

NoSQL Databases

Course 12

Table of contents

- NoSQL Databases
- NoSQL Data Models
 - Key-value
 - Document
 - Wide-column store (column-family)
 - Graph
- NoSQL Implementations: MongoDB

NoSQL Databases

- The **relational model** has the following **drawbacks**:
 - It is **not designed** to offer **horizontal scalability**
 - **Impedance mismatch** or the difference between the **in-memory structures** used by programmers in their applications and **the relational model**
 - **Rigid** schema
- **NoSQL (Not only SQL) databases** are designed to **solve** the drawbacks listed above
- While there is a huge diversity of NoSQL databases, they have some things in **common**:
 - **Horizontal scalability** (NoSQL databases are designed to run in a distributed environment)
 - **Flexible schema**
 - **Open source** (generally, NoSQL databases are *open source*)

NoSQL Data Models

- NoSQL databases are relatively new, they are very different from one another and they evolve very fast (there are significant differences between versions)
- NoSQL implementations are specialized in solving specific problems and for this reason, you have to pay attention when you make your choice

NoSQL Data Models

- There are **four** main NoSQL **data models**:
 - The **key-value** data model (collections of key-value pairs)
 - The **wide-column store/column-family** data model
 - The **graph** data model (graph)
 - The **document** data model (collections of documents)

Key-value data model

- It is the most basic NoSQL data model
- Data is stored as key-value pairs
- The **key part** of the key-value pair is also a *unique identifier* and it is used for data access
- The **value part** of the key-value pair is usually opaque at the database level
- The value part can contain complex structures, but these structures are not visible at the database level
- The following operations are available: insert a new value for a key, return the value for a given key and delete a key-value pair (these operations are fast)
- It allows horizontal scalability and high availability
- Key-value implementations: Riak, Redis

Wide-column store data model (column-family)

- It is based on the multi-dimensional map data structure
- In this context, a column is in fact a key-value pair in which the key is the name of the column and the value is the actual value for the column
- The value part can store complex structures as well (like arrays or objects)
- A column-family is a group of columns that contain related data (similar to a relational table)
- A column family resembles a relational table, but it has a flexible schema and does not store NULL values if there is no value assigned to a specific column
- Wide-column store (column-family) implementations: Apache Cassandra, Apache HBase, Bigtable

Graph data model

- It is based on the graph structure
- Data is stored as nodes and edges between nodes, with each node or edge having properties attached to it
- Nodes represent objects or entities, while edges represent relationships between objects or entities
- In contrast to other NoSQL data models, the horizontal scalability is problematic in this case
- It is best suited to store data that is highly interconnected
- It also offers great flexibility in interpreting relationships between entities
- Graph implementations: Neo4J, InfiniteGraph, GraphDB

Document data model

- Its base structure is the document
- Each document has a unique identifier (which corresponds to the key part of a key-value pair) and a value (which corresponds to the value part of a key-value pair)
- The most popular format is JSON (JavaScript Object Notation)
- A JSON document is a collection of key-value pairs, and the value part can be a complex structure (like a collection of values or a collection of key-value pairs)
- Other document formats are XML and YAML
- It allows very expressive query languages
- The structure of the value part is visible at the database level
- Document implementations: MongoDB, CouchDB, Couchbase

NoSQL Implementations: MongoDB

- MongoDB is a distributed, open source NoSQL implementation based on the document data model
- MongoDB offers horizontal scalability (automatic sharding) and high availability (data replication)
- Data is stored in a format similar to JSON, called BSON (Binary JSON)
- BSON extends JSON and offers ordered fields and additional data types
- JSON is a data-interchange text format that is human and machine-readable
- There are two structures on which JSON is built: the object (a collection of key-value pairs) and the array (a list of values)

NoSQL Implementations: MongoDB

- JSON document example:

```
{
  "firstname" : "Rose",
  "lastname" : "Jackson",
  "email" : ["rosej@gmail.com" , "rosejackson@yahoo.com"],
  "address" :
    {
      "street" : "Alameda Street",
      "number" : 1,
      "city" : "Los Angeles"
    }
}
```

NoSQL Implementations: MongoDB

- MongoDB query language is a very expressive query language based on **JavaScript** (it has a large number of operators)
- MongoDB is **case sensitive**
- The most important tool used to interact with the MongoDB database server is **mongosh** (an interactive JavaScript interpreter)
- **MongoDB Compass** is a graphical user interface for MongoDB
- MongoDB offers five client authentication mechanisms, but authentication is not enabled by default (you have to enable it)
- The authentication mechanisms supported are: SCRAM (default), X.509 Certificate Authentication, LDAP proxy authentication, Kerberos authentication and OpenID Connect authentication (in public preview at this moment)

NoSQL Implementations: MongoDB

- **MongoDB** is also available as **DBaaS** (Database-as-a-Service)
- The **MongoDB DBaaS** is called **MongoDB Atlas**
- Fully managed MongoDB instances can be deployed across **Azure**, **Google Cloud** and **AWS** with **MongoDB Atlas**
- **MongoDB Atlas** offers a free cloud database, with the following characteristics:
 - Shared RAM
 - 512 MB storage
 - Highly available replica set
 - End-to-end encryption
 - Automated patches
 - REST API

NoSQL Implementations: MongoDB

- MongoDB can be installed on:
 - Windows
 - Linux
 - macOS
- The current stable release is MongoDB 7.0.4:

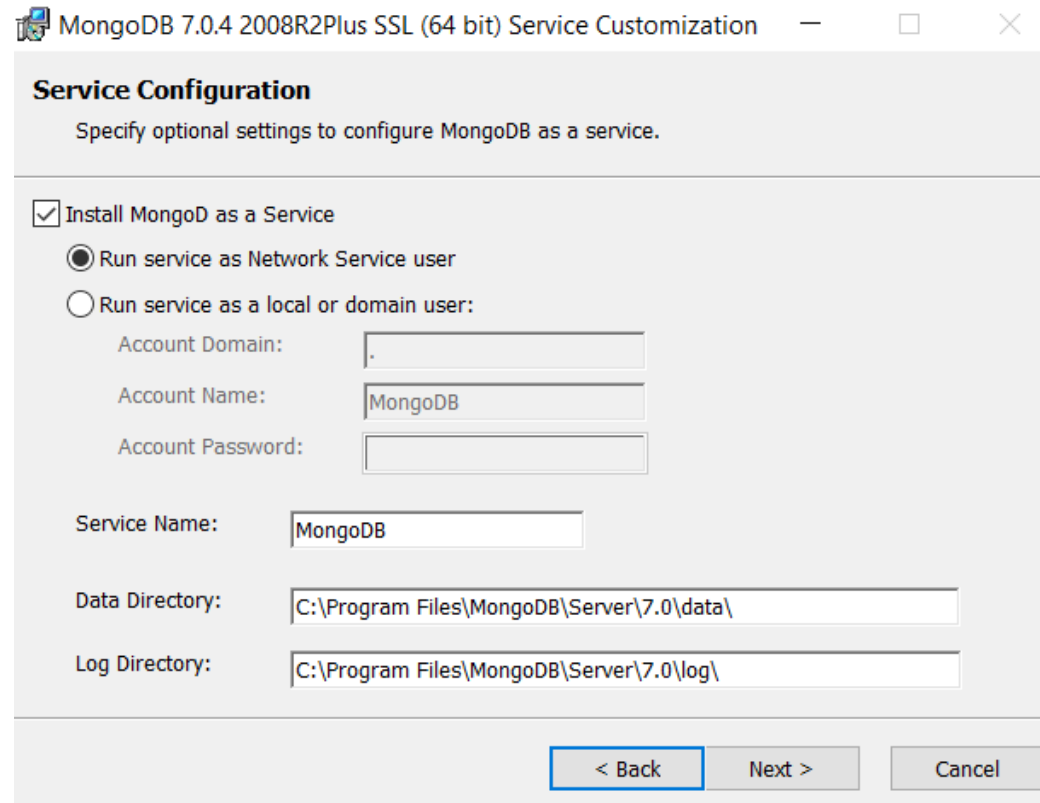
<https://www.mongodb.com/try/download/community>

MongoDB, Inc. offers free online courses:

<https://learn.mongodb.com/>

NoSQL Implementations: MongoDB

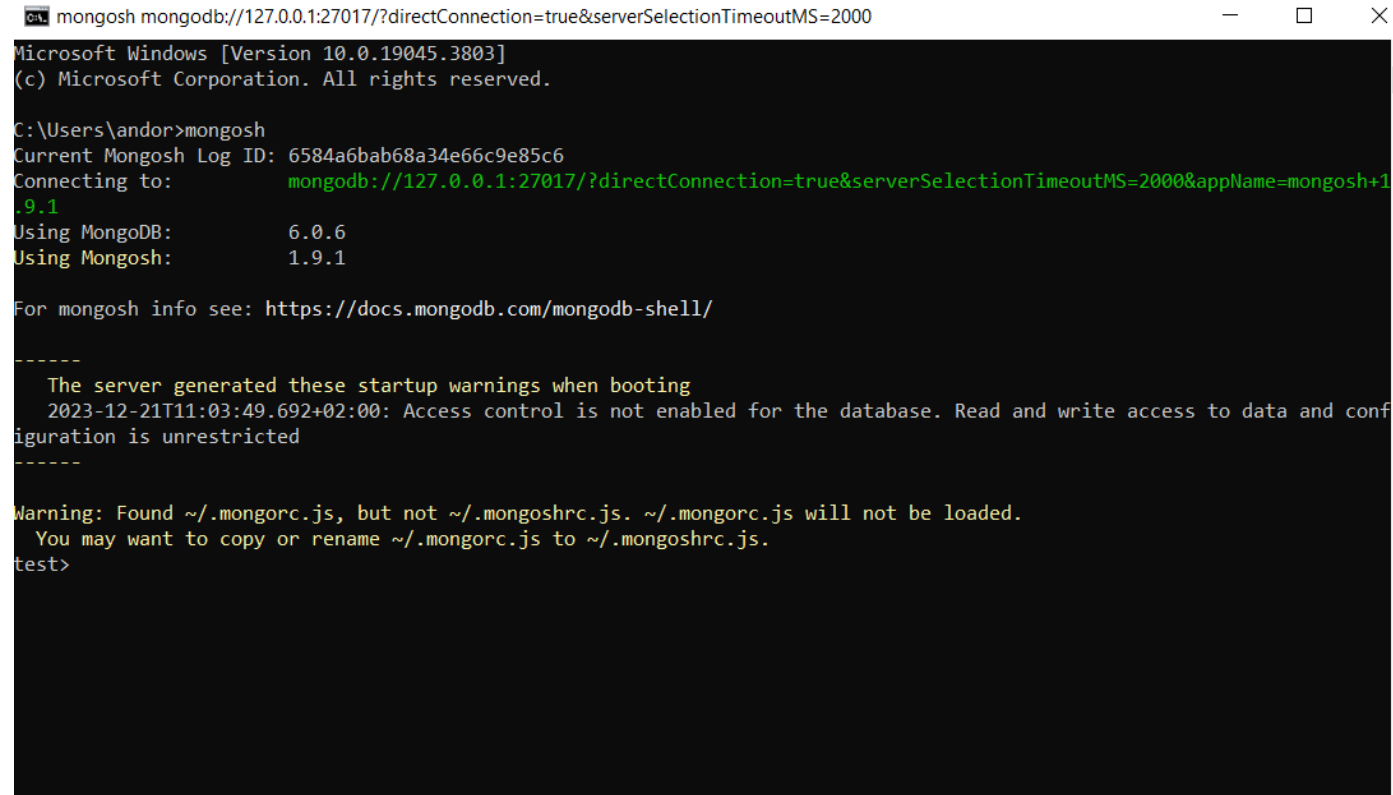
- If MongoDB was installed as a **service**, the *mongod* instance is already running, so you just have to connect to it using **mongosh** or **MongoDB Compass**:



The screenshot shows the 'MongoDB 7.0.4 2008R2Plus SSL (64 bit) Service Customization' window. The title bar includes the application icon, the title text, and standard window controls (minimize, maximize, close). The main content area is titled 'Service Configuration' and contains the instruction 'Specify optional settings to configure MongoDB as a service.' Below this, there is a checkbox labeled 'Install MongoD as a Service' which is checked. Underneath, there are two radio button options: 'Run service as Network Service user' (selected) and 'Run service as a local or domain user:'. The latter option has three associated text input fields: 'Account Domain:' (containing a single dot '.'), 'Account Name:' (containing 'MongoDB'), and 'Account Password:' (empty). Further down, there are three more text input fields: 'Service Name:' (containing 'MongoDB'), 'Data Directory:' (containing 'C:\Program Files\MongoDB\Server\7.0\data\'), and 'Log Directory:' (containing 'C:\Program Files\MongoDB\Server\7.0\log\'). At the bottom right, there are three buttons: '< Back' (highlighted with a blue border), 'Next >', and 'Cancel'.

NoSQL Implementations: MongoDB

- To connect to the MongoDB database server from **mongosh**, having **mongosh** installed, just open a new Command Prompt window, write **mongosh** and press Enter:



```
cal mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

C:\Users\andor>mongosh
Current Mongosh Log ID: 6584a6bab68a34e66c9e85c6
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.9.1
Using MongoDB:      6.0.6
Using Mongosh:      1.9.1

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

-----
  The server generated these startup warnings when booting
  2023-12-21T11:03:49.692+02:00: Access control is not enabled for the database. Read and write access to data and conf
  igation is unrestricted
  -----

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
  You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test>
```


NoSQL Implementations: MongoDB

- To display all databases stored on the MongoDB database server, you must write **show dbs** in mongosh and press Enter:

```
> show dbs
admin          0.000GB
collations     0.001GB
curs13         0.000GB
labs           0.000GB
local          0.000GB
m034           0.011GB
```

- To connect to a specific database, execute the following command:
use database_name
- By default, you are connected to a database called “test”

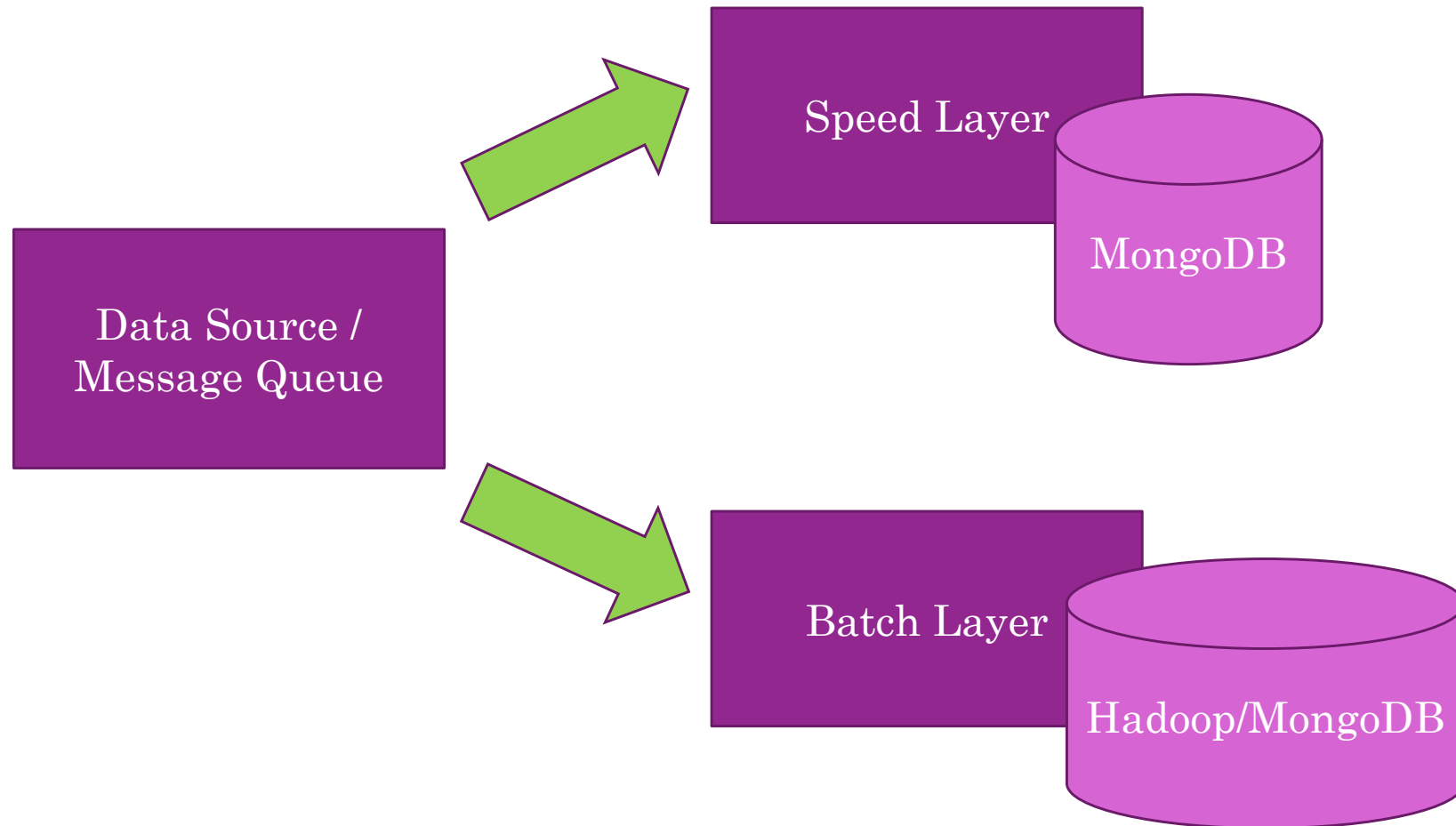
NoSQL Implementations: MongoDB

- To display all collections from the current database, write **show collections** in mongosh and press Enter:

```
> show collections
persons
students
```

- A collection of documents is similar to a relational table, but it has a flexible schema
- A document is similar to a row from a relational table, but it has a rich structure (it allows complex structures like embedded documents or arrays to be stored as values for keys)

MongoDB in a lambda architecture

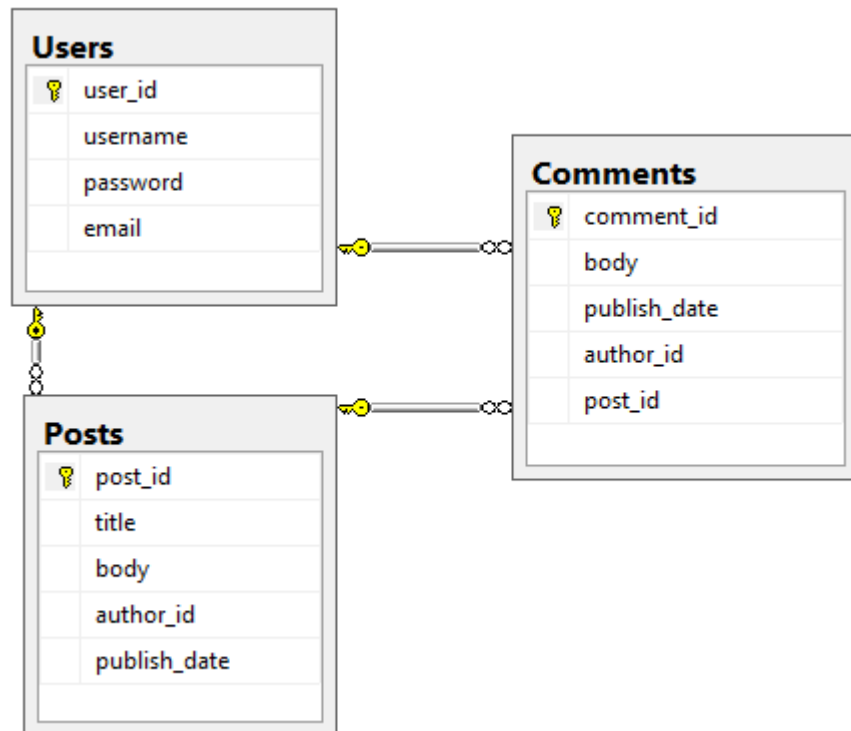


NoSQL Implementations: MongoDB

- Problem at hand:
- Create a database for a web application that belongs to the blog category. This database stores data about users and their posts. Each user has a username, a password and an email address. Each post has a title, a body, a publish date, an author and a list of comments. Each comment has an author, a body and a publish date. One user can have many posts, but each post belongs to a single user (author). A comment belongs to a single user (author), but a user can have many comments. A post can have many comments, but a comment belongs to a single post.

NoSQL Implementations: MongoDB

- Solution – Microsoft SQL Server



NoSQL Implementations: MongoDB

- The relational version of the database contains three tables: **Users**, **Posts** and **Comments**
- There is a **one to many** relationship between **Users** and **Posts** (one user can have many posts, but one post belongs to a single user)
- There is a **one to many** relationship between **Users** and **Comments** (one user can have many comments, but one comment belongs to a single user)
- There is a **one to many** relationship between **Posts** and **Comments** (one post can have many comments, but one comment belongs to a single post)
- There is also a **unique constraint** defined on the column “username”, because generally, the username is unique

NoSQL Implementations: MongoDB

- The SQL script used to generate the database and its tables:

```
CREATE DATABASE Blog;

GO

USE Blog;

--Table Users

CREATE TABLE Users

(user_id INT PRIMARY KEY IDENTITY,

username VARCHAR(100) UNIQUE,

password VARCHAR(50),

email VARCHAR(100)

);
```

NoSQL Implementations: MongoDB

--Table Posts

CREATE TABLE Posts

(post_id INT PRIMARY KEY IDENTITY,

title VARCHAR(200),

body VARCHAR(MAX),

author_id INT FOREIGN KEY REFERENCES Users(user_id),

publish_date DATETIME

);

NoSQL Implementations: MongoDB

```
--Table Comments
```

```
CREATE TABLE Comments
```

```
(comment_id INT PRIMARY KEY IDENTITY,
```

```
body VARCHAR(1000),
```

```
publish_date DATETIME,
```

```
author_id INT FOREIGN KEY REFERENCES Users(user_id),
```

```
post_id INT FOREIGN KEY REFERENCES Posts(post_id)
```

```
);
```

NoSQL Implementations: MongoDB

- Solution – MongoDB
- In MongoDB, a collection has a flexible schema (you don't have to define the schema when you create a collection)
- Documents that belong to the same collection can have different schemas (collections do not enforce document structure)
- MongoDB is a NoSQL implementation, so there is **no need** for normalization
- The things you have to think about are the data access pattern and the most frequent queries
- In MongoDB, a **one to many** relationship can be modeled in more than one way:
 - A single collection is created and the documents that belong to the **many** side of the relationship are embedded in the document that belongs to the **one** side of the relationship
 - Two collections are created and each document stored in the collection that belongs to the **many** side of the relationship contains a reference to a document stored in the collection that belongs to the **one** side of the relationship (and that field contains the value of the “_id” field)

NoSQL Implementations: MongoDB

- The “_id” field is the unique identifier that corresponds to each document stored in a collection in MongoDB
- There are three one to many relationships in our application’s case:
 - A one to many relationship between Users and Posts
 - A one to many relationship between Users and Comments
 - A one to many relationship between Posts and Comments
- Again, in our application’s case, the portions of data that are accessed together are:
 - A post and all its comments
 - All posts written by a user
- For each post and each comment, only the username is required
- In this case, the database will contain two collections:
 - **users** collection
 - **posts** collection

NoSQL Implementations: MongoDB

- In MongoDB, each document contains the “_id” field by default
- The “_id” field represents the unique identifier of each document at collection level (similar to a primary key)
- MongoDB assigns a value for the “_id” field that is unique at collection level (if there is no value specified when the document is inserted)
- The username will be specified as value for the “_id” field for every document inserted in the **users** collection (the username must be unique and will be used as a reference in the **posts** collection)
- When a post or a comment is displayed, only the username of its author is required (in this case, the value stored as reference can be used directly)

NoSQL Implementations: MongoDB

- A document that belongs to the **users** collection:

```
{  
  "_id" : "Bob",  
  "password" : "233567",  
  "email" : "bob@gmail.com"  
}
```

NoSQL Implementations: MongoDB

- A document that belongs to the **posts** collection:

```
{ "_id" : ObjectId("5a53b2c51d2f5a15e12d3109"),  
  "title" : "A sunny day",  
  "author" : "Bob",  
  "body" : "In a sunny day, I saw a nice flower.....",  
  "publish_date" : ISODate("2018-01-08T18:04:00Z"),  
  "comments" : [  
    { "author" : "Tom",  
      "body" : "Nice post ",  
      "publish_date" : ISODate("2018-01-08T19:04:00Z")  
    } ]  
}
```

NoSQL Implementations: MongoDB

- In MongoDB there are no constraints or relationships between collections
- Document references stored in a document that belongs to another collection are not verified at the database level
- There is no support for JOINS, so if JOINS are required, they must be implemented at the application level, as they are not available at the database level (a variant of JOIN - \$lookup exists in the Aggregation Framework but works only on collections that are stored in the same database)
- In our application's case, all comments that belong to a post are stored as an array of embedded documents that corresponds to the “comments” field

NoSQL Implementations: MongoDB

- **How to insert a new user:**

- **Microsoft SQL Server:**

```
INSERT INTO Users (username, password, email)  
VALUES ('Bob', '233567', 'bob@gmail.com');
```

- **MongoDB:**

```
db.users.insertOne({"_id" : "Bob", "password" : "233567", "email" : "bob@gmail.com"})
```

- **How to insert a new post:**

- **Microsoft SQL Server:**

```
INSERT INTO Posts (title, body, author_id, publish_date) VALUES  
('A sunny day', 'In a sunny day, I saw a nice flower.....', 1, GETDATE());
```


NoSQL Implementations: MongoDB

- **MongoDB:**

```
db.posts.insertOne({"title" : "A sunny day", "author" : "Bob", "body" : "In a sunny day, I saw a nice flower.....", "publish_date" : new Date()})
```

- **How to insert a new comment:**

- **Microsoft SQL Server:**

```
INSERT INTO Comments (body, publish_date, author_id, post_id)  
VALUES ('Nice post', GETDATE(), 2, 1);
```

- **MongoDB:**

```
db.posts.updateOne({"title" : "A sunny day"}, {$push:{"comments" : {"author" : "Tom", "body" : "Nice post", "publish_date" : new Date()}}})
```

NoSQL Implementations: MongoDB

- **How to return all users:**

- **Microsoft SQL Server:**

```
SELECT * FROM Users;
```

- Result:

	user_id	username	password	email
1	1	Bob	233567	bob@gmail.com
2	2	Tom	243567	tom@gmail.com

- **MongoDB:**

```
db.users.find().pretty()
```

- Result:

```
{ "_id" : "Bob", "password" : "233567", "email" : "bob@gmail.com" }  
{ "_id" : "Tom", "password" : "243567", "email" : "tom@gmail.com" }
```

NoSQL Implementations: MongoDB

- How to return all posts that have the title equal to “A sunny day” together with their comments:

- Microsoft SQL Server:

```
SELECT * FROM Posts INNER JOIN  
Comments ON Posts.post_id = Comments.post_id  
WHERE title = 'A sunny day';
```

- Result:

	post_id	title	body	author_id	publish_date	comment_id	body	publish_date	author_id	post_id
1	1	A sunny day	In a sunny day, I saw a nice flower...	1	2018-05-07 10:43:10.150	1	Nice post	2018-05-07 10:43:39.783	2	1

NoSQL Implementations: MongoDB

- **MongoDB:**

```
db.posts.find({"title" : "A sunny day"}).pretty()
```

- Result:

```
{
  "_id" : ObjectId("5af00095b3c687bb6e2ff41d"),
  "title" : "A sunny day",
  "author" : "Bob",
  "body" : "In a sunny day, I saw a nice flower.....",
  "publish_date" : ISODate("2018-05-07T07:30:29.831Z"),
  "comments" : [
    {
      "author" : "Tom",
      "body" : "Nice post",
      "publish_date" : ISODate("2018-05-07T07:36:07.634Z")
    }
  ]
}
```

NoSQL Implementations: MongoDB

- How to update the email address of the user called “Bob” and set its value to “bobt@gmail.com”:

- **Microsoft SQL Server:**

```
UPDATE Users SET email = 'bobt@gmail.com' WHERE username = 'Bob';
```

- Result:

	user_id	username	password	email
1	1	Bob	233567	bobt@gmail.com
2	2	Tom	243567	tom@gmail.com

- **MongoDB:**

```
db.users.updateOne({"_id" : "Bob"}, {$set : {"email" : "bobt@gmail.com"}})
```

- Result:

```
{ "_id" : "Bob", "password" : "233567", "email" : "bobt@gmail.com" }  
{ "_id" : "Tom", "password" : "243567", "email" : "tom@gmail.com" }
```

NoSQL Implementations: MongoDB

- How to delete the user called “Tom”:

- Microsoft SQL Server:

```
DELETE FROM Users WHERE username = 'Tom';
```

- MongoDB:

```
db.users.deleteOne({"_id" : "Tom"})
```

- In Microsoft SQL Server, the DELETE operation will fail because there is a foreign key constraint defined in the **Comments** table (the user called “Tom” has a comment)
- In MongoDB, the document that has the “_id” field value equal to “Tom” will be deleted from the **users** collection
- However, the comment inserted in the **posts** collection that has the “author” field value equal to “Tom” remains unchanged (there are no foreign key constraints in MongoDB)

Bibliography

- www.mongodb.com
- <https://www.json.org/>
- <https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/>
- <https://docs.mongodb.com/manual/tutorial/model-referenced-one-to-many-relationships-between-documents/>
- <https://www.mongodb.com/mongodb-architecture>
- <https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/>
- <https://www.mongodb.com/cloud/atlas>

Bibliography

- <https://docs.mongodb.com/manual/core/authentication/>
- <https://www.mongodb.com/json-and-bson>
- <https://docs.mongodb.com/mongodb-shell/>
- <https://www.mongodb.com/nosql-explained/examples>
- <https://www.mongodb.com/pricing>
- <https://www.mongodb.com/docs/manual/core/security-oidc/>
- Pramod J. Sadalage, Martin Fowler - NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence (1st Edition), Addison-Wesley Professional, 2012