NoSQL Databases

CSIT882: Data Management Systems

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

What is NoSQL?

- When people use the term “NoSQL”, they typically use it to refer to any non-relational database.
- Some say the term “NoSQL” stands for “not only SQL”.

- Either way, most agree that NoSQL databases are databases that store data in a format other than relational tables.

- First proposed in 2009 for the projects experimenting with alternate data storage, like BigTable (Google) and Dynamo (Amazon)

What is NoSQL?

- NoSQL database systems were developed in a response to the demands for processing **big data** produced by increasing Internet usage and mobile geo-location technologies
- Traditional solutions were either too expensive, not scalable, or required too much time to process data
- Modern NoSQL database systems made significant advances in **scalability** and **efficient processing** of diverse types of data such as text audio, video, image, and geo-location
- NoSQL database systems include the following types of database systems categorized by a logical view of data provided: **key-value DB, document DB, column stores**, **graph DB,** etc

Big Data

Very large volumes of data being collected

- Driven by growth of web, social media, and more recently internet-of-things
- Web logs were an early source of data
- Analytics on web logs has great value for advertisements, web site structuring, what posts to show to a user, etc

Big Data: different from data handled by earlier generation databases

- **V**olume: much larger amounts of data stored
- **V**elocity: much higher rates of insertions
- **V**ariety: many types of data, beyond relational data

Trends & Requirements

Trends

Volume of data

Velocity of data

Variety of data

Requirements

Real database scalability

- massive database distribution
- dynamic resource management
- horizontally scaling systems

Frequent update operations

Massive read throughput

Flexible database schema

- semi-structured data

RDBMS for Big Data

relational schema

- data in tuples
- a priori known schema

but current data are naturally flexible

schema normalization

- data split into tables (3NF)
- queries merge the data

inefficient for large data

slow in distributed environment

transaction support

- trans. management with ACID
- safety first

full transaction is very inefficient in distributed environment.

Why NoSQL?

- Flexible data models
  - A flexible schema allows to easily make changes to database as requirements change
- Horizontal scaling
  - SQL: require scale-up vertically (migrate to a larger, more expensive server)
  - NoSQL: add cheaper commodity servers whenever needed.
- Fast queries
  - SQL: data is normalised => join data from multiple tables
  - NoSQL: usually stored in a way optimised for queries

CAP Theorem

At most two of the following three can be maximized at one time

Consistency

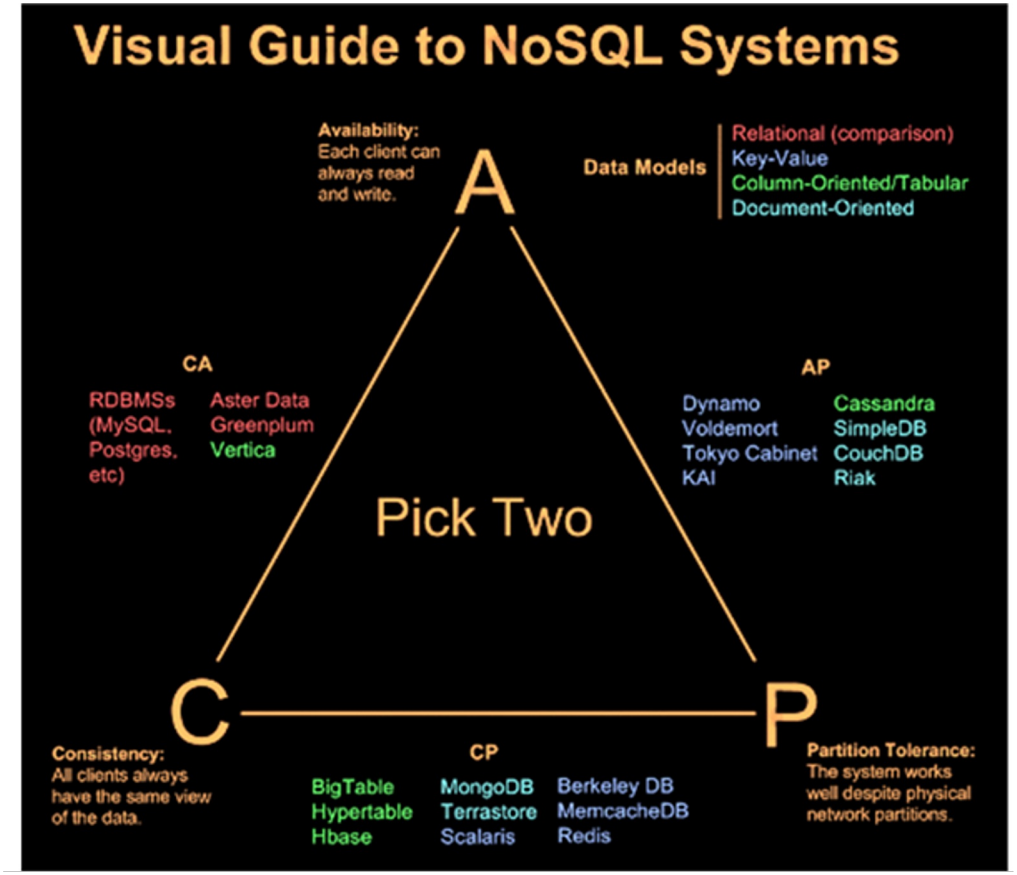
- Each client has the same view of the data

Availability

- Each client can always read and write

Partition tolerance

- System works well across distributed physical networks



NoSQL Databases

NoSQL: Database technologies that are (mostly):

- Not using the relational model (nor the SQL language)
- Designed to run on large clusters (horizontally scalable)
- No schema - fields can be freely added to any record
- Open source
- Based on the needs of the current big data era

Other characteristics (often true):

- easy replication support (fault-tolerance, query efficiency)
- eventually consistent (not ACID)

The End of RDBMS?

Relational databases are not going away

- are ideal for a lot of structured data, reliable, mature, etc.

RDBMS became one option for data storage

Using different data stores in different circumstances

Two trends

- NoSQL databases implement standard RDBMS features
- RDBMS are adopting NoSQL principles

NoSQL Databases

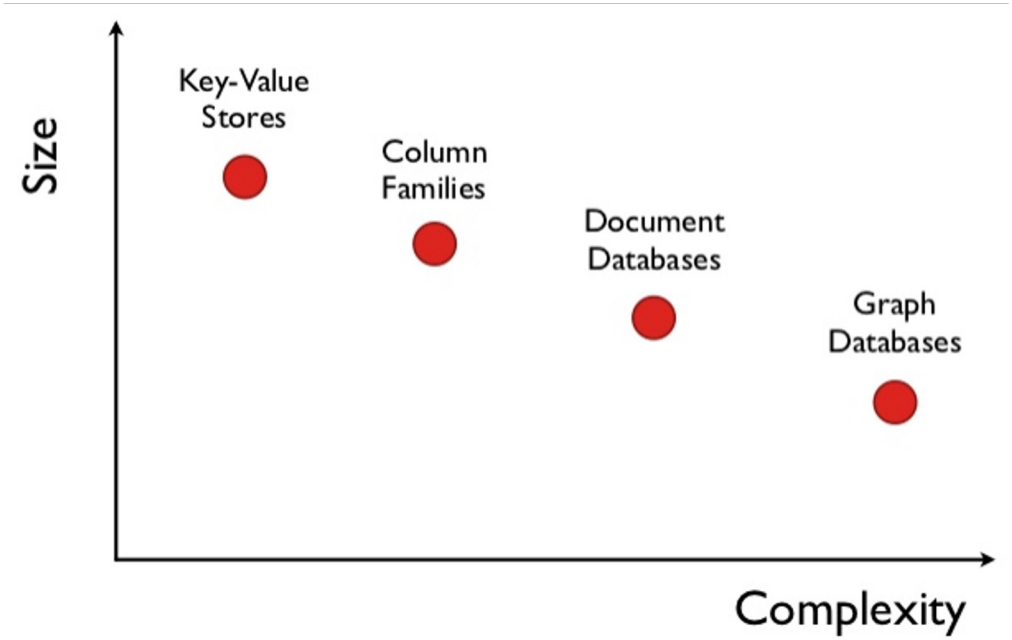
NoSQL database systems include the following types of database systems categorized by a logical view of data provided

Key-value stores

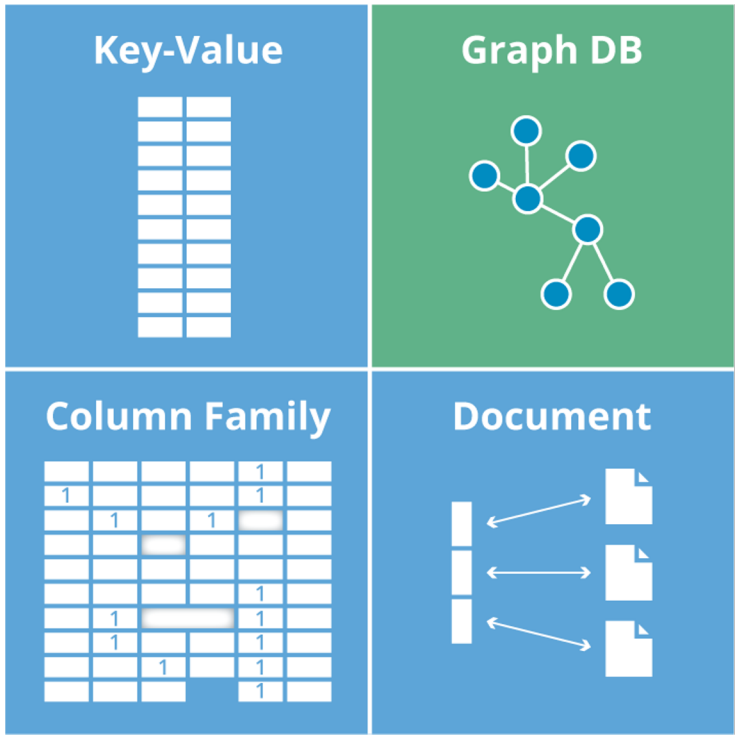
Document databases

Column-family stores

Graph databases



NoSQL Databases by Data Models



APACHE HBASE

Cassandra

CouchDB relax

riak

mongoDB

HYPERTABLE INC

Neo4j

redis

Key-Value

Come from a research paper by Amazon (Dynamo)

- Global Distributed Hash Table (Key-Value Stores)

A simple hash table (map), primarily used when all accesses to the database are via primary key

- key-value mapping

In RDBMS world: A table with two columns:

- ID column (primary key)
- DATA column storing the value (unstructured binary large object)

Key-Value Operations

Key-value stores support

- put(key, value): used to store values with an associated key
- get(key): which retrieves the stored value associated with the specified key
- delete(key): remove the key and its associated value

Some systems also support range queries on key values

Key-Value

Why?

- Simple Data Model: Hash Table is well-studied
- Good Scalability: Small System Cost, via good look-up locality and caching

Why not?

- Poor to complex (interconnected) data

Key-Value Vendors

Project Voldemort

Ranked list: http://db-engines.com/en/ranking/key-value+store

Document Stores

Basic concept of data: Document

Documents are self-describing pieces of data

- Hierarchical tree data structures
- Nested associative arrays (maps), collections
- XML, JSON (JavaScript Object Notation), BSON, …

Documents in a collection should be “similar”

- Their schema can differ

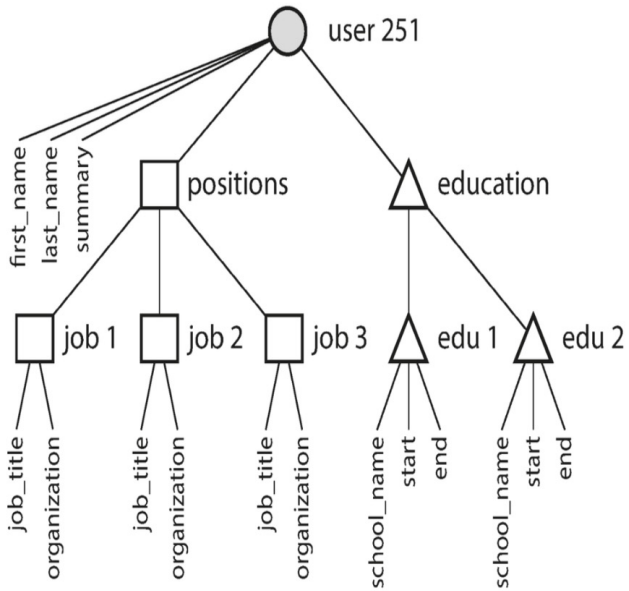
Documents stored in the value part of key-value

- Key-value stores where the values are examinable
- Building search indexes on various keys/fields

Document Stores - Example

```
{ "personID": 3,
  "firstname": "Martin",
  "likes": [ "Biking","Photography" ],
  "lastcity": "Boston",
  "visited": [ "NYC", "Paris" ] }

{ "personID": 5,
  "firstname": "Pramod",
  "citiesvisited": [ "Chicago", "London","NYC" ],
  "addresses": [
     { "state": "AK",
       "city": "DILLINGHAM" },
     { "state": "MH",
       "city": "PUNE" }  ],
  "lastcity": "Chicago‟ }
```



MongoDB

humongous => Mongo

Data is organized in collections. A collection stores a set of documents.

Collection is like a table and document is like a record

- but: each document can have a different set of attributes even in the same collection
- Semi-structured schema

Only requirement: every document should have an “_id” field

MongoDB Example

```
{     "_id”:ObjectId("4efa8d2b7d284dad101e4bc9"),
       "Last Name": ” Cousteau",
       "First Name": ” Jacques-Yves",
       "Date of Birth": ”06-1-1910" },

{     "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
       "Last Name": "PELLERIN",
       "First Name": "Franck",
       "Date of Birth": "09-19-1983",
       "Address": "1 chemin des Loges",
       "City": "VERSAILLES" }
```

MongoDB vs RDBMS

RDBMS	**MongoDB Equivalent**
database	database
table	collection
row	document
attributes	fields (field-name:value pairs)
primary key	the *'_id'* field, which is the key associated with the document

Relationships in MongoDB

Two options:

- store references to other documents using their _id values
- embed documents within other documents

```
{
  "_id":ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address_ids": [
    ObjectId("52ffc4a5d85242602e000000"),
    ObjectId("52ffc4a5d85242602e000001")
  ]
}
```

```
{
  "_id":ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address": [
    {
      "building": "22 A, Indiana Apt",
      "pincode": 123456,
      "city": "Los Angeles",
      "state": "California"
    },
    {
      "building": "170 A, Acropolis Apt",
      "pincode": 456789,
      "city": "Chicago",
      "state": "Illinois"
    }
  ]
}
```

MongoDB - Queries

Query language expressed via JSON

clauses: where, sort, count, sum, etc.

SQL: `SELECT * FROM users`
MongoDB: `db.users.find()`

- `SELECT * FROM users WHERE personID = 3`
- `db.users.find( { "personID": 3 } )`

Document Stores - Vendors

MS Azure DocumentDB

Ranked list: http://db-engines.com/en/ranking/document+store

Column-Family Stores

Origin from Google's BigTable

Also known as wide-column

Column families are groups of related data (columns) that are often accessed together

Column-Family Stores - Main Idea

Each table tends to have many attributes (from thousands ~ millions)

In most applications (in OLAP) we are only interested in a few attributes

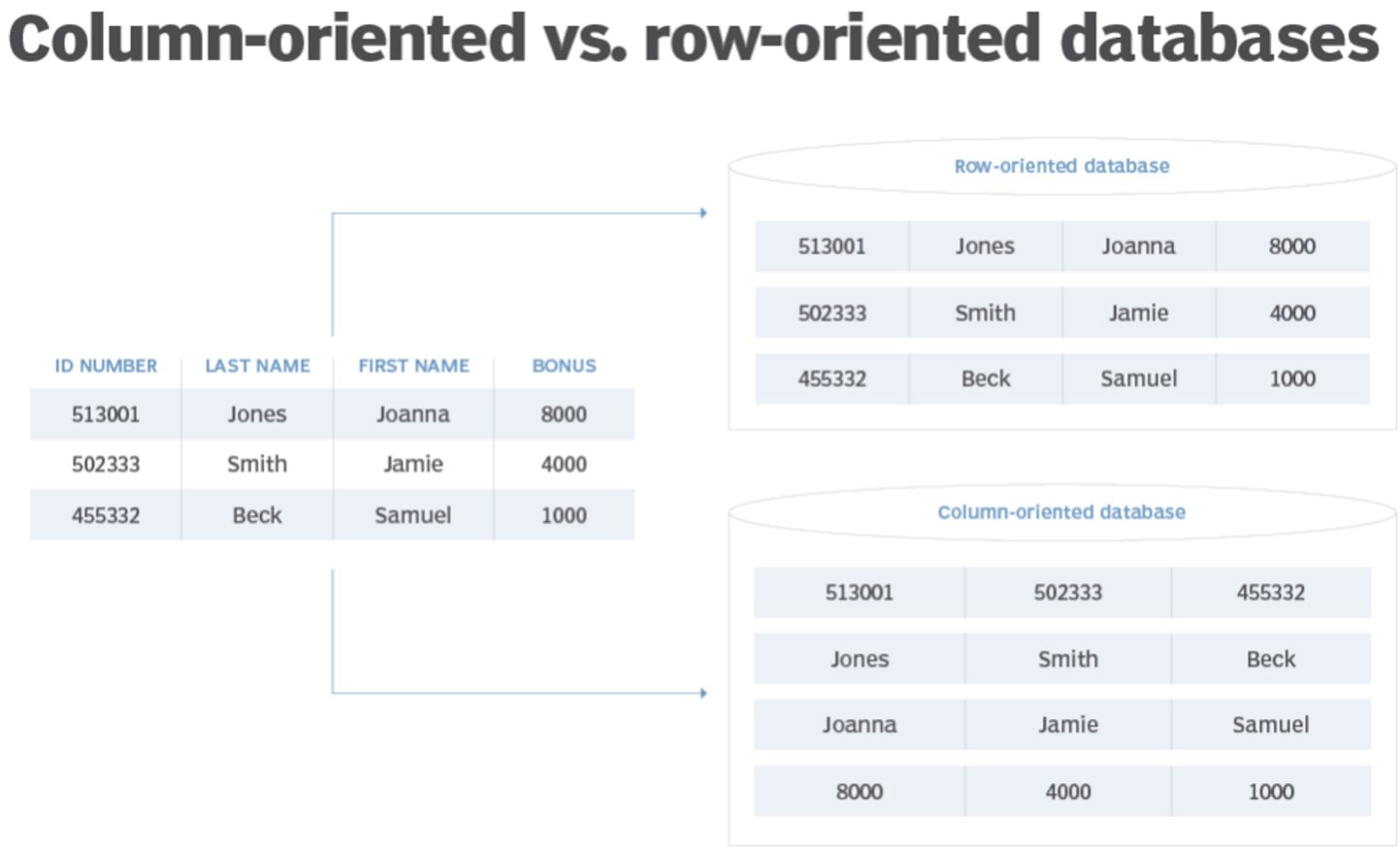
Traditional row-based

- Store each record in a sequential file
- We need to read the whole record to access only one attribute

Column-based

- Store the data by putting the same attribute in a sequential file
- Faster access and better compression

Column-Family Stores - Example



Column-Family Stores

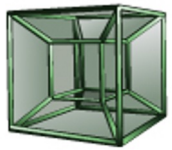
Why?

- Optimized for OLAP
- Semi-Structured Data: Each column can define its own schema

Why not? Not good for

- OLTP
- Row-specific Queries

Column-Family Stores - Vendors



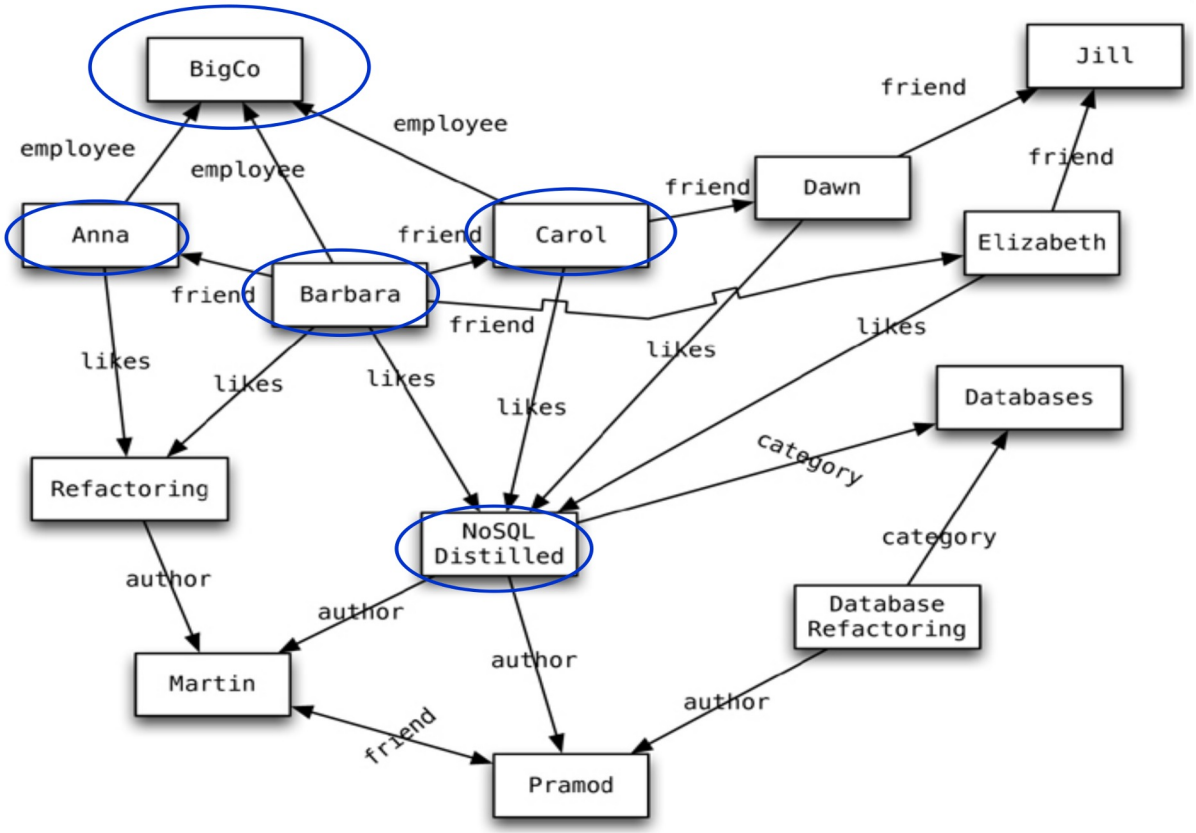
HYPERTABLE

Ranked list: http://db-engines.com/en/ranking/wide+column+store

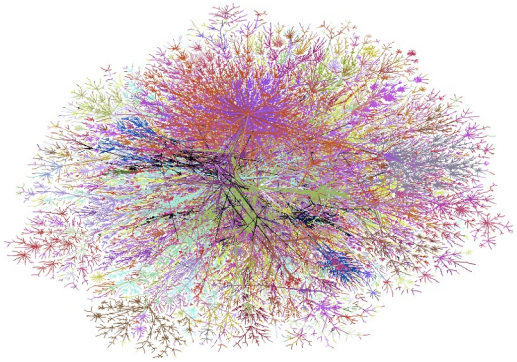
Graph Database

Data Model

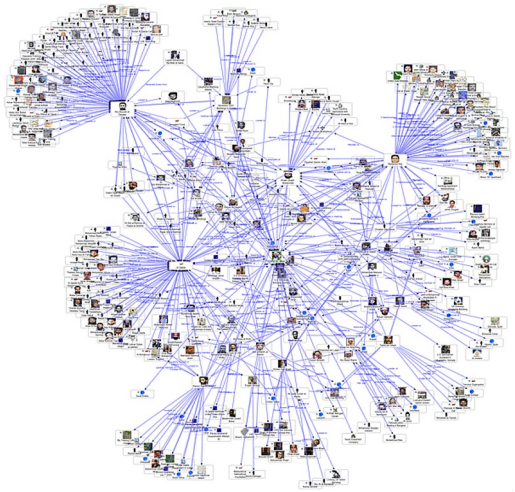
- Vertices (Nodes) -> Entities
- Edges -> Relations



Graphs are Everywhere

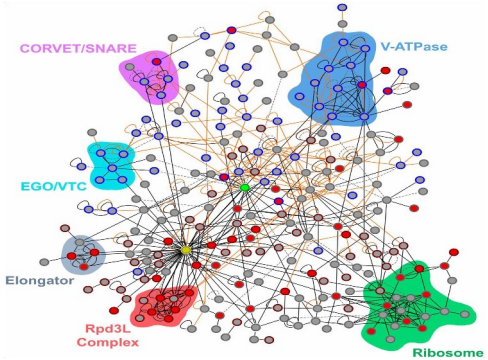


Internet

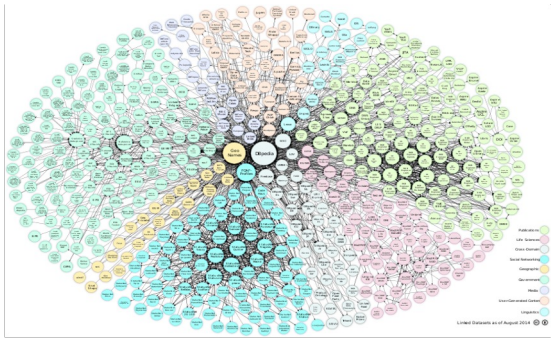


Social Networks

Road Networks

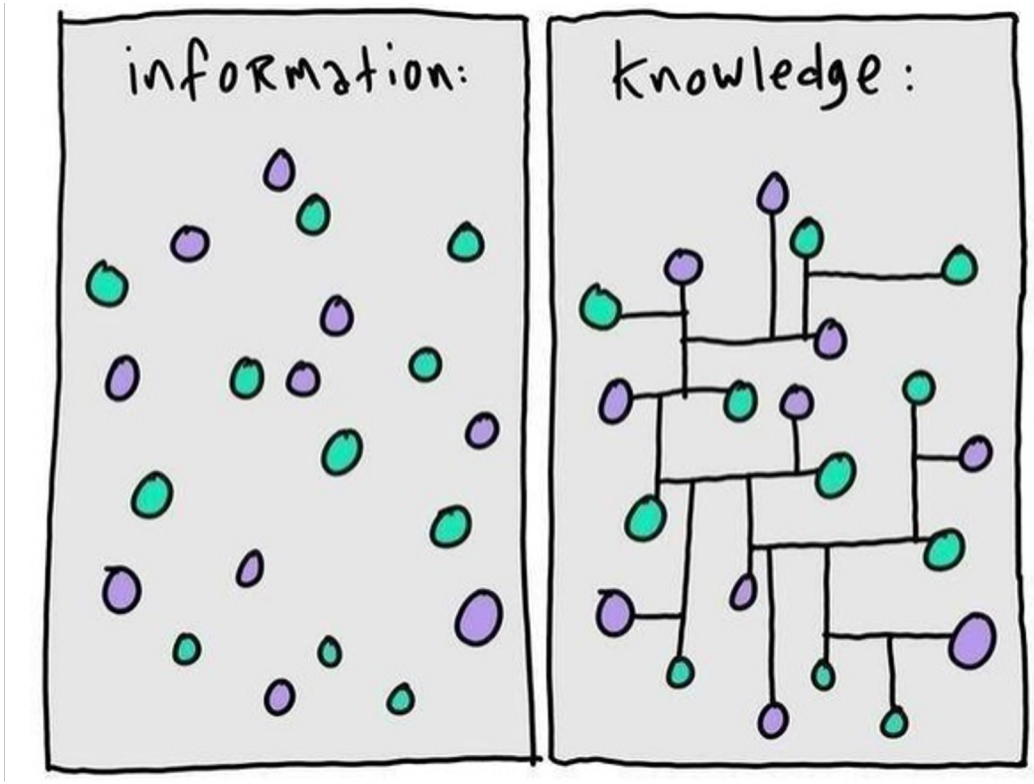


Biological Networks



Knowledge Graphs

Information vs Knowledge



Advantages

Performance

- Traditional Joins are inefficient
- Billion-scale data are common, e.g., Facebook social network, Google Web graph

Flexibility

- Relationships among entities can be arbitrary. It is not feasible to use 1000 tables to model 1000 types of relationships.

Agility

- Business requirements changes over time

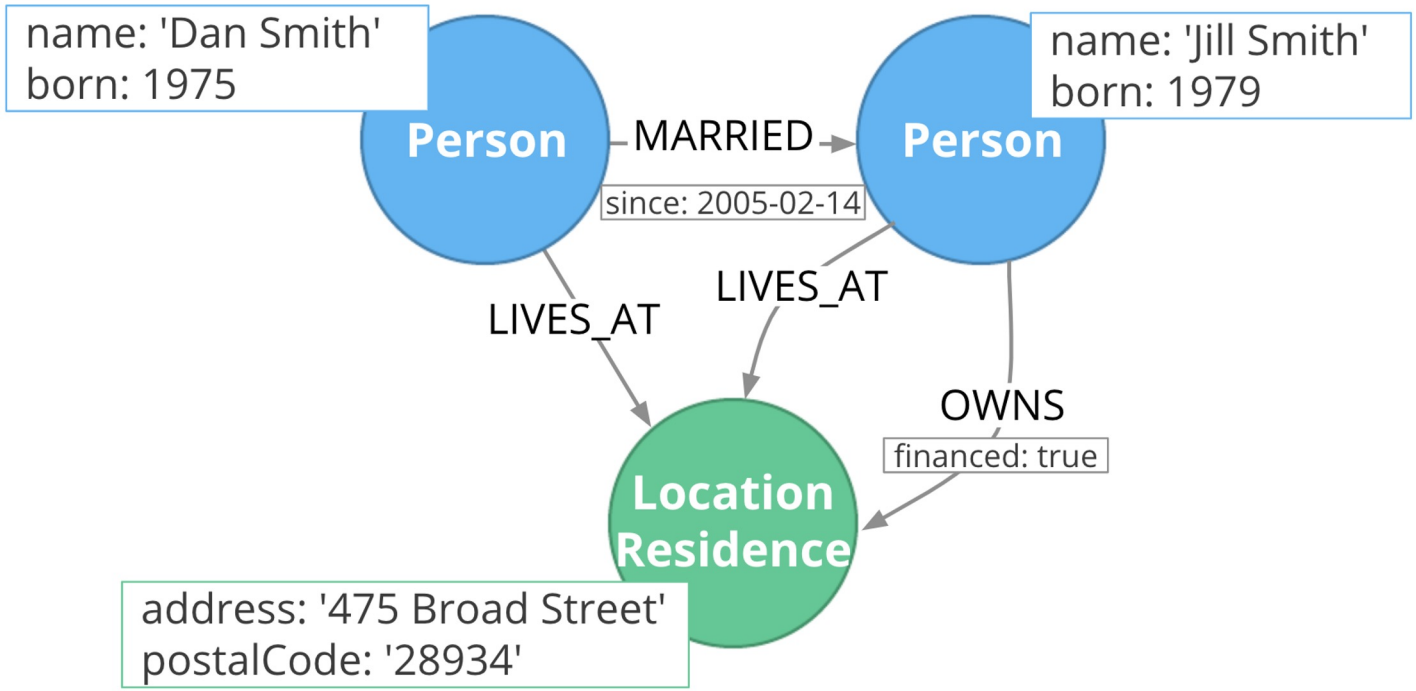
Property Graph Model

Nodes (Entities)

Relationships

Properties

Labels



Neo4j

The most popular Graph Database at present

Cypher query language

Developed in Java and open-source

Resources:

- Neo4j Cypher Manual: https://neo4j.com/docs/cypher-manual/4.0/
- Neo4j Developer Resources: https://neo4j.com/developer/resources/

Graph vs Relational Model - Example

Student

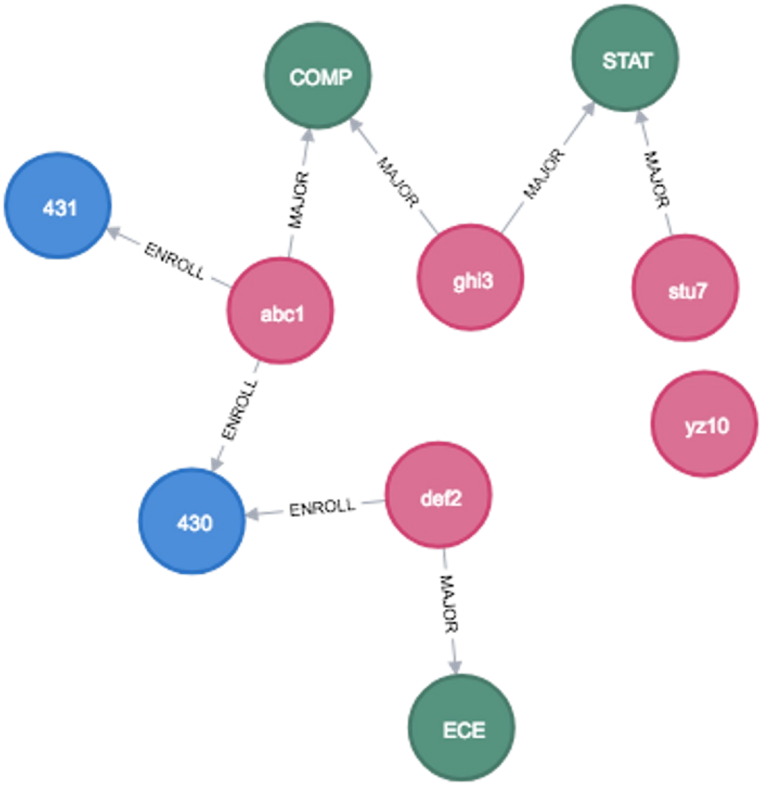
netId	FirstName
abc1	Albert
def2	Danielle
ghi3	Gary
stu7	Sandeep
yz10	Yusin

Enrolls

netId	Course
abc1	COMP 430
def2	COMP 430
abc1	COMP 431

Majors

netId	Major
ghi3	STAT
ghi3	COMP
abc1	COMP
def2	ECE
stu7	STAT



Types of Graph Queries

Graph Pattern Matching

- Given a graph pattern, find subgraphs in the database graph that match the query.
- Can be augmented with other (relational-like) features, such as projection.

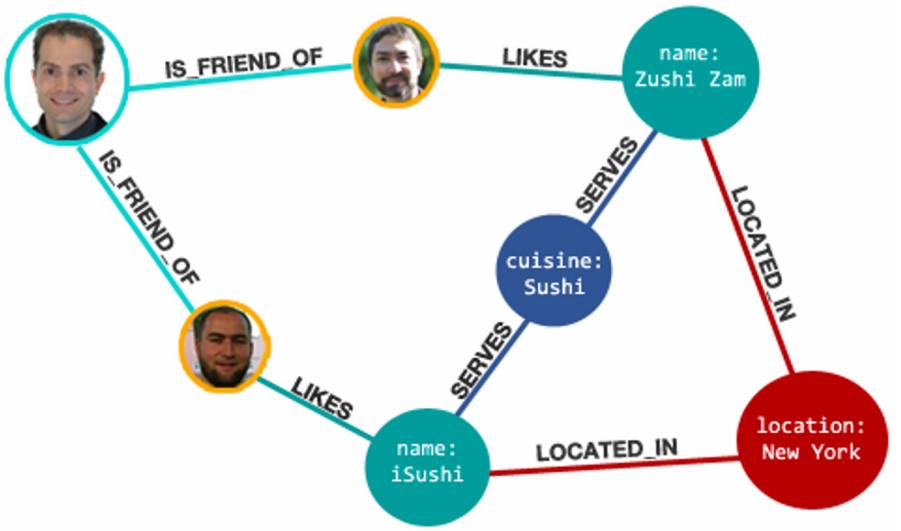
```
MATCH (p:Person)-[:LIKES]->(:Language {name = "SQL"})
RETURN p.name
```

Graph Navigation

- A flexible querying mechanism to navigate the topology of the data.
- Called path queries, since they require to navigate using paths (potentially variable length).

```
MATCH (p:Person)-[:KNOWS*1..2]->(:Person {name = "Alice"})
RETURN p.name
```

Cypher Query Language



An example of Cypher:

Find Sushi restaurants in New York

that my friend Philip like

```
MATCH (person:Person)-[:IS_FRIEND_OF]->(friend),
      (friend)-[:LIKES]->(restaurant:Restaurant),
      (restaurant)-[:LOCATED_IN]->(loc:Location),
      (restaurant)-[:SERVES]->(type:Cuisine)

WHERE person.name = 'Philip'
  AND loc.location = 'New York'
  AND type.cuisine = 'Sushi'

RETURN restaurant.name, count(*) AS occurrence
ORDER BY occurrence DESC
LIMIT 5
```

Graph Algorithms

The real power of graph databases

Can save huge amounts of programming effort

Include

- Centrality - node importance
- Community detection – node connectivity and partitions
- Path finding – routes through the network
- Similarity – of nodes
- Link prediction – closeness of nodes

https://neo4j.com/docs/graph-data-science/current/

Advantages/Disadvantages of NoSQL

- Which available data model to use should be decided on your data requirements.
- Not all NoSQL solutions are equal – i.e., different models serve different data requirements
- Generally, NoSQL solutions are considered lightweight and easy to implement (e.g., no schema required) – and could have high read/write throughput due to the relaxation in data consistency requirement
- However, NoSQL technologies are relatively new still – not as well optimised/developed as RDBMS
- Schema-less data storage could lead to less manageable database overtime.