



# Databases

9

## Advanced Databases: NoSQL databases

### Database systems 2

Instructor: Sara Comai  
[sara.comai@polimi.it](mailto:sara.comai@polimi.it)

DB EVOLUTION

**1990-2000**

## '70→'90

- 1970-72: Relational model (E.F. Codd)
- 1974-77: Two major RDBMS prototypes
  - Ingres (UCB - Berkeley) – QUEL language → Ingres (later Postgres), MS SQL Server, Sybase
  - System R (IBM) – SEQUEL language → DB2, Oracle, Allbase (HP)
- 1976: Entity-Relationship Model (P. Chen)
- 1980s: SQL standard
- 1990s:
  - Object Oriented Databases (OODB)
  - Object-relational Databases (ORDB)
  - Object-Relational Mapping (ORM) – most popular
- Mid '90: WEB! Client-server DBMSs
  - Internet DB connectors (Front Page, Active Server Pages, JDBC, Java Servlet, Dream Weaver, ColdFusion, Enterprise Java Beans, and Oracle Developer 2000)
- 1998: XML

## '90 - New needs

- Complex objects
    - complex aggregations
    - non-numerical information — images, spatial data, temporal series, ...
    - pre-defined and user-defined types (with reuse)
  - Complex operations
    - customized on the type of information — e.g., multimedia or user-defined types
    - long-lived transactions
- Supported by OO programming languages
- BUT: Object-oriented application development and relational data management use two incompatible models
    - IMPEDANCE MISMATCH PROBLEM!

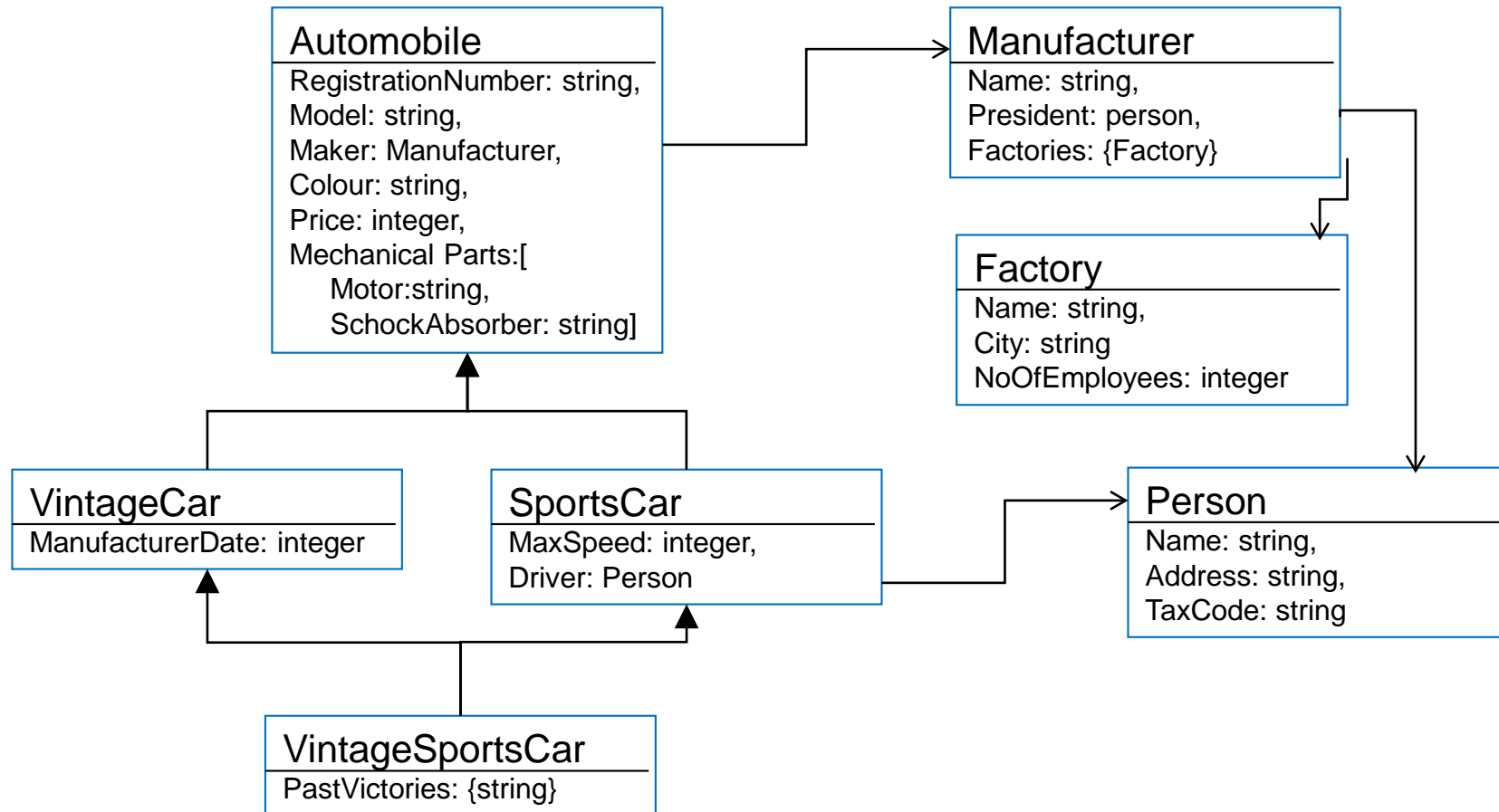
## ODBMS Technology

- First generation of ODBMS: it is represented by *persistent* object programming languages
  - offer only some of the services of DBMSs, with no support for queries
- Second generation of ODBMS: manage and store persistent objects → offer a greater number of DBMS services and typically offer support for queries
- Two main ODBMS solutions
  - OODBMS (*Object-Oriented*): a *revolutionary* solution, with respect to RDBMS
  - ORDBMS (*Object-Relational*): an *evolutionary* solution, with respect to RDBMS

## OODBMS

- Data represented as objects
  - object identity
  - attributes and methods
  - references, relationships, associations
- Extensible type hierarchy
  - user-defined types, abstract data types
  - single or multiple inheritance
  - overloading, overriding, late binding
- Declarative language for ad hoc purposes (ODL, OQL)
- Binding for object-oriented programming language

## OODBMS - Example



## OODBMS - Example of ODL schema

```
interface Automobile
{ extent Automobiles key RegistrationNumber}
{ attribute string RegistrationNumber;
  attribute string Model;
  attribute string Colour;
  attribute integer Price
  attribute structure MechanicalParts
  { string Motor,
    string ShockAbsorber};
  relationship <Manufacturer> Maker
    inverse Manufacturer::Builds;
  Automobile init (in string RegistrationNumber_par,
                  in string Model_par,
                  in string Colour_par,
                  in integer Price_par);
  void Increase (in integer Amount) raises(ExcessivePrice); }
```

Complex structure

Bidirectional relationships

Methods

inheritance

```
interface VintageCar: Automobile
{attribute integer ManufacturerDate}
```



## OODBMS - OQL by examples

- Find the registration numbers of the vintage cars built at Maranello and driven by Fangio:

```
select x.RegistrationNumber
from x in VintageSportsCar
where x.Driver.Name = "Fangio" and "Maranello" in x.Maker.Factories.Name
```

Path expressions

## OODBMS - OQL by examples

- Find the names of the manufacturers who sell sports cars at a price higher than 200000; for each of them, list the city and number of employees of the factories:


```
select distinct struct(Name: x.Maker.Name,  
                       Fact: (select struct (Cit: y.City,  
                                             Emp: y.NoOfEmployees)  
                             from y in Factory  
                             where y in x.Maker.Factories))  
from x in SportsCar  
where x.Price > 200000
```

Nested structure

Language-Integrated Query (LINQ), Microsoft .NET has a similar syntax

## Commercial OODBMS

- Objectivity/DB (C++/Java/Smalltalk/Python/SQL++ interfaces)
- ObjectStore PSE Pro for Java
- Versant (C++/Java/Smalltalk)
- O2 (C++/Java)
- db4o (Java/.NET)



```
List result = db.query(new Predicate() {  
    public boolean match(Pilot pilot) {  
        return pilot.getPoints() > 99 &&  
            pilot.getPoints() < 199 ||  
            pilot.getName().equals("Rubens  
Barrichello"); } });
```

```
Query query=db.query();  
query.constrain(Pilot.class);  
Query pointQuery=query.descend("points");  
query.descend("name").constrain("Rubens  
Barrichello") .or(pointQuery.constrain(new  
Integer(99)).greater()  
.and(pointQuery.constrain(new  
Integer(199)).smaller()));  
ObjectSet result=query.execute();  
listResult(result);
```

## ORDBMS

- Relational model extended
  - Nested relations
  - References
  - Sets
  - Row types, abstract types
  - Functions
- Declarative language extended
- Fundamental impedance mismatch remains

## Object-Relational databases (ORDBMSs)

- Revision of
  - Data model (non-1NF)
  - Query language
    - SQL-3 (SQL-99 is the official name)

## Tuple types

```
create row type PerType(  
    Name varchar(30) not null,  
    Residence varchar(30),  
    TaxCode char(16) primary key)
```

```
create table Person of type PerType  
create table Industrial of type PerType
```

```
create table Manufacturer of type ManufacturerType  
values for ManufID are system generated  
scope for President is Person
```

- Tuples correspond to the objects
- Relations correspond to the classes
- It is possible to use references and to include objects

## Queries

- Select the Name of the President of the Maker of Automobiles using the Motor "XV154"

```
select Maker -> President -> Name  
from Automobile  
where MechanicalParts -> Motor = 'XV154'
```

- Extract for each city the set of its Manufacturers

```
select City, set(name)  
from Manufacturer  
group by City
```

## Adoption of ODBMS in practice

- ODBMS originally thought of to replace RDBMS because of their better fit with object-oriented programming languages
- But:
  - high switching cost
  - introduction of ORDBMS
  - emergence of object-relational mappers (ORMs)have made RDBMS successfully defend their dominance in the data center for server-side persistence
- Object databases are now established as a complement
  - as embeddable persistence solutions in devices, on clients, in packaged software, in real-time control systems, and to power websites.



DB EVOLUTION

**2000-NOW**

# Big Data EveryWhere!



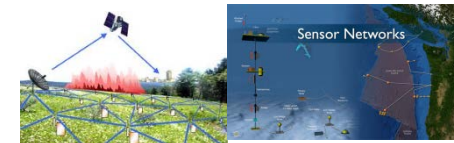
**Social media and networks**  
(all of us are generating data)



**Scientific instruments**  
(collecting all sorts of data)



**Mobile devices**  
(tracking all objects all the time)



**Sensor technology and networks**  
(measuring all kinds of data)

Needs for ability to manage, analyze, summarize, visualize, and discover knowledge from the collected data in a timely manner and in a scalable fashion

## Characteristics of Big Data: 1-Scale (Volume)

- **Data Volume**
  - 44x increase from 2009 to 2020
  - From 0.8 zettabytes to 35zb
- Data volume is increasing exponentially

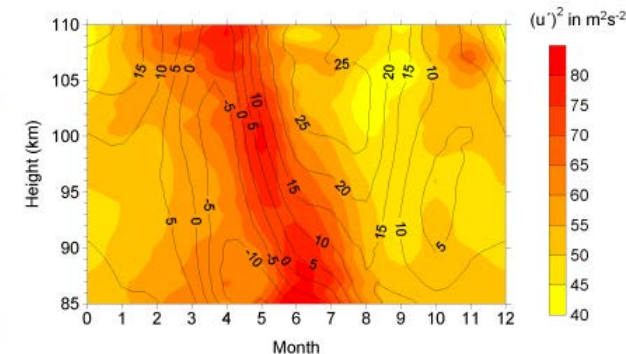
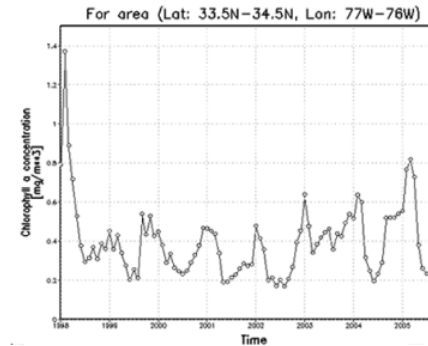
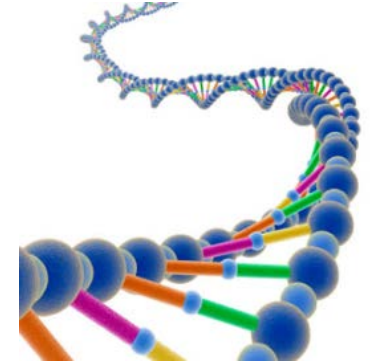
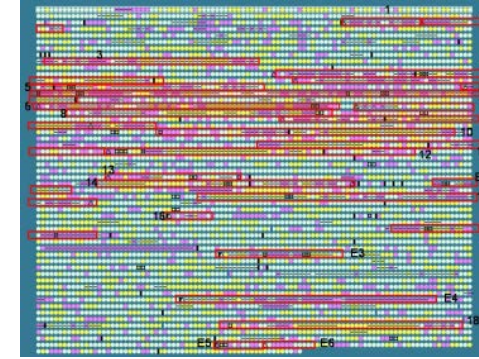
megabyte	MB	$10^6$
gigabyte	GB	$10^9$
terabyte	TB	$10^{12}$
petabyte	PB	$10^{15}$
exabyte	EB	$10^{18}$
zettabyte	ZB	$10^{21}$
yottabyte	YB	$10^{24}$



## Characteristics of Big Data: 2-Complexity (Variety)

- Various formats, types, and structures
- Text, numerical, images, audio, video, sequences, time series, social media data, multi-dim arrays, etc...
- Static data vs. streaming data
- A single application can be generating/collecting many types of data

To extract knowledge  
→ all these types of data need  
to be linked together



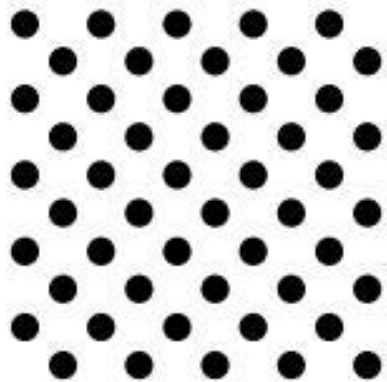
## Characteristics of Big Data: 3-Speed (Velocity)

- Data is begin generated fast and need to be processed fast
- Online Data Analytics
- Late decisions → missing opportunities
- **Examples**
  - **E-Promotions:** Based on your current location, your purchase history, what you like → send promotions right now for store next to you
  - **Healthcare monitoring:** sensors monitoring your activities and body → any abnormal measurements require immediate reaction



## Some Make it 4V's

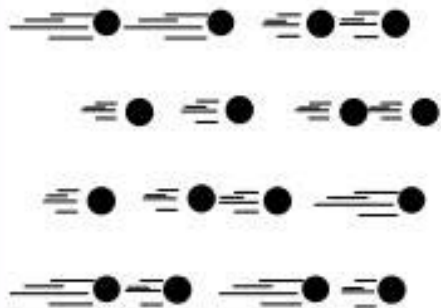
## Volume



## Data at Rest

Terabytes to  
exabytes of existing  
data to process

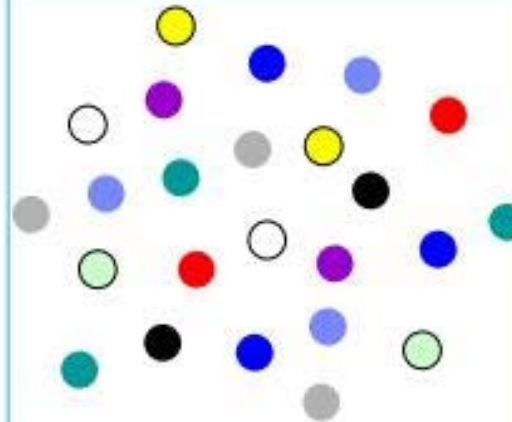
## Velocity



## Data in Motion

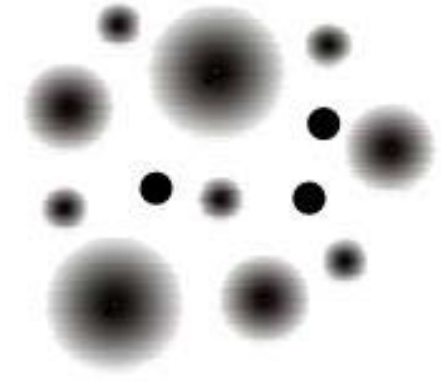
Streaming data,  
milliseconds to  
seconds to respond

## Variety

Data in Many  
Forms

Structured,  
unstructured, text,  
multimedia

## Veracity\*

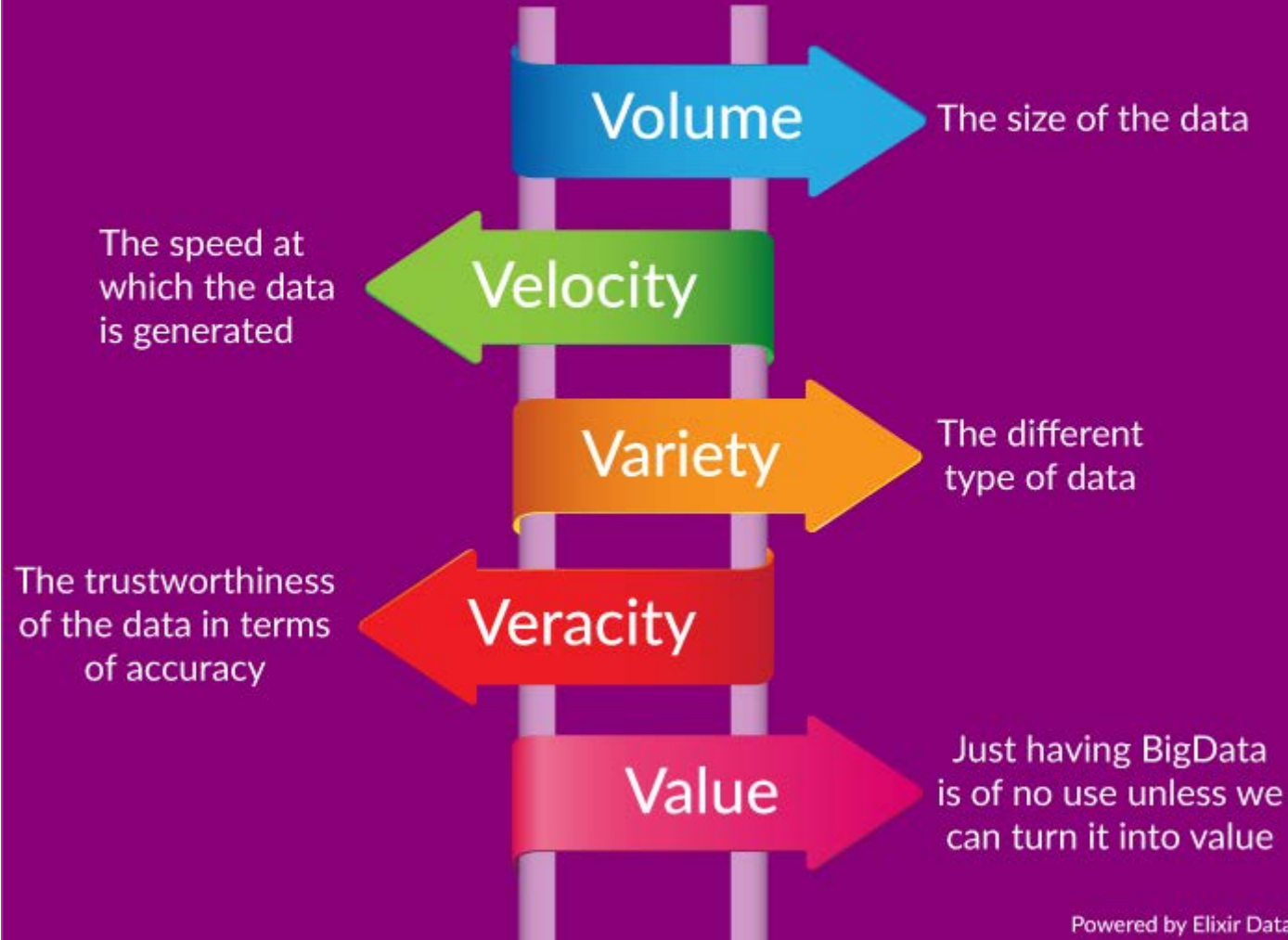


## Data in Doubt

Uncertainty due to  
data inconsistency  
& incompleteness,  
ambiguities, latency,  
deception, model  
approximations

Or 5V's

# The 5 Vs of BigData



## NoSQL

- “NoSQL” = “Not Only SQL”
  - Not every data management/analysis problem is best solved exclusively using a traditional DBMS
- Definition (from <http://nosql-database.org/> **N★SQL** )

Next Generation Databases mostly addressing some of the points: being **non-relational**, **distributed**, **open-source** and **horizontally scalable**.

The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: **schema-free**, **easy replication support**, **simple API**, **eventually consistent** / **BASE** (not ACID), a **huge amount of data** and more.



## NoSQL taxonomy

## Key-Value Stores



## Document Stores



## Graph/Triple Stores



## Column-Family Stores



## What do they miss compared to RDBMs?

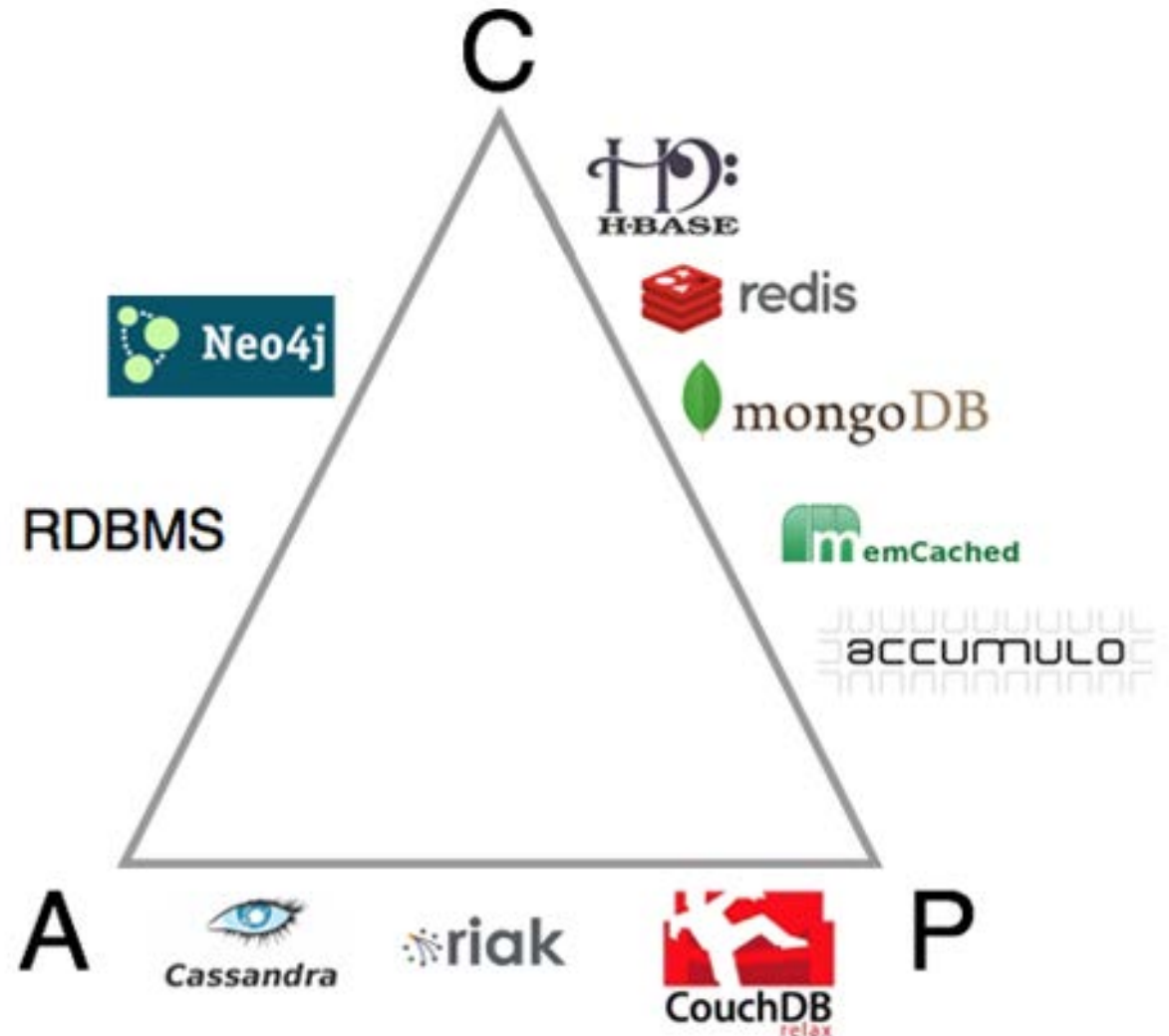
In general, they do not (fully) support:

- Joins
- Group by
- Order by
- ACID transactions

## Brewer's CAP Theorem

- Consistency: Every read receives the most recent write or an error
- Availability: Every request receives a (non-error) response – without the guarantee that it contains the most recent write
- Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

**Theorem:** for any system sharing data it is impossible to guarantee simultaneously all of these three properties



## Eventual Consistency

- BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) properties, as opposed to ACID
  - **S**oft state: copies of a data item may be inconsistent
  - **E**ventually Consistent – copies becomes consistent at some later time if there are no more updates to that data item
  - **B**asically **A**vailable – possibilities of faults but not a fault of the whole system

ACID	BASE
Strong consistency Isolation Focus on "commit" Nested transactions Availability? Conservative (pessimistic) Difficult evolution (e.g. schema)	Weak consistency – stale data OK Availability first Best effort Approximate answers OK Aggressive (optimistic) Simpler! Faster Easier evolution

## New RDBMs

- To increase **scalability**:

- Oracle RAC
- MySQL Cluster

Provide partitioning of load over multiple nodes

Other options: ScaleDB, VoltDB, NimbusDB, Clustrix, CockroachDB

→ also called **NewSQL**

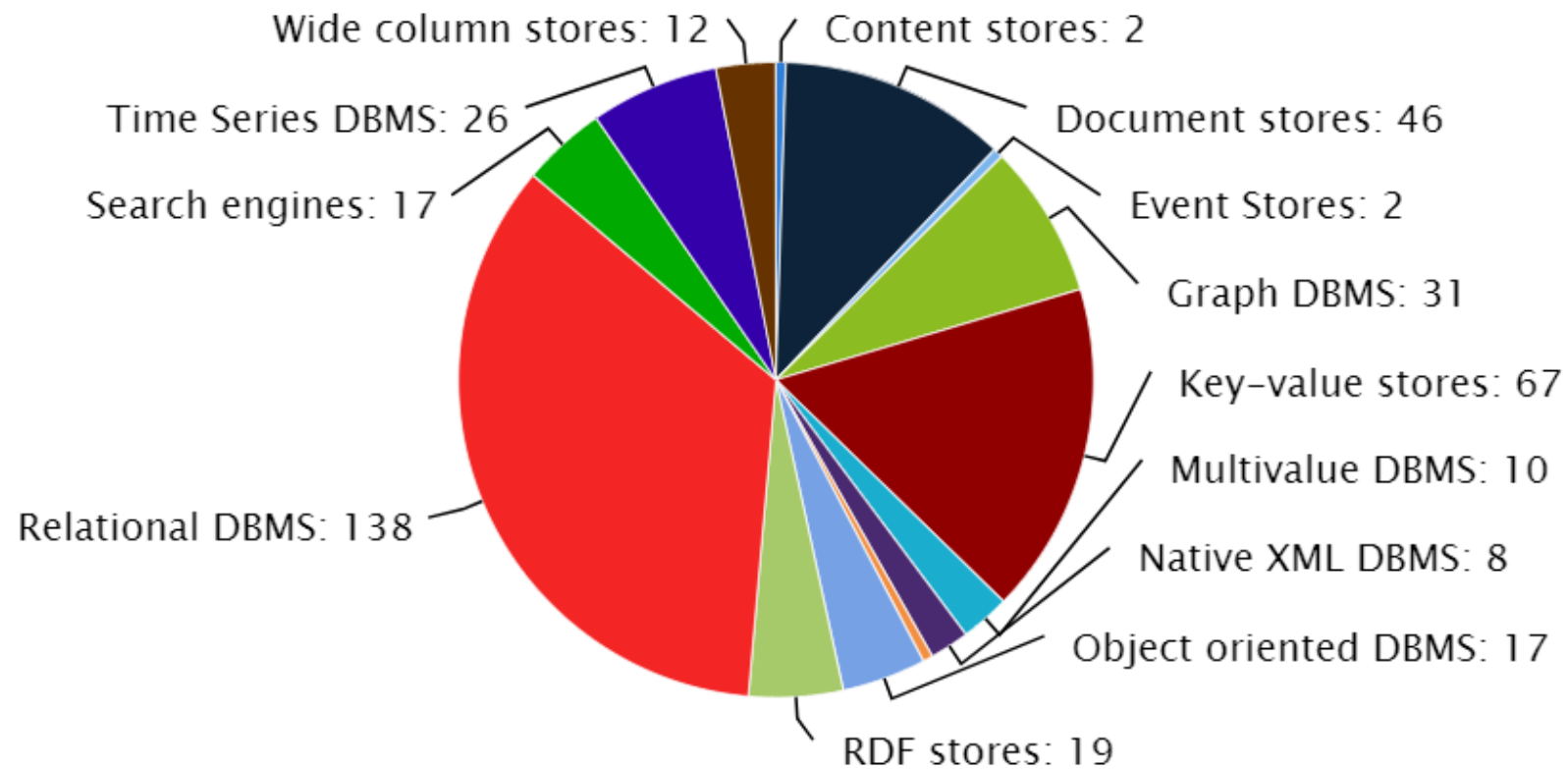
They support the relational data model and SQL, and guarantee ACID properties.

# Ranking from <http://db-engines.com/en/ranking>

343 systems in ranking, August 2018

Aug 2018	Rank		DBMS	Database Model	Score		
	Jul 2018	Aug 2017			Aug 2018	Jul 2018	Aug 2017
1.	1.	1.	Oracle +	Relational DBMS	1312.02	+34.24	-55.85
2.	2.	2.	MySQL +	Relational DBMS	1206.81	+10.74	-133.49
3.	3.	3.	Microsoft SQL Server +	Relational DBMS	1072.65	+19.24	-152.82
4.	4.	4.	PostgreSQL +	Relational DBMS	417.50	+11.69	+47.74
5.	5.	5.	MongoDB +	Document store	350.98	+0.65	+20.48
6.	6.	6.	DB2 +	Relational DBMS	181.84	-4.36	-15.62
7.	7.	↑ 9.	Redis +	Key-value store	138.58	-1.34	+16.68
8.	8.	↑ 10.	Elasticsearch +	Search engine	138.12	+1.90	+20.47
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	129.10	-3.48	+2.07
10.	10.	↓ 8.	Cassandra +	Wide column store	119.58	-1.48	-7.14
11.	11.	11.	SQLite +	Relational DBMS	113.73	-1.55	+2.88
12.	12.	12.	Teradata +	Relational DBMS	77.41	-0.82	-1.83

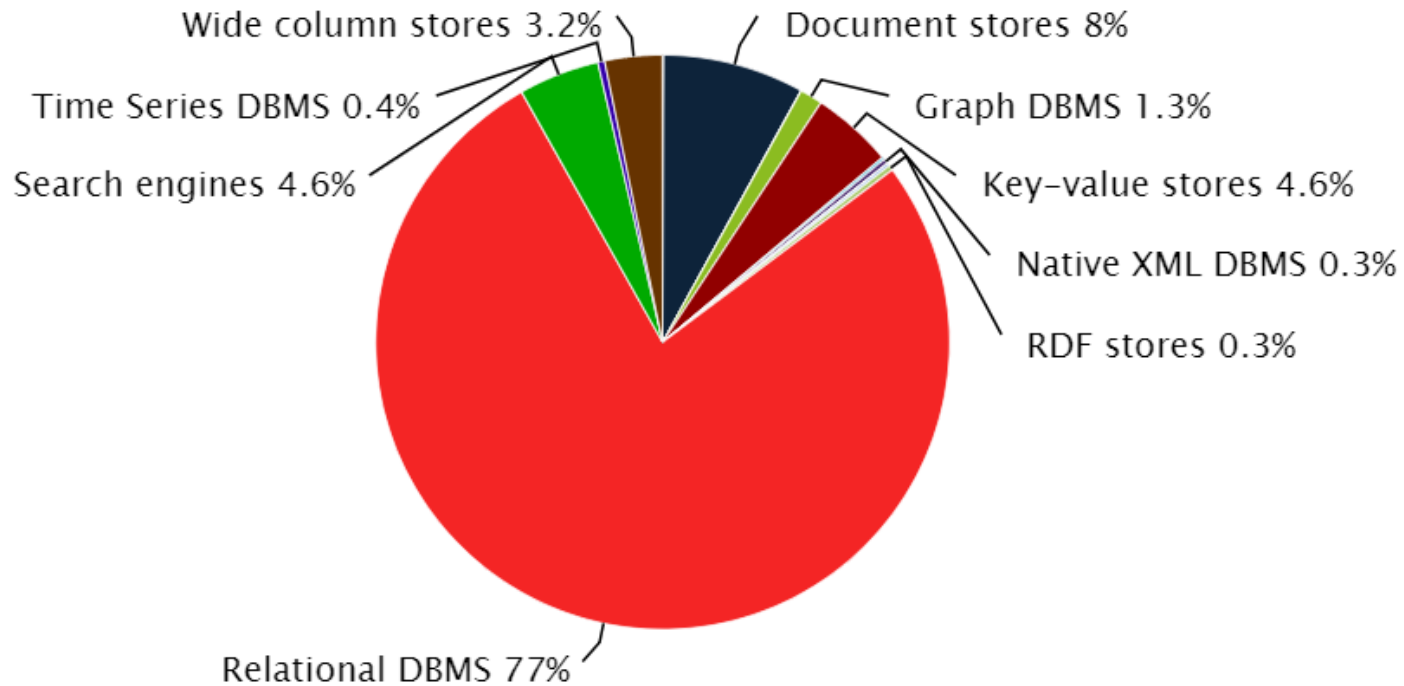
## Number of systems per category, August 2018



## Popularity of the models

<http://db-engines.com/en/ranking>

### Ranking scores per category in percent, August 2018

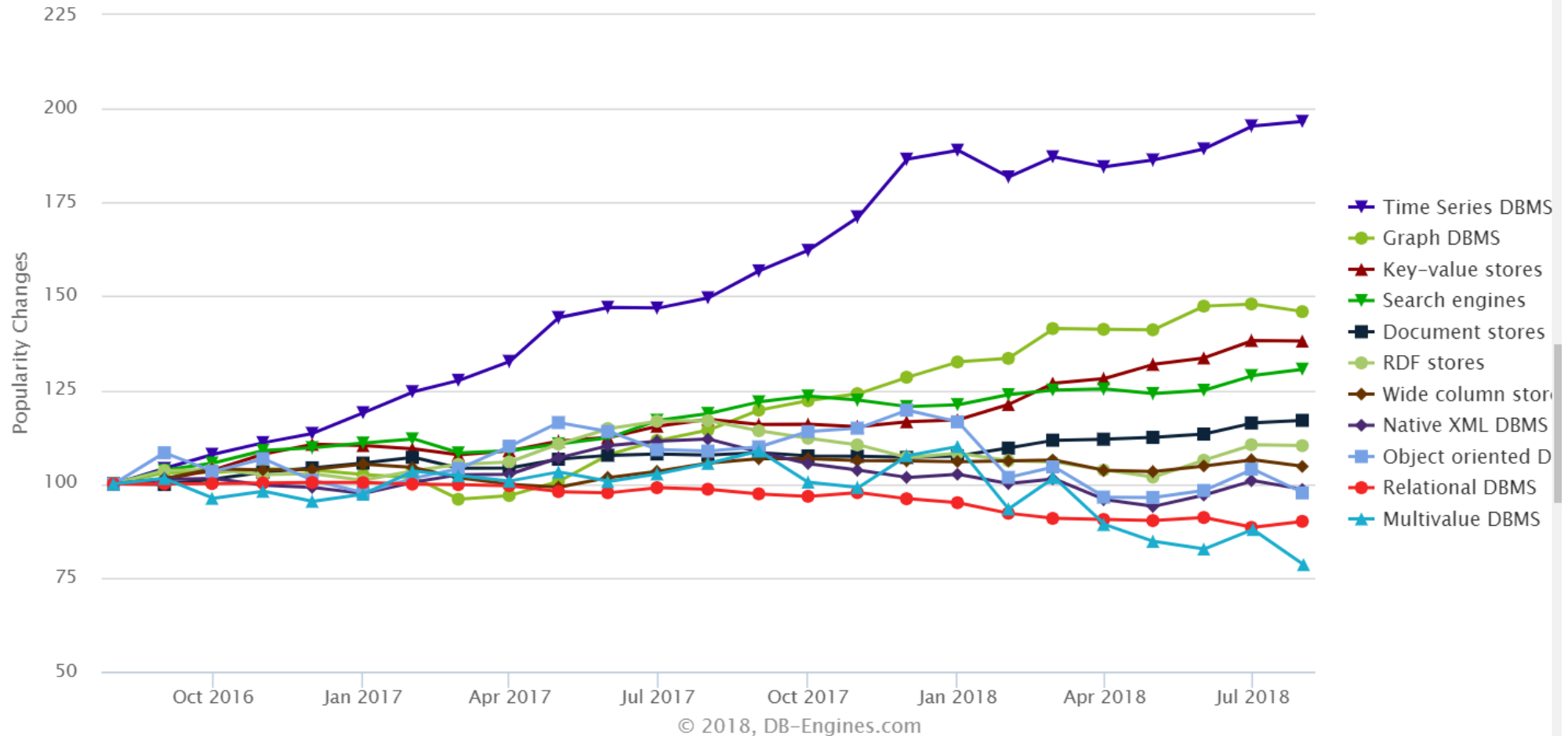




## Popularity of the models

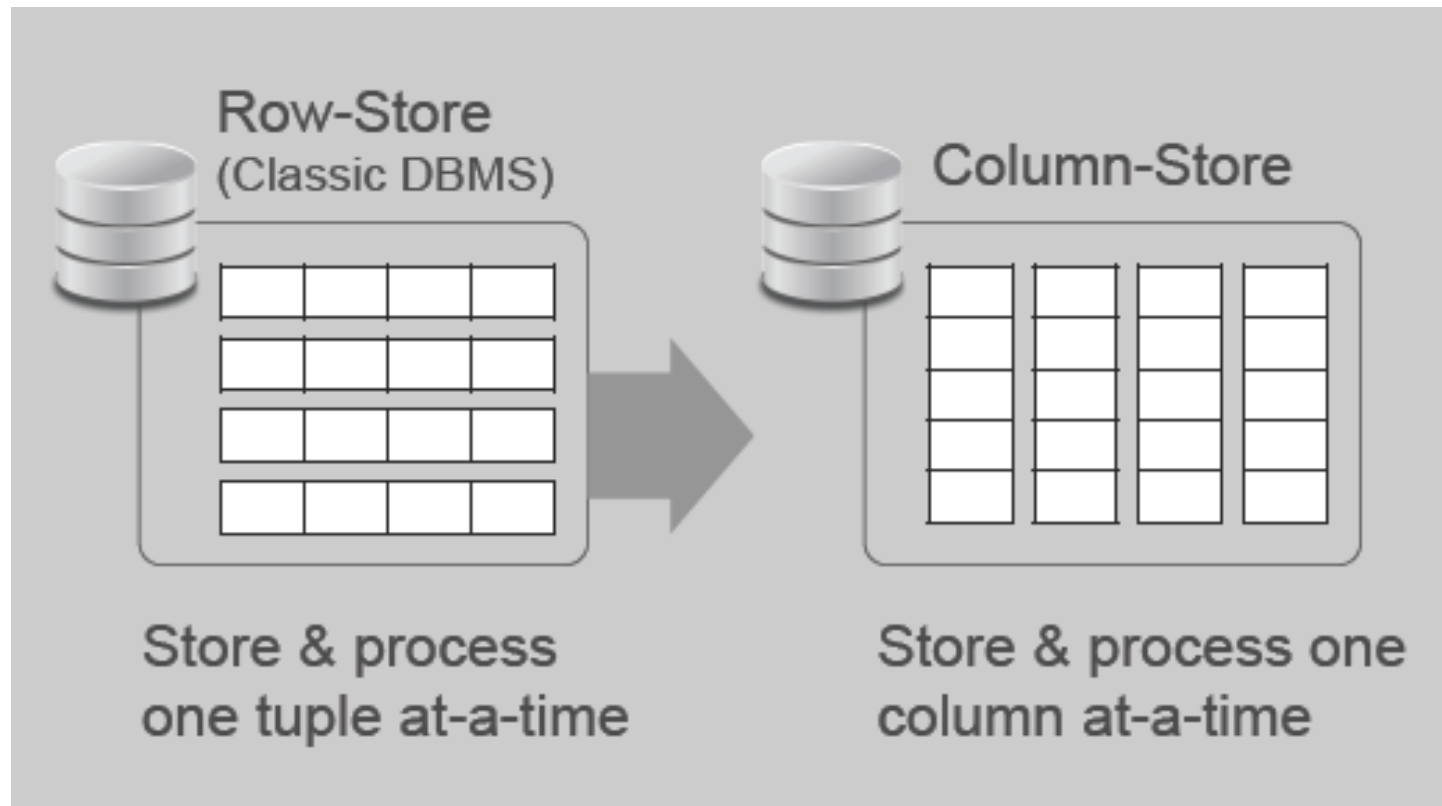
Trend of the last 24 months

<http://db-engines.com/en/ranking>



# COLUMN-STORES

## Column-store: an overview



Tuple Insertion:  Fast

 Requires multiple seeks

Queries:  May access more data than needed

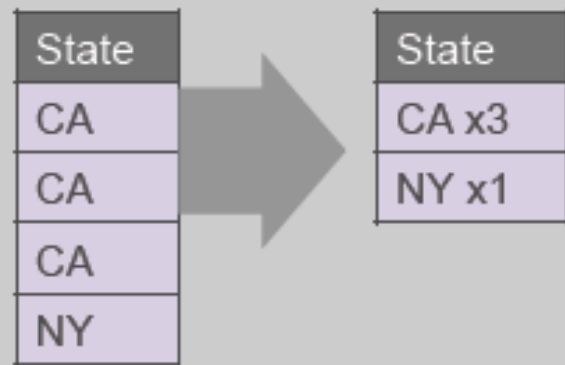
 Read only relevant data

Perfect for analytics!

## Compression

- Run-length encoding

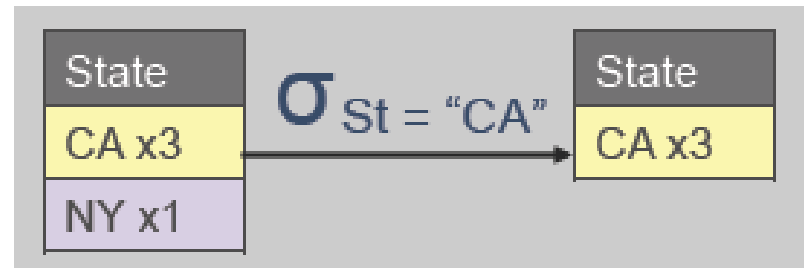
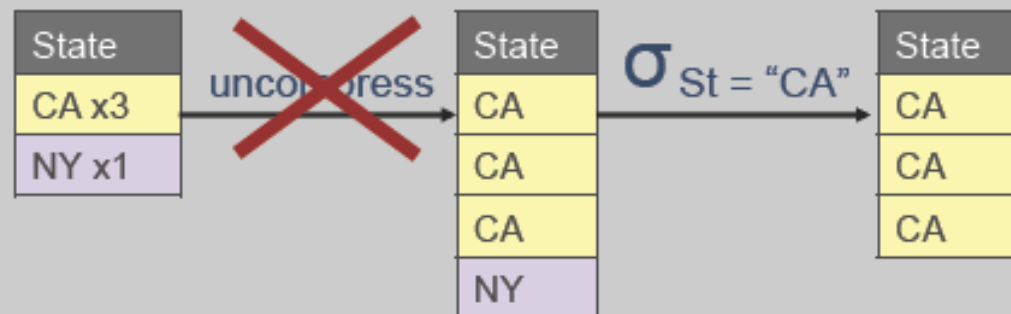
Replace list of identical values by pair (value, count)



⊕ Enables query processing on compressed data directly

⊕ Good for sorted columns

e.g., Select persons in CA



## Late tuple materialization

e.g., `SELECT Name FROM Person WHERE City="SD" AND State="CA"`

$\Pi_{\text{Name}}$

$\sigma_{\text{City} = \text{"SD"} \wedge \text{State} = \text{"CA"}}$

Create tuples

Name	Phone	City	State
Flintstone	858	SD	CA
Barney	619	SD	CA
Simpson	415	SF	CA
Bond	212	NY	NY

+ Reads relevant columns only

## Late tuple materialization

e.g., SELECT Name FROM Person WHERE City="SD" AND State="CA"

Extract

Name

$\sigma$

City = "SD"

$\sigma$

State = "CA"

1

1

0

0

1

1

1

0

Name

Flintstone

Barney

1

1

0

0

Position lists

Name	Phone	City	State
Flintstone	858	SD	CA
Barney	619	SD	CA
Simpson	415	SF	CA
Bond	212	NY	NY

# KEY-VALUE STORES

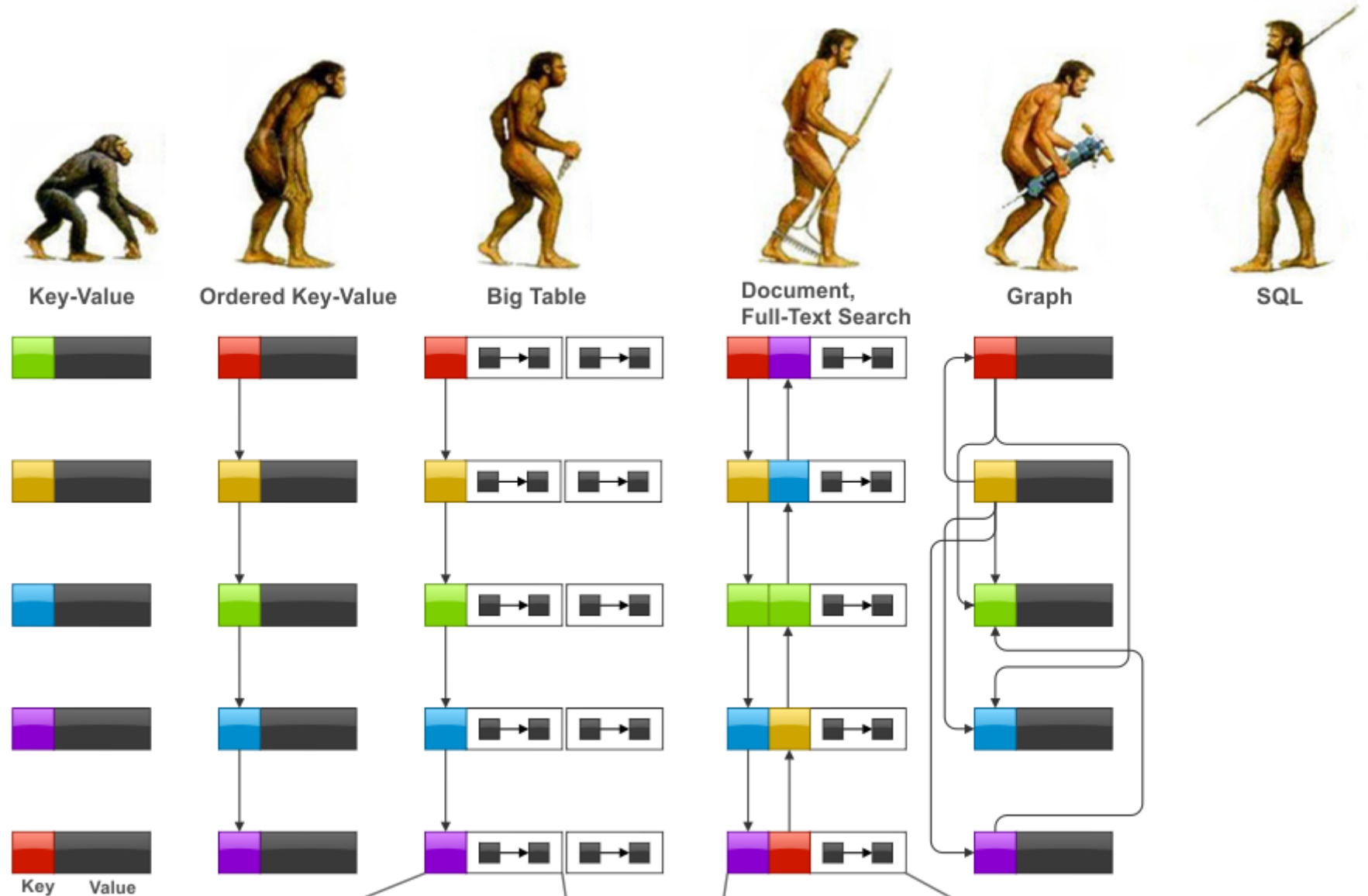
## Key-Value Stores

- Extremely simple interface
  - Data model: (key, value) pairs
  - Operations: Insert(key,value), Fetch(key), Update(key), Delete(key)



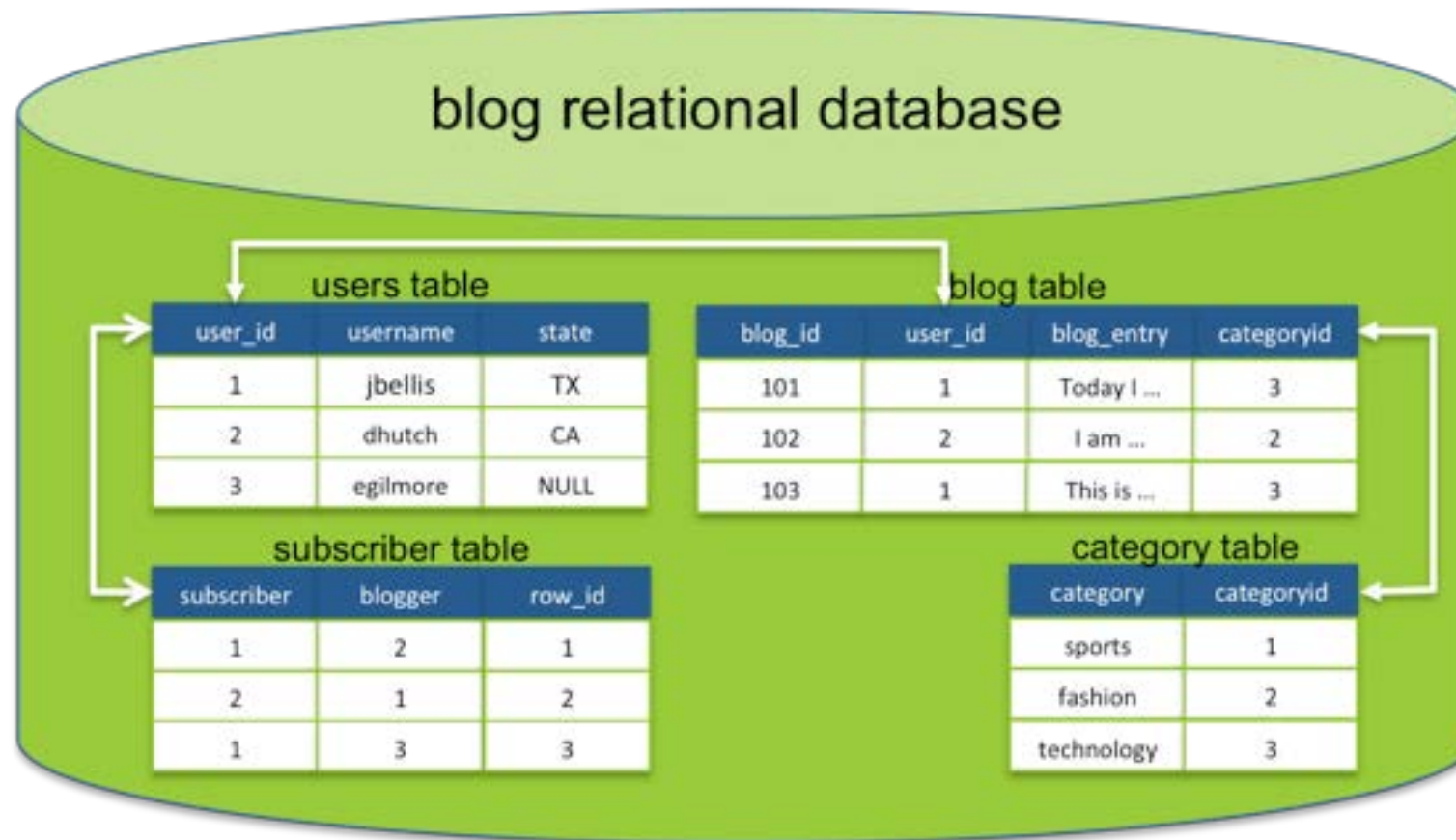
## Key-Value Stores

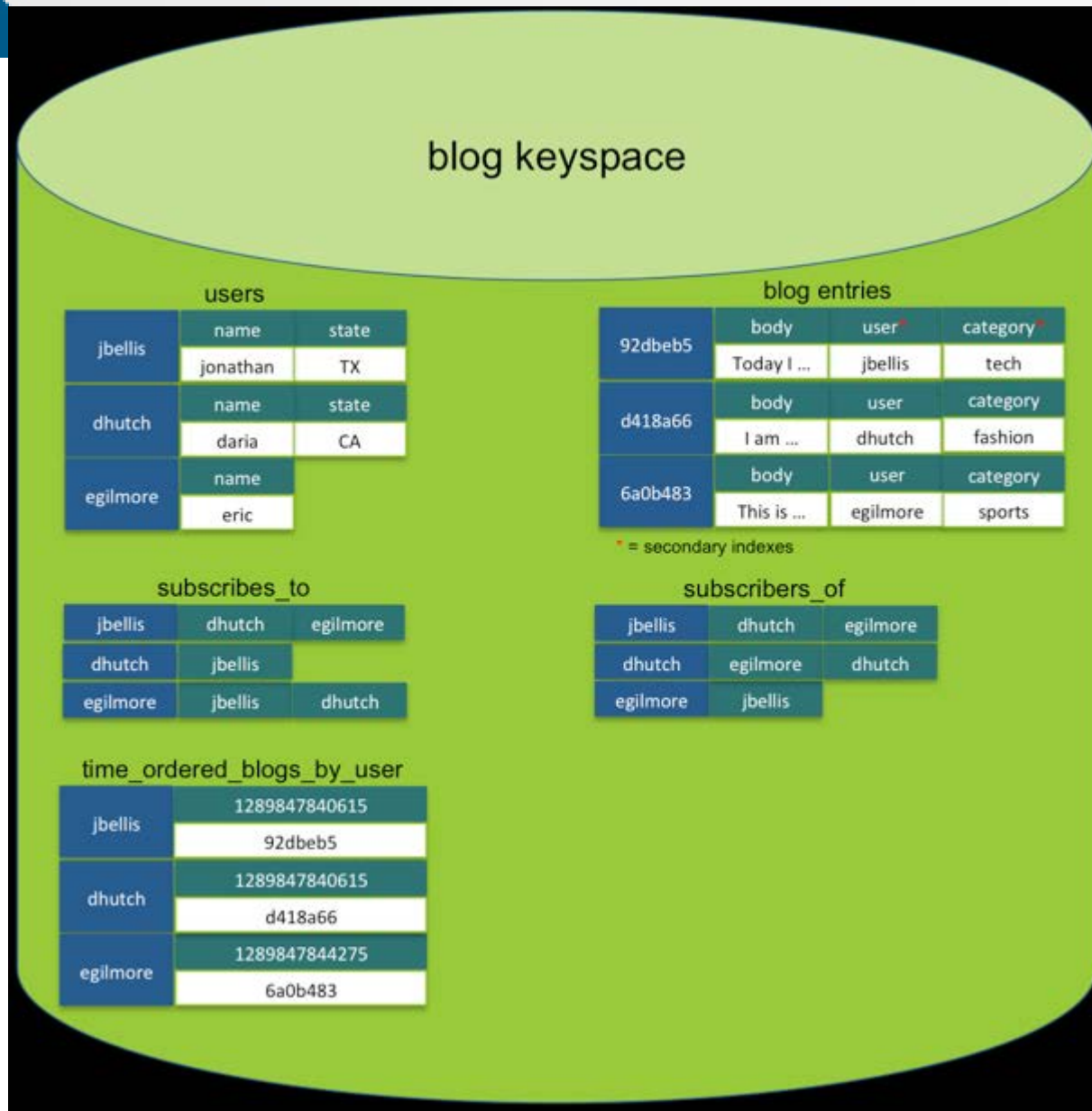
- Primary version of a DB!
- Like associative arrays
- Only for performance reasons



## Example (Cassandra data model)

- Consider the relational DB





- One or more *column family* objects, analogous to tables
- Identified by an application-supplied row *key*
- Each row in a column family is *not* required to have the same set of columns
- There are no formal foreign keys and joining column families at query time is not supported

## Cassandra Query Language

Examples:

- `INSERT INTO users (KEY, name, state)  
VALUES ('jbellis', 'jonathan', 'TX');`
- `Select * from users where KEY='jbellis';`

# DOCUMENT STORES

## Document Stores

- Like Key-Value Stores except that “value” is a document
  - Data model: (key, document) pairs
  - Document: JSON, XML, other semistructured formats
  - Basic operations: Insert(key,document), Fetch(key), Update(key), Delete(key)
  - Also Fetch based on document contents

## MongoDB example

*A document representing a wiki article may look like the following in JSON notation:*

```
{ title : " MongoDB " ,  
  last_editor : "172.5.123.91" ,  
  last_modified : new Date ("9/23/2010") ,  
  body : " MongoDB is a ..." ,  
  categories : [" Database " , " NoSQL " , " Document Database " ] ,  
  reviewed : false }
```

*Insertion into a MongoDB collection:*

```
db . < collection > . insert ( { title : " MongoDB " , last_editor : ... } ) ;
```

*Retrieval examples:*

```
db . < collection > . find ( { title : " MongoDB " } ) ;  
db . < collection > . find ( { categories : { $in : [" NoSQL " , " Document Databases " ] } } ) ;  
db . < collection > . find ( { categories : { $all : [" NoSQL " , " Document Databases " ] } } ) ;
```

# GRAPH DATABASES



## Graph Database Systems

- Data model: nodes and edges
- Nodes may have properties (including ID)
- Edges may have labels or roles
- Interfaces and query languages vary
- Example systems
  - Neo4j, FlockDB, Pregel, ...
- Aim to manage:
  - Multi-valued and/or complex attributes (Violations of the 1NF)
  - Dynamic data → accommodate changes
  - Data and query in a uniform way (to facilitate learning and reasoning)

## Database Representation

- Sailors(sid:integer, sname:char(10), rating: integer, age:real)
- Boats(bid:integer, bname:char(10), color:char(10))
- Reserve(sid:integer, bid:integer, day:date)

Sailors

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

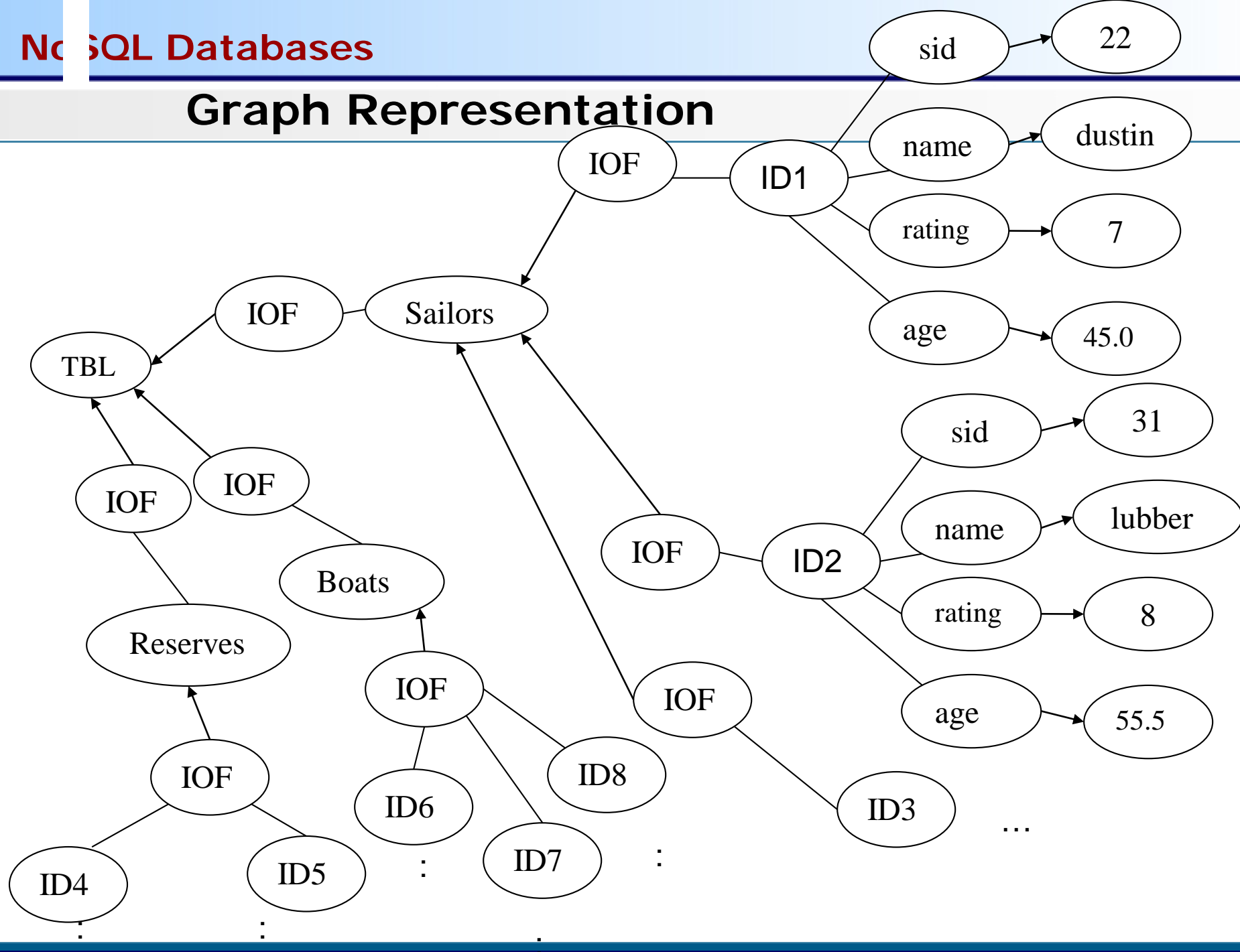
Reserves

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

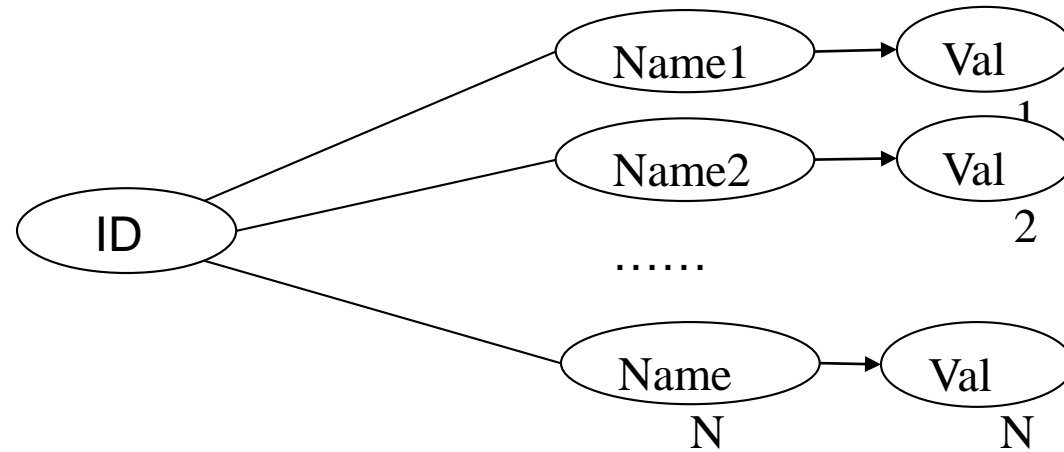
Boats

<u>bid</u>	bname	color
101	Interlake	red
102	Clipper	green
103	Marine	red

## Graph Representation



## Data Representation in the GDB DDL



- ID: (Name1=Val1,...,NameN=ValN)

Examples:

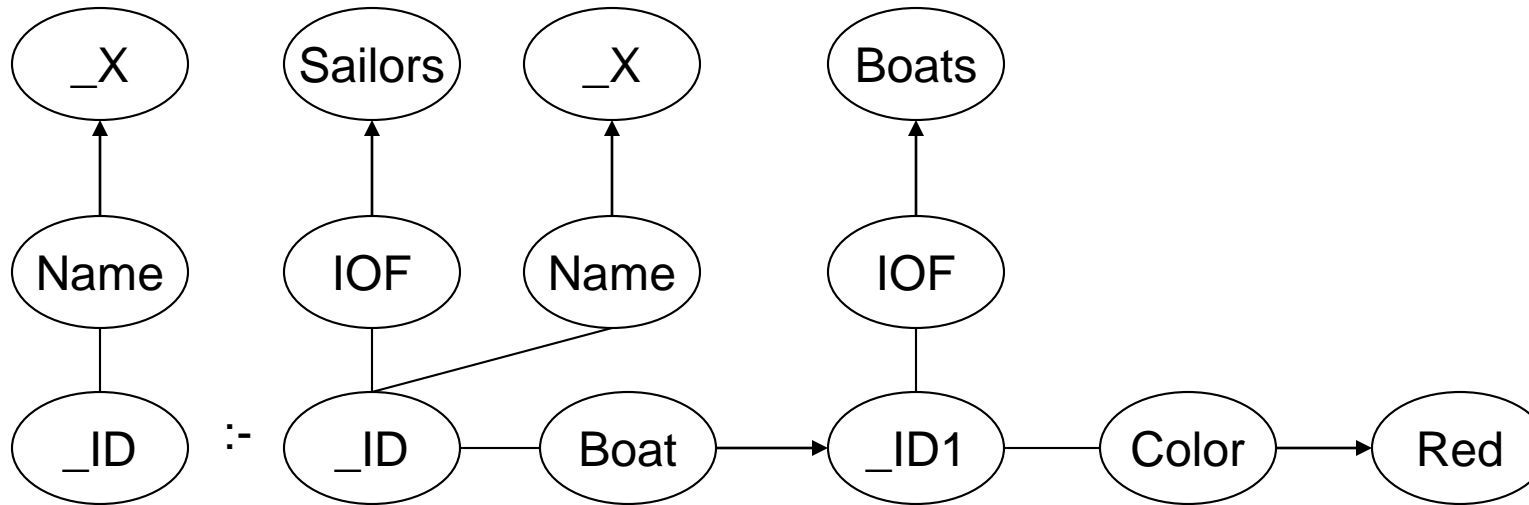
ID1: (sid=22, name="Dustin", rating=7, age=45.0)

ID4: (sailor=ID1, day="10/10/96", boat=ID6)

ID6: (bid=101, bname="Interlake", color="red")

## [DML-QL] Writing simple queries:

- The names of all sailors who have reserved a red boat



$\_ID:(Name = \_X) :-$

$\_ID:(IOF = Sailors, Boat = \_ID1, Name = \_X),$

$\_ID1:(IOF = Boats, Color = Red).$

The query is matched against the data graph

## Related courses

- TECHNOLOGIES FOR INFORMATION SYSTEMS (prof. Tanca)
- DATA MANAGEMENT FOR THE WEB (prof. Ceri)
- UNSTRUCTURED AND STREAMING DATA ENGINEERING (prof. Della Valle)

## Then, which solutions should we choose?

- RDBMs if performances can be achieved with a **single server** or possibly with indexed caching front-end (there are tools available with the different RDBMS solutions)
  - Thousands of transactions per second
  - ACID properties
- If you already use SQL and need some **scalability**: new RDBMs for scalability
  - Distributed architectures
  - High performances
- Key-value stores if you need to **partition** data across different RAMs in a LAN
- If you need to **scale to 100X** the single server → document or column stores
  - Both add flexibility
  - Handle volatile data

“One Size no longer fits all”

