# TASK 3
# Final Task

**Álvaro Tomás Lozano**

# INTRODUCTION NOTES

This practice is divided into the Theory Part which will come first and after the Practical Part. The Bibliography is based on references that I've taken from the internet for the Practical and Theory Part. Each section of the bibliography takes the same number for links about the same subject and the pages where I've done the research.

# THEORETICAL PART

## "Find 3 different databases, similar to MongoDB, PostgreSql and SOLR" and compare them.

PostgreSQL works with SQL statements. A database that we used previously in other subjects was Oracle Database. However these two while they are similar in their way to work for the user, have a couple of differences which are noted in the inside.
Oracle DB is a commercial relational database management system while PostgreSQL is made open-source. It was implemented with C and C++ and has secondary database models like the document store, key-value store, RDF store, and graph DBMS whereas PostgreSQL has secondary database models like Document store and key-value store models. It's remarkable that Oracle supports a lot more languages than PostgreSQL and in the ranking of DB-Engines, Oracle is ranked as the number #1 and PostgreSQL the number #4. **[2]**

An alternative to MongoDB is DynamoDB which is the flagship NoSQL database on AWS with its AWS platform cost, speed, and reliability. A big advantage of DynamoDB for someone who is using Mongo is that you can migrate the information from MongoDB straight to DynamoDB. The best part is there is no infrastructure to manage as Amazon AWS is the one who manages them.**[1]**
DynamoDB uses tables, items and attributes as the core components that you work with. A table is a collection of items, and each item is a collection of attributes. It uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility. While MongoDB uses JSON-like documents to store schema-free data. In MongoDB, collections of documents do not require a predefined structure and columns can vary for different documents.**[1]**
MongoDB has many features of a relational database, including an expressive query language and strong consistency. Since it is schema-free, MongoDB allows you to create documents without having to create the structure for the document first.**[1]**

For Solr I have found a similar one which is Elasticsearch. These ones are quite similar because they are used more as a text search engine rather than a database but Elasticsearch is also a Document-oriented database (No-SQL).

When it comes to choosing Solr vs Elasticsearch, each of them has its own set of strengths and weaknesses, so there's no right or wrong. Although it seems that Elasticsearch is more popular, each may be a better or worse fit depending on your needs and expectations.
Elasticsearch is able to index rapidly changing data almost instantly (in less than 1 sec). It is appropriate to use it in projects where a database is constantly updating.**[3]**
When the database grows, it becomes more difficult to look up. But Elasticsearch scales up while your DB gets bigger, so the search speed does not slow down.**[3]**
It can be used also as a data storage. Nevertheless, isn't recommend using it as a primary storage. It's better to keep data in a main DB for better security and reliability, and use Elasticsearch only to index data and store logs, but the option to do it is there.
In case of the searches, Solr is much more oriented towards text search while Elasticsearch is often used for analytical querying, filtering, and grouping. The team behind Elasticsearch is always trying to make these queries more efficient (through methods including the lowering of memory footprint and CPU usage) and improve performance at both the Lucene and Elasticsearch levels. When comparing both, it's clear that Elasticsearch is a better choice for applications that require not only text search but also complex time series search and aggregations.**[3]**

## **"Compare the 3 databases used pros and cons in your experience"**

In this task we have been said to use three different databases. One which is PostgreSQL, an ER-based database which uses SQL, MongoDB which is a No-SQL type of database and it uses collections that behave something similar to tables in ER databases, with "documents" to store information in a JSON format, and Solr which can be used as a text search but also as a database storing information in collections too and documents.

In the PostgreSQL implementation I have used it as I could use Oracle Database. It works excellent as a entity relationship database which works with SQL. The server interface is pretty clear and the query interface is useful. You can create a database easily in the pgAdmin screen and manage there your database. The only thing I really missed out is something like a graphic schema from the database you have. It would have been really nice to have a tool like that and get the schema directly from Postgres but I'm fine with that. The pros are that it works really fast inserting data and doing queries, and having to do that on SQL is a pro for me, as other sintaxes like the MongoDB and Solr were really strange. With SQL syntax you can perform queries easier than in Mongo, as you need to read the SPECIFIC documentation of Mongo about how to do queries there, and if you know SQL you can perform queries in PostgreSQL, Oracle, and many more databases. About doing the queries in PostgreSQL it has taken me much less time to implement them in PHP rather than the ones with Mongo and Solr, as sometimes the documentation of Mongo and Solr was deprecated and you had to install things with a package manager as Composer. Performing the queries is faster on PostgreSQL and the data is much clearer for the person who manages it as it uses an ER system.

On the MongoDB one, I have had to learn about how to implement the MongoDB database in PHP and how to connect to it. I had to download things and gave me some problems with

PHP but overall it wasn't as harder as Solr. It's database management with MongoDB Compass is great. The way you connect to Mongo through the interface and you can see all your documents in your collection is much simpler than Solr and PostgreSQL. Also you can set up a Cluster in the cloud and use it as a remote database which can be accessed by other users and Mongo will not charge you for using that service which is also really nice at the moment you are thinking to deploy an application or software. However I've seen that MongoDB works better having a few collections in your database. Performing queries isn't hard but it may be costly in efficient terms. In the case of aggregations you have a lot of options, like match, unwind, bucket, or lookup which can be used as a join in Mongo. These queries are really powerful but again they are expensive performing them. Note that MongoDB works much better with JavaScript and Node.js as I have researched but PHP was the option I've chosen for doing this practice so I can't complain about that. Also the MapReduce function required to have a $map and $reduce function which has to be written in JavaScript too, but the query of MapReduce I think it's pretty useful.

Finally, in Solr I have felt really overwhelmed about it. It has been by far the most expensive (in time) to make it working. I was working with their schemaless functionality and adding directly the information in arrays with a field => value specification but sometimes Solr expected a value for a field to be on a specific type of data, and then it couldn't be parsed in the Java core to what Solr expected and it returned an error and nothing were inserted. This made me headaches but I managed to solve that parsing almost everything to a string and then sending it to Solr. The query interface in the webpage manager is simple but a little bit confusing. Once you manage to a select query the syntax is really strange at least at first impression and you have to research about how to do queries as it doesn't resemble to anything like SQL and the Mongo syntax. It also lacks of having aggregates or joins in my case, as I haven't worked with different collections and was schemaless, and I haven't found a way to use a MapReduce query as it seems to don't have that functionality.

## **"In what ways can I scale the three solutions"**

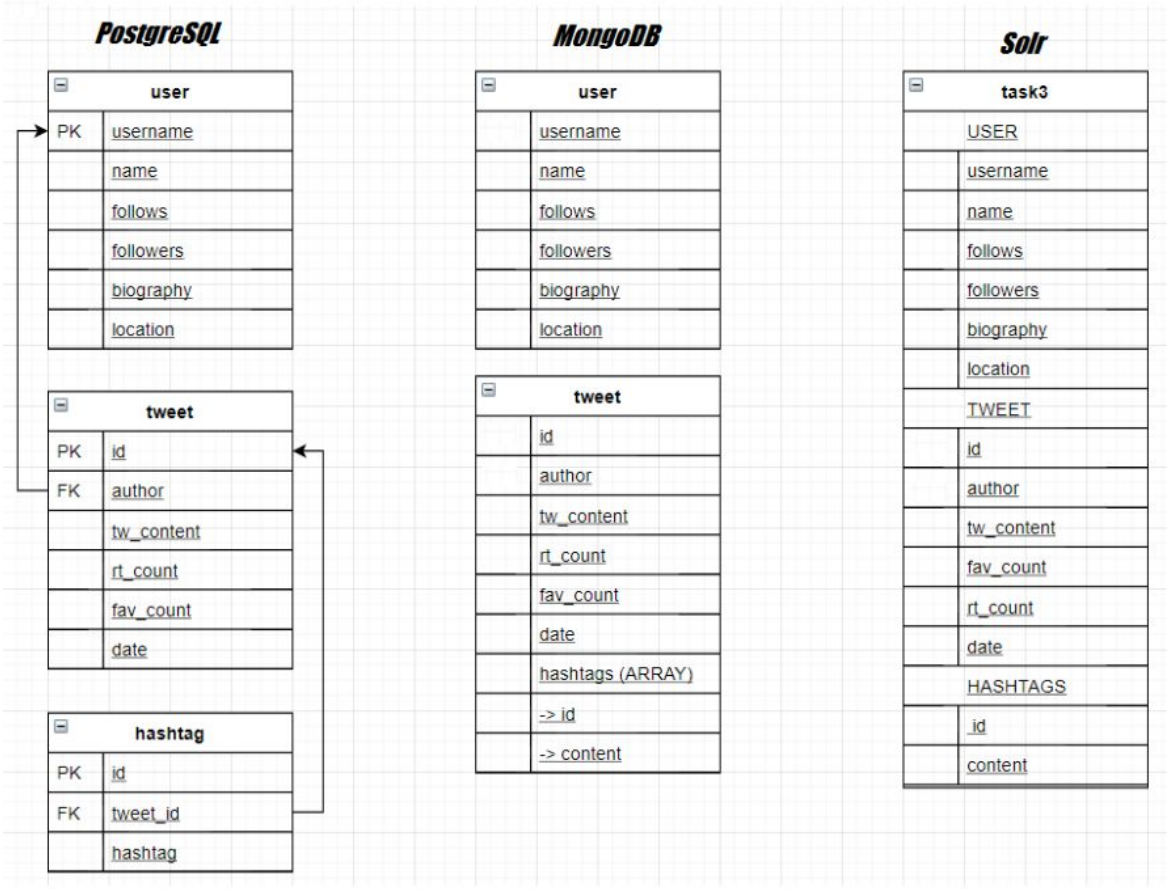
PostgreSQL can be escalated in different ways depending of your problem, which are the Vertical Scaling or Horizontal Scaling. Maybe you are suffering from a high workload from the reading or you are suffering it from the writing. In the case of the writing the solution could be spliting our database into multiple servers in order to handle the data that is being inserted. Dividing the tables in separated servers and redirecting the queries to them could help a lot to reduce that workload, and if you have a reading problem, you need to add replicas to your database balancing the queries in a round robin mode.**[4]**

MongoDB uses a sharding technique, more like an auto sharding, which divide the collection into "shards", so you can divide the documents into a wider manageable space, more than a server or service could handle, and this is done automatically as it takes more shards depending on the workload you have in your documents. This can also be made just by expanding the memory you have but as we know, memory is limited, and at sometime you will run out of memory, so why not divide it by shards?**[5]**

Also in Solr, the escalation is made by sharding. When you execute a Solr session it divides the information in shards which treat the data the same way that Mongo does. The queries that come to the server are distributed into each shard and then the results are combined into a single result. Solr has a specific way to do this with the replication factor of a collection, which improves a lot the performance adding servers and duplicating the collections, so you can handle a lot of queries at the same time and distribute them to the servers.**[6]**

# PRACTICAL PART

For doing this practice we need first to do a schema about how our databases will be. I wanted to keep the same schema and keys for every database, but the only one following an ER model is PostgreSQL.
For PostgreSQL I have 3 tables which two of them have foreign keys which references the other tables. In my case I found relevant to get the id of the tweet that has put the hashtag in the hashtag table and link it with the tweet id from the tweet table, and for the tweet table, linking the author to the same username in the user table.
For MongoDB I've chosen to divide the information in two collections, and the hashtags are stored into an array in the tweet collection.
For Solr the way to go for me was to use a schemaless model for the collection and store all the info there without worrying about tables.



For implementing the part of MongoDB and Solr in PHP we need to install Composer. Composer is a package manager for PHP which will allow us to use libraries in our PHP projects.
Having these libraries will ease the development and coding from Solr and MongoDB.

```
C:\xampp\htdocs\db-finaltask>composer require "mongodb/mongodb=^1.0.0"
./composer.json has been created
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Installing mongodb/mongodb (1.6.0): Downloading (100%)
Writing lock file
Generating autoload files

C:\xampp\htdocs\db-finaltask>composer require "solarium/solarium"
Using version ^5.2 for solarium/solarium
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 9 installs, 0 updates, 0 removals
  - Installing symfony/polyfill-php80 (v1.17.0): Downloading (100%)
  - Installing psr/event-dispatcher (1.0.0): Downloading (100%)
  - Installing symfony/event-dispatcher-contracts (v2.1.2): Downloading (100%)
  - Installing symfony/deprecation-contracts (v2.1.2): Downloading (100%)
  - Installing symfony/event-dispatcher (v5.1.2): Downloading (100%)
  - Installing psr/http-message (1.0.1): Downloading (100%)
  - Installing psr/http-factory (1.0.1): Downloading (100%)
  - Installing psr/http-client (1.0.0): Downloading (100%)
  - Installing solarium/solarium (5.2.0): Downloading (100%)
symfony/event-dispatcher suggests installing symfony/dependency-injection
symfony/event-dispatcher suggests installing symfony/http-kernel
solarium/solarium suggests installing minimalcode/search (Query builder compatible with Solarium, allows simplified solr-query handling)
Writing lock file
Generating autoload files
4 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
```

- **PostgreSQL Configuration**

In order to use PostgreSQL in php you have to modify the php.ini file in the xampp folder and add some lines to it.

```
extension=exif      ; Must be after mbstring as it depends on it
extension=php_mongodb.dll
extension=mysqli
;extension=oci8_12c  ; Use with Oracle Database 12c Instant Client
;extension=odbc
;extension=openssl
;extension=pdo_firebird
extension=pdo_mysql
;extension=pdo_oci
;extension=pdo_odbc
;extension=pdo_pgsql
extension=pdo_sqlite
extension=pgsql
;extension=shmop
```

This also is needed to work with the MongoDB library.

I have decided to work with different .php files and on each one I have a class defined by the database that's based on.

*pg_php* is the class for my PostgreSQL implementation in PHP

```php
public function __construct()
{
    $this->queries = new queries();
    $this->postgre = pg_connect("host=localhost dbname=twitter user=postgres password=admin") or die('No se ha podi
}
```

In order to connect the database I need to use the pg_connect command with my credentials. If everything it's okay it will connect without problems.

- **MongoDB Configuration**

For Mongo it's almost the same as PostgreSQL, you have to use your credentials in the constructor to create a connection but I expanded a little bit more the class. Also you will need to import the autoload.php for the libraries that code is gonna use. **[9]**

```php
require_once("vendor/autoload.php");
```

```php
private $mongo;
private $queries;
private $m_db;          //Database
private $m_col_tweet;   //Tweet Collection
private $m_col_user;    //User Collection
private $m_cur;         //Cursor

public function __construct()
{
    //CLOUD - $this->mongo = new MongoDB\Client('mongodb+srv://AbsydeAuberon:<password>@auberoncluster-ywak9.mongodb.net/test?authSource=adm
    //LOCALHOST -
    $this->mongo = new MongoDB\Client('mongodb://localhost:27017/?readPreference=primary&appname=MongoDB%20Compass%20Community&ssl=false');
    $this->queries = new queries();
    $this->m_db = $this->mongo->twitter;    //Database name:   "twitter".
    $this->m_col_tweet = $this->m_db->tweet; //Collection name: "tweet".
    $this->m_col_user = $this->m_db->user;   //Collection name: "user".

}
```

As I want to access all of my collections and database easily, I store them into variables for later uses in the calls to the query functions and insert functions. And when I initialize the MongoDB class in a variable, after that I have to initialize the databases.

```php
public function initialize_databases()
{
    $this->m_db = $this->mongo->twitter;
    $this->m_db->dropCollection("tweet");
    $this->m_db->dropCollection("user");
    $this->m_db->createCollection("tweet");
    $this->m_db->createCollection("user");
    $this->m_col_tweet = $this->m_db->tweet; //Collection name: "tweet".
    $this->m_col_user = $this->m_db->user;   //Collection name: "user".
    $this->m_col_tweet->createIndex(["$**" => "text"]); //Wildcard text index for the tweet collection in order to do text searches.
}
```

This will drop any previous information each time you are going to use again the web interface but later on I'll explain more about it. Note that the tweet collection has to have a wildcard text index in order to do search queries, which is something that MongoDB requires to you. You can specify the index but I preferred to let it to get any type of string and make it an index in order to have a better working search. **[9]**

- **Solr Configuration**

Solr gets a little more tricky. You need to use a library called Solarium which doesn't have a very well explained documentation and sometimes you have to go through people in stackoverflow in order to find someone who has used that library and have encountered the same error.
Also when executing the *solr start -e cloud* command in the CMD, you need to select the collection as "task3". **[14]**

For configuring it you have to set a config variable that you are going to pass to the connection.

```
private $solr;
private $cfg;

public function __construct()
{
    $this->cfg = array(
        "endpoint" => array(
            "localhost" => array(
                "host"=>"127.0.0.1",
                "port"=>"8983",
                "path"=>"/",
                "collection"=>"task3")));

    $this->solr = new Solarium\Client(new Solarium\Core\Client\Adapter\Curl(),
    new Symfony\Component\EventDispatcher\EventDispatcher(), $this->cfg);
}
```

You need to specify the port of the shard that you are using, the path and the collection that you are going to use. **[14]**

Now after having all the databases created and connected we need to fetch the tweets from the Twitter API.

- **Getting Tweets from Twitter API**

For getting tweets from Twitter we need to access the API. We have to have some special permissions for our account. As I requested the access but haven't been able to get my account validated so I've used the tokens you have given to me. **[16]**

```php
public function setConfig($username)
{
    /** Set access tokens here - see: https://dev.twitter.com/apps/ **/
    $settings = array(
    'oauth_access_token' => "25149590-QHAOa0MdNeWUW5LmQ2m60WtK53lCc91t9oKhHbgGd",
    'oauth_access_token_secret' => "8l3kJJdsEw6zuxdkz3JdMwNNNgpfQoOJaaEOwEFEOq4aa",
    'consumer_key' => "CWezp582B6fDkN4DpQnLBg",
    'consumer_secret' => "auW27yBE6k8pWcs5G8W81fzwZ7A4WwSTQmf15wiqty0"
    );

    $url = "https://api.twitter.com/1.1/statuses/user_timeline.json";
    $requestMethod = "GET";

    if (isset($_GET['user']))  {$user = preg_replace("/[^A-Za-z0-9_]/", '', $_GET['user']);}
    else {$user  = $username;}

    if (isset($_GET['count']) && is_numeric($_GET['count']))
    {
        $count =$_GET['count'];
    }
    else {$count = 10;}

    $getfield = "?screen_name=$user&count=$count&tweet_mode=extended";
    $this->twitter = new TwitterAPIExchange($settings);
    $this->feed = json_decode($this->twitter->setGetfield($getfield)
    ->buildOauth($url, $requestMethod)
    ->performRequest(),$assoc = TRUE);
    if(array_key_exists("errors", $this->feed)) {echo "<h3>Sorry, there was a problem.</h3><p>Twitter returned the following error message:</
}
```

You set a username from which you want to get the tweets and also you set the number of tweets you want to fetch from the API. After it has connected to the API, it will perform the request to the API and if you have all the required permissions, it will retrieve the tweets into an array, in this case, the variable $feed.

- **Inserting the Tweets**

After getting the tweets in a variable, it's time to insert them.
You have to have the databases initialized too with some functions that I've written in the *twitter_php* class, as it's easier to manage for me when coding.

```php
public function  __construct()
{
    //PostgreSQL
    $this->pg = new pg_php();
    //Mongo
    $this->mongo = new mongo_php();
    //Solr
    $this->solr = new solr_php();
}
```

After I've setted all the variables as the classes for my databases and created all the necessary collections or tables, it's time to insert the data.

I have created a class called *queries* in the queries.php file where I store all the needed queries for later use, like for example the insert, select queries for PostgreSQL, or the aggregate pipelines for MongoDB.

For inserting the data in PostgreSQL I have to call to an Insert query for each table that I have.

```php
public function pg_user_insert($username, $name, $follows, $followers, $bio, $location)
{
    $query = "INSERT INTO \"user\"
    (username, name, follows, followers, biography, location)
    VALUES('$username', '$name', $follows, $followers, '$bio', '$location');";

    return $query;
}

public function pg_tweet_insert($id, $author, $content, $rt_count, $fav_count, $date)
{
    $query = "INSERT INTO \"tweet\"
    (id, author, tw_content, rt_count, fav_count, date)
    VALUES($id, '$author', '$content', '$rt_count', '$fav_count','$date');";

    return $query;
}

public function pg_hashtag_insert($tweet_id, $hashtag)
{
    $query = "INSERT INTO \"hashtag\"
    (tweet_id, hashtag)
    VALUES($tweet_id, '$hashtag');";

    return $query;
}
```

In the Twitter class I have a function called insertData() which is functionality is to pass all the information I get from the tweets, go through each one of them and classify the information into a "user", a "tweet" and in case there are hashtags, into a "hashtag" structure. All of that is inserting the data into the databases while I classify the structures. After that function has executed I should have all the fetched tweets into my databases ordered by the tables or collections needed.

In case of PostgreSQL the data is inserted into the user, tweet and hashtag table. Each of them have a set of fields as the insert queries show. For this I haven't found any big problem, but the only one I found was the escape of the strings, as the name of the table has to be put with double comma and in PHP you need to put the slash double comma in order to get that recognized inside the string.

Then for inserting the data with MongoDB it's a little more complicated, as Mongo doesn't have a native integration with PHP and I have had to use a library in order to implement it.

```php
public function insert_user($user)
{
    $this->m_col_user->insertOne(array("username" => $user[0],"name" => $user[1],"follows" => $user[2],"followers" => $user[3],
    "biography" => $user[4],"location" => $user[5]));
}

public function insert_tweet($tweet, $hashtags)
{
    $this->m_col_tweet->insertOne(array("id" => $tweet[0],"author" => $tweet[1],"content" => $tweet[2],
    "rt_count" => $tweet[3],"fav_count" => $tweet[4],"date" => $tweet[5], "hashtags" => $hashtags));
}
```

For inserting the data, you only have to worry about in which collection are you gonna store it and you have to set a key => value for the field and value in the document. This is in my opinion a much easier way to store information rather than with the PostgreSQL queries and Solr implementation, but in the other hand, PostgreSQL give you a much more solid way to do this things with the SQL language.

And finally for inserting the data in Solr, you need to pass to Solr the data you are going to store in form of documents, similar to mongo.

```
public function insert_data($content)
{
    $update = $this->solr->createUpdate();
    $res = $this->solr->update($update);

    foreach($content as $array)
    {
        $update_query = $this->solr->createUpdate();
        $doc = $update->createDocument();

        foreach($array as $field => $value)
        {
            $doc->$field = $value;
        }
        $update_query->addDocument($doc);
        $update_query->addCommit();
        $res = $this->solr->update($update_query);
    }

}
```

In my case, as I'm working with a schemaless model, the way I did it is passing to the insert function an array with three different arrays. One of them is the array of the user, other is the array of tweets and the final one is the array of hashtags. These ones will be iterated through a foreach loop and will be setted into a document, filling it with a field and a value for that field, and after that it will add the document and commit it to the database.
This lead to me some problems, as the schemaless work a little bit by its own and sometimes it may generate problems converting the data to the required type that the document expects, and for example I suffered from one error which tried to convert a string to a long in java and it created an exception each time I tried to submit the data into the database, so my way to fix that was converting everything into a string and then push it so it should have any problem converting it.

Now about the concurrency, transactions, ACID and BASE. **[7][8][13]**
In our three databases I've found that they support concurrency as they allow multiple clients to read and write data. PostgreSQL and MongoDB are the ones which works better with transactions in order to support atomicity of reads and writes into multiple tables or documents, and in case of MongoDB, it can be used even across shards. For Solr I haven't found anything from the core, just some extensions in order to make it viable.
About BASE and ACID, the ACID term means atomicity, consistency, isolation and durability and the BASE term means Basically Available, Soft state, Eventual consistency. BASE gives up in consistency as it's only eventual and it fits better the NoSQL type of databases as it could be MongoDB and Sorl, and ACID is the set of properties of database transactions intended to guarantee validity, so it's the one that could fit a SQL model which is stronger and more solid, like PostgreSQL. **[15][17]**

The collections and tables get cleared and remade each time you execute for the first time the application, so some query results may vary.

- **Query the Tweets**

Now that we have everything in our databases, it's time to try to retrieve the information in order to use it.
We have been told to perform five kind of queries with the three databases and compare them.

First we would have the simple queries. A simple query could be performed as something like "retrieve me the tweets from this table and from this author", which is the way I tried to represent them.

In our main page of the PHP application I've developed we would find a pretty simple html doc, with some links. For the examples, I have taken tweets from the Real Madrid account, the jack account (creator of twitter) and the twitter account itself which put the most recent updates.

Below you will find the way the webpage looks by itself.

# POSTGRESQL

Simple Query

Join Query

Aggregate Query

Map Reduce Query

Text Search Query

# MONGODB

Simple Query

Join Query

Aggregate Query

Map Reduce Query

Text Search Query

# SOLR

Simple Query

~~Join Query~~

~~Aggregate Query~~

~~Map Reduce Query~~

Text Search Query

- **Simple Queries**

Now that we are analyzing the simple queries, I'll stract a little show of what you get from these queries and explain them.

**PostgreSQL Simple Query searching only tweets from realmadrid:**

author => realmadrid
tw_content => 🎯 ¡Preparados para visitar a la @RealSociedad! #RMCity | #RMLiga https://t.co/2j1Aju10Lt
rt_count => 147
fav_count => 1805
date => Sat Jun 20 16:00:01 +0000 2020

author => realmadrid
tw_content => 🥅👉😃👈 ¿Quieres la camiseta de @marcoasensio10 en su soñado regreso? 🙋 ¡Ya puedes pujar por las camisetas
#RealMadrid
rt_count => 112
fav_count => 1784
date => Sat Jun 20 15:00:01 +0000 2020

```php
public function simple_query_examples()
{
    echo '<h3>PostgreSQL Simple Query searching only tweets from realmadrid: </h3><br>';
    $query = $this->queries->pg_sq();
    $result = pg_query($query) or die('La consulta fallo: ' . pg_last_error());
    $this->show_data($result);
}
```

```php
public function pg_sq()
{
    $query = "SELECT author, tw_content, rt_count, fav_count, date FROM \"tweet\"
        WHERE author = 'realmadrid'";

    return $query;
}
```

The PostgreSQL simple query gets the tweets from the table tweet where the author is the account of Real Madrid. We can achieve this query using SQL as is the language PostgreSQL uses for doing queries.

**MongoDB Simple Query searching only tweets from jack:**

_id=> 5eee3ed774760000f1004fa3
id=> 1274375523741732865
author=> jack
content=> RT @TIDAL: Listen to @Beyonce - "BLACK PARADE" on TIDAL now. ▶️: https://t.co/PMjJFSFJaI https://t.co/c2mKnXlQhY
rt_count=> 5420
fav_count=> 0
date=> Sat Jun 20 16:16:16 +0000 2020
hashtags=>

_id=> 5eee3ed774760000f1004fa4
id=> 1274092610911563776
author=> jack
content=> RT @Twitter: 📍 Minneapolis 🎥 @FredTJoseph https://t.co/lNTOkyguG1
rt_count=> 4749
fav_count=> 0
date=> Fri Jun 19 21:32:04 +0000 2020
hashtags=>

```php
public function mongo_sq($select)
{

    switch($select)
    {
        case 1:
            //Query by author in Tweet Collection
            $query = ['author' => 'realmadrid'];
            return $query;
            break;

        case 2:
            //Query by author in Tweet Collection
            $query = ['author' => 'jack'];
            return $query;
            break;

        case 3:
            //Query by username in User Collection
            $query = ['username' => 'Twitter'];
            return $query;
            break;
    }
}
```

For MongoDB, the simple query is just to pass a query to the collection with the command find() and then use the setTypeMap **[10]** in the cursor that has be returned in the query in order to filter the information as arrays, otherwise it will be returned as an unreadable cursor from MongoDB. After that you can iterate it through a foreach loop and return the information into something readable and that can be introduced as an html element. The query itself it's pretty explanatory as it should be an equivalent in SQL as WHERE x = y. In my opinion it's easy to do queries in Mongo if you don't know SQL but I prefer the SQL method as it feels more consistent and you know why the query doesn't work if there's any error.

```php
public function simple_query_examples()
{
    echo '<h3>MongoDB Simple Query searching only tweets from jack: </h3><br>';
    $query = $this->queries->mongo_sq(2);
    $cursor = $this->m_col_tweet->find($query);
    $cursor->setTypeMap(['root' => 'array', 'document' => 'array', 'array' => 'array']);
    $this->show_data($cursor);
}
```

**Solr Simple Query searching for tweets with 100 or more RTs**

tweet_id: 0: 1274013096667840512
author: 0: jack
content: 0: RT @Lupita_Nyongo: 1. This is Opal Lee, the force behind the movement to make #Juneteenth a national holiday. Also known as Freedom Day, it…
rt_count: 0: 9424
fav_count: 0: 0
date: 0: Fri Jun 19 16:16:06 +0000 2020
id: 25969cc6-620f-42b5-977c-aff6647e8dcf
_version_: 1669974359846944768
score: 1

tweet_id: 0: 1274034244700930049
author: 0: Twitter
content: 0: RT @Blackbirds: Juneteenth is a celebration. It's about our freedom. And within that freedom is our joy. #BlackJoy is a form of resistance…
rt_count: 0: 9182
fav_count: 0: 0
date: 0: Fri Jun 19 17:40:09 +0000 2020
id: 44985062-c2c4-4ded-aa82-f2f0bed99d96
_version_: 1669974364548759552
score: 1

```php
public function simple_query_examples()
{
    echo '<h3>Solr Simple Query searching for tweets with 100 or more RTs</h3><br>';
    $query = $this->solr->createSelect();
    $query->setQuery('rt_count:[100 TO *]');
    $query->setStart(2)->setRows(10);
    $query->addSort('rt_count', $query::SORT_DESC);
    $result = $this->solr->select($query);

    $this->show_data($result);

}
```

For Solr, the simple query is to create a Select query from the database in order to set the parameters. You can set multiple things like sorting in the query by a field you want, or having a field selection through the ones you are searching. Also you can specify too the amount of rows you want to get from the query and after doing it, you get the result which can be printed easily in the HTML.

- **Join Queries**

For the Join Queries we can only analyze the PostgreSQL and MongoDB, at least in my case, as in Solr using a schemaless model with only one collection makes impossible to do a join query, it wouldn't make any sense. Join queries mix two tables in order to get a result that uses the two tables as it could be in my case to get the values of the user who has tweeted some tweets.

**PostgreSQL Join Query getting tweets from users registered in the user collection and showing its data:**

username => jack
name => jack
follows => 4473
followers => 4732168
tw_content => RT @TwitterDesign: Being Black in UX — let's discuss what that means in 2020. Join us next Thursday, 6/25, at 2pm PT for a virtual panel w…
date => Fri Jun 19 05:14:45 +0000 2020

username => jack
name => jack
follows => 4473
followers => 4732168
tw_content => RT @Twitter: Today is #Juneteenth  We're honored to have @opalayo, one of the three Black women who co-founded #BlackLivesMatter, speak on…
date => Fri Jun 19 16:05:03 +0000 2020

username => jack
name => jack
follows => 4473
followers => 4732168
tw_content => RT @temi_coker: I am a Nigerian-American Artist based in Dallas Texas. My work is a mix of vibrant colors, textures,& a love for evoking em…
date => Fri Jun 19 16:06:10 +0000 2020

```
public function pg_join()
{
    $query = "SELECT \"user\".username, name, follows, followers, \"tweet\".tw_content, \"tweet\".date
        FROM \"user\"
        INNER JOIN \"tweet\" ON \"tweet\".author = \"user\".username;";

    return $query;
}
```

With PostgreSQL it's as easy as doing an Inner Join of the two tables and comparing the author and username in the two tables in order to get the results that we want.
Here I've wanted to selected the user information and show the tweets from him and the date they were posted.

**MongoDB Join Query using aggregation with Slookup searching tweets which have the same author and username from jack and joining the information:**

_id=> 5eee3ed774760000f1004fa3
id=> 1274375523741732865
author=> jack
content=> RT @TIDAL: Listen to @Beyonce - "BLACK PARADE" on TIDAL now. ▶️: https://t.co/PMjJFSFJaI https://t.co/c2mKnXlQhY
rt_count=> 5420
fav_count=> 0
date=> Sat Jun 20 16:16:16 +0000 2020
hashtags=>
user=>
0=>
_id=> 5eee3ed774760000f1004fa2
username=> jack
name=> jack
follows=> 4473
followers=> 4732168
biography=> #bitcoin
location=>

```php
case 2:
    //Pipeline to search tweets from the collection user which are the same author.
    $pipeline = array(
        array('$match' => ['author' =>'jack']) ,
        array('$lookup'=>[
            'from'=>'user',
            'localField'=>'author',
            'foreignField'=>'username',
            'as'=>'user']));
    return $pipeline;
    break;
```

For Mongo the query is made the same way before but you will call the function aggregate() instead of find() and you have to pass a specific configuration called pipeline. In the end, the way to do a Join Query in Mongo is through an aggregation.
You have to use the $lookup setting and set the localField and foreignField as it may resemble to a Foreign Key in Entity Relationship models. In this query I merge the user and tweet collection for the tweets of jack and I output them in the same object.

- **Aggregate Queries**

Aggregation is a way to compute values and join them in a result, most likely a Join, but in Mongo it's a more powerful tool as it gets many options to manage the documents. In PostgreSQL you can use the aggregate functions which are a way to perform aggregations in values like the max() or sum() functions which will return a max value or sum the values. As for Solr, you don't do a real aggregation using the schema less model so in the end I've skipped that part too as I haven't been able to do it due to how my database design is made. **[11]**

**PostgreSQL Aggregate Query searching for the tweet with most RT in each account:**

author => jack
max => 10758

author => Twitter
max => 16846

author => realmadrid
max => 2743

```
public function pg_aggregate()
{
    $query = "SELECT author, max(CAST(rt_count AS INTEGER))
        FROM \"tweet\"
        GROUP BY author HAVING max(CAST(rt_count AS INTEGER))>1000;";

    return $query;
}
```

Here in PostgreSQL the aggregation is made asking for the tweet with the max number of retweets from each user that also is more than 1000 RTs so if some user would have that number of RT in any of his tweets, it wouldn't appear in the query.

**MongoDB Aggregate Query searching only tweets with hashtags with Sunwind:**

```
_id=> 5eee3ed774760000f1004fa6
id=> 1274030568217305094
author=> jack
content=> RT @BerniceKing: Don't stop. #Juneteenth https://t.co/CjLj62R8Lr
rt_count=> 421
fav_count=> 0
date=> Fri Jun 19 17:25:32 +0000 2020
hashtags=>
0=> 1274030568217305094
1=> Juneteenth
```

```
case 1:
    //Pipeline for splitting the hashtags array.
    $pipeline = array(
        array('$unwind' => '$hashtags')
        );
    return $pipeline;
    break;
```

And as for MongoDB, I wanted to use the $unwind aggregation type as I had the hashtag field as an array and the $unwind function will deconstruct the array and return me as something readable that I can show in the html interface. This aggregation query seemed relevant for me and that's why I've chosen it.

- **Map Reduce Queries**

The Map Reduce is a tool that has a lot of relevance in MongoDB and it's used a lot, but it can be translated too to PostgreSQL but in Solr there's no possible way to do a Map Reduce query as it doesn't have the functionality in its database part. Map Reduce what it does is to get two functions that are the map and the reduce and then use both together to get a result that suites what you want. The map function is defined to map the values and fields

to what you want to get from the query and the reduce function will perform operations to that values in order to reduce them into less data. It's computational cost is higher but the results it can get are unique.**[12]**

**PostgreSQL Map Reduce getting the sum of RT from every user tweets:**

author => jack
sum => 26681

author => Twitter
sum => 26621

author => realmadrid
sum => 5237

```
public function pg_map_reduce()
{
    $query = "SELECT author, SUM(CAST(rt_count AS INTEGER))
        FROM \"tweet\"
        GROUP BY author;";

    return $query;
}
```

For this query you use the SUM function from PostgreSQL and grouped by the author which is something similar to what we would get from a MapReduce query in MongoDB. You are reducing the results from each user into an operation which will return only two fields with the name and the total sum of the RTs.

**MongoDB Map Reduce Query displaying only the user and the rt_count amount in their tweets:**

_id=> Twitter
value=> 26621

_id=> jack
value=> 26681

_id=> realmadrid
value=> 5237

```php
public function map_reduce_query_examples()
{
    echo '<h3>MongoDB Map Reduce Query displaying only the user and the rt_count amount in their tweets: </h3><br>';
    $query = $this->queries->mongo_map_reduce(1);

    $map = new MongoDB\BSON\Javascript($query[0]);
    $reduce = new MongoDB\BSON\Javascript($query[1]);
    $out = ['inline' => 1];

    $cursor = $this->m_col_tweet->mapReduce($map,$reduce,$out);
    $this->show_data($cursor);

}
```

```php
public function mongo_map_reduce($select)
{

    switch($select)
    {
        case 1:
            //Query that returns the total amount of RTs of the tweets from that author.
            $map = 'function(){emit(this.author, this.rt_count)}';
            $reduce = 'function(key, values){return Array.sum(values)}';
            $query = array($map, $reduce);
            return $query;
            break;
    }
}
```

For Mongo we get the same query but made on MongoDB. The syntax is different, as we need to pass a $map and a $reduce function which must be written in JavaScript in order to do their work. The way to call this query is using the function from the collection mapReduce() and pass those parameters. It will return the author and the rt_count of each user.

- **Text Search Queries**

For the Text Search queries are just queries about being able to inputting any text in a variable and pass that text into a function that will send the query to the database. I haven't been able to create a text box and perform the search with that but the implementation is the same as I pass a string through a variable in order to search a word and then I receive the results. You can perform text search queries in any database but the only thing to remark is that in MongoDB you need to have indexed the database with text indexes. In the beginning of the work I've talked about the wildcard text index which allowed me to index every field with a string value as an index and with that you can find text in any value that is a string. I think it's easier to do the Text Search on PostgreSQL as it will return you any tweet with a text but in MongoDB you are more flexible as well as in Solr, but since I don't have different schemas in Solr, the info retrieved can be messed up a little bit.

**PostgreSQL Text Query with text search: RT**

id => 1274375523741732865
author => jack
tw_content => RT @TIDAL: Listen to @Beyonce - "BLACK PARADE" on TIDAL now. ▶️: https://t.co/PMjJFSFJaI https://t.co/c2mKnXlQhY
rt_count => 5420
fav_count => 0
date => Sat Jun 20 16:16:16 +0000 2020

id => 1274092610911563776
author => jack
tw_content => RT @Twitter: 📍 Minneapolis 🎥 @FredTJoseph https://t.co/lNTOkyguG1
rt_count => 4749
fav_count => 0
date => Fri Jun 19 21:32:04 +0000 2020

id => 1274031292640751616
author => jack
tw_content => RT @GodisRivera: 7/ Each billboard highlights Tweets from strong Black voices that channel the collective Black experience as the fight for…
rt_count => 169
fav_count => 0
date => Fri Jun 19 17:28:25 +0000 2020

```php
public function text_search_query_examples($text)
{
    echo '<h3>PostgreSQL Text Query with text search: '.$text.'</h3><br>';
    $query = "SELECT * FROM \"tweet\" WHERE tw_content LIKE '%{$text}%'";
    $result = pg_query($query) or die('La consulta fallo: ' . pg_last_error());
    $this->show_data($result);
}
```

This text search in PostgreSQL will find any tweet with the word RT or that has it between two letters. Most of the time, as you get tweets with the RT tag you will get those as that represents Retweets in the timeline of that user and the Retweet count will be from that tweet.
The query is able to do this using the LIKE and using % between the text in order to make you able to find those letters between any word.

**MongoDB text Query with text search: Minneapolis**

_id=> 5eee3edd74760000f1004fb4
id=> 1274087687469715457
author=> Twitter
content=> 📍 Minneapolis 🎥 @FredTJoseph https://t.co/lNTOkyguG1
rt_count=> 4749
fav_count=> 25054
date=> Fri Jun 19 21:12:30 +0000 2020
hashtags=>

_id=> 5eee3ed774760000f1004fa4
id=> 1274092610911563776
author=> jack
content=> RT @Twitter: 📍 Minneapolis 🎥 @FredTJoseph https://t.co/lNTOkyguG1
rt_count=> 4749
fav_count=> 0
date=> Fri Jun 19 21:32:04 +0000 2020
hashtags=>

```php
public function text_search_query_examples($text)
{
    echo '<h3>MongoDB text Query with text search: '.$text.'</h3><br>';
    $filter = ['$text' => ['$search' => $text]];
    $cursor =  $this->m_col_tweet->find($filter);

    $cursor->setTypeMap(['root' => 'array', 'document' => 'array', 'array' => 'array']);
    $this->show_data($cursor);

}
```

For MongoDB you get the query by passing the filter in which you search a field that contains a search and you pass the $search mode, in which you will try to find the content of the text you have passed to the function, in the case that I had a textbox to input the text, it would be that word. But in my case, I've tried using the "Minneapolis" word an it has found two tweets from the moment they were written.

**Solr Text Search Query searching user: jack and showing their most relevant tweets**

tweet_id: 0: 1274375523741732865
author: 0: jack
content: 0: RT @TIDAL: Listen to @Beyonce - "BLACK PARADE" on TIDAL now. ▶️: https://t.co/PMjJFSFJaI https://t.co/c2mKnXlQhY
rt_count: 0: 5420
fav_count: 0: 0
date: 0: Sat Jun 20 16:16:16 +0000 2020
id: afd63410-23fd-4e5b-a278-172483587200
_version_: 1670037594191167488
score: 1

tweet_id: 0: 1274092610911563776
author: 0: jack
content: 0: RT @Twitter: 📍 Minneapolis 🎥 @FredTJoseph https://t.co/lNTOkyguG1
rt_count: 0: 4749
fav_count: 0: 0
date: 0: Fri Jun 19 21:32:04 +0000 2020
id: 998ecf13-3015-46d7-a45a-218af5620409
_version_: 1670037594395639808
score: 1

```php
public function text_search_query_examples($text)
{
    echo '<h3>Solr Text Search Query searching user: '.$text.' and showing their most relevant tweets</h3><br>';
    $query = $this->solr->createSelect();
    $query->setQuery('author :*'.$text.'*');
    $query->setStart(2)->setRows(20);
    $query->addSort('rt_count', $query::SORT_DESC);

    $result = $this->solr->select($query);
    $this->show_data($result);
}
```

Finally for Solr, you can find the tweets from the author you pass to the function, in this case jack, and after that the tweets showed are ordered by the tweets with the most number of retweets, so usually the most relevant tweets. This query is pretty simple and works pretty well as you only have to put the word of the field and after the ":" you put the text and that's all. You get the results in your variable and can iterate through them.

# SHORT CONCLUSION

The practice part is by far the one that most time has consumed me in this year. There has been a lot troubles for connecting the databases to the web interface and overall it has been a good learning practice as it's an example of how many errors you will find by working with these kind of technologies and in my opinion my knowledge about these databases have grown. Definitely in the future if I have to choose to work with a database I will choose either PostgreSQL or MongoDB. Both have drawn my attention but I need to improve using MongoDB as it has its own type of syntax and in order to be efficient and proficient with it you have to understand it and then "parse" it in your mind to a entity relationship model. Also most of my problems were solved thanks to Javier Jimenez which has helped me a lot with the connection to the databases and how to implement things in PHP, as we both did the PAPI course this year and we decided to make it on PHP.

# BIBLIOGRAPHY


**[1] Alternatives to MongoDB:**
https://medium.com/better-programming/the-top-three-alternatives-to-mongodb-in-2019-399b324e53d8
**[1] DynamoDB vs MongoDB:**
https://blog.panoply.io/dynamodb-vs-mongodb#:~:text=DynamoDB%20uses%20primary%20keys%20to,can%20vary%20for%20different%20documents.
**[2] Oracle Database vs PostgreSQL:**
https://www.educba.com/oracle-vs-postgresql/
**[2] DB Engines PostgreSQL and Oracle:**
https://db-engines.com/en/system/Oracle%3BPostgreSQL
**[3] Elasticsearch differences with Solr:**
https://sematext.com/blog/solr-vs-elasticsearch-differences/
**[3] Elasticsearch vs Solr:**
https://greenice.net/elasticsearch-vs-solr-vs-sphinx-best-open-source-search-platform-comparison/
**[3] Solr vs Elasticsearch:**
https://logz.io/blog/solr-vs-elasticsearch/
**[3] DB Engines Solr and Elasticsearch:**
https://db-engines.com/en/system/Elasticsearch%3BSolr
**[4] Scaling PostgreSQL for Large Amounts of Data:**
https://severalnines.com/database-blog/scaling-postgresql-large-amounts-data
**[5] Divide and Conquer: High Scalability With MongoDB Sharding:**
https://dzone.com/articles/divide-and-conquer-high-scalability-with-mongodb-t#:~:text=MongoDB%20supports%20horizontal%20scaling%20through,with%20large%20sets%20of%20data.&text=Sharding%20allows%20you%20to%20add,runs%20on%20a%20different%20server.
**[6] Scaling Big Data with Hadoop and Solr - Second Edition:**
https://books.google.es/books/about/Scaling_Big_Data_with_Hadoop_and_Solr_Se.html?id=NAGzCAAAQBAJ&redir_esc=y
**[7] PostgreSQL Documentation:** https://www.postgresql.org/docs/
**[8] MongoDB Documentation:** https://docs.mongodb.com/
**[9] MongoDB PHP Implementation:**
https://docs.mongodb.com/php-library/v1.6/reference/class/MongoDBClient/
https://www.php.net/manual/en/class.mongodb.php
**[10] MongoDB SetTypeMap:**
https://www.php.net/manual/es/mongodb-driver-cursor.settypemap.php
**[11] MongoDB Aggregation:**
https://docs.mongodb.com/manual/aggregation/
https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/
https://docs.mongodb.com/manual/reference/method/db.collection.aggregate/
**[12] MongoDB MapReduce:**
https://docs.mongodb.com/manual/core/map-reduce/
https://docs.mongodb.com/php-library/v1.2/reference/method/MongoDBCollection-mapReduce/
**[13] Solr Documentation:** https://lucene.apache.org/solr/guide/8_5/
**[14] Solarium PHP Implementation:**
https://solarium.readthedocs.io/en/stable/

**[15] Explanation of BASE terminology:**
https://stackoverflow.com/questions/3342497/explanation-of-base-terminology#:~:text=The%20BASE%20acronym%20is%20used,certain%20databases%2C%20usually%20NoSQL%20databases.&text=There%20are%20only%20few%20articles,consistency%2C%20isolation%20and%20durability%20properties
**[16] Twitter API for PHP:**
https://iag.me/socialmedia/build-your-first-twitter-app-using-php-in-8-easy-steps/
**[17] ACID definition:**
https://en.wikipedia.org/wiki/ACID