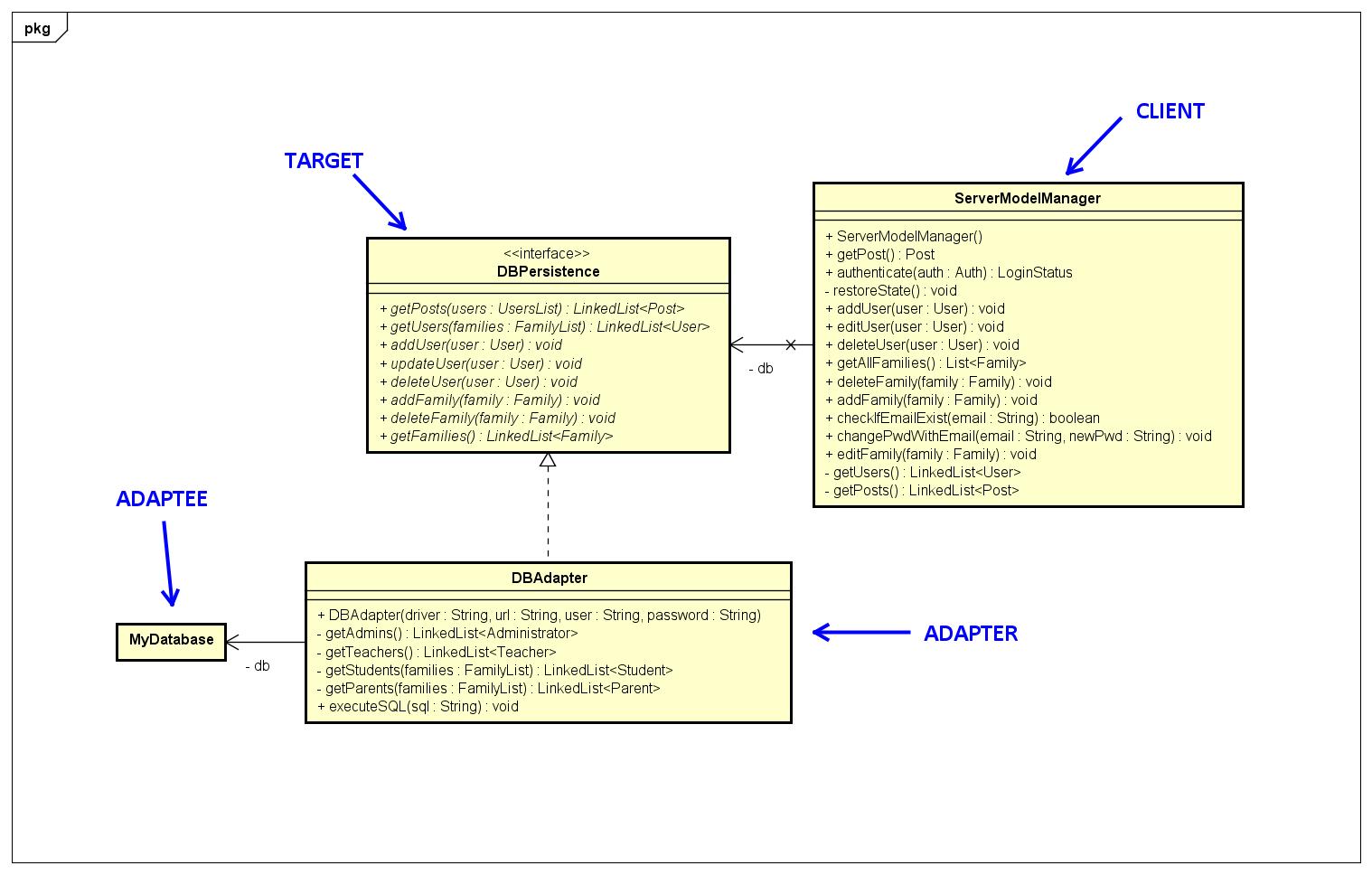
# Connection to the database

The connection between Java and the database is established by using standard Java API - JDBC (Java Database Connectivity). The latest version of JDBC Driver (PostgreSQL JDBC 4.2 Driver, 42.2.2) has been imported in order to access to the database’s data that is placed on eNTe’s private server with IP address - 207.154.237.196.

The adapter design pattern has been chosen for transforming data to and from the database. The main reason for using this design pattern is that it makes the system clearer, divides responsibility and provides the possibility to change the database without having to make any changes in other parts of the system.

It is implemented in the following way. The adapter class (DBAdapter) implements the interface DBPersistence that is like a target that holds methods needed for the model on the server’s side. Therefore the ServerModelManager has not any direct relations to the database (SQL language). The SQL strings are created by methods in DBAdapter class, which is responsible for calling these SQL statements on MyDatabase class, which is represented by the Adaptee in the adapter design pattern. MyDatabase class is made to be general in order to be able to access data from any kind of database.

NOTE: DBAdapter method called “executeSQL(String sql)” was created only for testing purposes to be able to delete all data after each particular test in DBAdapterTest class.

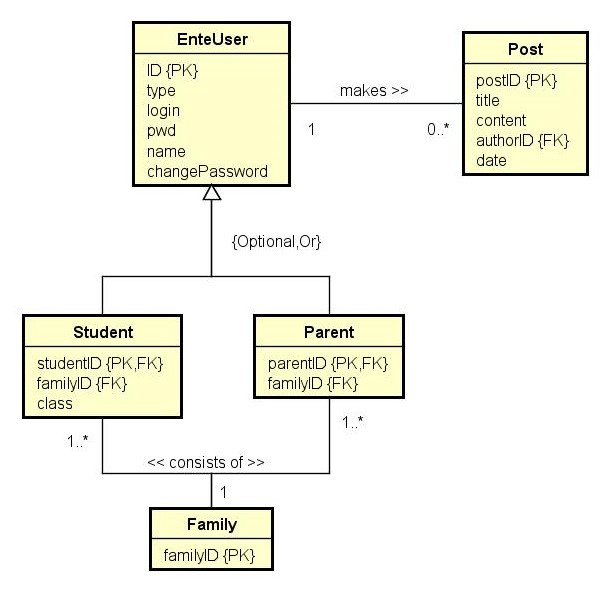


# Documentation DB design for managing users

There was a need to create a database for storing data about users from the system. Exact tasks for the database were following:

1. For every of 4 types of users (Administrator, Teacher, Student and Parent) eNTe wants to keep track of the user’s name, login and password.
2. For every student and parent it is needed to keep track of family that he/she belongs to. Student or parent can belong only to one family and a family can consist of multiple students (children) and parents.
3. For every student eNTe needs to hold student’s current class. Therefore, a student can only be a part of one class and a class can consist of multiple students.

The EER diagram below presents a possible solution for tasks mentioned above.



At the moment, there is no need to make Teacher or Administrator as an entity as they do not hold any specific data in comparison with enteuser. Therefore only Student and Parent are made as separate entities that hold specific data about family and also about class in the case of a student.

Following logical database design shows the primary and foreign keys for the given entities.

**EnteUser** (ID,type,login,pwd,name,changePassword)

PK: ID

**Post** (postID,title,content,authorID,date)

PK: postID

FK: authorID REFERENCES EnteUser (ID)

**Student** (studentID,familyID,class)

PK: studentID

FK: studentID REFERENCES EnteUser (ID)

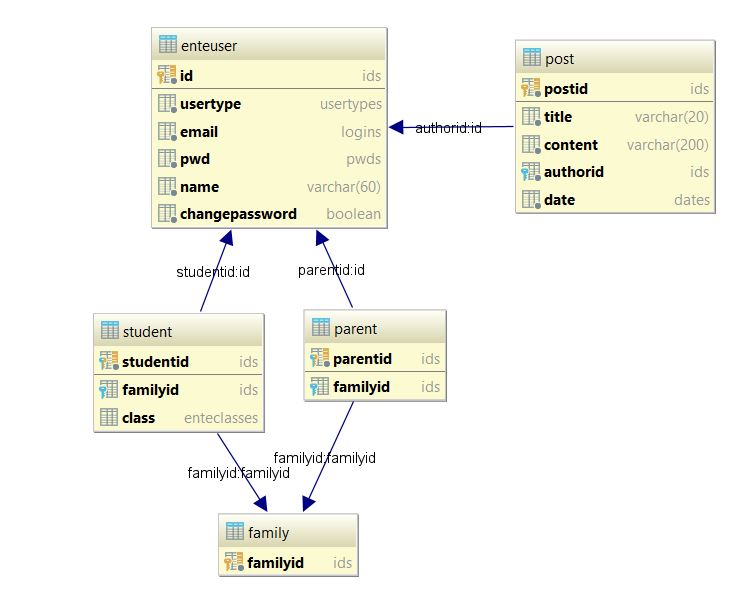
FK: familyID REFERENCES Family (familyID)

**Parent** (parentID,familyID)

PK: parentID

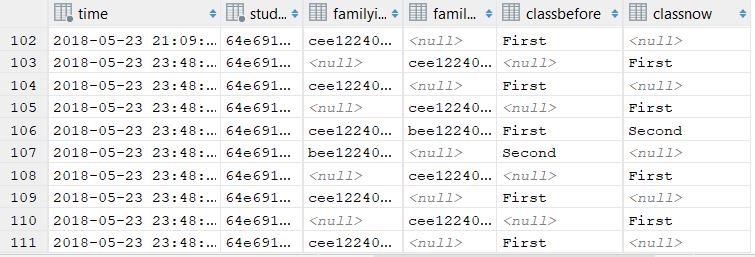
FK: familyID REFERENCES Family (familyID)

Physical database was based on the previous analysis (EER diagram, logical database design). It can be seen in the database visualization below.



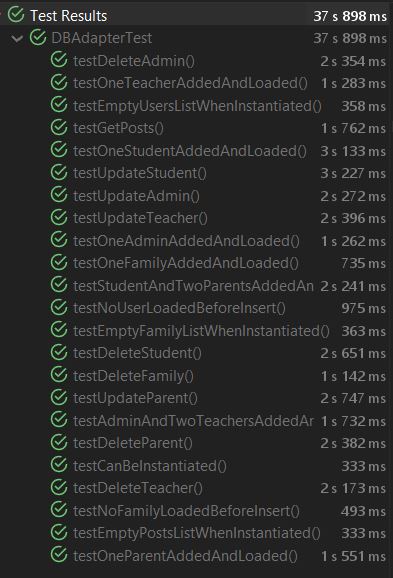
# JUnit Tests for managing users – DBAdapter

DBAdapter class was tested using unit testing framework for Java called Junit. In order to see data flow in the database, the structure (tables, domains) of the main schema were cloned into the testing schema. Moreover, log tables for all already existing tables were created in the testing schema with the intention of controlling data manipulation done by the DBAdapterTest class.



Log\_student table created in test schema of eNTe’s database

The reason of such a long execution of this test is that after each test all data from the database is dropped. That is because it is needed to have no data in any table while a particular test is being executed. Eventually, all tests passed.



Test list:

* Add user
* Delete user
* Update user
* Get users
* Add family
* Delete family
* Get family
* None data retrieved
* Connection
* Get post