Week2_MongoDB_Overview

Basics of MongoDB

Overview

- MongoDB is a document and a NoSQL Database
- Where data is structured in a non-relational way
- Document are associative arrays like JSON objects or Python dictionaries
- For example, a student document:

```
{
"firstName": :JP,
"lastName": "Doe",
"email": "jp.doe@email.com",
"studentId": 20217484
}
```

- Grouped in collections (a group of stored documents)
- For example, all student records in Students section (**collection**), and staff records in Employees section (**collection**)
- Databases stores collections. Students and Employees collections stored in a adatabase called CampusManagementDB
- field/property has values
- MongoDB supports various data types
- Model data as you read/write, not the other way
    - Traditional relational databases: create scheam first, then create tables
    - To store another field, you have to alter tables
- Bring structured/unstructured data
- High availability

It is a popular choice of database for:

- Large and unstructured
- Complex
- Flexible
- Self-managed, hybrid or cloud hosted

Advantages of MongoDB

- Flexibility with Schema
- Code-first approach: no complex table definitions, write as soon as you connect to DB
- Evolving Schema
- Store unstructured data in a single collection
- Querying and Analytics - MongoDB querying using MQL has a range of operators for complex queries
- MongoDB is natively a highly available system:
    - Resilience through redundancy
    - No system maintenance downtime
    - No upgrade downtime

Use cases for MongoDB

- Many Sources - One View
    - No more data silos
    - Easy data ingestion
    - Consolidate different data
    - Flexible schema
- Internet of Things (IoT)
    - Billions of IoT devices around the world
    - Vast amount of data
    - Scale
    - Expressive querying
- E-commerce
    - Products with different attributes
    - Optimise for read
    - Dynamic schema
- Real-time Analytics
    - Quick response to changes
    - Simplified ETL
    - Real time, along with Operational Data
- Gaming
    - Globally scalable
    - No downtime
    - Supporting rapid development

- Finance
  - Speed
  - Security
  - Reliability

Getting started with MongoDB

CRUD operations

`> use campusManagementDB`

`> show collections`

- **Create One**

```
> db.students.insertOne(
{
"firstName": :JP,
"lastName": "Doe",
"email": "jp.doe@email.com",
"studentId": 20217484
}
)
```

- **Create Many**

```
students_list = [
{
"firstName": :Sarah,
"lastName": "Jane",
"email": "sarah@email.com",
"studentId": 20264564
},
{
"firstName": :Peter,
"lastName": "Parker",
"email": "peter@email.com",
"studentId": 20312312
}
]

db.students.insertMany(students_list)
```

- **Read**

  `> db.students.findOne()` - first document

```
> db.students.findOne({"email":"sarah@email.com"})
```

```
> db.students.findOne({"lastName":"Doe"})
```

```
> db.students.count({"lastName":"Doe"})
```

- **Replace**

```
student = db.students.findOne({"lastName":"Doe"})

db.student['onlineOnly'] = true

db.student['email': 'johnd@campus.edu']

db.students.replaceOne({"lastName":"Doe"}, student)
```

- **Update**

```
changes = {"$set": {"onlineOnly": true, "email":"johnd@campus.edu"}}

db.students.updateOne({"lastName":"Doe"}, changes)

db.students.updateMany({}, {"$set": {"onlineOnly":true}})
```

- **Delete**

```
db.students.deleteOne({"studentId": 21231254})

db.students.deleteMany({"graduatedYear": 2019})
```

Indexes

- Help quickly locate data without looking for it everywhere
- For example, in british library, find the book A written by Martin Kleppman
  Without indexes, you will need to look through all of them
- With indexes: Go to Computers/Databases/M (for Martin)
- Indexes in MongoDB are special data structures
- They store the fields you are indexing
- They also store the location of document
- It stores indexes in a tree form
- Most frequent queries

```
db.courseEnrollment.find({"courseId": 1547})

db.courseEnrollment.createIndex({"courseId": 1}) -- store the index in
ascending order
```

```
###
# Use index to sort

db.courseEnrollment.find({"courseId": 1547}).sort({"studentId": 1})

db.courseEnrollment.createIndex({"courseId": 1, "studentId":1})
```

Aggregation Framework

Process..Stages..Outcome

`average student score in each course for 2020`

```
db.courseResults.aggregate([

{$match: {"year": 2020}},

{$group: {"_id": "$courseId", "avgScore": { $avg: "$score" }}}

])
```

- Common Aggregation Stages

`$project` - if you want to change the shape of documents

```
db.students.aggregate([
{$project: {"firstName":1, "lastName":1}}
])
```

`$sort` to sort your documents
`$count` to count and assign the result to a field
`$merge` takes the outcome from the previous stage and stores into a target collection

- Use Cases
  - Reporting
  - Analysis

Replication & Sharding

- Replication is the duplication of data and changes to it
- Replication provides fault tolerance, redundancy and high availability
- Replication does not provide disaster recovery
- You can use Sharding to scale horizontally

Accessing from python

`MongoClient` is a class that helps you interact with MongoDB

```
from pymongo import MongoClient
uri = "mongodb://USER:PASSWORD@uri/test"

client = MongoClient(uri)
campusDB = client.campusManagementDB
students = campusDB.students

## Insert operations
students.insert_one(
{
"firstName": :JP,
"lastName": "Doe",
"email": "jp.doe@email.com",
"studentId": 20217484
}
)

students_list = [
{
"firstName": :Sarah,
"lastName": "Jane",
"email": "sarah@email.com",
"studentId": 20264564
},
{
"firstName": :Peter,
"lastName": "Parker",
"email": "peter@email.com",
"studentId": 20312312
}
]

students.insert_many(students_list)
```

```
## Read Operations

students.find_one() #first document in natural order

students.find_one({"email":"sarah@mail.com"}) #only the first found will be
```

```
returned

students.find({"lastName":"Doe"})

students.count_documents({"lastName":"Doe"})
```

```
###
from bson.json_util import dumps

cursor = students.find({"lastName":"Doe"})
print(dumps(cursor, ident=4))

# Replace

student = students.find_one({"lastName":"Doe"})

student["onlineOnly"] = True

student["email"] = "johnd@campus.edu"

students.replace_one({"lastName":"Doe"}, student)
```

```
# Update

student = students.find_one({"lastName":"Doe"})

changes = {"$set": {"onlineOnly":True},
"email":"johnd@campus.edu"}

students.update_one({"lastName":"Doe"}, changes)

students.update_many({}, {"$set": {"onlineOnly": True}})
```

```
# Delete

students.delete_one({"studentId": 2024015})

students.delete_many({"studentId": 2024015})
```