

# Database Evolution

DB NoSQL

Linked Open Data

L. Vigliano

# NoSQL Database

## Requirements and features

- Large volumes of data.....increasing
- No regular data structure to manage
- Relatively homogeneous elements among them (no correlation between them)
- Simple types of operation

# NoSQL Database

## Needs and characteristics

- **Example : Twitter** (set of users who publish tweets)
- Few collections of interest(two entities : users and tweets), but massive
- Few operations (insert/update user, insert tweet)
- Data identified by a key, but only partially structured

# NoSQL Database

## Needs and characteristics

- →→ manage not strictly structured objects
- →→ manage data scalability.
- →→ offer only some of the features of traditional systems
- →→→ →→→ **NoSQL Systems**  
(Not only SQL)

# NoSQL Database

## Needs and characteristics

*“One size does not fit all”*

- Great scalability (many processors, horizontal data partitioning, distributed architecture at low cost)
- High availability, Replication and Eventual Consistency
- High Performance Data Access

# NoSQL Database

## Needs and characteristics

*“One size does not fit all”(2)*

- Replication
  - Master-Slave Replication
  - Master-Master Replication
- Scalability
  - Sharding Files
  - High performance to Data Access

# NoSQL Database

## Needs and characteristics

*“One size does not fit all”(3)*

- Relational model as a base, but it's not enough
- Not requiring a schema
- Adaptability to different application scenarios
- Languages for semistructured data : JSON, XML
- Less powerful Query languages (CRUD or SCRUD operations)

# NoSQL Database

## Transactional ? No, thanks

No ACID but **BASE** (Basically Available, Soft state, Eventually consistent)

- *CAP Theorem* : ‘In a distributed system is not possible to guarantee simultaneously: consistency, availability, partition tolerance’



# NoSQL Database Categories/Families

**each category is based on a specific data  
organization**

1. Key-value system
2. Document Store
3. Column-based store
4. Graph database
5. Other....

# NoSQL Database

## Key-value

- The data are key-value pairs defined by the program (databases without diagram).
- The design of objects is transparent to the system and chosen by the application that accesses them
- Eg. Oracle NoSQL, DynamoDB by Amazon (Voldemort).

# NoSQL Database

## Document Store

- Objects have a complex structure (documents) even if they are organized in collections. JSON format.
- Secondary indexes are not predefined and have no type
- Eg. MongoDB and CouchDB.

# NoSQL Database

## Column-based or Extensible record store

- Collections (tables) with no predefined structure, except for a first structure of 'families', or groups of columns.
- They can be nested.
- Eg. Big Table (Google), Hbase and HyperTable (Open Source).

# NoSQL Database

## Column-based and Key-value based

- NoSQL system that uses concepts from both key-value stores and column-based systems.
- Eg. Apache Cassandra by Facebook.

# NoSQL Database

## Graph Database

- Database that fit all the data that can be efficiently represented as graphs, even large.
- Eg. Neo4J or GraphBase for network topologies and traffic connections

# NoSQL Database

## Hybrid NoSQL Systems

- Combined concepts from many of the categories discuss above.
- Eg. OrientDB

# NoSQL Database

## other NoSQL Systems

- Based on object model or on native XML model.
- No high performance and replication.
- Eg. XML



# NoSQL Database

## Categories/Families

### Data organization - Summary

1. **Key-value Store**

value of the key - record, object, document or more complex structure

2. **Document Store**

document id - Json

3. **Column-based store**

Column families file - vertical partitioning

4. **Graph database**

Graphs - Path expression

5. **Other....**

# NoSQL Database

## MongoDB - goals

JSON documents gathered in collections

- High performance.
- High scalability.
- High reliability.
- Provide a simple set but full of features.

# NoSQL Database

## MongoDB - Data Model

Documents stored in collections (BSON format)

```
db.createCollection ("project", {capped:true, size:  
1310720, max:500})
```

```
db.createCollection ("worker", {capped:true, size:  
5242880, max:2000})
```

Only a field - **Object\_id**

Does not have a schema.....

# NoSQL Database

## MongoDB - Data Structure(1)

### Denormalized document

```
{_id: "P1",  
  Pname: "ProductX",  
  Plocation: "Bellaire",  
  Workers: [  
    { Ename: "John Smith",  
      Hours: 32.5  
    },  
    { Ename: "Joice English",  
      Hours: 20.0  
    }  
  ]  
};
```

# NoSQL Database

## MongoDB - Data Structure(2)

Embedded array of document references

```
{_id:          "P1",  
  Pname:      "ProductX",  
  Plocation:  "Bellaire",  
  WorkersId:  ["W1","W2"] }
```

```
{_id:          "W1",  
  Ename:      "John Smith",  
  Hours:      32.5}
```

```
{_id:          "W2",  
  Ename:      "Joice English",  
  Hours:      20.0}
```

# NoSQL Database

## MongoDB - Data Structure(3)

Normalized documents

```
{_id:          "P1",  
  Pname:      "ProductX",  
  Plocation:  "Bellaire",  
}
```

```
{_id:          "W1",  
  Ename:      "John Smith",  
  projectId:  "P1",  
  Hours:      32.5}
```

```
{_id:          "W2",  
  Ename:      "Joice English",  
  projectId:  "P1",  
  Hours:      20.0}
```

L. Vigliano

# NoSQL Database

## MongoDB - CRUD Operation

### Insert

`db.<Collection_name>.insert(<document(s)>)`

`Db.project.insert`

`( {_id:"P1",Pname:"ProductX",Plocation:"Bellaire" } )`

`Db.project.insert([`

`{_id:"W1",Ename:"John Smith",ProjectId:"P1",Hours:32.5 },`

`{_id:"W2",Ename:"Joice English",ProjectId:"P1",Hours:20 } ] )`

# NoSQL Database

## MongoDB - CRUD Operation

### Delete and update

```
db.<Collection_name>.remove(<condition>)
```

```
db.<Collection_name>.update(<condition>,<setclause>)
```



# NoSQL Database

## MongoDB - CRUD Operation

### Read

```
db.<Collection_name>.find(<condition>)
```

```
db.Project.find({}, {Ename:1,Hours:1});
```

# NoSQL Database

## MongoDB - more characteristics

- Lack of a schema definition.
- Lack of data typing.

# NoSQL Database

## SQL vs MongoDB - Query

SQL	MongoDB to insert a text
select a,b from Users;	dDb.users.find({}, {a:1,b:1});
select * from users where age=33;	db.users.find({age:33});
select * from users where age=33 order by name;	db.users.find({age:33}).sort({name:1});
create index myind on users(name);	db.users.ensureIndex({name:1});

# NoSQL Database

## MongoDB - distributed system characteristics

- Two-Phase Commit Protocol.
- Replication by Replica Set.
- Sharding (horizontal partitioning) and horizontal scaling(load balancing):
  - Range partitioning
  - Hash partitioning

# NoSQL Database

## BigTable - goals

- High scalability managing different servers and petabytes needed to store data.
- Performance control.
- Continuation and Fault Tolerance.
- Generating multi-dimensional sorted maps.

Distributed storage system, semi-structured data,  
based on Google File System.

# NoSQL Database

## BigTable - Data Format

- SSTable Format :
- Map persistent, orderly and unchanging association key-value, seen as arbitrary strings.
- Multi-dimensional keys
- Column : Column family and column qualifier

# NoSQL Database

## BigTable/Hbase - Data Model

- Namespace
- Table
- Column (Column family:Column  
qualifier)
- Row
- Data cell

# NoSQL Database

## BigTable - Data Model (2)

- Not relational, but based on the layout of each property of the DB.
- Multidimensional map, orderly, sparse, distributed and persistent, indexed by row key, column key and timestamp.
- Grouped rows dynamically.
- No predefined columns.
- Multiversioning data of each cell.



# NoSQL Database

## BigTable/Hbase - Data Model (3)

- **Table** is associated with **column families**.
- Column families associated with a table cannot be changed after the creation table

Creating a table :

Create 'EMP', 'Name', 'Address', 'Details'

# NoSQL Database

## BigTable/Hbase - Data Model (4)

- Each column family can be associated with many *not specified* **column qualifiers**
- A **column** is a combination  
ColumnFamily:ColumnQualifier

# NoSQL Database

## BigTable/Hbase - Data Model (5)

```
put 'EMP','row1','Name:Fname','John'  
put 'EMP','row1','Name:Lname','Smith'  
put 'EMP','row1','Name:Nickname','Johnny'  
put 'EMP','row1','Details:Job','Engineer'  
put 'EMP','row1','Details:Review','Good'  
put 'EMP','row2','Name:Fname','Alicia'  
put 'EMP','row2','Name:Lname','Zelaya'  
put 'EMP','row2','Name:Mname','Jennifer'  
put 'EMP','row2','Details:Job','DBA'  
put 'EMP','row2','Details:Supervisor','James Borg'  
put 'EMP','row3','Name:Fname','James'  
put 'EMP','row3','Name:Minit','E'  
put 'EMP','row3','Name:Lname','Borg'  
put 'EMP','row3','Name:Suffix','Jr.'  
put 'EMP','row3','Details:Salary','1,000,000'
```

L. Vigliano

# NoSQL Database

## **BigTable/Hbase** - CRUD Operation

### **Low level operations**

Create <tablename>,<column family>, <column family>,...

Put <tablename>,<rowid>,<column family>:<column  
qualifier>,<value>

Scan <tablename>

Get <tablename>,<rowid>

# Linked Open Data

- Tim Berners Lee

