Types of NoSQL Databases

Tuples and Document Stores

Tuple store

- A **tuple store** is similar to a key-value store, with the difference that it does not store pairwise combinations of a key and a value, but instead stores a unique key together with a vector of data
- Example:
  - marc → ("Marc", "McLast Name", 25, "Germany")
- No requirement to have the same length or semantic ordering (schema-less!)

Tuple store

- Various NoSQL implementations do, however, permit organizing entries in semantical groups, (aka collections or tables)
- Examples (collections Person and Painting):
  - Person:marc -> ("Marc", "McLast Name", 25, "Germany")
  - Person:harry -> ("Harry", "Smith", 29, "Belgium")
  - Painting:lamgods → (“Lam Gods”,”Van Eyck”,”Gent”)

Document store

- Document stores store a collection of attributes that are labeled and unordered, representing items that are semi-structured
- Example:

```
{
   Title      = "Harry Potter"
   ISBN        = "111-1111111111"
   Authors    = [ "J.K.  Rowling" ]
   Price      = 32
   Dimensions = "8.5 x 11.0 x 0.5"
   PageCount  = 234
   Genre      = "Fantasy"
}
```

Document store

- Most modern NoSQL databases choose to represent documents using JSON

```
{
 "title": "Harry Potter",
 "authors": ["J.K.  Rowling", "R.J.  Kowling"],
 "price": 32.00,
 "genres": ["fantasy"],
 "dimensions": {
         "width": 8.5,
         "height": 11.0,
         "depth": 0.5
 },
 "pages": 234,
 "in_publication": true,
 "subtitle": null
}
```

Document store: design

- The address document is embedded in the visitors document.
- The visitor document is referred to in the opinion documents

```
{
  _id: "V1",
  name: "Eva",
  address: {
          street: "Funstraat 3",
          city: "Aalst",
          zipcode: 9300,
          country: "Belgium"
          },
  language: "dutch"
}


{
  _id: "V2",
  name: "Adam",
  language: "dutch"
}
```

```
{
  _id: 234567891,
  day: "15/1/2017",
  hour: "14:00",
  visitor_id: "V1",
  place: "Room 1"
}

{
  _id: 234567892,
  day: "15/1/2017",
  hour: "14:01",
  visitor_id: "V1",
  place: "Room 1",
  comments: ["not nice", "crowded"],
  score: 2
}

{
  _id: 234567893,
  day: "15/1/2017",
  hour: "14:02",
  visitor_id: "V2",
  place: "Room 1",
  comments: ["Rembrandt is amazing"]
}
```

Document store: basic operations

- Example
  - db.opinions.insert({_id: 234567894, day: "15/1/2017", hour: "14:02", visitor_id: "B1", place: "Room 1"})
  - db.opinions.find()
  - db.opinions.find({place: "Room 1"})
  - db.opinions.find({score: {$lt: 5}})
  - db.opinions.createIndex({place: 1})
  - db.visitors.update({"_id": "V1"}, {$set: {"address.street": "Funstraat 5"}})
  - db.visitors.remove({"_id": "V1"})

- In the next chapter we see operations in detail

Items with Keys

- Most NoSQL document stores will allow you to store items in tables (collections) in a schema-less manner, but will enforce that a primary key be specified
    - E.g. Amazon’s DynamoDB, MongoDB (_id)
- Primary keys will be used as a partitioning key to create a hash and determine where the data will be stored (cf. previous chapter)

Filters and Queries

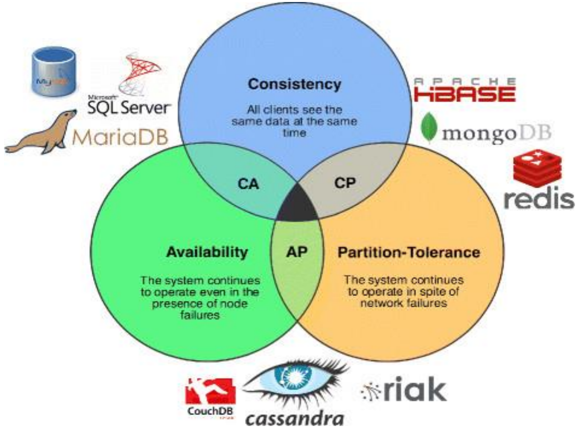
- Document stores deal with semi-structured items. They impose no particular schema on the structure of items stored in a particular collection, but assume that items nevertheless exhibit an implicit structure following from their representational format, representing a collection of attributes, using JSON, XML, etc.
- Just as with key-value stores, the primary key of each item can be used to rapidly retrieve a particular item from a collection, but since items are composed of multiple attributes, most document stores can retrieve items based on simple filters as well.

Filters and Queries

- The big difference between a key – value database and a document database is that you can query into the document structure and you can usually retrieve portions of the document or update portions of a document
- Document databases have been adopted more widely than any other type of NoSQL database
- Some popular document databases
  - MongoDB
  - CouchDB
  - …

MongoDB

- One of the most well-known and widely used implementations of a document store.
- MongoDB is strongly consistent by default: if you write data and read it back out, you will always be able to read the result of the write you just performed (if the write succeeded).
- This is because MongoDB is a so-called "single-master" system where all reads go to a primary node by default.
- If you do optionally enable reading from the secondary nodes, then MongoDB becomes eventually consistent where it's possible to read out-of-date results.


Consistency
All clients see the same data at the same time
Availability
The system continues to operate even in the presence of node failures
Partition-Tolerance
The system continues to operate in spite of network failures
CA
CP
AP
SQL Server
MariaDB
APACHE HBASE
mongoDB
redis
CouchDB
cassandra
riak

SQL After All

- Filtering and query operations are quite a challenge in MongoDB.
- We will also see how some operations can help perform complex queries and aggregations in document stores, even though these document stores do not support relational structures directly.
- It will become apparent that many traditional GROUP BY-style SQL queries are convertible to an equivalent MongoDB operations.
- That is the reason many document store implementations express queries using an SQL interface (most often using a subset of the SQL language), offering users a more familiar way of working rather than requiring them to think in map-reduce logic.

SQL After All

- Couchbase also allows to define foreign keys and perform join operations

```
SELECT books.title, books.genres, authors.name
FROM books
JOIN authors ON KEYS books.authorId
```

SQL After All

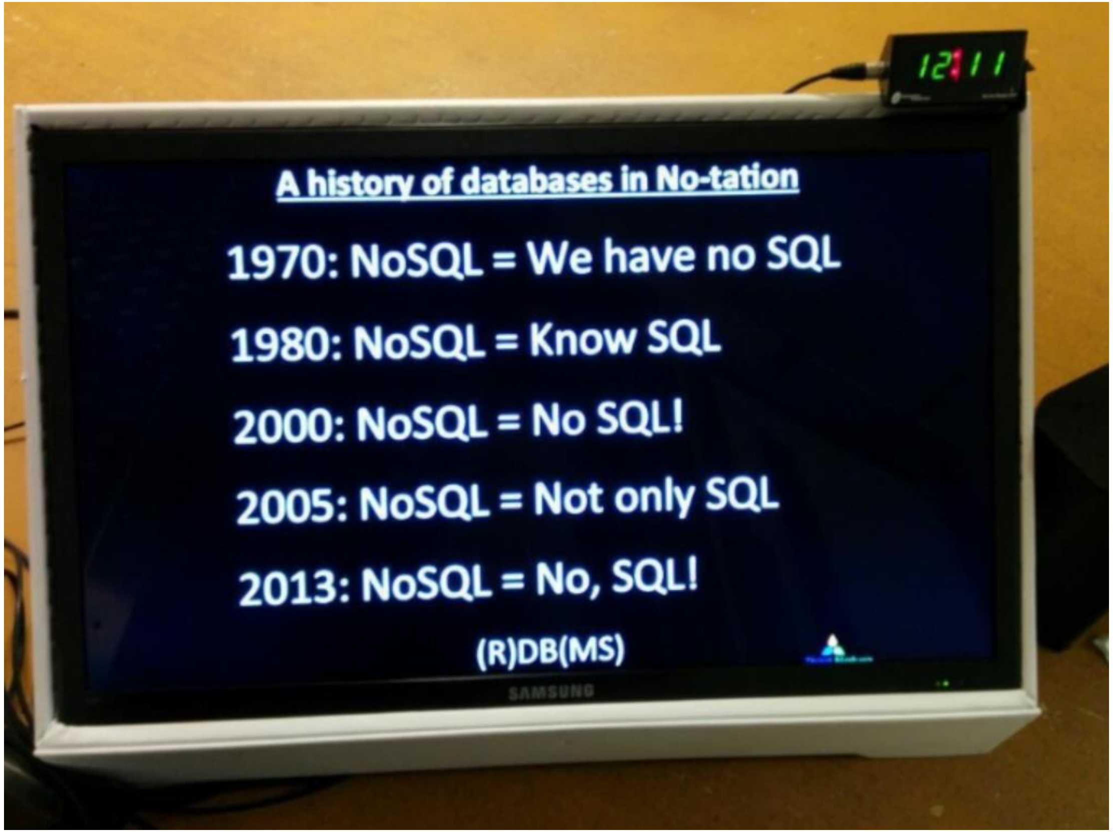
- Many RDBMS vendors start implementing NoSQL by
  - Focusing on horizontal scalability and distributed querying
  - Dropping schema requirements
  - Support for nested data types or allowing to store JSON directly in tables
  - Support for GROUP BY like operations
  - Support for special data types, such as geospatial data

SQL After All

- Example: recent versions of the open-source PostgreSQL database allow you to execute the following statements:

```
CREATE TABLE books (data JSONB);
INSERT INTO books (data) VALUES
('
    {
    "title": "Beginners Guide to Everything",
    "genres": ["educational", "fantasy"],
    "price": 200,
    }
')
SELECT DISTINCT data->>'title' AS titles FROM books;
```

SQL After All



A history of databases in No-tation

1970: NoSQL = We have no SQL

1980: NoSQL = Know SQL

2000: NoSQL = No SQL!

2005: NoSQL = Not only SQL

2013: NoSQL = No, SQL!

(R)DB(MS)