Non-Relational Databases

Chapter No.	26
Status	Completed

- Comparing SQL & NoSQL Databases
  - Comparison of Properties

SQL VS NoSQL (Properties)

SQL Database	NoSQL Database
Has a predefined schema	Has no predefined schema, making it dynamic
Data is stored in tables with a fixed type of data in each field	Data is stored in collections of documents with no fixed data types
Join operations are used to retrieve data from multiple tables	No join operations

  - Comparison of Terminology

SQL VS MongoDB (Terminology)

SQL Term	MongoDB Term
Database	Database
Table	Collection
Record/Row	Document
Field/Column	Field

- When to Use Each Type of Database
  - When To Use a SQL Database
    - The data stored has a fixed schema with the ACID properties critical to the database
      - The data stored is structured and is comprised of known data types
      - Atomicity
        - A transaction takes place completely or does not happen at all
      - Consistency
        - Integrity constraints are maintained at all times
      - Isolation
        - Multiple transactions can occur concurrently without leading to the inconsistency of the database state
      - Durability
        - Once a transaction has been completed, the updates and modifications to the database are saved even if the system fails or restarts
    - Complex and varied queries will be frequently performed
    - A high number of simultaneous transactions will be performed
  - When To Use a NoSQL Database
    - The data stored has a dynamic schema
      - The data is unstructured and is comprised of multiple unknown data types
    - Data storage needs to be performed quickly
    - Simple queries are often made

        - Making simple queries is faster on NoSQL databases as compared to SQL databases
    - An extremely large amount of data needs to be stored
    - All the data about a particular object needs to be easily accessed

- Benefits of NoSQL Databases Over SQL Databases
    - NoSQL databases can store unstructured data while SQL databases cannot
        - SQL databases have predefined schemas which are difficult to change
        - Should we wish to add a field to only a small number of records, we need to include the field for the entire table
        - NoSQL databases overcome this by storing data in documents which do not need to be of the same format
    - It is more usually cost-effective to store the same amount of data in a NoSQL database as compared to an SQL database
        - NoSQL databases support hierarchical data storage while SQL databases usually do not
            - Hierarchical data storage is the process of moving less frequently used data to cheaper, but slower, storage devices
        - NoSQL databases are horizontally scalable while SQL databases are usually vertically scalable
            - Horizontal scaling refers to improving the performance of a database by adding more servers
                - This is generally more cost-effective as mass-produced average-performance computers are easily available at low prices
            - Vertical scaling refers to improving the performance of a database by upgrading the existing server with better components
                - Such high-performance components can be expensive
                - There is an upper limit to how much a single server can be upgraded as the server only has a certain limited capacity
    - NoSQL databases are generally more reliable than SQL databases
        - SQL databases are stored on one server, thus the database will be unavailable if the server fails
        - NoSQL databases are stored on multiple servers so that if one server fails, the other servers can continue to provide access to the database

- Python & PyMongo
    - PyMongo Module Summary
        - MongoClient(ip_address, port)
            - Creates a MongoDB Client object via a connection to the given IP address and port number
    - Class Method Summaries
        - Client Class Methods
            - database_names( )
                - Shows all the databases in a list
            - get_database(name)
                - Declares a Database object of the given name
            - drop_database(name)
                - Deletes the Database object of the given name
            - close( )
                - Closes the connection to the MongoDB database

- Database Class Methods
  - collection_names( )
    - Shows all the collections in a list
  - get_collection(name)
    - Declares a Collection object of the given name
  - drop_collection(name)
    - Deletes the Collection object of the given name
- Collection Class Methods
  - insert_one(nosql)
    - Inserts one document given a nosql statement
  - insert_many(list_of_nosql)
    - Inserts multiple documents given a list of nosql statements
  - find(nosql)
    - Returns a Cursor object containing the documents which match the given nosql statment
  - find_one(nosql)
    - Returns a Cursor object containing the first document which matches the given nosql statment
  - update_one(nosql1, nosql2)
    - Finds the first document which matches the given nosql1 statement and updates it given the nosql2 statement
  - update_many(nosql1, nosql2)
    - Finds all the documents which match the given nosql1 statement and updates them given the nosql2 statement
  - delete_one(nosql)
    - Deletes the first document which matches the given nosql statement
  - delete_many(nosql)
    - Deletes all the documents which match the given nosql statement
  - count( )
    - Returns the total number of documents
- MongoDB Queries
  - Query Operators

MongoDB Query Operators

Query Operator	Meaning
$eq	equals to
$ne	not equals to
$gt	greater than
$gte	greater than or equal to
$lt	less than
$lte	less than or equal to
$in	in a specified list
$nin	not in a specified list
$or	logical OR
$and	logical AND
$not	logical NOT

Query Operator	Meaning
$exists	matches documents which have the named field

▼ Using $eq, $ne, $gt, $gte, $lt & $lte

```
#refers to documents which have the value 21 in their "age" field
{"age":{"$eq":21}}
```

▼ Using $in & $nin

```
#refers to documents which have the value "Math", "Chemistry" or "Computing" in their "subject" field
{"subject":{"$in":["Math", "Chemistry", "Computing"]}}
```

▼ Using $or, $and & $not

```
#refers to documents which have a value between 10 and 20 in their "age" field
{"$and":[{"age":{"$gt":10}}, {"age":{"lt":20}}]}
```

▼ Using $exists

```
#refers to documents which have the field "graduated"
{"graduated":{"$exists":True}}
```

▼ PyMongo Operations

▼ Connecting to a MongoDB Database

- MongoClient(ip_address, port)
- get_database(name)
- get_collection(name)

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("school_A")
coll = db.get_collection("class_1")

client.close()
```

▼ INSERT

- insert_one(nosql)
- insert_many(list_of_nosql)

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("school_A")
coll = db.get_collection("class_1")

coll.insert_one({"name":"Jack", "age":14, "cca":"tennis"})

to_insert = [{"name":"Bill", "age":14, "cca":"basketball"}, {"name":"Jeff", "age":14, "cca":"bowling"}, {"name":"Mary", "age"
coll.insert_many(to_insert)

client.close()
```

▼ FIND

- find(nosql)
- find_one(nosql)

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("school_A")
coll = db.get_collection("class_1")

cursor = coll.find({"name":"Jack"})
for doc in cursor:
  print(doc)

cursor = coll.find_one({"name":"Bill"})
for doc in cursor:
  print(doc["cca"])

client.close()
```

▼ UPDATE

- update_one(nosql1, nosql2)
- update_many(nosql1, nosql2)

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("school_A")
coll = db.get_collection("class_1")

search1 = {"name":"Mary"}
update1 = {"$set":{"gender":"F"}}
coll.update_one(search1, update1)

search2 = {"age":14}
update2 = {"$unset":{"cca":0}}
coll.update_many(search2, update2)

client.close()
```

▼ DELETE

- delete_one(nosql)
- delete_many(nosql)

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("school_A")
coll = db.get_collection("class_1")

coll.delete_one({"name":"Mary"})

coll.delete_many({"age":14})

client.close()
```

▼ COUNT

- count( )

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("school_A")
coll = db.get_collection("class_1")

number_of_doc = coll.count()
print(number_of_doc)

client.close()
```

▼ DROP

- drop_collection(name)

- drop_database(name)

```
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
db = client.get_database("school_A")
coll = db.get_collection("class_1")

db.drop_collection("class_1")

client.drop_database("school_A")

client.close()
```