Exercise 1: Database Handling Using Lists

Write a module **db.erl** that creates a database and is able to store, retrieve and delete elements in it. The function **destroy/1** will delete the database. Considering that Erlang has garbage collection, you do not need to do anything. Had the db module however stored everything on file, you would delete the file. We are including the destroy function so as to make the interface consistent. You may not use the **lists** library module, and have to implement all the recursive functions yourself.

Hint: Use lists and tuples your main data structures. When testing your program, remember that Erlang variables are single assignment.

Interface:

```
db:new() ⇒ Db.
db:destroy(Db) ⇒ ok.
db:write(Key, Element, Db) ⇒ NewDb.
db:delete(Key, Db) ⇒ NewDb.
db:read(Key, Db) ⇒{ok, Element} | {error, instance}.
db:match(Element, Db) ⇒ [Key1, ..., KeyN].
```

Example:

```
1> c(db).
{ok,db}
2>  Db = db:new().
[]
3>  Db1 = db:write(francesco, london, Db).
[{francesco,london}]
4> Db2 = db:write(lelle, stockholm, Db1).
[{lelle,stockholm},{francesco,london}]
5> db:read(francesco, Db2).
{ok,london}
6>  Db3 = db:write(joern, stockholm, Db2).
[{joern,stockholm},{lelle,stockholm},{francesco,london}]
7>  db:read(ola, Db3).
{error,instance}
8>  db:match(stockholm, Db3).
[joern,lelle]
9>  Db4 = db:delete(lelle, Db3).
[{joern,stockholm},{francesco,london}]
10> db:match(stockholm, Db4).
[joern]
11>
```

Note: Due to single assignment of variables in Erlang, we need to assign the updated database to a new variable every time.

Exercise 2: A Database Server

Write a database server that stores a database in its loop data. You should register the server and access its services through a functional interface. Exported functions in the **my_db.erl** module should include:

Interface

```
my_db:start() ⇒ ok.
my_db:stop() ⇒ ok.
my_db:write(Key, Element) ⇒ ok.
my_db:delete(Key) ⇒ ok.
my_db:read(Key) ⇒ {ok, Element} | {error, instance}.
my_db:match(Element) ⇒ [Key1, ..., KeyN].
```

Hint: Use the db.erl module as a back end and use the server skeleton from the echo exercise.

Example

```
1> my_db:start().
ok
2> my_db:write(foo, bar).
ok
3> my_db:read(baz).
{error, instance}
4> my_db:read(foo).
{ok, bar}
5> my_db:match(bar).
[foo]
```

Exercise 3: Database Handling Using Records

Take the db.erl module you wrote in exercise 5 in the sequential programming exercises. Rewrite it using records. Test it using your database server you wrote in exercise 4 of the concurrent programming. As a record, you could use the following definition. Remember to place it in an include file.

Record Definition: -record(data, {key, data}).

Note: Make sure you save a copy of your **db.erl** module using lists somewhere else (or with a new name) before you start changing in.

Exercise 4: Database Handling using ETS

Take the db.erl module from exercise 1, and rewrite it using ets tables instead of operations on lists. Use records to store your data in the ets tables. If you are having problems, use the table visualizer tool to debug your code.

Test your back end module using your database server you wrote in exercise 1 of the process design patterns.

Note: Just in case you still need that reminder. Make sure you save a copy of your db.erl module using lists somewhere else (or with a new name) before you start changing in.