

Petshop

Tenche Eduard & Tuslea Andreea

User Manual:

Legend:

- [CLI Manual](#)
- [REST Manual](#)

CLI Manual

The Command Line Interface can be used to interact with your database directly from the Java application.

To get started, simply run the application.

From the command line interface, you can input different commands to interact with the database. The commands are not case sensitive.

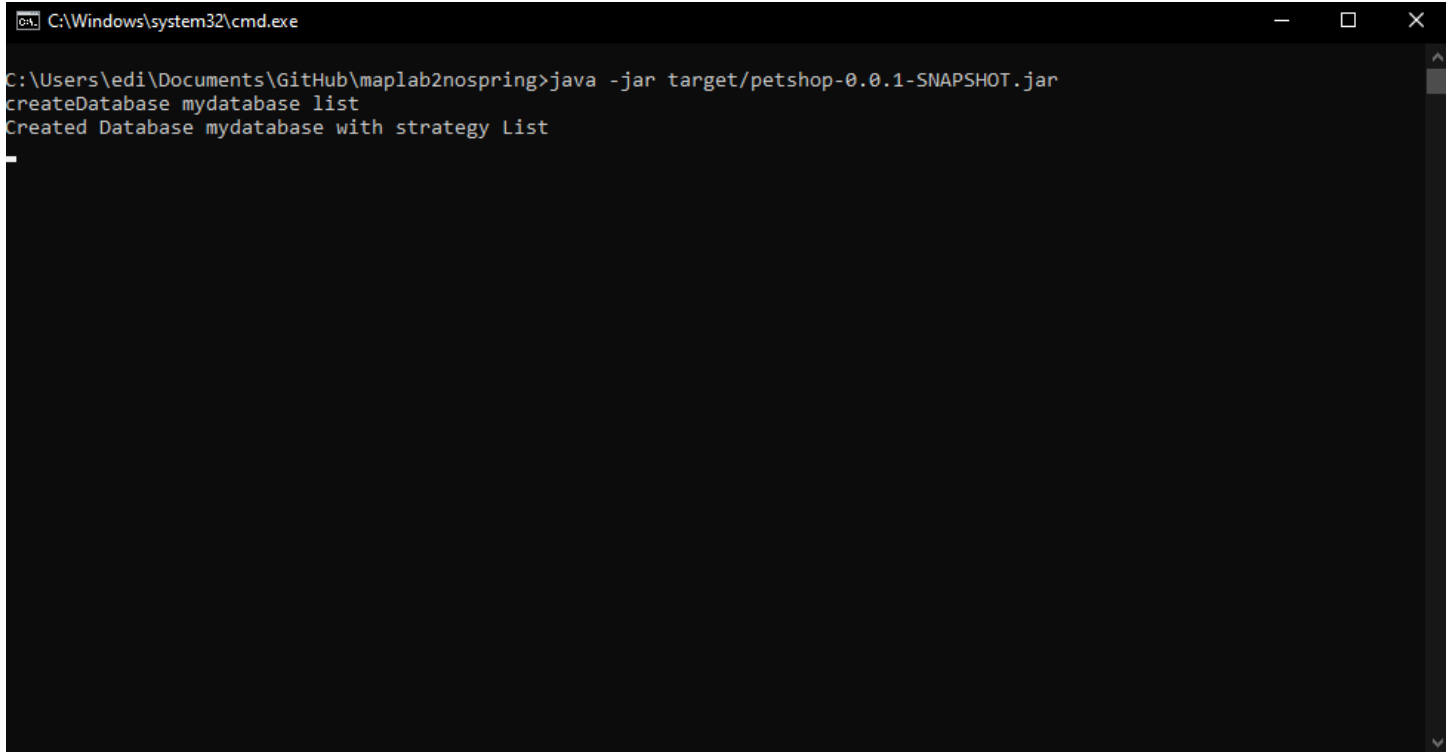
Legend:

- [Creating An In-Memory Database](#)
- [Connecting to a MySQL Database](#)
- [Adding tables to a Database](#)
- [Viewing Tables in a Database](#)
- [Removing tables from a Database](#)
- [Other commands](#)

Creating an In-Memory Database

You can create an In-Memory Database by using the command “createDatabase”. It requires the name and type of the database as arguments. The name can be whatever you want, the type has to be one of the following:

- List

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\system32\cmd.exe". The command prompt shows the following text:

```
C:\Users\edi\Documents\GitHub\maplab2nospring>java -jar target/petshop-0.0.1-SNAPSHOT.jar  
createDatabase mydatabase list  
Created Database mydatabase with strategy List
```

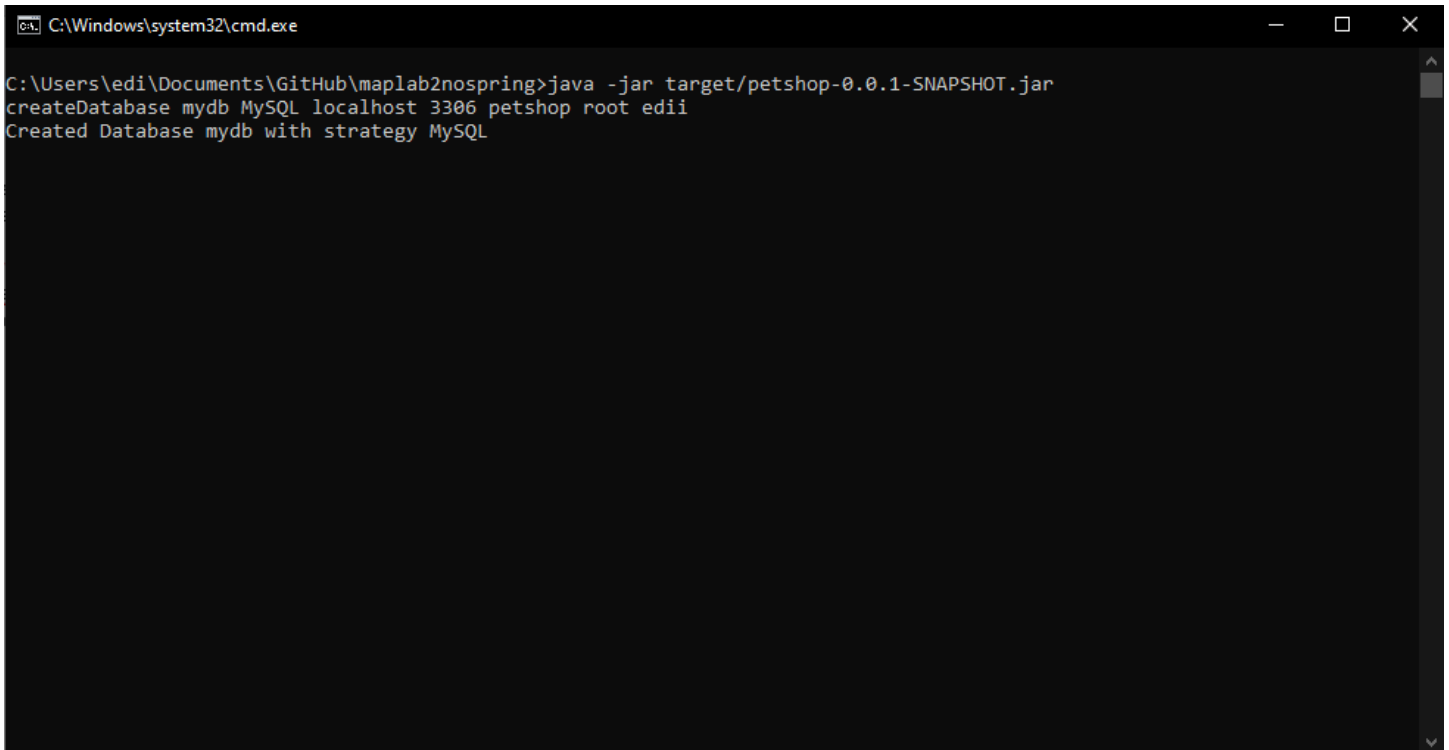
The text is displayed in a monospaced font on a dark background. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Connecting to a MySQL Database

In order to connect to a MySQL Database, you need to also use the `createDatabase` command. This time the arguments needed are:

```
createDatabase [localDatabaseName] MySQL [address] [port] [MySQLDatabaseName] [username] [password]
```

[localDatabaseName] is just the name that you will be using within the CLI, and does not have to match the name of the database in the MySQL Server.



```
C:\Windows\system32\cmd.exe

C:\Users\edi\Documents\GitHub\maplab2nospring>java -jar target/petshop-0.0.1-SNAPSHOT.jar
createDatabase mydb MySQL localhost 3306 petshop root edii
Created Database mydb with strategy MySQL
```

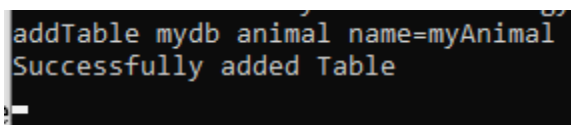
Adding tables to a Database:

You can add tables to a database using the `addTable` command. The required arguments are as follows:

```
addTable [databaseName] [tableName] [further arguments].
```

In [further arguments], you can specify the attributes of the table in the format [attributeName]=[value]. If you know the order the attributes are declared in, you can also use that instead of typing out the attribute name by just entering the values directly.

For example:



```
addTable mydb animal name=myAnimal
Successfully added Table
```

You can see all the tables and their attributes in the [Tables](#) section of the class reference.

Viewing tables in a Database:

You can view tables using the table command. The required arguments are as follows:

`table [databaseName] [tableName]`

```
table mydb animal
animal with Primary Keys: idAnimal(Integer) = 27, Fields: idAnimal(Integer) = 27, name(String) = pompom1, date(Date) = 1980-10-10, animalType(Integer) = null,
animal with Primary Keys: idAnimal(Integer) = 29, Fields: idAnimal(Integer) = 29, name(String) = null, date(Date) = null, animalType(Integer) = null,
animal with Primary Keys: idAnimal(Integer) = 30, Fields: idAnimal(Integer) = 30, name(String) = null, date(Date) = null, animalType(Integer) = null,
animal with Primary Keys: idAnimal(Integer) = 31, Fields: idAnimal(Integer) = 31, name(String) = null, date(Date) = null, animalType(Integer) = null,
animal with Primary Keys: idAnimal(Integer) = 33, Fields: idAnimal(Integer) = 33, name(String) = pompom1, date(Date) = 1980-01-01, animalType(Integer) = 1,
animal with Primary Keys: idAnimal(Integer) = 61, Fields: idAnimal(Integer) = 61, name(String) = null, date(Date) = null, animalType(Integer) = null,
animal with Primary Keys: idAnimal(Integer) = 64, Fields: idAnimal(Integer) = 64, name(String) = null, date(Date) = null, animalType(Integer) = null,
animal with Primary Keys: idAnimal(Integer) = 71, Fields: idAnimal(Integer) = 71, name(String) = pompom1, date(Date) = 1980-01-01, animalType(Integer) = 1,
animal with Primary Keys: idAnimal(Integer) = 78, Fields: idAnimal(Integer) = 78, name(String) = myAnimal, date(Date) = null, animalType(Integer) = null,
```

In a MySQL database, you can also execute queries using the execQuery command.

`execQuery [databaseName] [query]`

```
execQuery mydb SELECT idAnimal, name FROM animal
idAnimal | name |
27 | pompom1 |
29 | null |
30 | null |
31 | null |
33 | pompom1 |
61 | null |
64 | null |
71 | pompom1 |
78 | myAnimal |
```

Removing tables from a Database:

You can remove tables from a Database using the removeTable command. The syntax is similar to adding tables.

`removeTable [databaseName] [tableName] [further arguments]`

This command will remove any tables matching the [further arguments] argument. If it is left blank, it will remove all tables of the given type.

```
removeTable mydb animal name=pompom1
Removing tables
execQuery mydb SELECT idAnimal, name FROM animal
idAnimal | name |
29 | null |
30 | null |
31 | null |
61 | null |
64 | null |
78 | myAnimal |
```

Other commands:

To quickly check if a database contains a table of any type, you can use the hasTable command:

```
hasTable [databaseName] [tableName]
```

```
hasTable mydb animaltype
True
```

When an exception is thrown, by default, the output contains the full stack trace. You can disable this with the toggleStackTrace command.

```
toggleStackTrace [true/false]
```

Leaving the true/false field empty will just toggle the stack trace from true to false depending on what it already is.

```
removeTable mydb a
Could not invoke command
java.lang.reflect.InvocationTargetException
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:568)
    at com.eax.petshop.classes.UI.CLI.invoke(CLI.java:46)
    at com.eax.petshop.classes.UI.CLI.start(CLI.java:66)
    at com.eax.petshop.PetshopApplication.main(PetshopApplication.java:46)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:568)
    at org.springframework.boot.loader.launch.Launcher.launch(Launcher.java:91)
    at org.springframework.boot.loader.launch.Launcher.launch(Launcher.java:53)
    at org.springframework.boot.loader.launch.JarLauncher.main(JarLauncher.java:58)
Caused by: java.lang.NullPointerException: Cannot read the array length because "<local4>" is null
    at com.eax.petshop.classes.base.DatabaseProxy.removeTable(DatabaseProxy.java:67)
    at com.eax.petshop.classes.controllers.Controller.removeTable(Controller.java:112)
    at com.eax.petshop.classes.UI.CLI.removeTable(CLI.java:173)
    ... 14 more
toggleStackTrace
No longer printing full Stack Trace
removeTable mydb a
Could not invoke command
```

In order to exit the CLI, simply use the exit command.

```
exit [exitCode]
```

Exiting with code 1 will start the REST Service.

REST Manual

The REST Service allows you to interact with your database using applications such as Postman.

Legend:

- [Getting Started](#)
- [Creating an In-Memory Database](#)
- [Connecting to a MySQL Database](#)
- [Adding Tables to a Database](#)
- [Viewing Tables in a Database](#)
- [Removing Tables from a Database](#)

Getting Started

Simply run the application, and exit the Command Line Interface using the command 'exit 1'. This will boot up the REST Service.

```
C:\Windows\system32\cmd.exe
C:\Users\edi\Documents\GitHub\maplab2nospring>java -jar target/petshop-0.0.1-SNAPSHOT.jar
exit 1
Exiting CLI with code 1...

      .___.
     /___ \
    /   _ \
   /___  \
  /   ___ \
 /_____/ 
:: Spring Boot ::                (v3.2.0)

2024-01-12T15:56:54.206+02:00 INFO 12776 --- [          main] com.eax.petshop.PetshopApplication : Starting PetshopApplication v0.0.1-SNAPSHOT using Java 17.0.9 with PID 12776 (C:\Users\edi\Documents\GitHub\maplab2nospring\target\petshop-0.0.1-SNAPSHOT.jar started by edi in C:\Users\edi\Documents\GitHub\maplab2nospring)
2024-01-12T15:56:54.211+02:00 INFO 12776 --- [          main] com.eax.petshop.PetshopApplication : No active profile set, falling back to 1 default profile: "default"
2024-01-12T15:56:55.045+02:00 INFO 12776 --- [          main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2024-01-12T15:56:55.055+02:00 INFO 12776 --- [          main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-01-12T15:56:55.055+02:00 INFO 12776 --- [          main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.16]
2024-01-12T15:56:55.102+02:00 INFO 12776 --- [          main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-01-12T15:56:55.103+02:00 INFO 12776 --- [          main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext initialization completed in 810 ms
2024-01-12T15:56:55.503+02:00 INFO 12776 --- [          main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''
2024-01-12T15:56:55.520+02:00 INFO 12776 --- [          main] com.eax.petshop.PetshopApplication : Started PetshopApplication in 1.711 seconds (process running for 4.507)
```

Creating an In-Memory Database

You can create an In-Memory database just like with the CLI using the same command.

GET

localhost:8080/createDatabase?dbName=mydb&type=List

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	dbName	mydb			
<input checked="" type="checkbox"/>	type	List			
<input type="checkbox"/>	arguments				
	Key	Value	Description		

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 139 ms

Size: 204 B

Save as example

Pretty

Raw

Preview

Visualize

Text

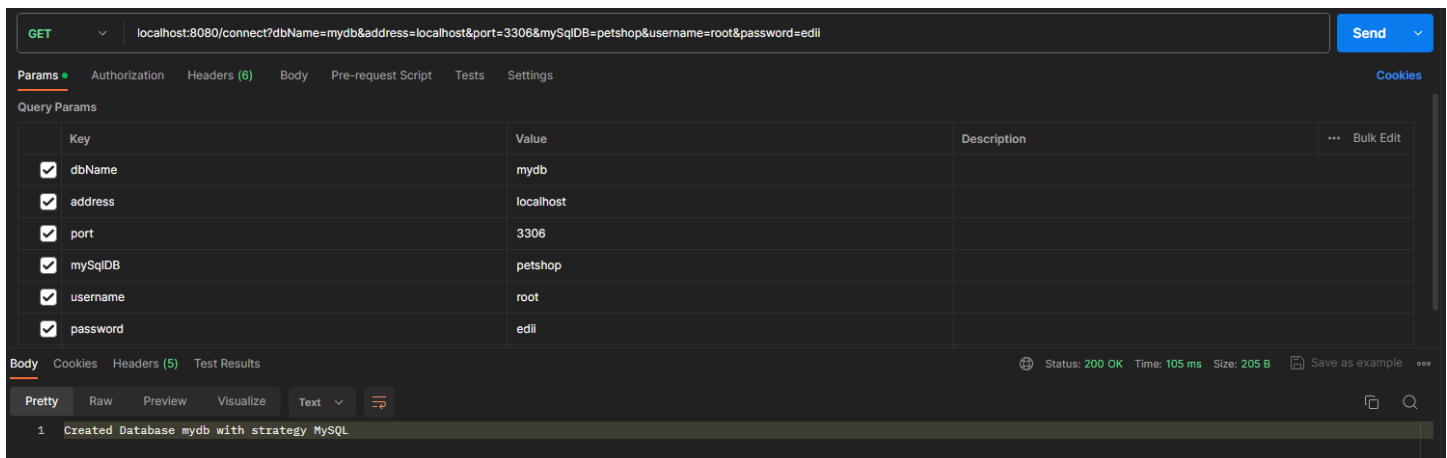
1

Created Database mydb with strategy List

Connecting to a MySQL Database

You can connect to a MySQL Database using the same syntax as with a CLI, and adding the ‘arguments’ Parameter, in which you specify the same things as before.

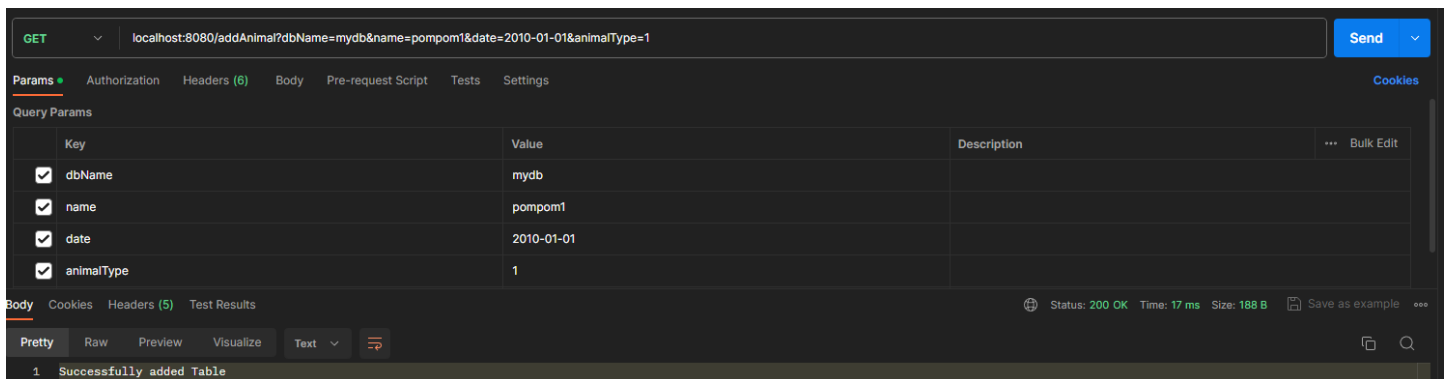
But you can also connect to a database using the “connect” command:



Adding Tables to a Database

You can add tables in two ways, either by using the addTable command, just like with a CLI, or you can use the add[TableName] commands.

Here’s an example:



The parameters are the same as the declared attributes for each table. You can view those in the [Tables](#) section of the class reference.

Viewing Tables in a Database

You can view tables using the table command:

The screenshot shows a REST client interface with the URL `localhost:8080/table?dbName=mydb&tableName=animal`. The response body is displayed in a text editor, showing a list of animals with their primary keys and fields.

Key	Value	Description
<input checked="" type="checkbox"/> dbName	mydb	
<input checked="" type="checkbox"/> tableName	animal	

```
1 animal with Primary Keys: idAnimal(Integer) = 29, Fields: idAnimal(Integer) = 29, name(String) = null, date(Date) = null, animalType(Integer) = null,
2 animal with Primary Keys: idAnimal(Integer) = 30, Fields: idAnimal(Integer) = 30, name(String) = null, date(Date) = null, animalType(Integer) = null,
3 animal with Primary Keys: idAnimal(Integer) = 31, Fields: idAnimal(Integer) = 31, name(String) = null, date(Date) = null, animalType(Integer) = null,
4 animal with Primary Keys: idAnimal(Integer) = 61, Fields: idAnimal(Integer) = 61, name(String) = null, date(Date) = null, animalType(Integer) = null,
5 animal with Primary Keys: idAnimal(Integer) = 64, Fields: idAnimal(Integer) = 64, name(String) = null, date(Date) = null, animalType(Integer) = null,
6 animal with Primary Keys: idAnimal(Integer) = 78, Fields: idAnimal(Integer) = 78, name(String) = myAnimal, date(Date) = null, animalType(Integer) = null,
7 animal with Primary Keys: idAnimal(Integer) = 83, Fields: idAnimal(Integer) = 83, name(String) = pompom1, date(Date) = 2010-01-01, animalType(Integer) = 1,
```

You can also use the `get[TableNames]` command

The screenshot shows a REST client interface with the URL `localhost:8080/getAnimals?dbName=mydb`. The response body is displayed in a text editor, showing a list of animals with their primary keys and fields.

Key	Value	Description
<input checked="" type="checkbox"/> dbName	mydb	

```
1 animal with Primary Keys: idAnimal(Integer) = 29, Fields: idAnimal(Integer) = 29, name(String) = null, date(Date) = null, animalType(Integer) = null,
2 animal with Primary Keys: idAnimal(Integer) = 30, Fields: idAnimal(Integer) = 30, name(String) = null, date(Date) = null, animalType(Integer) = null,
3 animal with Primary Keys: idAnimal(Integer) = 31, Fields: idAnimal(Integer) = 31, name(String) = null, date(Date) = null, animalType(Integer) = null,
4 animal with Primary Keys: idAnimal(Integer) = 61, Fields: idAnimal(Integer) = 61, name(String) = null, date(Date) = null, animalType(Integer) = null,
5 animal with Primary Keys: idAnimal(Integer) = 64, Fields: idAnimal(Integer) = 64, name(String) = null, date(Date) = null, animalType(Integer) = null,
6 animal with Primary Keys: idAnimal(Integer) = 83, Fields: idAnimal(Integer) = 83, name(String) = pompom1, date(Date) = 2010-01-01, animalType(Integer) = 1,
```

In a MySQL database, you can also execute queries using the `execQuery` command:

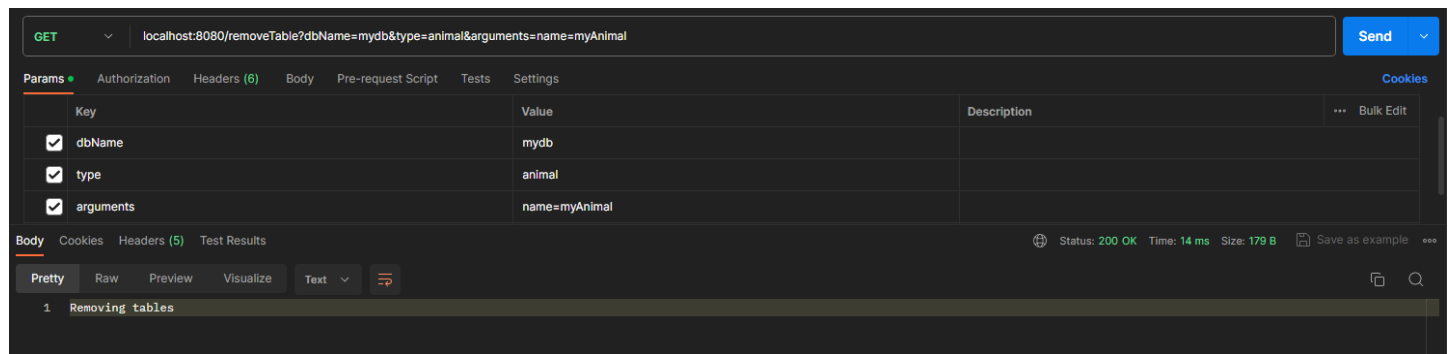
The screenshot shows a REST client interface with the URL `localhost:8080/execQuery?dbName=mydb&query=SELECT idAnimal, name FROM animal`. The response body is displayed in a text editor, showing the results of the SQL query.

Key	Value	Description
<input checked="" type="checkbox"/> dbName	mydb	
<input checked="" type="checkbox"/> query	SELECT idAnimal, name FROM animal	

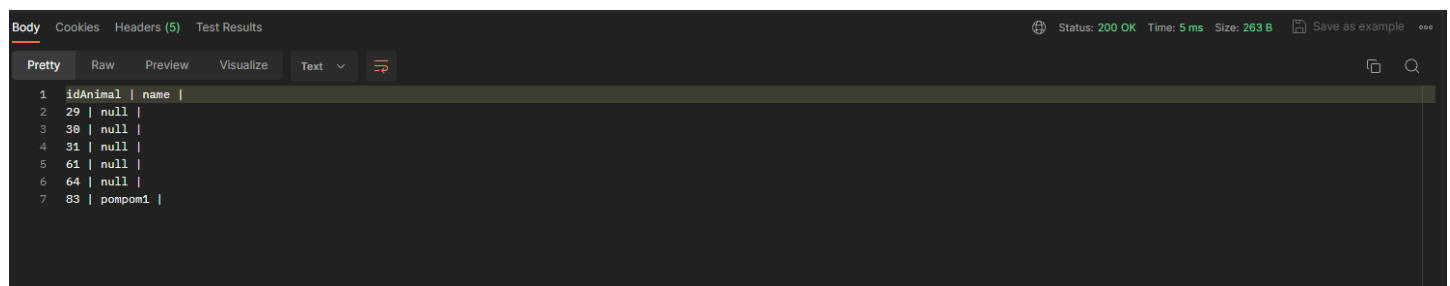
```
1 idAnimal | name |
2 29 | null |
3 30 | null |
4 31 | null |
5 61 | null |
6 64 | null |
7 78 | myAnimal |
8 83 | pompom1 |
```

Removing Tables from a Database

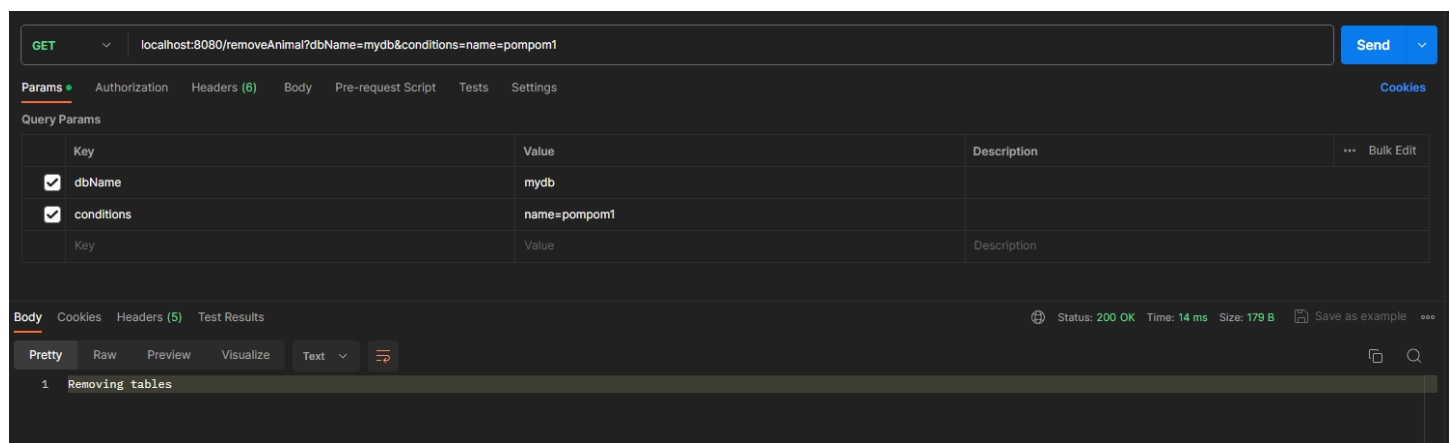
You can remove tables using the removeTable command:



This will remove all tables of the given type which match the 'arguments' parameter. You can also leave it blank to remove all tables of that type.



You can also use the remove[TableName] command:



Class Reference:

- Annotations
 - [AutoIncrement](#)
 - [CLICommand](#)
 - [PrimaryKey](#)
- Base
 - [CacheData](#)
 - [Database](#)
 - [DatabaseProxy](#)
 - [DatabaseStrategy](#)
 - [Observable](#)
 - [Observer](#)
 - [Repo](#)
 - [Table](#)
- Repositories
 - [ListRepo](#)
- Strategies
 - [ListSQLAdapter](#)
 - [ListStrategy](#)
 - [MySQLStrategy](#)
- Helpers
 - [ArrayHelper](#)
 - [StringConverter](#)
- Factories
 - [ProxyFactory](#)
- Controllers

- [Controller](#)
- [AnimalController](#)
- [AnimalTypeController](#)
- [ClientController](#)
- [HabitatController](#)
- [ReceiptController](#)
- [ReceiptAnimalController](#)
- Tables
 - [Animal](#)
 - [AnimalType](#)
 - [Client](#)
 - [Habitat](#)
 - [Receipt](#)
 - [ReceiptAnimal](#)
- UI
 - [CLI](#)
 - [REST](#)

AutoIncrement

Description

An Annotation used on Primary Key fields of Type Integer in classes that extend Table. When an object that has an attribute with this annotation is added to a Repo, it first searches for the highest existing value for the attribute in the repo, and increments it by 1 for the newly added object.

@Target is ElementType.FIELD

@Retention is RetentionPolicy.RUNTIME

CLICommand

Description

An Annotation used in the CLI class to mark a method as being available to be called by the user in the CLI. A CLICommand should return a String, and it should have a String[] as its only parameter.

@Target is ElementType.METHOD

@Retention is RetentionPolicy.RUNTIME

PrimaryKey

Description

An Annotation used on Fields in classes extending Table. It marks that Field as being unique, and is then used when comparing objects (when finding, adding, etc). Objects of the same type with the same values for its Primary Keys are considered equal, regardless of other attributes.

@Target is ElementType.FIELD

@Retention is RetentionPolicy.RUNTIME

CacheData

Description

A class that is used to hold information about a cached user command.

Properties:

String	table
String	function
String[]	parameters
String	output

Constructors:

CacheData (String table, String function, String[] parameters, String output)

Database extends [Observable](#) implements [Observer](#)

Description

Represents a Database. It can use different Strategies to determine how the data is stored and used.

Properties:

String	name
DatabaseStrategy	strategy

Constructors:

Database (String name)
Database (String name, DatabaseStrategy strategy)

Methods:

DatabaseStrategy	getStrategy ()
void	setStrategy (DatabaseStrategy strategy)
boolean	add (Table table) <i>throws Exception</i>
boolean	remove (Table table) <i>throws Exception</i>
Table []	getTables ()
Table []	getTables (String tableName)
String	getName ()
void @Override	update (Object arg)

DatabaseStrategy getStrategy ()

- Returns the *strategy* property

void setStrategy (**DatabaseStrategy** strategy)

- Sets the *strategy* property

boolean add (**Table** table) *throws Exception*

- Adds a table to the database
- Returns true if table was added successfully
- Returns false otherwise
- Notifies all proxies in order to clear cache that uses this table

boolean remove (**Table** table) *throws Exception*

- Removes a table from the database
- Returns true if table was removed successfully
- Returns false otherwise
- Notifies all proxies in order to clear cache that uses this table

Table[] getTables ()

- Returns all the tables in the database

Table[] getTables (**String** tableName)

- Returns all the tables of a certain type in the database

String getName ()

- Returns the *name* property

void update (**Object** arg)

- Object can be of type **String** or **DatabaseProxy**
- Receiving an update with an object of type **String** will send back the result of *getTables(arg)*.
- Receiving an update with an object of type **DatabaseProxy** will add that Proxy to the Observer list

DatabaseProxy extends [Observable](#) implements [Observer](#)

Description

Makes the connection between the User and the Database. All user Inputs are sent to the proxy, and the proxy then retrieves the necessary information from the database and sends it back to the user.

Properties:

CacheData []	cachedData
Database	database

Constructors:

DatabaseProxy ()
DatabaseProxy (Database database)

Methods:

String	getDatabaseName ()
DatabaseStrategy	getStrategy ()
String	getTablesByName (String tableName)
void	addTable (String tableName) <i>throws Exception</i>
void	addTable (String tableName, String... args) <i>throws Exception</i>
void	removeTable (String tableName, String... args) <i>throws Exception</i>
CacheData	getCachedData (String tableName, String function, String[] parameters)
void	cache (String tableName, String function, String[] parameters, String output)
void @Override	update (Object arg)

String getDatabaseName ()

- Returns *database.name*

DatabaseStrategy getStrategy ()

- Returns *database.strategy*

String getTablesByName (**String** tableName)

- Returns tables of type specified in *tableName*

void addTable (**String** tableName) *throws Exception*

- Creates a new table of type specified in *tableName* and adds it to the database.

void addTable (**String** tableName, **String...** args) *throws Exception*

- Creates a new table of type specified in *tableName* with its properties defined in *args*.
- *args* can either be of format “attributeName=value” or just “value”, but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

void removeTable(**String** tableName, **String...** args) *throws Exception*

- Removes all tables of type specified in *tableName* which meet the conditions set in *args*.
- *args* can either be of format “attributeName=value” or just “value”, but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

CacheData getCacheData(**String** tableName, **String** function, **String[]** parameters)

- Returns the cached data for the specified table, function, and parameters or null if it doesn't exist.

void cache (**String** tableName, **String** function, **String[]** parameters, **String** output)

- Caches a user-inputted command and its output.

void update (**Object** arg)

- If arg is of class Table, looks for all data cached with that table and removes it.
- This update is received from a Database, when a table of that type is added or removed, meaning that on next query the output has to be calculated again

DatabaseStrategy

Description

An interface that declares all the necessary methods for a database strategy.

Methods:

boolean	add (Table table) <i>throws Exception</i>
boolean	remove (Table table) <i>throws Exception</i>
Table []	get (String name)
Table []	getAll ()

Observable

Description

An abstract class that declares and defines all the necessary properties and methods for an Observable object.

Properties:

Observer[]	observers
-------------------	-----------

Methods:

void	notifyObservers ()
void	notifyObservers (Object arg)
void	notifyObservers (Object[] args)

void notifyObservers ()

- Calls update method of all Observers in *observers* with no arguments

void notifyObservers (**Object** arg)

- Calls update method of all Observers in *observers* with one argument

void notifyObservers (**Object[]** args)

- Calls update method of all Observers in *observers* with multiple arguments

Observer

Description

An interface that declares all the necessary methods for an Observer object.

Methods:

void	update ()
void	update (Object arg)
void	update (Object[] args)

void update ()

- Called whenever an [Observable](#) that is observed by this object calls its *notifyObservers* method with no arguments.

void update (**Object** arg)

- Called whenever an [Observable](#) that is observed by this object calls its *notifyObservers* method with one arguments.

void update (**Object[]** args)

- Called whenever an [Observable](#) that is observed by this object calls its *notifyObservers* method with multiple arguments.

Repo<T>

Description

An abstract class that declares and defines all the necessary properties and methods for a Repository that holds tables.

Properties:

Class<T>	type
-----------------------	------

Constructors:

<i>protected</i> Repo ()
Repo (Class<T> type)

Methods:

boolean	add (Object obj) <i>throws Exception</i>
boolean	addToRepo (T instance)
boolean	remove (Object obj)
boolean	removeFromRepo (T instance)
T	getValue (int index)
Integer	getMaxIncrement (Field field) <i>throws Exception</i>
int	getSize ()
boolean	contains (Object obj)
Class<?>	getType ()
T	castToGeneric (Object obj)

boolean add (**Object obj)**

- Casts *obj* to type **T** if possible and adds it to the Repo using *addToRepo*.
- This method also runs all the *@AutoIncrement* logic, so adding tables to the repo directly using *addToRepo* will not automatically increment any attributes that have that annotation.

boolean addToRepo (**T instance)**

- Adds *instance* to the Repo and returns true if successful.

boolean remove(**Object obj)**

- Casts *obj* to type **T** if possible and removes it from the Repo using *removeFromRepo* if it contains it.

boolean removeFromRepo(**T instance)**

- Removes *instance* from the Repo and returns true if successful.

T getValue (**int index)**

- Gets the value of the object at *index*.

Integer getMaxIncrement (**Field field)**

- Returns the highest value of the specified field in the Repo.

int getSize ()

- Returns the amount of tables in the Repo.

boolean contains (**Object obj)**

- Returns true if *obj* is in the Repo.

Class<?> getType ()

- Returns the *type* property.

T castToGeneric(**Object obj)**

- Casts *obj* to type **T** or null if it cannot be casted. Used mostly internally.

Table

Description

An abstract class that declares all the methods and properties necessary for a table object. Objects extending Table must have an attribute with the [PrimaryKey](#) annotation, or else they will throw an exception whenever instantiated. Objects extending Table should have lowercase names and attribute names (because mysql lowercase shit).

Properties:

Field[]	primaryKey
----------------	------------

Constructors:

Table () <i>throws Exception</i>
--

Methods:

Field[]	getFields ()
Field	getField (Integer index)
Field	getField (String name)
Field[]	getPrimaryKeys ()
Object	getProperty (Integer index) <i>throws IllegalAccessException</i>
Object	getProperty (String name) <i>throws IllegalAccessException</i>
void	setProperty (Integer index, Object value) <i>throws Exception</i>
void	setProperty (String name, Object value) <i>throws Exception</i>
boolean	checkPrimaryKey () <i>throws Exception</i>

Field[] getFields ()

- Returns all properties of the table.

Field getField (**Integer** index)

- Returns the field at *index*. Index is based on the order in which the Fields are declared in the class.

Field getField (**String** name)

- Returns the field by *name*.

Field[] getPrimaryKeys ()

- Returns all fields with the [PrimaryKey](#) annotation.

Object getProperty(**Integer** index) *throws IllegalAccessException*

- Returns the value of the **Field** at *index*. Index is based on the order in which the Fields are declared in the class.

Object getProperty(**String** name) *throws IllegalAccessException*

- Returns the value of the **Field** by *name*.

Object setProperty(**Integer** index, **Object** value) *throws Exception*

- Finds **Field** at *index* and changes its value to *value*. Index is based on the order in which the Fields are declared in the class.

Object setProperty(**String** name, **Object** value) *throws Exception*

- Finds **Field** by *name* and changes its value to *value*.

boolean checkPrimaryKey () *throws Exception*

- Returns false if the table has no properties with the [PrimaryKey](#) annotation.

ListRepo<T> extends [Repo<T>](#)

Description

An implementation of Repo<T> using ArrayLists.

Properties:

List<T>	list
----------------------	------

Constructors:

<i>protected</i> ListRepo ()

Methods:

<i>static</i> ListRepo<S>	newRepo (Class<S> clazz) <i>throws Exception</i>
--	--

static **ListRepo<S>** newRepo (**Class<S>** clazz) *throws Exception*

- Returns a new Repo of type S.

ListSQLAdapter extends [ListStrategy](#)

Description

An adapter class that converts a MySQLStrategy into a ListStrategy.

Properties:

MySQLStrategy	sqlStrategy
----------------------	-------------

Constructors:

ListSQLAdapter (MySQLStrategy sqlStrategy) <i>throws Exception</i>
--

ListStrategy implements [DatabaseStrategy](#)

Description

An implementation of [DatabaseStrategy](#) using an ArrayList.

Properties:

List<ListRepo<?>>	list
--------------------------------------	------

Methods:

boolean	contains (Repo<?> repo)
----------------	--

boolean contains (**Repo<?>** repo)

- Returns true if *list* contains *repo*.

MySQLStrategy implements [DatabaseStrategy](#)

Description

An implementation of DatabaseStrategy using a connection to a MySQL server.

Properties:

Connection	connection
String	dbName

Constructors:

MySQLStrategy (**String** address, **String** port, **String** dbName, **String** username, **String** password)

Methods:

<u>Table</u> []	getTablesFromQuery (String query)
String	query (String query)

[Table](#)[] getTablesFromQuery (**String** query)

- Returns an SQL query as an array of [Table](#) objects.

String query (**String** query)

- Returns an SQL query formatted as a string.

ArrayHelper

Description

A helper class that contains static methods to make working with arrays easier.

S is a generic type declared in every method as <S extends Object>

Methods:

S[]	addElement (S[] array, S object, int index)
S[]	removeElement (S[] array, int index)
S[]	addElement (S[] array, S object)
boolean	contains (S[] array, S object)
S[]	clone (S[] array, int startFrom, int EndAt)
S[]	clone (S[] array, int startFrom)
S[]	except (S[] array, S[] array2)
S[]	concatenate (S[] array, S[] array2)

S[] addElement (S[] array, S object, int index)

- Returns a new array that is a copy of *array* with *object* at *index*.

S[] removeElement (S[] array, int index)

- Returns a new array that is a copy of *array* without the object at *index*.

S[] addElement (S[] array, S object)

- Returns a new array that is a copy of *array* with *object* at the end.

boolean contains(S[] array, S object)

- Returns true if *array* contains *object*.

S[] clone (S[] array, int startFrom, int EndAt)

- Returns a clone of *array* that starts at *startFrom* and ends at *EndAt*.

S[] clone (S[] array, int startFrom)

- Returns a clone of *array* that starts at *startFrom*.

S[] except (S[] array, S[] array2)

- Returns an array that has all elements of *array* that are not in *array2*.

S[] concatenate (S[] array, S[] array2)

- Returns an array that is a clone of *array* with *array2* concatenated at the end.

StringConverter

Description

A helper class that contains a static method to convert different strings to objects. Used mainly when interpreting user input.

Methods:

Object	<u>convert</u> (String string, Class <?> converted) <i>throws Exception</i>
---------------	--

Object convert(**String** string, **Class**<?> converted)

- Converts *string* to an **Object** of class *converted*.

ProxyFactory

Description

A factory class that creates [DatabaseProxies](#).

Methods:

DatabaseProxy	create(String name, String type, String... args)
String[]	options ()

[DatabaseProxy](#) create(**String** name, **String** type, **String...** args)

- Creates a new [DatabaseProxy](#) that contains a [Database](#) with *name*, a strategy matching *type*. Args is used for further instructions, and are necessary for a [MySQLStrategy](#) in order to connect to the MySQL Server.

Controller

Description

A controller class which handles all the commands possible to be run by user input.

Properties:

DatabaseProxy []	proxies
----------------------------------	---------

Methods:

DatabaseProxy	getProxy (String name)
String	getCachedOutput (DatabaseProxy proxy, String table, String function, String [] parameters)
void	cache (DatabaseProxy proxy, String table, String function, String [] parameters, String output)
String	execQuery (String dbName, String query)
String	createDatabase (String name, String choice, String ... extraArgs)
String	table (String dbName, String tableName)
String	addTable (String dbName, String tableName, String ... extraArgs) <i>throws Exception</i>
String	removeTable (String dbName, String tableName, String ... extraArgs) <i>throws Exception</i>
String	hasTable (String dbName, String tableName)

DatabaseProxy getProxy(**String** name)

- Gets a proxy whose *database.name* is equal to *name*.

String getCachedOutput(**DatabaseProxy** proxy, **String** table, **String** function, **String[]** parameters)

- Returns the output from a **CacheData** matching the parameters.

void cache (**DatabaseProxy** proxy, **String** table, **String** function, **String[]** parameters, **String** output)

- Creates a **CacheData** object in *proxy* matching the arguments.

String execQuery (**String** dbName, **String** query)

- Returns the result of an SQL query (only works on a MySQLStrategy)

String createDatabase(**String** name, **String** choice, **String...** extraArgs)

- Creates a new **DatabaseProxy** of type matching *choice*.

String table (**String** dbName, **String** tableName)

- Returns a list of all the **tables** of type matching *tableName* from the **Database** with name *dbName*.

String addTable (**String** dbName, **String** tableName, **String...** extraArgs) *throws Exception*

- Adds a **table** of type *tableName* to the **Database** with name *dbName*.
- *extraArgs* can either be of format “attributeName=value” or just “value”, but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

String removeTable (**String** dbName, **String** tableName, **String...** extraArgs) *throws Exception*

- Removes all [tables](#) of type *tableName* matching the conditions set in *extraArgs* to the [Database](#) with name *dbName*.
- *extraArgs* can either be of format “attributeName=value” or just “value”, but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

boolean hasTable (**String** dbName, **String** tableName)

- Returns true if the [Database](#) with name *dbName* contains [tables](#) of type *tableName*.

AnimalController

Description

A co

Methods:

String	addAnimal (Controller controller, String dbName, Integer idAnimal, String name, Date date, Integer animalType) <i>throws Exception</i>
String	removeAnimal (Controller controller, String dbName, String... conditions) <i>throws Exception</i>
String	getAnimals (Controller controller, String dbName)

String addAnimal ([Controller](#) controller, **String** dbName, **Integer** idAnimal, **String** name, **Date** date, **Integer** animalType) *throws Exception*

- Creates an **Animal** object in the **Database** with name *dbName* from *controller*.

String removeAnimal ([Controller](#) controller, **String** dbName, **String...** conditions) *throws Exception*

- Removes all **Animal** objects matching *conditions* in the **Database** with name *dbName* from *controller*.
- *conditions* can either be of format "attributeName=value" or just "value", but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

String getAnimals ([Controller](#) controller, **String** dbName)

- Returns a list of all **Animal** objects in the **Database** with name *dbName* from *controller*.

animal extends [Table](#)

Description

A table representing an animal.

Properties:

@PrimaryKey Integer	idAnimal.
String	name
Date	date
Integer	animalType

animaltype extends [Table](#)

Description

A table representing an animaltype.

Properties:

@PrimaryKey Integer	idType
String	name

client extends [Table](#)

Description

A table representing a client.

Properties:

String	name
String	lastName
String	email
@PrimaryKey Integer	idClient

habitat extends [Table](#)

Description

A table representing a habitat.

Properties:

@PrimaryKey Integer	idHabitat
Integer	idType

receipt extends [Table](#)

Description

A table representing a receipt.

Properties:

Integer	idClient
@PrimaryKey Integer	idReceipt
Date	date

receiptanimal extends [Table](#)

Description

A table creating the many-to-many connection between receipt and animal.

Properties:

@PrimaryKey Integer	idReceipt
@PrimaryKey Integer	idAnimal
Integer	price

CLI

Description

A Command Line Interface to allow the user to interact with the database.

Properties:

Controller	controller
boolean	stop
int	exitCode
boolean	printStackTrace

Constructors:

CLI (Controller controller)

Methods:

void	invoke (String input)
void	start ()
@CLICommand String	tgs (String[] args)
@CLICommand String	exit (String[] args)
@CLICommand String	toggleStackTrace (String[] args)
@CLICommand String	execQuery (String[] args)
@CLICommand String	createDatabase (String[] args)
@CLICommand String	cdb (String[] args)
@CLICommand String	table (String[] args)
@CLICommand String	addTable (String[] args) <i>throws Exception</i>
@CLICommand String	removeTable (String[] args) <i>throws Exception</i>
@CLICommand String	hasTable (String[] args) <i>throws Exception</i>

boolean stop

- Determines whether the while loop from [start \(\)](#) should end.

int exitCode

- Exit code used to determine the state that the CLI exited in.

boolean printStackTrace

- If true, prints entire stack trace when an exception is thrown from an invoked command.
- If false, prints only “Could not invoke command”.

void invoke (String input)

- Invokes a command annotated with [@CLICommand](#).
- Syntax: “*commandname arg1 arg2 arg3 arg4...*”
- Splits *input* into multiple strings using the format: [“arg1”, “arg2”, “arg3”, “arg4” ...]

void start ()

- As long as [stop](#) is false, takes user input and calls *invoke* with it.

@CLICommand String tgs (String[] args)

- Short form for [toggleStackTrace \(\)](#).

@CLICommand String exit (String[] args)

- Exits the CLI.
- *args[0]* is the [exit code](#).

@CLICommand String toggleStackTrace (String[] args)

- If no argument is given, toggles [printStackTrace](#)
- *args[0]* can be “1” or “True” in order to set it to true, or “0” or “False” in order to set it to false.

@CLICommand String execQuery (String[] args)

- Executes a MySQL query using [controller.execQuery \(\)](#).
- Arguments: [*dbName*, *query*]

@CLICommand String createDatabase (String[] args)

- Creates a **Database** using [controller.createDatabase \(\)](#).
- Arguments: [*name*, *strategy*, *args*]

@CLICommand String cdb (String[] args)

- Short form for [createDatabase \(\)](#).

@CLICommand String table (String[] args)

- Returns a list of all tables with the specified name using [controller.table\(\)](#).
- Arguments: [*dbName*, *tableName*]

@CLICommand String addTable (String[] args)

- Adds a table to the specified database with the given arguments using [controller.addTable \(\)](#).
- Arguments: [*dbName*, *tableName*, *extraArgs*]

@CLICommand String removeTable (String[] args)

- Removes all tables from the specified database with the given arguments using [controller.removeTable \(\)](#).
- Arguments: [*dbName*, *tableName*, *extraArgs*]

@CLICommand String hasTable (String[] args)

- Returns the output of [controller.hasTable \(\)](#) using the given arguments.
- Arguments: [*dbName*, *tableName*]

REST

Description

A **RestController** that uses Spring Boot in order to allow the user to interact with the database using Posts.

Properties:

Controller	controller
String	password
AnimalController	animalController
AnimalTypeController	animalTypeController
ClientController	clientController
HabitatController	habitatController
ReceiptAnimalController	receiptAnimalController
ReceiptController	receiptController

Constructors:

Rest ()

Methods:

void	setController (Controller controller)
@GetMapping("/print") String	print (String message)
@GetMapping("/query") String	query (String dbName, String query)
@GetMapping("/add[TableName]") String	add[TableName] (String dbName, [<i>attributes</i>])
@GetMapping("/remove[TableName]") String	remove[TableName] (String dbName, String conditions)
@GetMapping("/get[TableName]") String	get[TableName] (String dbName)

void setController ([Controller](#) controller)

- Sets *this.controller* to *controller*.

String print (**String** message)

- Prints a message.

String add[TableName] (**String** dbName, [*attributes*])

- [TableName] should be replaced with the actual table name (ex. addAnimal)
- [*attributes*] should be replaced with the actual table's attributes
- Ex.: Animal's [*attributes*] are **Integer** idAnimal, **String** name, **String** date, **Integer** animalType.

String remove[TableName] (**String** dbName, **String** conditions)

- [TableName] should be replaced with the actual table name (ex. removeAnimal)
- *conditions* can either be of format "attributeName=value" or just "value", but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

String get[TableName] (**String** dbName)

- [TableName] should be replaced with the actual table name (ex. getAnimals)