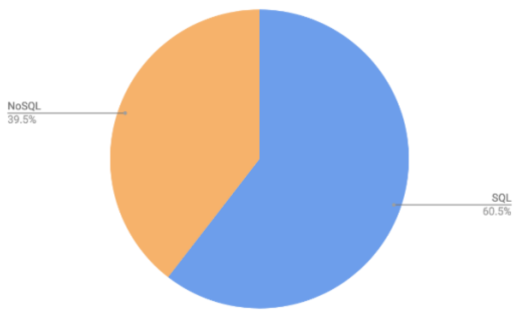
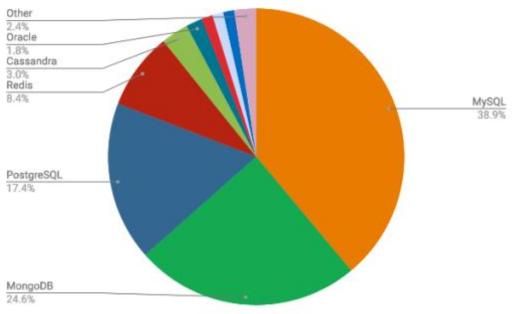
Databases Comparison Task

Introduction

SQL vs. NoSQL databases. A survey at DeveloperWeek revealed that the most popular databases in 2019 were MySQL, MongoDB, PostgreSQL and Redis. However, as we´re in an educational background I´ll just talk about open-source, free projects. The following graphs shows the market share SQL vs. NoSQL and the most used ones:





There are a whole world of databases: ElasticSeach, Hive for NoSQL, Couchbase for SQL, and neo4j or AllegroGraph for Graphs. Among many others, these are experimenting a rise in popularity in the last years. Despite this comparisons and rankings, 44.3% projects use multi-database systems to support their products, and 75.6% use SQL + NoSQL in conjunction (Source 1). These powerful database management systems can complement each other and fill the gaps in terms of data needs.

Parameters to compare

Transaction

Transactions allow to bundle multiple database updates, changes… into a single atomic operation and also give you the ability to undo an operation. But this doesn´t come for free, you need to set it up ahead of time.

- Database state: each operation alters the state of the database, moving it forward in time.
- BEGIN; COMMIT; ROLLBACK;

ACID

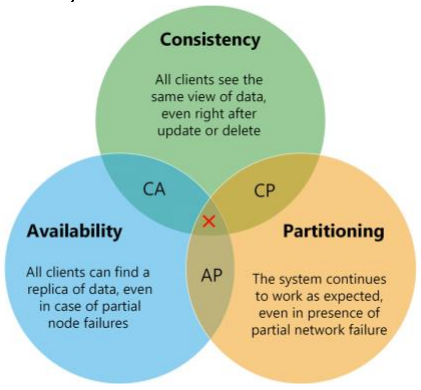
- **A**tomic → all changes to the data must be performed successfully or not at all. You never get half of an operation.
  - Reasons why operations may not work: one or more constraints violated, datatype mismatch, syntax error.
- **C**onsistent → Data must be in a consistent state (only valid data) before and after the transaction.
- **I**solated: no other process can change the data while the transactions is running.
- **D**urable: the changes made by a transaction must persist. (Source 3)

Concurrency

Concurrency refers to the simultaneous access to data, which the database must orchestrate carefully to avoid inconsistencies and the measure of how much work the database is doing at a point in time.

BASE

There is a computer science theorem that quantifies the inevitable trade-offs. Eric Brewer's CAP theorem says that if you want consistency, availability, and partition tolerance, you have to settle for two out of three (Source 4).


Consistency
All clients see the same view of data, even right after update or delete
CA
CP
×
Availability
All clients can find a replica of data, even in case of partial node failures
AP
Partitioning
The system continues to work as expected, even in presence of partial network failure

An alternative to ACID is BASE: **B**asically **A**vailable, **S**oft state, **E**ventual consistency. Rather than requiring consistency after every transaction, it is enough for the database to eventually be in a consistent state. It's harder to develop software in the fault-tolerant BASE world compared to the fastidious ACID world, but Brewer's CAP theorem says you have no choice if you want to scale up (Source 5). The **eventual consistency** is simply an acknowledgement that there is an unbounded delay in propagating a change made on one machine to all the other copies which might lead to stale data.

BASE trades consistency for availability and doesn't give any ordering guarantees at all. If the database system only supports eventual consistency, then the application will need to handle the possibility of reading stale (inconsistent) data. (Source 6)

Scalability

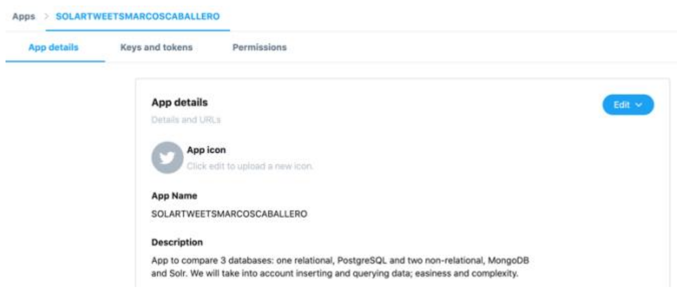
Scalability is the ability of a system to handle growing amount of work. It is achieved by two methods:

- **Vertical scaling or scaling up**: adding more resources to the existing node (CPU, memory, disk…). No matter how scalable or powerful your machine is, vertical scaling has a limit: you cannot make a single computer more powerful than a particular point. This is where horizontal scaling comes in.
- **Horizontal scaling or scaling out**: rather than making a node more powerful, you add more nodes. There is usually a linear correlation between the number of computers and the performance. Factors over vertical scaling:
    - Prices for commodity computers going down and power of commodity computers going up.
    - It works on a distributed system model. You need specialized programs that are designed to take advantage of the system.

There was a time when vertical scaling was a better option because computers were already expensive. Now commodity computers are getting cheaper and more powerful; horizontal scaling is likely to produce more value in the future. (Source 7)

Twitter API

Although not required, I will make a general explanation of how I obtained the tweets. I first applied for a twitter developer account at Twitter Developer:



App details

Details and URLs

App icon

Click edit to upload a new icon.

App Name

SOLARTWEETSMARCOSCABALLERO

Description

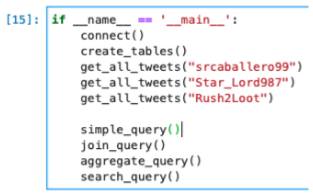
App to compare 3 databases: one relational, PostgreSQL and two non-relational, MongoDB and Solr. We will take into account inserting and querying data; easiness and complexity.

When I was granted access, I got my application keys and tokens and used this handy python script to download all of a user´s tweets into a csv, although the use cases varied depending on the database. This method only allows access to the most recent 3240 tweets. The API also has a limit of extracting 200 tweets at once. What the method does is to store them in a list until is done making a checkpoint in the id of the oldest tweet less one. There are also some other interesting rules which I discovered in source 8.

Twitter objects

Tweepy´s documentation was hard for me to read and understand. In spite this difficulty I managed to understand how twitter objects work, which resulted to be quite straightforward and convenient. I did this mostly thanks to source 8.

PostgreSQL



```
[15]: if __name__ == '__main__':
          connect()
          create_tables()
          get_all_tweets("srcaballero99")
          get_all_tweets("Star_Lord987")
          get_all_tweets("Rush2Loot")

          simple_query()
          join_query()
          aggregate_query()
          search_query()
```

Postgres + Python is an easy combination due to the psycopg2 library. This webpage provides so valuable information and tutorials about it. From there I obtained all of the functions that have to do with Postgre SQL database management.

Inserting data

However, I came across some difficulties while inserting tweets. As mentioned before, I didn´t know how to use tweepy´s provided twitter objects. Trying to insert JSONs was definitely not a good idea. The best choice finally was to just insert individually each twitter and user separately. As obvious, I had to create two tables, one with users and another with tweets. This could be considered as a disadvantage. Another clear disadvantage is that I´m not able to create a table with other kind of data that is not varchar. It prompts a type of data error every time I try to do something like that.

Queries

The error I talked before is dragged to the queries, in which I have to cast if I want to filter integers or booleans, not always obtaining the full functionality of the query that I would like. My way of working was using PgAdmin to do the queries and once they worked, translate them to python.

Conclusion

I don´t see any disadvantage aside from the three ones mentioned before. Personally, I must say that choosing Postgres for a Twitter analytics big data project is not ideally the best option as the Twitter API works mainly with JSON. The tweepy library makes your life very easy yet I would rather choose a Non-SQL database.

Rival → MariaDB

MariaDB is an open source, free, community-developed, commercially supported fork of the MySQL relational database management system under the GNU General Public License.

Features:

- Supports ACID data processing and parallel data replication.
- Implementation languages: C and C++
- Supported in the most common programming languages, including Python.
- Supports JSON, XML
- Storage engines: Azure and Navicat.
- OS support: FreeBSD, Linux, Solaris, Windows.

Pros

Open source database from the creators of MySQL, enhanced with enterprise tools and services. Optimized for performance and offering high availability, security and interoperability. High availability, scalability, performance beyond MySQL and other databases.

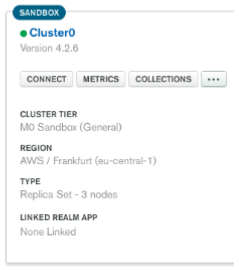
Cons

A lot of time has to be spent to get MySQL to do things that other systems do automatically, like create incremental backups. Support is available for the free version, but you'll need to pay for it.

MongoDB

MongoDB + Python is an easy combination due to the pymongo library. This webpage provides so valuable information and tutorials about it. From there I obtained all of the functions that have to do with MongoDB database management.

Inserting data



SANDBOX
Cluster0
Version 4.2.6

CONNECT METRICS COLLECTIONS ...

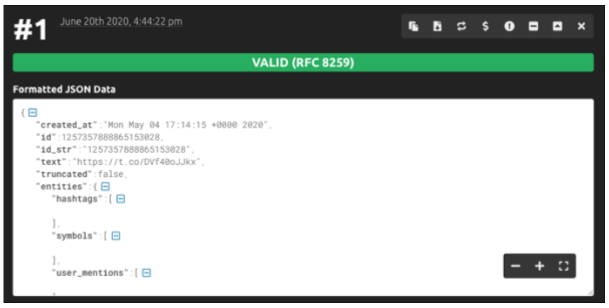
CLUSTER TIER
M0 Sandbox (General)

REGION
AWS / Frankfurt (eu-central-1)

TYPE
Replica Set - 3 nodes

LINKED REALM APP
None Linked

Before beginning to insert any data, I have to separately download MongoDB Compass (this is optional yet convenient) to work out the queries and create a cluster with MongoDB Atlas. Among AWS, Google Cloud and Azure, I chose Amazon due its high reliability and market leadership. Once my instance is created and everything regarding MongoDB is set up, I can proceed inserting tweets. I had some problems accessing my cluster as my public IP changed frequently. I solved it by granting access to anyone with correct username and password.



#1 June 20th 2020, 4:44:22 pm

VALID (RFC 8259)

Formatted JSON Data

```
{
   "created_at":"Mon May 04 17:14:15 +0000 2020",
   "id":1257357888865153028,
   "id_str":"1257357888865153028",
   "text":"https://t.co/DVf40oJJkx",
   "truncated":false,
   "entities":{
      "hashtags":[

      ],
      "symbols":[

      ],
      "user_mentions":[
```

For non-relational databases I will be choosing a JSON schema. This webpage helped me a lot while designing it. I simply copied and pasted the JSON I created with dictionaries and the Python JSON native library and saw an OK or NOT OK answer. Finally, I inserted the tweets individually with pymongo´s built in function. I didn’t have any problems inserting data.

Queries

My way of working was using MongoDB Compass to do the queries (Analyze Schema function is so good) and once they worked, translate them to python. I had no problem with data types (dates, integers, booleans…).

Join and Aggregate queries

When designing the schema I raised the question of creating one or two collections in order to do join queries. I finally created one and did the join query with an AND as it would be the easiest way. However, MongoDB offers a $lookup(aggregation) aggregation pipeline stage, which offers the same functionality as in relational databases. I did, in spite, an aggregate query, which, thanks to MongoDB´s documentation and an extra tutorial (source 9) was so easy.

Conclusion

This was by far my preferred database to handle twitter data in JSON format. It is just so simple to manage, insert and query data with MongoDB. In addition, MongoDB Atlas and Compass integrated tools and helpers make it so easy even for the learners unexperienced ones. I haven´t encountered more difficulties than the associated ones with the previous tasks.

Rival → Cassandra

Apache Cassandra is open source, free non-relational database management system designed to handle large amounts of data across many commodity servers

Features:

- Data model: wide column store; hybrid between a key-value and a tabular. Rows are organized into tables: the first component of a table's primary key is the partition key; within a partition, rows are clustered by the remaining columns of the key. Other columns may be indexed separately from the primary key.
- Data is distributed across the cluster (so each node contains different data). Data is automatically replicated to multiple nodes for fault-tolerance.
- Tables may be created, dropped, and altered at run-time without blocking updates and queries.
- Cannot do joins or subqueries (instead has collections).
- Supported in the most common programming languages, including Python.
- Related products and services: CData, DataStax Enterprise, Instaclustr and DBHawk.
- OS support: FreeBSD, Linux, Solaris, Windows.

Advantages

High availability, little to no failures. Robust support for clusters spanning multiple datacentres. Asynchronous master less replication allowing low latency operations for all clients.

Solr

Solr + Python is an easy combination due to the native python URL spport with urlib and requests libraries. There is little to no valuable documentation and information and I had to ask the teacher several time because I lacked resources to reach to my goals. The pages I´ve most used come from Solr´s webpage: a tutorial and a Python+JSON page in the client APIs section. Before starting I have to say this is the database which I most difficulties had with.

The first problem comes connecting: via terminal, I run the "./bin/solr start -e cloud" command and select the default choices. However, randomly, even if I´ve stopped previously the Solr server, the ports seem not to close correctly. I´ve run through this issue from the start of working with Solr and coped with it with this Stack Overflow resolution.

Inserting data

Inserting data is quite easy, once you´ve discovered the correct way to do it. I first tried the PySolr library but it is no longer maintained since 2015. I discovered a very nice script but I couldn´t make it work since it was for Solr 4 and in the 5th version allegedly things changed quite a lot. I desperately didn´t know what to do and tried other interesting tutorial which instead of inserting JSON, worked with tuples and encoded them with urlib. This neither worked. My last attempt was to execute shell commands with python using the os module. After trying out 3 methods with negative results, I reached the teacher for help.

He told me to use HTTP POST and GET methods with the URLs Solr generates for inserting and querying data. This has ended up as my final method of making this work (for God´s sake xd).

Queries

The only query I was not able to reproduce is the join query. Queries in Solr are a downside for me as they don´t provide very useful feedback if done wrong.

Rival → Elasticsearch

Elasticsearch is a highly scalable open source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements. Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. Elasticsearch is developed in Java and is available under the Apache 2.0 with additional free and paid features under the Elastic License.
Features:

- Open source.
- Real-time index.
- Search and analysis.
- Horizontally scalable.
- Distributed.
- Resilient.

Final conclusion

My favourite database is, with difference, MongoDB. It is the fastest (with Postgre and Solr following), the easiest to use (same order as before). Boolean and text search queries were both very easy in the three databases thanks to regular expression parsers available.

- Map-reduce and aggregate queries are possible only in Mongo and Solr.
- You can reply the SQL join queries only in MongoDB.

*Note: I have included the relational schema in the Extra folder as well as some helpful csv and json file which helped me make my mind and organise information in my head to take decisions.

*Note 2: Sources 10, 11 and 12 are quite interesting comparisons between the three databases and its rivals.

Bibliography

1. https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/
2. https://db-engines.com/en/systems (Applies for all of the Rival sections)
3. https://www.youtube.com/watch?v=5Pia4UFuMKo
4. https://www.youtube.com/watch?v=kR8yvxZ2nqU
5. https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
6. https://ivoroshilin.wordpress.com/2012/12/13/brewers-cap-theorem-explained-base-versus-acid/
7. https://www.youtube.com/watch?v=wte3dmk8fmc
8. https://towardsdatascience.com/twitter-data-collection-tutorial-using-python-3267d7cfa93e
9. https://cloudnweb.dev/2019/11/learn-mongodb-aggregation-with-real-world-example/
10. https://db-engines.com/en/system/Elasticsearch%3BSolr
11. https://db-engines.com/en/system/Cassandra%3BMongoDB
12. https://db-engines.com/en/system/MariaDB%3BPostgreSQL