universidad SANJORGE
GRUPO SANVALERO

Deep Twitter Analysis

By Raúl Martínez

Index

Final Task – Deep Twitter Analysis

This new service consists of indexing in the databases Solr, MongoDB and PostgreSQL the most interesting Tweets so that people can consult them on the Internet through a small graphical interface. A small google-style search engine, the most interesting tweets. The objective is to implement a real service and compare the pros and cons of each of the data base. To discover how to use these 3 databases in a real application. The program should have 3 parts and will aim to automate as much as possible all tasks.[1]

1. Getting Tweets – Two methods

The first approach to obtain tweets was to download a huge JSON dump, filter it by keywords and code an adapter to insert them into our model.

To do this, we downloaded a small piece of Twitter compiled by the Internet Archive.

Nombre	Tamaño	Progreso	Prioridad de descarga
archiveteam-twitter-stream-2019-05	57,61 GiB	100%	Mixta

Downloading a couple of tweets from "archiveteam-twitter-stream-2019-05" [2]

Then, after uncompressing them, we decided to filter some of them by the following words: bitcoin, blockchain and cryptocurrency, ignoring case.

```
rme@DESKTOP-I9FRKM8:/mnt/c/Users/RME/Downloads/archiveteam-twitter-stream-2019-05/twitter_stream_2019_05_30/30$
 grep -Ehi 'bitcoin|blockchain|cryptocurrency' */*.json > output.json
```

Grepping like a pro

After this process we obtained the following output file with just the Tweets that contained any of our terms:

output.json	28/05/2020 0:16	Archivo JSON	21.883 KB

Final file containing 4093 unique tweets and their profiles.

To read this file we have built a custom TwitterService_json.php, this allows us to work indistinctly with JSON, Twitter API, MongoDB, PostgreSQL and Solr by calling each service.

TwitterService_json.php	13/06/2020 16:37	Archivo PHP	4 KB

Then, as required by the task we implemented a way to get Tweets directly from the Twitter API.

Standard search API

Returns a collection of relevant Tweets matching a specified query.

Please note that Twitter's search service and, by extension, the Search API is not meant to be an exhaustive source of Tweets. Not all Tweets will be indexed or made available via the search interface.

To learn how to use Twitter Search effectively, please see the Standard search operators page for a list of available filter operators. Also, see the Working with Timelines page to learn best practices for navigating results by `since_id` and `max_id`.

Resource URL

`https://api.twitter.com/1.1/search/tweets.json`

Resource Information

Response formats	JSON
Requires authentication?	Yes
Rate limited?	Yes
Requests / 15-min window (user auth)	180
Requests / 15-min window (app auth)	450

Parameters

Name	Required	Description	Default Value	Example
q	required	A UTF-8, URL-encoded search query of 500 characters maximum, including operators. Queries may additionally be limited by complexity.		`@noradio`

Twitter Search API documentation [3]

After reading the documentation, we registered a new API key and looked for a simple PHP library to interact with the API.

twitter-api-php

Simple PHP Wrapper for Twitter API v1.1 calls

downloads 2.6M build passing version 1.0.6

Changelog || Examples || Wiki

Instructions in StackOverflow post here with examples. This post shows you how to get your tokens and more. If you found it useful, please upvote / leave a comment! :)

The aim of this class is simple. You need to:

- Include the class in your PHP code

"The simplest PHP Wrapper for Twitter API v1.1 calls" [4]

After reading how it worked, we implemented it as TwitterService_api.php

TwitterService_api.php	10/06/2020 1:03	Archivo PHP	4 KB
TwitterAPIExchange.php	21/05/2020 22:17	Archivo PHP	11 KB

Each one of this TwitterService classes implement similar methods so they are easily interchangeable to switch between input and output mediums.

2. Inserting Tweets – Three Databases

As explained before, currently we have TwitterService_json.php to read from JSON and TwitterService_api.php to read from the Twitter API. Now we need some classes to read and write from all our three databases.

Class Name	Backend	Features
TwitterService JSON	JSON flat file	READ ONLY
TwitterService API	Twitter Search API	READ ONLY
TwitterService SQL	PostgreSQL (or other PDO*)	READ & WRITE
TwitterService MongoDB	MongoDB	READ & WRITE
TwitterService Solr	Apache Solr	READ & WRITE

* TwitterService SQL was implemented using PHP PDO so it supports PostgreSQL but also Cubrid, MS SQL Server, Firebird, IBM, Informix, MySQL, Oracle, ODBC, SQLite and 4D controllers. [5]

TwitterService_api.php	10/06/2020 1:03	Archivo PHP	4 KB
TwitterService_json.php	13/06/2020 16:37	Archivo PHP	4 KB
TwitterService_mongodb.php	17/06/2020 11:14	Archivo PHP	6 KB
TwitterService_solr.php	17/06/2020 12:55	Archivo PHP	9 KB
TwitterService_sql.php	18/06/2020 11:47	Archivo PHP	8 KB

Application Setup Instructions

The setup script is located on \TwitterFinalTask\fetch_tweets.php and should be run on the browser like: http://localhost/TwitterFinalTask/fetch_tweets.php after setting up the desired actions.

```
How to use:
0. Optional: Create JSON file on \TwitterFinalTask\data\tweets.json
0. Configure Twitter API key on \TwitterFinalTask\components\TwitterService_api.php
0. Configure PostgreSQL database login on \TwitterFinalTask\components\config\sql_config.php
0. Configure MongoDB database string on \TwitterFinalTask\components\TwitterService_mongodb.php
0. Configure Solr database string on \TwitterFinalTask\components\TwitterService_solr.php

1. Set $twitterReader to the INPUT method: API or JSON

2. Set $twitterWriter to the OUTPUT method: SQL, MONGODB or SOLR

3. Read tweets & users or statuses
    3.1 For SQL use $tweets = $tweetReader->getTweets() and $users  = $tweetReader->getUsers();
    3.2 For MONGODB or SOLR use $statuses = $twitterService_json->getStatuses();

4. Write tweets & users or statuses
    4.1 For SQL use $tweetWriter->updateTweets($tweets); and $tweetWriter->updateUsers($users);
    4.2 For MONGODB or SOLR use $tweetWriter->updateStatuses($statuses);

5. Finished, now you can check on the database and use the application.
```

Scaling solutions

PostgreSQL can be scaled vertically by using more powerful hardware as explained in "Scaling PostgreSQL" [45].

A master/slave system is also possible to scale horizontally, this way you can READ from any slave server and write on master.

MongoDB scaling can be also achieved vertically by upgrading the hardware or horizontally by using Sharding. "*Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations*." [46]. Mapreduce operations are compatible with sharding.

Solr can scale in two ways, Distribution "*To distribute an index, you divide the index into partitions called shards, each of which runs on a separate machine. Solr then partitions searches into sub-searches, which run on the individual shards, reporting results collectively*" and Replication "*You have a large search volume which one machine cannot handle, so you need to distribute searches across multiple read-only copies of the index.*" [47]

3. Query Interface – Web interface

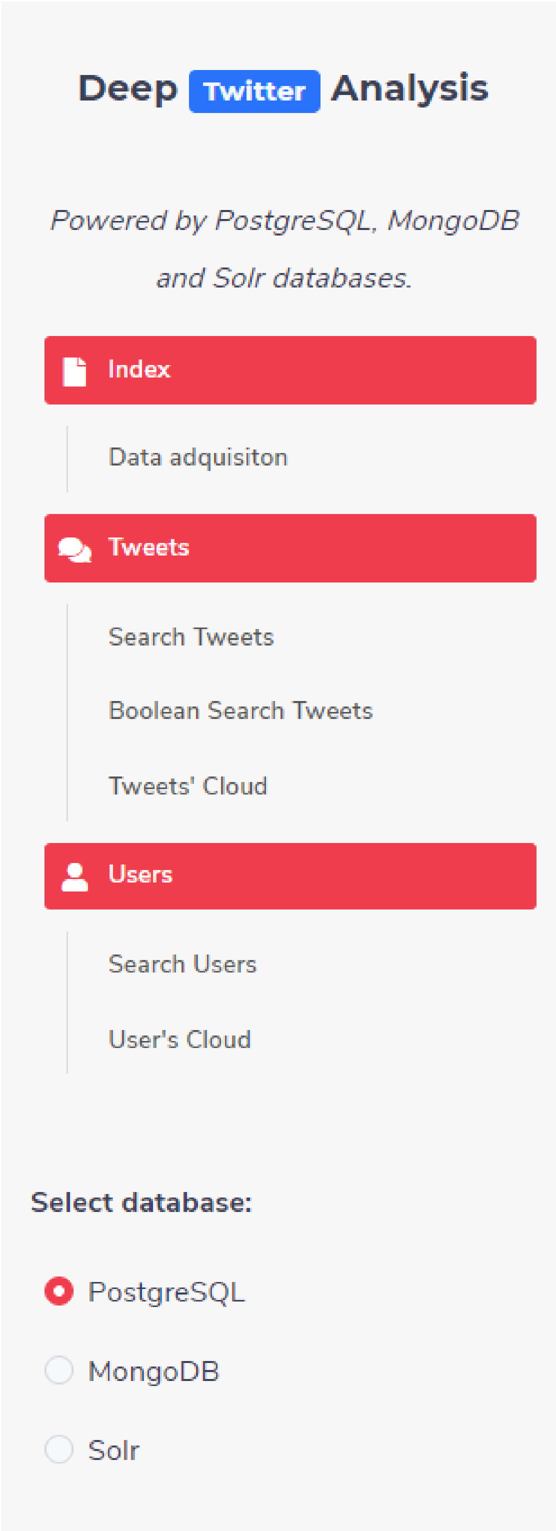
For the UI, we have designed a nice query web interface with the help of the Cirrus CSS framework [7] was chosen because it is lightweight and has very beautiful components that we can reuse. It should be noted that components were used but the structure is custom created based on the documentation.

The requirements for this setup are the following:

- PHP and Apache2 (XAMMP can be used)
- MongoDB php extension
- PostgreSQL
- MongoDB
- Solr
- Twitter API keys

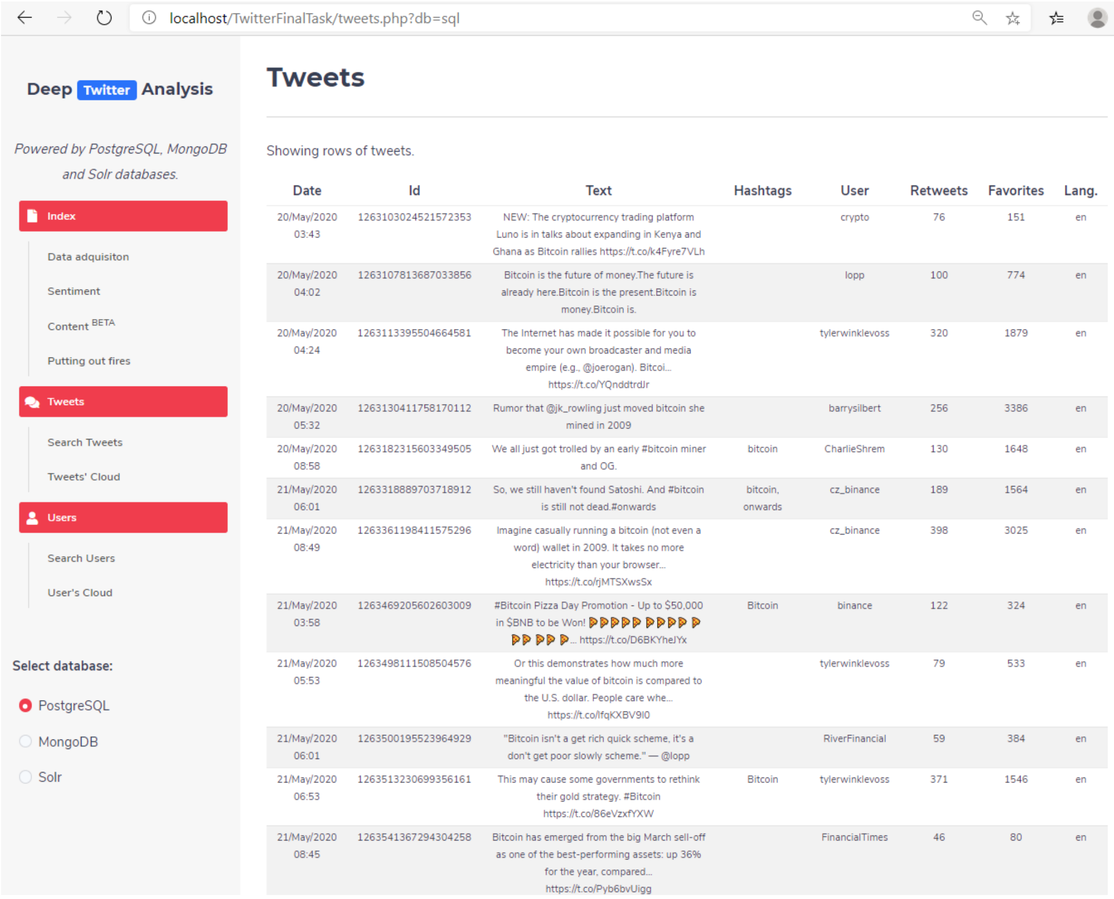
The MongoDB php extension is required and not installed by default, the other two databases work by default. [8]
Install the correct version depending on the PHP version and drop it on /php/ext/ folder.

Interface: Side bar



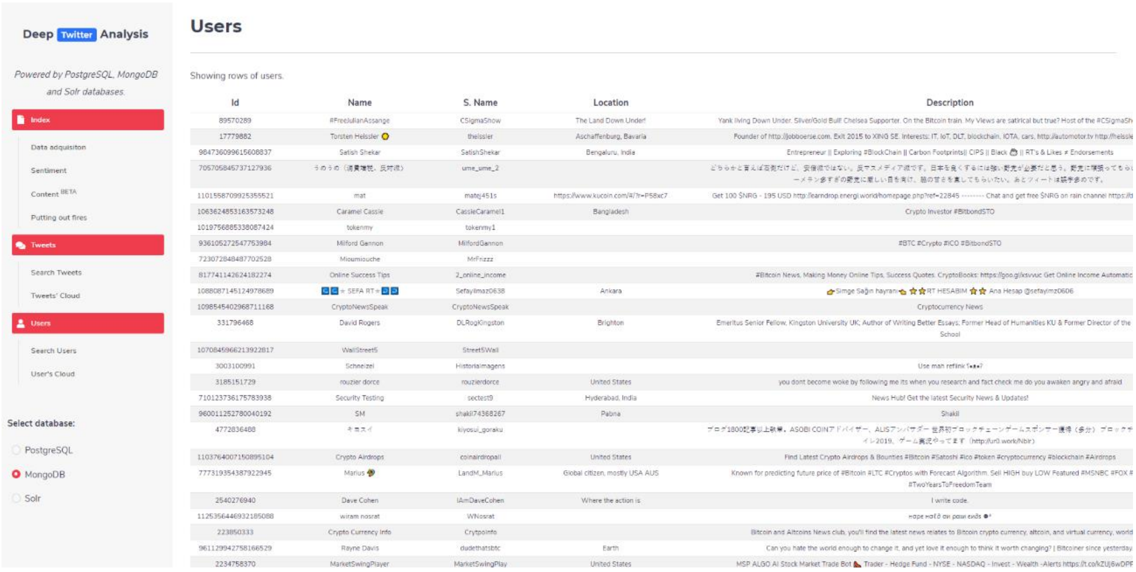
At the top, the logo of the site "Deep Twitter Analysis", below the description, then the menu, and at the bottom the database selection tool.

Interface: Show all tweets



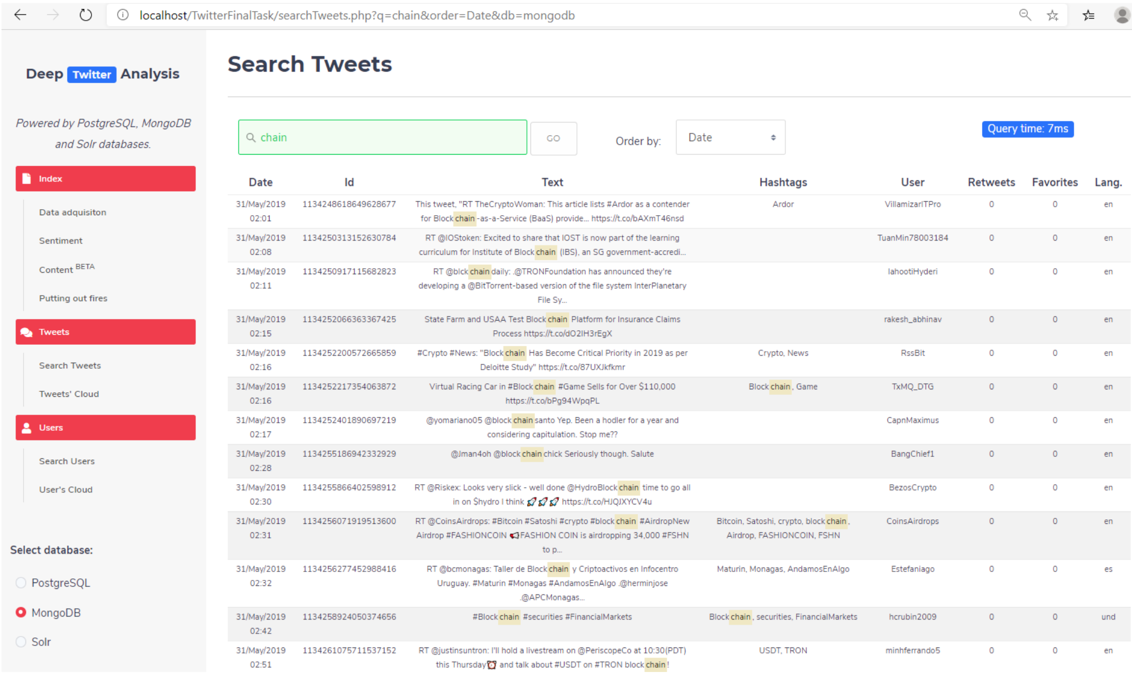
By clicking "Tweets" you can visualize all of them on a nice table.

Interface: Show all users



By clicking “Users” you can visualize all of them on a nice table. Appreciate that we selected MongoDB as backend.

Interface: Search tweets

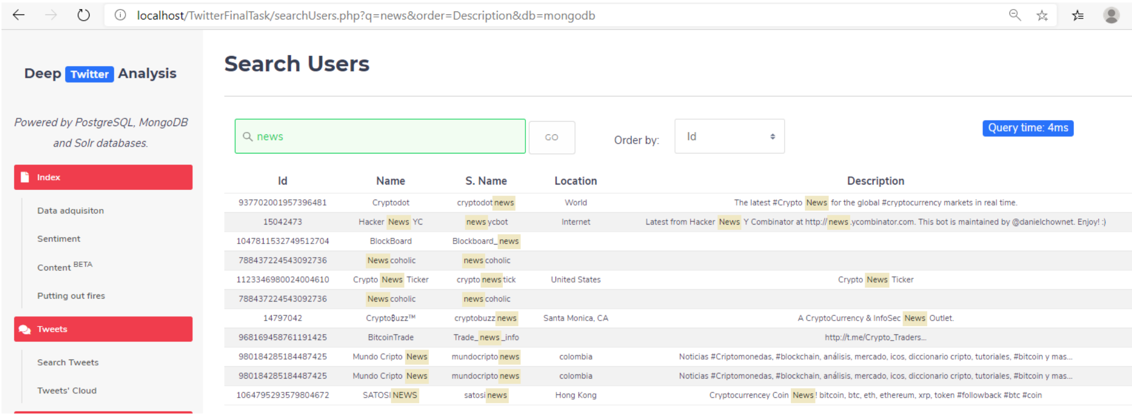


By clicking on "search tweets" we can perform searches on tweets and order the results. Also, we can see the query time on the right to evaluate the performance.

Notice that the searched word appears highlighted, this is provided by mark.js [44]

It can be noticed that some tweets have 0 retweets and 0 favorites, this is not a bug, this is the result of using the Tweet Streaming API to collect the tweets, as the tweets are obtained just as they are published so no interactions have been submitted yet by other users.

Interface: Search users



By clicking on "search users" we can perform searches on tweets and order the results. Also, we can see the query time on the right to evaluate the performance.

Notice that the searched word appears highlighted, this is provided by mark.js [44]

Interface: Tweets wordcloud

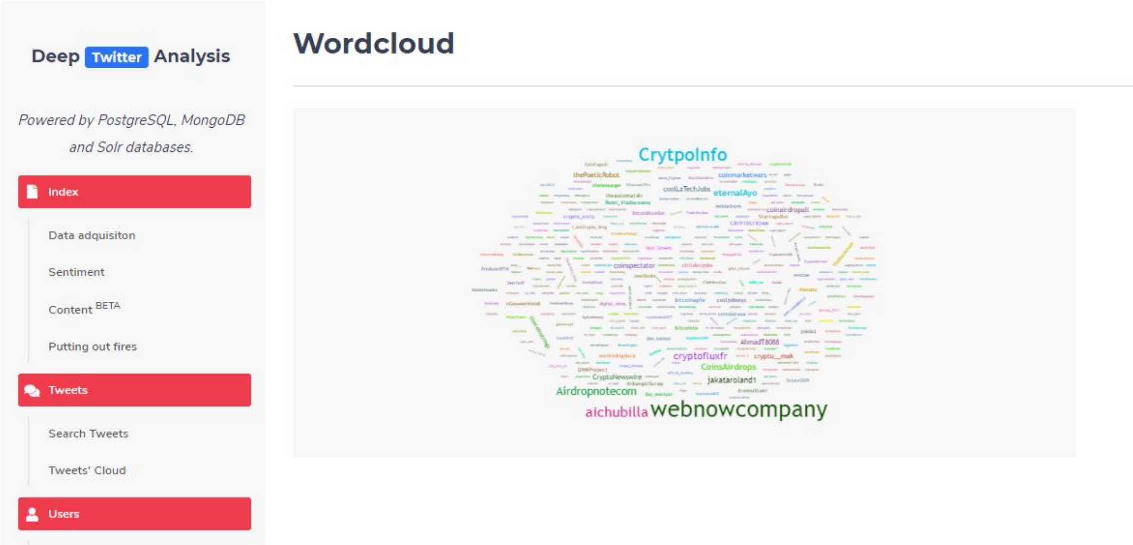
By clicking on "tweets cloud" we can visualize users with popular tweets. Implemented using the wordcloud2.js library [43].

It represents the popularity aggregated by user, if a user has more popularity it appears bigger on the chart.

```
<script>
var list = [
 ['crypto', 227],
 ['lopp', 874],
 ['tylerwinklevoss', 2199],
 ['barrysilbert', 3642],
 ['CharlieShrem', 1778],
 ['cz_binance', 1753],
 ['cz_binance', 3423],
 ['binance', 446],
 ['tylerwinklevoss', 612],
 ['RiverFinancial', 443],
 ['tylerwinklevoss', 1917],
 ['FinancialTimes', 126],
 ['YourBTCC', 312],
 ['lopp', 512],
 ['TrustWalletApp', 216],
];
</script>
```

The database query generates this JavaScript list that is feed to the library.

Interface: Users wordcloud



By clicking on "user's cloud" we can visualize users with popular tweets. Implemented using the wordcloud2.js library [43].

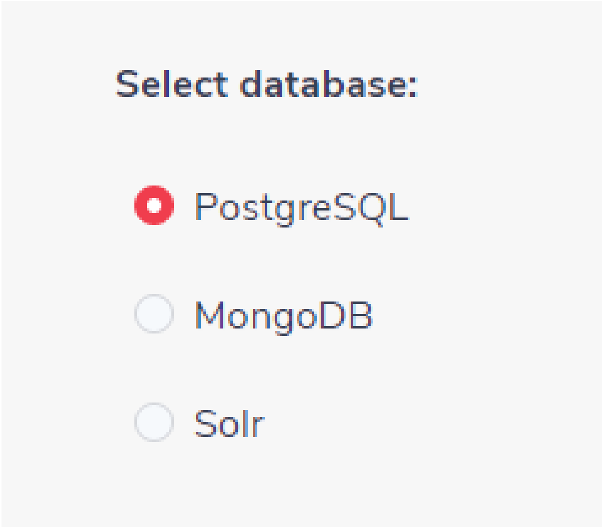
It represents the number of tweets aggregated by user, if a user has more tweets it appears bigger on the chart.

```
<script>
var list = [
 ['preadyplayerone', 2],
 ['money_online_b', 2],
 ['jucktion', 2],
 ['itsme_anne11', 2],
 ['childdevjobs', 5],
 ['HotwaxMedia', 2],
 ['Worship_Emma', 2],
 ['kuriharan', 2],
 ['IT360NewsCom', 3],
 ['CoinDebate', 2],

[...]

 ['kristiandanica', 2],
 ['mbarto6', 2],
 ['BurniteGeorge', 2],
 ['daro3612', 3],
 ['julianor', 2],
 ['SammarA12281909', 2],
 ['twisteddotcom', 2],
 ['eraser', 2],
 ['cryptorandolph', 2],
 ['xtdisnkfe', 2],
 ['EdSn0wd3n', 2],
 ['kasootsuuka', 4],
 ['jiminell1', 2],
 ['how_to_coin', 2],
 ['apagut', 2],
 ['RssBit', 3],
 ['CoinLook', 3],
 ['aichubilla', 13],
 ['cryptoretreat', 3],
];
</script>
```

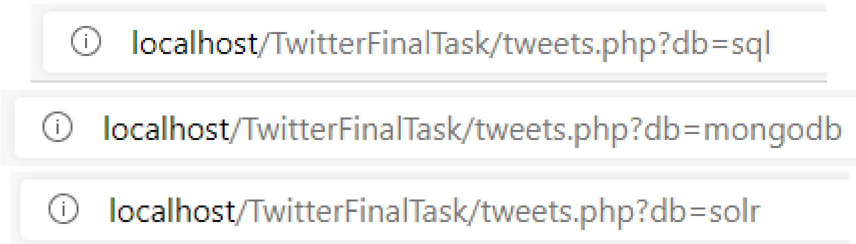
The database query generates this JavaScript list that is feed to the library.

- **Being able to do text searches in any database**

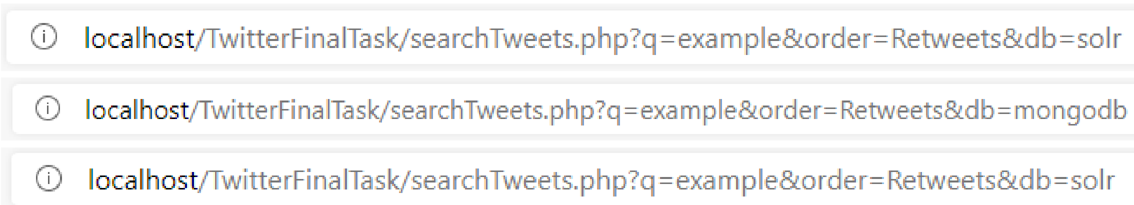
As explained before, a database selector can be found on the left part of the website. This selector contains three radio buttons that once allows to select one database. Once you select, the page will be reloaded and you will now use that database while the query persists the change.



The database selector is shown on every page



When a database is selected, the value is passed on the URL.



If the database is changed and a query already exists, the database changes but the query remains.

- **To be able to facet all databases.**

PostgreSQL

```
$statement = $this->db->prepare("SELECT users.screen_name, count(*) FROM users
RIGHT JOIN tweets ON users.screen_name = tweets.user GROUP BY users.screen_name");
```

MongoDB

```
$command = new MongoDB\Driver\Command(
[
    'aggregate' => 'tweets',
    'pipeline' =>
    [
        ['$group' => ['_id' => '$user', 'count' => ['$sum' => 1]]],
        ['$match' => ['count' => ['$gt' => 1] ] ],
    ],
    'cursor' => new stdClass,
]);
$cursor = $this->manager->executeCommand('twitter', $command);
```

Solr

```
// Query Settings
$action = 'select';
$q = '*:*';      //Query
$wt = 'json';           //writer type, ouput format
$rows = 1000;
$facet_field = 'screen_name';

// Building query
$q = urlencode($q);
$url = $this->endpoint . '/solr/' . $this->collection . '/' . $action . '?' .'facet.field=
' . $facet_field . '&facet=on'. '&q=' . $q . '&wt=' . $wt . '&rows=' . $rows ;
```

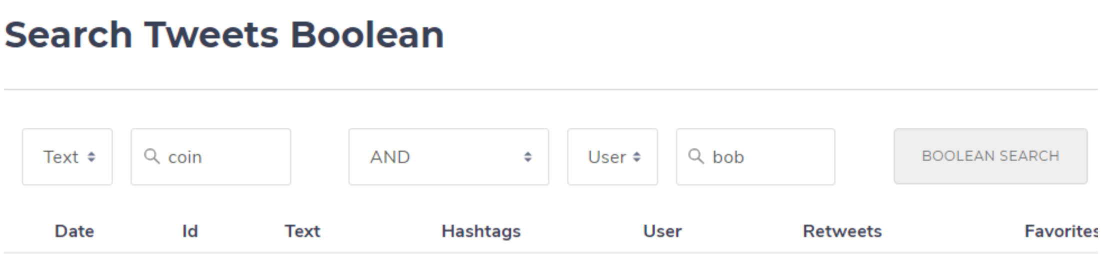
	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Facet Implementation	Group By & Count(*) [10]	$facet (aggregation) [11]	Facet [12]

The result of this facet is explained below in “Wordcloud”.

- **Being able to search with Boolean expressions.**
  Its possible to search with Boolean expressions including AND, OR and NOT between two fields.

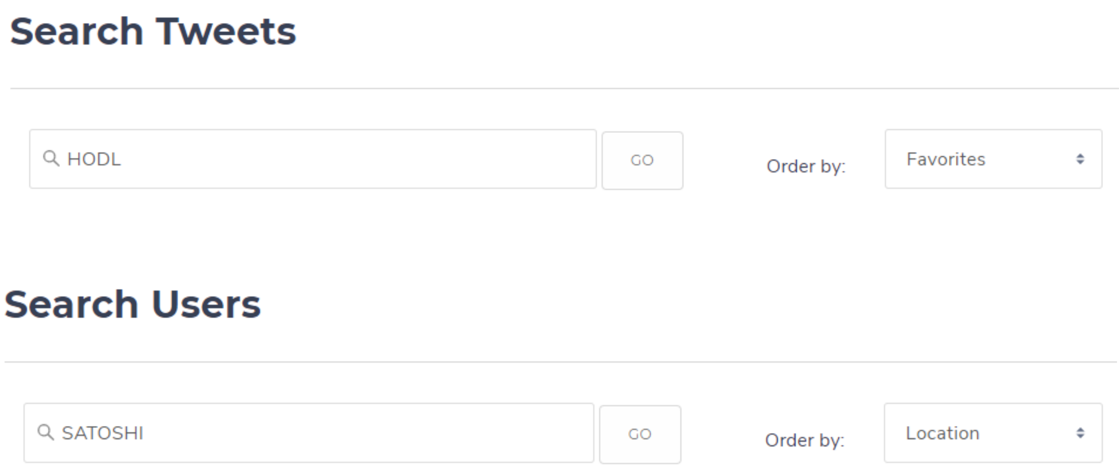
```
public function getTweetsByBooleanSearch($field1 = '', $q1 = '', $operator = '', $field2 = '', $q2 = ''): array
{
    try
    {
        if($field1 == '')
        {
            $statement = $this->db->prepare("SELECT * FROM tweets");
        }
        else
        {
            $statement = $this->db->prepare("SELECT * FROM tweets WHERE $field1 LIKE :q1 $operator $field2
            LIKE :q2");
        }
        $statement->execute([
            ":q1" => '%'.$q1.'%', ":q2" => '%'.$q2.'%'
        ]);

        $rows = $statement->fetchAll(PDO::FETCH_CLASS, 'Tweet');    // Load rows in array
        $db = null;     // Close connection

        return $rows;
    }
    catch (PDOException $exception)
    {
        echo $exception->getMessage();
    }
}
```

	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Boolean Expressions	AND, OR and NOT [13]	$and, $not, $nor, $or [14]	AND, OR and NOT [15]



- **To be able to search by categories.**

  The interface allows to search by categories, for example by Tweets or by Users as they have different fields.



Obviously, this functionality is implemented in all three databases.

PostgreSQL
```
$statement = $this->db->prepare("SELECT * FROM $table WHERE $collumn LIKE :string")
```
MongoDB
```
$rows = $this->manager->executeQuery('twitter.statuses', $query);
```
Solr
```
//Picking different fields from the query result
```

	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Categories Implementation	WHERE [16]	$where [17]	Fields [15]

Wordcloud:
We have implemented two wordclouds using the wordcloud2.js library [43] to demonstrate the facet search.

One of the clouds represents the number of tweets aggregated by user, if a user has more tweets it appears bigger on the chart.

The second cloud represents the influence of a user counted by the number of likes and retweets that the tweets for that user have in total.

4. Database Comparison

As an introduction we are going to compare briefly the three database engines:

	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Main usage	Relational DBMS	Document store	Search engine
Popularity	#1 Most popular	#2 Less popular	#3 Niche
Language	C	C++	Java
Scheme	Yes	schema-free	Yes
MapReduce	No	Yes	Spark-solr
Consistency	Immediate	Eventual / Immediate	Eventual
Foreign Keys	Yes	No	No
Transactions	ACID	Multi-document ACID, snapshot isolation	Optimistic locking

Source: [6]

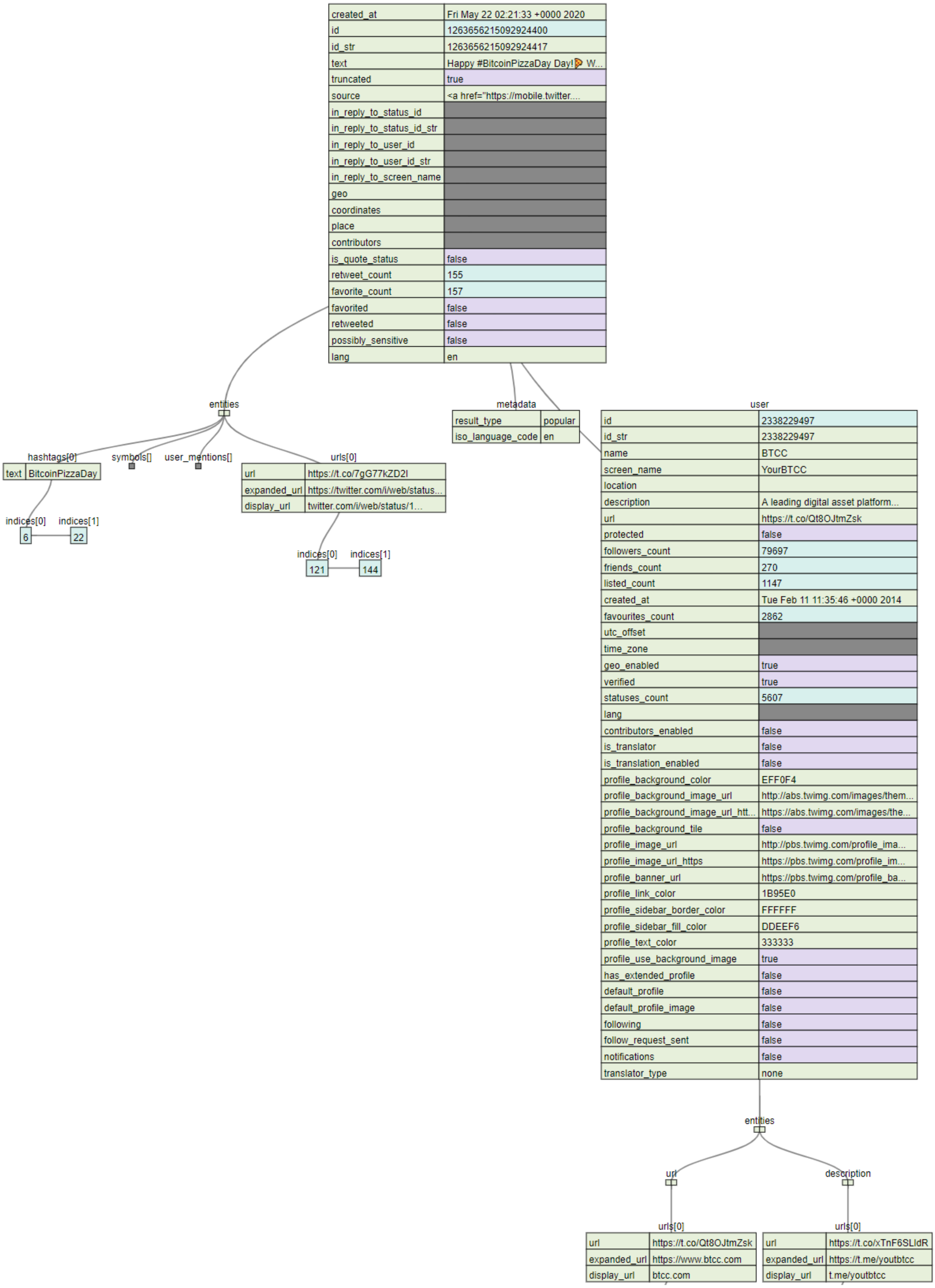
- **Design a schema to introduce users and tweets: a table with the three different schemas, a clear description of differences, PROS and CONS with (0,5 point)**

  PostgreSQL:
  We will be using a typical SQL schema consisting of two tables, primary and foreign keys.

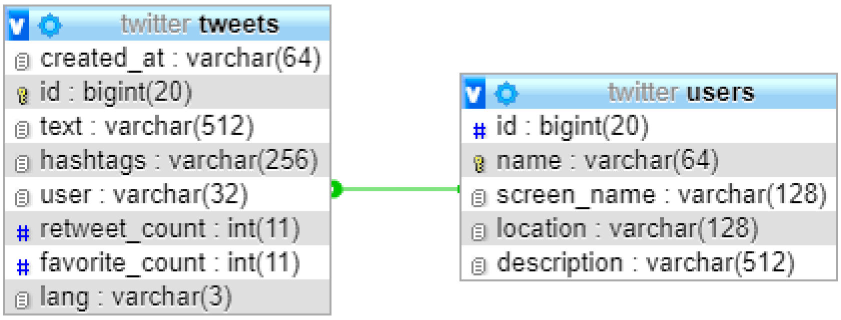
  MongoDB:
  We will be using one collection that contains one document per tweet and inside each document details about the tweet and the user who posted it.

  Solr:
  We will be using an XML format that contains one <doc> per tweet, this <doc> contains all the fields from the tweet and all the fields from the user without nesting.

Twitter API & JSON Schema *(Visualized with VTREE [9])*



PostgreSQL schema



tweets Examinar Estructura Buscar Insertar Vaciar Eliminar

created_at	id	text	hashtags	user	retweet_count	favorite_count	lang
Wed May 20 18:58:27 +0000 2020	1263182315603349505	We all just got trolled by an early #bitcoin miner...	[{"text":"bitcoin","indices":[36,44]}]	CharlieShrem	130	1648	en
Thu May 21 04:01:09 +0000 2020	1263318889703718912	So, we still haven't found Satoshi. And #bitcoin i...	[{"text":"bitcoin","indices":[40,48]},{"text":"onw...	cz_binance	189	1564	en
Thu May 21 06:49:16 +0000 2020	1263361198411575296	Imagine casually running a bitcoin (not even a wor...	[]	cz_binance	398	3025	en

users Examinar Estructura Buscar Insertar Vaciar Eliminar

id	name	screen_name ▲ 1	location	description
396045469	Barry Silbert	barrysilbert	New York	Founder/CEO @DCGco, parent of @GrayscaleInvest @Ge...
877807935493033984	Binance	binance		Exchange The World. Customer Service Enquiries: ...
22682896	Charlie Shrem	CharlieShrem	Sarasota, FL	ðŸŽ§Host & Storyteller https://t.co/tWQMV7zDkk ðŸ“...
928759224599040001	Bloomberg Crypto	crypto		A look at how cryptocurrencies and blockchain are ...

MongoDB schema

```
_id ObjectId
created_at String
id_str String
text String
hashtags String
screen_name String
retweet_count Int32
favorite_count Int32
lang String
user Object
```

twitter.statuses DOCUMENTS 4.1k TOTAL SIZE 2.3MB AVG. SIZE 585B

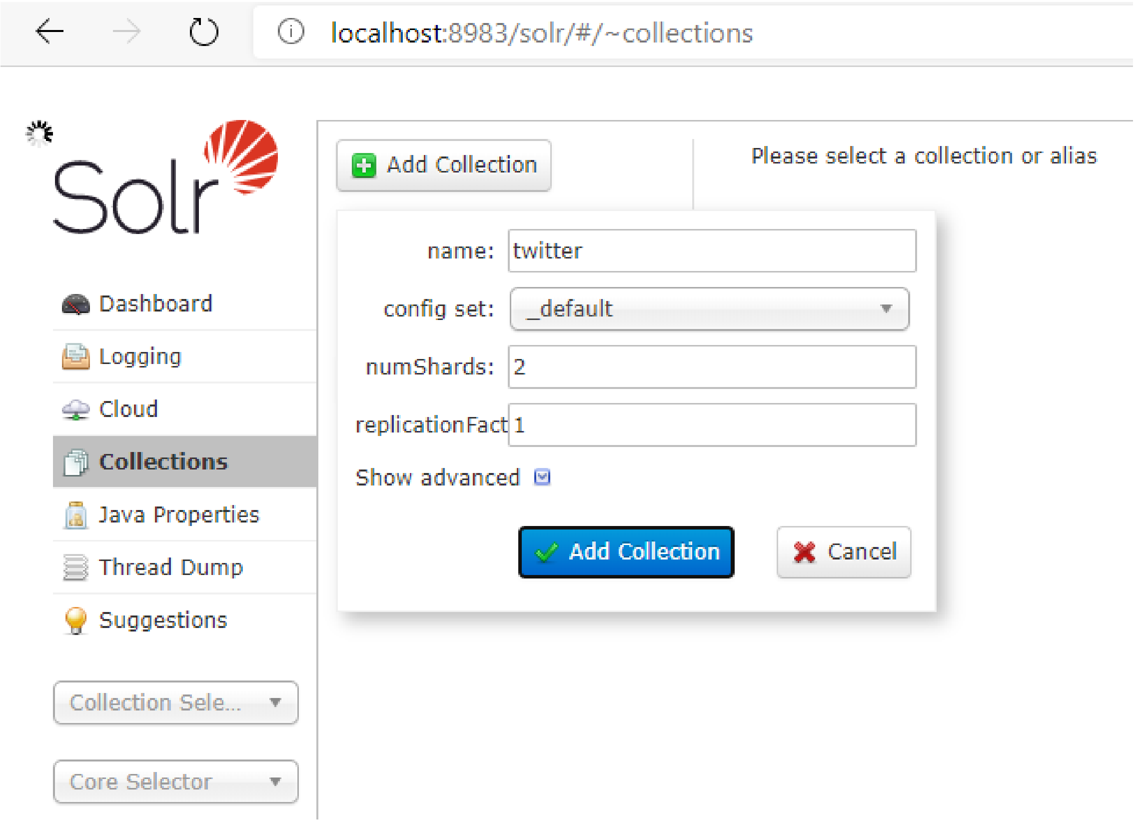
```
_id: ObjectId("5ee4e4bba554830dcc008bdc")
created_at: "Thu May 30 06:29:08 +0000 2019"
id_str: "1133983664449003520"
text: "Soooo not the Russians Bill??"
hashtags: "[]"
screen_name: "CSigmaShow"
retweet_count: 0
favorite_count: 0
lang: "en"
user: Object
    id_str: "89570289"
    name: "#FreeJulianAssange"
    screen_name: "CSigmaShow"
    location: "The Land Down Under!"
    description: "Yank living Down Under. Silver/Gold Bull! Chelsea Supporter. On the Bi..."
```

Solr schema

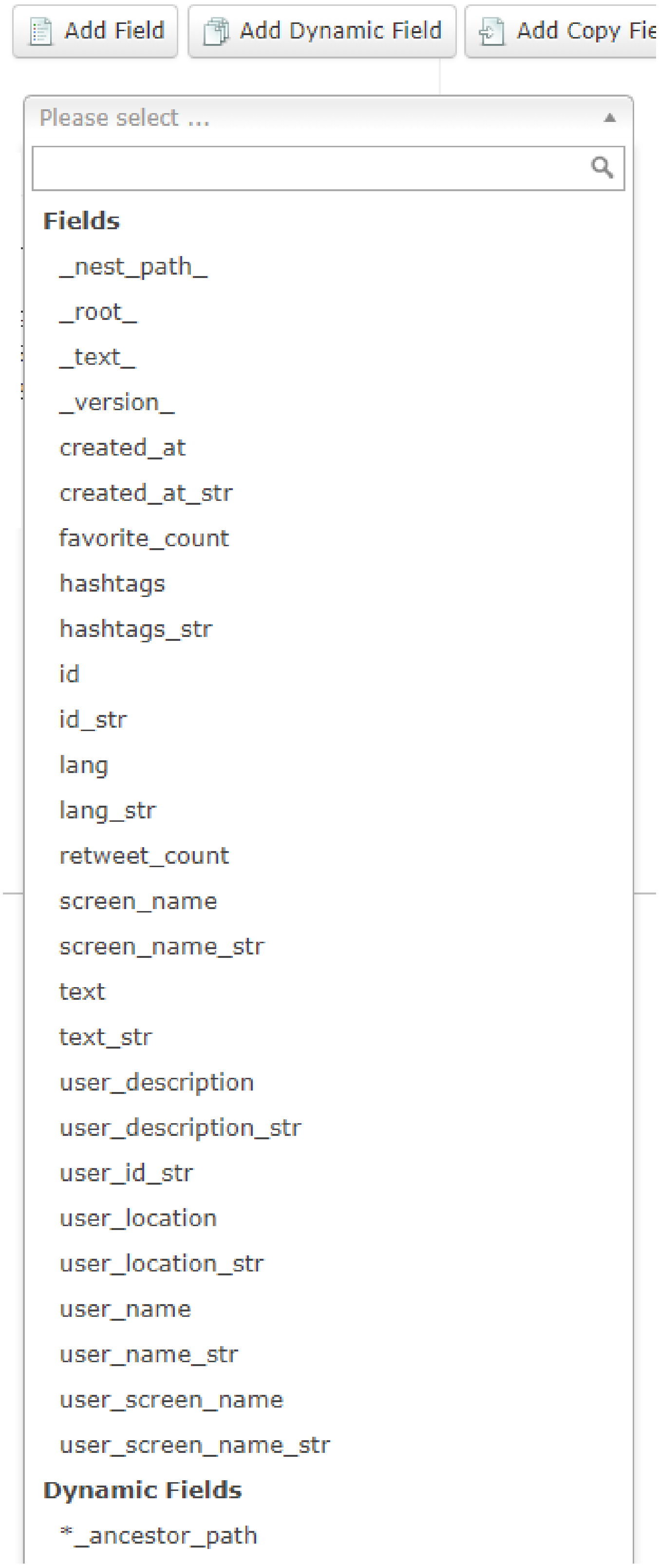
For the Solr search platform we have chosen an XML input format, to show our ability to adapt to multiple data representation formats.

name	field_Text
created_at	Thu May 30 18:30:57 +0000 2019
id_str	1134165315590524929
text	"El Foro Económico Mundial (FEM) anunció la formación de seis "cuartos consejos de la revolución industrial" separa… https://t.co/BodAORiNIq
hashtags	[]
screen_name	CienciadBolsiyo
retweet_count	0
favorite_count	0
lang	es
user_id_str	3557533637
user_name	Ciencia del Bolsillo
user_screen_name	CienciadBolsiyo
user_location	Venezuela
user_description	Impulsamos el conocimiento, la preparación, la descentralización y la usabilidad para la adopción de la #Blockchain las #Criptomonedas y la #Economía tokenizada

```xml
<doc>
    <field name="created_at">Thu May 30 18:30:57 +0000 2019</field>
    <field name="id_str">1134165315590524929</field>
    <field name="text">"El Foro Económico Mundial (FEM) anunció la formación de seis
    "cuartos consejos de la revolución industrial" separa… https://t.co/BodAORiNIq</field>
    <field name="hashtags">[]</field>
    <field name="screen_name">CienciadBolsiyo</field>
    <field name="retweet_count">0</field>
    <field name="favorite_count">0</field>
    <field name="lang">es</field>
    <field name="user_id_str">3557533637</field>
    <field name="user_name">Ciencia del Bolsillo</field>
    <field name="user_screen_name">CienciadBolsiyo</field>
    <field name="user_location">Venezuela</field>
    <field name="user_description">Impulsamos el conocimiento, la preparación, la
    descentralización y la usabilidad para la adopción de la #Blockchain las #Criptomonedas
    y la #Economía tokenizada</field>
</doc>
```



Creating the collection with two shards



List of fields in the schema as shown by the Solr Admin

Comparison Table:

	PostgreSQL / any SQL	MongoDB	Solr
Schema rigidity	High	None	Medium
Tables	Yes	No	No
Relations	Yes	No	No
Typed fields	Yes	No	Yes
Basic units	Databases & Tables	DBs & Collections	Collections

Pros. and Cons. Table:

	PostgreSQL	*MongoDB*	*Solr*
Pros.	Predictable fields	Flexibility, Performance	Flexibility, Performance, Search performance
Cons.	Rigidity, Search performance	Redundant information	Redundant information

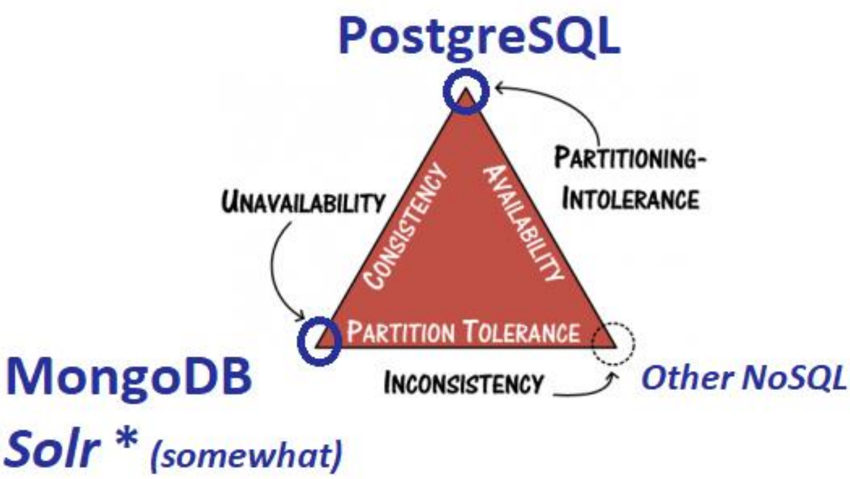
There is redundant information in MongoDB and Solr because every tweet includes details about the user that created it.

This is solved in PostgreSQL by having relation between two tables, users and tweets.

- **Inserting Data (pros and cons of the three databases), concurrency, transaction, ACID, BASE of the different databases. Example of the insert method in the three languages (insert one or multiple) (2 points)**

Comparison Table:

	PostgreSQL	MongoDB	Solr
Document Upload	Not directly	JSON	XML, JSON, CSV…
Concurrency	MVCC [19]	multi-granularity locking [20]	Atomic updates [21], in-place updates [22], optimistic concurrency [23]
Transactions	Yes, [24]	Yes, in 4.0+ [25]	No [26]
ACID	Yes, [27]	Kind of, [28]	No [29]
BASE	No	Yes	No
Cap Theorem	Avail. & Consist.	Consist. & Part. Tol	Somewhat Consist. & Part. Tol [30]
Insert	SQL INSERT	db.collection.insert()	API /solr/update/json
Inserted Data	Tweets & Users separated	Statuses (combination)	Statuses (combination)



Disadvantages of MVCC

"Because different transactions will have visibility to a different set of rows, Postgres needs to maintain potentially obsolete records. Therefore, an UPDATE creates a new row and why DELETE doesn't really remove the row: it merely marks it as deleted and sets the XID values appropriately. As transactions complete, there will be rows in the database that cannot possibly be visible to any future transactions. These are called dead rows. Another problem that comes from MVCC is that transaction IDs can only ever grow so much – they are 32 bits and can "only" support around 4 billion transactions. When the XID reaches its max, it will wraparound and start back at zero. Suddenly all rows appear to be in future transactions, and no new transactions would have visibility into those rows.

Both dead rows and the transaction XID wraparound problem are solved with VACUUM. This should be routine maintenance, but thankfully Postgres comes with an auto vacuum daemon that will run at a configurable frequency. It is important to keep an eye on this because different deployments will have different needs when it comes to vacuum frequency." [19]

ACID

- *Atomicity: The database transaction must completely succeed or completely fail. Partial success is not allowed.*

- *Consistency: During the database transaction, the RDBMS progresses from one valid state to another. The state is never invalid.*

- *Isolation: The client's database transaction must occur in isolation from other clients attempting to transact with the RDBMS.*

- *Durability: The data operation that was part of the transaction must be reflected in nonvolatile storage and persist after the transaction successfully completes. Transaction failures cannot leave the data in a partially committed state. [31]*

BASE

- *Basically Available: The system is guaranteed to be available for querying by all users. (No isolation here.)*
- *Soft State: The values stored in the system may change because of the eventual consistency model, as described in the next bullet.*
- *Eventually Consistent: As data is added to the system, the system's state is gradually replicated across all nodes. For the short period the state of the file system is not consistent. [31]*

CAP Theorem

- *Consistency: Like the C in ACID, all nodes in the system would have the same view of the data at any time.*
- *Availability: The system always responds to requests.*
- *Partition tolerance: The system remains online if network problems occur between system nodes. [31]*

	ACID	BASE
Consistency	Strong	Weak
Guarantee	Reliable	Best effort
Isolation	Yes	No
Availability	When possible	Yes
Atomicity	Yes	Approximate answers
Approach	Conservative	Aggressive
Developed	Time ago	Recent
Generation	Boomers	Millennials & Zoomers

Source [32]

Inserting on PostgreSQL:

```
public function updateTweet($tweet)
{
    try
    {
        $statement = $this->db->prepare("REPLACE INTO tweets (created_at, id,
        text, hashtags, user, retweet_count, favorite_count, lang)
            VALUES (:created_at, :id, :text, :hashtags, :user, :retweet_count,
            :favorite_count, :lang)");
        $statement->execute([
            ":created_at" => $tweet->created_at, ":id" => $tweet->id, ":text" =>
            $tweet->text, ":hashtags" => $tweet->hashtags, ":user" =>
            $tweet->user, ":retweet_count" => $tweet->retweet_count,
            ":favorite_count" => $tweet->favorite_count, ":lang" => $tweet->lang
        ]);
        $db = null; // Close connection
        return true;
      }
      catch (PDOException $exception)
      {
        echo $exception->getMessage();
      }
}
```

TwitterService_sql.php has a updateTweet function to insert tweets.

```
public function updateUser($user)
{
    try
    {
        $statement = $this->db->prepare("REPLACE INTO users (id, name,
        screen_name, location, description)
            VALUES (:id, :name, :screen_name, :location, :description)");
        $statement->execute([
            ":id" => $user->id, ":name" => $user->name, ":screen_name" =>
            $user->screen_name, ":location" => $user->location, ":description"
            => $user->description
        ]);

        $db = null; // Close connection
        return true;
      }
      catch (PDOException $exception)
      {
        echo $exception->getMessage();
      }
}
```

TwitterService_sql.php has a updateUser function to insert users.

tweets Examinar Estructura Buscar Insertar Vaciar Eliminar

created_at	id	text	hashtags	user	retweet_count	favorite_count	lang
Wed May 20 18:58:27 +0000 2020	1263182315603349505	We all just got trolled by an early #bitcoin miner...	[{"text":"bitcoin","indices":[36,44]}]	CharlieShrem	130	1648	en
Thu May 21 04:01:09 +0000 2020	1263318889703718912	So, we still haven't found Satoshi. And #bitcoin i...	[{"text":"bitcoin","indices":[40,48]},{"text":"onw...	cz_binance	189	1564	en
Thu May 21 06:49:16 +0000 2020	1263361198411575296	Imagine casually running a bitcoin (not even a wor...	[]	cz_binance	398	3025	en

users Examinar Estructura Buscar Insertar Vaciar Eliminar

id	name	screen_name ▲ 1	location	description
396045469	Barry Silbert	barrysilbert	New York	Founder/CEO @DCGco, parent of @GrayscaleInvest @Ge...
877807935493033984	Binance	binance		Exchange The World. Customer Service Enquiries: ...
22682896	Charlie Shrem	CharlieShrem	Sarasota, FL	ðŸŽ§Host & Storyteller https://t.co/tWQMV7zDkk ðŸ“...
928759224599040001	Bloomberg Crypto	crypto		A look at how cryptocurrencies and blockchain are ...

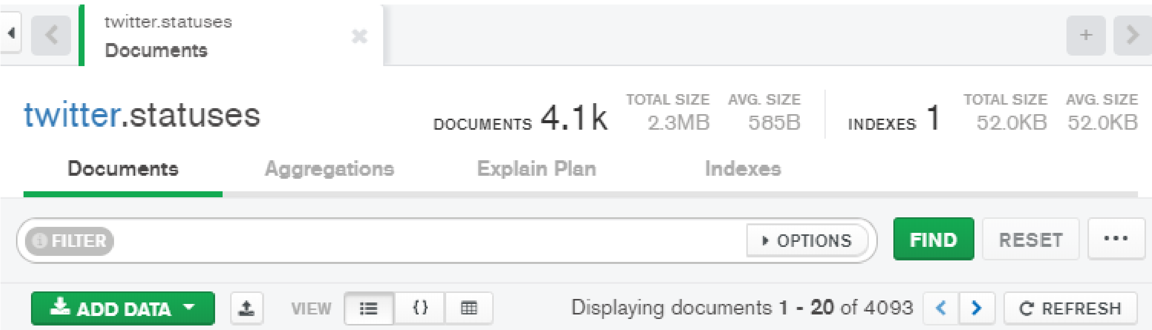
We can use any SQL database explorer to check the inserted data.

Inserting on MongoDB:

```php
public function updateStatuses($statuses)
{
    $bulk = new MongoDB\Driver\BulkWrite;

    foreach ($statuses as &$status)
    {
      $bulk->insert($status);
    }

    $this->manager->executeBulkWrite('twitter.statuses', $bulk);
}
```

TwitterService_mongodb.php has a updateStatuses function. Inserting to MongoDB is easy as it accepts JSON right away.

** Statuses contain both tweets and user details.*



We can use MongoDB compass to proof that our documents have been inserted.

```
_id: ObjectId("5ee4e4bba554830dcc008bdc")
created_at: "Thu May 30 06:29:08 +0000 2019"
id_str: "1133983664449003520"
text: "Soooo not the Russians Bill??"
hashtags: "[]"
screen_name: "CSigmaShow"
retweet_count: 0
favorite_count: 0
lang: "en"
user: Object
  id_str: "89570289"
  name: "#FreeJulianAssange"
  screen_name: "CSigmaShow"
  location: "The Land Down Under!"
  description: "Yank living Down Under. Silver/Gold Bull! Chelsea Supporter. On the Bi..."
```

Example of the fields of one document.

Inserting on Solr:

```
public function updateStatuses($statuses)
    {
        $xml = '<add>
        ';

        foreach ($statuses as &$status)
        {
            $xml .= '<doc>
            ';
            foreach ($status as $clave => $valor)
            {
                if( isset( $valor ) )
                {
                    $content = $valor;
                }
                else
                {
                    $content = 'null';
                }

                if( is_array($content) )
                {
                    foreach ($content as $uclave => $uvalor)
                    {
                        $xml .= '<field name="user_'.$uclave.'">'.htmlspecialchars($uvalor).'</field>
                        ';
                    }
                }
                else
                {
                    $xml .= '<field name="'.$clave.'">'.htmlspecialchars($content).'</field>
                    ';
                }

            }
            $xml .= '</doc>
            ';
        }

        $xml .= '</add>
        ';
}
```

TwitterService_solr.php has a updateStatuses function to create a valid XML.
** Statuses contain both tweets and user details.*

```
// Query Settings
$action = 'update';

// Building query
$url = $this->endpoint . '/solr/' . $this->collection . '/' . $action ;


// POSTing to SOLR API
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL,            $url );
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1 );
curl_setopt($ch, CURLOPT_POST,           1 );
curl_setopt($ch, CURLOPT_POSTFIELDS,     $xml );
curl_setopt($ch, CURLOPT_HTTPHEADER,     array('Content-Type: text/xml'));

$result=curl_exec ($ch);
```

Then, that XML is POSTed to the Solr endpoint using Curl.

```xml
        <?xml version="1.0" encoding="UTF-8"?>
<response>

<lst name="responseHeader">
  <int name="rf">1</int>
  <int name="status">0</int>
  <int name="QTime">3012</int>
</lst>
</response>
```

Response after inserting 62000+ XML lines http://localhost:8983/solr/twitter/select?q=*%3A*


```
{
  "responseHeader":{
    "zkConnected":true,
    "status":0,
    "QTime":9,
    "params":{
      "q":"*:*",
      "_":"1592382242602"}},
  "response":{"numFound":4099,"start":0,"maxScore":1.0,"docs":[
      {
        "created_at":["Thu May 30 06:30:00 +0000 2019"],
        "text":["RT @MdBdesGrauens: lol @nicosemsrott #maischberger zerstört https://t.co/pIPm06RhGH"],
        "hashtags":["[{\"text\":\"maischberger\",\"indices\":[37,50]}]"],
        "screen_name":["theissler"],
        "retweet_count":[0],
        "favorite_count":[0],
        "lang":["de"],
        "user_name":["Torsten Heissler 🌞"],
        "user_screen_name":["theissler"],
        "user_location":["Aschaffenburg, Bavaria"],
        "user_description":["Founder of http://jobboerse.com. Exit 2015 to XING SE. Interests: IT, IoT, DLT,
        "id":"04c1e701-c445-4c1a-b899-d5d0d6950326",
        "_version_":1669733621269790720},
      {
        "created_at":["Thu May 30 06:29:08 +0000 2019"],
        "text":["Soooo not the Russians Bill??"],
        "hashtags":["[]"],
        "screen_name":["CSigmaShow"],
        "retweet_count":[0],
        "favorite_count":[0],
        "lang":["en"],
        "user_name":["#FreeJulianAssange"],
        "user_screen_name":["CSigmaShow"],
        "user_location":["The Land Down Under!"],
        "user_description":["Yank living Down Under. Silver/Gold Bull! Chelsea Supporter. On the Bitcoin trai
        "id":"c101210d-c6eb-4e87-aec6-835fc9f2df19",
        "_version_":1669733668285841408},
      {
```


After more than 5 minutes after inserting, the tweets and user details are displayed. That's a long inserting time.

- **Query Data (pros and cons of the three databases) (3 points)**

Comparison Table:

	PostgreSQL / any SQL	**MongoDB**	**Solr**
HTTP Query API	No	No	Yes
Native Search Engine	No	No	Yes
SQL Queries	Yes	No	No
Performance on Searches	Worst	Average	Best
Query Complexity	Lowest	Highest	Medium
JSON Answers	No	Yes	Yes

Pros. and Cons. Table:

	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Pros.	Easy SQL syntax, Natural language feature	Fast, JSON	Faster, autogenerated indexes
Cons.	Not ideal for text search	Query complexity	Query Complexity

- **You should compare simple queries in three databases, analyze the different syntax and languages of the DBs (0,75)**

PostgreSQL:

```
$statement = $this->db->prepare("SELECT * FROM $table WHERE $collumn LIKE :string");
$statement->execute([ ":string" => "%".$string."%" ]);
$rows = $statement->fetchAll(PDO::FETCH_CLASS, $class);     // Load rows in array
```

MongoDB:

```
$options =
[
    'limit' => $limit,
    'sort' => [$order => 1]
];

$filter =
[
    'text' =>
    [
        '$regex' => $search
    ]
];

$query = new MongoDB\Driver\Query($filter, $options);

$rows = $this->manager->executeQuery('twitter.statuses', $query);
```

Solr:

```
// Query Settings
$action = 'select';
$q = '*:*';                 //Query
$wt = 'json';               //writer type, ouput format

// Building query
$q = urlencode($q);
$url = $this->endpoint . '/solr/' . $this->collection . '/' . $action . '?' . 'q=' . $q . '&wt=' . $wt ;

// Call the API
$json = file_get_contents($url);
$response = json_decode($json, true);
```

Comparison Table:

	PostgreSQL / any SQL	**MongoDB**	**Solr**
SQL	Yes	No	No
HTTP API	No	No	Yes
Returns JSON	No	Yes	Yes

- **Joins queries in three databases, for example make 2 tables in PostgreSQL, with tweets and other with users, make a join query to analyze, which user has more tweets with a join. Translate this example to MongoDB and Solr if it is possible (PROS and CONS) (0,75)**

PostgreSQL:

```
$statement = $this→db→prepare("SELECT users.screen_name, count(*) FROM users RIGHT
    JOIN tweets ON users.screen_name = tweets.user GROUP BY users.screen_name");
$statement→execute();
$rows = $statement→fetchAll();    // Load rows in array
```

MongoDB:

```
$command = new MongoDB\Driver\Command(
[
    'aggregate' ⇒ 'tweets',
    'pipeline' ⇒
    [
        ['$group' ⇒ ['_id' ⇒ '$user', 'count' ⇒ ['$sum' ⇒ 1]]],
        ['$match' ⇒ ['count' ⇒ ['$gt' ⇒ 1] ] ],
    ],
    'cursor' ⇒ new stdClass,
]);
$cursor = $this→manager→executeCommand('twitter', $command);
```

Solr:

Implemented in the PHP class.

Pros. and Cons. Table:

	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Pros.	Join queries	n/a	n/a
Cons.	n/a	No Join	No Join

- **Aggregate Queries in three databases, try to use Aggregate from MongoDB in the other DBs, is it possible? PROS and CONS (0,75)**

PostgreSQL:

```
$statement = $this→db→prepare("SELECT users.screen_name, count(*) FROM users RIGHT
    JOIN tweets ON users.screen_name = tweets.user GROUP BY users.screen_name");
$statement→execute();
$rows = $statement→fetchAll();    // Load rows in array
```

MongoDB:

```
$command = new MongoDB\Driver\Command(
[
    'aggregate' ⇒ 'tweets',
    'pipeline' ⇒
    [
        ['$group' ⇒ ['_id' ⇒ '$user', 'count' ⇒ ['$sum' ⇒ 1]]],
        ['$match' ⇒ ['count' ⇒ ['$gt' ⇒ 1] ] ],
    ],
    'cursor' ⇒ new stdClass,
]);
$cursor = $this→manager→executeCommand('twitter', $command);
```

Solr:
Facet

Comparison Table:

	PostgreSQL / any SQL	**MongoDB**	**Solr**
Implementation	Group by & Count(*)	Aggregate & Pipelines	Facets

Pros. and Cons. Table:

	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Pros.	Simple	n/a	Simple
Cons.	n/a	Complex	n/a

- **Map reduce Queries in three databases, try to use MapReduce from MongoDB in the other DBs, is it possible? PROS and CONS (0,75)**

<u>Mapreduce</u>

"MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a map procedure, which performs filtering and sorting (such as sorting students by first name into queues, one queue for each name), and a reduce method, which performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance." [33]

Comparison Table:

	PostgreSQL / any SQL	**MongoDB**	**Solr**
Implementation	No	mapReduce [34]	SolrReducer [36]
Possible Solution	Clustering & query distribution managed by the application [35]	n/a	n/a

Pros. and Cons. Table:

	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Pros.	No need to understand mapreduce as it is not supported	Easy to query big quantities of data in a cluster	Easy to search big quantities of data in a cluster
Cons.	No mapreduce	Complex syntax	Complex syntax

MongoDB:

```
public function demoMapReduce($statuses)
{
    $map = new MongoCode('function() {
        var total = 0;
        for (count in this.news) {
        total +=  this.tweet[count];
        }
        emit(this._id, {id: this.id, total: total});
    }');

    $reduce = new MongoCode('function(key, values) {
            var result = {id: null, total: 0};
            values.forEach(function(v) {
            result.id = v.id;
            result.total = v.total;
            });
            return result;
    }');

    $totals = $this->manager->$db->command(array(
        'mapreduce' => 'statuses', // collection name
        'map' => $map,
        'reduce' => $reduce,
        'query' => array('tweet' => 'user'),
        "out" => "totals" // new collection name
    ));
}
```

This mapReduce concept query explains the basics of how to define a mapReduce process on PHP that creates a new collection called totals with the count of tweets per user.

The ideal approach would be to try to execute this on MongoDB atlas cluster but due to the extension of the task and the deadline this was not tested so its here only for reference.

- **Search Text using the power of the different databases, Queries in the three databases (0,5).**

PostgreSQL:

```
private function queryNaturalLanguage($string, $table, $class): array
{

    if($table == 'tweets')
    {
        $match = 'text,hashtags,user';
    }else if($table == 'users')
    {
        $match = 'name,screen_name,location,description';
    }

    try
    {
        if($string == ''){
            $statement = $this->db->prepare("SELECT * FROM $table");
        }else{
            $statement = $this->db->prepare("SELECT * FROM $table WHERE MATCH ($match) AGAINST (:string IN
            NATURAL LANGUAGE MODE)");
        }
        $statement->execute([ ":string" => "%".$string."%" ]);
        $rows = $statement->fetchAll(PDO::FETCH_CLASS, $class);    // Load rows in array
        $db = null;      // Close connection

        //print_r($statement);

        return $rows;
    }
    catch (PDOException $exception)
    {
        echo $exception->getMessage();
    }
}
```

MongoDB:

```
public function getTweetsBySearch($limit = 100, $order = null, $search = null): array
{
    $options =
    [
        'limit' => $limit,
        'sort' => [$order => 1]
    ];

    $filter =
    [
        'text' =>
        [
            '$regex' => $search
        ]
    ];
    return $this->queryTweets($filter, $options);
}
```

Solr:

```
public function getTweetsBySearch($limit = null, $order = null, $search = ''): array
{
    // Query Settings
    $action = 'select';
    $q = 'text:'.$search.'*';   //Query
    $wt = 'json';           //writer type, ouput format
    $rows = 1000;

    // Building query
    $q = urlencode($q);
    $url = $this->endpoint . '/solr/' . $this->collection . '/' . $action . '?' . 'q=' . $q . '&wt=' . $wt . '&rows=' . $rows ;

    // Call the API
    $json = file_get_contents($url);
    $response = json_decode($json, true);
    $rows = $response['response']['docs'];

    // Print
    //print_r($response);

    $tweets = array();

    foreach ($rows as $row) {
        $tweet = Tweet::construct(
            $row['created_at'][0],
            $row['id'],
            $row['text'][0],
            $row['hashtags'][0],
            $row['screen_name'][0],
            $row['retweet_count'][0],
            $row['favorite_count'][0],
            $row['lang'][0]
        );

      $tweets[] = $tweet;
    }

    return $tweets;
}
```

Comparison Table:

	PostgreSQL / any SQL	**MongoDB**	**Solr**
Implementation	NATURAL LANGUAGE MODE [37]	$text [38]	Native [39]
Performance	Worst	Medium	High
Designed to search text	No	No	Yes

- **You should modify the index of the different databases in order to improve the performance of the database (0,2)**

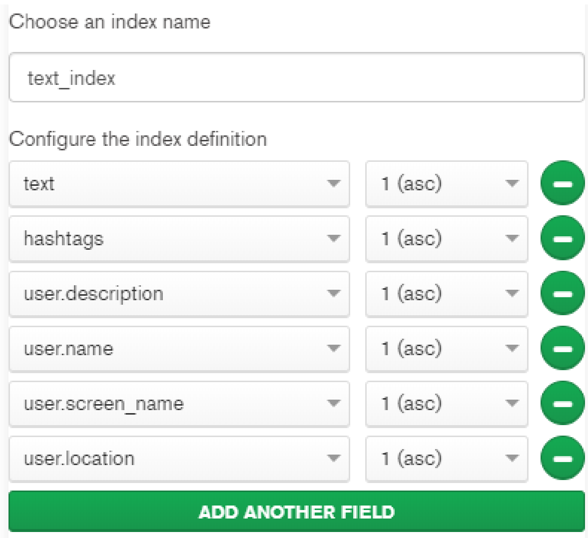
PostgreSQL:

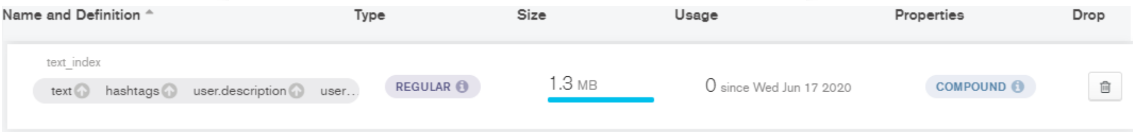
Create index [40]

```
ALTER TABLE tweets ADD FULLTEXT
index_tweetsall (text,hashtags,user);

ALTER TABLE users ADD FULLTEXT
index_usersall (name,screen_name,location,description);

SELECT * FROM tweets
    WHERE MATCH (text,hashtags,user)
    AGAINST ('crypto' IN NATURAL LANGUAGE MODE);

SELECT * FROM users
    WHERE MATCH (name,screen_name,location,descriptio)
    AGAINST ('crypto' IN NATURAL LANGUAGE MODE);
```

MongoDB:

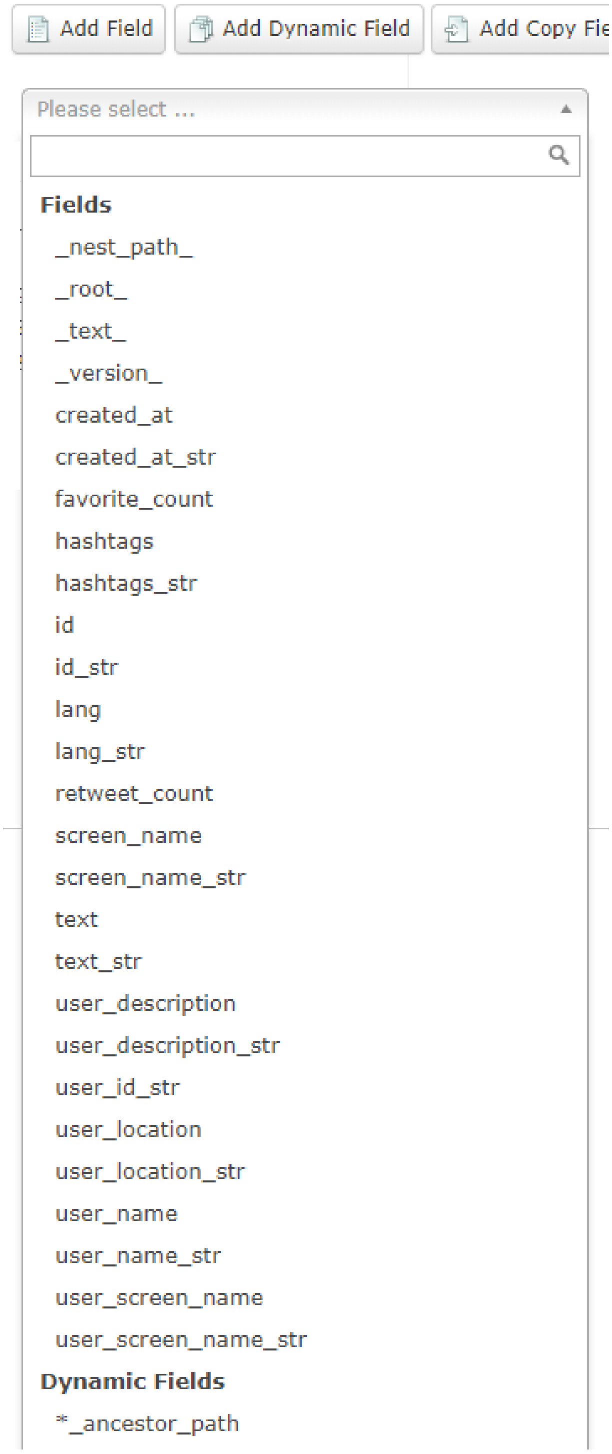
Indexes [41]





Solr:

Indexes [42]

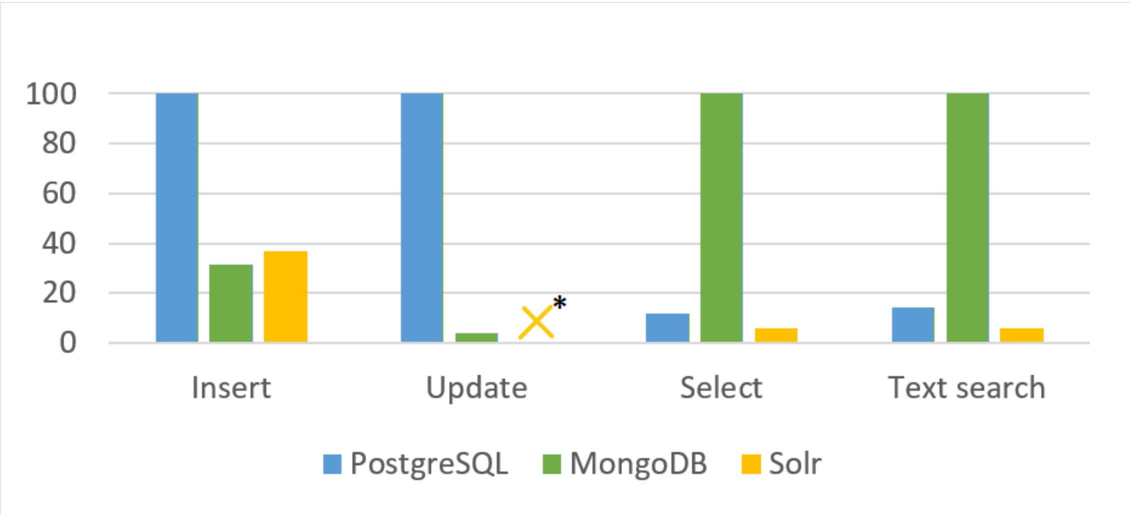


Executive summary

With the preceding study and implementations, we can appreciate the following findings: **Solr** is the best approach to implement a search engine due to its performant engine and transformation contraptions like stemming, filters and more. **PostgreSQL** is the best approach for data integrity, but has the slowest insertions and updates. **MongoDB** is the best approach for fast updates and distributed real-time systems but not suitable for searches and bad for complex selections.

In our opinion, a hybrid approach that combines the three engines would be perfect for this kind of application. **PostgreSQL** can be used to store app users, settings and final reports for integrity and consistency, **MongoDB** can be used to store the Tweets in real-time using the **Twitter Streaming API** and then **Solr** every night indexes all the tweets and is used for the main purpose of the application, the search engine and faceting.

Performance comparison (lower is better performance):



Source: own elaboration based on [48], [49], [50] and [51] benchmark results and normalizing the data on a 0 to 100 scale being 100 the slowest result.
** Solr does not have update performance because is not a supported operation.*

Full comparison table:

	***PostgreSQL* / any SQL**	***MongoDB***	***Solr***
Main usage	Relational DBMS	Document store	Search engine
Popularity	#1 Most popular	#2 Less popular	#3 Niche
Language	C	C++	Java
Scheme	Yes	schema-free	Yes
MapReduce	No	Yes	Spark-solr
Consistency	Immediate	Eventual / Immediate	Eventual
Foreign Keys	Yes	No	No
Transactions	ACID	Multi-document ACID, snapshot isolation	Optimistic locking
Integrity Model	ACID	BASE	Other
Isolation	Yes	No	No
Referential Integrity	Yes	No	No
Full text search	Yes	Yes	Yes
Facet Implementation	Group By & Count(*) [10]	$facet (aggregation) [11]	Facet [12]
Boolean Expressions	AND, OR and NOT [13]	$and, $not, $nor, $or [14]	AND, OR and NOT [15]
Categories Implementation	WHERE [16]	$where [17]	Fields [15]
Schema rigidity	High	None	Medium
Tables	Yes	No	No
Relations	Yes	No	No
Typed fields	Yes	No	Yes
Basic units	Databases & Tables	DBs & Collections	Collections
Document Upload	Not directly	JSON	XML, JSON, CSV…
Concurrency	MVCC [19]	multi-granularity locking [20]	Atomic updates [21], in-place updates [22], optimistic concurrency [23]

Transactions	Yes, [24]	Yes, in 4.0+ [25]	No [26]
ACID	Yes, [27]	Kind of, [28]	No [29]
BASE	No	Yes	No
Cap Theorem	Avail. & Consist.	Consist. & Part. Tol	Somewhat Consist. & Part. Tol [30]
Insert	SQL INSERT	db.collection.insert()	API /solr/update/json
Inserted Data	Tweets & Users separated	Statuses (combination)	Statuses (combination)
HTTP Query API	No	No	Yes
Native Search Engine	No	No	Yes
SQL Queries	Yes	No	No
Performance on Searches	Worst	Average	Best
Query Complexity	Lowest	Highest	Medium
JSON Answers	No	Yes	Yes
SQL	Yes	No	No
HTTP API	No	No	Yes
Returns JSON	No	Yes	Yes
"faceting"	Group by & Count(*)	Aggregate & Pipelines	Facets
"mapReduce"	No	mapReduce [34]	SolrReducer [36]
Best Text Search	NATURAL LANGUAGE MODE [37]	$text [38]	Native [39]
Performance	Worst	Medium	High
Designed to search text	No	No	Yes

Sources were referenced in the tables in sections above.

Conclusion

With this extensive work, we have researched, analyzed and implemented a working web application in the form of a Minimum Viable Product (MVP) that fulfills all the requirements including but not limited to data acquisition using Twitter API or tweet dumps in JSON files; data inserting into three databases; data analysis including searching, faceting, and more; performance analysis of the queries with a millisecond counter; visual representation in nice tables; visual representation in word clouds; full text searches in all three databases with nice interface and sorting.

For the theory section, we have analyzed the differences between SQL databases, noSQL databases and search engines, detailing pros. and cons. in each application.

The new knowledge acquired was an extensive understanding to the three databases and its practical implementation in a PHP application with a real problem.

Some skills were already acquired during classes but have been obviously reinforced by this work.

Some collaboration with the teacher took place in order to fully comprehend the extension of the task and concrete details.

An exact number of hours dedicated to this final task cannot be assessed as we have dedicated more than 30 days to it, but this effort can be somehow measured by looking at the PHP code lines, the words embodied on this document, its bibliography references, and by talking personally with me.

Bibliography

[01] Guía de estudio 3.1 - Inglés.pdf

[02] https://archive.org/details/archiveteam-twitter-stream-2019-05

[03] https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets

[04] https://github.com/J7mbo/twitter-api-php

[05] https://www.php.net/manual/es/pdo.drivers.php

[06] https://db-engines.com/en/system/MongoDB%3BPostgreSQL%3BSolr

[07] https://spiderpig86.github.io/Cirrus/

[08] https://www.php.net/manual/es/mongodb.installation.windows.php

[09] https://vanya.jp.net/vtree/

[10] https://www.w3schools.com/sql/sql_groupby.asp

[11] https://docs.mongodb.com/manual/reference/operator/aggregation/facet/

[12] https://lucene.apache.org/solr/guide/6_6/faceting.html

[13] https://www.w3schools.com/sql/sql_and_or.asp

[14] https://docs.mongodb.com/manual/reference/operator/query/#logical

[15] https://lucene.apache.org/solr/guide/6_6/the-standard-query-parser.html

[16] https://www.w3schools.com/sql/sql_where.asp

[17] https://docs.mongodb.com/manual/reference/operator/query/where/#example

[18] https://lucene.apache.org/solr/guide/6_6/the-standard-query-parser.html#TheStandardQueryParser-SpecifyingFieldsinaQuerytotheStandardQueryParser

[19] https://devcenter.heroku.com/articles/postgresql-concurrency

[20] https://docs.mongodb.com/manual/faq/concurrency/#mgl-ref

[21] https://lucene.apache.org/solr/guide/6_6/updating-parts-of-documents.html#UpdatingPartsofDocuments-AtomicUpdates

[22] https://lucene.apache.org/solr/guide/6_6/updating-parts-of-documents.html#UpdatingPartsofDocuments-In-PlaceUpdates

[23] https://lucene.apache.org/solr/guide/6_6/updating-parts-of-documents.html#UpdatingPartsofDocuments-OptimisticConcurrency

[24] https://www.postgresql.org/docs/8.3/tutorial-transactions.html

[25] https://docs.mongodb.com/manual/core/transactions/

[26] https://stackoverflow.com/a/12971674

[27] https://www.juancarlosmoral.es/postgresql-acid-test/

[28] https://techcrunch.com/2018/02/15/mongodb-gets-support-for-multi-document-acid-transactions/

[29] https://lucidworks.com/post/solr-and-rdbms-the-basics-of-designing-your-application-for-the-best-of-both/

[30] http://people.apache.org/~yonik/presentations/solr4_nosql_apachecon_2012.pdf

[31] https://www.dummies.com/programming/big-data/hadoop/acid-versus-base-data-stores/

[32] http://blog.mattcallanan.net/2012/10/nosql-acid-vs-base.html

[33] https://en.wikipedia.org/wiki/MapReduce

[34] https://docs.mongodb.com/manual/core/map-reduce/

[35] https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling

[36] https://lucene.apache.org/solr/6_0_1/solr-map-reduce/org/apache/solr/hadoop/SolrReducer.html

[37] https://dev.mysql.com/doc/refman/8.0/en/fulltext-natural-language.html

[38] https://docs.mongodb.com/manual/text-search/

[39] https://lucene.apache.org/solr/guide/7_3/overview-of-searching-in-solr.html

[40] https://www.postgresql.org/docs/9.1/sql-createindex.html

[41] https://docs.mongodb.com/manual/indexes/

[42] https://lucene.apache.org/solr/guide/6_6/introduction-to-solr-indexing.html

[43] https://wordcloud2-js.timdream.org/

[44] https://markjs.io/

[45] https://pgdash.io/blog/scaling-postgres.html

[46] https://docs.mongodb.com/manual/sharding/

[47] https://lucene.apache.org/solr/guide/6_6/introduction-to-scaling-and-distribution.html

[48] https://www.simform.com/mongodb-vs-mysql-databases/

[49] https://www.dbbest.com/blog/lucene-vs-sql-server-fts/

[50] https://www.palepurple.co.uk/full-text-searching-with-solr

[51] http://www.openwebspider.org/2015/04/03/full-text-search-mysql-vs-mongodb-vs-sphinx/