UWE Bristol | University of the West of England

Presented by

Dr. Trupti Padiya

School of Computing and Creative Technologies

Advanced Databases UFCFU3-15-3

Introduction - NoSQL Databases

Slides with modification adapted from Dr. Elias Pimenidis

NoSQL databases

- NoSQL (Not Only SQL) or non-SQL databases refers to non relational databases. NoSQL databases store and manipulate data in other formats/data models than tabular relations/relational model. NoSQL databases do not require a fixed schema.
- Designed for horizontal scalability
- High availability
- Eventual consistency
- Support different data models. Some examples include key-value store, document store, graph databases etc.

Why NoSQL

- Massive data volumes
  - Massively distributed architecture required to store the data
  - Google, Amazon, Yahoo, Facebook – 10-100K servers
- Flexibility
  - Data model
  - Schema free design – easy to process structured and semi structured data e.g. JSON, XML
  - Schema evolution
- Performance
  - Extreme Query workload – complex query joins in RDBMS could be inefficient at scale

NoSQL Pros and Cons

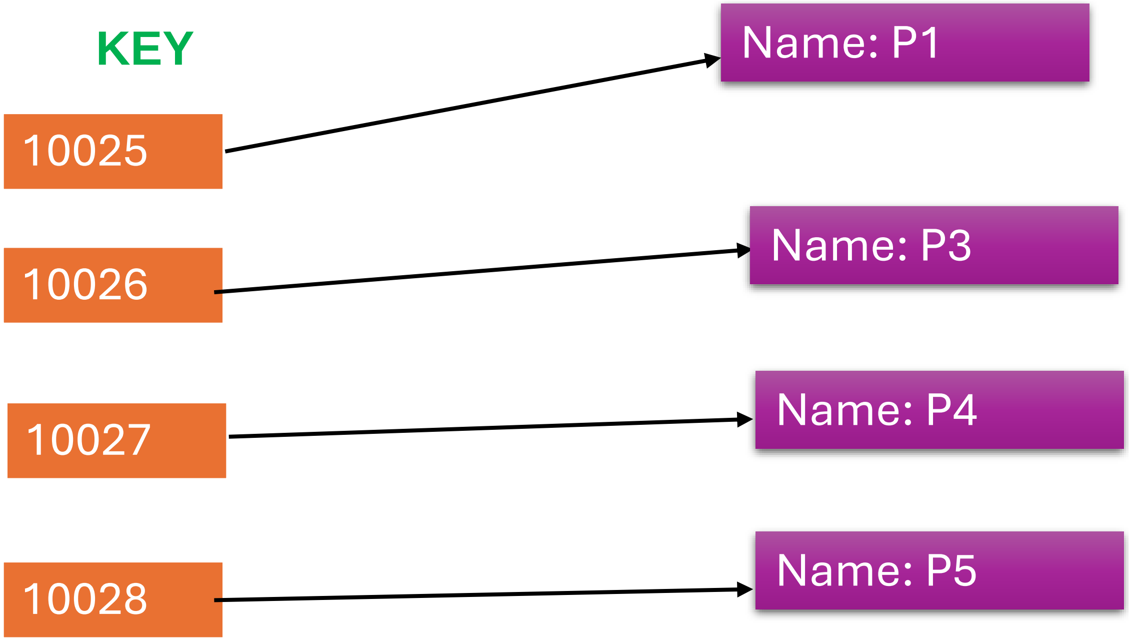
- **Advantages**
  - Massive scalability
  - High availability
  - Lower cost (than competitive solutions at that scale)
  - (usually) predictable elasticity
  - Schema flexibility, sparse & semi-structured data
- **Disadvantages**
  - Limited query capabilities (so far)
  - Eventual consistency is not intuitive to program for
  - No standardization
  - Portability might be an issue
  - Insufficient access control

NoSQL Categorization

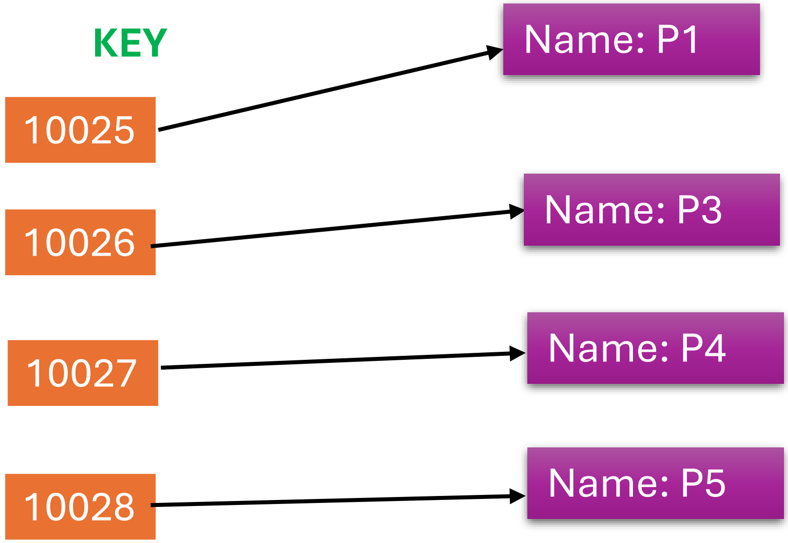
- Key-Value Stores
- Column Stores
- Document Stores
- Graph databases

Key-Value Stores

- Simple Data model –Key value pairs
- Key-value based database stores data as (key, value) pairs
  - Keys are unique
  - Hash map, hash table or dictionary



Key-Value Stores



- Keys are hashed by means of a so-called **hash function**
  - A hash function takes an arbitrary value of arbitrary size and maps it to a key with a fixed size, which is called the hash value.
  - Each hash can be mapped to a space in computer memory

Hash map/Hash tables

Key
10025
10026
10027
10028

Hash Function

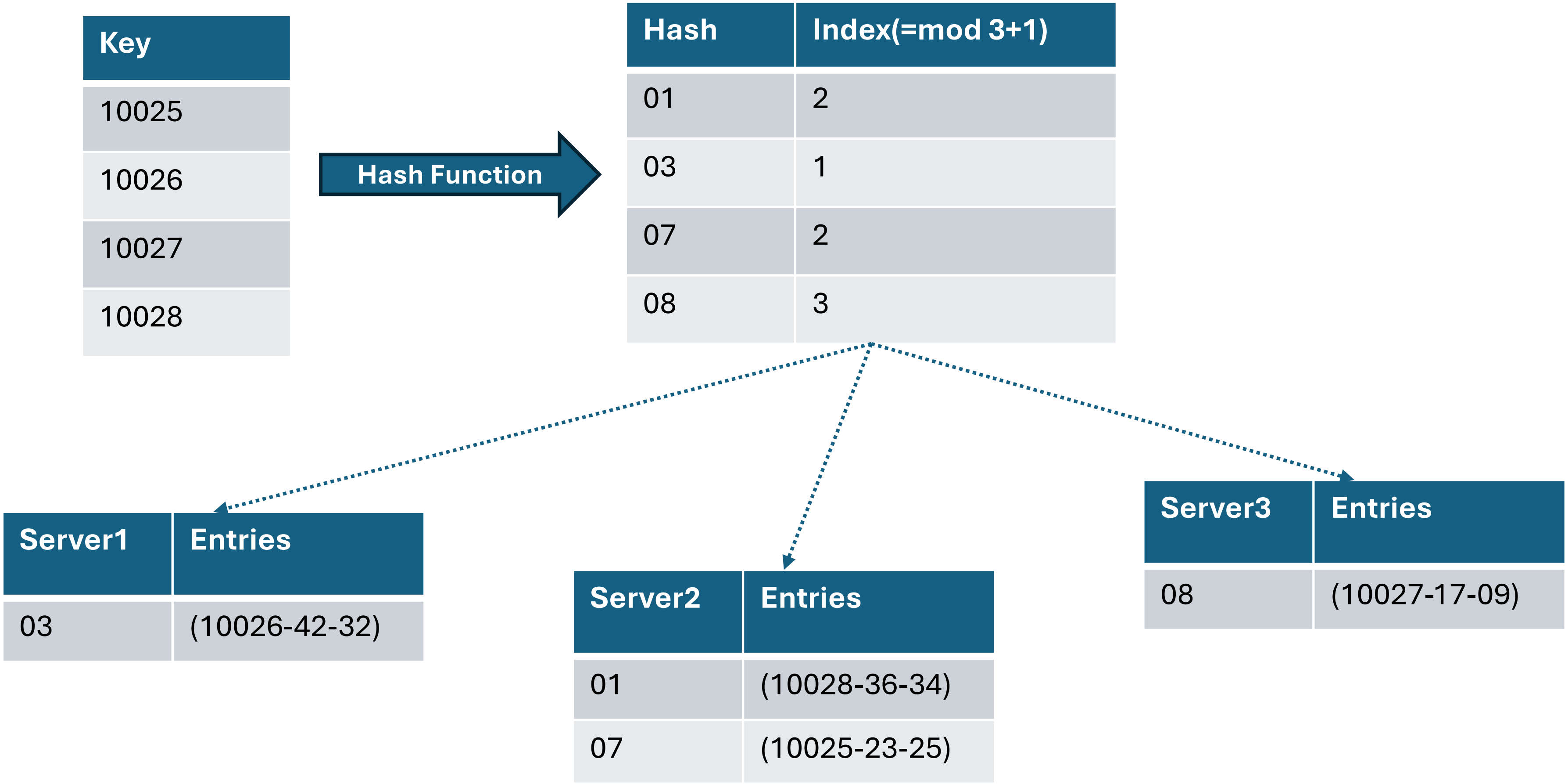
Hash	Key
01	(10028-36-34)
03	(10026-42-32)
07	(10025-23-25)
08	(10027-17-09)

Key-Value Store Sharding

- NoSQL databases are built with horizontal scalability support in mind
- Distribute hash table over different locations
- Assume we need to spread our hashes over three servers
  - Hash every key to a server identifier
  - index(hash) = mod(hash, nrServers) + 1

- Note: A database shard, or simply a shard, is a horizontal partition of data

Key-Value Store Sharding



Key
10025
10026
10027
10028

Hash	Index(=mod 3+1)
01	2
03	1
07	2
08	3

Server1	Entries
03	(10026-42-32)

Server2	Entries
01	(10028-36-34)
07	(10025-23-25)

Server3	Entries
08	(10027-17-09)

a mod b = a − (Int [a / b] x b) , where Int is an integer part of the value

Key-Value Stores Pros and Cons

- **Pros:**
  - simple model
  - very fast
  - very scalable
  - able to distribute horizontally
- **Cons:**
  - many data structures (objects)
  - cannot do complex queries
  - key value pairs

Tuple Stores

- A **tuple store** is similar to a key-value store, with the difference that it does not store pairwise combinations of a key and a value, but instead stores a unique key together with a vector of data
- Example:
  - marc -> ("Marc", "McLast Name", 25, "Germany")
- Some Tuple Stores supports Schema-less design - No requirement to have the same length or semantic ordering
- Various NoSQL implementations do, however, permit organizing entries in semantical groups (aka collections or tables)

Tuple Stores Pros and Cons

- Pros
  - Efficient querying
  - Support for Complex data structure
  - Schema Flexibility
  - Optimized for OLTP
- Cons
  - Performance issues with complex joins
  - Generally not considered good for OLAP
  - Write based operations

Column Stores

- A **column store** is a database management system that stores data tables as sections of columns of data
- Useful if
  - aggregates are regularly computed over large numbers of similar data items
  - data is sparse, i.e. columns with many null values

Column Stores

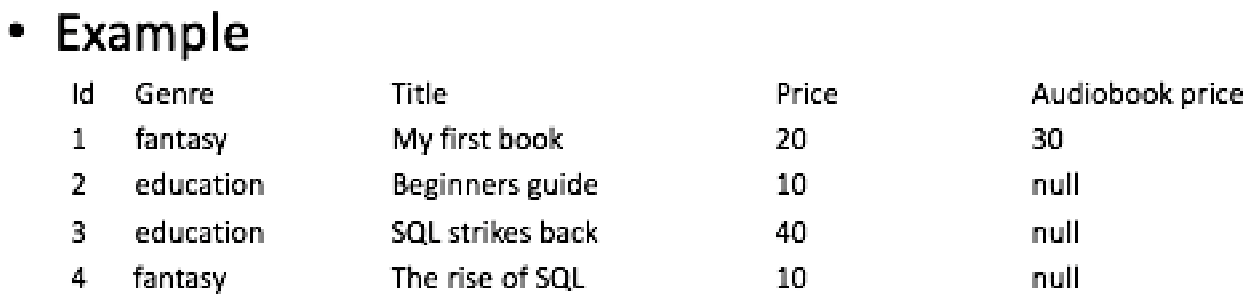
- Row based databases are not efficient at performing operations that apply to the entire data set
  - Need indexes which add overhead
- In a column-oriented database, all values of a column are placed together on disk

  Genre: fantasy: 1, 4education: 2, 3

  Title: My first book:1 Beginners guide:2. SQL strikes back:3 The rise of SQL:4

  Price: 20:1 10:2,4 40:3

  Audiobook price: 30:1
- A column matches the structure of a normal index in a row-based system
- Operations such as: find all records with price equal to 10 can now be executed directly
- Null values do not take up storage space anymore



- Example

Id	Genre	Title	Price	Audiobook price
1	fantasy	My first book	20	30
2	education	Beginners guide	10	null
3	education	SQL strikes back	40	null
4	fantasy	The rise of SQL	10	null

Column Stores

- Pros
  - Schema Flexibility
  - Analytical queries
  - Data Compression
  - Read based operations
- Cons
  - Write based operation
  - Generally not considered good for OLTP
  - Single row queries
  - Join operations could be slowed down

Document Stores

- **Document stores** store a collection of attributes that are labeled and unordered, representing items that are semi-structured
- Example:

```
{
  Title      = "Harry Potter"
  ISBN       = "111-1111111111"
  Authors    = ["J.K.  Rowling"]
  Price      = 32
  Dimensions = "8.5 x 11.0 x 0.5"
  PageCount  = 234
  Genre      = "Fantasy"
}
```

Document Stores

- Most modern NoSQL databases choose to represent documents using JSON

```
{
    "title": "Harry Potter",
    "authors": ["J.K.  Rowling", "R.J.  Kowling"],
    "price": 32.00,
    "genres": ["fantasy"],
    "dimensions": {
          "width": 8.5,
          "height": 11.0,
          "depth": 0.5
    },
    "pages": 234,
    "in_publication": true,
    "subtitle": null
 }
```

Documents with keys

- Most NoSQL document stores will allow you to store documents in tables (collections) in a schema-less manner, but will enforce that a primary key be specified
  - e.g. Amazon’s DynamoDB, MongoDB (_id)
- Primary key will be used as a partitioning key to create a hash and determine where the data will be stored

Document stores and complex queries

- Document stores do not support relations
- **First approach**: embedded documents

```
{
        "title": "Databases for Beginners",
        "authors": ["J.K.  Sequel", "John Smith"],
        "pages": 234
}
{
        "title": "Databases for Beginners",
        "authors": [
 {"first_name": "Jay Kay", "last_name": "Sequel", "age": 54},
 {"first_name": "John",   "last_name": "Smith",   "age": 32}
],
        "pages": 234
}
```

But: Data duplication!

Document stores and complex queries

- **Second approach**: create two collections

book collection:

```
{
   "title": "Databases for Beginners",
   "authors": ["Jay Kay Sequel", "John Smith"],
   "pages": 234
}
```

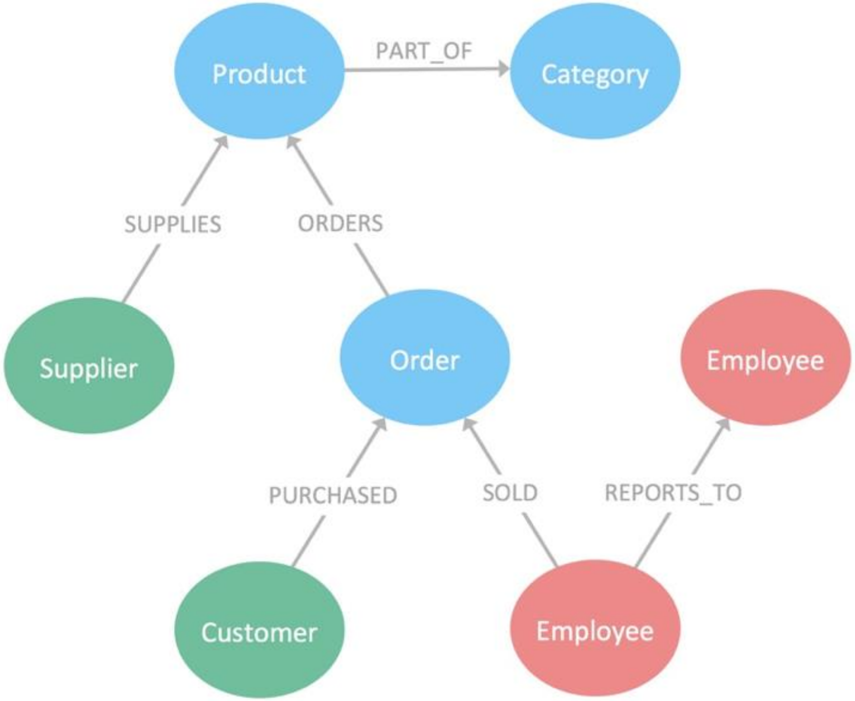
authors collection:

```
{
   "_id": "Jay Kay Sequel",
   "age": 54
}
```

But: Need to resolve complex relational queries in application code!

Graph databases

- **Graph databases** apply graph theory to the storage and retrieval of data
- Graphs consist of **nodes** and **edges**



Some (extra) resources/links:

- https://www.mongodb.com/resources/basics/databases/document-databases
- https://www.couchbase.com/blog/columnar-store-vs-row-store/
- https://neo4j.com/docs/getting-started/graph-database/
- https://www.mongodb.com/resources/basics/databases/key-value-database#:~:text=Key%20value%20databases%2C%20also%20known,associated%20value%20with%20each%20key.