THE UNIVERSITY OF MELBOURNE

INFO20003: Database Systems

Dr Renata Borovica-Gajic

Lecture 21

NoSQL Databases

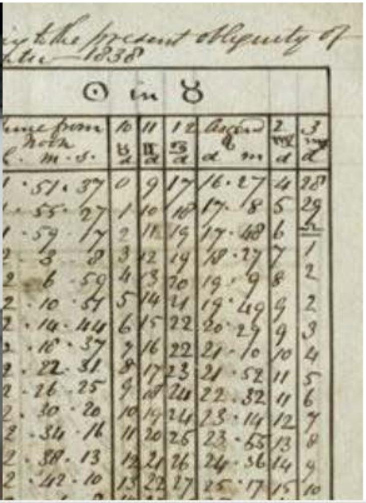
Week 11

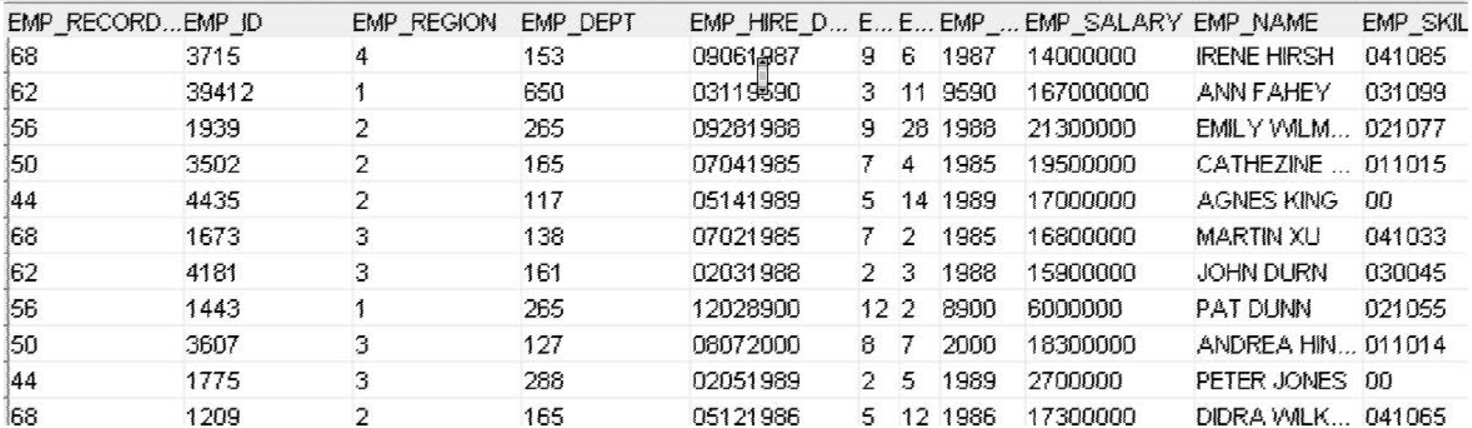
Learning Objectives

- By the end of this session, you should be able to:
  - Define what Big Data is
  - Describe why databases go beyond relational DBs
  - Understand why we need NoSQL
  - Types of NoSQL
  - CAP theorem

* material in this lecture is drawn from http://martinfowler.com/books/nosql.html, including talk at GOTO conference 2012
and Thoughtworks article at https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Much of business data is tabular





EMP_RECORD...	EMP_ID	EMP_REGION	EMP_DEPT	EMP_HIRE_D...	E...	E...	EMP_...	EMP_SALARY	EMP_NAME	EMP_SKIL
68	3715	4	153	09061987	9	6	1987	14000000	IRENE HIRSH	041085
62	39412	1	650	03119590	3	11	9590	167000000	ANN FAHEY	031099
56	1939	2	265	09281988	9	28	1988	21300000	EMILY WILM...	021077
50	3502	2	165	07041985	7	4	1985	19500000	CATHEZINE ...	011015
44	4435	2	117	05141989	5	14	1989	17000000	AGNES KING	00
68	1673	3	138	07021985	7	2	1985	16800000	MARTIN XU	041033
62	4181	3	161	02031988	2	3	1988	15900000	JOHN DURN	030045
56	1443	1	265	12028900	12	2	8900	6000000	PAT DUNN	021055
50	3607	3	127	08072000	8	7	2000	18300000	ANDREA HIN...	011014
44	1775	3	288	02051989	2	5	1989	2700000	PETER JONES	00
68	1209	2	165	05121986	5	12	1986	17300000	DIDRA WILK...	041065

The dominance of the relational model

- Pros of relational databases
  - simple, can capture (nearly) any business use case
  - can integrate multiple applications via shared data store
  - standard interface language SQL
  - ad-hoc queries, across and within "data aggregates"
  - fast, reliable, concurrent, consistent
- Cons of relational databases
  - Object Relational (OR) impedance mismatch
  - not good with big data
  - not good with clustered/replicated servers

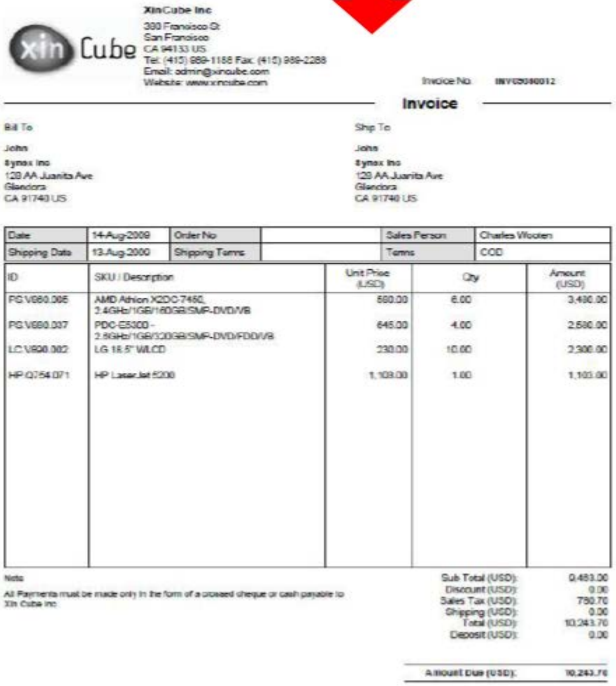
(for big data)

table can only have columns → fairly simple
lacks flexibility
(compare to "object").
"class"

- Adoption of NoSQL driven by "cons" of Relational
- but 'polyglot persistence' = Relational will not go away

But some data is not inherently tabular

One business object (in aggregate form) is stored across many relational tables.



XinCube Inc
380 Francisco St
San Francisco
CA 94133 US
Tel: (415) 889-1188 Fax: (415) 989-2288
Email: admin@xincube.com
Website: www.xincube.com

Invoice No. INV05080012

Invoice

Bill To

John
Synex Inc
128 AA Juanita Ave
Glendora
CA 91740 US

Ship To

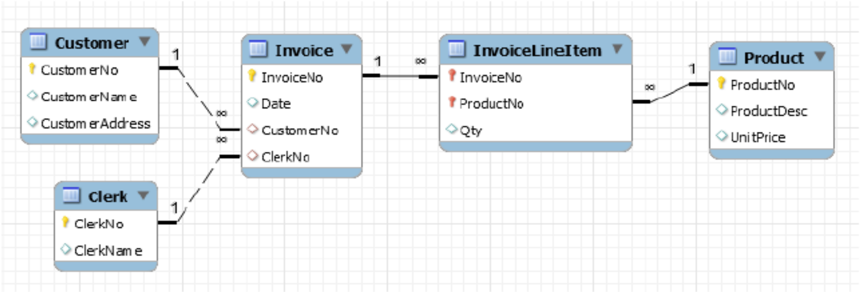
John
Synex Inc
128 AA Juanita Ave
Glendora
CA 91740 US

Date	14-Aug-2009	Order No		Sales Person	Charles Wooten
Shipping Data	13-Aug-2000	Shipping Terms		Terms	COD

ID	SKU / Description	Unit Price (USD)	Qty	Amount (USD)
PS.V860.065	AMD Athlon X2DC-7450, 2.4GHz/1GB/160GB/SMP-DVD/VB	580.00	6.00	3,480.00
PS.V880.037	PDC-E5300 - 2.6GHz/1GB/320GB/SMP-DVD/FDD/VB	645.00	4.00	2,580.00
LC.V890.002	LG 18.5" WLCD	230.00	10.00	2,300.00
HP.Q754.071	HP LaserJet 5200	1,103.00	1.00	1,103.00

Note
All Payments must be made only in the form of a crossed cheque or cash payable to Xin Cube Inc

Sub Total (USD):	9,483.00
Discount (USD):	0.00
Sales Tax (USD):	760.70
Shipping (USD):	0.00
Total (USD):	10,243.70
Deposit (USD):	0.00
Amount Due (USD):	10,243.70



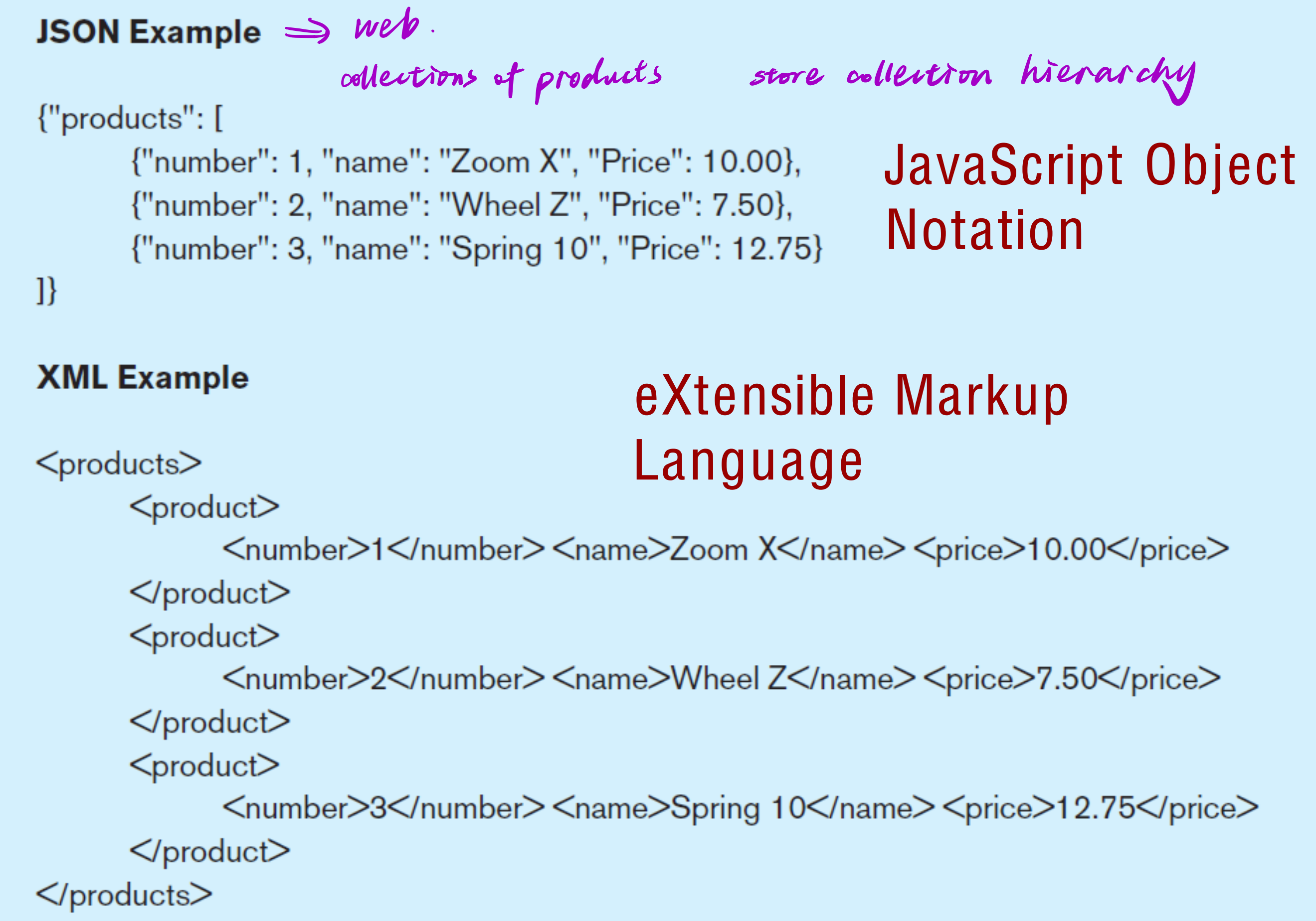
This enables analytical queries like:

select productno, sum(qty) from InvoiceLineItem
group by productno;

Inefficient to join lots of tables

But there is a lot of work to dissemble and reassemble the aggregate.

Data in Aggregate form: Examples of JSON and XML



JSON Example ⇒ web.

collections of products   store collection hierarchy

```
{"products": [
        {"number": 1, "name": "Zoom X", "Price": 10.00},
        {"number": 2, "name": "Wheel Z", "Price": 7.50},
        {"number": 3, "name": "Spring 10", "Price": 12.75}
]}
```

JavaScript Object Notation

XML Example

eXtensible Markup Language

```
<products>
        <product>
                <number>1</number> <name>Zoom X</name> <price>10.00</price>
        </product>
        <product>
                <number>2</number> <name>Wheel Z</name> <price>7.50</price>
        </product>
        <product>
                <number>3</number> <name>Spring 10</name> <price>12.75</price>
        </product>
</products>
```

Big Data and its 3Vs

- Data that exist in very large volumes and many different varieties (data types) and that need to be processed at a very high velocity (speed).
  - **Volume** – much larger quantity of data than typical for relational databases
  - **Variety** – lots of different data types and formats
  - **Velocity** – data comes at very fast rate (e.g. mobile sensors, web click stream)

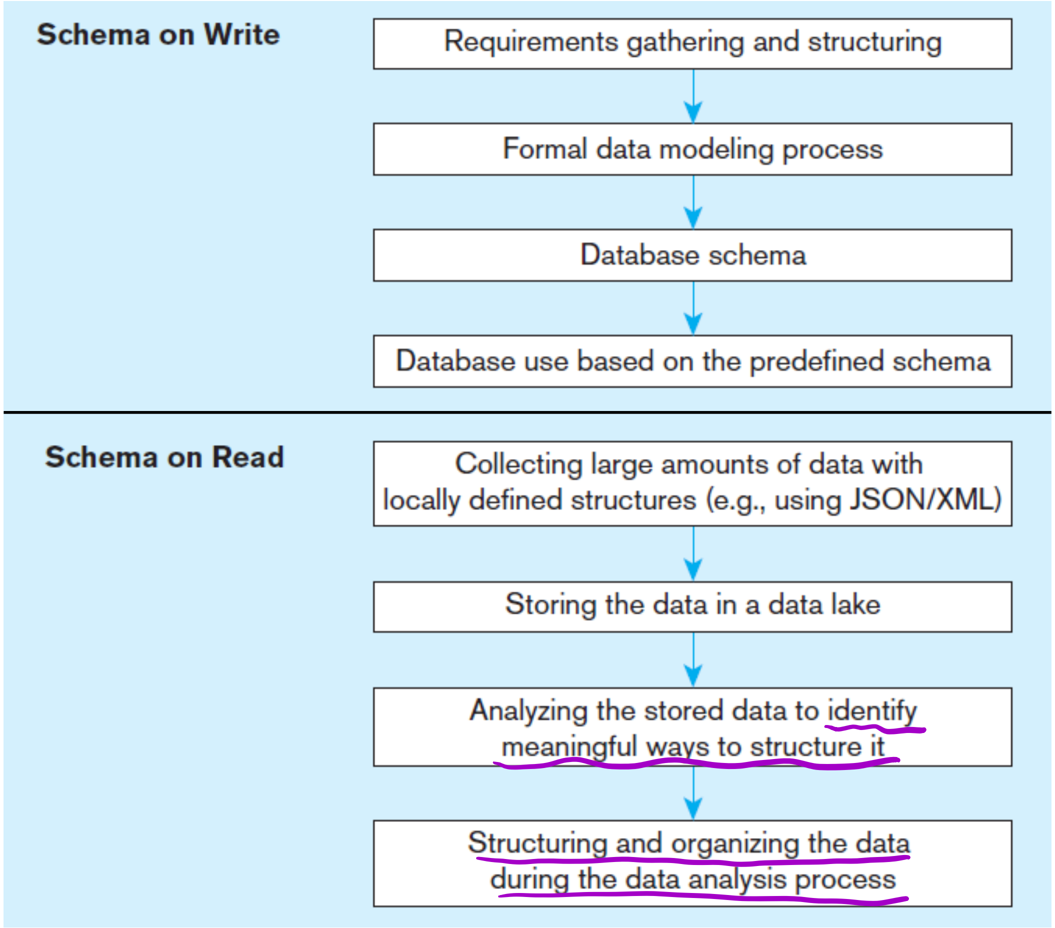
Big Data Characteristics

- Schema on Read, rather than Schema on Write
  - Schema on Write– preexisting data model, how traditional databases are designed (relational databases)
  - Schema on Read – data model determined later, depends on how you want to use it (XML, JSON)
  - Capture and store the data, and worry about how you want to use it later

- Data Lake
  - A large integrated repository for internal and external data that does not follow a predefined schema
  - Capture everything, dive in anywhere, flexible access

Jeff Hoffer, Ramesh Venkataraman and Heikki Topi , Modern Database Management: Global Edition

Schema on write vs. schema on read



Schema on Write

Requirements gathering and structuring

Formal data modeling process

Database schema

Database use based on the predefined schema

Schema on Read

Collecting large amounts of data with locally defined structures (e.g., using JSON/XML)

Storing the data in a data lake

Analyzing the stored data to identify meaningful ways to structure it

Structuring and organizing the data during the data analysis process

Traditional database design

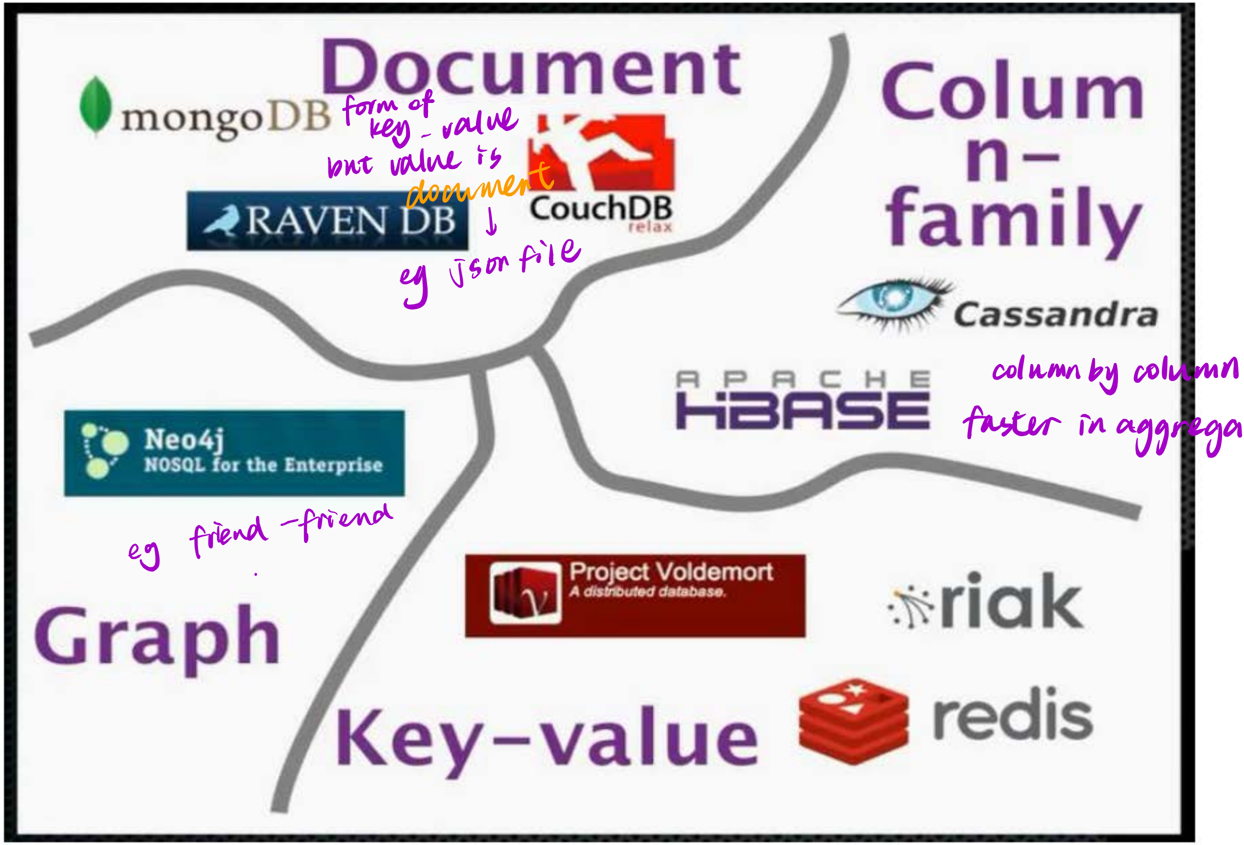
The big data approach

NoSQL database properties

- Features
  - Doesn’t use relational model or SQL language
  - Runs well on distributed servers
  - Most are open-source
  - Built for the modern web
  - Schema-less (though there may be an "implicit schema")
  - Supports schema on read
  - Not ACID compliant → cannot running transaction
  - ‘Eventually consistent'
- Goals
  - support hierarchy / tree
  - to improve programmer productivity (OR mismatch)
  - to handle larger data volumes and throughput (big data)

from NoSQL Databases: An Overview
by Pramod Sadalage, Thoughtworks(2014)

Types of NoSQL databases



form of key-value but value is document ↓ eg json file

column by column faster in aggregation

eg friend-friend

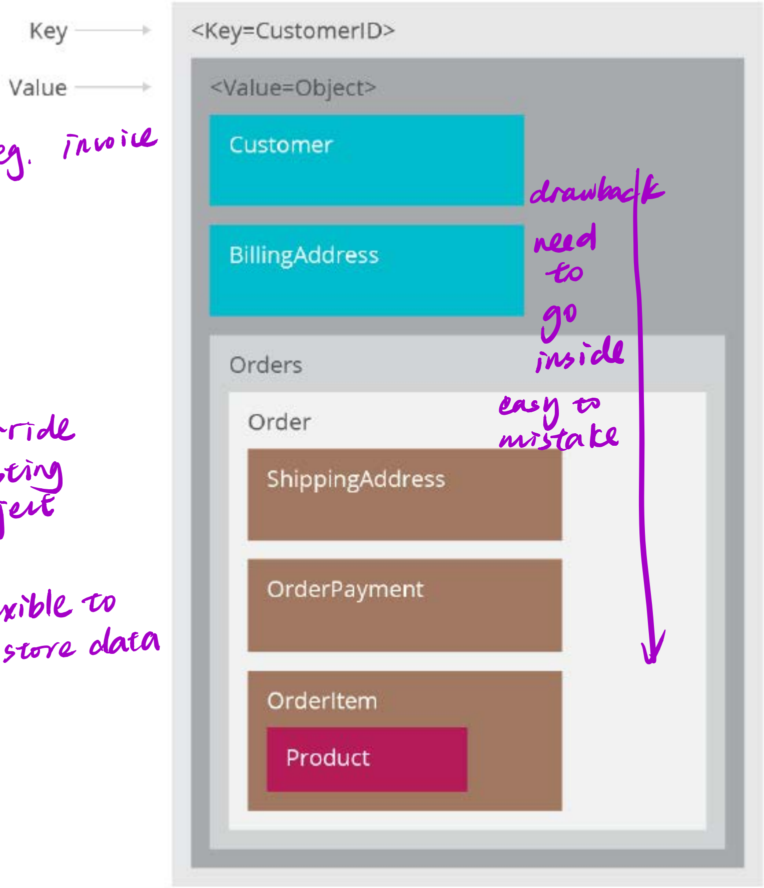
(diagram from Martin Fowler)

Types of NoSQL: key-value stores

- Key = primary key
- Value = anything (number, array, image, JSON) *–the application is in charge of interpreting what it means*
- Operations: *Put* (for storing), *Get* and *Update* → override existing object
- **Examples**: Riak, Redis, Memcached, BerkeleyDB, HamsterDB, Amazon DynamoDB, Project Voldemort, Couchbase

eg. invoice

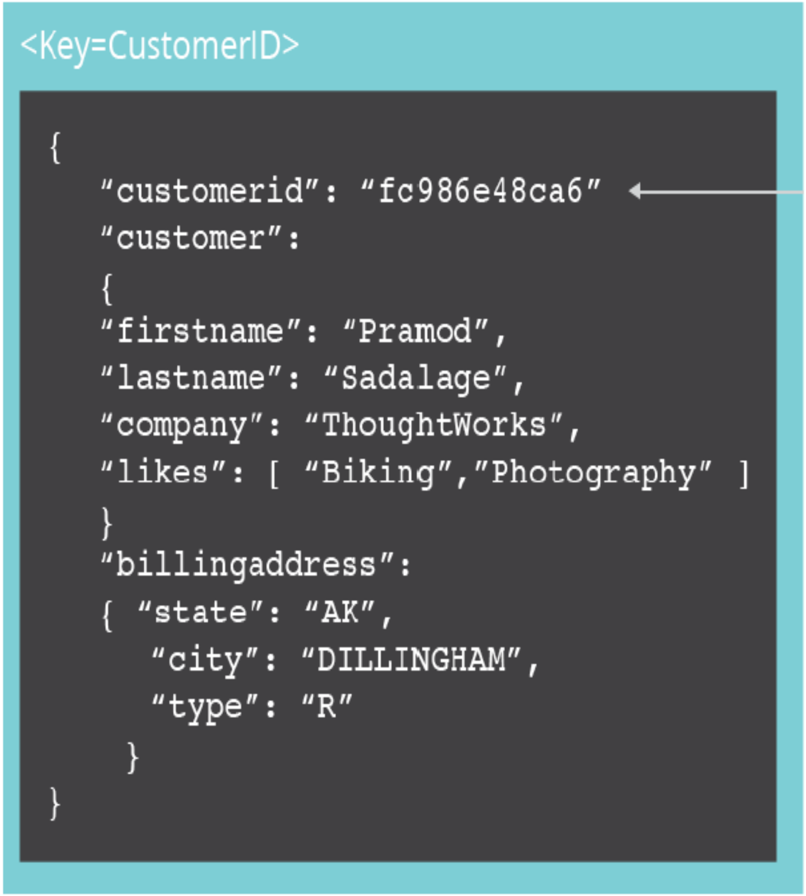
flexible to store data



drawback need to go inside easy to mistake

Types of NoSQL: document databases

- Similar to a key-value store except that the document is "examinable" by the databases, so its *content can be queried*, and parts of it updated
- Document = JSON file
- **Examples:** MongoDB, CouchDB, Terrastore, OrientDB, RavenDB

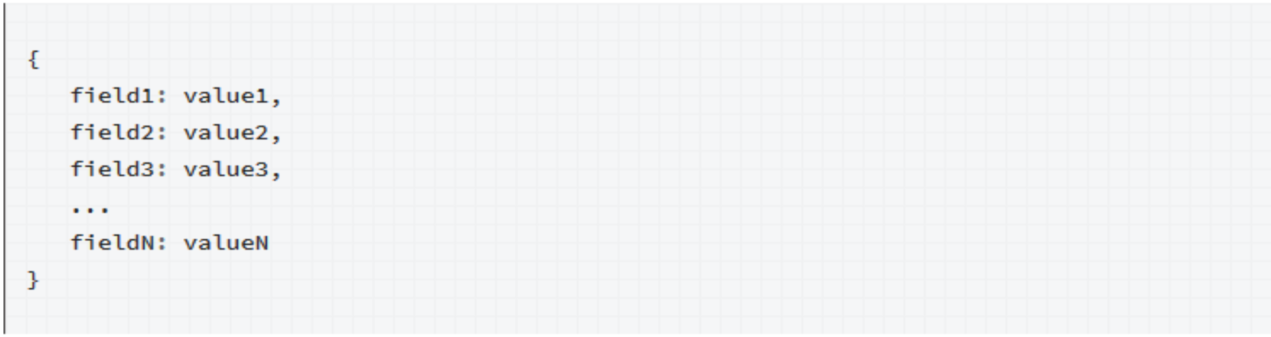


<Key=CustomerID>

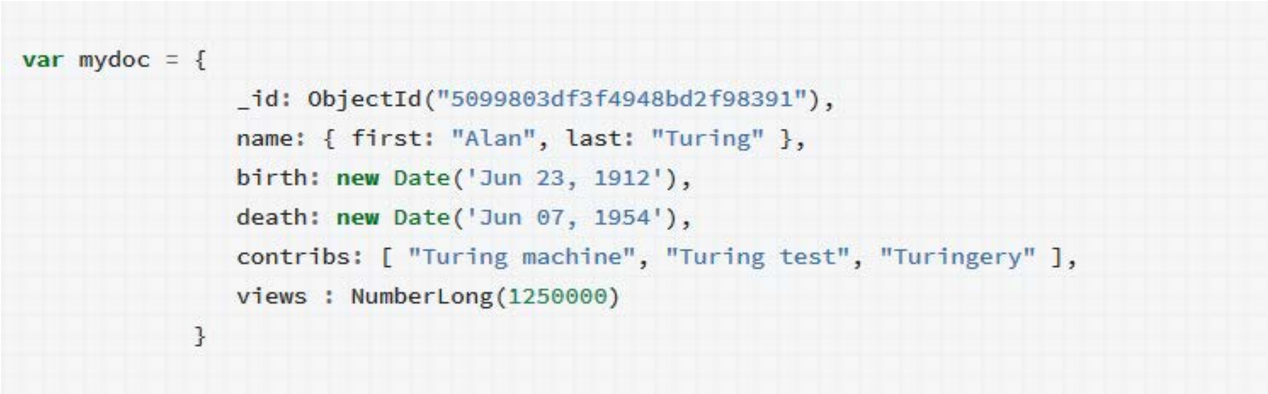
```
{
    “customerid”: “fc986e48ca6” ←
    “customer”:
    {
    “firstname”: “Pramod”,
    “lastname”: “Sadalage”,
    “company”: “ThoughtWorks”,
    “likes”: [ “Biking”,”Photography” ]
    }
    “billingaddress”:
    { “state”: “AK”,
      “city”: “DILLINGHAM”,
      “type”: “R”
     }
}
```

MongoDB Document Structure

- MongoDB documents are composed of field-and-value pairs



```
{
    field1: value1,
    field2: value2,
    field3: value3,
    ...
    fieldN: valueN
}
```



```
var mydoc = {
               _id: ObjectId("5099803df3f4948bd2f98391"),
               name: { first: "Alan", last: "Turing" },
               birth: new Date('Jun 23, 1912'),
               death: new Date('Jun 07, 1954'),
               contribs: [ "Turing machine", "Turing test", "Turingery" ],
               views : NumberLong(1250000)
            }
```

THE UNIVERSITY OF MELBOURNE

MongoDB *document store*

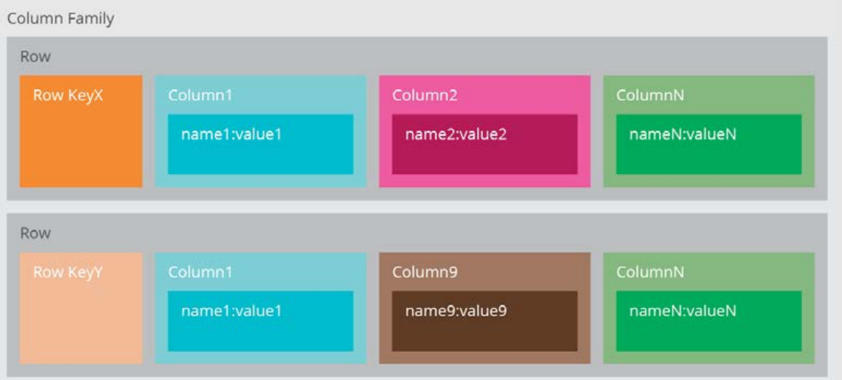
start the mongodb server, then start the mongo shell with "mongo"

```
show dbs// show a list of all databases
use test// use the database called 'test'
show collections// show all collections in the database 'test'
db.students.insert( {name: "Jack", born: 1992} )// add a doc to collection
db.students.insert( {name: "Jill", born: 1990} )// add a doc to collection
db.students.find()// list all docs in students
db.students.find( {name: "Jill"} )// list all docs where name field = 'Jill'
db.students.update( {name: "Jack"}, {$set: {born: 1990}} ) // change Jack's year
db.students.remove( {born: 1990} ) // delete docs where year = 1990

// now insert complex documents from file –note repeating group, no schema
db.students.find().forEach(printjson)// print all docs in neat JSON format
db.students.find( {born:1990}, {name: true} )// print names for all born in 1990
db.students.update( {id:222222}, {$addToSet:
{subjects: {subject: "English", result: "H1"}}} )          Update data deep in hierarchy
db.students.find( {id:222222}, {_id:false, subjects:true} ).forEach(printjson)

db.students.insert( {name: “John", color: “blue”} )
// add a new student – different schema but still works
```

Types of NoSQL: column families

- Columns rather than rows are stored together on disk.
- Makes analysis faster, as less data is fetched.
- This is like automatic vertical partitioning.
- Related columns grouped together into ‘families’.
- **Examples**: Cassandra, BigTable, HBase

https://www.youtube.com/watch?v=8KGVFB3kVHQ



Aggregate-oriented databases

- *Key-value*, *document store* and *column-family* are "aggregate-oriented- store business object in its entirety" databases (in Fowler's terminology)

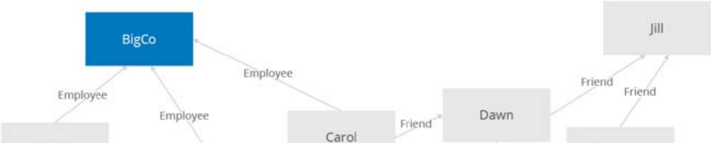
- Pros:
  - *entire* aggregate of data is stored together (no need for transactions)
  - efficient storage on clusters / distributed databases

- Cons:
  - hard to analyse across subfields of aggregates
  - e.g. sum over products instead of orders

THE UNIVERSITY OF MELBOURNE

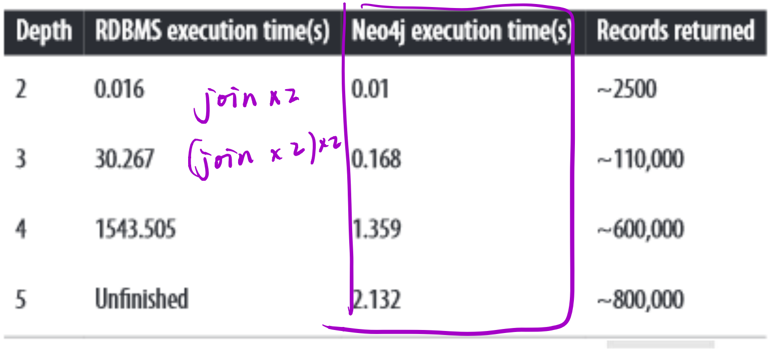
Types of NoSQL: graph databases

- A 'graph' is a node-and-arc network
- Social graphs (e.g. friendship graphs) are common examples linked in , facebook.
- Graphs are difficult to program in relational DB
- A *graph DB* stores entities and their relationships
- Graph queries deduce knowledge from the graph



- **Examples**:
  - Neo4J
  - Infinite Graph
  - OrientDBv
  - FlockDB
  - TAO

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

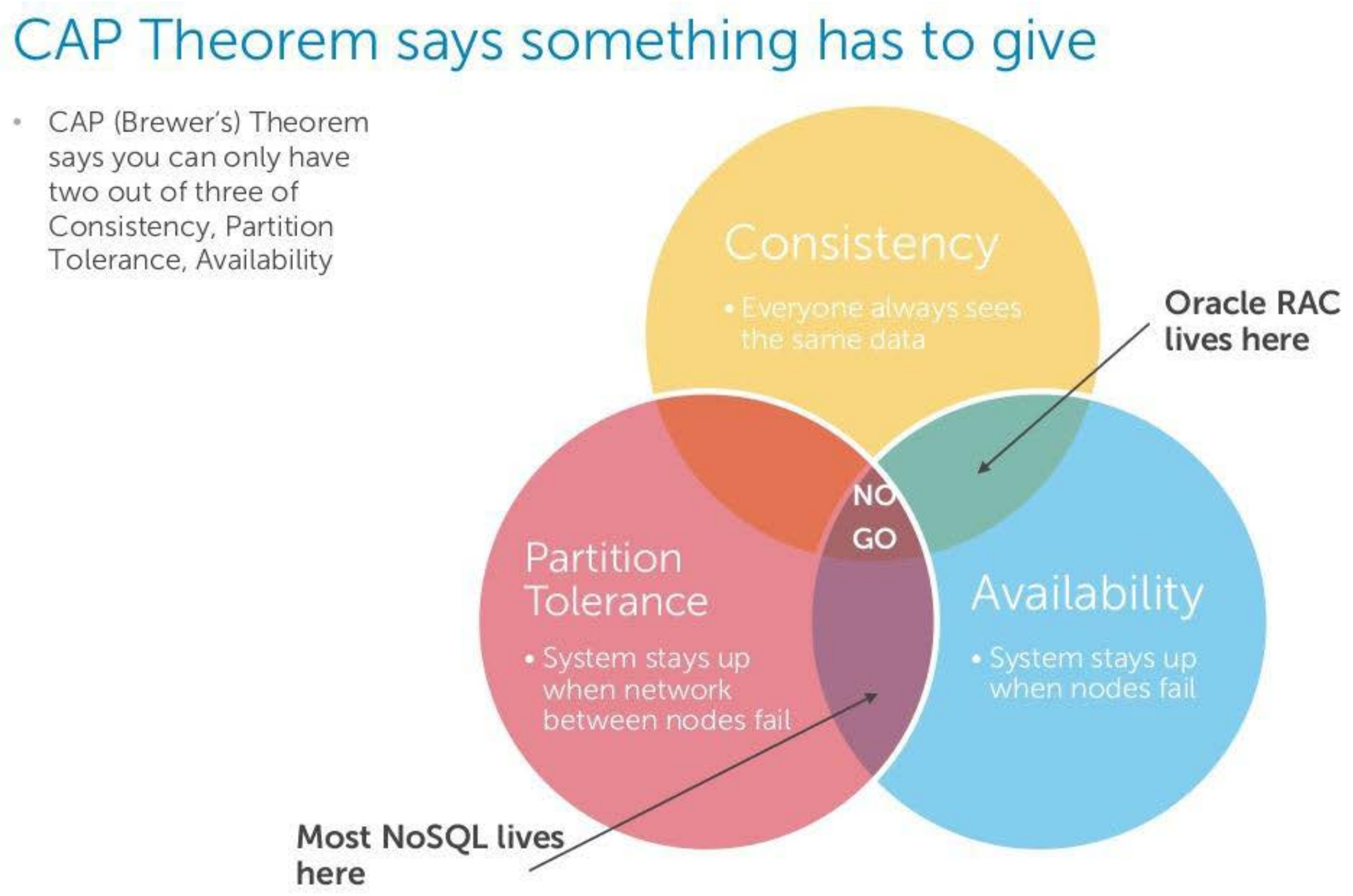


Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016 join x2	0.01	~2500
3	30.267 (join x2)x2	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Summary: NoSQL Classifications

- Key-value stores flexible.
  - A simple pair of a key and an associated collection of values. Key is usually a string. The database has no knowledge of the structure or meaning of the values.
- Document stores less flexible , easy to manipulate
  - Like a key-value store, but "document" goes further than "value". The document is structured, so specific elements can be manipulated separately.
- Column-family stores
  - Data is grouped in "column groups/families" for efficiency reasons.
- Graph-oriented databases
  - Maintain information regarding the relationships between data items. Nodes with properties.

Distributed data: the CAP theorem



CAP Theorem says something has to give

- CAP (Brewer's) Theorem says you can only have two out of three of Consistency, Partition Tolerance, Availability

Consistency

- Everyone always sees the same data

Oracle RAC lives here

NO GO

Partition Tolerance

- System stays up when network between nodes fail

Availability

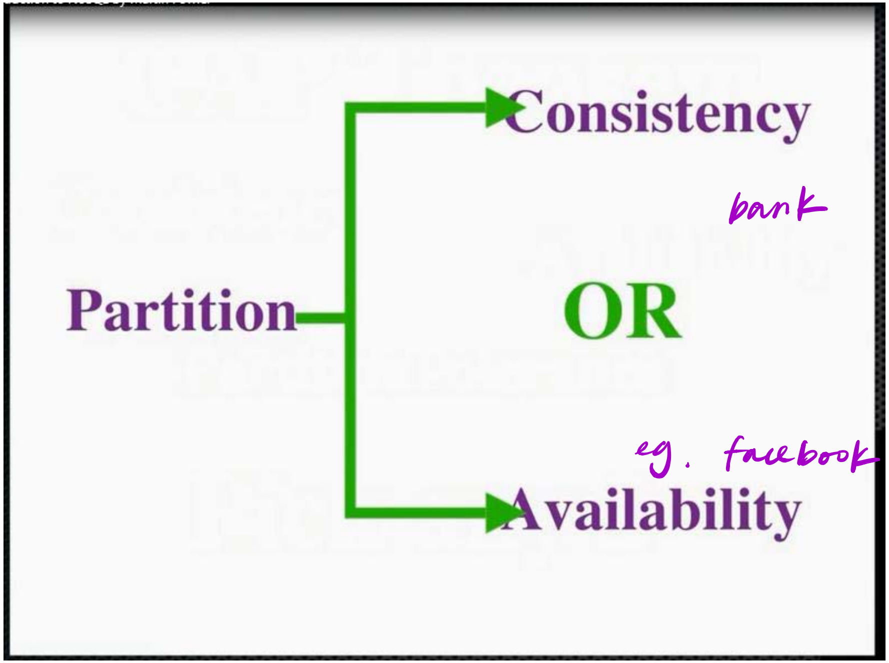
- System stays up when nodes fail

Most NoSQL lives here

CAP theorem: alternative presentation

- Fowler's version of CAP theorem: *If you have a distributed database, when a partition occurs, you must then choose consistency OR availability.*

ANY NoSQL database is distributed database



Partition

Consistency

OR

Availability

bank

eg. facebook

relational database

(comment)

NoSQL

ACID vs BASE

use by NoSQL

ACID (Atomic, Consistent, Isolated, Durable)

relational database
ACID compliant

vs

Base (Basically Available, Soft State, Eventual Consistency)

No SQL doesn't guarantee ACID

- **Basically Available**: This constraint states that the system does guarantee the *availability* of the data; there will be a response to any request. But data may be in an *inconsistent* or *changing* state.
- **Soft state**: The state of the system could change over time -even during times without input there may be changes going on due to 'eventual consistency'.
- **Eventual consistency**: The system will eventually become consistent once it stops receiving input. The data will propagate to everywhere it needs to, sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one.

If you want to know more

More technical details (I won’t ask you these things):

https://www.youtube.com/watch?v=YUWUH_7aWHs&index=11&list=PLdQddgMBv5zHcEN9RrhADq3CBColhY2hl

What is examinable?

- What is big data/NoSQL?
- What are the characteristics of NoSQL databases
- Types of NoSQL databases
- CAP theorem/BASE

Next lecture

- Databases of the future (non-examinable research avenues)