



# COURSEWORK -1

## NOSQL STORAGE

Jason Shawn D' Souza | H00202543 | F20BD – Big Data Management | 24/2/2018

## Contents

INTRODUCTION .....	2
MONGODB AS NOSQL STORAGE.....	2
DATA MODELLING .....	3
Route Schema .....	4
Countries schema .....	6
DATA ETL.....	9
Data Extraction .....	10
Transforming Data.....	11
Loading DATA INTO MONGODB .....	13
Main method.....	13
<b>PROOF OF ETL</b> .....	13
Queries.....	16
Query -1 .....	16
Query – 2 .....	18
Query-3 .....	19
Query-4 .....	21
Query-5 .....	22
Code – Listing .....	24
References .....	28

## INTRODUCTION

In this assignment, I was asked to move from a relational database(MySQL) to a NoSQL storage of my choice. I chose MongoDB, and the MySQL dump provided to me was successfully extracted, transformed and then loaded into MongoDB as will be discussed in this report.

## MONGODB AS NOSQL STORAGE

The reason I chose MongoDB over other NoSQL stores such as Neo4j and many others is because I felt having a document store would be better than a graph database (Neo4J). Mongo DB is also the most popular NoSQL database in the world currently and with good reason [1].

It is easier to shard (Splitting large databases into smaller, faster and easily accessible parts) in MongoDB and cannot be seen in Neo4J [2] (Partitioning method in the table on the website referenced). In addition to sharding, document stores like mongodb have no fixed schema to follow and this is good for businesses and companies that continually expand. The Aviation Industry is always on the rise and in using MongoDB the data entered the storage will achieve either eventual or immediate consistency [2] (Consistency Concepts field in the table on the website referenced)

Another interesting point to note about MongoDB is that it follows a replication model similar to that of MySQL – the Master-Slave replication model [3]. What this means is that this model consists of one “Master” and several “Slave” databases which allows for no downtime as one slave can be taken offline and the other slaves take on its work until the slave returns and can be synced back to the master resulting in **little to no downtime.**

## DATA MODELLING

The MySQL data dump provided consists of 5 tables –

1. Airports
2. Airlines
3. Cities
4. Countries
5. Routes

The data to be transferred to MongoDB was modeled to make data retrieval quick and efficient. For example, some airports are in the same city so it would be inefficient to make it so Airports collection contained cities (as embedded documents) as sometimes cities would be repeated (Real-life example, If I made an airports collection Maktoum Intl. Airport and Dubai Intl. Airport would have the same city Dubai and would be repeated).

So, considering this I modelled the Cities collection to contain an array of airports so if the city had no airports, 1 airport or more they would appear in this array and would avoid replication which would not be the case in the airports and embedded cities collection.

The schema was written in reference with the official MongoDB documentation [4]

### Cityschema (contain cities with embedded airport/s)

```
db.createCollection("cities", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      properties: {
        cid: {
          bsonType: "int",
          description: "must be a string "
        },
        name: {
          bsonType: "string",
          description: "Must be a string "
        },
        country: {
          bsonType: "string",
          description: "must be a string "
        },
        timezone: {
          bsonType: "int",
          description: "must be a int"
        },
        tzid: {
          bsonType: "string",
          description: "must be a string "
        },
      },
    },
  },
})
```

## ROUTE SCHEMA

After settling on the previous schema, it was obvious to me what my other collections would look like, I would make no changes to the routes table from MySQL and I would have airlines embedded in the countries collection. Following is the schema for the routes collection:

#### RouteSchema

```
db.createCollection("routes", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      properties: {
        alid: {
          bsonType: "int",
          description: "must be a int "
        },
        src_ap: {
          bsonType: "string",
          description: "Must be a string "
        },
        dst_ap: {
          bsonType: "string",
          description: "must be a string"
        },
        src_apid: {
          bsonType: "int",
          description: "must be a int "
        },
        dst_apid: {
          bsonType: "int",
          description: "must be a int "
        },
        codeshare: {
          enum: [ "", "Y" ],
          description: "can only be one of the enum values"
        },
        stops: {
          bsonType: "string",
          description: "String of how many stops"
        },
        equipment: {
          bsonType: "string",
          description: "must be a string and is req."
        },
        rid: {
          bsonType: "int",
          description: "must be an int and is req."
        }
      }
    }
  }
})
```

```
})
```

## COUNTRIES SCHEMA

The logic behind having airlines array embedded in a countries document is also based on real-life. For example, Emirates and Etihad Airways both belong to the United Arab Emirates. So instead of having an airlines collection where a country that owns multiple airlines can be repeated, I decided it was better to have a collection of countries where the airline array would contain the airlines they own. The schema is as follows

### Countries Schema (countries with embedded airlines)

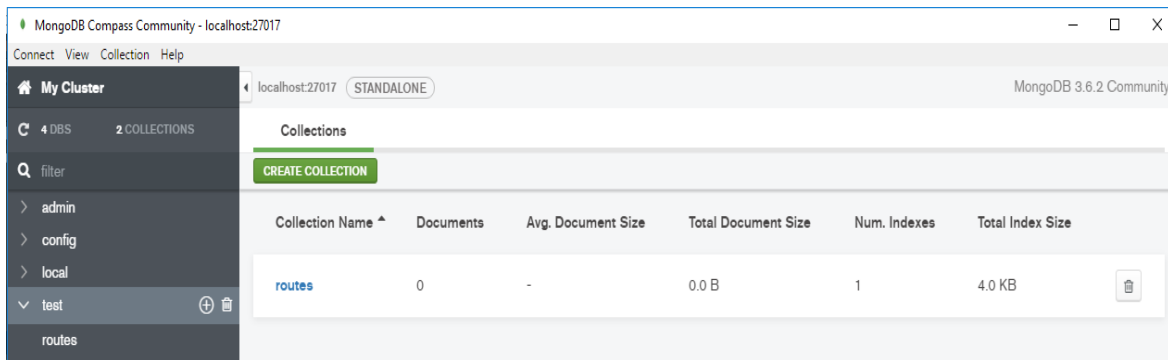
```
db.createCollection("countries", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      properties: {
        code: {
          bsonType: "string",
          description: "must be a string "
        },
        name: {
          bsonType: "string",
          description: "Must be a string "
        },
        oa_code: {
          bsonType: "string",
          description: "must be a string"
        },
        dst: {
          enum: [ "U", "E", "N", "S", "A", "Z", "" ],
          description: "can only be one of the enum values "
        },
        airlines: {
          bsonType: "array",
          "items":{
            bsonType : "object",
            properties: {
              name: {
                bsonType: "string",
                description: "must be a string "
              },
              iata: {
                bsonType: "string",
                description: "Must be a string "
              },
              icao: {
                bsonType: "string",
```





```
Command Prompt - mongo
...      src_apid: {
...          bsonType: "int",
...          description: "must be a int "
...      },
...      dst_apid: {
...          bsonType: "int",
...          description: "must be a int and is optional"
...      },
...      codeshare: {
...          enum: [ "", "Y" ],
...          description: "can only be one of the enum values"
...      },
...      stops: {
...          bsonType: "string",
...          description: "String must be entered"
...      },
...      equipment:{
...          bsonType: "string",
...          description: "must be a string ."
...      },
...      rid: {
...          bsonType: "int",
...          description: "must be an int "
...      }
...  }
... }
... }
... })
{ "ok" : 1 }
```

Inputting the schema



MongoDB Compass shows the schema has been inputted successfully

Then after inputting the routes schema, the other two schemas also were successfully imported as can be seen from the GUI in the following screenshot

Connect View Collection Help					
My Cluster	localhost:27017 STANDALONE				
DBS COLLECTIONS					
filter	CREATE COLLECTION				
admin					
config					
local					
test					
cities					
countries					
routes					

Collection Name ^	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
cities	0	-	0.0 B	1	4.0 KB
countries	0	-	0.0 B	1	4.0 KB
routes	0	-	0.0 B	1	4.0 KB

## DATA ETL

Although I could have done this part manually, I chose to write a python script that does the etl process ( manually transforming the sql data would be a pain).

**NOTE: - ALL CODE WILL BE MADE AVAILABLE AT THE END OF THE REPORT TO MAINTAIN QUALITY**

Also, pymysql was used as the import of choice instead of mysql as the following error would pop up on trying to install it (pip install)

ailed to install package mysql

```
Executed command:
pip install mysql

Error occurred:
! error: command 'C:\\Program Files (x86)\\Microsoft Visual Studio 14.0\\VC\\BIN\\link.exe' failed with exit status 1120

Proposed solution:
Try to run this command from the system terminal. Make sure that you use the correct version of 'pip' installed for your Python interpreter located at 'C:\\Users\\Jason Shawn D' Souza\\PycharmProjects\\cw\\venv\\Scripts\\python.exe'.

Command output:
mysqlclient.lib(typelib.obj) : error LNK2001: unresolved external symbol __iob_func
mysqlclient.lib(viossifactories.obj) : error LNK2001: unresolved external symbol __iob_func
mysqlclient.lib(my_winfile.obj) : error LNK2001: unresolved external symbol __iob_func
mysqlclient.lib(my_messnc.obj) : error LNK2001: unresolved external symbol __iob_func
mysqlclient.lib(client.obj) : error LNK2001: unresolved external symbol __iob_func
mysqlclient.lib(my_thr_init.obj) : error LNK2001: unresolved external symbol __iob_func
mysqlclient.lib(my_init.obj) : error LNK2001: unresolved external symbol __iob_func
mysqlclient.lib(default.obj) : error LNK2001: unresolved external symbol __iob_func
mysqlclient.lib(default.obj) : error LNK2001: unresolved external symbol _printf
build\\lib.win32-3.6\\_mysql.cp36-win32.pyd : fatal error LNK1120: 2 unresolved externals
error: command 'C:\\Program Files (x86)\\Microsoft Visual Studio 14.0\\VC\\BIN\\link.exe' fai

-----

Failed building wheel for MySQL-python
Command "'C:\\Users\\Jason Shawn D' Souza\\PycharmProjects\\cw\\venv\\Scripts\\python.exe" -u -c "import
```

## DATA EXTRACTION

```
def extract_sqldata(sql_cursor):
    #http://pymysql.readthedocs.io/en/latest/user/examples.html
    #instead of individual querying i made a method for it
    cities = sql_query('select * from cities', sql_cursor, 'fetchall')
    countries = sql_query('select * from countries', sql_cursor, 'fetchall')
    airports = sql_query('select * from airports', sql_cursor, 'fetchall')
    airlines = sql_query('select * from airlines', sql_cursor, 'fetchall')
    routes = sql_query('select * from routes', sql_cursor, 'fetchall')
    data = (cities, countries, airports, airlines, routes)
    return data
```

*Function that extracts MySQL data*

As can be seen in the full code in the appendix, this method extracts all data from the MySQL tables. (This method is preceded by a function that facilitates fetching all the data/fetchall and other functions that initialize both the MySQL and MongoDB databases).

The following screenshot shows methods that initialize mysql and the sql\_query method that enables the data extraction.

```
def initialise_mysql():
    #Initialize mysql with given parameters
    return pymysql.connect(
        host="localhost",
        user="root",
        password="",
        db="cw"
    )

def sql_query(sql, cursor, query_type):
    #Function that will be used to carry out queries
    if query_type == "fetchall":
        #http://pymysql.readthedocs.io/en/latest/modules/cursors.html
        #documentnation was used
        cursor.execute(sql)
        return cursor.fetchall()
    else:
        print("fetchall only try again ")
```

## TRANSFORMING DATA

```
def transform_to_mongo(collection, sqltable):
    collection_test = []
    sqltable_data = {}
    if sqltable == "routes":
        for item in collection[4]:
            sqltable_data['alid'] = replaceNullInt(item[0])
            sqltable_data['src_ap'] = replaceNull(item[1])
            sqltable_data['src_apid'] = replaceNullInt(item[2])
            sqltable_data['dst_ap'] = replaceNull(item[3])
            sqltable_data['dst_apid'] = replaceNullInt(item[4])
            sqltable_data['codeshare'] = replaceNull(item[5])
            sqltable_data['stops'] = replaceNullInt(item[6])
            sqltable_data['equipment'] = replaceNull(item[7])
            sqltable_data['rid'] = replaceNullInt(item[8])
            collection_test.append(copy.copy(sqltable_data))
        #appended
        return collection_test
    elif sqltable == "cities":
```

*Function that transforms the MySQL*

The above screenshot shows the function that transforms the MySQL data. The “routes” in the if condition references the extract methods “routes”. This means that all data from the MySQL

routes table is transferred here and placed in a dictionary (sqltable\_data) and finally appended to a list (collection\_test) that is then loaded in the MongoDB database.

The replaceNull and replaceNullInt that act on data are methods written by me ( with reference to gis.stackexchange.com) that replaces null with "" or nothing to preserve string/number (int or double) identity.

```
def replaceNull(x):  
    if x is None:  
        return ""  
    elif x is "Null":  
        return ""  
    else:  
        return x
```

Function that replaces null

The following screenshot shows how embedding is carried out through my python script (Countries collection)

```
elif sqltable == "countries":  
    for item in collection[1]:  
        sqltable_data['code'] = replaceNull(item[0])  
        sqltable_data['name'] = replaceNull(item[1])  
        sqltable_data['oa_code'] = replaceNull(item[2])  
        sqltable_data['dst'] = replaceNull(item[3])  
        airlines_collection=[]  
        secondtest={}  
        for airlineitem in collection[3]:  
            if airlineitem[4]==sqltable_data['code']:  
                #matching tables  
                secondtest['alid'] = replaceNullInt(airlineitem[5])  
                secondtest['name'] = replaceNull(airlineitem[0])  
                secondtest['iata'] = replaceNull(airlineitem[1])  
                secondtest['icao'] = replaceNull(airlineitem[2])  
                secondtest['callsign'] = replaceNull(airlineitem[3])  
                secondtest['alias'] = replaceNull(airlineitem[6])  
                secondtest['mode'] = replaceNull(airlineitem[7])  
                secondtest['active'] = replaceNull(airlineitem[8])  
                airlines_collection.append(copy.copy(secondtest))  
        sqltable_data['airline'] = airlines_collection  
        collection_test.append(copy.copy(sqltable_data))  
    return collection_test
```

Transforming countries schema that contains embedded airlines

The second for loop matches the airlines country with the country code and only then will it be embedded in the countries collection.

## LOADING DATA INTO MONGODB

```
def initialise_mongo():  
    return MongoClient("localhost", 27017)["test"] # MongoClient ( host,port no.(default is 27017) [dbname]
```

### Function to initialize mongo

```
def load_data(mongo_collection, collection_test):  
    if DB_EXISTS: #mentioned earlier flag to check if country exists  
        mongo_collection.delete_many({})  
    else:  
        pass  
    return mongo_collection.insert_many(collection_test)
```

### Function to load data into mongo

After initializing MongoDB as shown in the first screenshot, the load\_data method first checks if the database exists, if it doesn't it just inserts all data. If the database does exist, it first deletes the contents then inserts. DB\_EXISTS is a Boolean flag that is set to FALSE initially when the DB had no data present but during testing stage (after the first attempt) it did contain some data so the flag was set to true or else documents would be duplicated (instead of deleting and then adding them again).

## MAIN METHOD

The main method just executes these functions/methods and contains print functions that let you know at what stage of pipeline/what the pipeline process is currently carrying out.

NOTE: - ALL CODE WILL BE MADE AVAILABLE IN THE APPENDIX

## PROOF OF ETL

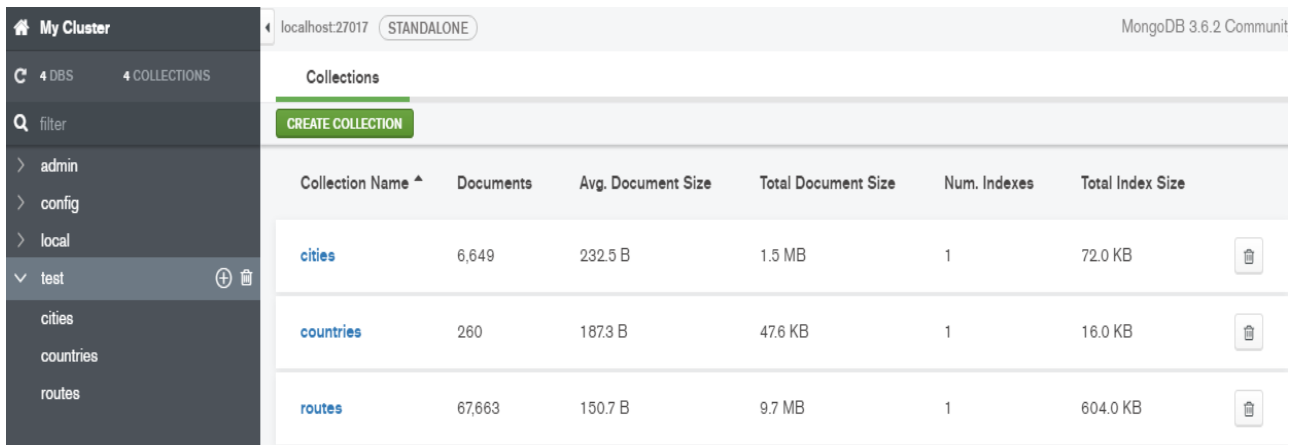
The following screenshot shows the running of the script and its completion

```
Successfully loaded routes now cities
Loading cities
Stage 3 completed! Data successfully loaded
Closing MySQL connection
MySQL connection closed successfully
Ending data pipeline

Process finished with exit code 0
```

Terminal on running the script (ending of the script)

And on visiting the GUI it shows the changes that took place after the ETL process has finished



The screenshot shows the MongoDB GUI interface. On the left, a sidebar lists the database structure: 'My Cluster' (localhost:27017, STANDALONE), '4 DBS', '4 COLLECTIONS', and a 'test' database containing 'cities', 'countries', and 'routes' collections. The main panel displays a table of collections for the 'test' database.

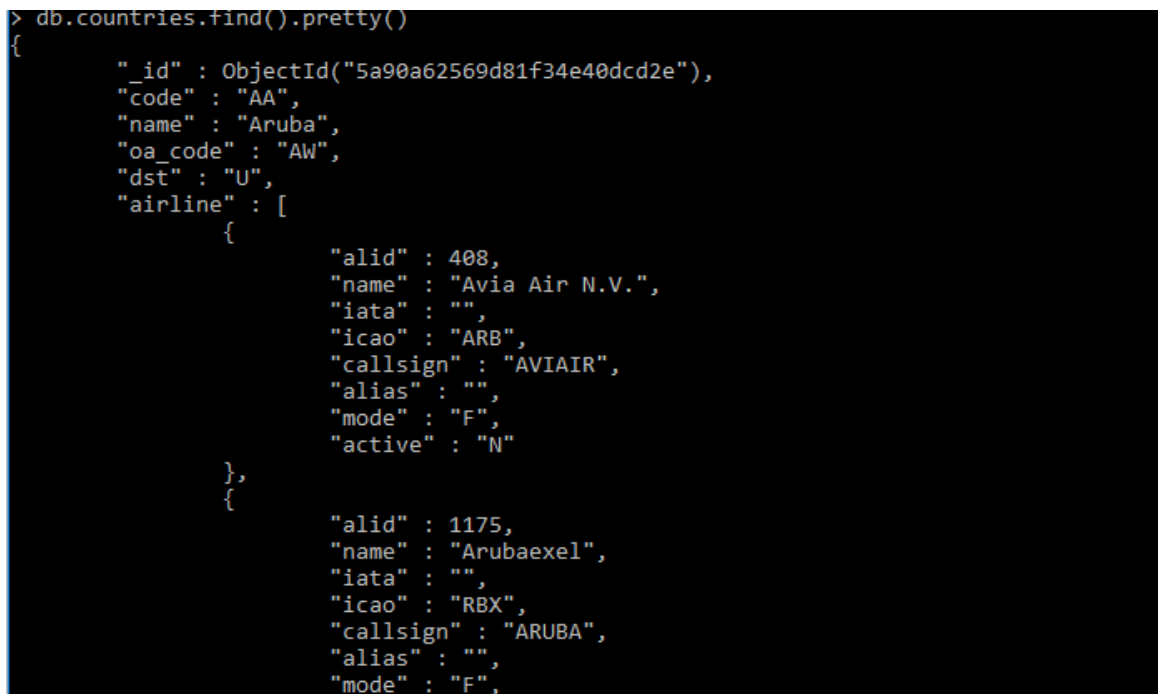
Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size
cities	6,649	232.5 B	1.5 MB	1	72.0 KB
countries	260	187.3 B	47.6 KB	1	16.0 KB
routes	67,663	150.7 B	9.7 MB	1	604.0 KB

MongoDB showing that a successful ETL has taken place

The number of documents prove that all data has been successfully loaded into MongoDB. For further proof I will open the countries collection:

```
_id: ObjectId("5a90a62569d81f34e40dcd2e")
code: "AA"
name: "Aruba"
oa_code: "AW"
dst: "U"
airline: Array
  ✓ 0: Object
    alid: 408
    name: "Avia Air N.V."
    iata: ""
    icao: "ARB"
    callsign: "AVIAIR"
    alias: ""
    mode: "F"
    active: "N"
  > 1: Object
  > 2: Object
  > 3: Object
  > 4: Object
  > 5: Object
```

#### Visualizing the Countries collection using MongoDB Compass



```
> db.countries.find().pretty()
{
  "_id" : ObjectId("5a90a62569d81f34e40dcd2e"),
  "code" : "AA",
  "name" : "Aruba",
  "oa_code" : "AW",
  "dst" : "U",
  "airline" : [
    {
      "alid" : 408,
      "name" : "Avia Air N.V.",
      "iata" : "",
      "icao" : "ARB",
      "callsign" : "AVIAIR",
      "alias" : "",
      "mode" : "F",
      "active" : "N"
    },
    {
      "alid" : 1175,
      "name" : "Arubaexel",
      "iata" : "",
      "icao" : "RBX",
      "callsign" : "ARUBA",
      "alias" : "",
      "mode" : "F",
      "active" : "N"
    }
  ]
}
```

#### Visualizing the Countries collection using the mongo shell

The screenshots are further proof that all data has been successfully transferred from MySQL to MongoDB.



## Queries

### QUERY -1

```
db.cities.aggregate([{$lookup:{from: "countries",localField: "country",foreignField: "code",as:
"countrylink" } },
```

```
{ $match : { $and : [{timezone: 4},{country: "AE"},{name:"Dubai"}}] }).pretty()
```

The above query connects to the cities collection and joins it with the country collection if they have the same country code (country in the cities collection and code in the country collection) . This join is called country link and in this query I look for the country of UAE and the city of dubai with the correct timezone of 4 (+4 GMT). **This query gives a list of airports in the city and the airlines the country owns.** The reason of inclusion of timezone is because some country code AE had a different timezone (of -5) and was not relevant to the UAE as will be shown in the following screenshot

	_id ObjectId	cid Int32	name String	country String	timezone Double
1	5a90a62569d81f34e40db354	32	"Abu Dhabi"	"AE"	4
2	5a90a62569d81f34e40db397	99	"Al Ain"	"AE"	4
3	5a90a62569d81f34e40db39e	106	"Al Hamra"	"AE"	4
4	5a90a62569d81f34e40db48b	343	"Arzana"	"AE"	4
5	5a90a62569d81f34e40db7fb	1223	"Clarksville"	"AE"	-5
6	5a90a62569d81f34e40db8d7	1443	"Das Island"	"AE"	4
7	5a90a62569d81f34e40db959	1573	"Dubai"	"AE"	4

City with different timezone

On running the query, the following is observed:

```
> db.cities.aggregate([{$lookup:{from: "countries",localField: "country",foreignField: "code",as: "countrylink" } },
... { $match : {$and : [{timezone: 4},{country: "AE"},{name:"Dubai"}]} }]).pretty()
{
  "_id" : ObjectId("5a90a62569d81f34e40db959"),
  "cid" : 1573,
  "name" : "Dubai",
  "country" : "AE",
  "timezone" : 4,
  "tz_id" : "Asia/Dubai",
  "airport" : [
    {
      "apid" : 2188,
      "name" : "Dubai International Airport",
      "iata" : "DXB",
      "icao" : "OMDB",
      "x" : 55.3643989563,
      "y" : 25.2527999878,
      "elevation" : 62
    },
    {
      "apid" : 8076,
      "name" : "Al Maktoum International Airport",
      "iata" : "DWC",
      "icao" : "OMDW",
      "x" : 55.161389,
      "y" : 24.896356,
      "elevation" : 114
    }
  ],
  "countrylink" : [
    {
      "_id" : ObjectId("5a90a62569d81f34e40dcd30"),
      "code" : "AE",
      "name" : "United Arab Emirates",
      "oa_code" : "AE",
      "dst" : "U",
      "airline" : [
        {
          "alid" : 27,
          "name" : "Aerovista Airlines",
          "iata" : "",
          "icao" : "AAP",
          "callsign" : "AEROVISTA GROUP",
          "alias" : ""
        }
      ]
    }
  ]
}
```

Running the query gives list of cities airports and airlines in the country

```
"countrylink" : [
  {
    "_id" : ObjectId("5a90a62569d81f34e40dcd30"),
    "code" : "AE",
    "name" : "United Arab Emirates",
    "oa_code" : "AE",
    "dst" : "U",
    "airline" : [
      {
        "alid" : 27,
        "name" : "Aerovista Airlines",
        "iata" : "",
        "icao" : "AAP",
        "callsign" : "AEROVISTA GROUP",
        "alias" : "",
        "mode" : "F",
        "active" : "N"
      },
      {
        "alid" : 329,
        "name" : "Air Arabia",
        "iata" : "G9",
        "icao" : "ABY",
        "callsign" : "ARABIA",
        "alias" : "",
        "mode" : "F",
        "active" : "Y"
      },
      {
        "alid" : 366,
```

Query returning a list of airlines owned by the country

As can be seen in the above two screenshots ( both screenshots have been included so as to confirm the query returns a list of airlines owned by the country) the query successfully returns a list of airports in a given city and also airlines owned by the country the city is in ( timezone optional but should be used for more accurate results).

## QUERY – 2

```
db.cities.aggregate([ { $match : { name: "Abu Dhabi" } } , { $project:
{numberOfAirportsInTheCity: { $size: "$airport" }}}]).pretty()
```

**The above query counts the amount of airports a city contains.** (Match with a city name, \$project enables me to choose what fields to show ,for example, \_id : 0 disables the printing/display of the objectId)

```
> db.cities.aggregate([ { $match : { name: "Abu Dhabi" } } ]).pretty()
{
  "_id" : ObjectId("5a90a62569d81f34e40db354"),
  "cid" : 32,
  "name" : "Abu Dhabi",
  "country" : "AE",
  "timezone" : 4,
  "tz_id" : "Asia/Dubai",
  "airport" : [
    {
      "apid" : 2179,
      "name" : "Abu Dhabi International Airport",
      "iata" : "AUH",
      "icao" : "OMAA",
      "x" : 54.651100158691406,
      "y" : 24.433000564575195,
      "elevation" : 88
    },
    {
      "apid" : 2180,
      "name" : "Bateen Airport",
      "iata" : "AZI",
      "icao" : "OMAD",
      "x" : 54.458099365234375,
      "y" : 24.428300857543945,
      "elevation" : 16
    },
    {
      "apid" : 2184,
      "name" : "Al Dhafra Air Base",
      "iata" : "",
      "icao" : "OMAM",
      "x" : 54.547698974599996,
      "y" : 24.248199462900004,
      "elevation" : 77
    }
  ]
}
```

*General query showing Abu Dhabi has 3 airports*

```
> db.cities.aggregate([{$match : { name: "Dubai" } } , { $project: {numberOfAirportsInTheCity: { $size: "$airport" }}}]
).pretty()
{
  "_id" : ObjectId("5a90a62569d81f34e40db959"),
  "numberOfAirportsInTheCity" : 2
}
> db.cities.aggregate([{$match : { name: "Abu Dhabi" } } , { $project: {numberOfAirportsInTheCity: { $size: "$airport"
}}}] ).pretty()
{
  "_id" : ObjectId("5a90a62569d81f34e40db354"),
  "numberOfAirportsInTheCity" : 3
}
```

*My query confirming Abu Dhabi has 3 airports and Dubai has 2*

From the above 2 images it is confirmed that the query works, and that Abu Dhabi has 3 airports.

OPTIONAL:-

To disable seeing the \_id the query would be

```
db.cities.aggregate([{$match : { name: "Abu Dhabi" } } , { $project:
{numberOfAirportsInTheCity: { $size: "$airport" },_id : 0}}]).pretty()
```

\_id: 0 disables the viewing of the object id

### QUERY-3

```
db.cities.aggregate([{$project: {cid : "$cid",name : "$name",numberOfAirportsInTheCity:
{ $size: "$airport" } } },
{ $sort: {"numberOfAirportsInTheCity":-1 } },{$limit : 10 }]).pretty()
```

**The above query goes through the cities collection and sorts(by descending order) the cities with the most number of airports**

Explanation:

```
{ $sort: {"numberOfAirportsInTheCity":-1 } },{$limit : 10 }]).
```

The -1 is to sort by descending order (1 is to sort by ascending and \$limit implies the no. of cities that should be returned are 10 as will be shown in the following 2 screenshots:

Command Prompt - mongo

```
> db.cities.aggregate([{$project: {cid: "$cid", name: "$name", numberOfAirportsInTheCity: { $size: "$airport" }} },
... { $sort: { "numberOfAirportsInTheCity": -1 } }, {$limit : 10 }]).pretty()
{
  "_id" : ObjectId("5a90a62569d81f34e40db335"),
  "cid" : 1,
  "name" : "",
  "numberOfAirportsInTheCity" : 30
}
{
  "_id" : ObjectId("5a90a62569d81f34e40db839"),
  "cid" : 1285,
  "name" : "Columbus",
  "numberOfAirportsInTheCity" : 7
}
{
  "_id" : ObjectId("5a90a62569d81f34e40dc03f"),
  "cid" : 3339,
  "name" : "London",
  "numberOfAirportsInTheCity" : 6
}
{
  "_id" : ObjectId("5a90a62569d81f34e40dbc82"),
  "cid" : 2382,
  "name" : "Houston",
  "numberOfAirportsInTheCity" : 6
}
{
  "_id" : ObjectId("5a90a62569d81f34e40dc268"),
  "cid" : 3892,
  "name" : "Moscow",
  "numberOfAirportsInTheCity" : 6
}
{
  "_id" : ObjectId("5a90a62569d81f34e40dbd2b"),
  "cid" : 2551,
  "name" : "Jacksonville",
  "numberOfAirportsInTheCity" : 5
}
{
  "_id" : ObjectId("5a90a62569d81f34e40dc68d"),
  "cid" : 4953,
  "name" : "Rio De Janeiro",
  "numberOfAirportsInTheCity" : 5
}
```

*The query showing first 7(the following 3 entries will be shown in the next screenshot)*

Due to lack of space to take all 10 I could fit the first 7 in the first screenshot. I will continue counting from the 7<sup>th</sup> city in the next screenshot (Rio De Janeiro).

```
{
  "_id" : ObjectId("5a90a62569d81f34e40dc68d"),
  "cid" : 4953,
  "name" : "Rio De Janeiro",
  "numberOfAirportsInTheCity" : 5
}
{
  "_id" : ObjectId("5a90a62569d81f34e40dbd22"),
  "cid" : 2542,
  "name" : "Izmir",
  "numberOfAirportsInTheCity" : 5
}
{
  "_id" : ObjectId("5a90a62569d81f34e40dbb77"),
  "cid" : 2115,
  "name" : "Greenville",
  "numberOfAirportsInTheCity" : 5
}
{
  "_id" : ObjectId("5a90a62569d81f34e40db4a8"),
  "cid" : 372,
  "name" : "Atlanta",
  "numberOfAirportsInTheCity" : 5
}
>
```

The above screenshots prove the query works

#### QUERY-4

```
db.cities.aggregate([  
  { $match : { $and : [{ timezone: { $gt: 0 } }, { country: "AE" }] } } ] ).pretty()
```

The above query is slightly similar to the first but its purpose and returning values are different.

It makes use of the greater than function (aggregate) of MongoDB. As mentioned in the case of the first query, there were instances where a city had the same country code but a different timezone ( it was in -5 GMT). Since UAE is in the +4 GMT the above query takes into account all positive timezones AND with the country code ae which **gives a list of all airports in the many cities of the United Arab Emirates.**

**IN SHORT THIS QUERY IS DIFFERENT FROM THE FIRST AS IT LISTS ALL AIRPORTS IN ALL CITIES OF UAE**

**(THE FIRST QUERY SHOWED A LIST OF AIRPORTS IN A CITY AND ALL AIRLINES OWNED BY THE COUNTRY THE CITY IS IN)**

```

> db.cities.aggregate([
... { $match : { $and : [{ timezone: { $gt: 0 } }, { country: "AE" } ] } } ]).pretty()
{
  "_id" : ObjectId("5a91aeb669d81f3ed4425af5"),
  "cid" : 32,
  "name" : "Abu Dhabi",
  "country" : "AE",
  "timezone" : 4,
  "tz_id" : "Asia/Dubai",
  "airport" : [
    {
      "apid" : 2179,
      "name" : "Abu Dhabi International Airport",
      "iata" : "AUH",
      "icao" : "OMAA",
      "x" : 54.651100158691406,
      "y" : 24.433000564575195,
      "elevation" : 88
    },
    {
      "apid" : 2180,
      "name" : "Bateen Airport",
      "iata" : "AZI",
      "icao" : "OMAD",
      "x" : 54.458099365234375,
      "y" : 24.428300857543945,
      "elevation" : 16
    },
    {
      "apid" : 2184,
      "name" : "Al Dhafra Air Base",
      "iata" : "",
      "icao" : "OMAM",
      "x" : 54.547698974599996,
      "y" : 24.248199462900004,
      "elevation" : 77
    }
  ]
}
{
  "_id" : ObjectId("5a91aeb669d81f3ed4425b38"),
  "cid" : 99,
  "name" : "Al Ain",
  "country" : "AE",

```

*The query showing first 7(the following 3 entries will be shown in the next screenshot)*

#### QUERY-5

```

db.countries.aggregate(
[
  { $group : { _id : "$name", airline: { $push: "$$ROOT" } } }
]
)

```

This query is used to know what airlines belong to which country.

This query makes use of the \$group aggregate function and uses the name of the countries as ids and returns all the airlines they own.

```
> db.countries.aggregate(
...   [
...     { $group : { _id : "$name", airline: { $push: "$$ROOT" } } }
...   ]
... ).pretty()
{
  "_id" : "Zimbabwe",
  "airline" : [
    {
      "_id" : ObjectId("5a91aeb669d81f3ed44275d2"),
      "code" : "ZI",
      "name" : "Zimbabwe",
      "oa_code" : "ZW",
      "dst" : "U",
      "airline" : [
        {
          "alid" : 608,
          "name" : "Air Zimbabwe",
          "iata" : "UM",
          "icao" : "AZW",
          "callsign" : "AIR ZIMBABWE",
          "alias" : "",
          "mode" : "F",
          "active" : "Y"
        },
        {
          "alid" : 1266,
          "name" : "Avient Aviation",
          "iata" : "Z3",
          "icao" : "SMD",
          "callsign" : "AVAVIA",
          "alias" : "",
          "mode" : "F",
          "active" : "Y"
        },
        {
          "alid" : 1328,
          "name" : "Air Zambezi",
          "iata" : "",
          "icao" : "TZI",
          "callsign" : "ZAMBEZI",
          "alias" : "",
          "mode" : "F",
          "active" : "N"
        }
      ]
    }
  ]
}
```

Query showing name of country as id and relevant airline information



## Code – Listing

```
import pymysql #
from pymongo import MongoClient
import copy

DB_EXISTS = True #initially was false now true because db exists after testing
display = True # Print on screen to know process being done during pipeline process
def initialise_mongo():
    return MongoClient("localhost", 27017)["test"] # MongoClient ( host,port no.(default is 27017)[dbname]

def initialise_mysql():
    #Initialize mysql with given parameters
    return pymysql.connect(
        host="localhost",
        user="root",
        password="",
        db="cw"
    )

def sql_query(sql, cursor, query_type):
    #Function that will be used to carry out queries
    if query_type == "fetchall":
        #http://pymysql.readthedocs.io/en/latest/modules/cursors.html
        #documentation was used
        cursor.execute(sql)
        return cursor.fetchall()
    else:
        print("fetchall only try again ")

#With use of a sql cursor,Extract (E from ETL)
#data from db provided
def extract_sqldata(sql_cursor):
    #http://pymysql.readthedocs.io/en/latest/user/examples.html
    #instead of individual querying i made a method for it
    cities = sql_query('select * from cities', sql_cursor, 'fetchall')
    countries = sql_query('select * from countries', sql_cursor, 'fetchall')
    airports = sql_query('select * from airports', sql_cursor, 'fetchall')
    airlines = sql_query('select * from airlines', sql_cursor, 'fetchall')
    routes = sql_query('select * from routes', sql_cursor, 'fetchall')
    data = (cities, countries, airports, airlines, routes)
    return data

#Source https://gis.stackexchange.com/questions/116655/replacing-
# null-value-with-zero-in-geodatabase-table-using-python-parser-of-arcgi
def replaceNull(x):
    if x is None:
        return ""
    elif x is "Null":
        return ""
    else:
        return x

def replaceNullInt(x):
    if x is None:
        return 0
    elif x is "Null":
```

```

        return None
    elif x is "null":
        return None
    else:
        return x

#Function for T of ETL(Transform)
#Here the transformation of the sql sqltables to relevant json schema
#as will be mentioned in report will be done
def transform_to_mongo(collection,sqltable):
    collection_test = []
    sqltable_data = { }
    if sqltable == "routes":
        for item in collection[4]:
            sqltable_data['alid'] = replaceNullInt(item[0])
            sqltable_data['src_ap'] = replaceNull(item[1])
            sqltable_data['src_apid'] = replaceNullInt(item[2])
            sqltable_data['dst_ap'] = replaceNull(item[3])
            sqltable_data['dst_apid'] = replaceNullInt(item[4])
            sqltable_data['codeshare'] = replaceNull(item[5])
            sqltable_data['stops'] = replaceNullInt(item[6])
            sqltable_data['equipment'] = replaceNull(item[7])
            sqltable_data['rid'] = replaceNullInt(item[8])
            collection_test.append(copy.copy(sqltable_data)) #https://www.python-course.eu/deep_copy.php reference
            #appeneded
        return collection_test
    elif sqltable == "cities":
        for item in collection[0]:
            sqltable_data['cid'] = replaceNullInt(item[0])
            sqltable_data['name'] = replaceNull(item[1])
            sqltable_data['country'] = replaceNull(item[2])
            sqltable_data['timezone'] = replaceNullInt(item[3])
            sqltable_data['tz_id'] = replaceNull(item[4])
            airports_collection=[]
            tempdb={}
            for airportitem in collection[2]:
                #new airport object
                if airportitem[1]==sqltable_data['cid']:
                    tempdb['apid']=replaceNullInt(airportitem[7])
                    tempdb['name']= replaceNull(airportitem[0])
                    tempdb['iata']=replaceNull(airportitem[2])
                    tempdb['icao']=replaceNull(airportitem[3])
                    tempdb['x']=replaceNullInt(airportitem[4])
                    tempdb['y'] = replaceNullInt(airportitem[5])
                    tempdb['elevation'] = replaceNull(airportitem[6])
                    airports_collection.append(copy.copy(tempdb))
            sqltable_data['airport'] = airports_collection #add as airport
            collection_test.append(copy.copy(sqltable_data))
        return collection_test
    elif sqltable == "countries":
        for item in collection[1]:
            sqltable_data['code'] = replaceNull(item[0])
            sqltable_data['name'] = replaceNull(item[1])
            sqltable_data['oa_code'] = replaceNull(item[2])
            sqltable_data['dst'] =replaceNull(item[3])
            airlines_collection=[]
            secondtest={}
            for airlineitem in collection[3]:
                if airlineitem[4]==sqltable_data['code']:
                    #matching tables
                    secondtest['alid'] = replaceNullInt(airlineitem[5])
                    secondtest['name'] = replaceNull(airlineitem[0])

```

```

        secondtest['iata'] = replaceNull(airlineitem[1])
        secondtest['icao'] = replaceNull(airlineitem[2])
        secondtest['callsign'] = replaceNull(airlineitem[3])
        secondtest['alias'] = replaceNull(airlineitem[6])
        secondtest['mode'] = replaceNull(airlineitem[7])
        secondtest['active'] = replaceNull(airlineitem[8])
        airlines_collection.append(copy.copy(secondtest))
    sqltable_data['airline'] = airlines_collection
    collection_test.append(copy.copy(sqltable_data))
    return collection_test

#Loads transformed sqldata into mongo db( L of ETL)
def load_data(mongo_collection, collection_test):
    if DB_EXISTS: #mentioned earlier flag to check if country exists
        mongo_collection.delete_many({})
    else:
        pass
    return mongo_collection.insert_many(collection_test)

def main():
    #starts pipeline
    if display:
        print('Starting automated data pipeline for ETL data from mysql to mongodb')
        print('Initialising MySQL connection')
    mysql = initialise_mysql()

    if display:
        print('MySQL connected')
        print('Starting data pipeline stage 1 : Extracting data from MySQL')
    mysql_cursor = mysql.cursor()
    mysql_data = extract_sqldata(mysql_cursor)

    if display:
        print('Stage 1 completed! Data successfully extracted from MySQL')
        print('Starting data pipeline stage 2: Transforming data from MySQL for MongoDB')
        print('Transforming routes dataset')
    routes_collection = transform_to_mongo(mysql_data, "routes")
    if display:
        print('Successfully transformed routes now cities')
        print('Transforming cities')
    cities_collection = transform_to_mongo(mysql_data, "cities")
    if display:
        print('Successfully transformed cities now countries')
        print('Transforming countries')
    countries_collection = transform_to_mongo(mysql_data, "countries")

    if display:
        print('Successfully transformed cities now init mongo')
        print('Data successfully transformed')
        print('Initialising MongoDB connection')
    mongo = initialise_mongo()

    if display:
        print('MongoDB connection successfully')
        print('Loading transformed data into mongo')
    result = load_data(mongo['routes'], routes_collection)
    if display:
        print('Successfully loaded routes now cities')
        print('Loading cities')

```

```
result = load_data(mongo['cities'], cities_collection)
if display:
    print('Successfully loaded routes now cities')
    print('Loading cities')
result = load_data(mongo['countries'], countries_collection)

if display:
    print('Stage 3 completed! Data successfully loaded')
    print('Closing MySQL connection')
mysql.close()
if display:
    print('MySQL connection closed successfully')
    print('Ending data pipeline')

main()
```

## References

- [1] DB-Engines, "DB-Engines," 24 February 2018. [Online]. Available: <https://db-engines.com/en/ranking>.
- [2] DB-Engines, "DB-Engines Ranking," Solid IT, - - -. [Online]. Available: <https://db-engines.com/en/system/MongoDB;Neo4j>. [Accessed 24 February 2018].
- [3] MongoDB, "MongoDB," MongoDB, - - -. [Online]. Available: <https://docs.mongodb.com/manual/core/master-slave/>. [Accessed 23 February 2018].
- [4] MongoDB, "JsonSchema Validation," MongoDB, - - -. [Online]. Available: <https://docs.mongodb.com/manual/core/schema-validation/>. [Accessed 18 February 2018].

