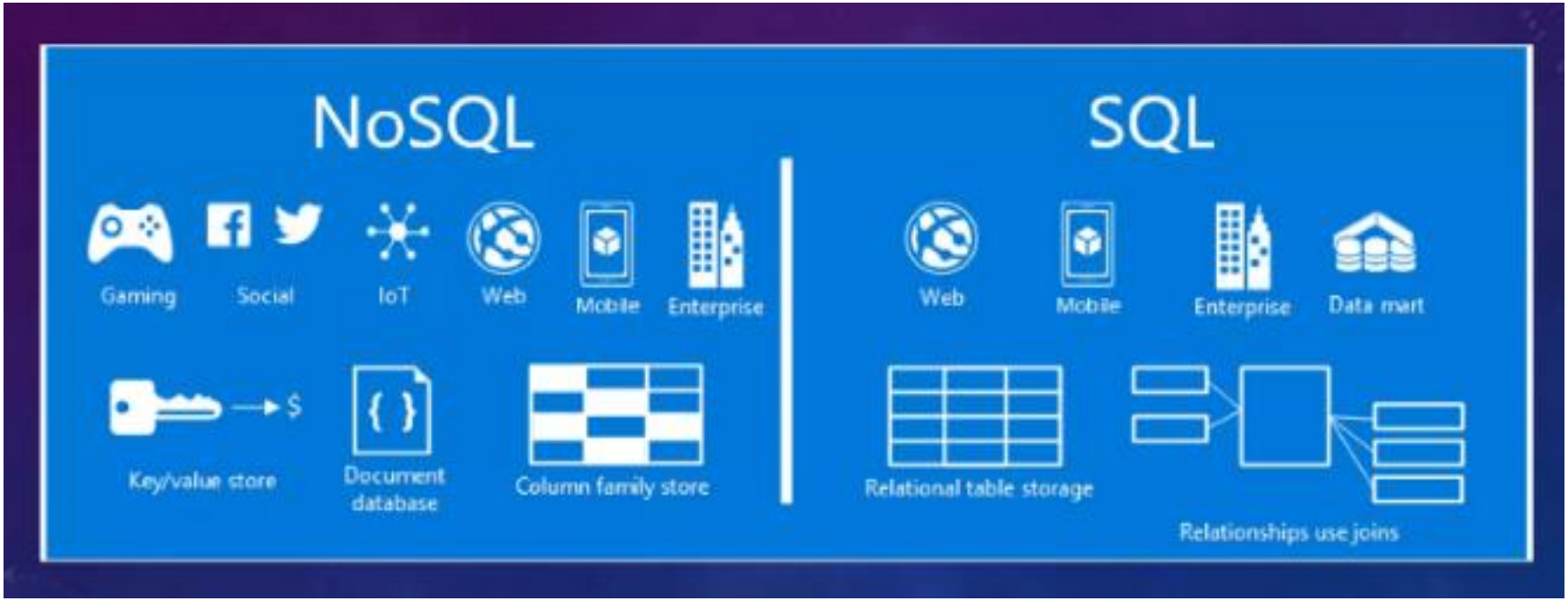


SQL vs. NoSQL


Products
Primary Key
Index
Columns
Product ID
Type
Price
Desc.
1
Book
$11.50
One of 2 major …
2
Album
$14.95
The Partitas
3
Movie
$8.99
Chaplin's first …
Rows
Books
Book ID
Author
Title
Date
1
Homer
Odyssey
1871
Albums
Album ID
Artist
Title
2
Bach
6 Partitas
Movies
Movie ID
Title
Genre
Director
3
The Kid
Drama, Comedy
Chaplin
Tracks
Album ID
Track ID
Track
2
1
Partita No. 1


Primary Key
Products
Partition Key
Sort Key
Attributes
Product ID
Type
Schema is defined per item
Items
1
Book ID
Odyssey
Homer
1871
2
Album ID
6 Partitas
Bach
2
Album ID: Track ID
Partita No. 1
3
Movie ID
The Kid
Drama, Comedy
Chaplin

SQL vs. NoSQL


NoSQL
Gaming
Social
IoT
Web
Mobile
Enterprise
Key/value store
Document database
Column family store
SQL
Web
Mobile
Enterprise
Data mart
Relational table storage
Relationships use joins

SQL vs. NoSQL

- What is SQL?
  - Structured Query Language (SQL) databases store data in structured tables with predefined schema.
- What is NoSQL?
  - NoSQL databases store unstructured or semi-structured data and support flexible schema designs

- SQL relational (Normalized) databases are **optimized for storage** while NoSQL is **optimized for Compute** (De-normalized).

- SQL **scale vertically** while NoSQL **scale horizontally**.

SQL Databases - Characteristics

- Uses structured tables with relationships.
- Ensures data integrity through ACID properties.
- Examples: MySQL, PostgreSQL, Oracle, SQL Server.
- Best for applications requiring strong consistency (e.g., banking, ERP systems

NoSQL Databases - Characteristics

- Flexible data storage (JSON, XML, key-value pairs, graphs, etc.).
- Supports high availability and horizontal scaling.
- Examples: MongoDB, Cassandra, Redis, Neo4j.
- Best for big data, real-time applications, and distributed systems.

SQL vs. NoSQL

NOSQL VS MYSQL

- Data:
  - NoSQL:
    - Offers flexibility as not every record needs to store the same properties.
    - New properties can be added on the fly.
    - Good for semi structure, complex and nested data.
    - Relation captured by denormalizing data and presenting data in single object in a single record.
  - SQL:
    - Used where the solution for every record has same property.
    - Adding properties may require altering schema or backfilling of data.
    - Good for structured data.
    - Relations are captured in normalized model using joins to resolve reference across the tables.

Limitations of Hadoop

- Batch Processing:Hadoop's MapReduce framework and HDFS are designed for batch processing.
  - suitable for large-scale analytics and offline processing but not for real-time operations.
- Random Read/Write Access: HDFS is optimized for sequential read/write operations.
- Limited Support for Low-Latency Queries:Hadoop, especially the MapReduce framework, is not optimized for low-latency queries.

What is HBase?

- It is modeled after **Google Bigtable**
- HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.
- It leverages the fault tolerance provided by the Hadoop File System (HDFS).
- It is a part of the Hadoop ecosystem that provides **random real-time read/write access to data** in the Hadoop File System.
- One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase.
- HBase sits on top of the Hadoop File System and provides read and write access.

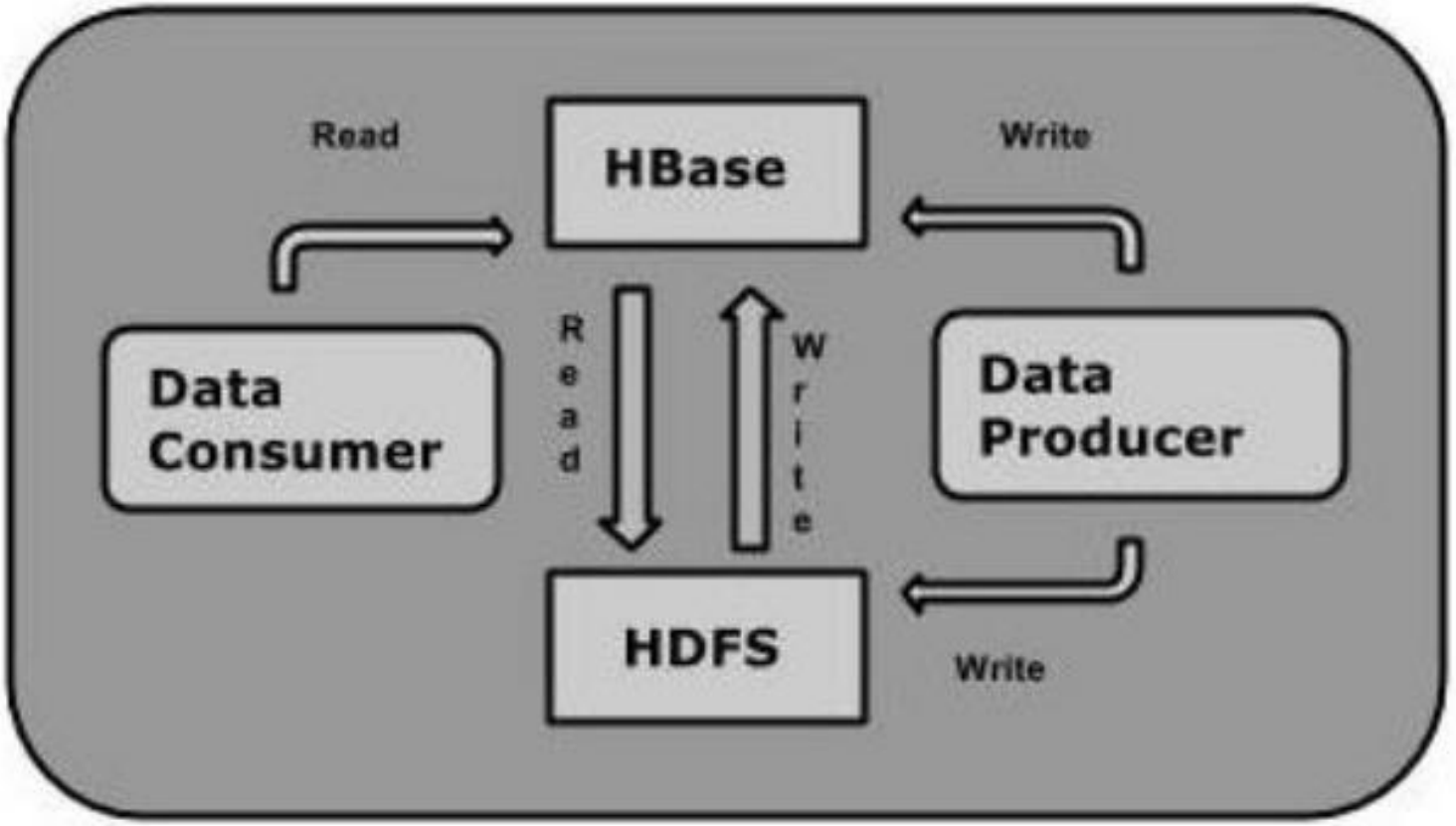
Bigtable

- Google Bigtable is a **NoSQL distributed storage system** for managing petabyte-scale **structured data**. Bigtable is designed for fast, low-latency access to data, with scalability and reliability in mind.
- Internally, Google uses Bigtable for a number of services, including Google Earth, web indexing, and Google Analytics. While Bigtable stores data in a **tabular format**, it is *not* a relational database.
- Google Bigtable is a distributed, column-family-based NoSQL database that stores data in highly scalable tables. The tables are made up of rows and columns, and are organized into blocks called tablets.

Bigtable

- The tables are stored in the SSTable format on Google's Colossus file system
- SSTables are files that contain sorted key-value pairs
- Each row is indexed by a row key
- Columns that are related are grouped into column families
- Each column is identified by a column family and a column qualifier
- Each row/column intersection can contain one or more cells
- Each cell contains a timestamped version of the data
- The tables are sparse, so cells that don't contain data don't use any storage space

HBase


Read
HBase
Write
Data Consumer
Read
Write
Data Producer
HDFS
Write

HDFS vs. HBase

HDFS	HBase
HDFS is a distributed file system suitable for storing large files.	HBase is a database built on top of the HDFS.
HDFS does not support fast individual record lookups.	HBase provides fast lookups for larger tables.
It provides high latency batch processing; no concept of batch processing.	It provides low latency access to single rows from billions of records (Random access).
It provides only sequential access of data.	HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

Storage Mechanism

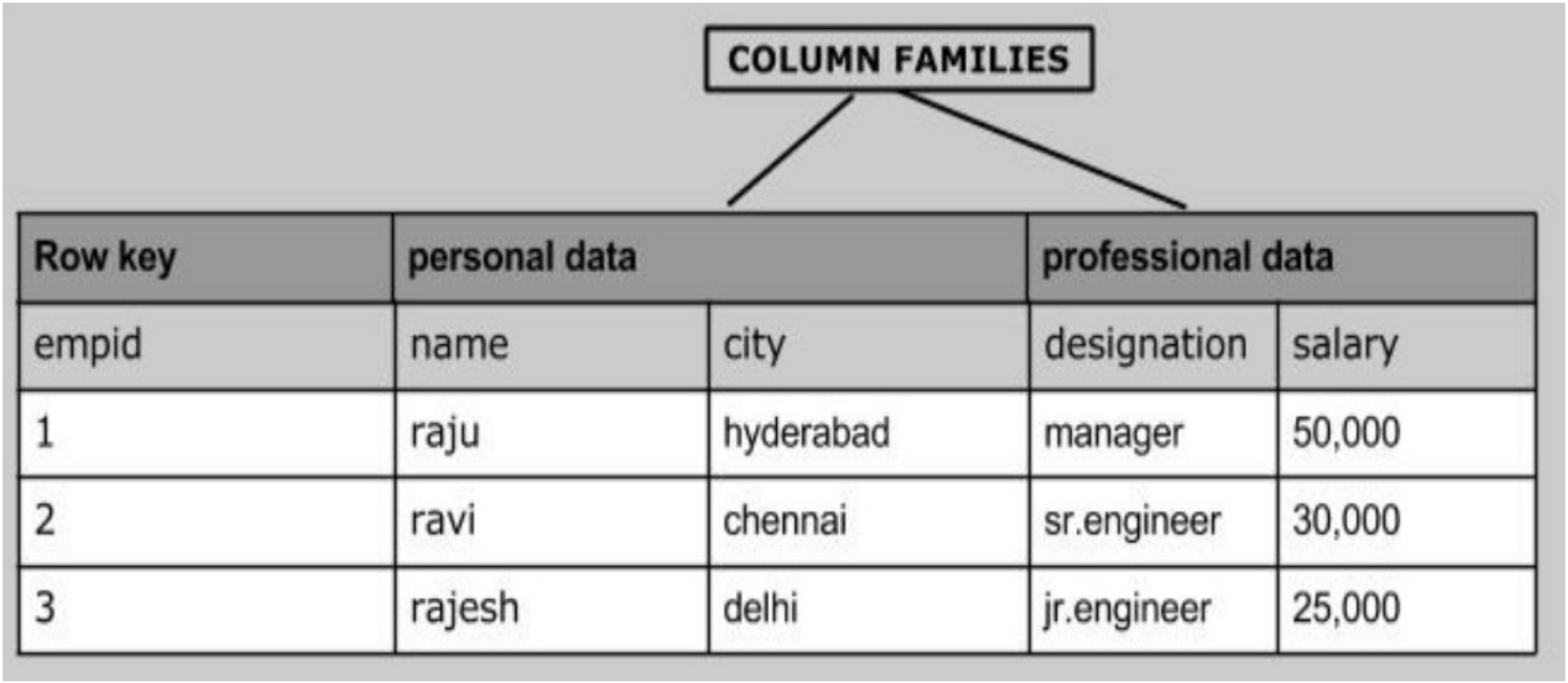
- HBase is a column-oriented database and the tables in it are sorted by row.
- The table schema defines only column families, which are the key value pairs.
- A table have multiple column families and each column family can have any number of columns.
- Subsequent column values are stored contiguously on the disk. Each cell value of the table has a timestamp. In short, in an Hbase:
  - Table is a collection of rows.
  - Row is a collection of column families.
  - Column family is a collection of columns.
  - Each table must have at least one column family,
  - Column is a collection of key value pairs.

Storage Mechanism

- Table of user data, you might have column families like
  - PersonalInfo (with columns like name, email)
  - AccountInfo (with columns like last_login, account_status).

Rowid	Column Family			Column Family			Column Family			Column Family		
	clo1	col 2	col 3	col 1	col 2	col 3	col1	col2	col3	col1	col2	col3
1												
2												
3												

Example:



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

Column Oriented vs. Row Oriented

how data is physically stored and accessed on disk,

1. **Row-Oriented Databases (Traditional Model):D**ata is stored row by row

e.g. e-commerce, banking, or order processing systems

2. Column-Oriented Databases:data is stored column by column

This means that all values of a particular column are stored together.

Used in (OLAP) applications, which typically perform complex queries like aggregations, filtering, and scanning on large datasets.

Row-Oriented Database	Column-Oriented Database
It is suitable for Online Transaction Process (OLTP).	It is suitable for Online Analytical Processing (OLAP).
Such databases are designed for small number of rows and columns.	Column-oriented databases are designed for huge tables.

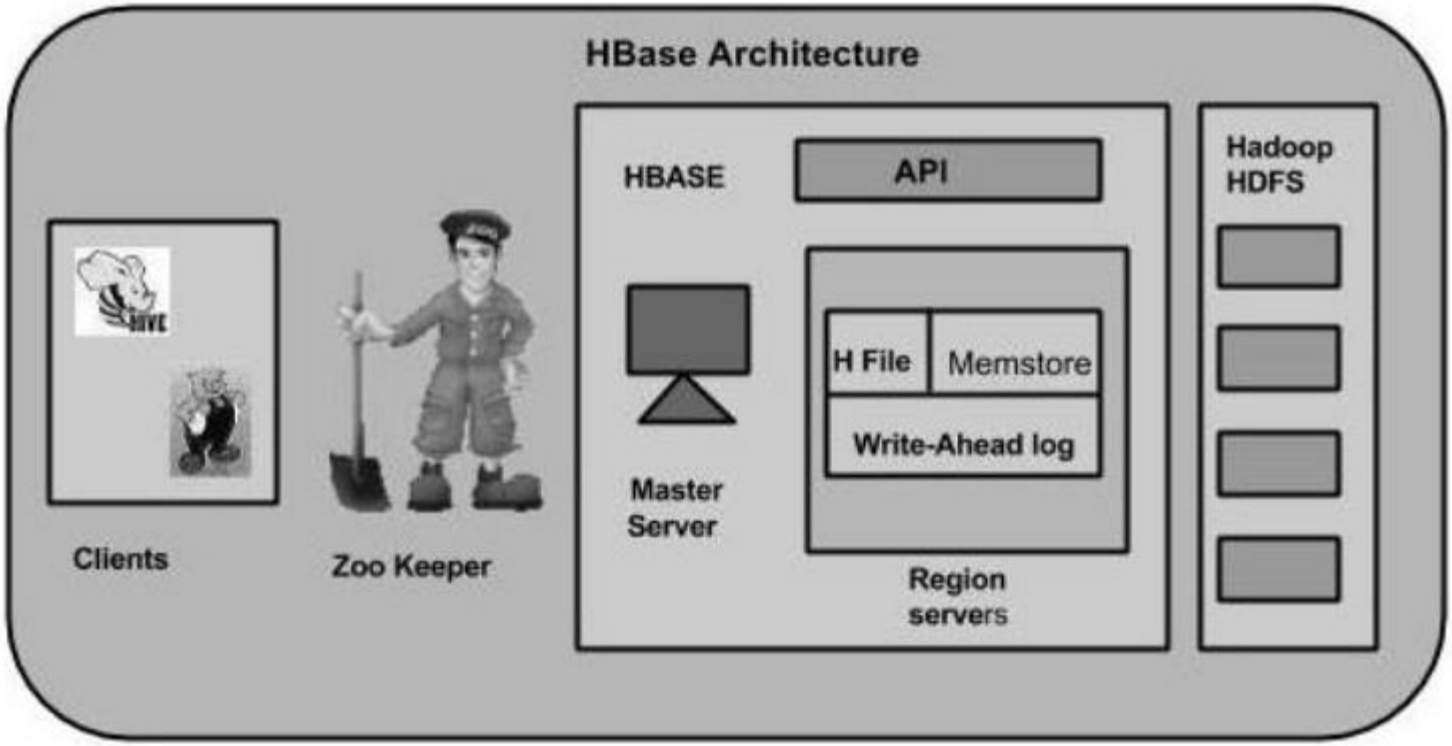
Features

- HBase is linearly scalable.
- It has automatic failure support.
- It provides consistent read and writes.
- It integrates with Hadoop, both as a source and a destination.
- It has easy java API for client.
- It provides data replication across clusters.

Uses

- It is used whenever there is a need to write heavy applications.
- HBase is used whenever we need to provide fast random access to available data.
- Companies such as Facebook, Twitter, Yahoo, and Adobe use Hbase internally.

Architecture



Master server

- Assigns regions to the region servers and takes the help of Apache ZooKeeper for this task.
- Handles load balancing of the regions across region servers. It unloads the busy servers and shifts the regions to less occupied servers.
- Maintains the state of the cluster by negotiating the load balancing.
- Is responsible for schema changes and other metadata operations such as creation of tables and column families.

Region server

- Regions:Regions are nothing but tables that are split up and spread across the region servers.
- Region server:The region servers have regions
- The region servers also store data in HFile format on HDFS.
  - Communicate with the client and handle data-related operations.
  - Handle read and write requests for all the regions under it.
  - Decide the size of the region by following the region size thresholds.

Zookeeper

- HBase relies on **Apache ZooKeeper** for **coordination, failure detection, and distributed management** of **RegionServers** and **HMaster servers**.
- Zookeeper has ephemeral nodes representing different region servers. Master servers use these nodes to track active RegionServers dynamically.
- In addition to availability, the nodes are also used to track server failures or network partitions.
- Clients communicate with region servers via zookeeper.
- In standalone modes, HBase itself will take care of zookeeper.

- **MemStore**:
  - **It** is an in-memory data store where recent writes are buffered before being flushed to disk.
  - Data is written to MemStore before being written to HFiles on disk.
- **HFile**:on-disk storage format used by HBase for storing data.
  - HFiles are immutable, meaning they cannot be updated.
  - When updates occur, they are written to MemStore, and periodically, MemStore is flushed to HDFS as a new HFile.

Hbase Shell

- HBase contains a shell using which you can communicate with HBase.
- Hbase uses the Hadoop File System to store its data.
- It will have a master server and region servers. The data storage will be in the form of regions (tables).
- These regions will be split up and stored in region servers.

General Commands

- Status: Provides the status of HBase, for example, the number of servers.
- version: Provides the version of HBase being used.
- table_help: Provides help for table-reference commands.
- whoami: Provides information about the user.

DDL Commands

- create: Creates a table.
- list: Lists all the tables in HBase.
- disable: Disables a table.
- is_disabled: Verifies whether a table is disabled.
- enable: Enables a table.
- is_enabled: Verifies whether a table is enabled.
- describe: Provides the description of a table.
- alter: Alters a table.
- exists: Verifies whether a table exists.
- drop: Drops a table from HBase.

DML Commands

- put: Puts a cell value at a specified column in a specified row in a particular table.
- get: Fetches the contents of row or a cell.
- delete: Deletes a cell value in a table.
- deleteall: Deletes all the cells in a given row.
- scan: Scans and returns the table data.
- count: Counts and returns the number of rows in a table.
- truncate: Disables, drops, and recreates a specified table.

Create table

Row key	personal data	professional data

- create 'emp', 'personal data', ‘professional data’

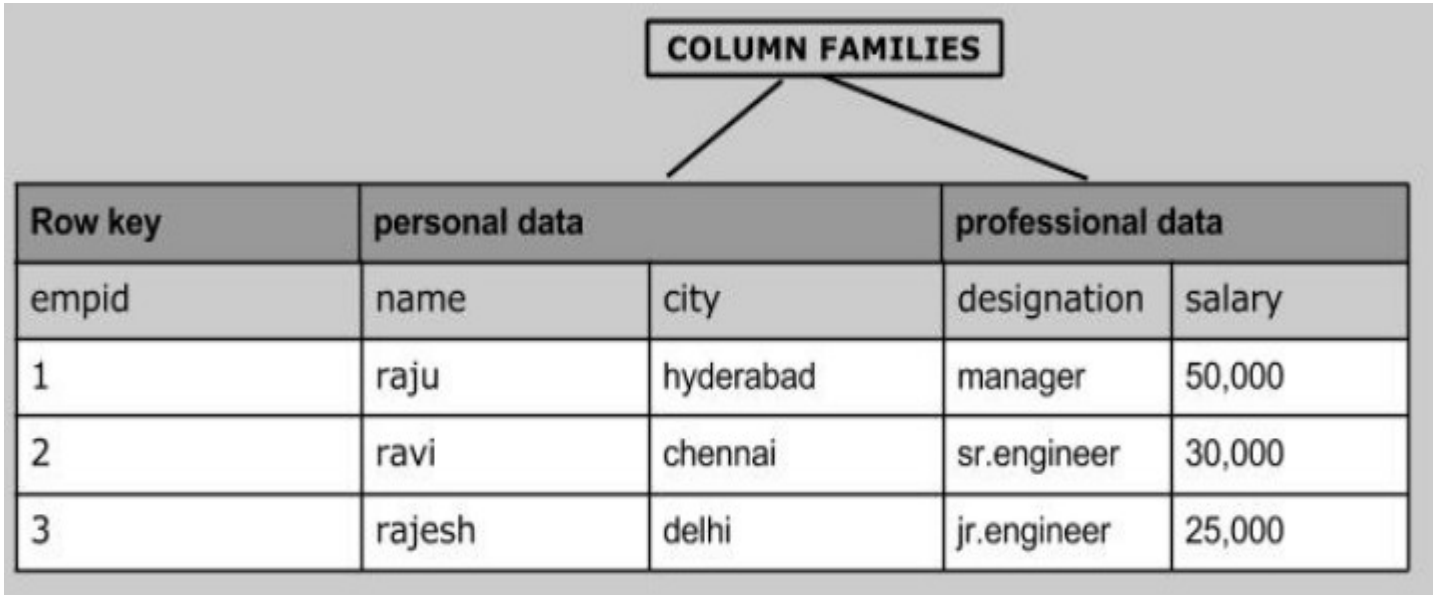
Describe table

- describe ‘tablename’

Create data

- To create data in an HBase table, the following commands and methods are used:
  - put command,
  - add() method of Put class, and
  - put() method of HTable class.

Example:



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

Example:

- put 'emp','1','personal data:name','raju'
- put 'emp','1','personal data:city','mumbai'
- put 'emp','1','professional data:designation','manager'
- put 'emp','1','professional data:salary','50000'
-

Display:

- scan ‘tablename’

Update:

- put 'emp',1 ,'personal:city','Delhi'

Read Data:

- get 'emp', '1'
- get 'emp', '1', {COLUMN=>'personal:name'}

Read Data:

- delete 'emp', '1', 'personal data:city'

Count and truncate

- You can count the number of rows of a table using the count command.

  count ‘emp’
- This command disables drops and recreates a table.

  truncate ‘tablename’