Caching PostgreSQL with Redis

Goal
Discover different caching strategies with the Redis key-value database system

set up ans starting redis
You download the Redis system from the redis.io site (a version 6 or 7). Follow the installation instructions (no need for special options).
In the src / directory, start the server with the command ./redis-server. Launch the client in another console with the command ./redis-cli (always from the src directory). The customer's invitation tells us that the port used by redis is 6379.
First queries
Redis est un key/value store. La granularité de ce que l 'on peut mettre à la position valeur
Redis is a key / value store. The granularity of what we can put at the value position can vary depending on the data structure used.
In a first test, we limit ourselves to a value corresponding to a string of characters.
To record a new key/value pair: `set user:1 "joe"`
Here, user:1 is tehe key and "joe" is the value.
To get the value of the user1 key : : `get user:1`
By executing a get on a key not found in the database, for example get user: 2, we get :
`(nil)`

We can use a hash structure in a value

```
HSET user:1000 name "John Smith"
HSET user:1000 email "john.smith@example.com"
HSET user:1000 password "s3cret"
and get all the pairs of a key withHGETALL user:1000
```
With HMSET, we register several pairs of the hash in a single command: `HMSET user:`
```
1001 name "Mary Jones" password "hidden" email "mjones@example.com"
HGETALL user:1001
1) "name"
2) "Mary Jones"
3) "password"
4) "hidden"
5) "email"
6) "mjones@example.com"
```

Terminology
A **cache** is an application or terminal's memory allocation for the temporary storage of data that will likely be accessed soon. The objective is to have a performance gain for the next accesses to the cached data.
In-memory Key / value databases (for example, memcached or redis) are frequently used to support a caching strategy.
When reading data, the most common strategy is to try to get the data from the cache. If it is there, it is called **cache-hit**.

Otherwise, we name the situation of **cache-miss**, and we must then read from the database (for example on an RDBMS) and store it in the cache.
The objective for the system to be efficient is to have a good cache-hit ratio compared to the cache-miss, that is to say that we mainly obtain the data sought from the cache and not from the RDBMS. This report is called the **cache-ratio**.

First steps with Jedis
There are several Java clients for handling Redis (Redisson, Lettuce). We will be working with jedis which is relatively light, fast, has been adopted by the Spring framework and has a large community.
To familiarize yourself with Jedis, you create a java project allowing you to store 3 maps of persons (id, name and first name) and to read the values of all the persons in the database.
You opt for a Maven project with the following dependency:

```
    <dependency>
        <groupId>redis.clients</groupId>
        <artifactId>jedis</artifactId>
        <version>3.1.0</version>
    </dependency>
```

You are using a Jedis pool with a default connection:

```
    private JedisPool pool = new JedisPool(new JedisPoolConfig(),"localhost");
```

Caching
You are using Redis as the Data Cache from last lab assignment's denormalized database. The cache must contain cip7 as a key (in the form “drug: cip7”) and the values correspond to the drug name and the cis code. You will put 1000 pairs in your Redis database.

To access the Postgresql database containing the drug database, you use JDBC by adding the following dependency in your pom.xml:

```
<dependency>
    <groupId>postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>9.1-901-1.jdbc4</version>
</dependency>
```

Conduct a performance test (duration from connection creation to closing the connection to the data source) between Postgresql and Redis accesses for a drug that you know is in the cache.

Update the cache
During a cache-miss, you must update the Cache by inserting the data from the PostgreSQL database. You verify that the Cache has been updated after this operation (see DBSIZE from the console of the Redis client).

Stratégie lors de l’écriture
A new data has just been inserted into the PostgreSQL database. We want to store it in the Cache but it is full (it's up to you to simulate this situation in your code - for example by allowing a certain number of entries in the Cache). Cache entries must be deleted to integrate

more relevant data. The most common strategy in this situation is **LRU** (Least Recently Used). The idea is to delete the least used data from the Cache.
You must make the right choice among the structures offered by Redis to manage the counting of accesses to the different keys of the Cache.

Few Redis commands
DBSIZE to know the number of keys in the database
FLUSHDB to delete all the keys from the database