Database Systems:

Module 13, Lecture 1 – NoSQL Solutions

Instructor: Alan Paradise

NoSQL

LESSON OBJECTIVES

- Describe the overall common characteristics of NoSQL Database software solutions
- Demonstrate understanding of the "aggregate" data model
- Describe and differentiate the four most common NoSQL system architectures

NoSQL

Recap

- Relational DBMS software has worked very well for many decades
- Companies have invested lots of money in software built upon relational DBMS infrastructure
- Companies have invested in staff/talent skilled in RDBMS technologies

BUT

- RDBMS systems struggle to adequately scale to support Big Data's volume, variety, and velocity demands
- Big Data is exploding faster than RDBMS technologies can handle

NoSQL

The database software marketplace is shifting toward NoSQL ("Not Only SQL")

NoSQL systems

- Use clustering
- Distribute the Data via replication and sharding
- Distribute Compute Processing
- Scale Horizontally
    - Simply add nodes for expansion
- Use Replication to provide
    - Redundancy: multiple copies of the data spread across multiple nodes
    - High availability: node failure is expected
    - Parallelization: faster query throughput

NoSQL

NoSQL (“Not Only SQL”)

- Requires Less Structure
- Does not store data in tables requiring rigid row/column structures
- Uses the Aggregate model (data is very *de*-normalized)
- Restricts join capabilities
- Relaxes ACID transaction compliance
- Uses a query language other than SQL
- Often open source, very low-cost software acquisition

NoSQL

Relational

- Schema defines rigid structure
  - Tables, Rows, Columns
- Foreign Key relationships
  - Which support joins
- Uses SQL
- Maintains ACID compliance
- Normalized: store a value only once
- Clustering available,
  but challenging
- Leading DBMS solutions can be quite expensive

NoSQL

- Stores related values in aggregates
- Flexible structure:
  - Ranges from none to some
- No joins
- Uses non-SQL query language
- Relaxes ACID compliance
- De-normalized
- Designed to support clustering
- Almost all major software options are open source and low-cost

NoSQL “aggregate” Data Model

RDBMS requires Tables, Rows, Columns as data stores

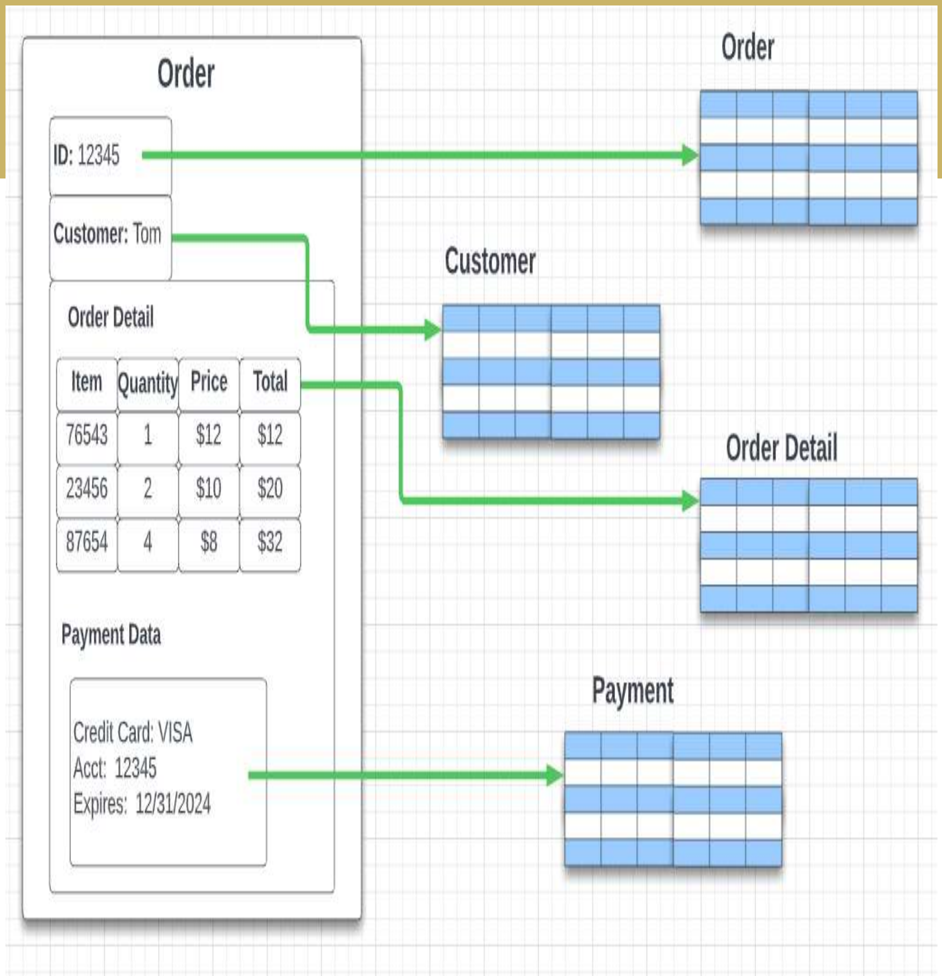
- Columns have restricted “domains”
- Third normal form – no multi-values, “store it once”

NoSQL systems use AGGREGATES as data stores

- Data values are grouped together as users need them
- Think of an unnormalized document, or an “object”
- An aggregate contains related values that are retrieved and manipulated together

NoSQL

Aggregate



NoSQL

Aggregates are conceptually the opposite of 3rd Normal Form

Why aggregates?

- It is difficult to spread a relational model across nodes in a cluster
- Replication and sharding introduce big challenges in data consistency
- Each query should minimize the number of nodes being accessed across the cluster
- Data values that are accessed together should live on the same node

NoSQL

Four NoSQL Data Schema/Models

- Document Store (using XML or JSON format)
- Graph (using node/edge structures with properties)
- Key-Value pairs
- Wide Columnar store (rows with dynamic columns holding key-value pairs)

NoSQL

Document Store
- MongoDB

Graph
- Neo4j

Key-Value Pairs
- Amazon Dynamo

Wide Column
- Google BigTable, Apache Cassandra

NoSQL

Document Database

- Organized around a “document” containing text
- Can handle very large data volumes
- Provides Speed and Scalability
- Document format is easily understood by humans
- No “schema”, but JSON/XML provides internal structure within a document
- Documents are indexed and stored within “collections”
- Supports full text search

Popular Implementations (open source)

MongoDB

CouchDB

NoSQL

JSON


```
{
"_id" : ObjectId("5e97444c99cddc2f99933a94"),
              "address" : {
              "building" : "284",
              "coord" : [
                            -73.9829239,
                            40.6580753
              ],
              "street" : "Prospect Park West",
              "zipcode" : "11215"
              },
"borough" : "Brooklyn",
"cuisine" : "American",
"grades" : [
    {
     "date" : ISODate("2012-12-05T00:00:00Z"),
     "grade" : "A",
     "score" : 13
     },
     {
     "date" : ISODate("2012-05-17T00:00:00Z"),
     "grade" : "A",
     "score" : 11
     }
     ],
"name" : "The Movable Feast",
"restaurant_id" : "40361606"
}
```

NoSQL

XML

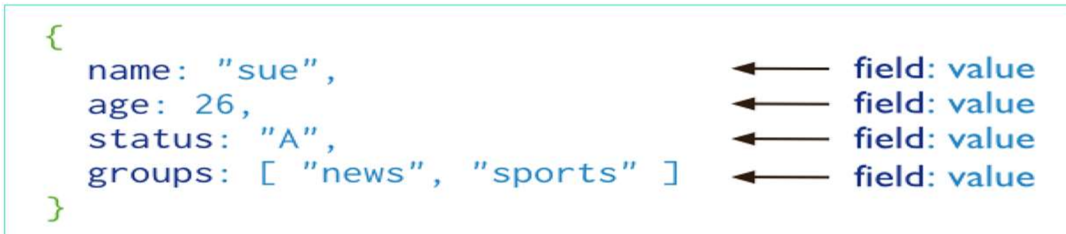
```
<contact>
        <firstname>Bob</firstname>
        <lastname>Smith</lastname>
        <phone type="Cell">(123)555-0178</phone>
        <phone type="Work">(890)555-0133</phone>
        <address>
                <type>Home</type>
                <street>123 Black St.</street>
                <city>Big Rock</city>
                <state>AR</state>
                <zip>23225</zip>
                <country>USA</country>
        </address>
</contact>
```

NoSQL

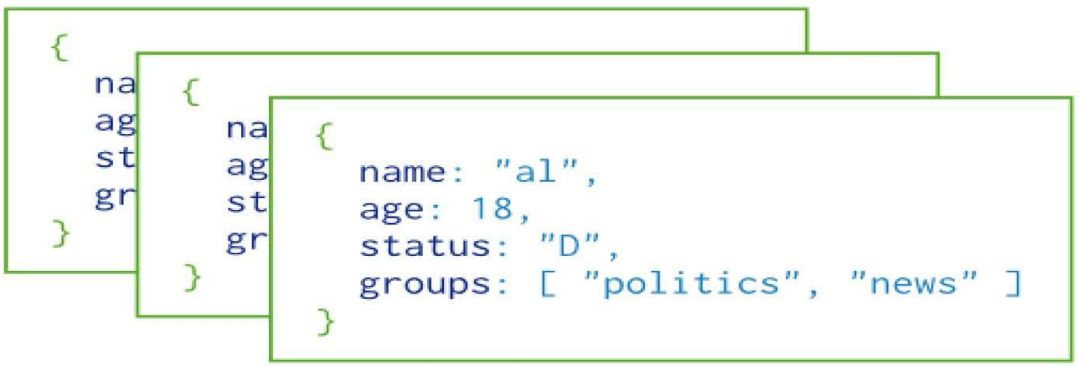
Document Database

- Keeps related information together (not normalized into tables)
- Access to a document is fast (index/key)
- ACID compliance is maintained only within a document
- Cannot easily "join" across documents
- Documents are kept in "collections"

NoSQL



A MongoDB document.



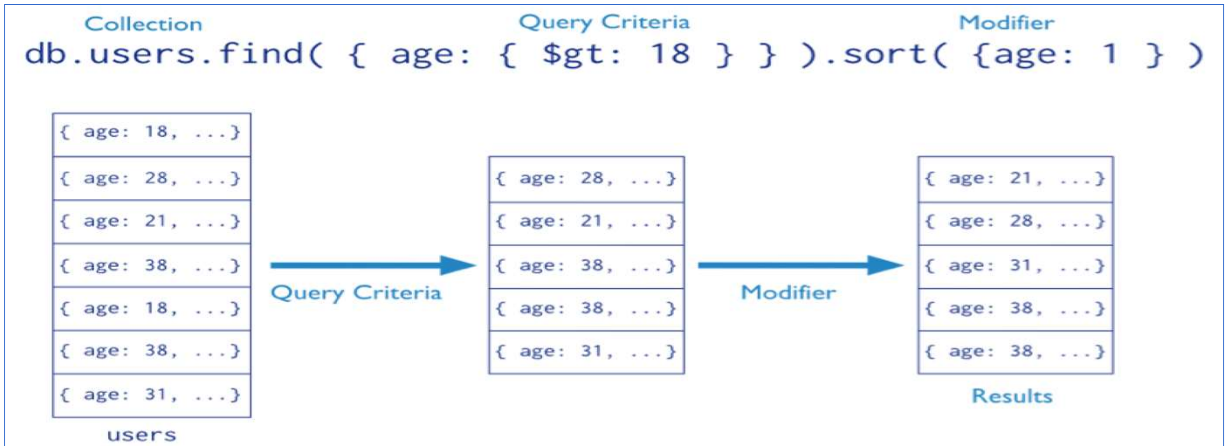
Collection

A collection of MongoDB documents.

NoSQL

In MongoDB a query targets a specific collection of documents. Queries specify criteria, or conditions, that identify the documents that MongoDB returns to the clients.

A query may include a *projection* that specifies the fields from the matching documents to return. You can optionally modify queries to impose limits, skips, and sort orders.



The stages of a MongoDB query with a query criteria and a sort modifier.

NoSQL

Graph Database

Uses a graph structure consisting of

- Nodes – Represents an entity (like a person)
- Edges – Represents a relationship between entities
- Properties – Attributes associated with Nodes and Edges

Supports navigation along edges from a starting point node

Designed for applications tracking the inter-connections among entities

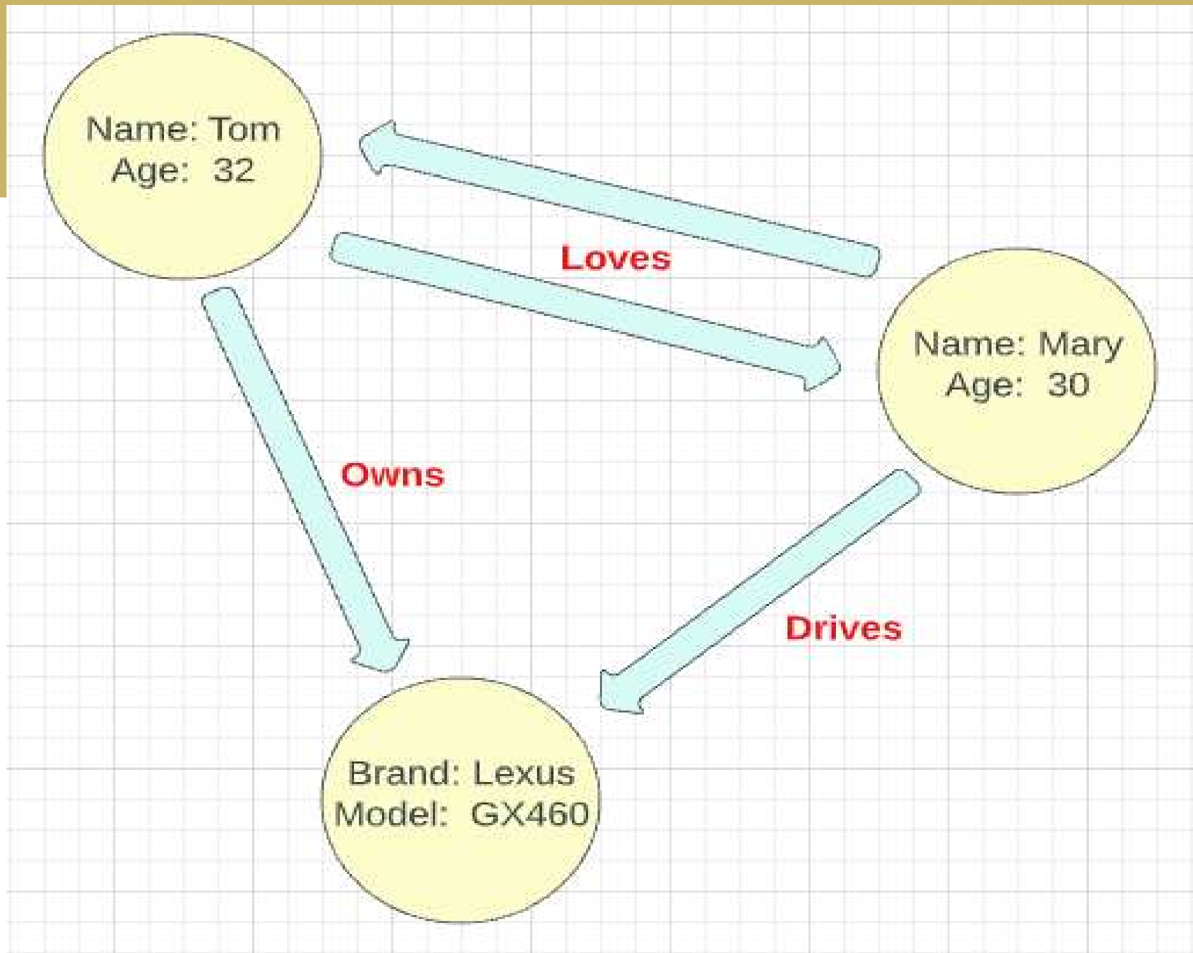
- Who is friends with whom? (In a social network application)
- Who is following me, who am I following?

Uses a pattern matching query language to navigate nodes & edges

Popular Implementations (open source)

Neo4j

NoSQL


Name: Tom
Age: 32
Loves
Name: Mary
Age: 30
Owns
Drives
Brand: Lexus
Model: GX460

NoSQL

Key:Value Pairs Database

- "Schemaless" – no structure
- Maps a key to an opaque value

  (That is, the database doesn't understand anything within the value)
- Simple query operations (put, get, remove, modify)
- Keys are unique in a collection
- May be a building block for other data models (such as key:value pairs within a document or a graph)

Popular Implementations

- Amazon Dynamo (available via AWS in the cloud)
- Redis (open source)

NoSQL

Product Catalog Example

```
Item = {
        Id: "207",
        Title: "27-Bicycle 207",
        Description: "207 description",
        BicycleType: "Touring",
        Brand: "ParaBikes",
        Price: 899,
        Color: ["Blue", "White"],
        ProductCategory: "Bike",
        QuantityOnHand: 6,
        RelatedItems: [
            342,
            478,
            644
        ],
        Pictures: {
            FrontView: "http://example.com/products/207_front.jpg",
            RearView: "http://example.com/products/207_rear.jpg",
            SideView: "http://example.com/products/207_left_side.jpg"
        },
        ProductReviews: {
            FiveStar: [
                "Love this bike !!",
                "Top quality components"
            ],
            OneStar: [
                "The paint chips easily"
            ]
        }
    }
```

Sample **Key:Value** aggregate

- One primary key
- Each attribute has a key and a value
- May store multiple values in an array
- Key:Value Pairs provide some structure

Not all documents will have the same key:value pairs

NoSQL

Key-Value Pairs Database Example (Amazon DynamoDB)

- The key value (Id) is 206.
- Most of the attributes have simple data types, such as String, Number, Boolean and Null.
- One attribute (Color) is a String Set.
- The following attributes are document data types:

  - A List of Related Items. Each element is an Id for a related product.

  - A Map of Pictures. Each element is a short description of a picture, along with a URL for the corresponding image file.

  - A Map of Product Reviews. Each element represents a rating and a list of reviews corresponding to that rating. Initially, this map will be populated with five-star and one-star reviews.

NoSQL

Wide-Column (Column Family) Store Database

- A TWO-LEVEL aggregate
- Data is stored within “collections” of dynamic related columns
- Similar to key/value with the pairs having columnar structure

  Based on Google’s “Big Table”

Popular Implementations (open source)

Cassandra

HBase

NoSQL

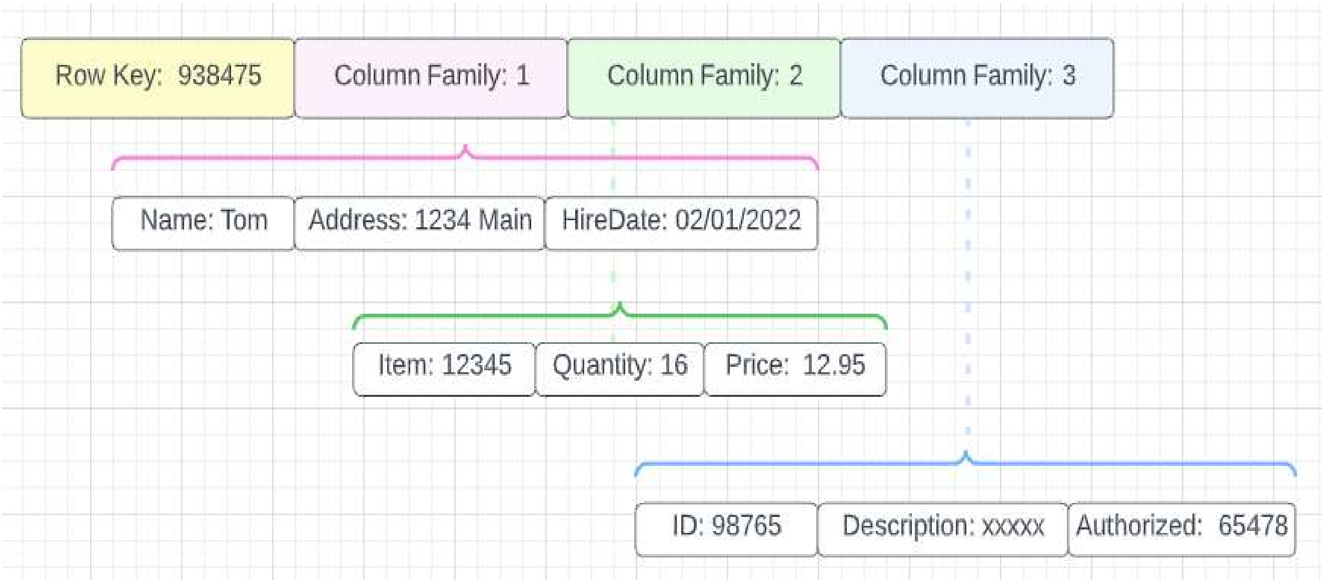
The first key is the row-key

- The entire row is an aggregate of related data
- The row consists of many key-value pairs (“columns”)

The second key is the column family

- Each column family consists of sparse key-value pairs
- “Sparse” means the column value isn’t stored if it isn’t needed

NoSQL


Row Key: 938475
Column Family: 1
Column Family: 2
Column Family: 3
Name: Tom
Address: 1234 Main
HireDate: 02/01/2022
Item: 12345
Quantity: 16
Price: 12.95
ID: 98765
Description: xxxxx
Authorized: 65478

NoSQL

Benefits of a Wide Column Store

- Lookup of rows by row key can be **very fast**
- In a distributed cluster system, the complete row is stored on one node
- May be stored redundantly across the cluster
- Can be retrieved with one access
- Good fit for **scale-out** systems
- Can scale to large capacity and high availability
- The columns are "**sparse**"
  - That is, columns with no values are absent, saving space
- Flexible access to column data in a row
  - You can flexibly query on them

NoSQL

Issues with a Wide Column Store

- Client code **may be extensive**
- Data structures are custom-built by query
- Different from relational model where you maintain one set of data, and just write queries as needed
- However, wide column stores tend to have much more scale out capability -- Which is why we use them in the first place

NoSQL

Next Topic: MongoDB