UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Campus d'Excel·lència Internacional

DTIM

NOSQL Databases

OSCAR ROMERO

DTIM RESEARCH GROUP (HTTP://WWW.ESSI.UPC.EDU/DTIM/)

UNIVERSITAT POLITÈCNICA DE CATALUNYA - BARCELONATECH

Table of contents

Table of contents

Table of contents

NOSQL: A Paradigm Shift

FROM RELATIONAL TO NOSQL

A New Business Model

Traditionally databases have been seen as a passive asset

- OLTP systems: Data gathered is structured to facilitate (automate) daily operations
- The relational model as *de facto* standard

Soon, many realized data is a valuable asset for any organization. So, *use* it!

- Decisional systems: Stored data is analysed to better understand our activity (*I want to know*)
- Data warehousing as *de facto* standard

Instagram's Fable

Other Examples

- The overkilling approach
  - Facebook: Facebook + Instagram + Whatsapp + Oculus VR + …
  - Google: Android + Google Search + Calendar + Gmail + Doodle + …
- Even if most of us are not Facebook or Google we can still benefit from data
  - Cross available data (companies fusion, buying data, open data, agreements, etc.)
  - Digitalise the organization processes (e.g., the national health system, tax collection, e-banking, etc.)
  - Monitor the user (phone apps, internet navigation, service usage, wearables, RFID, etc.)
    - Sometimes, in an indirect way (e.g., provide (free) services to learn habits; such as free wi-fi to geolocate the user)
  - Sensors (Smart Cities, Internet of Things, etc.)
  - …
- Bottom line: most of the times, the most interesting data is not available (innovation comes to play!)

Data as the New Cornerstone

We have witnessed the bloom of a new business model based on data analytics: *Data is not a passive but an active asset*

- «*Data is the new oil*!» - Clive Humby, 2006
- «*No! Data is the new soil*» - David McCandless, 2010

The effective use of data to make decisions gave rise to the **data-driven society** concept

The confluence of three major socio-economic and technological trends makes data-driven innovation a new phenomenon today. These three trends include (OECD):

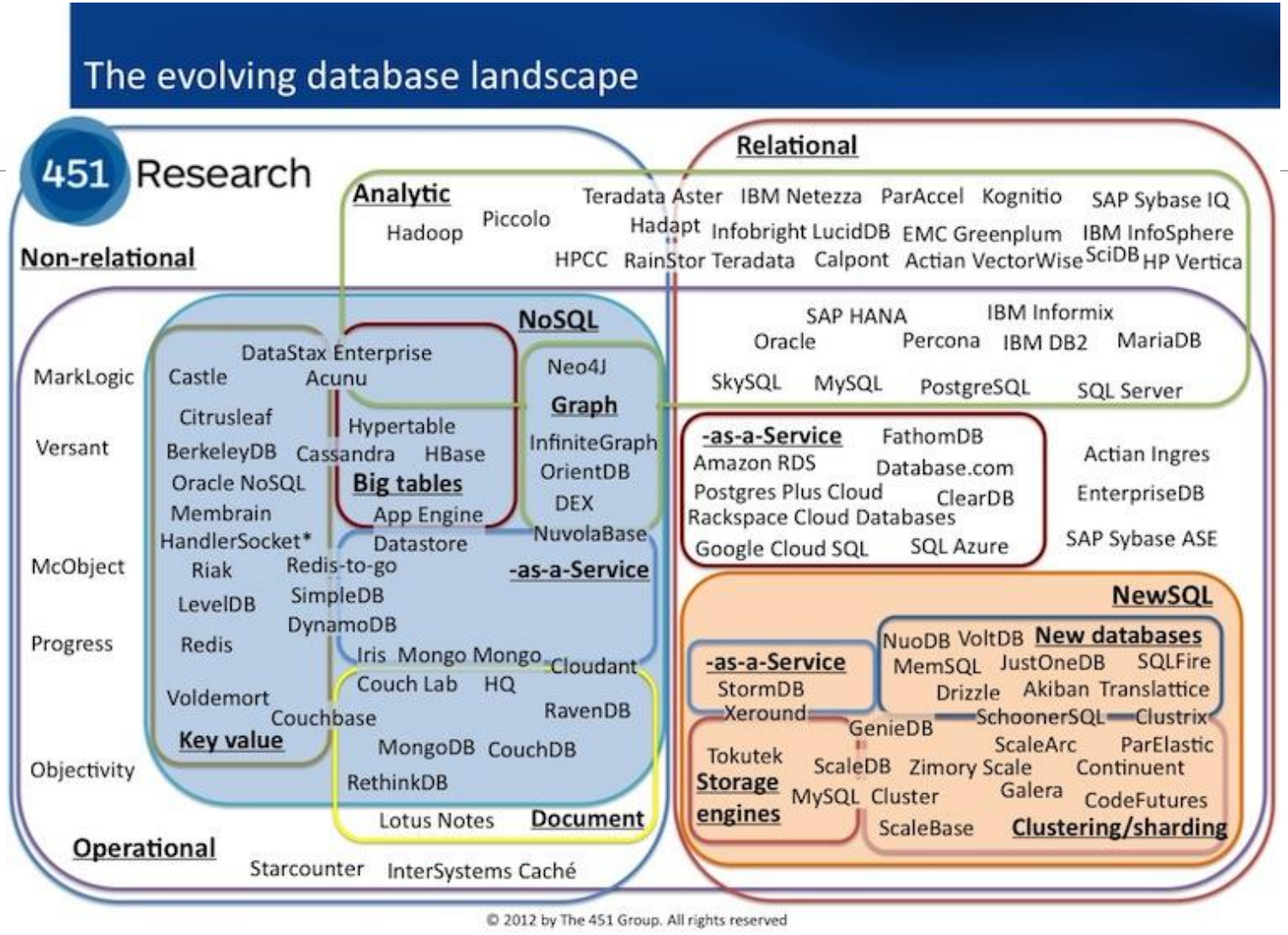
- The exponential growth in data generated and collected,
- the widespread use of data analytics including start-ups and small and medium enterprises (SMEs), and
- the emergence of a paradigm shift in knowledge

Organizations must adapt their **infrastructures** to benefit from the data deluge

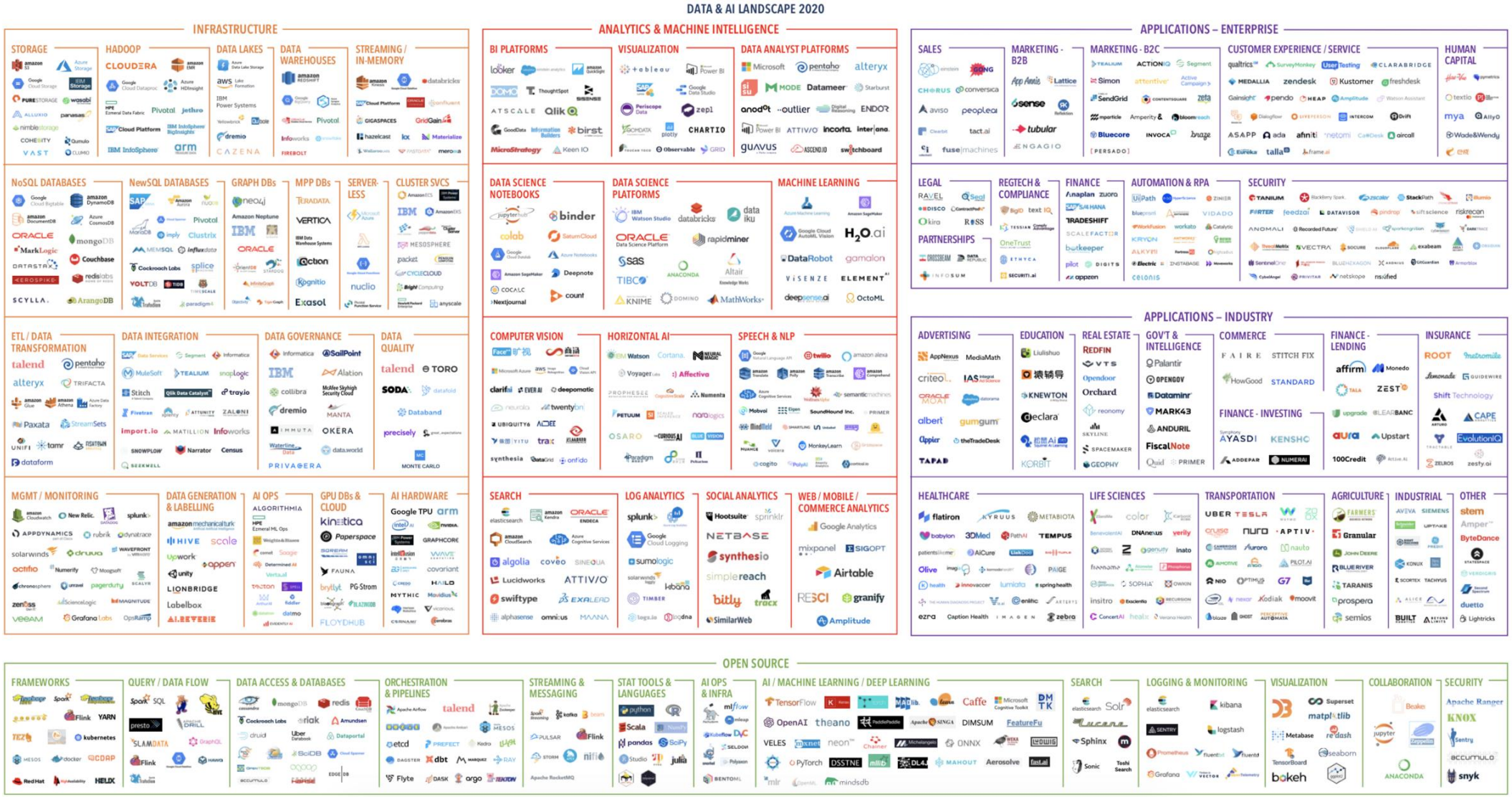
- Digital data doubling every 18 months (IDC)

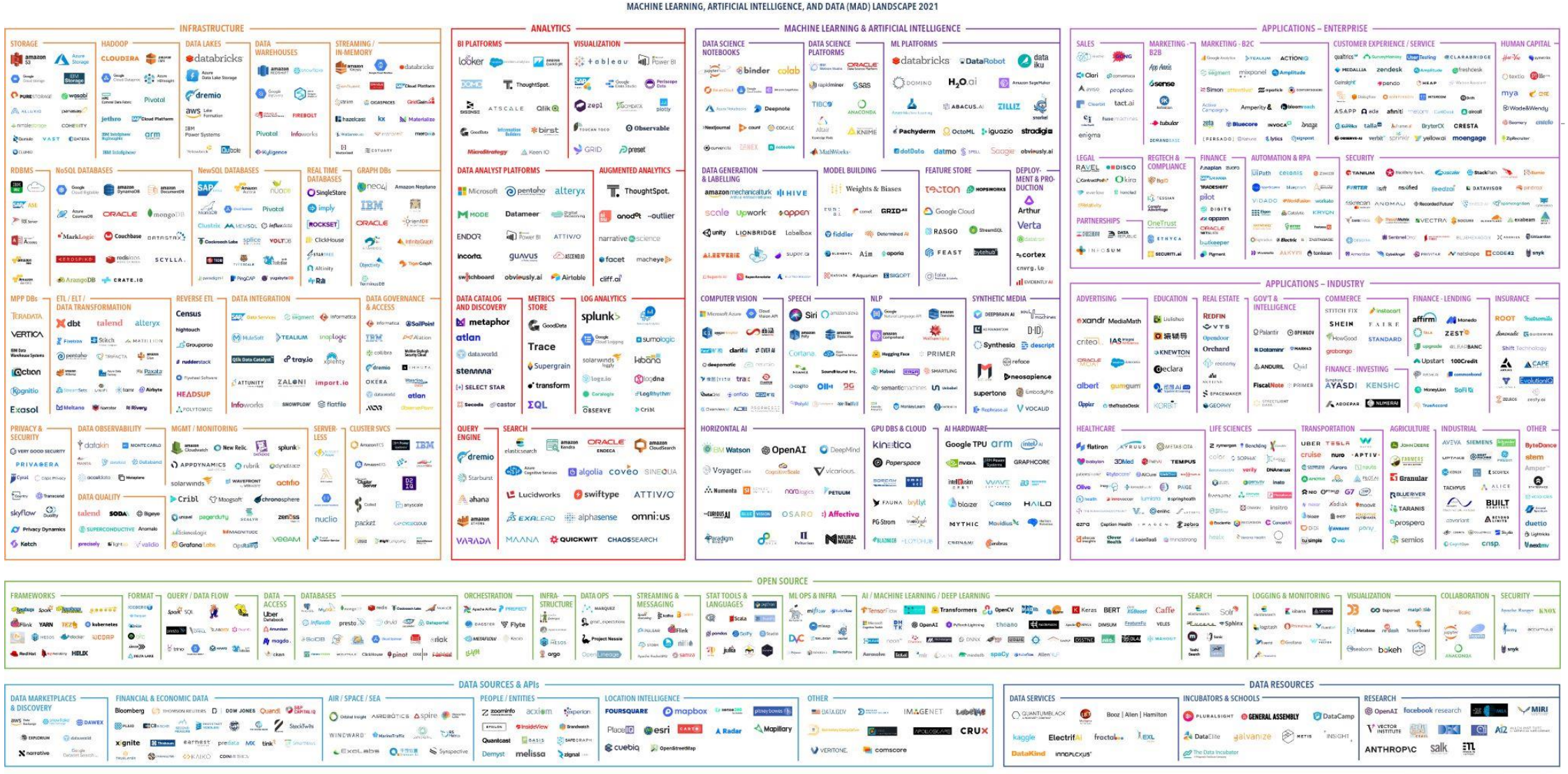
Innovation is mandatory!

The Data Landscape (2012)



By 451 Research


DATA & AI LANDSCAPE 2020
INFRASTRUCTURE
STORAGE
HADOOP
DATA LAKES
DATA WAREHOUSES
STREAMING / IN-MEMORY
NoSQL DATABASES
NewSQL DATABASES
GRAPH DBs
MPP DBs
SERVER-LESS
CLUSTER SVCS
ETL / DATA TRANSFORMATION
DATA INTEGRATION
DATA GOVERNANCE
DATA QUALITY
MGMT / MONITORING
DATA GENERATION & LABELLING
AI OPS
GPU DBs & CLOUD
AI HARDWARE
ANALYTICS & MACHINE INTELLIGENCE
BI PLATFORMS
VISUALIZATION
DATA ANALYST PLATFORMS
DATA SCIENCE NOTEBOOKS
DATA SCIENCE PLATFORMS
MACHINE LEARNING
COMPUTER VISION
HORIZONTAL AI
SPEECH & NLP
SEARCH
LOG ANALYTICS
SOCIAL ANALYTICS
WEB / MOBILE / COMMERCE ANALYTICS
APPLICATIONS – ENTERPRISE
SALES
MARKETING - B2B
MARKETING - B2C
CUSTOMER EXPERIENCE / SERVICE
HUMAN CAPITAL
LEGAL
REGTECH & COMPLIANCE
FINANCE
AUTOMATION & RPA
SECURITY
PARTNERSHIPS
APPLICATIONS – INDUSTRY
ADVERTISING
EDUCATION
REAL ESTATE
GOV'T & INTELLIGENCE
COMMERCE
FINANCE - LENDING
INSURANCE
FINANCE - INVESTING
HEALTHCARE
LIFE SCIENCES
TRANSPORTATION
AGRICULTURE
INDUSTRIAL
OTHER
OPEN SOURCE
FRAMEWORKS
QUERY / DATA FLOW
DATA ACCESS & DATABASES
ORCHESTRATION & PIPELINES
STREAMING & MESSAGING
STAT TOOLS & LANGUAGES
AI OPS & INFRA
AI / MACHINE LEARNING / DEEP LEARNING
SEARCH
LOGGING & MONITORING
VISUALIZATION
COLLABORATION
SECURITY


MACHINE LEARNING, ARTIFICIAL INTELLIGENCE, AND DATA (MAD) LANDSCAPE 2021
INFRASTRUCTURE
ANALYTICS
MACHINE LEARNING & ARTIFICIAL INTELLIGENCE
APPLICATIONS – ENTERPRISE
APPLICATIONS – INDUSTRY
OPEN SOURCE
DATA SOURCES & APIs
DATA RESOURCES
Version 1.0 - September 2021
© Matt Turck (@mattturck), John Wu (@john_d_wu) & FirstMark (@firstmarkcap)
mattturck.com/data2021
FIRSTMARK
EARLY STAGE VENTURE CAPITAL

Watch Out! The Problem is **NOT** SQL

SQL is a query language. SQL is **cool**! Not a problem

The problem is that relational systems are too generic…

- OLTP: stored procedures and simple queries
- OLAP: ad-hoc complex queries
- Key-value: large, high-throughput data
- Documents: large objects
- Streams: time windows with volatile data
- Graphs: flexible data modeling considering topological aspects
- Scientific: uncertainty and heterogeneity

… But the overhead of RDBMS has nothing to do with SQL

- Low-level, record-at-a-time interface is not the solution

SQL Databases vS. NoSQL Databases
Michael Stonebraker
Communications of the ACM, 53(4), 2010

NOSQL Three Pillars

Flexible data management to cope with Big Data (Volume, Velocity and Variety)

1. Distributed Data Management
   - Divide-and-conquer principle
     - Use (and abuse) of parallelism
2. Flexible data models
   - Reduce the *impedance mismatch*
     - Reduce the code overhead to make transformations between different data models
3. New architectural solutions
   - Relational database architectures date back to the 70s
     - Some modules may not be needed (e.g., concurrency control in read-only environments)
     - Some modules need to be rethought (e.g., from disk-based to memory-based architectures)

Data Management

DATA LIFECYCLE

Data Management

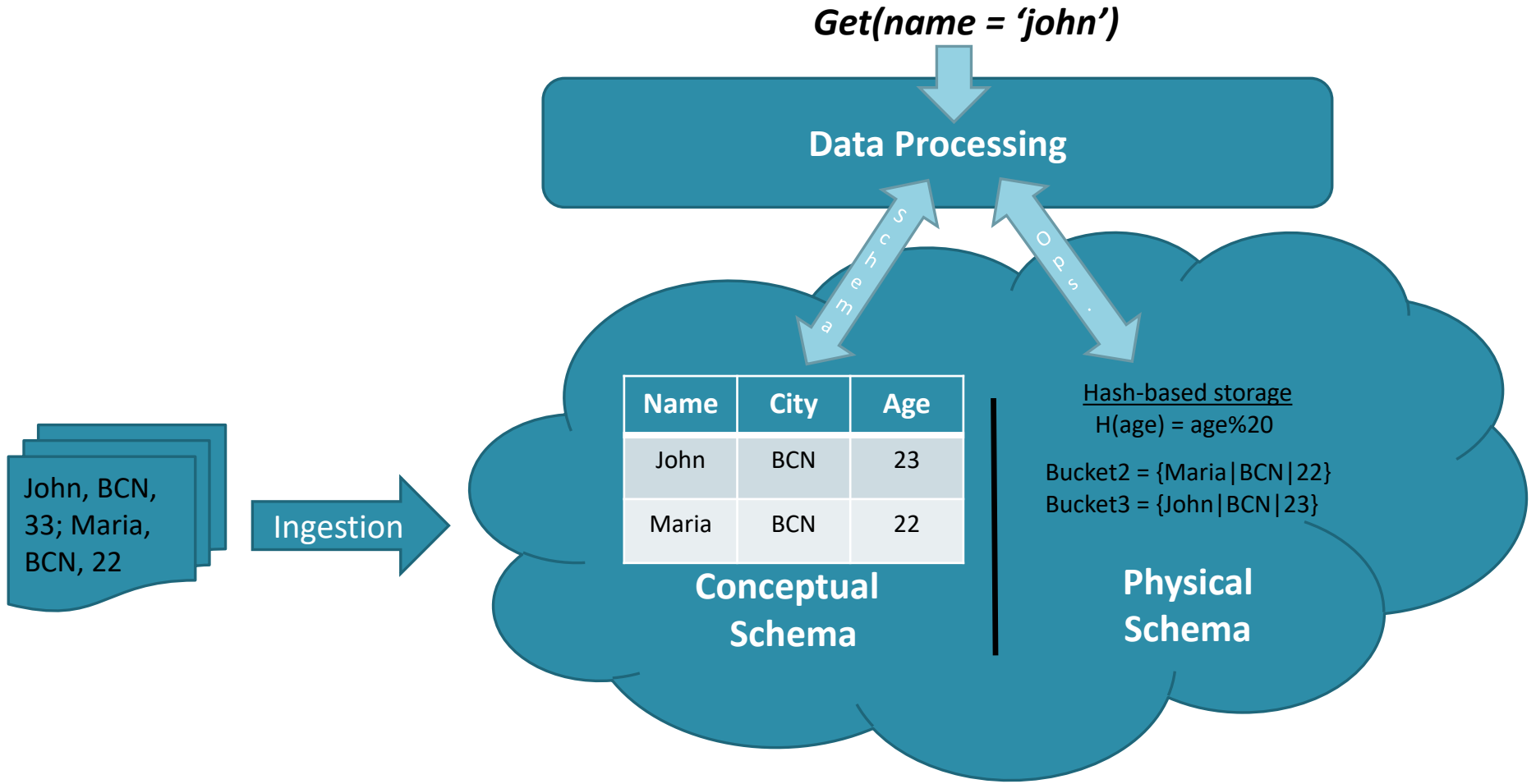
Data management refers to the tasks a database management system (DBMS) must provide to cover the data lifecycle within an IT system. In this context, we focus on:

- **Ingestion**: the means provided to insert /upload / put data into the DBMS (in SQL, the INSERT command)
- **Modeling**: the *conceptual* data structures used to arrange data within the DBMS (e.g., tables in the relational model, graphs in graph modelling, etc.)
- **Storage**: the *physical* data structures used to persist data in the DBMS (e.g., hash, b-tree, heap files, etc.)
- **Processing**: the means provided (many times, in algebraic form) to manipulate data once stored in the the DBMS *physical* data structures (in SQL, the relational algebra)
- **Querying / fetching**: the means provided to allow the DBMS user to specify the data processing she would like to perform (in SQL, SELECT queries). It is typically triggered in terms of the *conceptual* data structures

In **Big Data settings**, it refers to the **same** concepts but assuming a NOSQL system is behind

- Typically, a distributed system
- Possibly with an alternative data model to the relational one
- Implementing ad-hoc architectural solutions

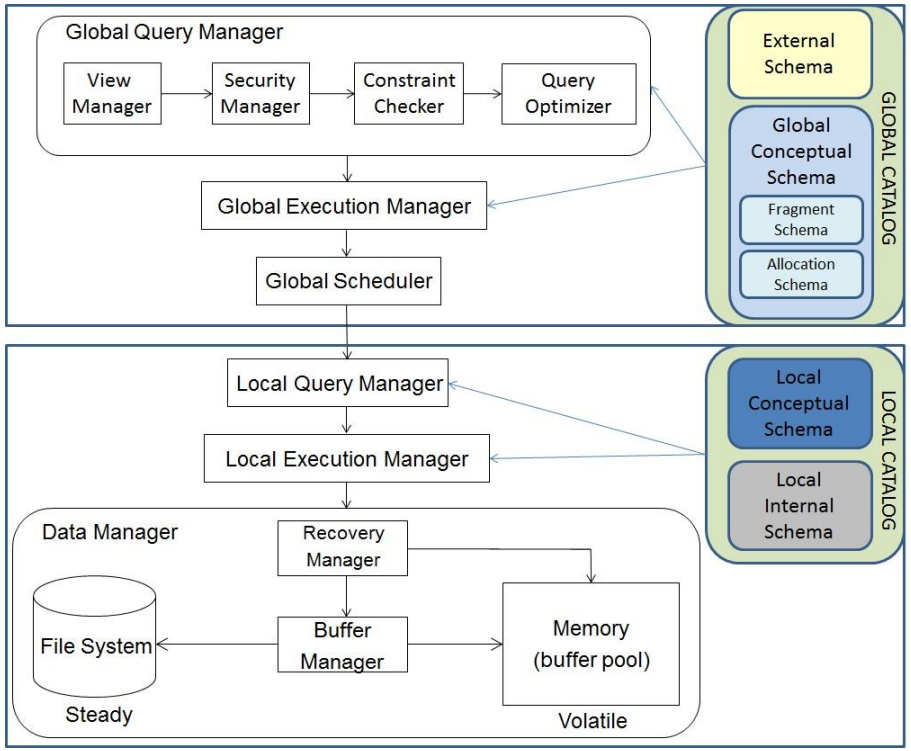
Data Management



On the Need of New Architectures

NOSQL introduces a critical reasoning on the reference database architecture

- ALL relational databases follow System-R architecture (late 70s!)



Bottlenecks

Distributed RDBMS (DRDBMS) are not flexible enough

- Logging
  - Persistent redo log
  - Undo log
- CLI interfaces (JDBC, ODBC, etc.)
- Concurrency control (locking)
- Latching associated with multi-thread
- Two-phase commit (2PC) protocol in distributed transactions
- Buffers management (cache disk pages)
- Variable length records management
  - Locate records in a page
  - Locate fields in a record

ACID properties

Data structure

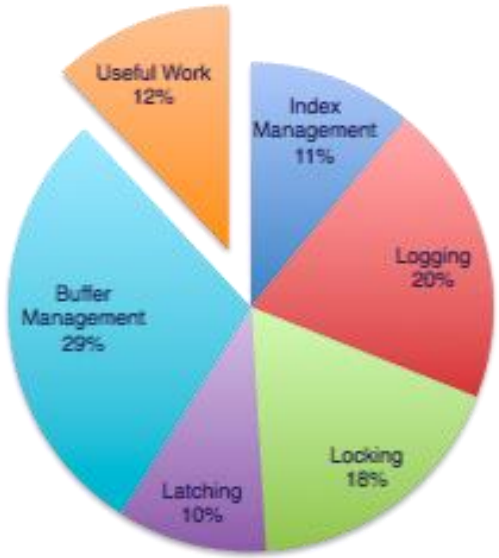
ACID properties

Architectural limitation

ACID properties

Architectural limitation

Data structure



The End of an Architectural Area. It's Time for a Complete Rewrite. Michael Stonebraker et al.
VLDB Proceedings, *2007*

NOSQL Architectures

RELATIONAL DBMS

Generic architecture that can be tuned according to the needs:

Mainly write-only Systems (e.g., OLTP)
- Normalization
- Indexes: B+, Hash
- Joins: BNL, RNL, Hash-Join, Merge Join

Read-only Systems (e.g., DW)
- Denormalized data
- Indexes: Bitmaps
- Joins: Star-join
- Materialized Views

NOSQL

Concepts that gained weight
- Primary indexes
- Sequential reads
- Vertical partitioning
- Compression (e.g., run length encoding)
- Fixed-size values
- Materialized views
- In-memory processing
- Bloom filters
- Indexing
- Denormalization
- …

Such architectures are very **specific** and good (but very good) in solving a specific problem

Data Modeling

RELATIONAL DBMS

Based on the relational model

- Tables, rows and columns
- Each row represents an instance, columns attributes / features
- Constraints are limited
  - PK, FK, Check, …

When creating (i.e., at creation time) the tables you **MUST** specify the table schema (i.e., columns and constraints)

NOSQL

No single reference model

- Key-value
- Document
- Stream
- Graph
- …

The closer the model looks to the way data is later stored internally the better (**impedance mismatch**)

Further, ideally, the schema should be defined at insertion time and not at definition time (**schemaless databases**)

Key-Value

Characteristics

- Entries in form of key-values (can be seen as enormous hash tables)
- One key maps only to one value
- Query on key only
- Schemaless

Suitable for

- Very large repositories of data (scalability is a must)
- Unstructured data whose schema is difficult to predict
- Main way to access data is by key

Systems

- Hadoop ecosystem

Document

Characteristics

- Entries in form of key-values where the value is a XML or JSON document
- One key maps only to one document
- It is possible to **index** any of the document entries (in JSON, the JSON keys, in XML the XML tags)
- Query on key or indexed document entries
- Schemaless

key

SocialNetwork

document


```
{
   "title": "The Social network",
   "year": "2010",
   "genre": "drama",
   "summary": "On a fall night in 2003, Harvard undergrad and computer
   programming genius Mark Zuckerberg sits down at his computer
   and heatedly begins working on a new idea. In a fury of blogging
   and programming, what begins in his dorm room soon becomes a global
   social network and a revolution in communication. A mere six years
   and 500 million friends later, Mark Zuckerberg is the youngest
   billionaire in history... but for this entrepreneur, success leads
   to both personal and legal complications.",
   "country": "USA",
 "director": {
     "last_name": "Fincher",
     "first_name": "David",
     "birth_date": "1962"
 },
 "actors": [
   {
       "first_name": "Jesse",
       "last_name": "Eisenberg",
       "birth_date": "1983",
       "role": "Mark Zuckerberg"
   },
   {
       "first_name": "Rooney",
       "last_name": "Mara",
       "birth_date": "1985",
       "role": "Erica Albright"
   },
   {
       "first_name": "Andrew",
       "last_name": "Garfield",
       "birth_date": "1983",
       "role": "  Eduardo Saverin "
   },
   {
       "first_name": "Justin",
       "last_name": "Timberlake",
       "birth_date": "1981",
       "role": "Sean Parker"
   }
 ]
}
```


Suitable for

- Data produced in XML / JSON format (web data)
- Unstructured data whose schema is difficult to predict
- Main way to access data is by key and all the document is processed together

Systems

- MongoDB, CouchDB, ElasticSearch, ArangoDB

Graph

Characteristics

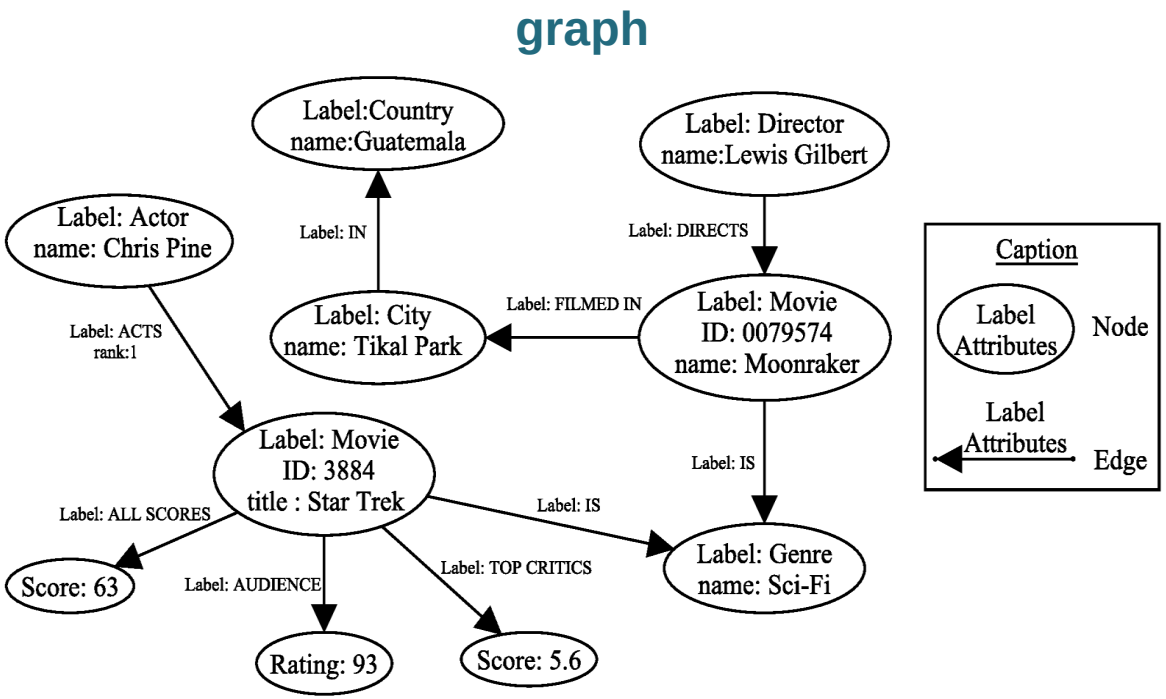
- Data stored as nodes and edges
- Relationships as first-class citizens
- Schemaless

Suitable for

- Unstructured data whose schema is difficult to predict
- Topology queries (based on the shape of the graph)

Systems

- Neo4J, Titan, Sparksee, GraphDB, Stardog

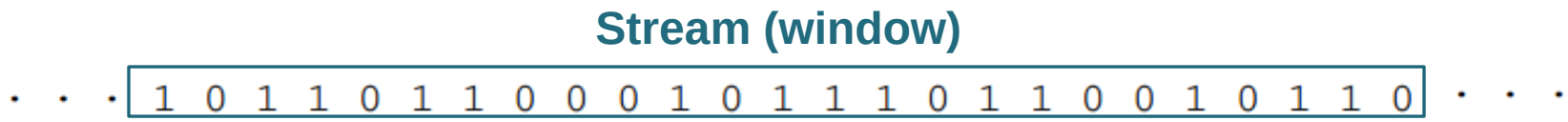


http://grouplens.org/datasets/movielens/

Streams & Complex-Event Processing

Characteristics

- Strong temporal locality
- The whole dataset is not available but a portion of it (**window**)



Suitable for

- Real-time applications
- Data per se is not important but to gain insight (e.g., sensor data)
- Approximate answers are enough

Systems

- Apache Flink, Spark Streaming, ~~Storm~~, Apache Kafka

DOCUMENT-STORES

MONGODB AND ELASTICSEARCH

Structuring the Value

Essentially, **they are key-value stores**

- Same design and architectural features

There is no explicit schema but **the *value* is further structured as a document**

- XML (e.g., MarkLogic)
- JSON (e.g., MongoDB, CouchDB, ElasticSearch)

Tightly related to the Web

- Normally, they provide RESTful HTTP APIs

Benefits

- New data model (collections and documents)
  - **New atom: from rows to documents**
- Secondary Indexing

Types of Document-Stores

XML-like documents
- eXist, MarkLogic
  - Natively supported
- Relational extensions for Oracle, PostgreSQL, etc.
  - Mapped to relational (heavy impedance mismatch!)

XML is a semistructured data model proposed as the standard for data exchange on the Web
- Can be elegantly represented as a tree
  - Document: the root node of the XML document, denoted by "/"
  - Element: element nodes that correspond to the tagged nodes in the document
  - Attribute: attribute nodes attached to Element nodes
  - Text: text nodes, i.e., untagged leaves of the XML tree

Support Xpath, Xquery and XSLT
- Xpath is a language for addressing portions of an XML document
  - Subset of XQuery
- XQuery is a query language for extracting information from collections of XML documents
- XSLT is a language for specifying transformations (from XML to XML)

Types of Document-Stores

JSON-like documents

- MongoDB
- CouchDB
- ElasticSearch
- ArangoDB (promising, as it combines documents and graphs)

JSON is a lightweight data interchange format

- Brackets ([]) represent ordered lists
- Curly braces ({}) represent key-value dictionaries
  - Keys must be strings, delimited by quotes (")
  - Values can be strings, numbers, booleans, lists, or key-value dictionaries

Natively compatible with JavaScript

- Web browsers are natural clients for document-stores

  http://www.json.org/index.html

Example

Definition:

A document is an object represented with an unbounded nesting of array and object constructs

```
{
    "title": "The Social network",
    "year": "2010",
    "genre": "drama",
    "summary": "On a fall night in 2003, Harvard undergrad and computer
    programming genius Mark Zuckerberg sits down at his computer
    and heatedly begins working on a new idea. In a fury of blogging
    and programming, what begins in his dorm room soon becomes a global
    social network and a revolution in communication. A mere six years
    and 500 million friends later, Mark Zuckerberg is the youngest
    billionaire in history... but for this entrepreneur, success leads
    to both personal and legal complications.",
    "country": "USA",
  "director": {
      "last_name": "Fincher",
      "first_name": "David",
      "birth_date": "1962"
  },
  "actors": [
    {
        "first_name": "Jesse",
        "last_name": "Eisenberg",
        "birth_date": "1983",
        "role": "Mark Zuckerberg"
    },
    {
        "first_name": "Rooney",
        "last_name": "Mara",
        "birth_date": "1985",
        "role": "Erica Albright"
    },
    {
        "first_name": "Andrew",
        "last_name": "Garfield",
        "birth_date": "1983",
        "role": "  Eduardo Saverin "
    },
    {
        "first_name": "Justin",
        "last_name": "Timberlake",
        "birth_date": "1981",
        "role": "Sean Parker"
    }
  ]
}
```

Documents as Data Model

Schemaless databases: No explicit schema

- Implicit schema (e.g., tags from JSON or XML)
  - One of the tags play the role of the “key”
- They do not support schema info such as XML
  - Two documents may have different schema

https://docs.mongodb.com/manual/core/document/

Design Strategies

Do not think relational-wise

- Break 1NF to avoid joins
- One basic rule: minimize the I/O (i.e., number of documents read) by maximizing the effective read ratio
- Use primary and secondary indexes to identify the right data granularity

Consequences

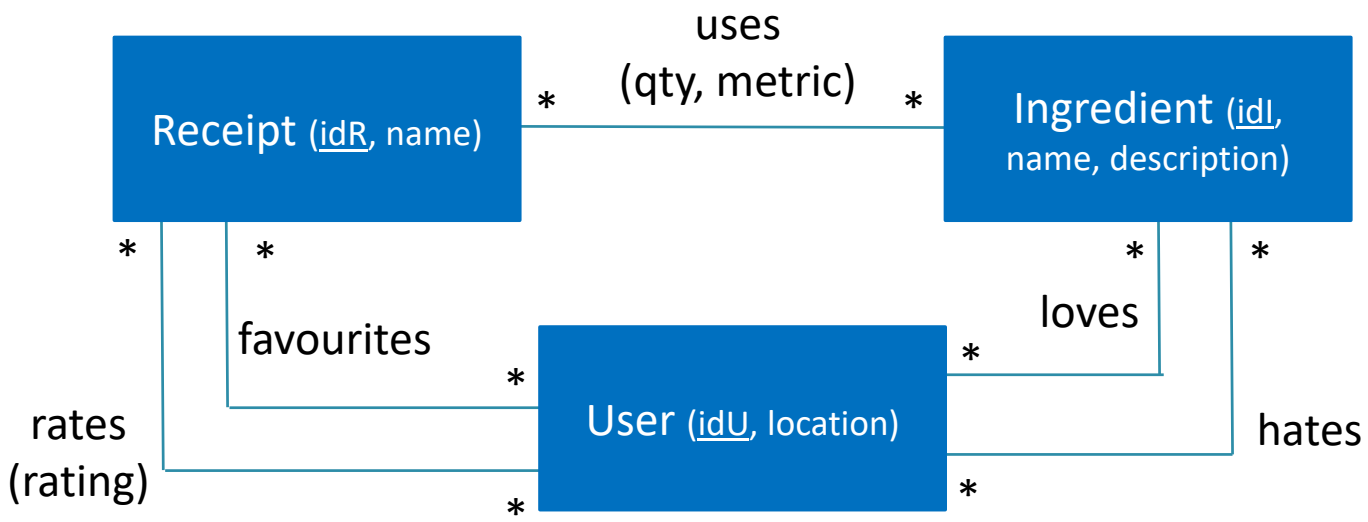
- Massive denormalization
  - If properly handled, it boosts performance (performance-oriented)
  - A natural source of insertion, deletion and update anomalies (due to redundancy)
  - A change in the application layout might be dramatic
    - It may entail a massive rearrangement of the database documents
- Independent documents
  - Avoid pointers (i.e., FKs)
  - References are possible in document-stores but they must be handled with careas they generate joins

Activity

Objective: Recognise the impact of modeling

Tasks:

1. (20') By pairs, answer the following questions:
   I. Come up with at least two different document-oriented models for this schema (e.g., play with the degree of normalization)
   II. Briefly discuss under which assumptions each of these solutions would perform better
2. (5') Discussion



How and When Use Documents?

Documents store perform very well for domain-oriented solutions

- A document-store cannot be designed as a generic solution
- Thus, be aware changes in the system layout may significantly impact on the database

To create domain-oriented solutions we need to model our database considering:

- The system workload
- The internal database structures (specially, the primary indexing strategy)

The difference with regard to key-value is on the structure of the value thus, bear in mind:

- Document-stores do not scale out as much as key-values,
- But they have richer structures (e.g., indexing) and can be better tailored to a problem and perform better

What Document-Stores?

Many out there, CouchDB, MongoDB, ArangoDB, ElasticSearch, Solr…

We will give you the rationale about how to evaluate a document-store with two examples

- MongoDB
- ElasticSearch

You are supposed to apply the same rationale to any other document-store

MONGODB

AN EXAMPLE OF DOCUMENT-STORE

MONGODB: DATA MODEL

AN EXAMPLE OF DOCUMENT-STORE

MongoDB: Data Model

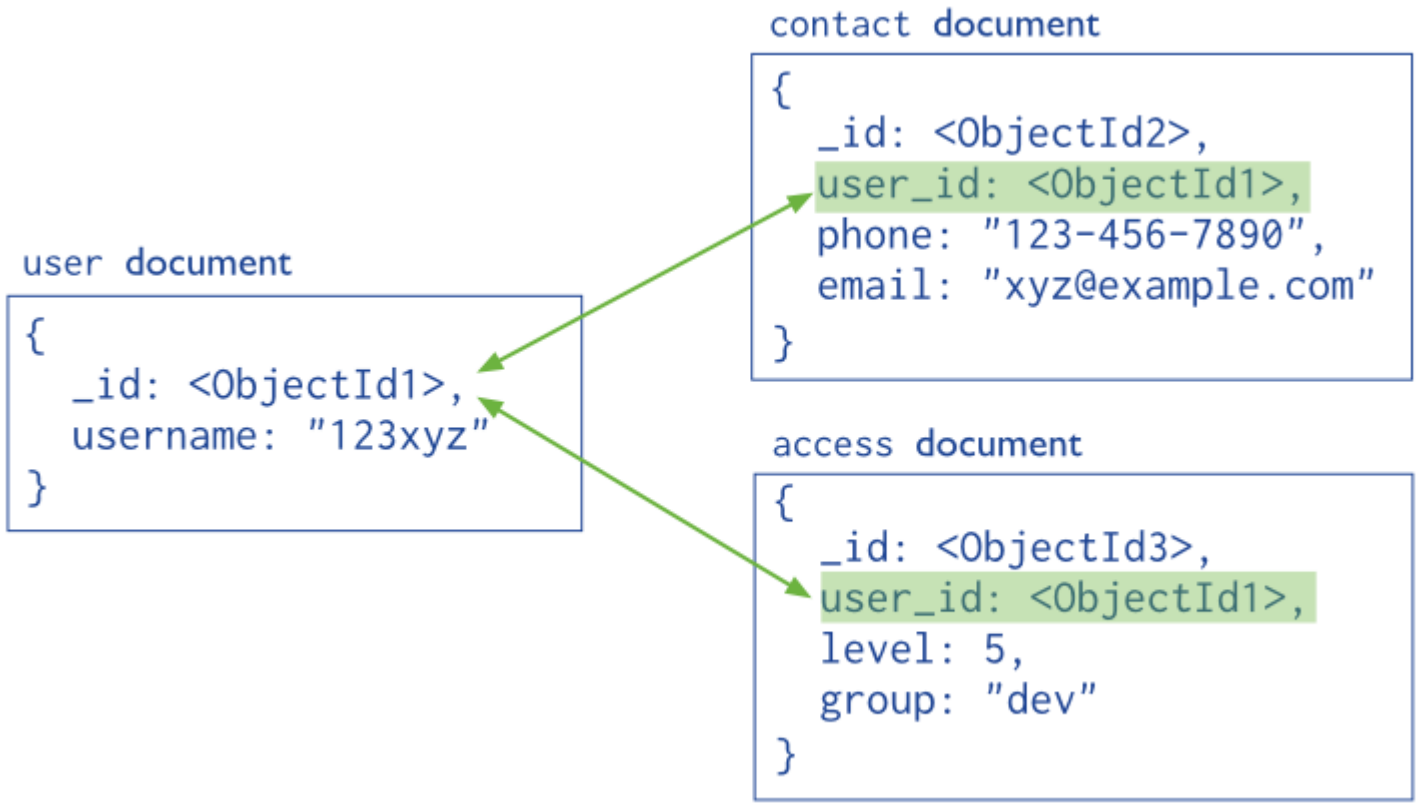
Collections

- *Definition*: A grouping of MongoDB documents
  - A collection exists within a single database
  - Collections do not enforce a schema
- MongoDB Namespace: *database.collection*

Documents

- *Definition*: JSON documents (serialized as BSON)
  - Basic atom
  - Identified by *_id* (user or system generated)
  - Aggregated view of data
  - May contain
    - References (**NOT FKs!**) and
    - Embedded documents

MongoDB: Document Example

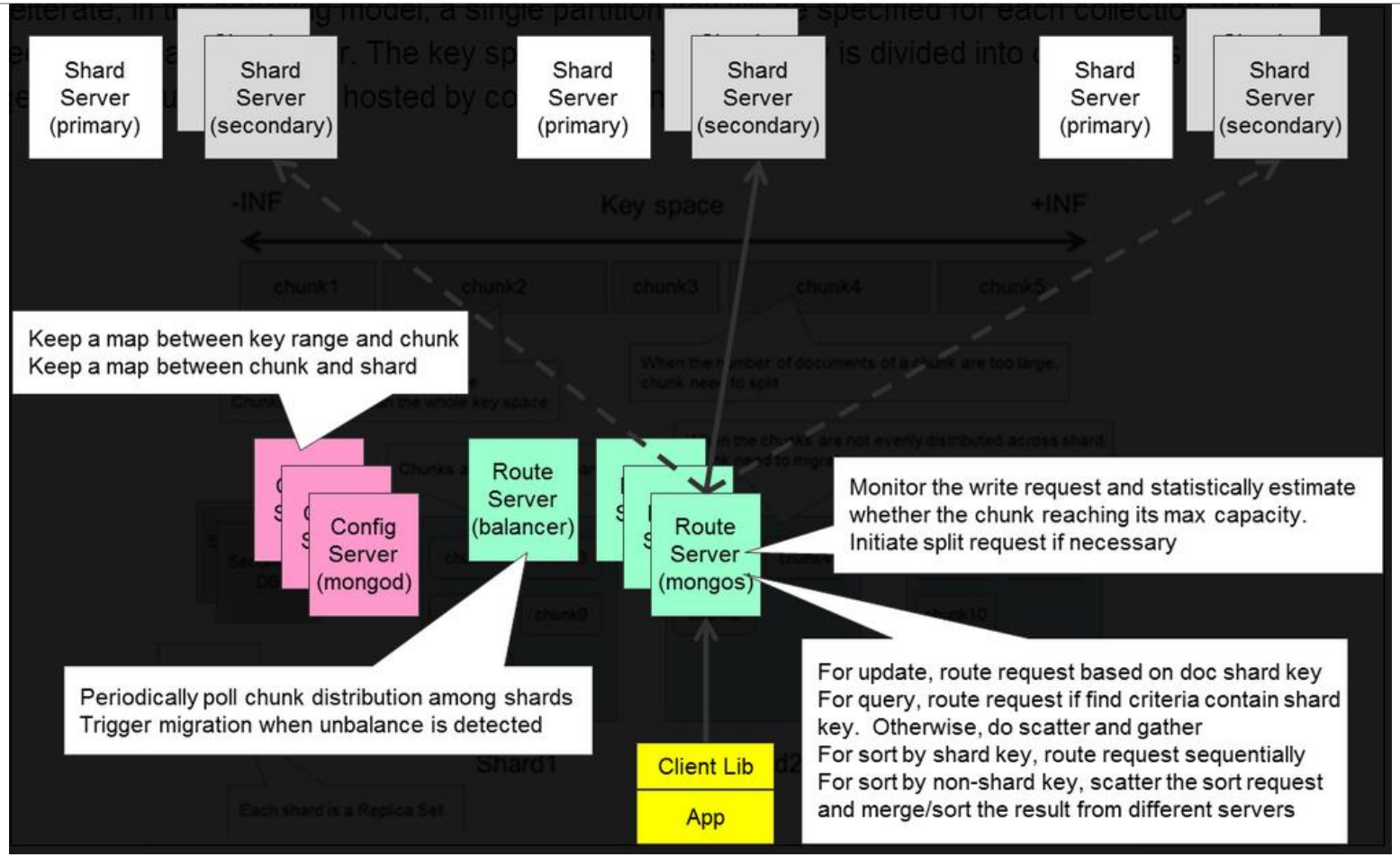


MongoDB: Document Example

MONGODB: ARCHITECTURE

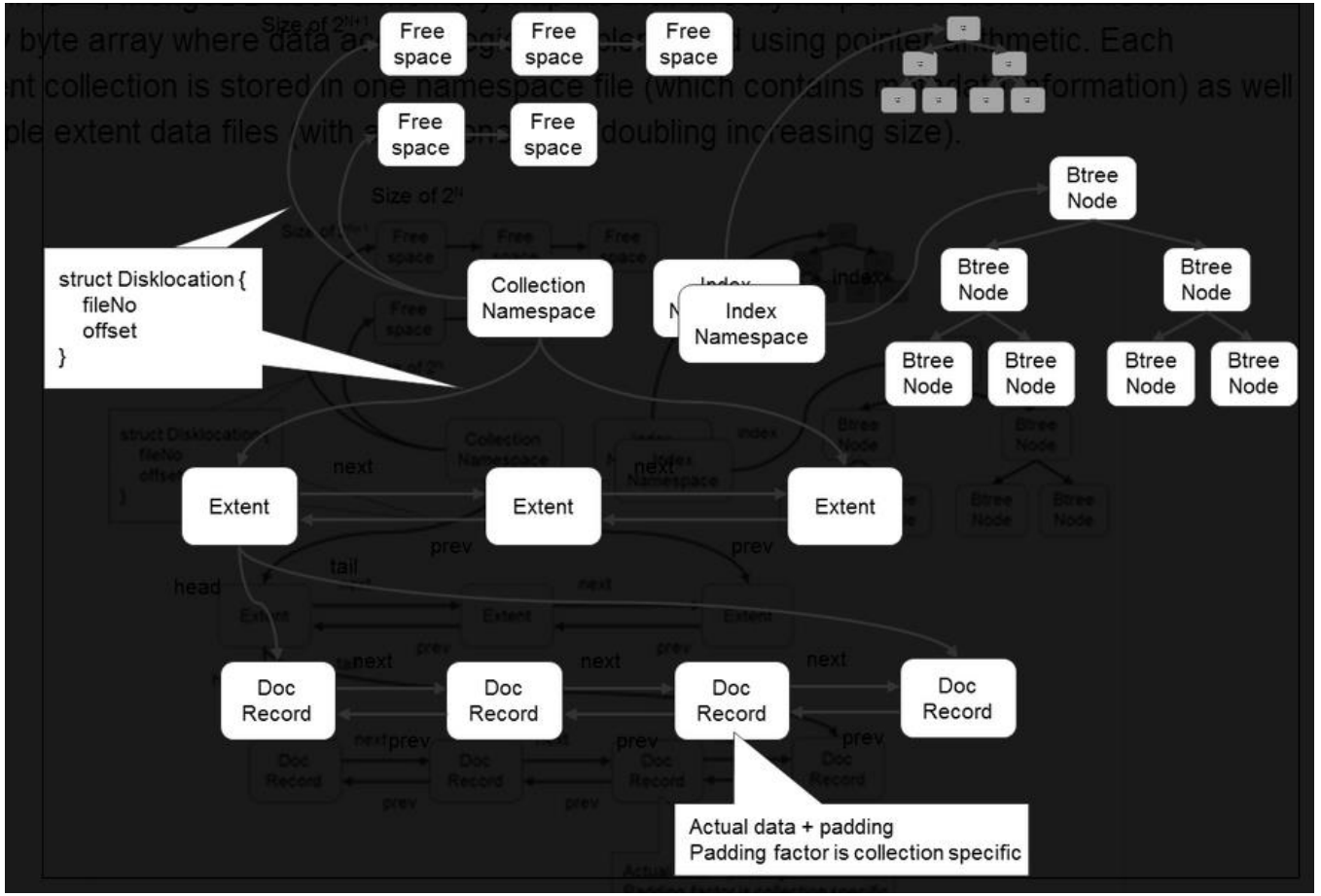
AN EXAMPLE OF DOCUMENT-STORE

MongoDB: Architecture 2.X



http://horicky.blogspot.com.es/2012/04/mongodb-architecture.html

MongoDB: Storage 2.X



http://horicky.blogspot.com.es/2012/04/mongodb-architecture.html

MongoDB 3.0: Main Changes

They introduced the concept of "*pluggable storage architecture*"
- MMAP V1 – based on consistent hashing (see previous slides) – default –
- WiredTiger – based on LSM
- In-Memory (experimental) – Enterprise license

WiredTiger
- Moves to LSM (welcome range queries!)
- Theoretically, we can choose if store data row-oriented or column-oriented (however, up to now, MongoDB has not enabled column-oriented storage)
  - First boosts writes, hurts reads. Just the opposite for the second one
- First-class citizen compression

It fixes the mess with blocking and concurrency
- MVCC: multi-version concurrency control
- At the document level (i.e., no **transaction support**)

Logging is now more efficient
- Still based on write-ahead logging but now checkpointing added

The API and the aggregation framework remain the same

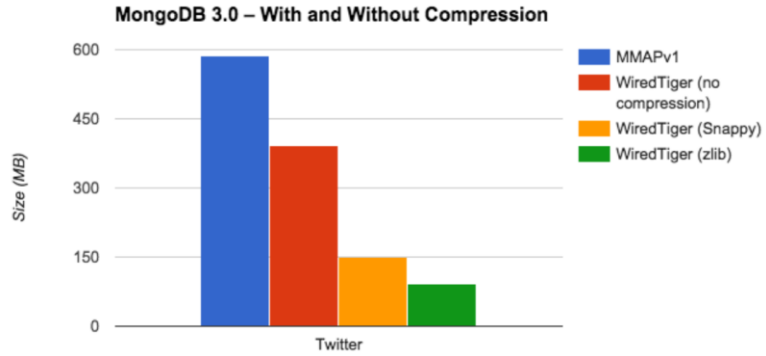
Compression Options

Blocks are read from disk and stored in pages in memory

Typically, you can compress per block or per instance (document) in disk
- Data is decompressed when fetched to memory

WiredTiger introduces three compression options **per collection and per block**
- No compression
- Snappy (by default) – light CPU usage, moderate compression ratios
- Zlib (similar to gzip) – CPU intensive, high compression ratios

Compression options for 200K tweets:



In general, CPU intensive compression algorithms are not interesting in Big Data since they are CPU intensive (and thus, add latency to the I/O operations)
- They are specially interesting to avoid large redundancies. For example, compress text or the JSON field names (remember MongoDB has no catalog)

https://www.mongodb.com/blog/post/new-compression-options-mongodb-30

Compression Options

WiredTiger also introduces prefix compression for indexes

- Data is not decompressed when fetched in memory
- Searches can still be applied on compressed data

Prefix compression:

0, 0d, 0ful, 0fulness **[Use->0]**

Performance Test:

Collection *db* contains 10 million documents:

```
{
   employeeID: <long>,
   firstName: <string>,
   lastName: <string>,
   income: <long>,
   supervisor: {ID: <long>, 'firstName': <string>,
'lastName': <string>;}
}
```

We create the following indexes:

```
Index 1: db.ensureIndex({'employeeID':1});
Index 2: db.ensureIndex({'lastName':1, 'firstName':1});
Index 3: db.ensureIndex({'income':1});
Index 4: db.ensureIndex({'supervisor.lastName':1,
'supervisor.firstName':1})
```

Comparison:

Index name	MMAP index size (MB)	WT Index size (MB)	% Reduction in size
{employeeID:1}	230.7	94	59%
{lastName:1, firstName:1}	1530	36	97%
{income:1}	230	94	59%
{'supervisor.lastName':1, 'supervisor.firstName':1}	1530	35	97%

https://scalegrid.io/blog/index-prefix-compression-in-mongodb-3-0-wiredtiger/

MongoDB: Querying

Find and findOne methods (~query by example)

- database.collection.find()
- database.collection.find(*{ qty: { $gt: 25 } }*)
- database.collection.find(*{ field: { $gt: value1, $lt: value2 } }*);

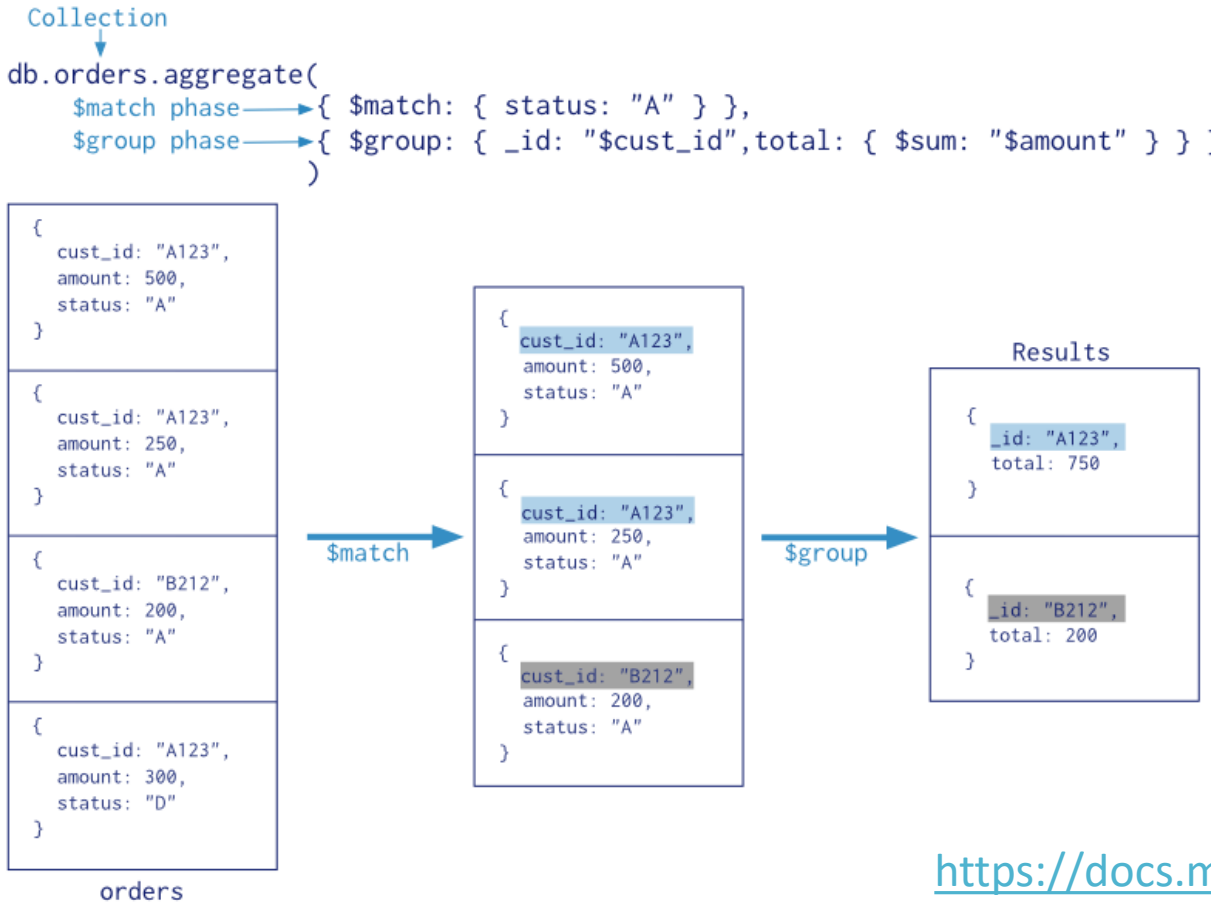
The Aggregation Framework

- An aggregation pipeline
- Documents enter a multi-stage pipeline that transforms the documents into an aggregated results
  - *Filters* that operate like queries
  - *Document transformations* that modify the form of the output
  - Grouping
  - Sorting
  - Other operations

MapReduce

https://docs.mongodb.com/manual/core/map-reduce/
https://runnable.com/blog/pipelines-vs-map-reduce-to-speed-up-data-aggregation-in-mongodb

The Aggregation Framework



https://docs.mongodb.com/manual/aggregation/

Architectural Limitations

Architectural Issues

- Thumb rule: 70% of the database must fit in memory
- Be careful with updates! (padding)
  - Holes caused by reallocation
  - Compact the database from time to time
  - In WiredTiger this is left for compactation (the delta memstore smooths it)
- Limited number of collections per database
- A database cannot be bigger than 32TBs
- Theoretically, sharding is automatic and transparent. But in practice it is not. Most typical ones:
  - Size of the sharding key is limited (512 bytes)
  - Max. number of elements to migrate (when balancing the workload)
  - LSM + sequential keys will hit only one node (be careful with the key!!)
  - Parallelism is heavily limited by the aggregation framework provided

Document Issues

- The resulting document of an aggregation pipeline cannot exceed the maximum document size (16Mb)
  - GridFS for larger documents
- No more than 100 nesting levels (i.e., embedded documents nesting)
- Attribute names are kept as they are (no catalog)

http://docs.mongodb.org/manual/reference/limits/

ELASTICSEARCH

AN EXAMPLE OF DOCUMENT-STORE

ELASTICSEARCH: DATA MODEL

AN EXAMPLE OF DOCUMENT-STORE

ElasticSearch: Data Model

Documents

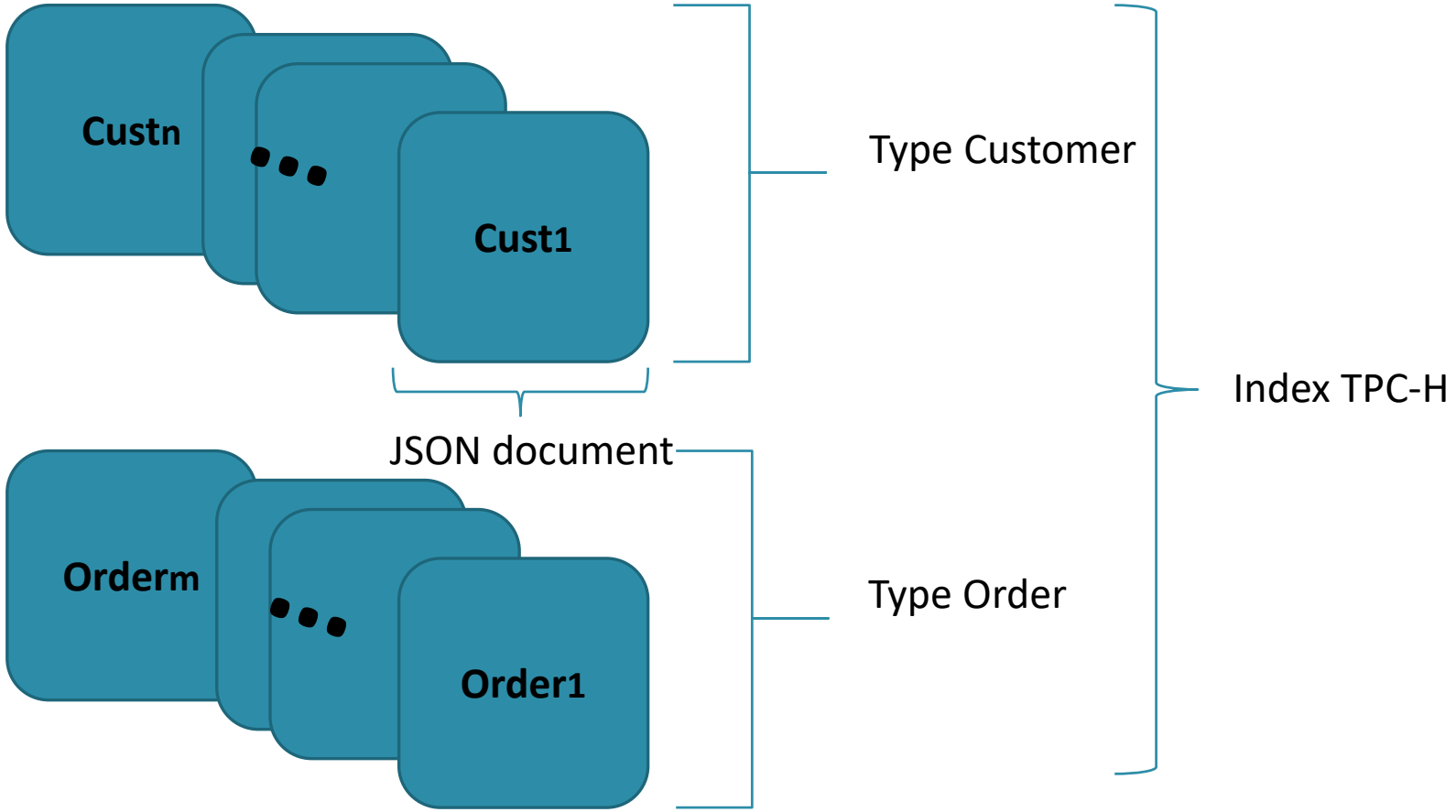
JSON documents

- Basic atom
- Besides your data, the document contain metadata added by ElasticSearch:
  - The document *id* (user or system generated)
  - Type of the document (class it belongs to) –**deprecated in 2.X**
  - Index or logical namespace it belongs to
  - A version
- A document is univocally identified by **index + type + id**

Types are like class labels you add to your documents

An index is a logical group of documents

ElasticSearch Data Model: Example



ELASTICSEARCH: ARCHITECTURE

AN EXAMPLE OF DOCUMENT-STORE

Apache Lucene

ElasticSearch builds on top of the well-known ***Lucene***[1] indexing library

Lucene is an Information Retrieval (IR) tool, specifically, a Text Retrieval tool

- Indexes text
- Enables **full-text** search

It was the first IR tool for Big Data

- It distributes on a cluster
- Incremental indexing

[1] https://lucene.apache.org/core/

Full-Text Search

Assume a data corpus or a text database DBt, which contain set of documents D, each of them containing free text T

Full-text search refers to the capability of searching the whole DBt

Two main phases: indexing and search

Indexing

Words (or sentences) are indexed

- Typically, NLP processing is applied prior to indexing (e.g., removing stop words, tokenize, etc.)

One main indexing structure: **inverted indexes**

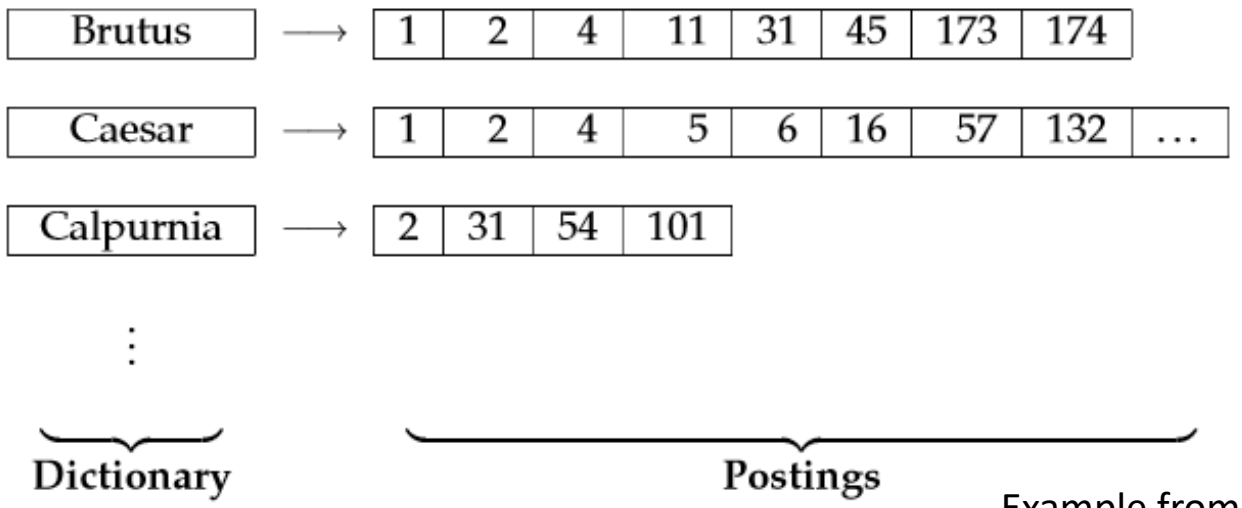
- They are regular B-tree indexes
  - The key is a word
  - The value is the list of documents and the position within the document, where you can find such word

Inverted Index

Two structures

- The dictionary or lexicon: list of all words (typically, dictionary entries are called terms)
- For each term, the index keeps trace of its postings (i.e., documents, and positions within the document, where it appears)

An Inverted Index



Example from: https://nlp.stanford.edu/IR-book/

Steps to Build an Inverted Index

We assume a text database, which contains a list of documents and each document has an associated id

1) NLP pre-processing, removing stop words, tokenizing, …
2) For each doc, for each term, generate <*term, 1, docID, pos*>, where 1 is the frequency counter
3) Sort by term the whole list of quadruples generated
4) When two consecutive positions refer to the same term, merge terms and lists and sum the frequency counter
5) From the resulting list, generate a B-tree

Activity

Objective: Understand inverted indexes

Given the following docs:

- Doc1: I did enact Julius Caesar: I was killed in the Capitol; Brutus killed me.
- Doc2: So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.

Tasks:

1. (10') By pairs, answer the following questions:
   I. Remove stop words, tokenize text
   II. Build the <term, frequency, docID, pos> quadruples
   III. Sort and merge the quad list
   IV. [Generate the inverted tree]
2. (5') Discussion

Search

Even if you can perform the typical equality / range search on an inverted index, in IR a search typically:

- Returns top-K results depending on the meaning of the search,
  - Measuring relevance
  - Any metric from information retrieval: tf-idf, pagerank, etc.
- Shows the results contextualised
  - It is related with the task of ***highlighting***

Example: **Julius Caesar – Wikipedia**
https://en.wikipedia.org/wiki/Julius_Caesar
Gaius Julius ***Caesar*** usually called Julius ***Caesar***, was a Roman politician and general who played a critical role in the events that led to the demise of the Roman Republic and the rise of the Roman Empire. He is also known as a notable author of Latin prose. In 60 BC, ***Caesar***, Crassus and Pompey formed a political ...Caesar (disambiguation) · Temple of Caesar · Julius Caesar (play) · Caesar's Comet

Index Only Query Answering

It is possible to use several indexes (each of them indexing a different term) to answer more complex queries

The theoretical background is the same as that of index-only query answering

AND / OR predicates can be naturally resolved with the index intersection algorithm

- Caesar AND Brutus
- Caesar OR Brutus
- NOT Caesar AND Brutus

Index Intersection Algorithm

1. Put the predicate in CNF
   1. Remove negations of parenthesis
      - NOT (A OR B) = NOT A AND NOT B
      - NOT (A AND B) = NOT A OR NOT B
   2. Move disjunctions into the parenthesis
      - (A AND B) OR C = (A OR C) AND (B OR C)
      - (A AND B) OR (C AND D) = (A OR C) AND (A OR D) AND (B OR C) AND (B OR D)
      - (A AND B) OR (C AND B) = (A OR C) AND B
2. For each predicate, use the available index to retrieve the list of RIDs fulfilling that predicate
3. For each disjunction or conjunction (apply in order)
   1. Intersect the list of RIDs in the conjunction
   2. Unite the list of RIDs in the disjunction
4. The resulting RIDs meet the query predicate
   1. Use TF-IFD or any other criteria to rank results
   2. Contextualise results

Example: Index Intersection

Query: Caesar AND Brutus

The predicate is already in CNF

Use the inverted index for Caesar and the inverted index for Brutus separately

- Caesar: doc1, doc2
- Brutus: doc1, doc2

For the results obtained, since it is a conjunction, only the results in BOTH lists should be retrieved

- Result: doc1, doc2

Rank results: $TFD\text{-}IFD_{doc1} > TFD\text{-}IFD_{doc2}$

Contextualise results (e.g., bring whole paragraph)

About Lucene and ElasticSearch

Lucene is very powerful but it is difficult to use. The user needs to understand the whole process explained in these slides

ElasticSearch aims at making all this process easier

- It automatically deals with Analyzers, QueryParser, IndexSearcher, etc.

Realise though that you need to understand what is going on in the background

- As any other NoSQL tool, it cannot be used as a blackbox!

They fully rely on indexing (therefore they cannot deal with high-throughput data by definition)

GRAPH DATABASES

RELATIONSHIPS AS FIRST-CLASS CITIZENS

Graph Databases in a Nutshell

Occurrence-oriented

- May contain millions of instances
  - Big Data!
- It is a form of schemaless databases
  - There is no explicit database schema
  - Data (and its relationships) may quickly vary
- Objects and relationships as first-class citizens
  - *An object o relates (through a relationship r) to another object o'*
  - Both objects and relationships may contain properties
- Built on top of the graph theory
  - Euler (18th century)
  - More natural and intuitive than the relational model

An Example: Will They Split Up?

Crossing data from social networks it is possible to identify a graph like the one that follows:

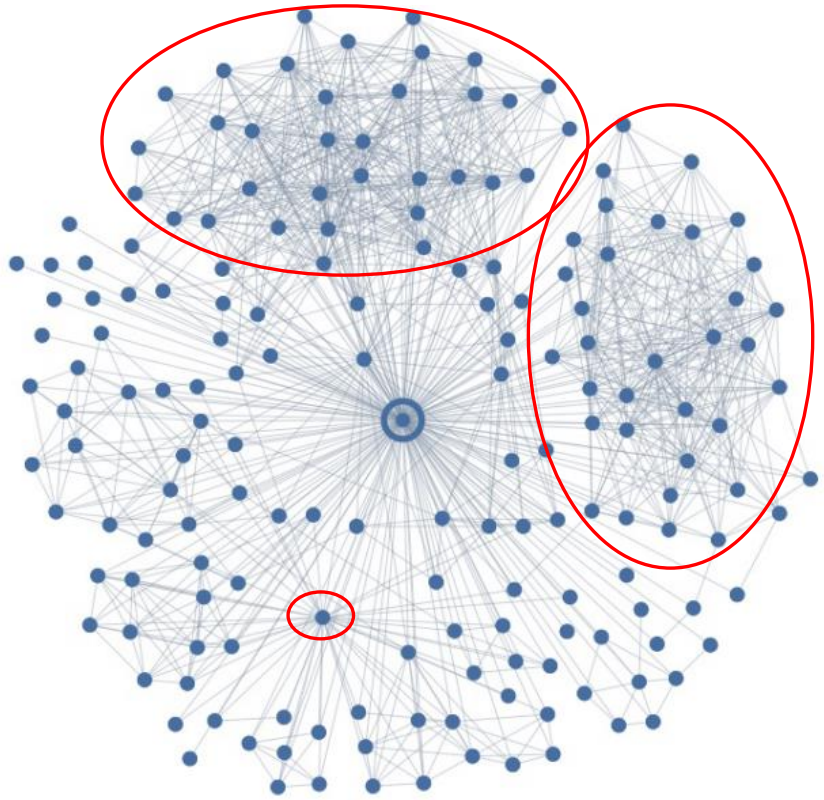
- In the centre there is a specific person *P*
- The rest are *P* connections and connections among them

Using sociology techniques…

- We can identify *P social foci*:
  - Dense clusters of friends corresponding to long periods of interaction
  - Typically, college friends, coworkers, relatives, etc.
- The *significant other* can be identified by a high *dispersion* rate
  - Highly connected with *P* connections,
  - But with a high dispersion degree wrt *P* social foci

Hypothesis: when the node with higher dispersion degree Identified is not the partner, this couple is likely to split up in a period of 60 days

L. Backstrom, J. Kleinberg. Romantic Partnerships and the Dispersion of Social Ties: A Network Analysis of Relationship Status on Facebook https://arxiv.org/pdf/1310.6753v1.pdf



Notation (I)

A **graph** *G* is a set of nodes and edges: *G (N, E)*

N - **Nodes** (or vertices): n1, n2, … Nm

E - **Edges** are represented as pairs of nodes: (n1, n2)

- An edge is said to be **incident** to n1 and n2
- Also, n1 and n2 are said to be **adjacent**
- An edge is drawn as a line between n1 *and* n2
- **Directed edges** entail direction: *from* n1 *to* n2
- An edge is said to be **multiple** if there is another edge exactly relating the same nodes
- An **hyperedge** is an edge inciding in more than 2 nodes.

Multigraph: If it contains at least one multiple edge.

Simple graph: If it does not contain multiple edges.

Hypergraph: A graph allowing hyperedges.

Notation (II)

Size (of a graph): #edges

Degree (of a node): #(incident edges)
- The degree of a node denotes the node adjacency
- The neighbourhood of a node are all its adjacent nodes

Out-degree (of a node): #(edges leaving the node)
- Sink node: A node with 0 out-degree

In-degree (of a node): #(incoming edges reaching the node)
- Source node: A node with 0 in-degree

Cliques and trees are specific kinds of graphs
- **Clique**: Every node is adjacent to every other node
- **Tree**: A connected acyclic simple graph

The Property Graph Data Model

Two main constructs: nodes and edges.

- Nodes represent entities,
- Edges relate pairs of nodes, and may represent different types of relationships.

Nodes and edges might be labeled,

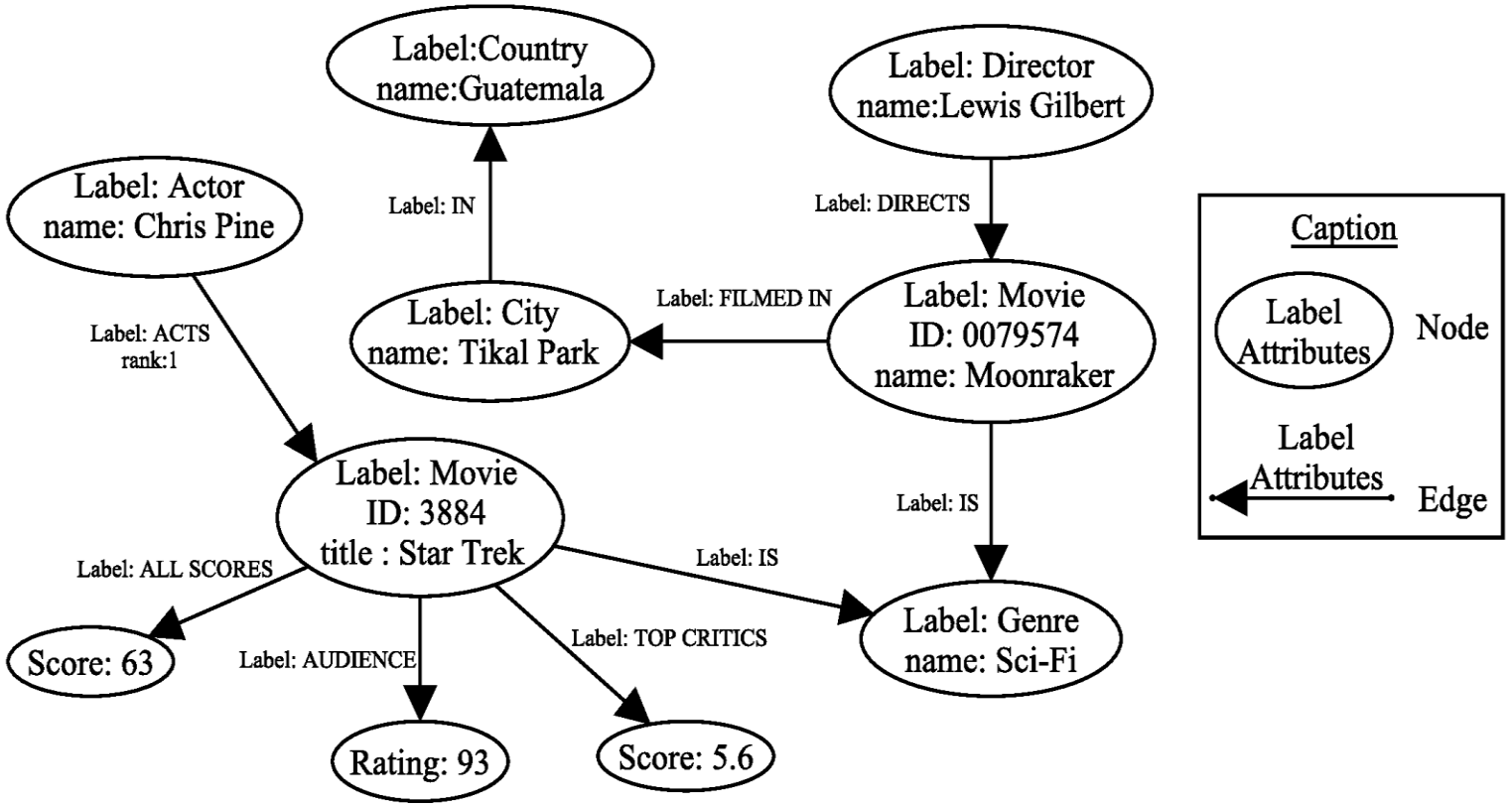
and may have a set of properties represented as attributes (key-value pairs)***

Further assumptions:

- Edges are directed,
- Multi-graphs are allowed.

*** *Note: in some definitions edges are not allowed to have attributes*

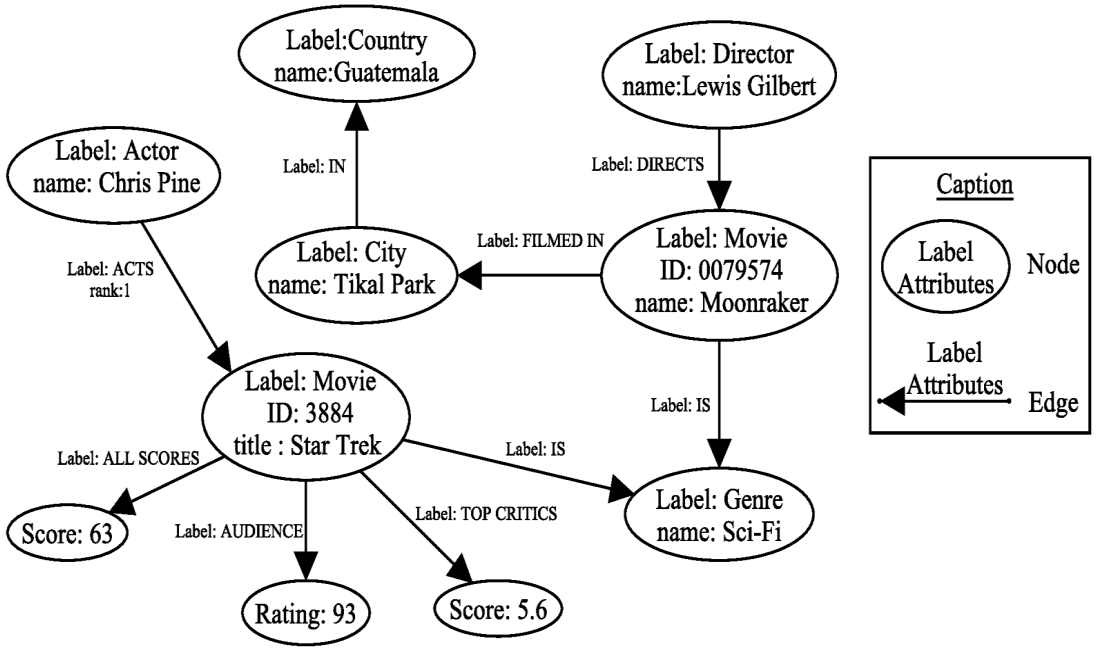
Example of Graph Database



http://grouplens.org/datasets/movielens/

Activity: Querying Graph Databases

Objective: understand the main differences with regard to RDBMS

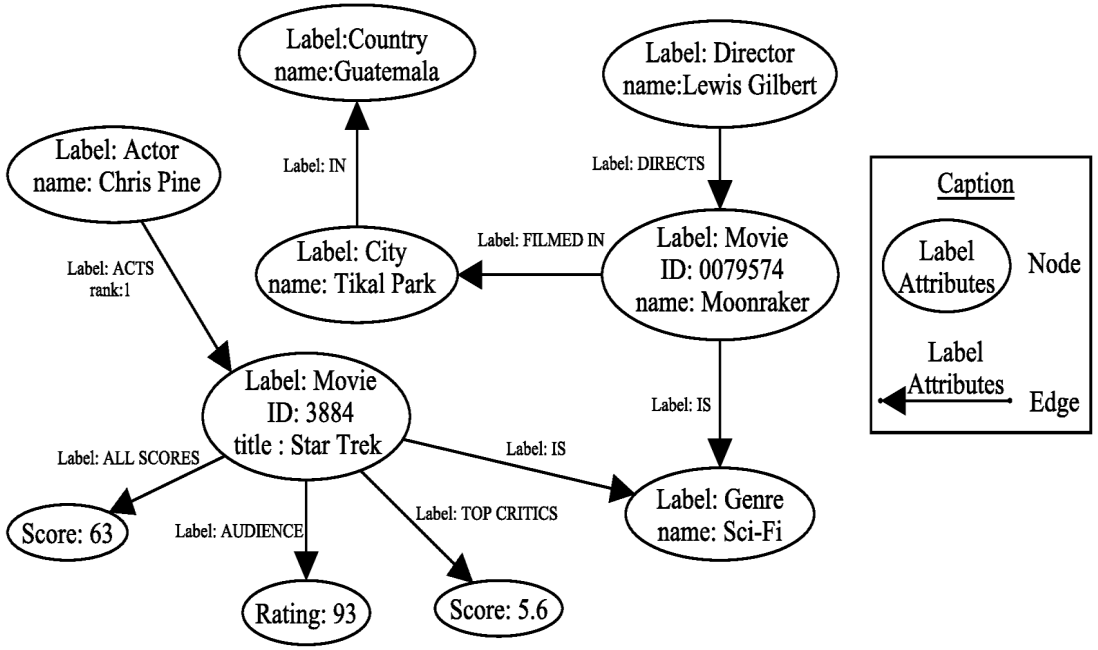


What movies did Lewis Gilbert direct?

- How would a RDBMS perform this query?

Activity: Querying Graph Databases

Objective: understand the main differences with regard to RDBMS



What movies did receive a rating lower than 60 by the audience?

- How would a RDBMS perform this query?

GDBs Keystone: Traversal Navigation

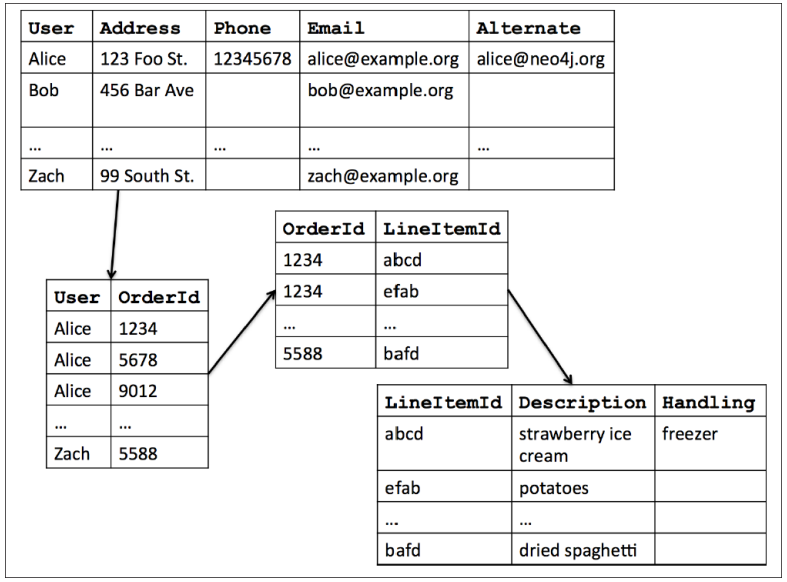
We define the graph traversal pattern as: *"the ability to **rapidly** traverse structures to an **arbitrary depth** (e.g., tree structures, cyclic structures) and with an **arbitrary path description** (e.g. friends that work together, roads below a certain congestion threshold)"* [Marko Rodriguez]

Totally opposite to set theory (on which relational databases are based on)

- Sets of elements are operated by means of the relational algebra

Traversing Data in a RDBMS

In the relational theory, it is equivalent to **joining** data (schema level) and select data (based on a value)



User	Address	Phone	Email	Alternate
Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
Bob	456 Bar Ave		bob@example.org	
...	...	...	...	...
Zach	99 South St.		zach@example.org	

User	OrderId
Alice	1234
Alice	5678
Alice	9012
...	...
Zach	5588

OrderId	LineItemId
1234	abcd
1234	efab
...	...
5588	bafd

LineItemId	Description	Handling
abcd	strawberry ice cream	freezer
efab	potatoes	
...	...	
bafd	dried spaghetti	

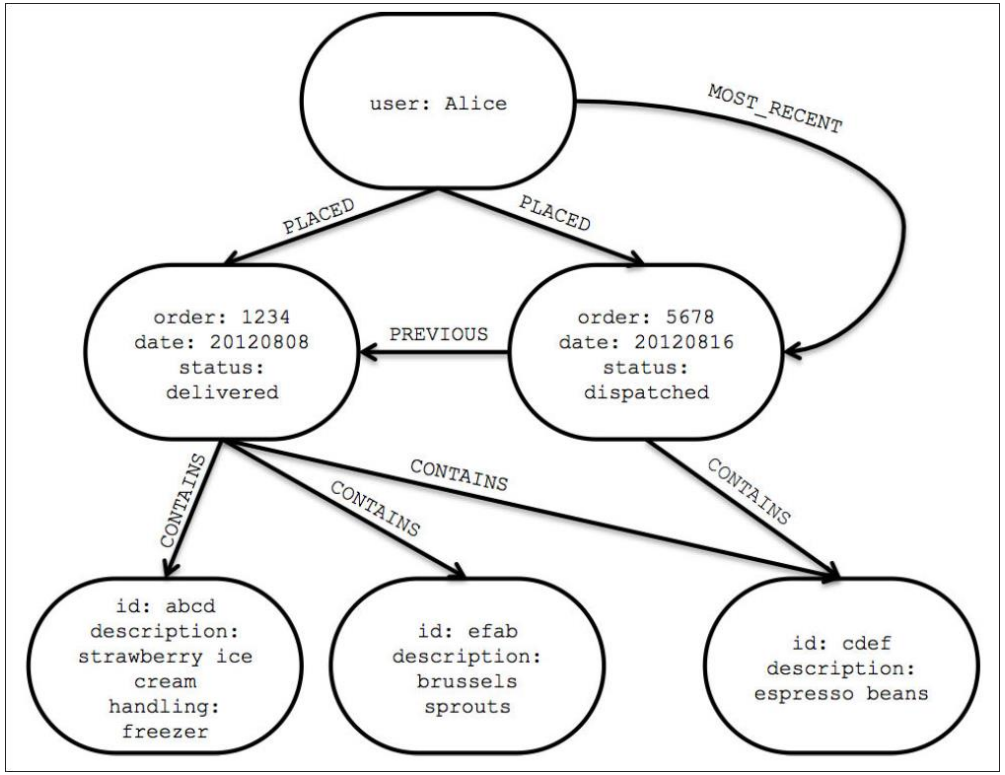
```
SELECT *
FROM user u, user_order uo,
orders o, items i
WHERE u.user = uo.user AND
uo.orderId = o.orderId AND
i.lineItemId = i.LineItemId
AND u.user = 'Alice'
```

Cardinalities:

|User|: 5.000.000
|UserOrder|: 100.000.000
|Orders|: 1.000.000.000
|Item|: 35.000

Query Cost?!

Traversing Data in a Graph Database



Cardinalities:

|User|: 5.000.000
|Orders|: 1.000.000.000
|Item|: 35.000

Query Cost?!

O(N)

TYPICAL GRAPH OPERATIONS

REFRESHING SOME BASICS ON GRAPHS

Typical Graph Operations

Content-based queries

- The value is relevant
  - Get a node, get the value of a node / edge attribute, etc.
  - A typical case are summarization queries (i.e., aggregations)

Topological queries

- Only the graph topology is considered
- Typically, several business problems (such as fraud detection, trend prediction, product recommendation, network routing or route optimization) are solved using graph algorithms exploring the graph topology
  - Computing the betweenness centrality of a node in a social network an analyst can detect influential people or groups for targeting a marketing campaign audience.
  - For a telecommunication operator, being able to detect central nodes of an antenna network helps optimizing the routing and load balancing across the infrastructure.

Hybrid approaches

Topological Queries (I)

Categories of queries

- Adjacency queries
  - Basic node / edge adjacency
  - K-neighbourhood of a node

    Linear cost (on the number of edges to explore)

    Examples: Return all the friends of a person
- Reachability queries (formalized as a traversal)
  - Fixed-length paths (fixed #edges and nodes)
  - Regular simple paths (restrictions as regular expressions) – Hybrid if the restriction is in the *content*
  - Shortest path

    Hard, to compute, in general, NP-complete

    Examples: Friend-of-a-friend

Topological Queries (II)

Categories of queries (cont'd)

- Pattern matching queries (formalized as the graph isomorphism problem)
  - Hard, to compute, in general, NP-complete
  - Examples: people without telephone
- Graph metrics
  - Compute the graph / node order, the min / max degree in the graph, the length of a path, the graph diameter, the graph density, closeness / betwenness of a node, the pageRank of a node, etc.

    Depends on the metric to be computed.

    Examples: Compute business processes bottlenecks (betweenness)

Topological Queries

Support provided by current GBDs

	Adjacency		Reachability				
Graph Database	Node/edge adjacency	k-neighborhood	Fixed-length paths	Regular simple paths	Shortest path	Pattern matching	Summarization
Allegro	•		•			•	
DEX	•		•	•	•	•	
Filament	•		•			•	
G-Store	•		•	•	•	•	
HyperGraph	•					•	
Infinite	•		•	•	•	•	
Neo4j	•		•	•	•	•	
Sones	•					•	
vertexDB	•		•	•		•	

R. Angles. A Comparison of Current Graph Database Models (as of 2012)

Hybrid Queries

Emerging categories

- Mining queries: Finding all frequent subgraphs and patterns
  - Frequent subgraph mining: Discover frequent subgraphs from a set of graphs.
  - Frequent proximity pattern mining: Strict isomorphism is not desired. Based on distances.

    Examples: online recommendation, viral marketing, intrusion detection
- Selection queries
  - Graph Skyline queries: given a set of multidimensional points, retrieve those points such as none is dominated by any other
    - Typically computed from the distance plus some similarity metrics

      Examples: Typically used to correlate several criteria (e.g., find the top-k co-actors of a given actor)

Thanks! Any Question?

OROMERO@ESSI.UPC.EDU

HOMEPAGE: HTTP://WWW.ESSI.UPC.EDU/DTIM/PEOPLE/OROMERO

TWITTER: @ROMERO_M_OSCAR