

# NoSQL Storage Systems

A Data Modeling Techniques Perspective

---

Guillaume Raschia — Nantes Université

Last update: November 16, 2023

1

## R-DBMS Limitations

---

### Contents

R-DBMS Limitations

NoSQL Models

Polyglot Persistence

NewSQL

2

### The R-DBMS World in Brief

#### Relational Model from the 70's

- tables, rows, fixed columns
- attributes are pre-defined in a schema
- CREATE TABLE-like statements (DDL part of SQL)
- User-centered query language (SQL) as an abstraction layer

#### DB Design

Normalization normalization normalization!

- prevent from redundancy and anomalies

3

## The R-DBMS World in Brief (cont'd)

### ACID Transactions

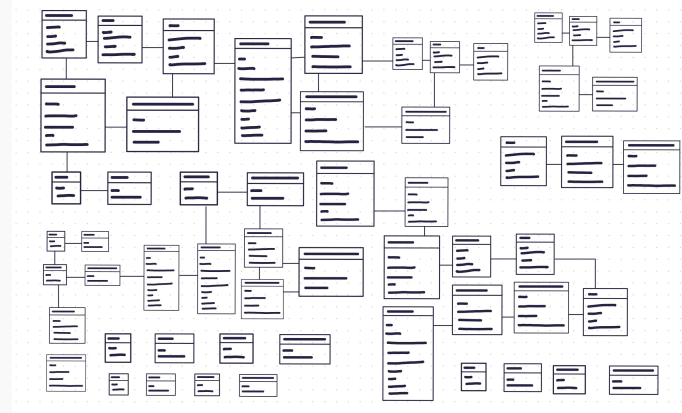
Atomicity - Consistency - Isolation - Durability

### Consistency and integrity enforcement

- Schema-data consistency
- Domain constraints (value range, regexp)
- References (FK and inclusion dependencies)
- Uniqueness (PK)
- Tuple-based constraints (e.g.,  $R.A > R.B/2$ )

4

## Relational Model: a –Sort of– Real-Life Example



Source: [Data Model](#), Directus docs.

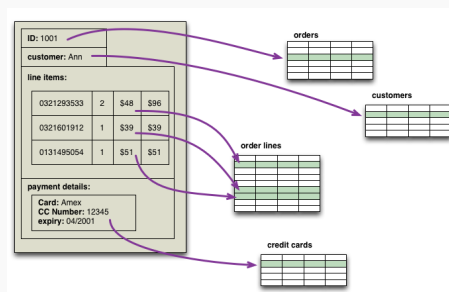
6

## Relational Database Limitations

### Impedance mismatch

How to fit complex objects from PL to flat tables ?!

- decomposition into many vertical fragments: ORM to the rescue



Source: [Aggregate Oriented Database](#), M. Fowler, 19 Jan. 2012.

5

## Database Tuning

### Overcome the performance bottleneck

- Hardware/system/software architecture (e.g. caching)
- Query workload and application-driven model
  - Indexes, views, hints, SQL S-F-W rewriting
  - Denormalization : introduce redundancy
  - Clustering relations : speed up join queries
  - Partitioning

### The ultimate DB system

Pre-compute query answers and cache them locally!

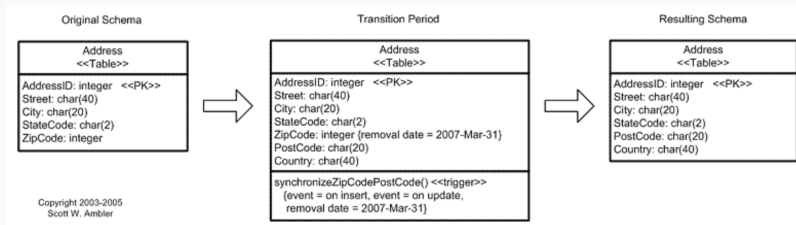
7

## Relational Database Limitations (bis)

### Evolving/unknown structural requirements

#### Database refactoring

- alter schema + migrate data is expensive
- don't forget to sync apps



8

## Scaling

### Vertical scaling

- Upgrade hardware resources: CPU/RAM/Disk
- Expensive and limited to the most recent technologies

### Parallel processing and data distribution

#### Not that easy

- Drawback n°1: global locking/logging for ACID Tx
- Drawback n°2: joins on multiple nodes

10

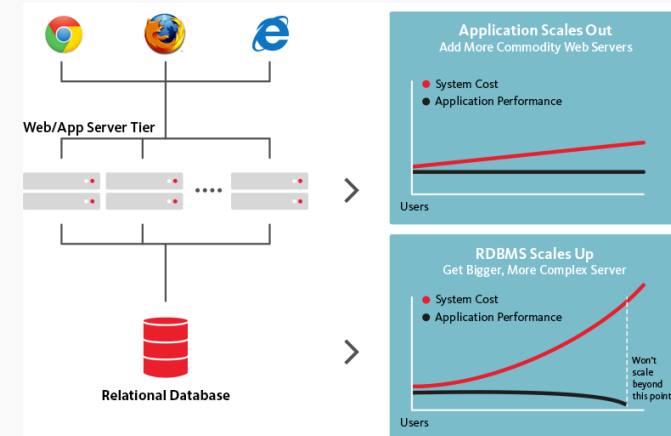
## Relational Database limitations (ter)

### Overhead cost at run time

- the price to pay for the **friendly declarative layer** :
  - parsing, optimization, sub-optimal execution, buffer management
- the price to pay for the **normalization** :
  - recomposition of entities with deadly joins
- the price to pay for the **consistency of ACID Tx** :
  - latches, locks and logs management

9

## Scale Up vs. Scale Out



Source: Database Scaling Made Simple, DZone (src. Paris Technology), 24 May 2017.

11

## NoSQL

## Key Concepts

- **Aggregate** data model
  - group logical pieces of data units and distribute by key (DHT)
  - duplicate pieces among aggregates if necessary
  - must conform to data access patterns
- “Free structure”
  - **schema-less** design
- Data **distribution** over many servers
  - Horizontal partitioning (aka. *sharding*) w.r.t. query scope: increase data volume
  - Replication: increase fault tolerance (availability)
- **Weak consistency** (vs. ACID Tx)
- **Low-level call interface** (vs. SQL)

13

## No(t only)SQL

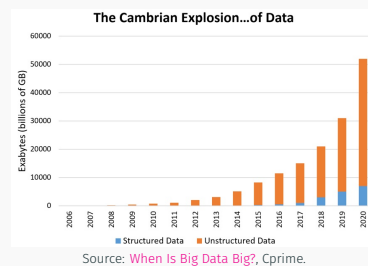
No standard definition

New apps for new requirements

- “Cloud OLTP” vs. traditional [TPC-C]-like serving workload
- “big OLAP” addresses Volume-Velocity-Variety-Veracity

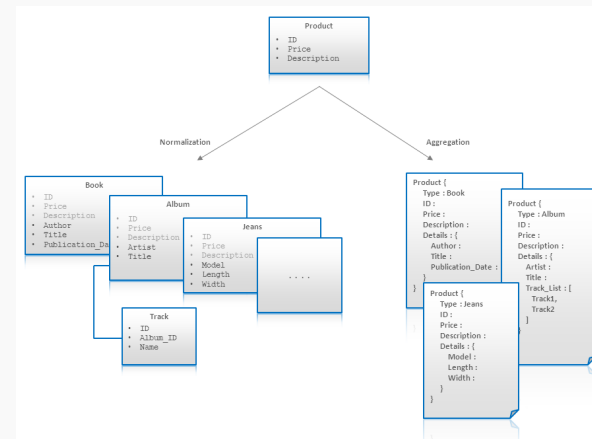
New (big) data

90% of the World’s data was created in the last 2 years



12

## The Aggregation Principle



Source: [NoSQL Data Modeling Techniques](#), Highly Scalable Blog, 01 March 2012.

14

## Distribution: main techniques, main ideas

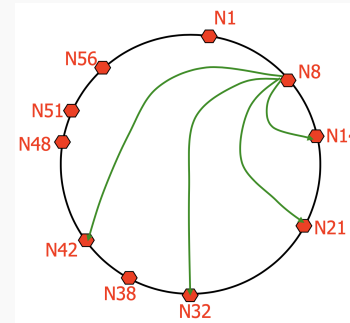
1. Distributed Hash Tables (DHT)
  - Rendezvous Hashing, Consistent Hashing
2. Consistency: 2PC and Paxos (strong), Vector Clocks (eventual)
3. The CAP Theorem

### Warning

- Much more to do with distributed systems rather than a databases course
- But super relevant to NoSQL

15

## Ring DHT



### Finger Table at Node $N_8$

idx	hashcode	root
0	$8 + 2^0 = 9$	$N_{14}$
1	$8 + 2^1 = 10$	$N_{14}$
2	$8 + 2^2 = 12$	$N_{14}$
3	$8 + 2^3 = 16$	$N_{21}$
4	$8 + 2^4 = 24$	$N_{32}$
5	$8 + 2^5 = 40$	$N_{42}$

find root of K54 from  $N_8$  : to  $N_{42}$ , then  $N_{51}$ , and finally reach next node ( $N_{56}$ )

17

## Distributed Hash Table

Implements a distributed storage over  $N$  servers

- Each key-value pair  $(k, v)$  is stored at some server  $h(k)$
- API : `write(k, v)` ; `read(k)`

Use standard hash function : service key  $k$  by server  $h(k)$

- Problem n°1 : a client knows only one random server, doesn't know how to access  $h(k)$
- Problem n°2 : if new server joins, then  $N \rightarrow N + 1$ , and the entire hash table needs to be reorganized
- Problem n°3 : we want replication, i.e. store the object at more than one server

16

## CAP Theorem : Pick 2 out of 3!

Conjecture from E. Brewer (PODC 2000)

Proof from S. Gilbert and N. Lynch (SIGACT News 2002)

### Consistency

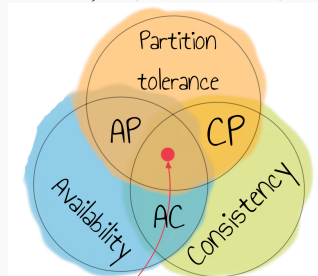
All nodes see the same data at the same time

### Availability

Every request receives a response about whether it succeeded or failed

### Partition tolerance

The system continues to operate despite arbitrary partitioning due to network failures



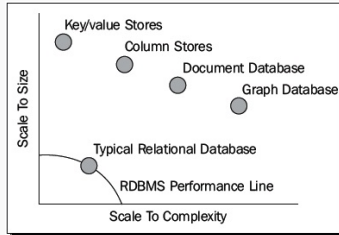
Source: [Big Data World, Part 5: CAP Theorem](#), P.

Finkelshteyn, JetBrains Blog, 3 June 2021.

18

## The NoSQL Data Models

Warning: Typical Ad for NoSQL but fake news inside...



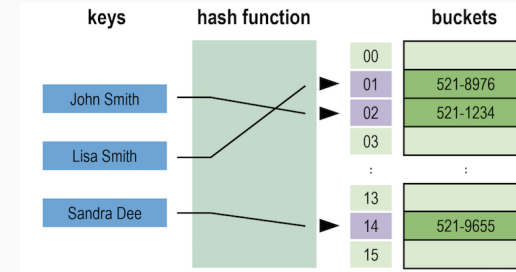
- There exist tons of data stores : see **NOSQL**
- **XML and RDF Stores** are in-between Document and Graph Stores
- **Column Stores** are actually stores of **Extensible Records**, a.k.a. Columnar/Column-Family/Wide Column Stores

19

## K-V Store (cont'd)

Data structure of an associative array, map or dictionary

- Hash partitioning with **consistent hashing**
- Distributed Hash Tables (**DHT**)



SVG source file "Hash table 4 1 1 0 0 1 0 LL" by J. Stolfi - Own work

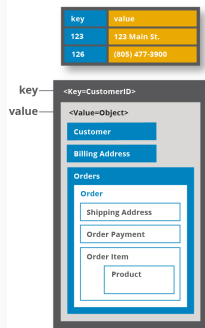
21

## Key-Value Store

Think *File system* or *LDAP repos.* more than database

### Products

Riak, Redis, Voldemort, Memcached, LevelDB



Key	Value
AB5D	→ 010001101101110101010100
AC4F	→ 0110111010001111100100010
2A45	→ 1101110011111010100001011
...	→ ...

20

## K-V Store (cont'd)

- Only primary index : lookup value  $v$  by key  $k$
- Simple operations :
  - `get(k)`
  - `put(k,v)`
  - `delete(k)`
- Value is **obfuscated**

### Ordered K-V Store

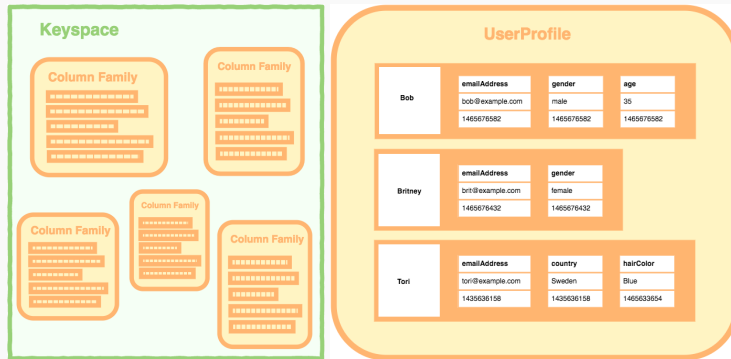
- Sorted keys b.t.w. of range partitioning
- short-scan range queries  $[[k, k + n]]$

22

## Column-Family Store

### Products

Cassandra, HBase, Hypertable



Source: [What is a Column Store Database?](#), Ian, Database Guide, 23 June 2016.

23

## Column-Family Store (cont'd)

- Column Family is close to relational table
- 1 CF = 1 file: expose the physical model to the user
- Group columns by query scope/logical units :
  - {name, address}, financial info, login info
- Random sharding by hash code from keys onto DHT

Popularized by Google BigTable

25

## Column-Family Store (cont'd)

CF Store encodes a 4-level hash map

[Keyspace][ColumnFamily][Key][Column]

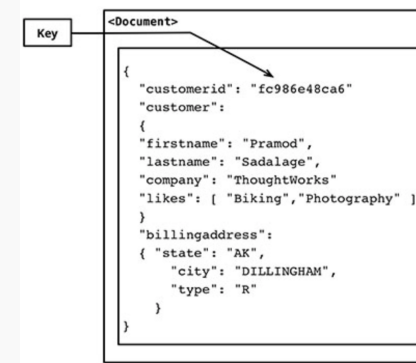
### Example

```
"ApplicationData": {  
  "UserInfo": {  
    "Alice": {  
      "age": 25,  
      "email": "alice@mit.org",  
      "state": "MA" } } }  
}
```

- Keys are shared among CF's
- Columns are sorted (not row keys)
- May have one more level of nesting (*Super Columns*)

24

## Document Store



26

## Document Store (con't)

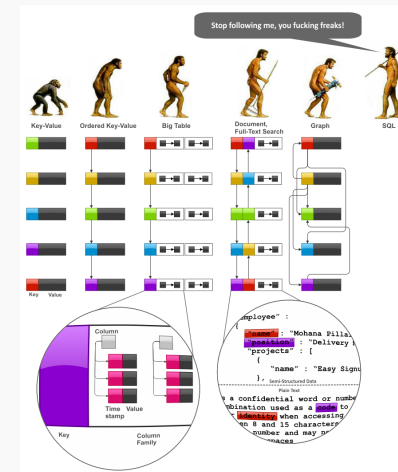
A document is a **pointerless object**

- e.g. JSON
- nested values + extensible records (schema-less)

In addition to K-V store : may have **secondary indexes**

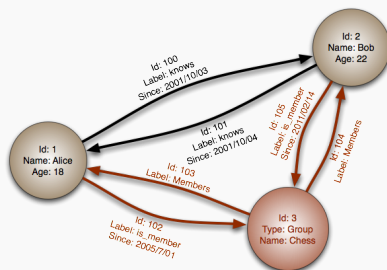
27

## SQL Freaks



29

## Graph Store



Originally uploaded by Ahzf (Transferred by Obersachse)

- Partitioning is not an option ! It is not a “pure NoSQL” data model
- An exception: TAO@Facebook, VLDB 2012

28

## Polyglot Persistence



## How to Choose the Right Database System ?

The sixty-four-thousand-dollar question !

150+ options on the **N★SQL** repos ...

- First attempt : Pros & Cons
- Second attempt : compare
- Too many offers, too many criteria !

What about your own requirements ?

30

## Data Model

- Complexity
  - determine the degree of structure : from FS to Graph
  - denormalize : no need to recompose entities in queries
    - embed one-to-many relationships
    - anyway, joins in apps for
      - (a) many-to-many relationships
      - (b) frequent updates (e.g., msgs of a user)
- Volume
- Schema flexibility
- Integrity constraints
- Data access patterns

32

## Data

Examine and segment the data

- Event
- Domain
- Critical
- Business
- Temporal
- Geo
- Meta
- Session
- Log
- Message
- ...

31

## Queries

Typical Queries look like ? SQL needed? LINQ needed ? BI/Analytic-Tools needed?  
MapReduce needed ? Ad-Hoc Queries needed? Background Data Analytics ?  
Secondary Indices? Range queries ? Complex Aggregations ? ColumnDB needed  
for Analytics ? Views needed ? ...

33

## Queries (cont'd)

Carefully profile the workload

**Example - Altoros Tech. Report, 2013** Source : Yahoo Cloud Serving Benchmark (YCSB) [Cooper et al., SoCC 2010]

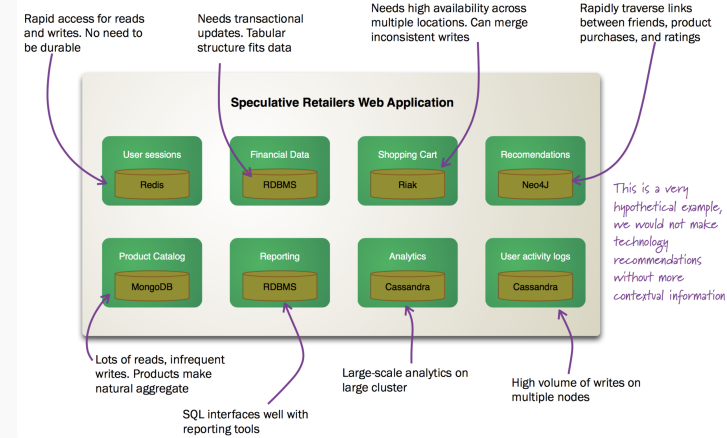
- update-heavily : e-commerce
- read-mostly : content tagging
- read-only : user profile cache
- 95/5 read/insert ratio : user status updates or inbox messages
- scan-short-ranges : threaded conversations
- 50/50 read-modify-write/read ratio : access to a user database
- 10/90 read/insert ratio : data migration

Similar approaches in CouchBase and Datastax (Cassandra) Whitepaper

34

## Polyglot Persistence

### "One Size Does Not Fit All"



36

## The Many Other Requirements

- Persistence design
  - on-disk, on-memory, SSTable, append-only, ...
- Consistency model
  - strong, weak, eventual, read-your-writes, ...
- Performance
  - latency, throughput, degree of concurrency
- Architecture
  - distributed, grid, cloud, mobile, p2p, replication, auto-scaling, load balancing, partitioning, ...
- Any non-functional requirement!
  - refactoring frequency, 24/7 system, dev. qualification, simplicity, security, licence model, community support, documentation, ...

35

NewSQL

## Back to the Future

### The Six SQL Urban Myths by M. Stonebraker

- Myth n°1 : SQL is too slow, so use a lower level interface
- Myth n°2 : I like a K-V interface, so SQL is a non-starter
- Myth n°3 : SQL systems don't scale
- Myth n°4 : There are no open source, scalable SQL engines
- Myth n°5 : ACID is too slow, so avoid using it
- Myth n°6 : In CAP, choose AP over CA

37

## App #1

Web application that needs to display lots of customer information; the user's data is rarely updated, and when it is, you know when it changes because updates go through the same interface.

Store this information persistently using a **K/V store**

39

## Auxiliary activities of an R-DBMS

### TPC-C on the Shore prototype

[S. Harizopoulos et al. SIGMOD 2008]

- logging 17% : everything written twice, log must be forced
- latching 19% : dbms is multithreaded (latch for the lock table)
- locking 17% : required for ACID semantics
- B-tree and buffer management operations 35%
- Useful work is 12% only!

### Recipe to Scalable R-DBMS = NewSQL

Give up with the 4 time-consuming activities yet keeping SQL and ACID Tx whenever it is necessary

38

## App #2

Department of Motor Vehicle (DMV) : lookup objects by multiple fields (driver's name, license number, birth date, etc) ; "eventual consistency" is ok, since updates are usually performed at a single location.

**Document store**

40

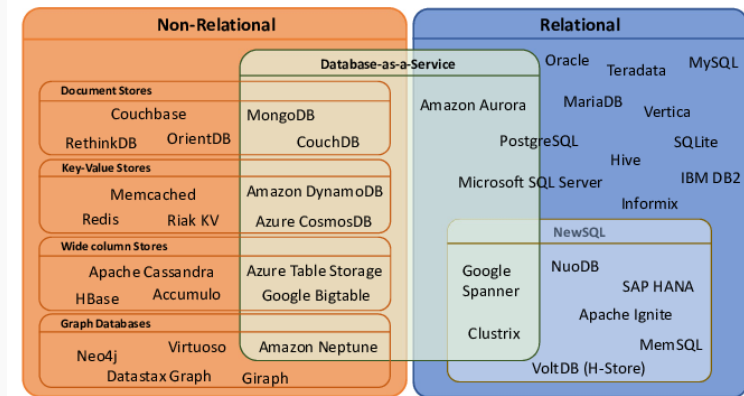
## App #3

eBay style application. Cluster customers by country ; separate the rarely changed 'core' customer information (address, email) from frequently-updated info (current bids).

Column-Family store

41

## Database Landscape



43

## App #4

Everything else (e.g. a serious DMV application)

Scalable R-DBMS

42