NoSQL Databases

What are SQL databases?

Relational Databases

… Rows & Columns

What are SQL databases?

- Structured Data Management
- Tables with rows (records) and columns (fields)

Col1	Col2	Col3	Col4
A	–	–	–
B	–	–	–

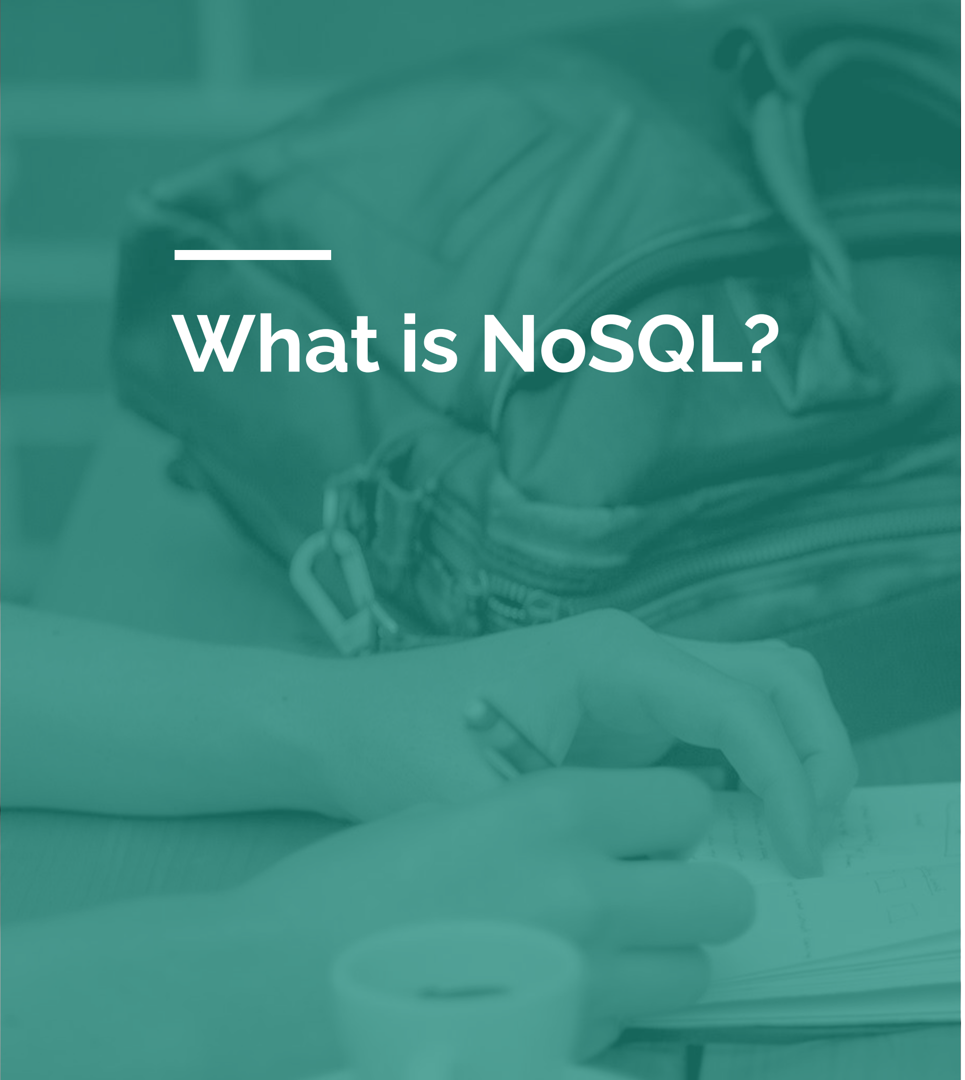
- SQL (Structured Query Language) for data manipulation

Moving Beyond Relational Databases

- Relational databases (RDBMS) have ruled for decades.
- Big Data brings new challenges:
  - **Volume, Velocity, Variety**
- Rigid schemas can become a bottleneck.

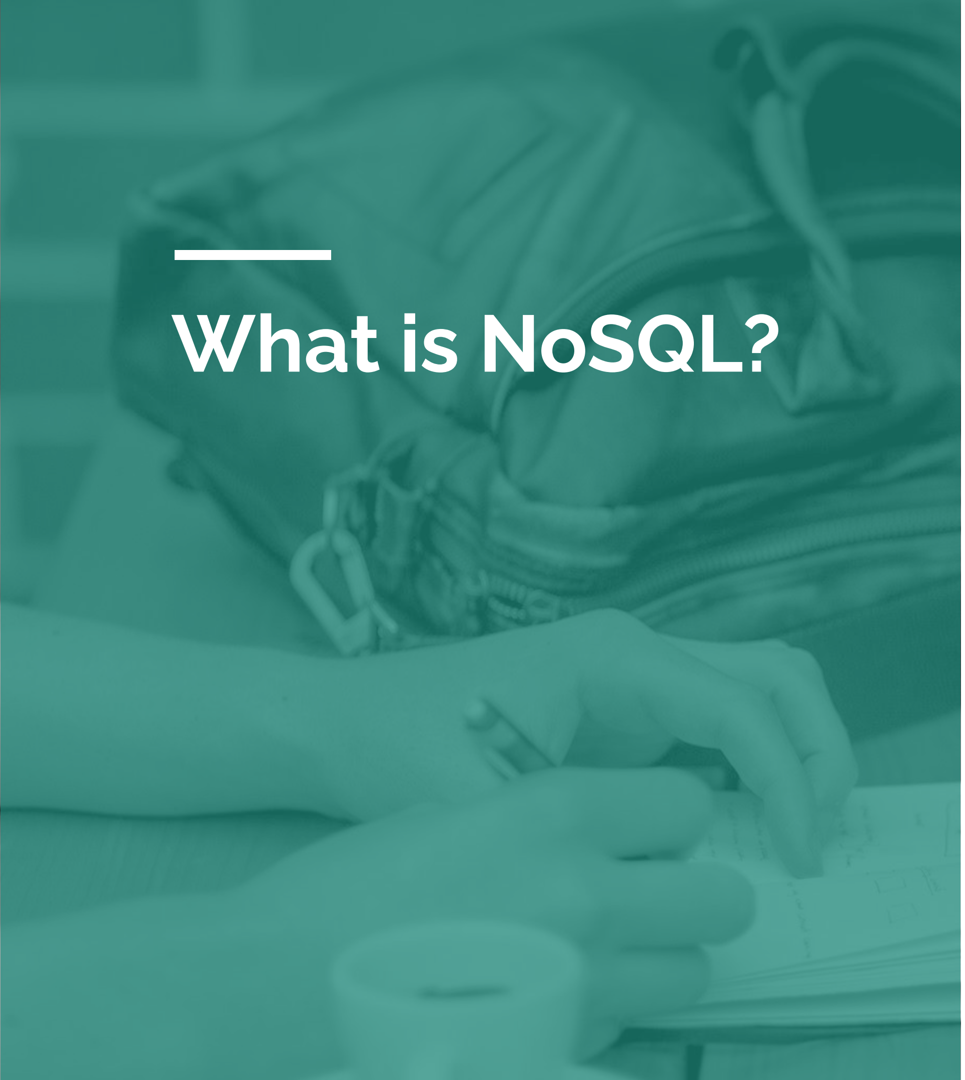
What is NoSQL?

What is NoSQL?



NoSQL stands for "Not Only SQL"

(but sometimes means "Non-SQL")

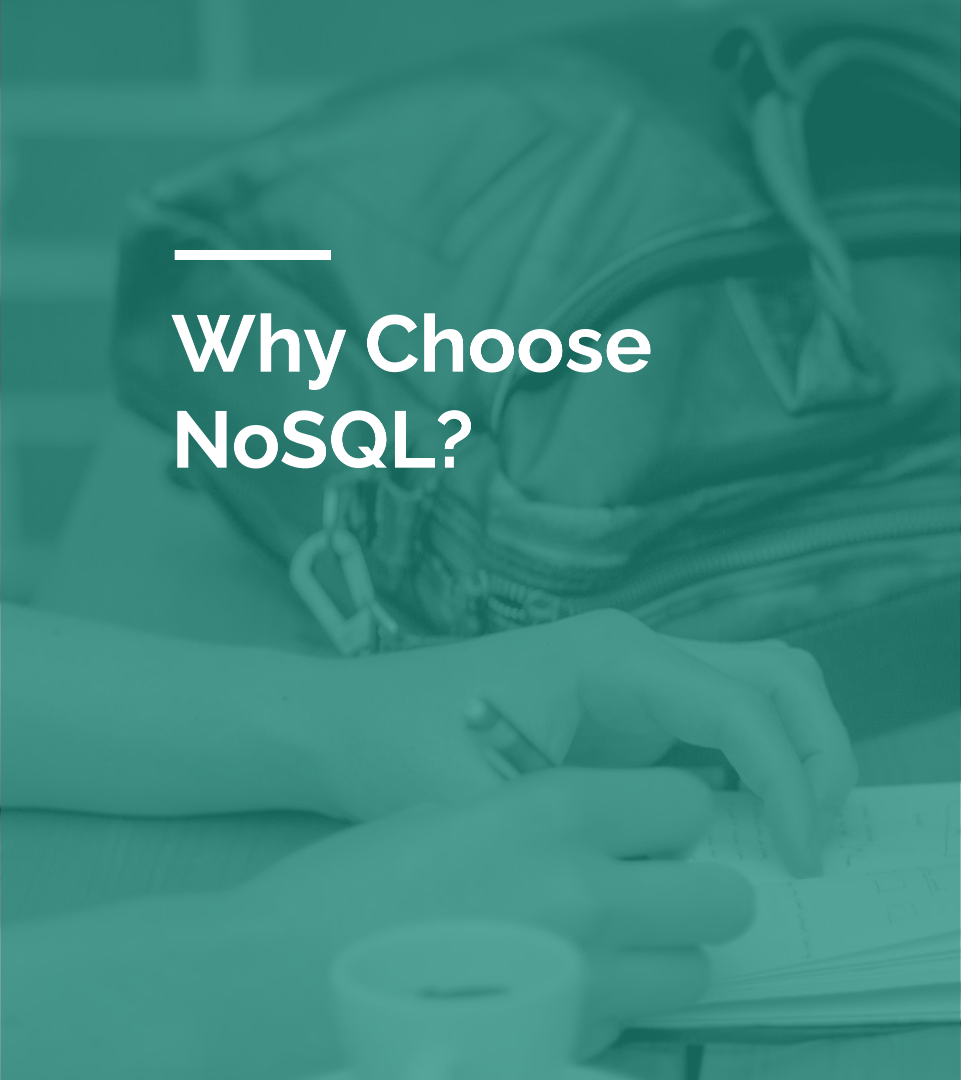


What is NoSQL?

NoSQL stands for "Not Only SQL"

(but sometimes means "Non-SQL")

An umbrella term for databases that don't use SQL as their primary query language



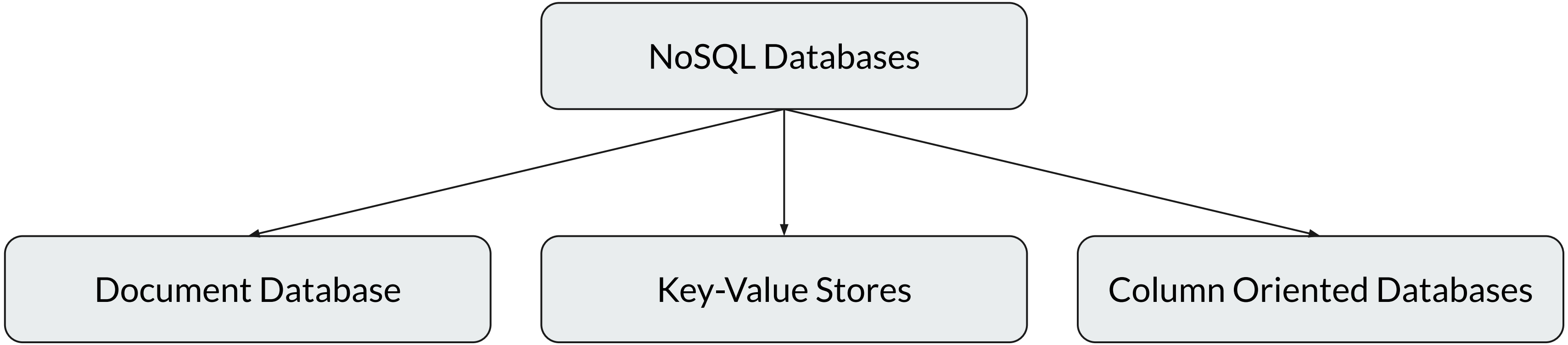
Why Choose NoSQL?

Why Choose NoSQL?

- Handling massive, semi-structured or unstructured data
- Flexibility for evolving data models.

Types of NoSQL Databases

Types of NoSQL Databases



NoSQL Databases

Document Database

Key-Value Stores

Column Oriented Databases

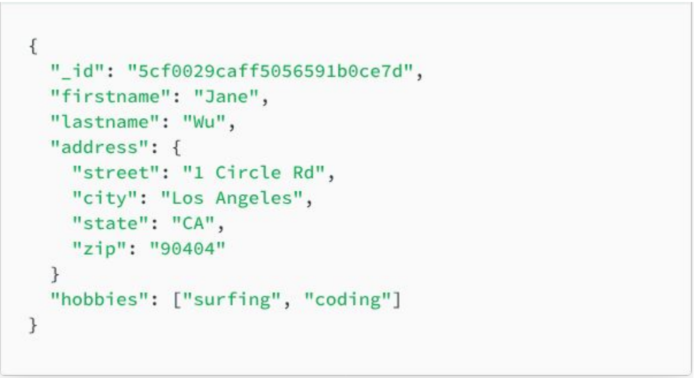
Document Databases

Document Databases: Flexibility for Complex Data

Document Databases

Document Databases: Flexibility for Complex Data

- Store data in documents (flexible structures like JSON)
- Documents can have nested structures (like profiles with addresses)



```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  }
  "hobbies": ["surfing", "coding"]
}
```

Document Databases

Document Databases: Flexibility for Complex Data

- Store data in documents (flexible structures like JSON)
- Documents can have nested structures (like profiles with addresses)
- Ideal for storing semi-structured or unstructured data

Key-Value Stores

Key-Value Stores: Fast Lookups for Simple Data

- Simplest NoSQL model: key-value pairs.
    - Keys are unique identifiers (like user IDs, product codes)
    - Values can be various data types (strings, numbers, JSON)
- Ideal for fast retrievals based on the key

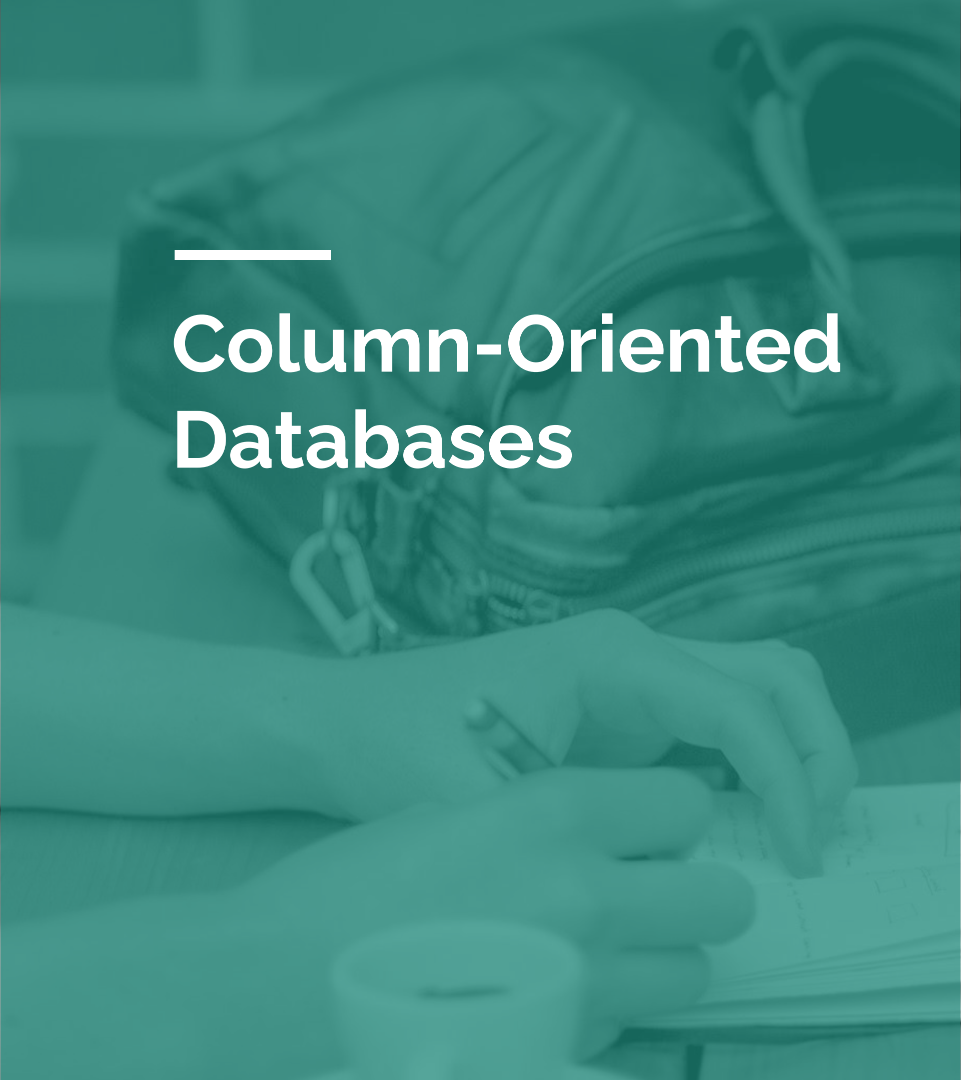
Key-Value Stores

Redis

- In-memory database known for lightning-fast performance.

- Used by Craiglist, **Instagram**, StackOverfow, flickr, ...

Instagram needed to keep around a **mapping** of about 300 million photos back to the **user ID** that created them.



Column-Oriented Databases

Column-Oriented Databases: Built for Speed and Analytics

- Store data by columns instead of rows (unlike relational databases)
- Optimized for reading specific columns – faster analytics
- Handle schema changes more efficiently (add new columns easily)

Information storage in Column-Oriented Storage

Product ID	Product Name	Price	Quantity
1	Widget Deluxe	24.99	50
2	Super Gadget	15.50	20
3	Shiny Thing		

But then how do I access all info for **product_id=1**?

Traditional Relational (Row-Oriented) Storage

- **Row 1:** 1, Widget Deluxe, 24.99, 50
- **Row 2:** 2, Super Gadget, 15.50, 20
- **Row 3:** 3, Shiny Thing, 9.99, 100

Column-Oriented Storage (like Bigtable)

- **Product ID Column:** 1, 2, 3
- **Product Name Column:** Widget Deluxe, Super Gadget, Shiny Thing
- **Price Column:** 24.99, 15.50, 9.99
- **Quantity Column:** 50, 20, 100

Information storage in Column-Oriented Storage

Product ID	Product Name	Price	Quantity
1	Widget Deluxe	24.99	50
2	Super Gadget	15.50	20
3	Shiny Thing	9.99	100

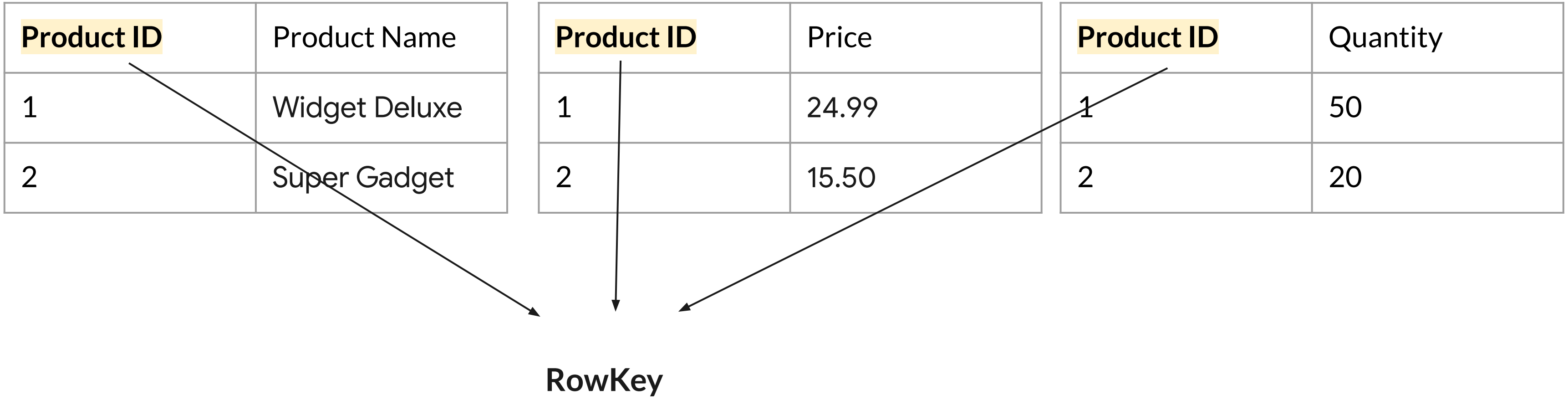
Product ID	Product Name
1	Widget Deluxe
2	Super Gadget

Product ID	Price
1	24.99
2	15.50

Product ID	Quantity
1	50
2	20

Replication

Information storage in Column-Oriented Storage



Product ID	Product Name
1	Widget Deluxe
2	Super Gadget

Product ID	Price
1	24.99
2	15.50

Product ID	Quantity
1	50
2	20

RowKey

Row keys offer a mechanism to uniquely identify and logically group data.

Column-Oriented Databases

Google's Bigtable

Google's Bigtable: A NoSQL Pioneer for Massive Data

- Column-oriented, distributed NoSQL database developed by Google.
- Designed to handle petabytes of data.
- Powers many Google services (Search, Maps, Gmail)

Column-Oriented Databases

Google's Bigtable

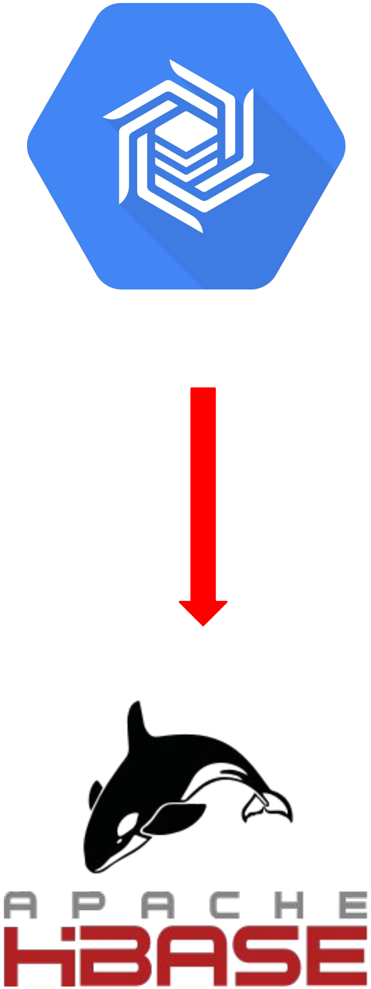
Google's Bigtable: A NoSQL Pioneer for Massive Data

- Column-oriented, distributed NoSQL database developed by Google.

Bigtable's success inspired creation of HBase!

The Rise of HBase

Bigtable's Influence: The Rise of HBase



Bigtable's Influence: The Rise of HBase

Google's Bigtable: A NoSQL Pioneer for Massive Data

- Google published a paper (2006) describing Bigtable's design and concepts.
- This sparked interest in column-oriented NoSQL solutions outside of Google.
- HBase emerged as an open-source implementation heavily inspired by Bigtable

Bigtable and HBase

A Comparative Look

Bigtable and HBase
A Comparative Look

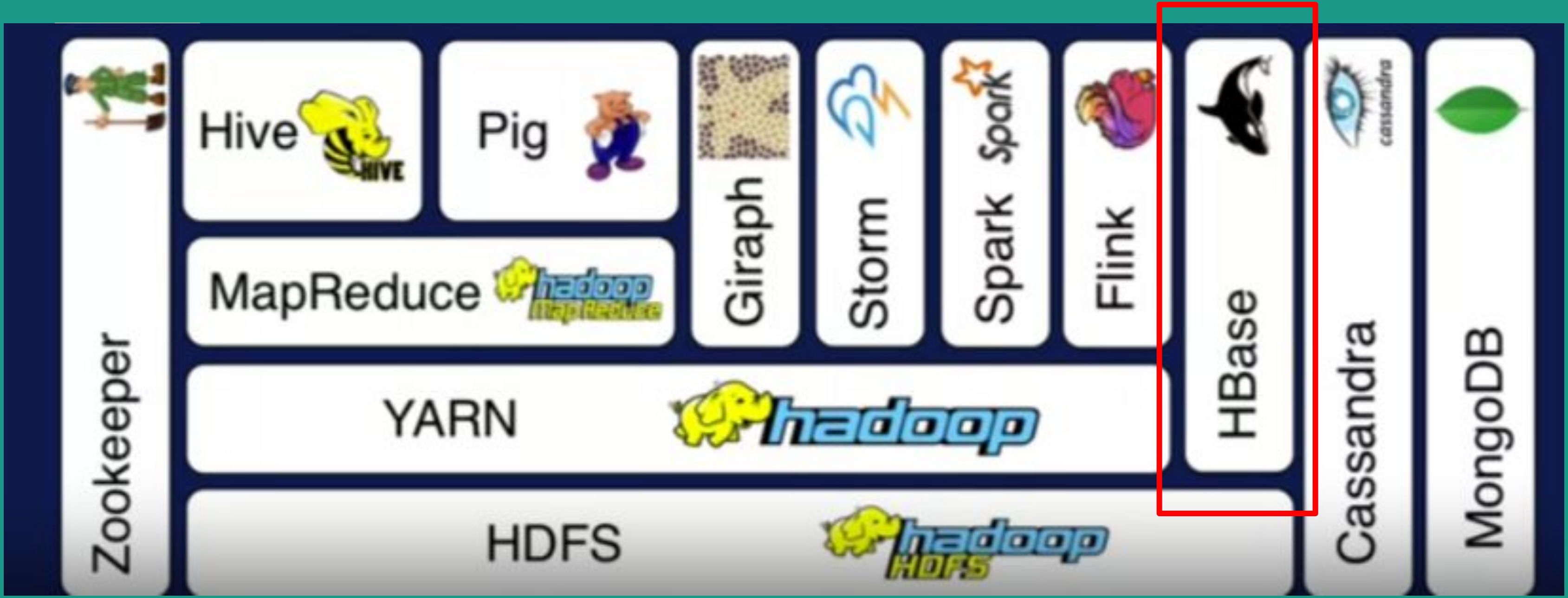
Google's Bigtable: A NoSQL Pioneer for Massive Data

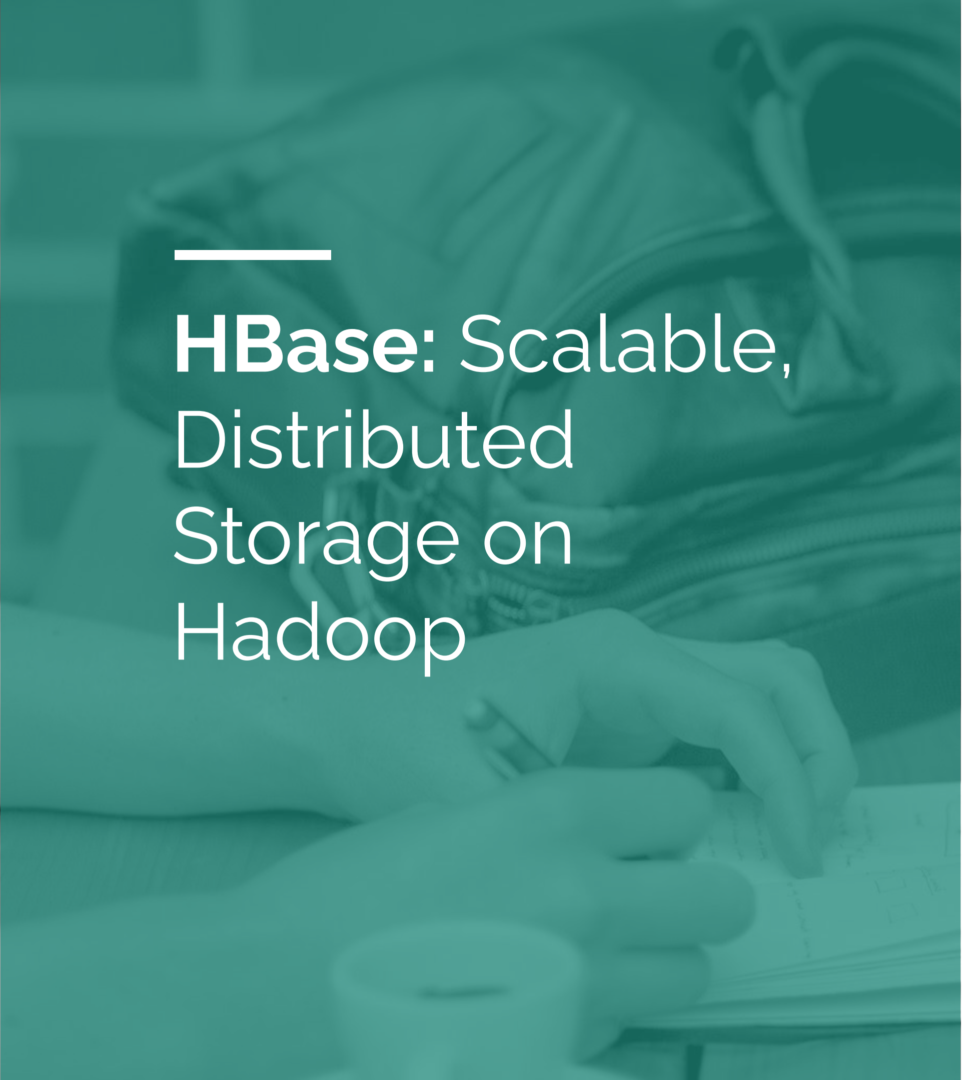
- Similarities:
  - Column-oriented data model
  - Sparse table structure
  - Built for scalability and high performance.

- Differences:
  - Bigtable runs on Google's infrastructure; HBase runs on Hadoop.
  - HBase has stronger consistency guarantees.
  - Bigtable's proprietary API vs. HBase's more open interfaces.

APACHE
HBASE

Welcome to the Zoo!





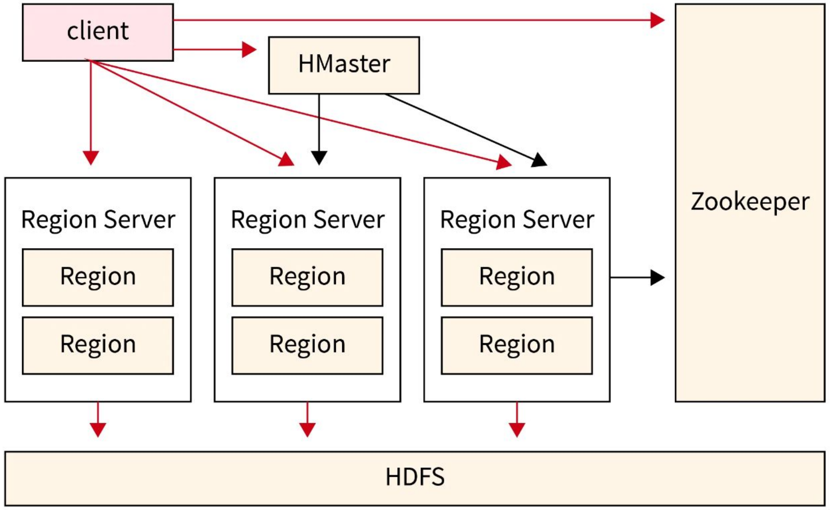
HBase: Scalable, Distributed Storage on Hadoop

HBase: Scalable, Distributed Storage on Hadoop

- Open-source, column-oriented NoSQL database.
- Part of the Apache Hadoop ecosystem, built on top of HDFS.
- Designed for large-scale, structured and semi-structured data.
- Real-time read/write access.

HBase Architecture

HBase Architecture



Apache HBase Architecture

HMaster

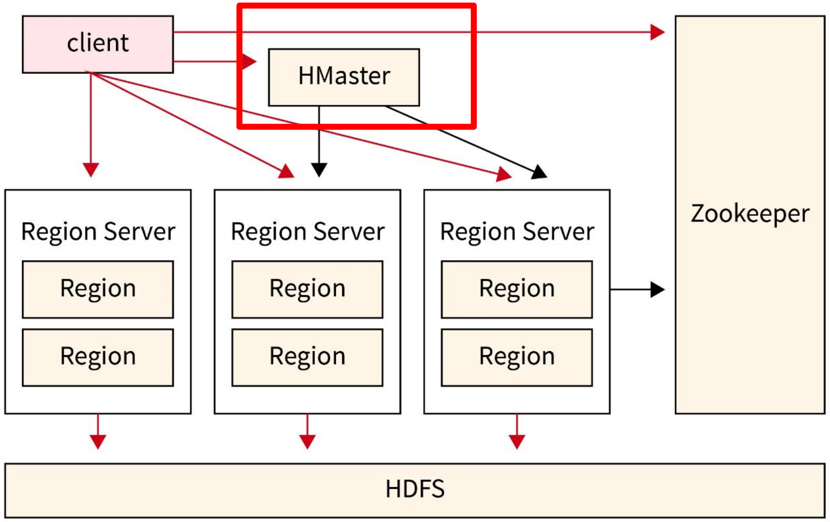
- Manages the cluster's RegionServers.

- Assigns Regions (chunks of data) to RegionServers.

- Responsible for load balancing across the cluster.

- Monitors RegionServer health and handles failures.

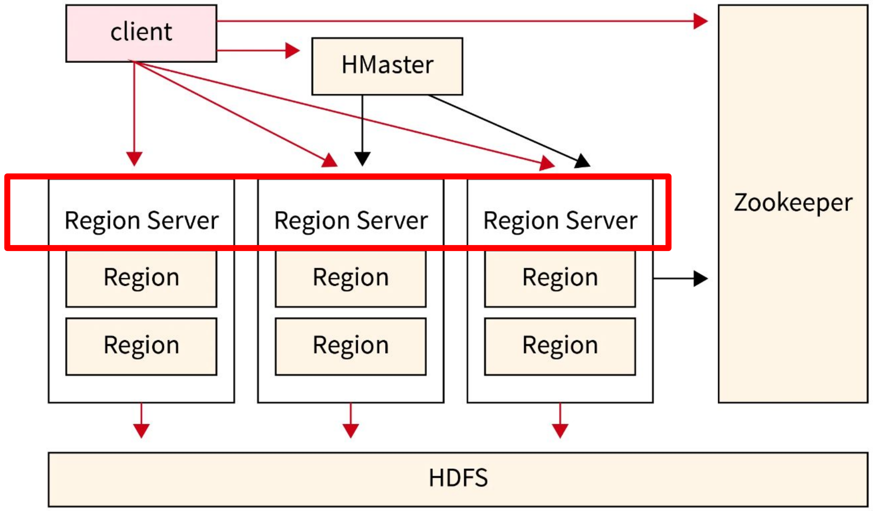
- Handles metadata operations (table creation, etc.)



Apache HBase Architecture

RegionServers

- Serve data for read and write requests.
- Manage Regions, which are segments of an HBase table.
- Split Regions that grow too large.
- Handle data compactions for storage efficiency.

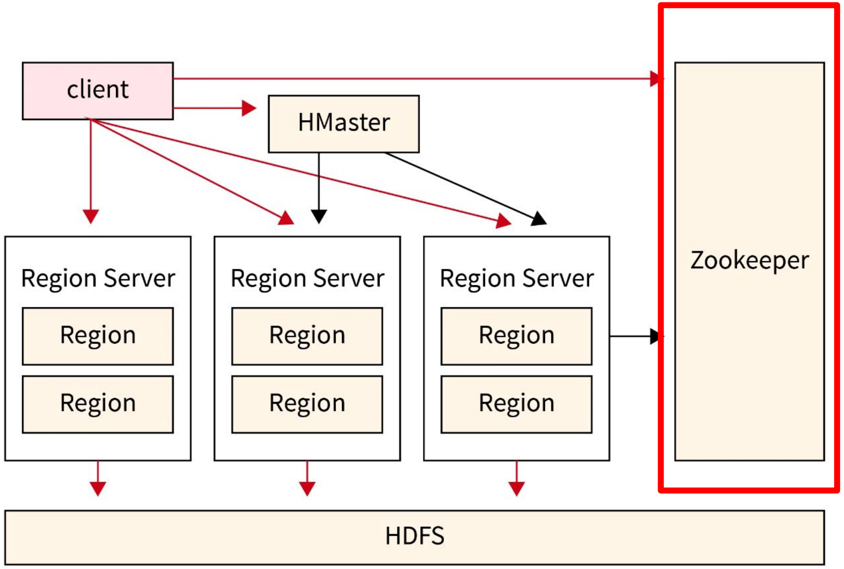


Apache HBase Architecture

Zookeeper

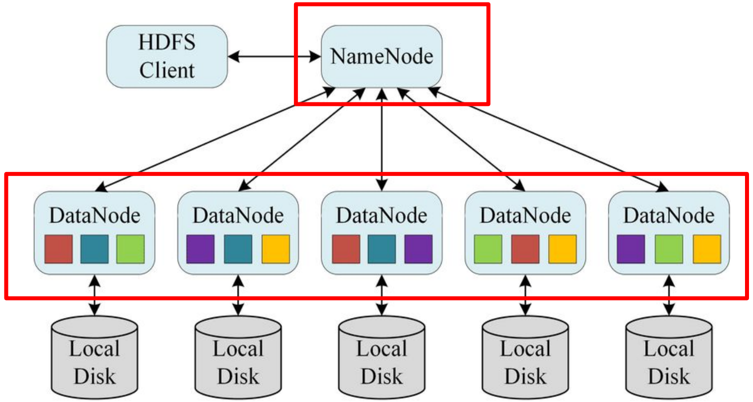
- A coordination service essential for distributed systems.

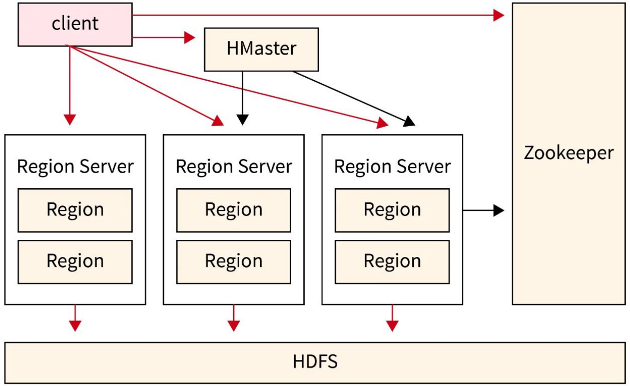
Have we seen a similar Master-Slave storage setup before?



Apache HBase Architecture

Recall …

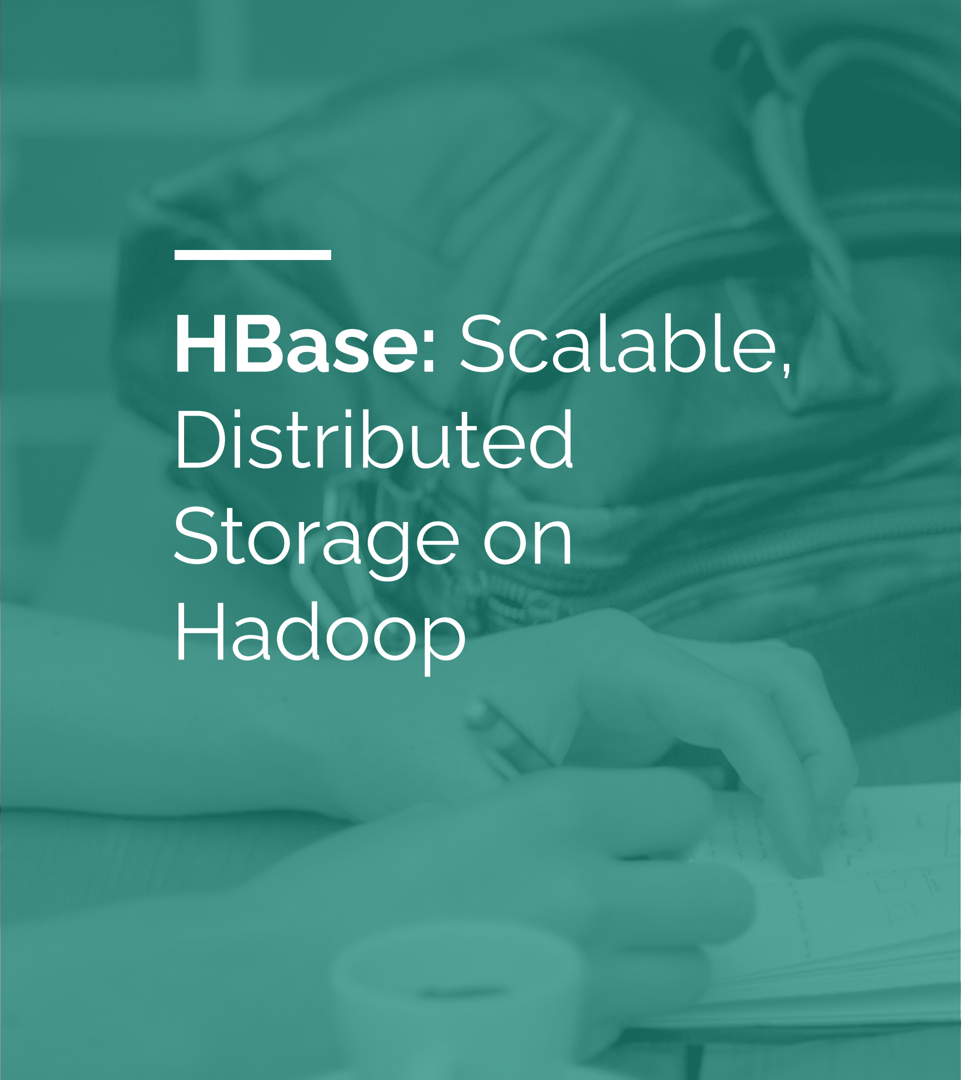




Apache HBase Architecture

HBase vs HDFS?

- **NameNode**: A **librarian** keeping track of which books are on which shelves in the library.
- **HMaster**: The manager of a specific section of the library (e.g., the fiction section), deciding where to put new books, rearranging shelves, and handling requests from readers within that section.



HBase: Scalable, Distributed Storage on Hadoop

HBase vs HDFS?

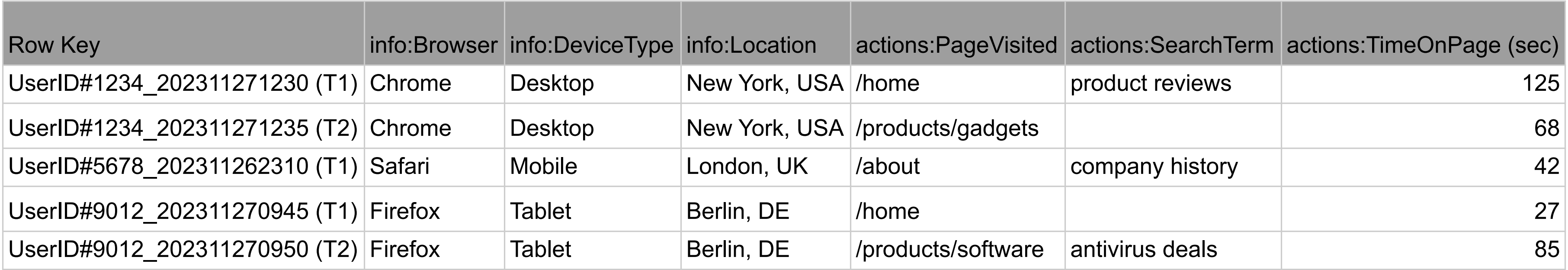
- Layer of Operation:
  - **HMaster** operates at the level of HBase, managing its specific data structures and requests.
  - **NameNode** operates at the level of HDFS, responsible for the file system itself.

- NameNode primarily handles metadata (data about data).
- HMaster manages metadata about HBase tables

- HBase depends on HDFS for underlying storage.

HBase Data Model

Sample Database



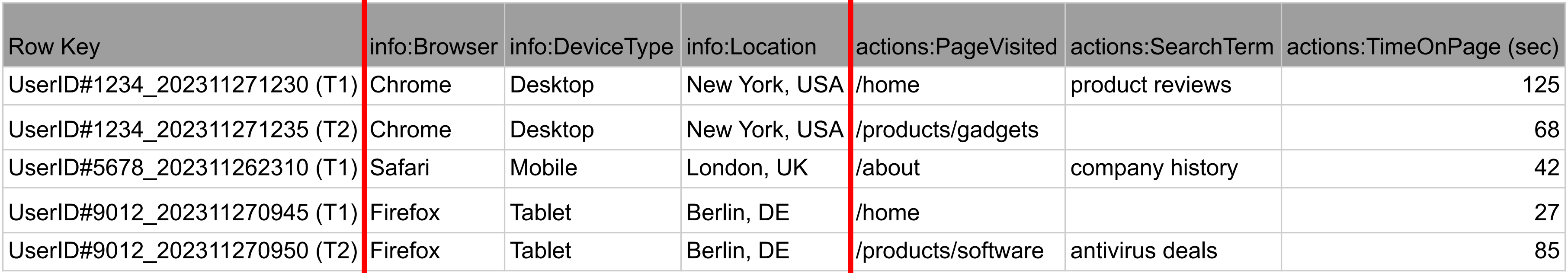
Row Key	info:Browser	info:DeviceType	info:Location	actions:PageVisited	actions:SearchTerm	actions:TimeOnPage (sec)
UserID#1234_202311271230 (T1)	Chrome	Desktop	New York, USA	/home	product reviews	125
UserID#1234_202311271235 (T2)	Chrome	Desktop	New York, USA	/products/gadgets		68
UserID#5678_202311262310 (T1)	Safari	Mobile	London, UK	/about	company history	42
UserID#9012_202311270945 (T1)	Firefox	Tablet	Berlin, DE	/home		27
UserID#9012_202311270950 (T2)	Firefox	Tablet	Berlin, DE	/products/software	antivirus deals	85

Sample Database

Row Key	info:Browser	info:DeviceType	info:Location	actions:PageVisited	actions:SearchTerm	actions:TimeOnPage (sec)
UserID#1234_202311271230 (T1)	Chrome	Desktop	New York, USA	/home	product reviews	125
UserID#1234_202311271235 (T2)	Chrome	Desktop	New York, USA	/products/gadgets		68
UserID#5678_202311262310 (T1)	Safari	Mobile	London, UK	/about	company history	42
UserID#9012_202311270945 (T1)	Firefox	Tablet	Berlin, DE	/home		27
UserID#9012_202311270950 (T2)	Firefox	Tablet	Berlin, DE	/products/software	antivirus deals	85

RowKey: Unique identifier for each row.

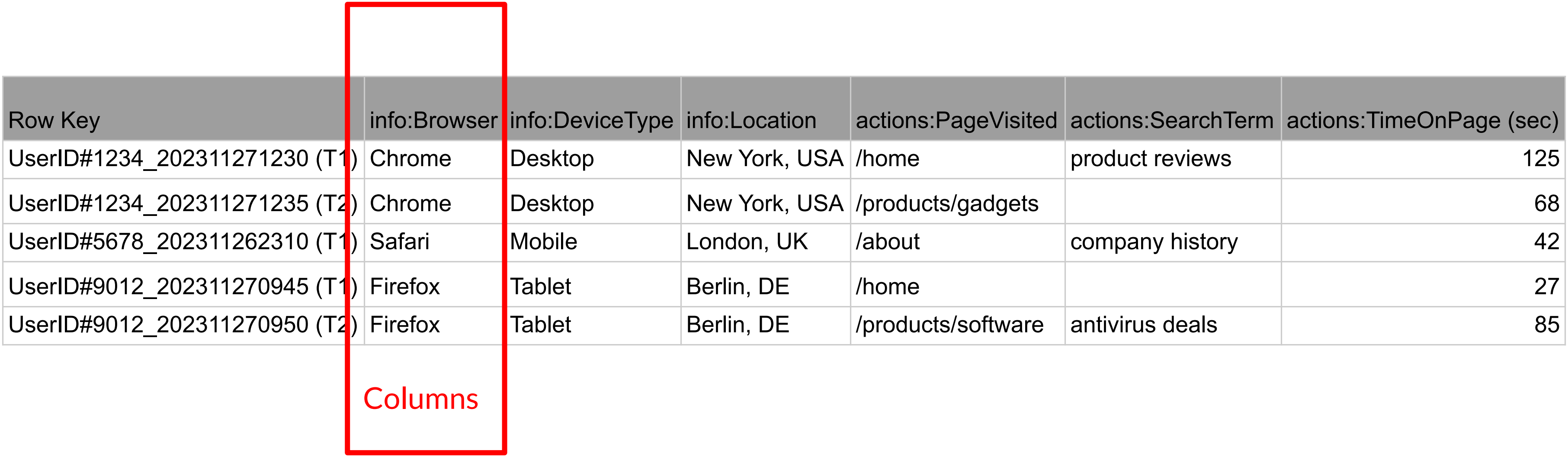
Sample Database



Row Key	info:Browser	info:DeviceType	info:Location	actions:PageVisited	actions:SearchTerm	actions:TimeOnPage (sec)
UserID#1234_202311271230 (T1)	Chrome	Desktop	New York, USA	/home	product reviews	125
UserID#1234_202311271235 (T2)	Chrome	Desktop	New York, USA	/products/gadgets		68
UserID#5678_202311262310 (T1)	Safari	Mobile	London, UK	/about	company history	42
UserID#9012_202311270945 (T1)	Firefox	Tablet	Berlin, DE	/home		27
UserID#9012_202311270950 (T2)	Firefox	Tablet	Berlin, DE	/products/software	antivirus deals	85

Column Families: Groups of related columns.

Sample Database



Row Key	info:Browser	info:DeviceType	info:Location	actions:PageVisited	actions:SearchTerm	actions:TimeOnPage (sec)
UserID#1234_202311271230 (T1)	Chrome	Desktop	New York, USA	/home	product reviews	125
UserID#1234_202311271235 (T2)	Chrome	Desktop	New York, USA	/products/gadgets		68
UserID#5678_202311262310 (T1)	Safari	Mobile	London, UK	/about	company history	42
UserID#9012_202311270945 (T1)	Firefox	Tablet	Berlin, DE	/home		27
UserID#9012_202311270950 (T2)	Firefox	Tablet	Berlin, DE	/products/software	antivirus deals	85

Columns

Sample Database

Row Key	info:Browser	info:DeviceType	info:Location	actions:PageVisited	actions:SearchTerm	actions:TimeOnPage (sec)
UserID#1234_202311271230 (T1)	Chrome	Desktop	New York, USA	/home	product reviews	125
UserID#1234_202311271235 (T2)	Chrome	Desktop	New York, USA	/products/gadgets		68
UserID#5678_202311262310 (T1)	Safari	Mobile	London, UK	/about	company history	42
UserID#9012_202311270945 (T1)	Firefox	Tablet	Berlin, DE	/home		27
UserID#9012_202311270950 (T2)	Firefox	Tablet	Berlin, DE	/products/software	antivirus deals	85

Timestamp:
To keep track
of changes

Sample Database

Row Key	info:Browser	info:D	ctions:TimeOnPage (sec)
UserID#1234_202311271230 (T1)	Chrome	Desk	125
UserID#1234_202311271235 (T2)	Chrome	Desk	68
UserID#5678_202311262310 (T1)	Safari	Mobi	42
UserID#9012_202311270945 (T1)	Firefox	Table	27
UserID#9012_202311270950 (T2)	Firefox	Table	85

RowKey: Unique identifier for each row.

RowKey = UserID + Timestamp

RowKey Design

The Importance of Row Key Design

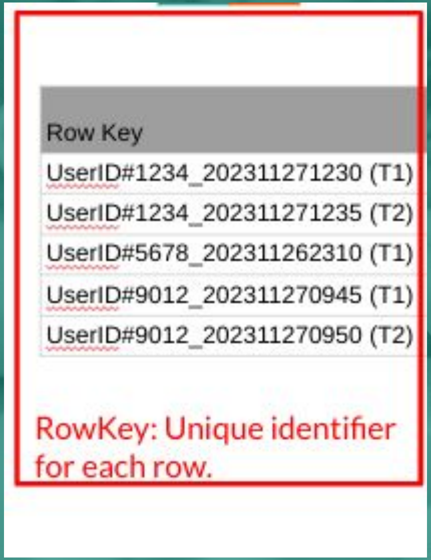
- **Distribution**: Row keys determine how data is spread across the cluster.
- **Avoid hotspots**: Poor row key design can lead to concentrated load on certain machines.
- **Consider access patterns**: How will you query your data?

RowKey Design

The Importance of Row Key Design

- HBase stores data **sorted by Row Key**. This enables:
    - Fast Lookups: If you know the Row Key, HBase can quickly navigate to the correct data location.
    - Range Scans: Efficiently retrieves sets of rows within a key range.

RowKey Design



Row Key
UserID#1234_202311271230 (T1)
UserID#1234_202311271235 (T2)
UserID#5678_202311262310 (T1)
UserID#9012_202311270945 (T1)
UserID#9012_202311270950 (T2)

RowKey: Unique identifier for each row.

The Importance of Row Key Design

- **Consider access patterns**: How will you query your data?

Think about how you'll be querying your data. Will you often fetch by:

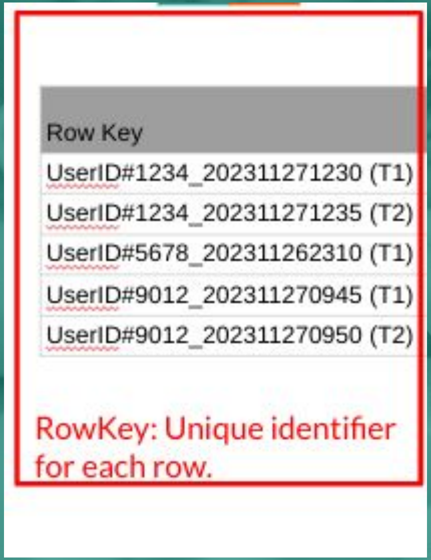
- UserID?
- Date or Time Range?
- Location?
- Some other attribute? Your Row Key should make those common queries efficient.

RowKey Design

The Importance of Row Key Design

- **Distribution**: Row keys determine how data is spread across the cluster.
- **Avoid hotspots**: Poor row key design can lead to **concentrated load** on certain machines.
- **Consider access patterns**: How will you query your data?

RowKey Design



Row Key
UserID#1234_202311271230 (T1)
UserID#1234_202311271235 (T2)
UserID#5678_202311262310 (T1)
UserID#9012_202311270945 (T1)
UserID#9012_202311270950 (T2)

RowKey: Unique identifier for each row.

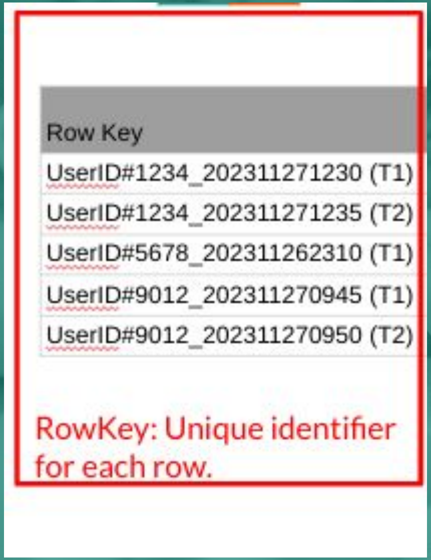
The Importance of Row Key Design

- **Avoid hotspots**: Poor row key design can lead to **concentrated load** on certain machines.

Why add timestamp to the rowkey?

If our Row Key was simply the UserID?

RowKey Design



Row Key
UserID#1234_202311271230 (T1)
UserID#1234_202311271235 (T2)
UserID#5678_202311262310 (T1)
UserID#9012_202311270945 (T1)
UserID#9012_202311270950 (T2)

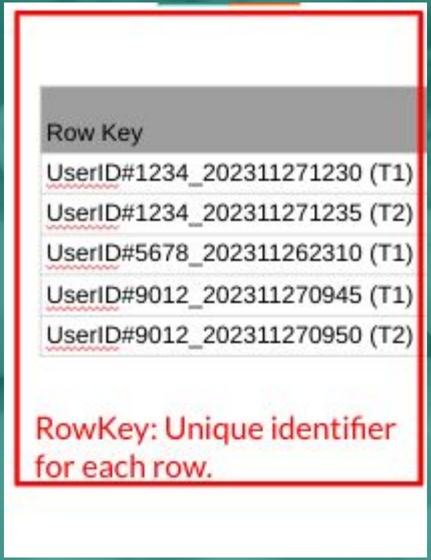
RowKey: Unique identifier for each row.

The Importance of Row Key Design

If our Row Key was simply the UserID?

- **Hotspots:** Popular users cause load imbalances.
- **Sequential Nature:** New user data concentrates on a few machines.
- **Limited Querying:** Difficult to analyze data based on time ranges.

RowKey Design



Row Key
UserID#1234_202311271230 (T1)
UserID#1234_202311271235 (T2)
UserID#5678_202311262310 (T1)
UserID#9012_202311270945 (T1)
UserID#9012_202311270950 (T2)

RowKey: Unique identifier for each row.

The Importance of Row Key Design

How does adding timestamp to UserID help?

- **Distributes Recent Data:** Actions from the same user are spread out.
- **Enables Time-Based Queries:** Fetch data within time ranges and analyze temporal trends.

Common HBase Commands

Creating and Manipulating Tables

- **create 'table_name', 'col_family1', 'col_family2'** ... (Creates a new table)
- **list** (Shows all tables)
- **describe 'table_name'** (Displays table's schema)
- **disable 'table_name'** (Disables a table)
- **drop 'table_name'** (Deletes a table)

Working with Data

- **put 'table_name', 'row_key', 'col_family:column', 'value'** (Inserts data)

- **get 'table_name', 'row_key'** (Retrieves data by row key)

- **scan 'table_name'** (Scans the contents of a table – use with caution on large tables)

- **delete 'table_name', 'row_key'** – (Deletes a row)