CIS 8045

Unstructured Data Management
NoSQL

Ling Xue, Associate Professor

CIS Department
J Mack Robinson College of Business
Georgia State University

What is unstructured data?

a generic label for describing any corporate information that is not in a database

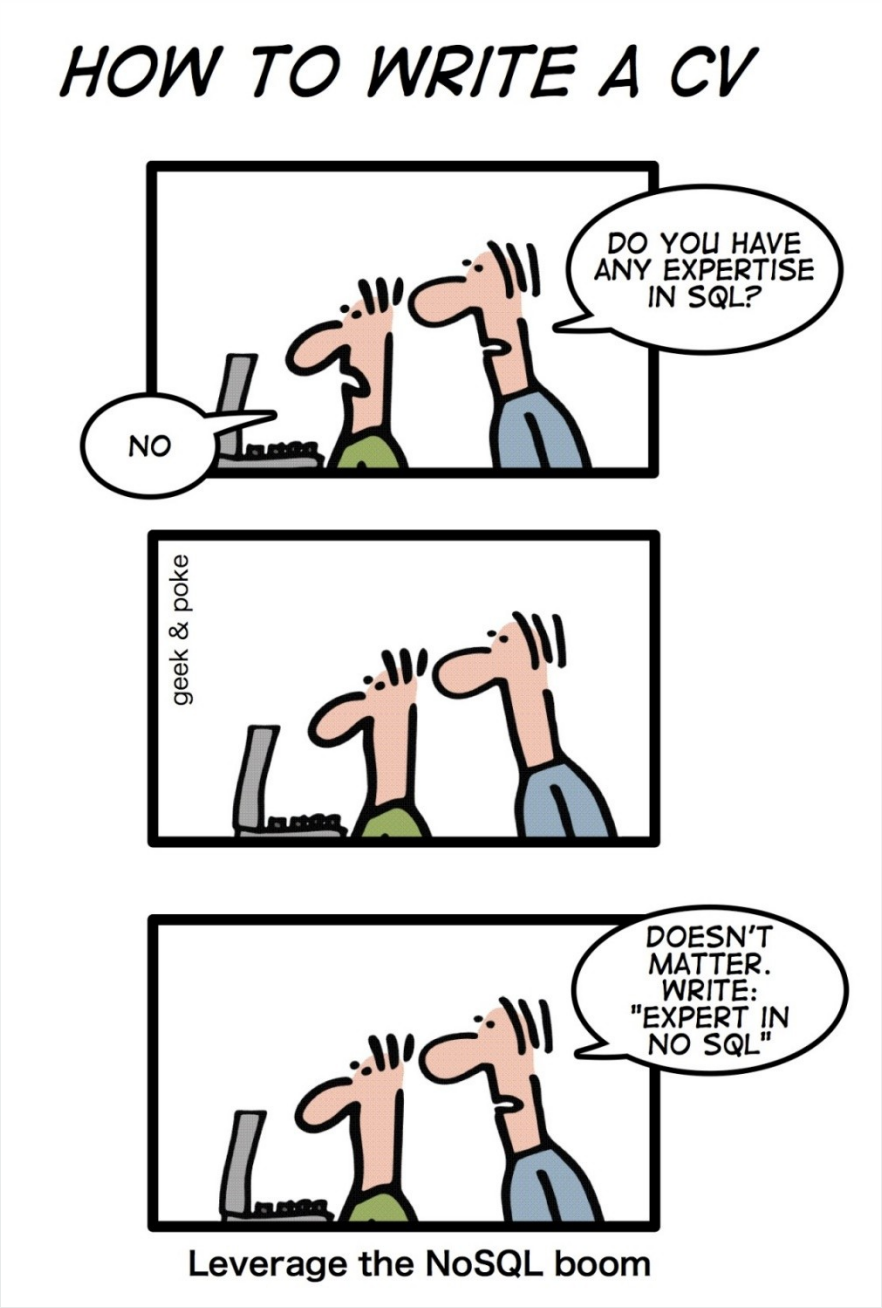
information that either does not have a pre-defined data model or is not organized in a pre-defined manner

unstructured data : Data that does not reside in fixed locations

any data that has no identifiable structure

NoSQL

- NoSQL is a generic term used to refer to any data store that **does not follow the traditional RDBMS model**— specifically, the data is non-relational and it does not use SQL as the query language. It is used to refer to the databases that attempt to solve the problems of **scalability and availability** against that of atomicity or consistency.



CAP Theorem (1/2)

- It is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

  - **C**onsistency (all nodes see the same data at the same time)
  - **A**vailability (a guarantee that every request receives a response about whether it was successful or failed)
  - **P**artition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system)

A distributed system can satisfy any two of these guarantees at the same time, but not all three.

CAP Theorem (2/2)

- In other words, **CAP** can be expressed as "If the network is broken, your database won't work"
  - "won't work" = down OR inconsistent
- In RDBMS we do not have **P** (network partitions)
  - **C**onsistency and **A**vailability are achieved
- In NoSQL we want to have **P**
  - Need to select either **C** or **A**
  - Drop A -> Accept waiting until data is consistent
  - Drop C -> Accept getting inconsistent data sometimes

ACID vs BASE

- Scalability and better performance of NoSQL is achieved by sacrificing **ACID** compatibility.

Atomic, **C**onsistent, **I**solated, **D**urable

- NoSQL is having **BASE** compatibility instead.

Basically **A**vailable, **S**oft state, **E**ventual consistency

ACID – Focused on by Traditional SQL DBs

- **A**tomicity. All of the operations in the transaction will complete, or none will.
- **C**onsistency. Transactions never observe or result in inconsistent data.
- **I**solation. The transaction will behave as if it is the only operation being performed upon the database (i.e. uncommitted transactions are isolated)
- **D**urability. Upon completion of the transaction, the operation will not be reversed (i.e. committed transactions are permanent)

BASE

- **Basically Available: Each request is guaranteed a response— successful or failed execution.**
  - Use replication and sharing to reduce the likelihood of data unavailability and use sharing, or partitioning the data among many different storage servers, to make any remaining failures partial.
- **Soft State**: **The state of the system may change over time.**
  - NoSQL systems allow data to be inconsistent and relegate designing around such inconsistencies to application developers.
- **Eventually Consistent: The database may be momentarily inconsistent but will be consistent eventually.**
  - Although applications must deal with instantaneous inconsistency, NoSQL systems ensure that at some future point in time the data assumes a consistent state.

NoSQL – What More?

- **Schemaless data representation** : This means that you don’t have to think too far ahead to define a structure and you can continue to evolve over time— including adding new fields or even nesting the data, for example, in case of JSON representation.
- **Development time**: One doesn’t have to deal with complex SQL queries.
- **Speed**: Even with the small amount of data that you have, if you can deliver in milliseconds rather than hundreds of milliseconds— especially over mobile and other intermittently connected devices— you have much higher probability of winning users over.
- **Plan ahead for scalability**: Your application can be quite elastic— it can handle sudden spikes of load.

NoSQL Databases

- Key-value

- Graph database

- Document-oriented

- Column family

NoSQL Databases

- **Document-oriented**
  - Semi-structured *documents* with key-value pairs
  - Can be retrieved with queries
- **Graph DB**
  - Graph Databases are built with nodes, relationships between nodes (edges) and the properties of nodes.
- **Key-Value**
  - based on a hash table where there is a unique key and a pointer to a particular item of data
- **Column Family**
  - Store and process very large amounts of data distributed over many machines.
  - Keys point to multiple columns

NoSQL: Downside?

- **Inability to define (many-to-many) relationships**
- **Absence of transactions** (Neo4j is an exception)
- **Unavailability of ACID properties in operations** (CouchDB and Neo4j are the exceptions) Using Cassandra or CouchDB can be overkill if we compare them to, for example, MySQL or PostgreSQL— they do not unleash their true power in a single-machine installation
- **Absence of support for JOIN and cross-entity query**, though document-oriented stores support it by way of MapReduce but the efforts may be substantial as the queries get complex

Document store

RDBMS		MongoDB
Database	→	Database
Table, View	→ →	Collection
Row	→	Document (JSON, BSON)
Column	→	Field
Index	→	Index
Join	→ →	Embedded Document
Foreign Key		Reference
Partition		Shard

Document store

RDBMS		MongoDB
Database	⇒	Database
Table, View	⇒ ⇒	Collection
Row	⇒	Document (BSON)
Column	⇒	Field
Index	⇒	Index
Join	⇒ ⇒	Embedded Document
Foreign Key		Reference
Partition		Shard

```
> db.user.findOne({age:39})
{
        "_id" :
ObjectId("5114e0bd42…"),
        "first" : "John",
        "last" : "Doe",
        "age" : 39,
        "interests" : [
                "Reading",
                "Mountain Biking ]
        "favorites": {
                "color": "Blue",
                "sport": "Soccer"}
}
```