Dr Greg Wadley

THE UNIVERSITY OF MELBOURNE

INFO90002 Database Systems & Information Modelling

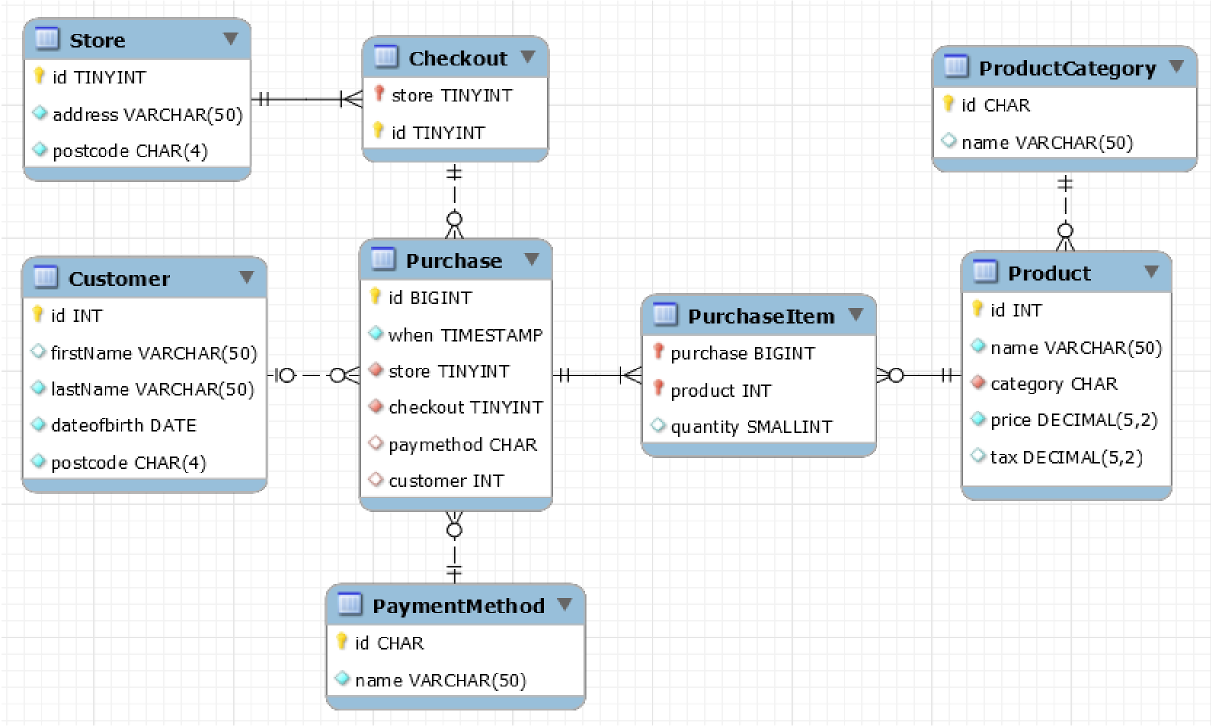
Week 10

NoSQL databases

Today's Session…

- Relational model
- Why NoSQL?
- example 1: MongoDB
- example 2: Firebase
- example 3: Neo4J
- Types of NoSQL
- CAP theorem
- ACID vs BASE
- NoSQL users

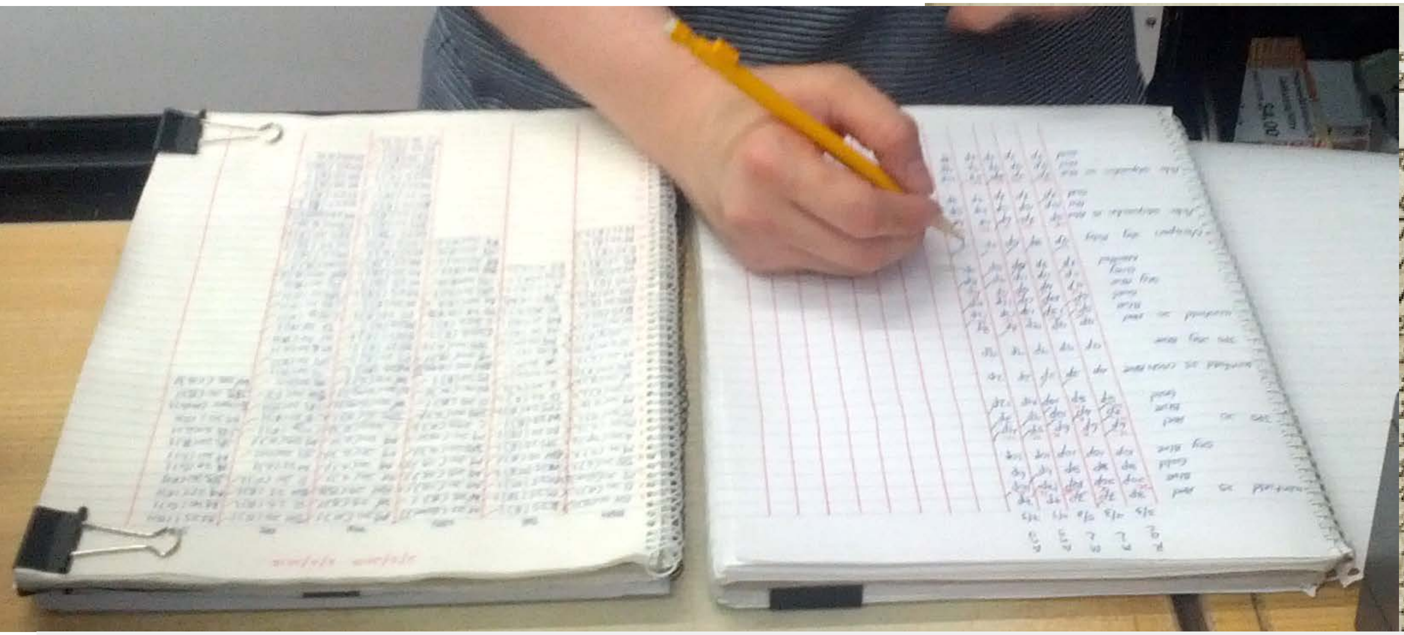
material in this lecture is drawn from http://martinfowler.com/books/nosql.html, including talk at GOTO conference 2012 and Thoughtworks article at https://www.thoughtworks.com/insights/blog/nosql-databases-overview

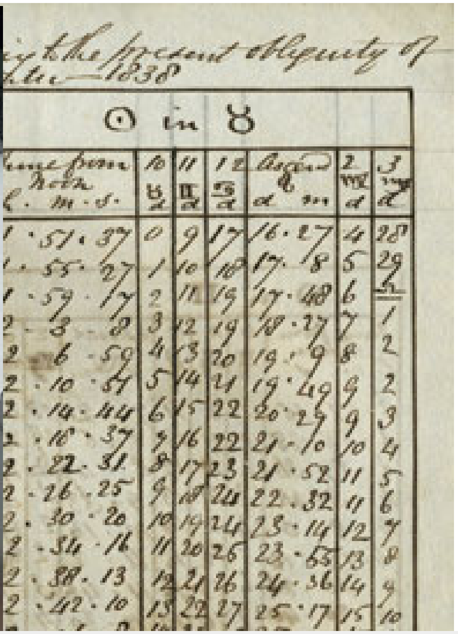


The dominance of the relational model

- Advantages of relational db
  - flexible, suits any data model
  - can integrate multiple applications via shared data store
  - standard within and between organizations
  - standard interface language SQL
  - ad-hoc queries, across and within "data aggregates"
  - fast, reliable, concurrent, consistent
- Problems of relational db
  1. object-relational (OR) “impedance mismatch”
  2. not good with big data
  3. not good with distributed (partitioned) databases
- adoption of NoSQL driven by “cons” of Relational
- but ‘polyglot persistence‘ = Relational will not go away

Problem 1: Much data is tabular …

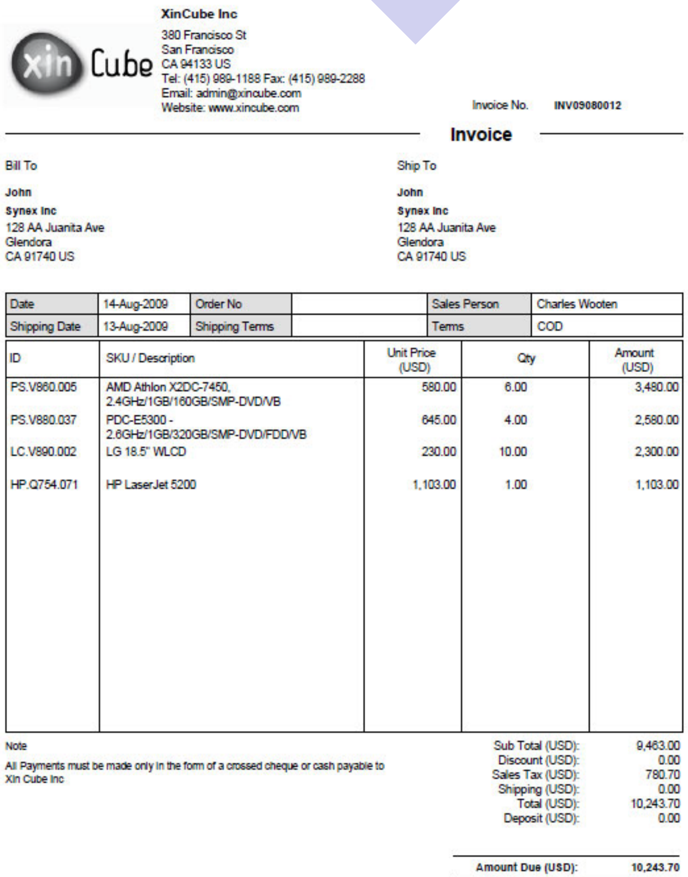




EMP_RECORD...	EMP_ID	EMP_REGION	EMP_DEPT	EMP_HIRE_D...	E...	E...	EMP_...	EMP_SALARY	EMP_NAME	EMP_SKIL
68	3715	4	153	09061987	9	6	1987	14000000	IRENE HIRSH	041085
62	39412	1	650	03119590	3	11	9590	167000000	ANN FAHEY	031099
56	1939	2	265	09281988	9	28	1988	21300000	EMILY WILM...	021077
50	3502	2	165	07041985	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

… but some is not inherently tabular

sometimes one aggregate is stored across many tables



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

Invoice No. INV09080012

Invoice

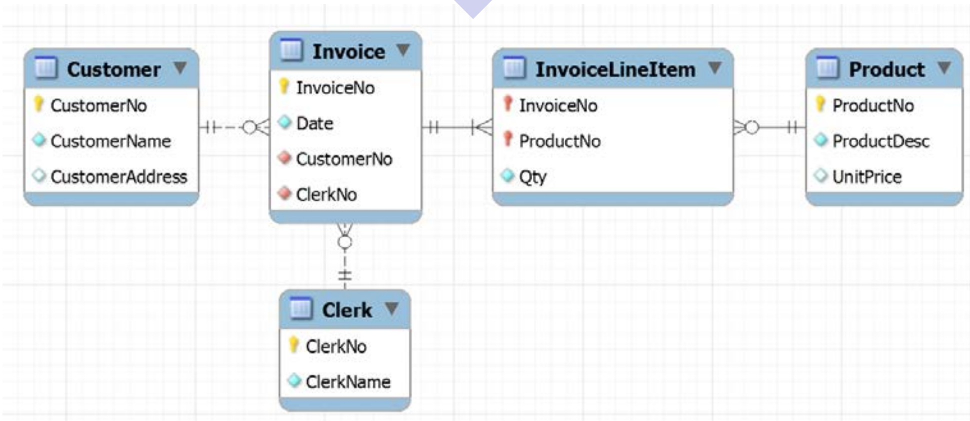
Bill To	Ship To
John Synex Inc 128 AA Juanita Ave Glendora CA 91740 US	John Synex Inc 128 AA Juanita Ave Glendora CA 91740 US

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

ID	SKU / Description	Unit Price (USD)	Qty	Amount (USD)
PS.V860.005	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):	9,463.00
Discount (USD):	0.00
Sales Tax (USD):	780.70
Shipping (USD):	0.00
Total (USD):	10,243.70
Deposit (USD):	0.00
Amount Due (USD):	**10,243.70**

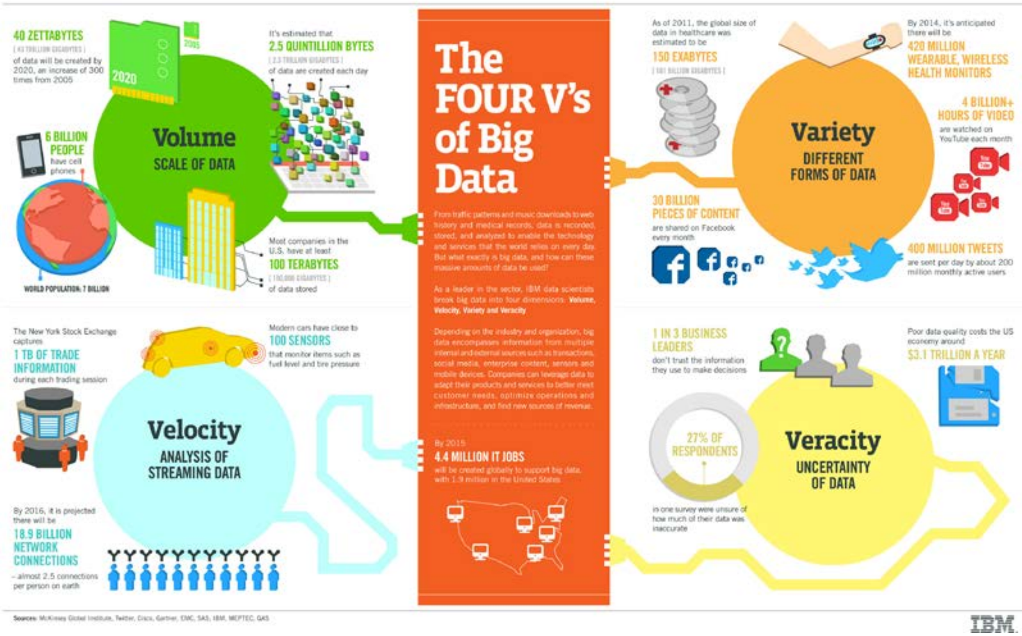


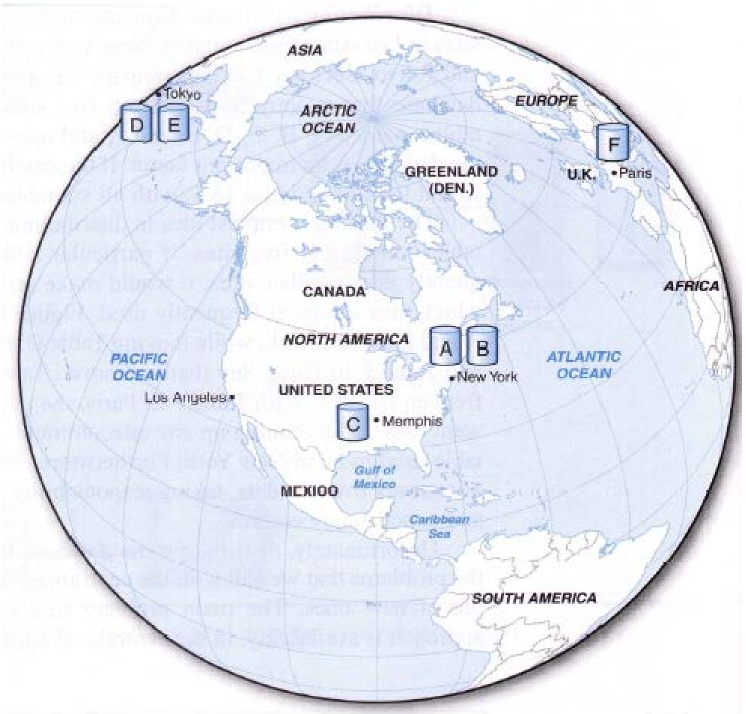
There is a lot of work to dissemble and reassemble the aggregate.

but it enables queries *across* aggregates such as:

SELECT productno, sum(qty)
FROM InvoiceLineItem
GROUP BY productno;

Problems 2, 3: Big Data, Distributed





Big Data = data sets with volume, velocity and variety that traditional relational databases struggle to cope with.

Distributed, especially partitioned, databases are not a good fit for some relational features e.g. foreign keys and transactions.

What is a NoSQL database?

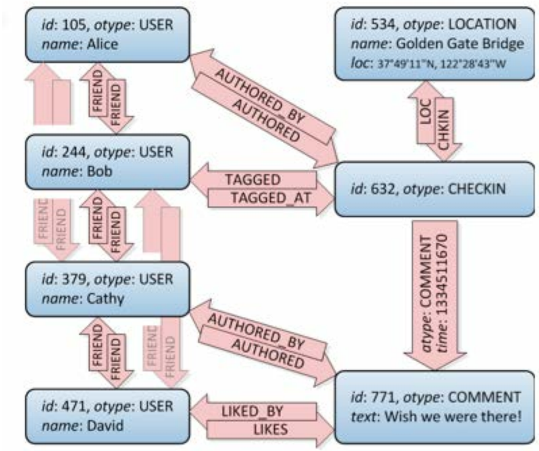
- Features
  - doesn’t use relational model (tables)
  - doesn’t use SQL language
  - designed to run on distributed servers
  - most are open-source
  - built for the modern web
  - schema-less (though "implicit schema" in the application)
  - 'eventually consistent'
- Goals
  - to improve programmer productivity (OR mismatch)
  - to handle larger data volumes and throughput (big data)

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

Three Demos

- We'll see 3 NoSQL databases in action:
  - MongoDB (https://www.mongodb.com/)
  - Firebase (https://firebase.google.com/)
  - Neo4J (https://neo4j.com/)
- Before we start, we need to know a bit about:
  - JSON
  - Graphs


```
{
    id: 111111,
    name: "Alan",
    born: 1990,
    address: "1 Smith st",
    subjects: [
        { subject: "Database", result: "H1" },
        { subject: "Programming", result: "H2A" }
    ]
}
```



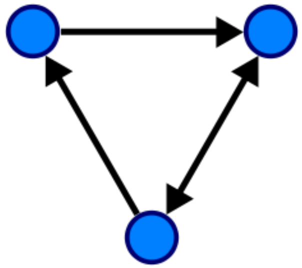
JSON documents

- JavaScript Object Notation
- represents a (JavaScript) object and its properties
- an object consists of a set of attribute-value pairs, including arrays of objects
- has a ‘tree’ structure
- originally used for transmitting data between computers
- now the storage format for Document databases

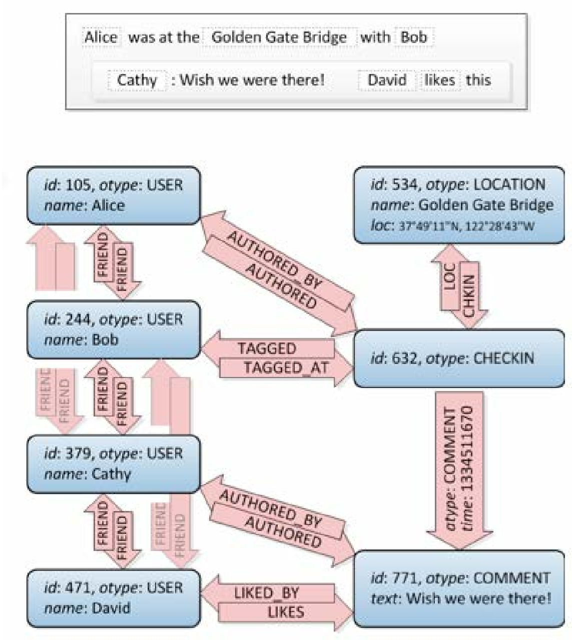
```
[       // array of students
{
    id: 111111,
    name: "Alan",
    born: 1990,
    address: "1 Smith st",
    subjects: [
        { subject: "Database", result: "H1" },
        { subject: "Programming", result: "H2A" }
    ]
}
,
{
    id: 222222,
    name: "Betty",
    born: 1992,
    address: "2 Two st",
    awards: "Best Student",
    subjects: [
        { subject: "Maths", result: "H1" },
        { subject: "Science", result: "H1" },
        { subject: "History", result: "H1"}
    ]
}
,
{
    id: 333333,
    name: "Chris",
    born: 1990,
    address: "3 Three st",
    subjects: [
        { subject: "Database", result: "H1" }
    ]
}
]
```

Graphs

- A data structure consisting of nodes/vertices and arcs/edges
- Nodes represent entities
- Arcs represent relationships
- May be directed or undirected
- In a graph database:
  – nodes and arcs can have properties and types
  – the emphasis is on relationships



directed graph (source: Wikipedia)



social graph (source: Facebook)

demo: the *MongoDB* document store

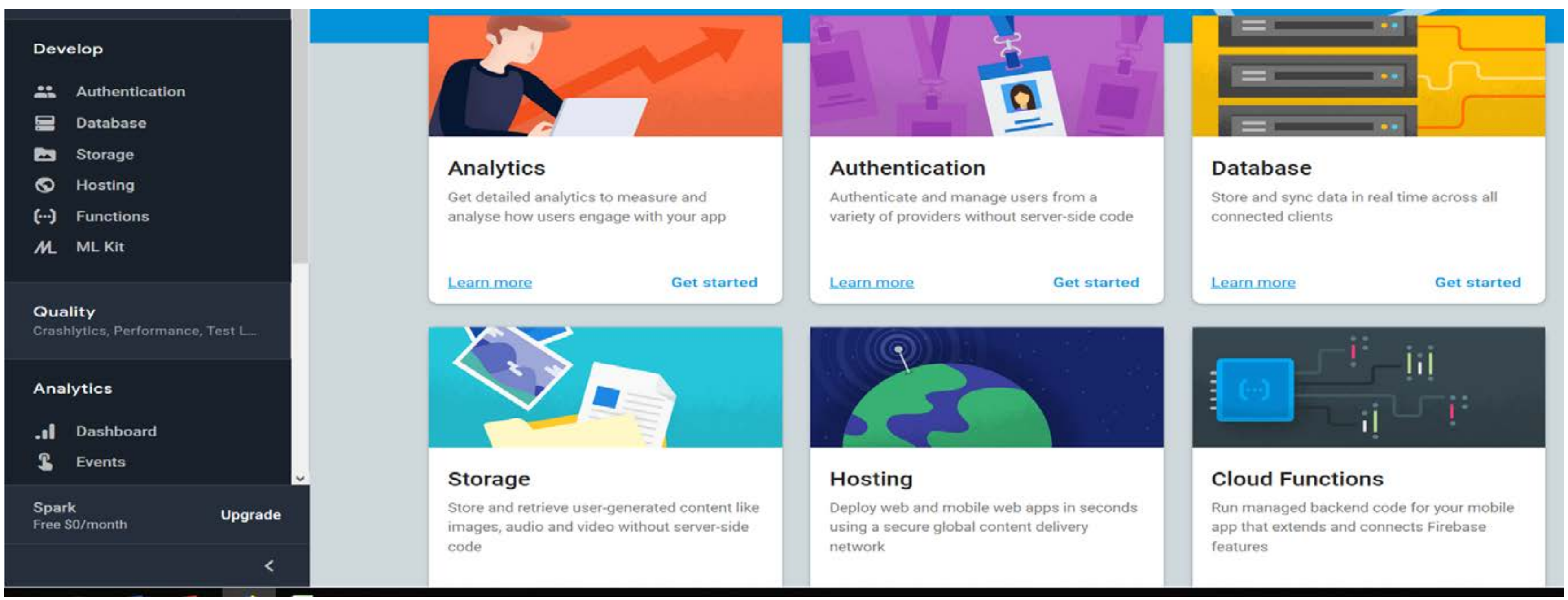
Log into the mongod server using the "mongo“ shell.

```
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: 1982} )    // add a doc to collection
db.students.insert( {name: "Jill", born: 1980} )    // 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"}, {$set: {born: 1980}} ) // change Jack's year
db.students.update( {id:222222}, {$addToSet:
        {subjects: {subject: "English", result: "H1"}}} )

db.students.find( {id:222222}, {_id:false, subjects:true} ).forEach(printjson)
db.students.remove( {born: 1980} )          // delete docs where year = 1980
```

demo: *Google Firebase*

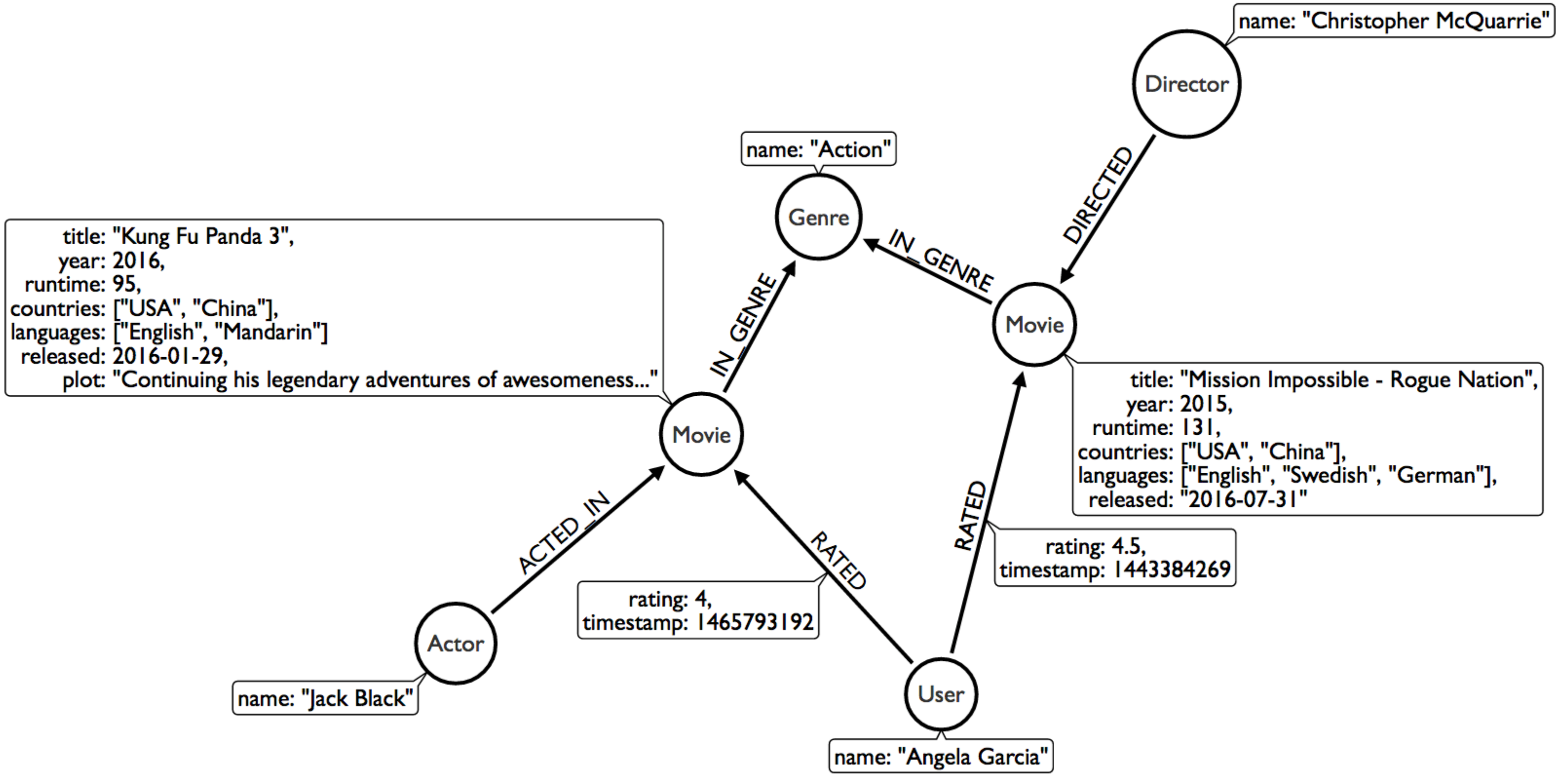
“The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. When you build cross-platform apps with our iOS, Android, and JavaScript SDKs, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.”

messageJSON
- 04:48:19
  - msg: "\"amm\""
  - user: "\"Greg\""
- 04:49:32
  - msg: "\"b\""
  - user: "\"Mary\""
- 18:15:55
  - msg: "\"hello there\""
  - user: "\"Alice\""



demo: the *Neo4J* graph database

Nodes
Movie, Actor, Director, User, Genre are the labels used in this example.
Relationships
ACTED_IN, IN_GENRE, DIRECTED, RATED are the relationships used in this example.
Properties
title, name, year, rating are some of the properties used in this example.



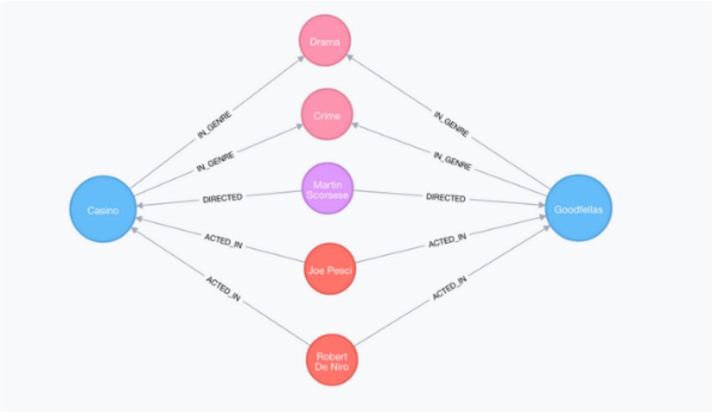
Neo4J queries

queries are written in the *Cypher* language

find	MATCH (m:Movie)<-[:RATED]-(u:User)	Search for an existing graph pattern
filter	WHERE m.title CONTAINS "Matrix"	Filter matching paths to only those matching a predicate
aggregate	WITH m.title AS movie, COUNT(*) AS reviews	Count number of paths matched for each movie
return	RETURN movie, reviews	Specify columns to be returned by the statement
order	ORDER BY reviews DESC	Order by number of reviews, in descending order
limit	LIMIT 5;	Only return first five records

Content-Based Filtering

Recommend items that are similar to those that a user is viewing, rated highly or purchased previously.

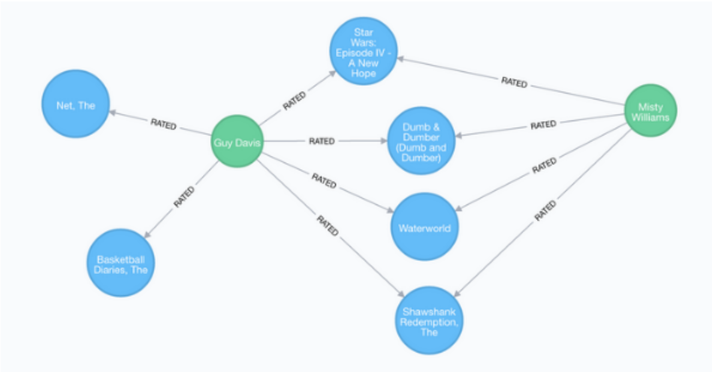


"Products similar to the product you're looking at now"

```
MATCH p=(m:Movie {title: "Net, The"})-[:ACTED_IN|:IN_GENRE|:DIRECTED*2]-()
RETURN p LIMIT 25
```

Collaborative Filtering

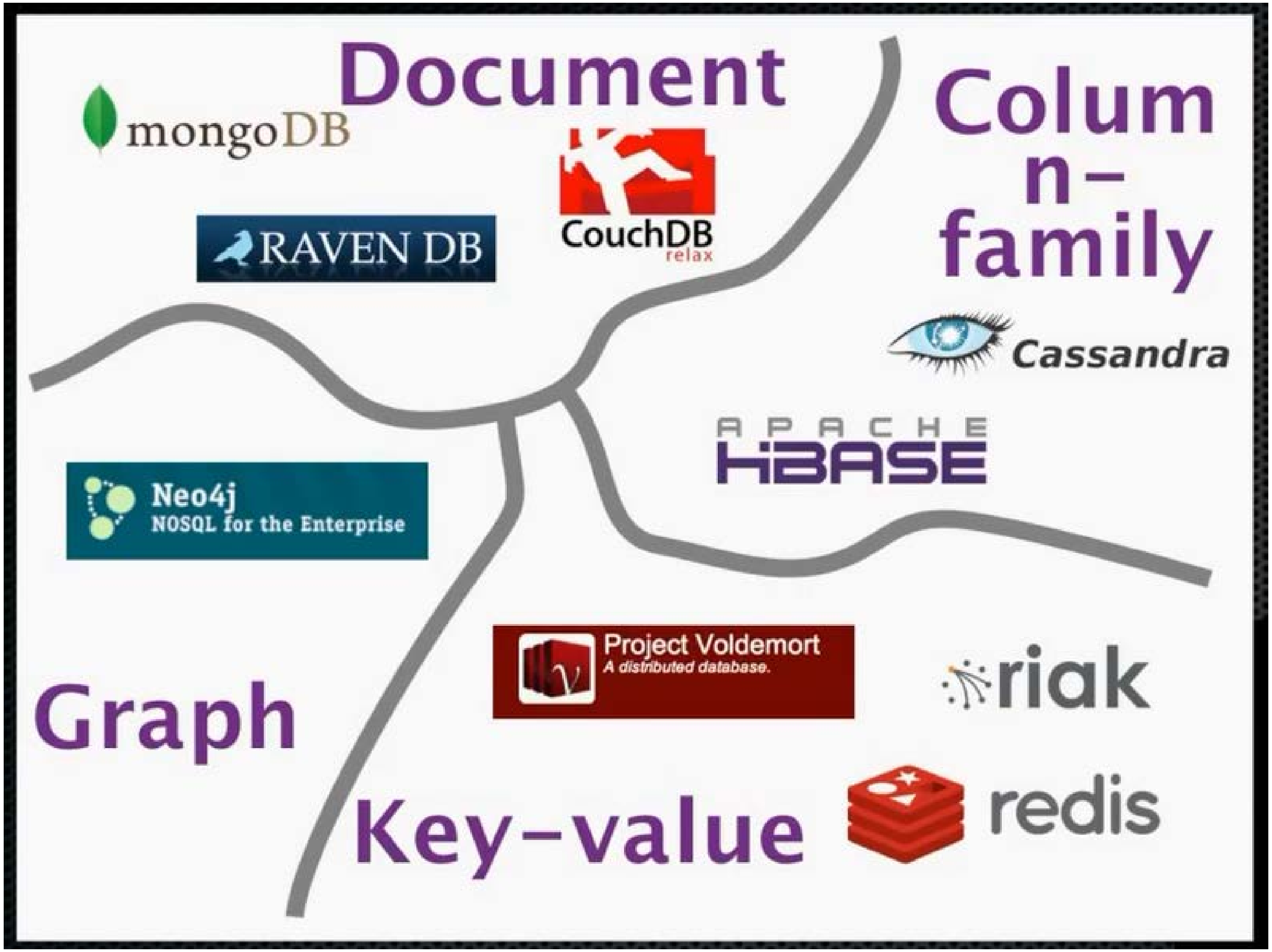
Use the preferences, ratings and actions of other users in the network to find items to recommend.



"Users who bought this thing, also bought that other thing."

```
MATCH (m:Movie {title: "Crimson Tide"})<-[:RATED]-(u:User)-[:RATED]->(rec:Movie)
RETURN rec.title AS recommendation, COUNT(*) AS usersWhoAlsoWatched
ORDER BY usersWhoAlsoWatched DESC LIMIT 25
```


Types of NoSQL database



(diagram from Martin Fowler)

Types of NoSQL: key-value store

Key = primary key

Value = anything
(number, array, image, JSON)
The application is in charge of interpreting what it means.

examples: Riak, Redis, Memcached, Berkeley DB, Project Voldemort, Couchbase

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Types of NoSQL: document database

Like a key-value db,
except that the "value" (document) is "examinable" by the db, so its contents can be queried and updated

document = object represented as JSON file

examples: MongoDB, CouchDB, Terrastore, OrientDB, RavenDB,

<Key=CustomerID>

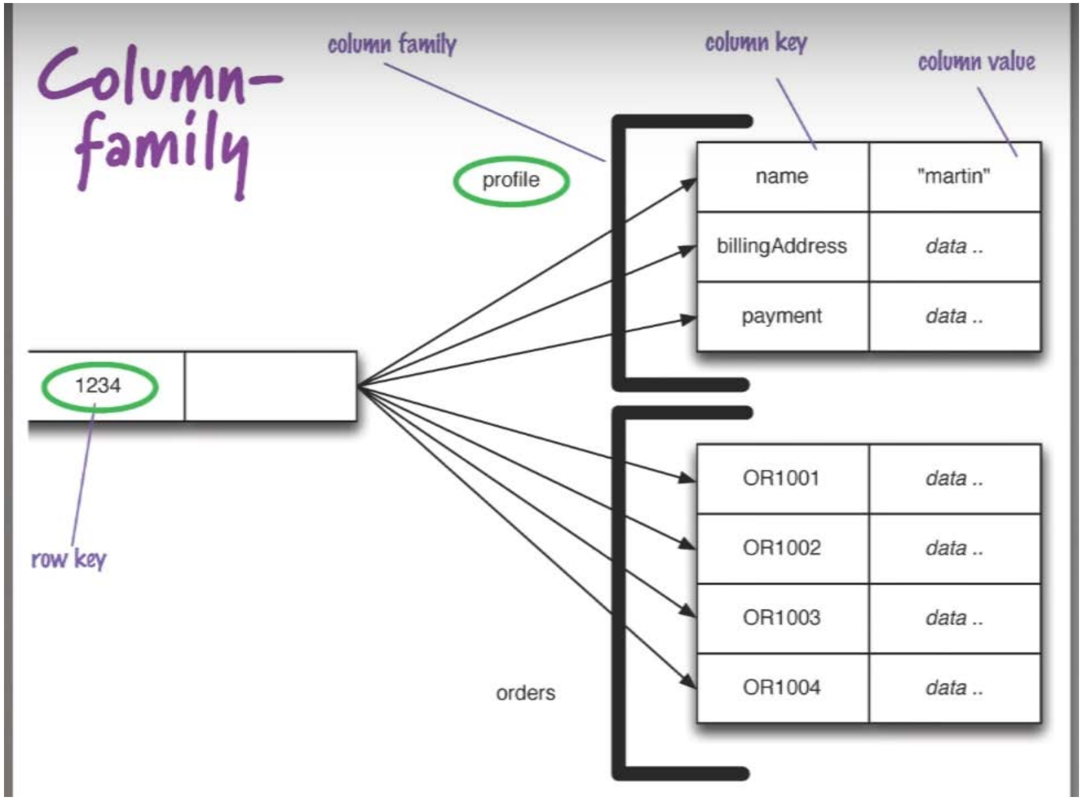
```
{
   "customerid": "fc986e48ca6"
   "customer":
   {
   "firstname": "Pramod",
   "lastname": "Sadalage",
   "company": "ThoughtWorks",
   "likes": [ "Biking","Photography" ]
   }
   "billingaddress":
   { "state": "AK",
     "city": "DILLINGHAM",
     "type": "R"
    }
}
```

Types of NoSQL: column family

"Column family" is like a relational table.
It contains many "rows".
But each row can store a *different set of columns*.

Columns rather than rows are stored together on disk. Makes analysis by column faster – not for OLTP.

examples: Cassandra, BigTable, HBase, DynamoDB



Aggregate-oriented databases

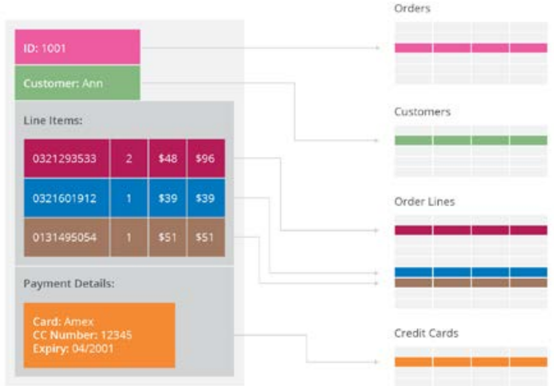
Key-value, *document store* and *column-family* are "aggregate-oriented" databases (in Fowler's terminology)

Pros

- entire aggregate of data is stored together
- less need for transactions
- efficient storage on clusters / distributed databases

Cons

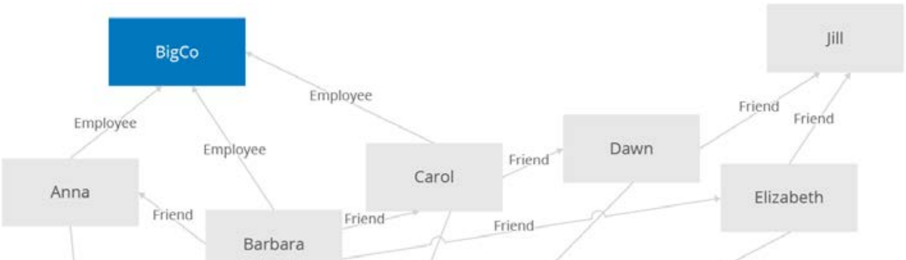
- hard to analyse across subfields of aggregates
  e.g. sum over products instead of orders



Product	revenue	prior revenue
321293533	3083	7043
321601912	5032	4782
131495054	2198	3187
...	...	...

Types of NoSQL: graph database

A ‘graph' is a node-and-arc network
Social graphs (e.g. friendship graphs) are common examples
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
- OrientDB
- FlockDB
- TAO

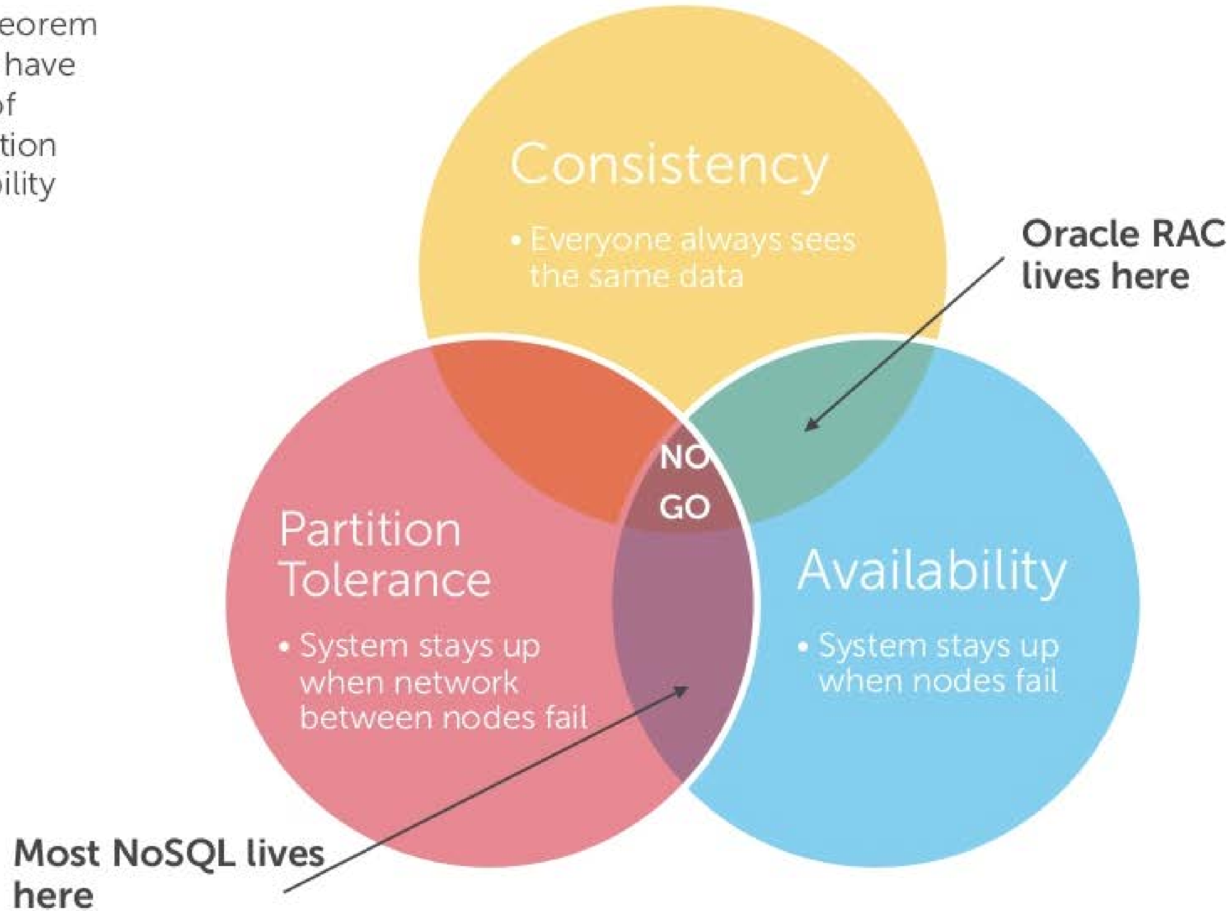
Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

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

Distributed data: the CAP theorem

CAP Theorem says something has to give

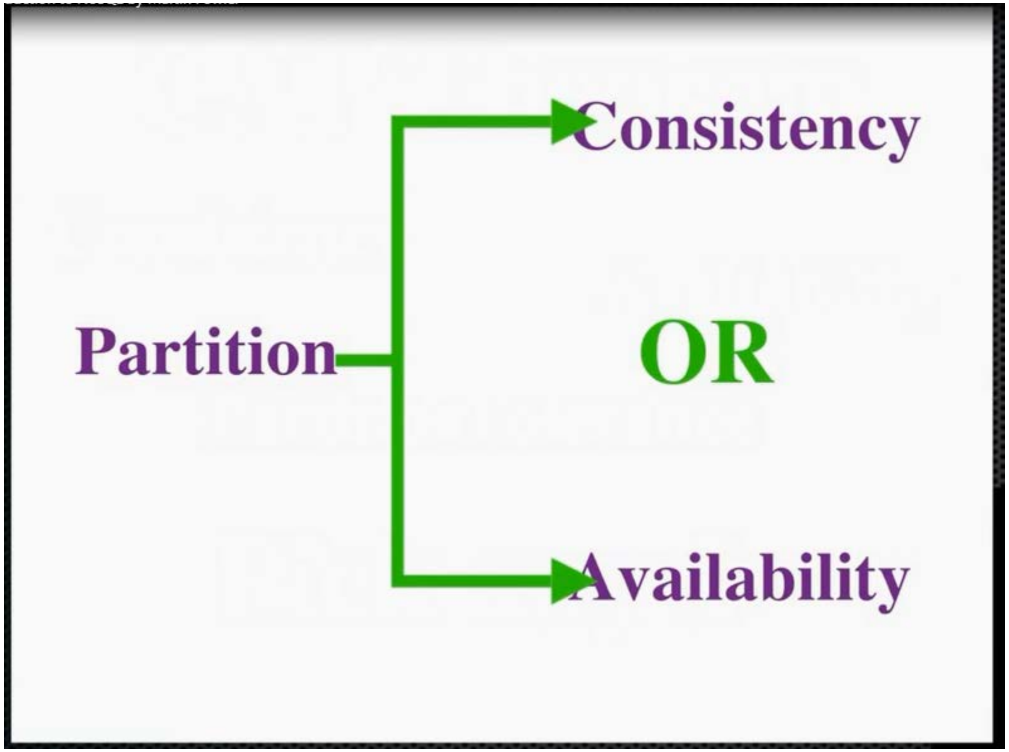
- CAP (Brewer's) Theorem says you can only have two out of three of Consistency, Partition Tolerance, Availability



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.



ACID vs BASE

ACID (Atomic, Consistent, Isolated, Durable)
vs
BASE (Basically Available, Soft state, Eventual consistency)

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.

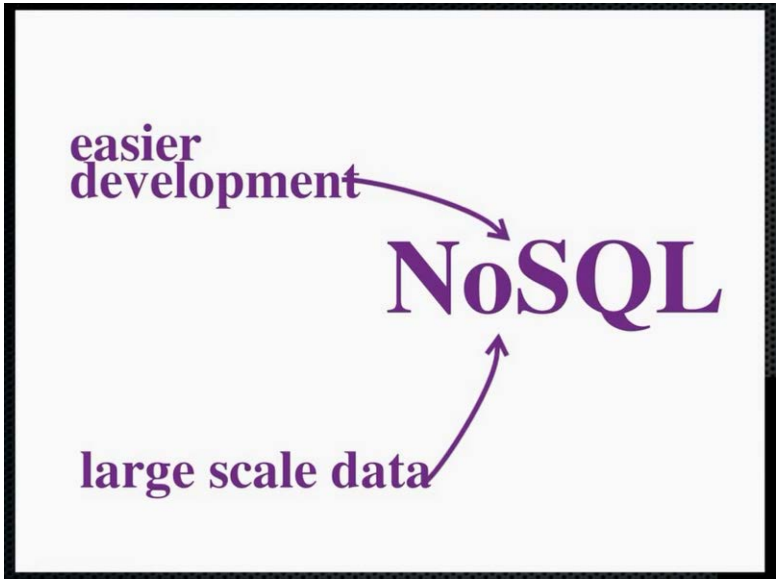
Some notable NoSQL users

- Google – BigTable
  - search, gmail, maps, youtube
- Facebook – Cassandra, Tao, Giraph
  - messaging, social graph
- Amazon – SimpleDB, DynamoDB
  - large scale e-commerce and analytics, cloud db
- Instagram - Cassandra
  - social media newsfeed
- LinkedIn – CouchDB, MongoDB
  - monitoring and analysis of operational data
- The Guardian - MongoDB
  - newspaper articles, user identity
- FourSquare - MongoDB
  - venues and user checkins

Do *you* need to know NoSQL?

Q. Do only big web companies like Google, Amazon and Facebook need NoSQL?

A. In fact, any organization is likely to have to start dealing with large amounts of data (due to web, mobile, sensors etc), while some are adopting NoSQL to avoid object-relational mismatch (making programming easier).



but Relational DBMS will probably continue to be used in many applications