Dr Sophia Tsoka

Department of Informatics

Faculty of Natural and Mathematical Sciences

March 2021

4CCS1DBS – Database Systems

Week 11 – Advanced Database Topics

Topic 1: Database Security Aspects

List of Topics

Term week	Topic	Lecturer
22	Introduction to Database Systems	Dr Tsoka
23	Data Modeling Using the Entity-Relationship Model	Dr Tsoka
24	Relational Data Model	Dr Curcin
25	SQL 1	Dr Curcin
26	SQL 2	Dr Curcin
27	**----CATCH UP WEEK---**	
28	Relational Algebra 1	Dr Curcin
29	Relational Algebra 2	Dr Curcin
30	Normalisation 1	Dr Tsoka
31	Normalisation 2	Dr Tsoka
32	Advanced Topics and Security Issues	Dr Tsoka

In this recording

- Security aspects in Databases
- noSQL Databases
  - Key/Value
  - Columnar
  - Document
  - Graph
- This material will not be on the exam, but will be important for your future database-related knowledge
    - (eg 5CCS2SEG: requires real-world databases)

Database Security

- Security considerations
  - Legal and ethical issues regarding the right to information access
  - Policy issues at governmental/institute/corporate level
    - (credit ratings / personal medical records)
  - System-related issues
    - (where should security be handled, physical hardware / OS / DBMS?
  - Multiple security levels

What are types of threats to databases?

- Threats to databases
  - Loss of ***integrity*** – against improper modification to data intentionally or accidentally
  - Loss of ***availability*** – for human user or program
  - Loss of ***confidentiality*** – against unauthorised disclosure

DBMS: Security and authorisation subsystem

- Two types of database security mechanisms:
  - ***Discretionary*** security mechanisms – owner grants privileges to users, to access data files, records, or fields in a specified mode (read, insert, delete, or update). Access is at the discretion of the owner
  - ***Mandatory*** security mechanisms - to enforce multilevel security by classifying the data and users into various security classes/levels and implementing the appropriate security policy. A user with “secret” access, cannot see “top secret” DB.
- The discretionary access control has traditionally been the main security mechanism for relational database systems.

Discretionary Access Control

- Based on granting and revoking privileges
- The SQL GRANT statement grants privileges to user accounts
  - GRANT privilege TO userName
- The REVOKE statement enables system administrators to revoke privileges from user accounts
  - REVOKE *privilege FROM userName*

Discretionary Access Control

- Granting and revoking privileges at 2 levels:
  - Account level: Privileges that each account holds independently of the relations in the database

PRIVILEGE	MEANING
SHOW DATABASES	Enable SHOW DATABASES to show all databases

  - The relation (or table level): Privilege to access each individual relation or view in the database

PRIVILEGE	MEANING
SELECT ON *tableName*	Enable use of SELECT on table *tableName*

Account Level: Privileges

- The privileges at the account level apply to the capabilities provided to the account itself and include the following:

PRIVILEGE	MEANING
CREATE	Enable database and table creation
CREATE VIEW	Enable views to be created or altered
ALTER	Enable use of ALTER TABLE
DROP	Enable databases, tables, and views to be dropped

Relation Level: Privileges

- Specify for each user the individual relations on which each type of command can be applied. Some privileges also refer to individual columns (attributes) of relations.
  - Includes the following:

PRIVILEGE	MEANING
ALTER ON *tableName*	Enable use of ALTER TABLE for *tableName*
SELECT ON *tableName*	Enable use of SELECT for *tableName*
INSERT ON *tableName*	Enable use of INSERT into *tableName*
UPDATE ON *tableName*	Enable use of UPDATE *tableName*

Example 1

- Suppose that the DBA creates 3 accounts --A1, A2, A3-- and wants only A1 to be able to create relations
  - **GRANT CREATE TO A1;**
- A1 creates 2 tables: EMPLOYEE and DEPARTMENT; A1 is then owner of these two relations and hence all the relation privileges on each of them.
- Suppose that A1 wants to grant A2 the privilege to insert and delete tuples in both of these relations:
  - **GRANT INSERT, DELETE ON EMPLOYEE, DEPARTMENT TO A2;**
- Suppose that A1 wants to allow A3 to retrieve information from either of the two tables:
  - **GRANT SELECT ON EMPLOYEE, DEPARTMENT TO A3;**
- Suppose that A1 decides to revoke the SELECT privilege on the EMPLOYEE relation:
  - **REVOKE SELECT ON EMPLOYEE FROM A3;**

Example 2

- *The database has three tables:*
    - *CUSTOMER*
    - *ORDER*
    - *ORDER-ITEM*
- *The system is to process sales orders. Entry and access to documents and the progressing of orders is handled by the sales office. There are currently four staff in this office:*
    - *Tracey, the supervisor, needs to be able to see and change everything*
    - *Bill, Sheila and Govind cannot create new customers.*
    - *Orders above £1000, cannot be handled by anyone except Tracey and Govind.*
- Develop a matrix of users against database resources:
    - Database resources in question are the base tables and views (virtual tables).
    - Your matrix will make it clear where access is to be permitted and to whom.

Example 2

- *The database has three tables: CUSTOMER, ORDER, ORDER-ITEM*
- *There are currently four members of staff in this office*
  - *Tracey, the supervisor, needs to be able to see and change everything*
  - *Bill, Sheila and Govind cannot create new customers.*
  - *Orders above £1000, cannot be handled by anyone except Tracey and Govind.*

	Tracey	**Bill**	**Govind**	**Sheila**
CUSTOMER	R/W	R	R	R
ORDER	R/W		R/W	
Small-ORDER (view)	R/W	R/W	R/W	R/W

Other Security Issues Related to Databases

- **Inference Control**: controls the security problem associated with access to a *statistical* database (statistical information or summaries of values based on various criteria), so that information about the individuals cannot be accessed.
- **Flow Control**: prevents information from flowing in such a way that it reaches unauthorised users.
- **Data Encryption**: protects sensitive data (such as credit card numbers) that is being transmitted via some type communication network.

Dr Sophia Tsoka

Department of Informatics

Faculty of Natural and Mathematical Sciences

March 2021

KING'S College LONDON

4CCS1DBS – Database Systems

Week 11 – Advanced Database Topics

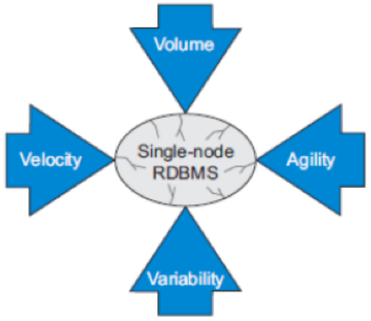
Topic 2: NoSQL Databases

In this recording

- Security aspects in Databases
- noSQL Databases
  - Key/Value
  - Columnar
  - Document
  - Graph
- This material will not be on the exam, but will be important for your future database-related knowledge
  - (eg 5CCS2SEG: requires real-world databases)

What is NoSQL

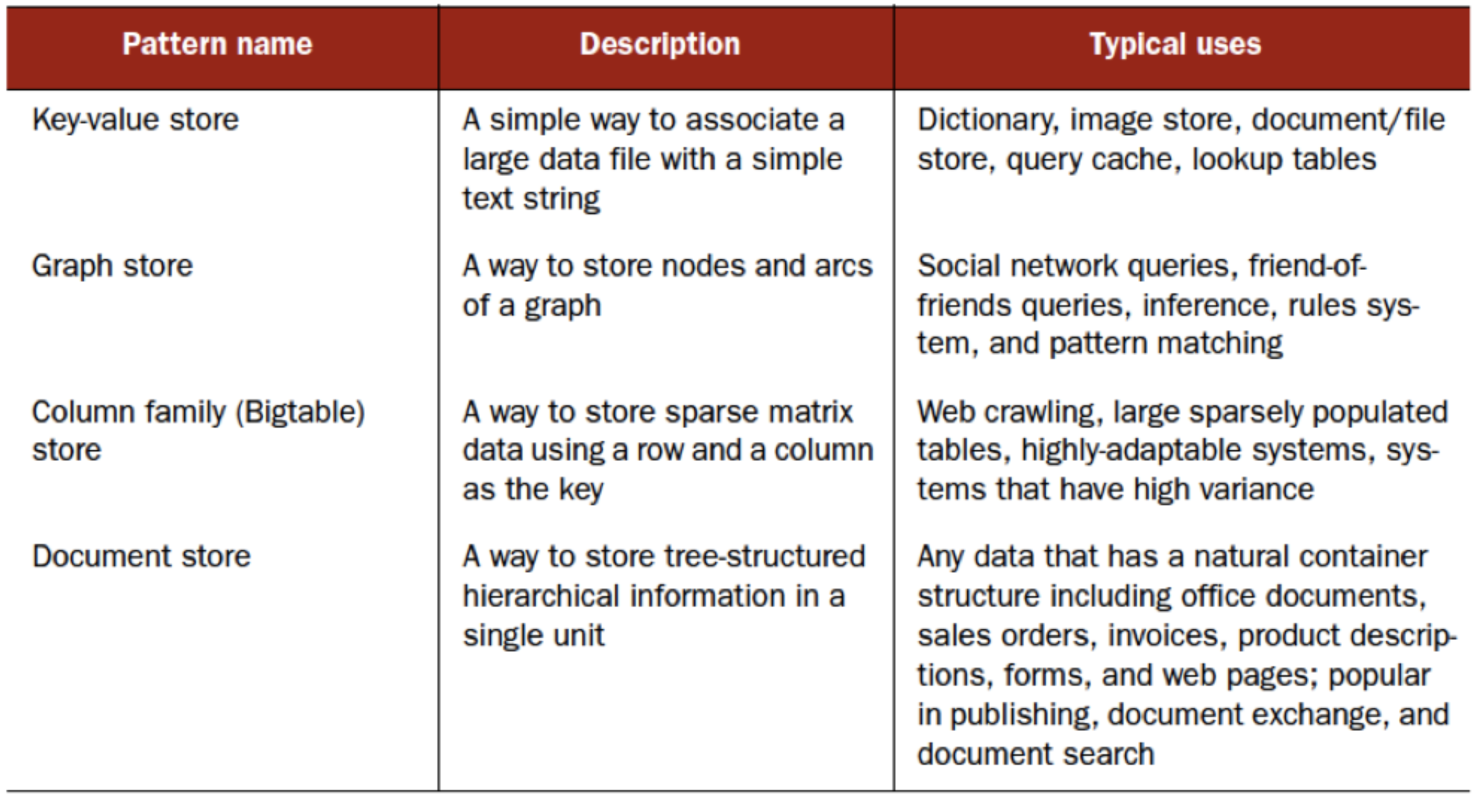
- **noSQL** - ***"non-relational"*** database
  - Traditionally SQL / Relational DBs dominate in usage
  - Since ~2009, alternatives to Relational Database started appearing
    - Google / Amazon Web-Services / Start-ups
  - *BIG Data* requirements (how do you query more data that can be held in memory?) Google BigTable/MapReduce
- NoSQL is a set of concepts that allows the rapid and efficient processing of data sets with focus on performance, reliability, and agility



noSQL - Alternatives to Relational Databases

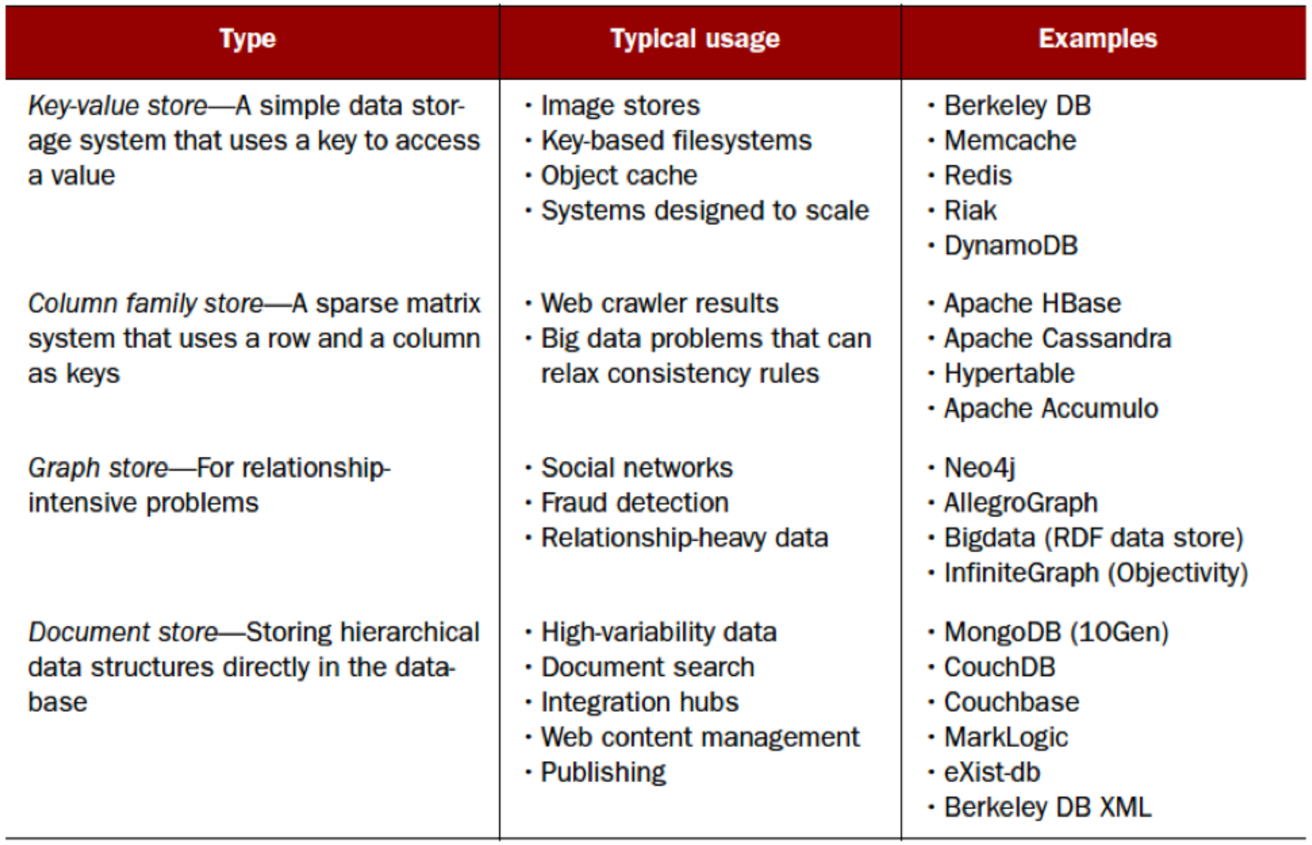
- **noSQL** - ***“not-only SQL”***
- More than rows in tables
  - NoSQL systems store and retrieve data from many formats
- Free of joins
  - NoSQL systems allow you to extract your data using simple interfaces without joins
- It's schema-free
  - NoSQL systems allow you to drag-and-drop your data into a folder and then query it without creating an entity-relational model.
- It works on many processors
- NoSQL systems allow you to store your database on multiple processors and maintain high-speed performance

NoSQL data stores



Pattern name	Description	Typical uses
Key-value store	A simple way to associate a large data file with a simple text string	Dictionary, image store, document/file store, query cache, lookup tables
Graph store	A way to store nodes and arcs of a graph	Social network queries, friend-of-friends queries, inference, rules system, and pattern matching
Column family (Bigtable) store	A way to store sparse matrix data using a row and a column as the key	Web crawling, large sparsely populated tables, highly-adaptable systems, systems that have high variance
Document store	A way to store tree-structured hierarchical information in a single unit	Any data that has a natural container structure including office documents, sales orders, invoices, product descriptions, forms, and web pages; popular in publishing, document exchange, and document search

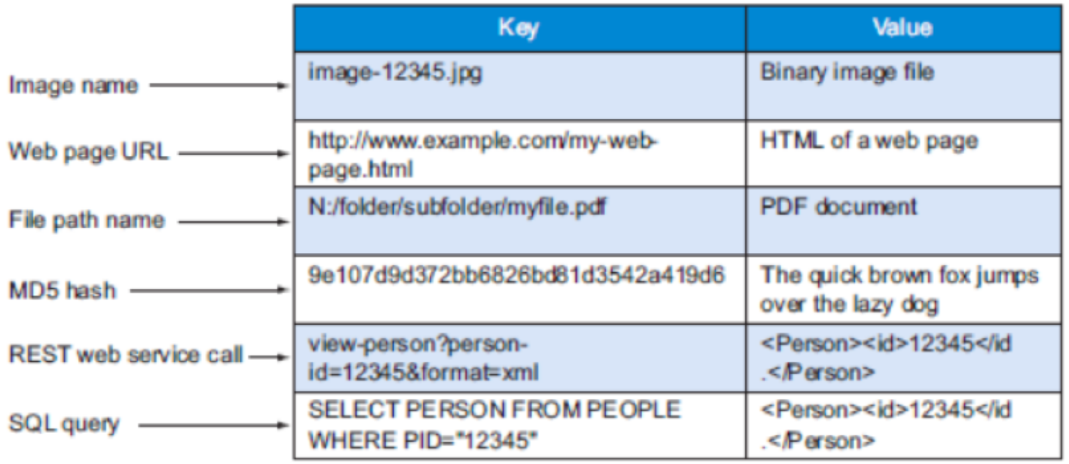
Types of NoSQL data stores



Type	Typical usage	Examples
Key-value store—A simple data storage system that uses a key to access a value	· Image stores · Key-based filesystems · Object cache · Systems designed to scale	· Berkeley DB · Memcache · Redis · Riak · DynamoDB
Column family store—A sparse matrix system that uses a row and a column as keys	· Web crawler results · Big data problems that can relax consistency rules	· Apache HBase · Apache Cassandra · Hypertable · Apache Accumulo
Graph store—For relationship-intensive problems	· Social networks · Fraud detection · Relationship-heavy data	· Neo4j · AllegroGraph · Bigdata (RDF data store) · InfiniteGraph (Objectivity)
Document store—Storing hierarchical data structures directly in the database	· High-variability data · Document search · Integration hubs · Web content management · Publishing	· MongoDB (10Gen) · CouchDB · Couchbase · MarkLogic · eXist-db · Berkeley DB XML

Key-Value Stores

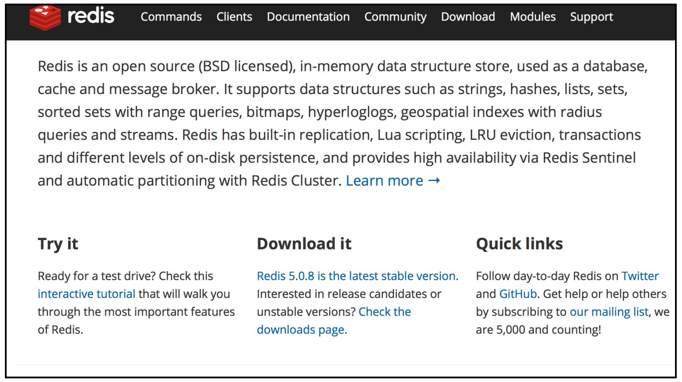
- Is a simple database that when presented with a simple string (the key) returns an arbitrary large BLOB (Binary Large OBject) of data (the value)
- Has no query language; it provides a way to add and remove key-value pairs
- Is indexed by the key; resulting in rapid retrieval



	Key	Value
Image name	image-12345.jpg	Binary image file
Web page URL	http://www.example.com/my-web-page.html	HTML of a web page
File path name	N:/folder/subfolder/myfile.pdf	PDF document
MD5 hash	9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
REST web service call	view-person?person-id=12345&format=xml	<Person><id>12345</id .</Person>
SQL query	SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id .</Person>

Using Key-Value Stores

- There are three operations performed on a key-value store:
  1. **put** adds a new key-value pair to the table and will update a value if this key is already present.
  2. **get** returns the value for any given key, or it may return an error message if there's no key in the key-value store.
  3. **delete** removes a key and its value from the table, or it many return an error message if there's no key in the key-value store.
- Key-value store has two rules:
  - Distinct keys: all the keys in any given key-value store are unique.
  - No queries on values: You can't perform queries on the values of the table.



Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster. Learn more →

Try it

Ready for a test drive? Check this interactive tutorial that will walk you through the most important features of Redis.

Download it

Redis 5.0.8 is the latest stable version. Interested in release candidates or unstable versions? Check the downloads page.

Quick links

Follow day-to-day Redis on Twitter and GitHub. Get help or help others by subscribing to our mailing list, we are 5,000 and counting!

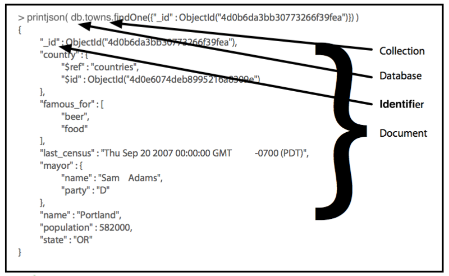
- ***Example:*** *http://redis.io/*
  - *http://try.redis.io/*

Key-Value Databases

- **Advantages:**
  - Maps simple keys to (possibly) more complex values like a huge hashtable (associative array)
  - Relatively simple - primitive to complex data-types
  - Fast
  - Easily distributed - Horizontally scalable
- **Drawbacks:**
  - Lacks indexes and good scanning capabilities
  - Not good for relational data (i.e. doing a JOIN)
  - Only basic CRUD (Create, Read, Update, Delete)
  - Lacks more complex queries (i.e. Aggregates and Group By)

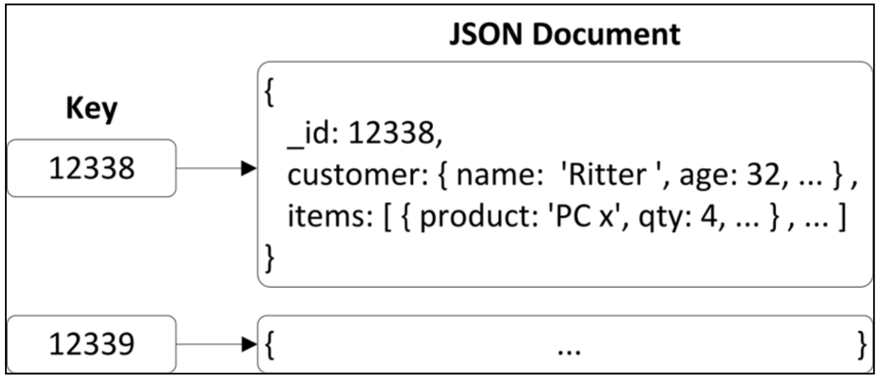
Document-oriented databases

- A document-oriented database (or document store) is a computer program designed for storing, retrieving, and managing document-oriented information
- Document-oriented databases are a subclass of the key-value store
- In a key-value store the data is considered to be inherently opaque
- A document-oriented system relies on internal structure in the document in order to extract metadata that the database engine uses for further optimization
- The document content may be referenced via retrieval or editing methods



```
> printjson( db.towns.findOne({"_id" : ObjectId("4d0b6da3bb30773266f39fea")}) )
{
        "_id" : ObjectId("4d0b6da3bb30773266f39fea"),
        "country" : {
                "$ref" : "countries",
                "$id" : ObjectId("4d0e6074deb8995216a8309e")
        },
        "famous_for" : [
                "beer",
                "food"
        ],
        "last_census" : "Thu Sep 20 2007 00:00:00 GMT        -0700 (PDT)",
        "mayor" : {
                "name" : "Sam    Adams",
                "party" : "D"
        },
        "name" : "Portland",
        "population" : 582000,
        "state" : "OR"
}
```

mongoDB FOR GIANT IDEAS



```
{
  _id: 12338,
  customer: { name:  'Ritter ', age: 32, ... } ,
  items: [ { product: 'PC x', qty: 4, ... } , ... ]
}
```

```
{                    ...                    }
```

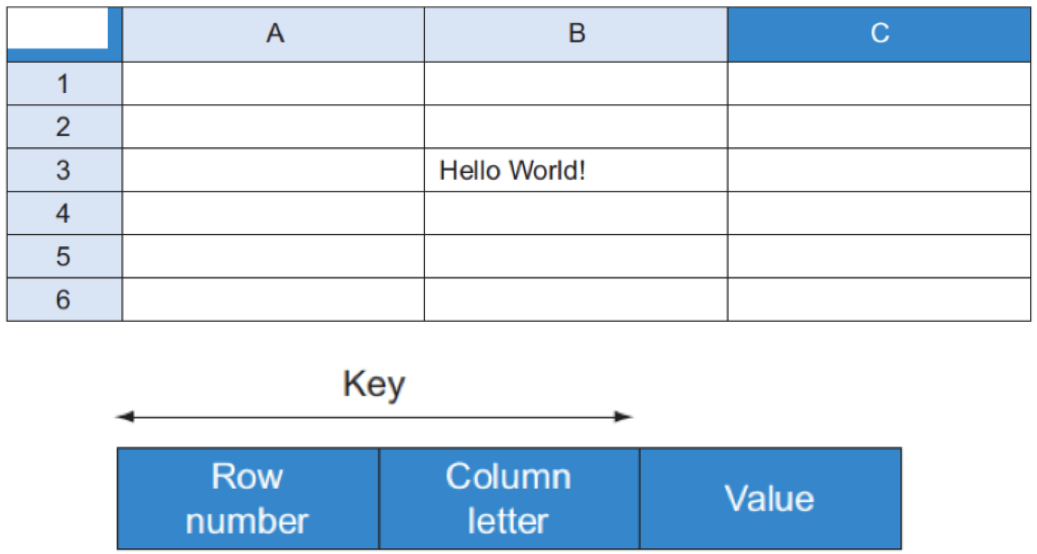
Document databases

- **Advantages**
  - Document databases allow for any number of fields per data element
  - Assume a standard file-encodings per object: XML, YAML, JSON, etc.
  - Great for highly variable domains unsure of what the schema will be

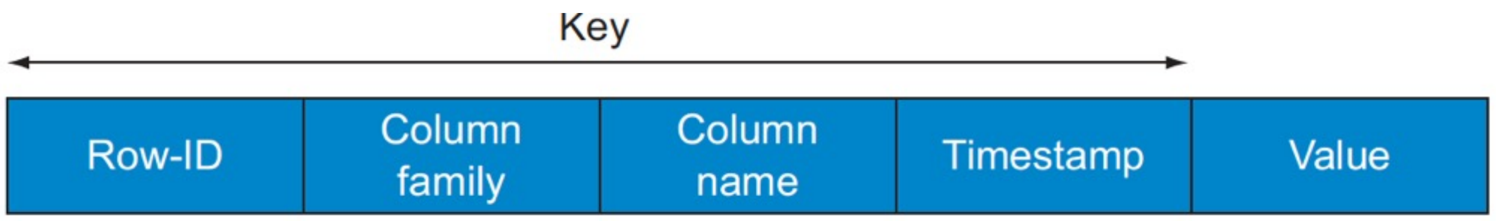
- **Drawbacks**
  - Document should be contained with everything relevant
  - Denormalising the data is common (redundant embedded data)

Columnar (Column-oriented / column-family)

- Columnar stores use row and column identifiers as general purposes keys for data lookup
- They lack features you may expect to found in traditional databases (data types, secondary indexes, triggers, and query languages)



Column-family Stores

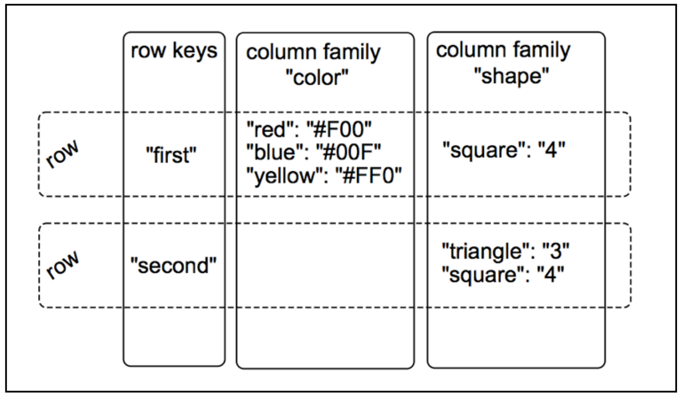


- A column family is used to group similar column names together
    - Column family members are stored together on the filesystem.
    - Tunings and storage specifications are done at the column family level
- A timestamp in the key also allows each cell in the table to store multiple versions of a value over time

Columnar (Column-oriented / column-family)

- **Advantages**
  - Columnar databases store like data by columns, rather than keeping data together by rows
  - Adding Columns is trivial
  - Versioning of data is often built-in
  - Good for Big Data problems, easy to distribute

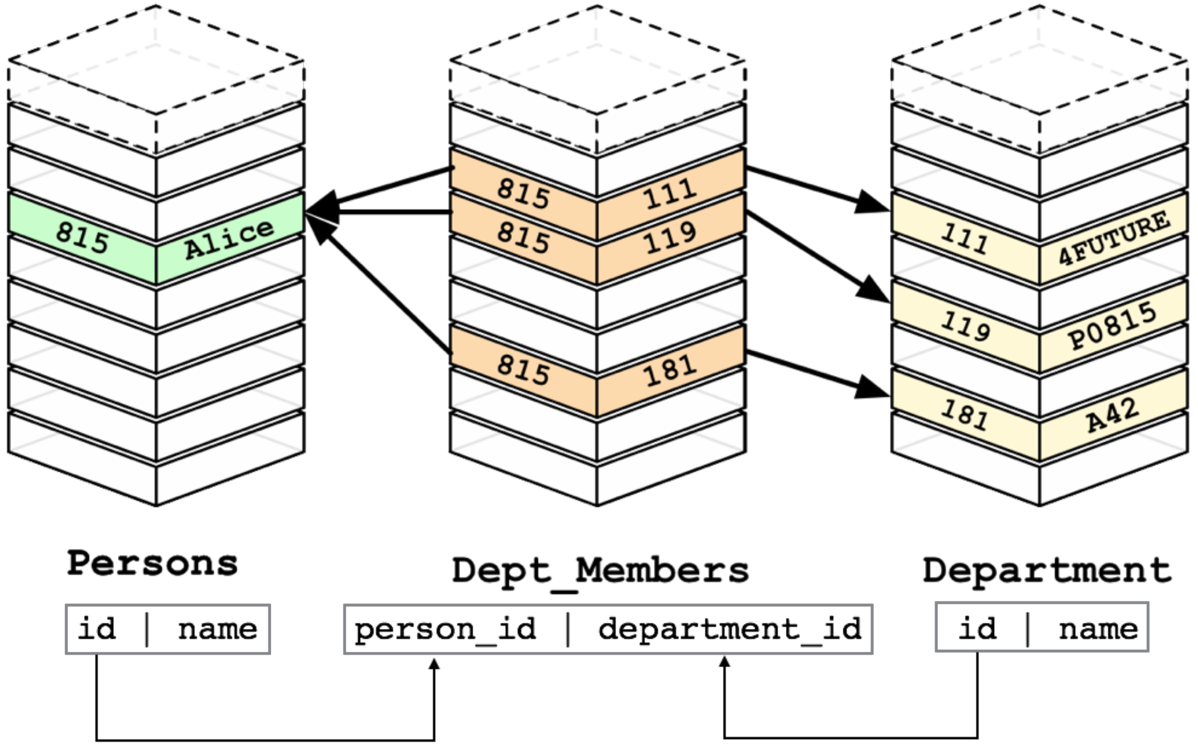
- **Disadvantages**
  - Design of schema based on how the data will be queried
  - Not the best solution for OLTP (Online Transaction Processing), where all columns in a row are often accessed and updated



Example:

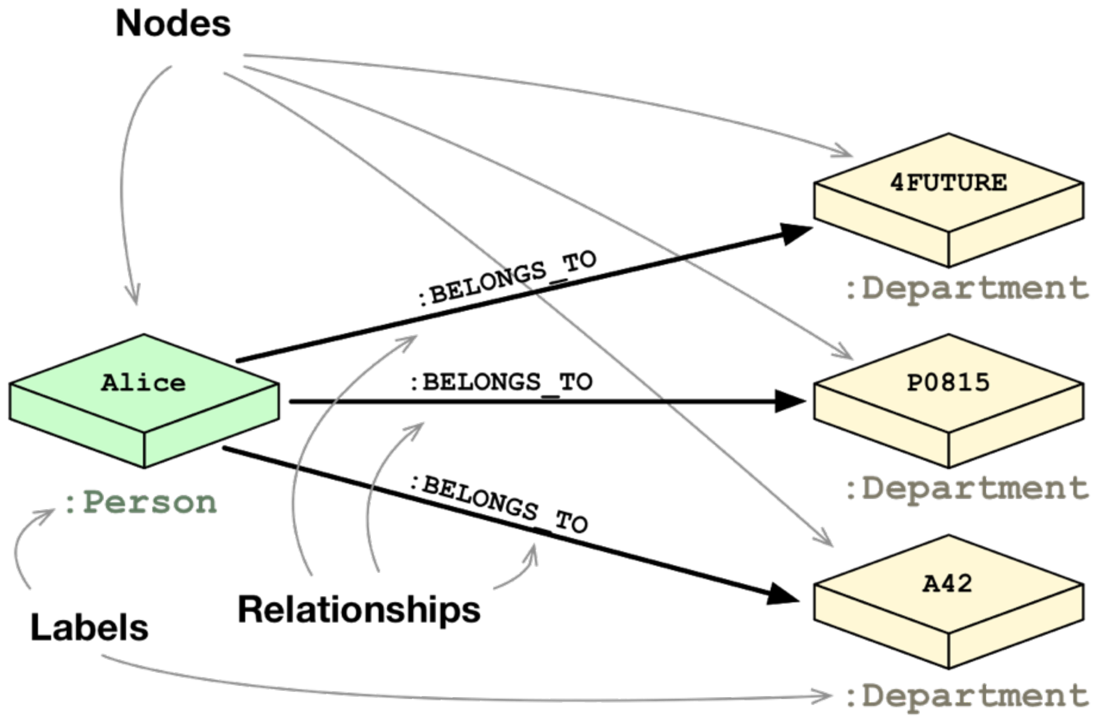
***HBase** (https://hbase.apache.org/)*

Graph Databases



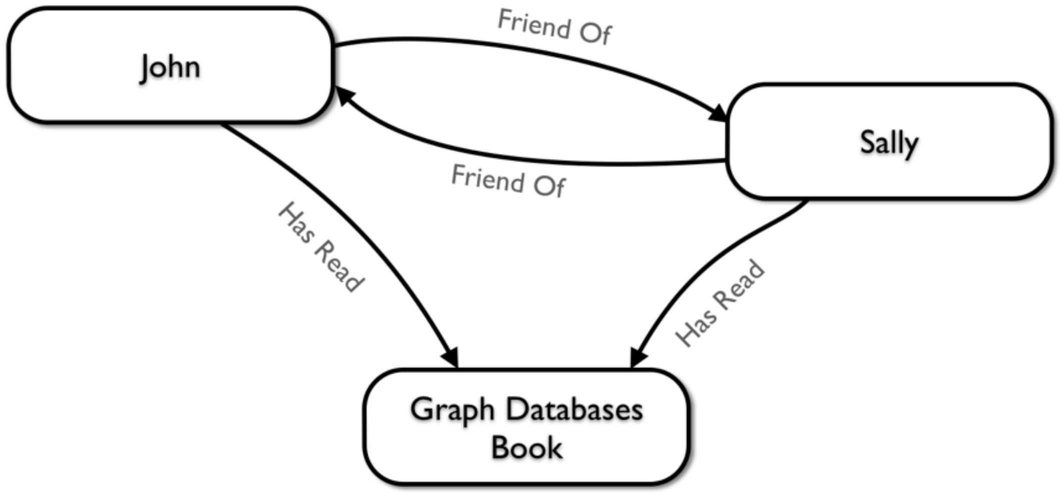
http://neo4j.com/developer/graph-db-vs-rdbms/

Graph Databases



neo4j.com/developer/graph-db-vs-rdbms/

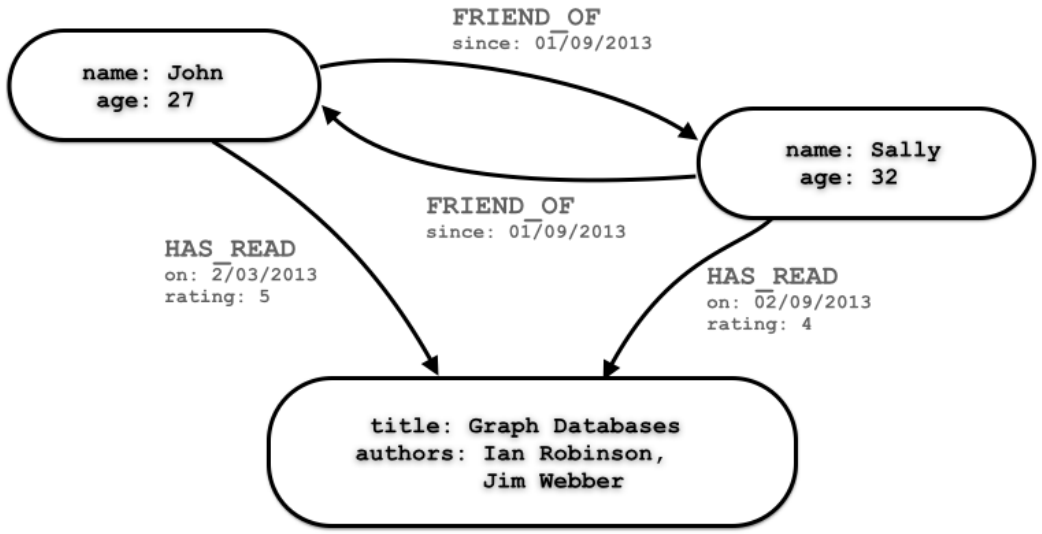
Graph Databases



•*Similar to ER Diagram -* ***except data instances*** are stored on a graph

http://neo4j.com/developer/guide-data-modeling/

Graph Databases



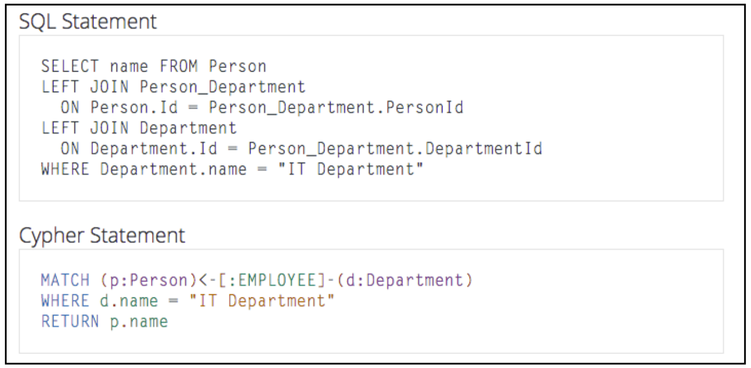
- *Attributes stored on* ***nodes*** *and* ***edges***

http://neo4j.com/developer/guide-data-modeling/

Graph Databases

- Graph databases store data on a graph with interrelation of data
- Queries based on *traversing nodes* (following edges)
- Can represent specific use cases well, e.g. social networking applications, recommendation engines, access control lists, geographic data
- Good for object-oriented systems (references between objects)

- **Drawbacks**
    - Bad performance when partitioning a network (i.e. distributing across nodes with network hops)
    - Scales poorly
    - Not the best choice for tabular analytical data

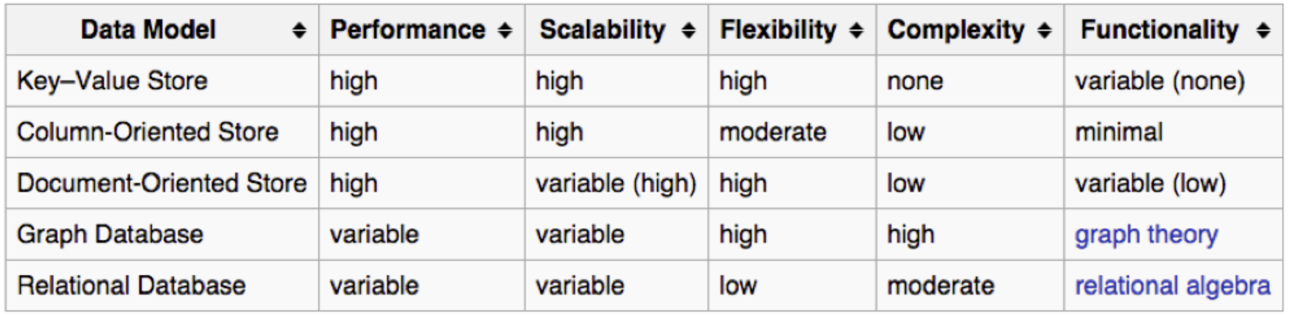

SQL Statement

```
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId
WHERE Department.name = "IT Department"
```

Cypher Statement

```
MATCH (p:Person)<-[:EMPLOYEE]-(d:Department)
WHERE d.name = "IT Department"
RETURN p.name
```

Summary



Data Model	Performance	Scalability	Flexibility	Complexity	Functionality
Key–Value Store	high	high	high	none	variable (none)
Column-Oriented Store	high	high	moderate	low	minimal
Document-Oriented Store	high	variable (high)	high	low	variable (low)
Graph Database	variable	variable	high	high	graph theory
Relational Database	variable	variable	low	moderate	relational algebra

Source: http://www.slideshare.net/bscofield/nosql-codemash-2010

- Despite progress in developing noSQL technologies, such databases are the topic of current research, so they are still evolving