**2. [Analiza tematu]**

The topic of this thesis is strongly connected with databases. Databases are places where great amounts of data in electronic form can be stored. Databases provide fast and convenient access and data processing.

Taking into consideration, informal definition, a database is a data set with a determined internal structure of stored resources. Natural language, so easy and comfortable to communicate in society is not a precise tool while saving and storing information in a computer. Properly collected and structured information act as data models - the basis in order to create a database system.

Nowadays, incorrect conviction about databases has been widespread. It is said that databases and their managing systems concern a trivial domain of memorizing and obtaining data. It may be because of English word *datum* derived from Latin. It means literally a fact, but data very often not necessarily is connected with a fact. Data may not be explicit and could regard to some idea.

It is assumed, that the main problem is not just about a data and the manner of data storing but in the way of the interpretation. It is clear, that the high level of abstraction is needed to interpret a data in order to tolerate the certain disturbances of the real world, and at the same time to be sufficiently close, giving an idea of ​​how the data is interrelated. The conceptual construction that provides this interpretation is called the data model. The basic properties of the phenomenon being described are: the **static** and **dynamic** property classes.

Static class properties, called database schemas are constant, unchangeable in time. Such a schema corresponds to what is usually called **Data Definition Language - DLL.** That Language defines acceptable data structures within a given model. It means, this language determines the properties that must be true for all the database instances of the specified schema. There are two complementary ways to define data structures:

1. Permitted objects and their compounds are specified by means of general rules of defining for the category they belong to.

2. Unacceptable objects and relationships are excluded by defining constraints, i.e. imposing restrictions on the category.

Dynamic properties, called database states, are expressed in a set of operations

corresponding to the **Data Manipulation Language- DML.** This set defines the permitted operations that can beperformed on a certain instance of one database in order to receive an instanceof the other. A good example of such an operation is updating data. The updates change database from one state to another. The new state is introduced as a result that may be a true, or by denying facts it stops being true any more. Dynamic class also includes operations that do not change in the occurrence, and yet are dynamic because they cause a change in state databases, e.g. queries. the query does not modify the database but it is mainly used to check if a fact or group of facts are met in the given state of considered database.

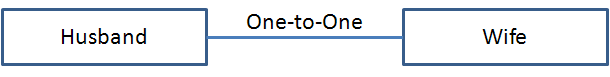
**Relational database**

A relational database is a set of formally described, multiple data collections formed in tables. In further thesis considerations table will be also called entity. An entity is an object that just exists. It does not have to do anything special. It is just a place where data is stored. Entity can represent single object, person or place. Data in such tables is organized in columns and records. A column is a vertical entity in a table that contains all information associated with a specific field in a table. On the other hand a row is each individual entry that exists in the database.

**Relations**

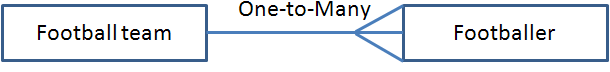
Tables communicate with the others and share information, simplifying data organization and search ability. Relational database comes with a well defined relationship between database tables. It organizes data in different ways. Each record contains unique data representation, as an instance or key for the characteristics defined by the columns. One or more rows relate to one or more record from another table forming the functional dependencies. Those relations are described as follows:

* One to One: One table record relates to another record in another table.



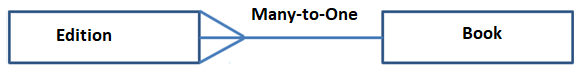
*Photo 1. One to one relation*

* One to Many: One table record relates to many records in another table.



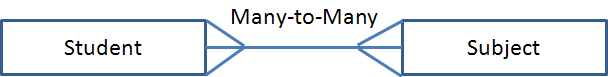
*Photo 2. One to many relation*

* Many to One: Many table records relate to one record in another table.



*Photo 3. Many to one relation*

* Many to One: Many table records relate to many records in another table.



*Photo 4. Many to many relation*

**Constraints**

Every relation has specified constraints that must be held in order to keep the relation valid. These requirements are called **Relational Integrity Constraints**. Six main constraints can be distinguished:

* **Primary key** - it is a main condition that must contain the unique value for each row in the table and cannot contain null values. A table can have only one primary key. Primary keys can consist of single or multiple fields.
* **Foreign key** - it is a field in one table, uniquely identifying row in another table. The table which contains the foreign key is called the referencing table. On the other hand, a table which contains the candidate key is called the referenced one.
* **Not null** - this constraint indicates that the column cannot contain null values. Updating or inserting a new record is impossible without adding a value to this field.
* **Unique key** - constraint ensuring that all values in a column are different. For instance Primary key has automatically a unique value. There can be many unique constraints in one table.
* **Check** - it is responsible for limiting the range of value in a column. Defined on a single column allows setting only certain values for considering column. If the check constraint is defined for the table, it can limit the values for certain column based on the values from another column in the row.
* **Domain constraint** - it is a data type defined by the user. It consists of entry data type and one of the available constraints.

**Database schema**

A database schema is a main structure describing the logical representation of the database. It defines what data types together with their constraints are to be stored in the database. The relations between entities are specified too. What is more, the schema determines also indexes, events, triggers, functions or even procedures. A database schema can appear as:

* **Logical Database Schema -** it describes all the logical parts of the database by mean of tables with their data types, integrity constraints and views.
* **Physical Database Schema -** defines the form and the way how data is stored and represented (e.g. indices, files).

**Object-Relational mapping**

In this chapter, the fully detailed idea of the Object-Relational Mapping (ORM) is explained. The main application of ORM with some examples will be presented. After that, some pros and cons of that solution will be introduced.

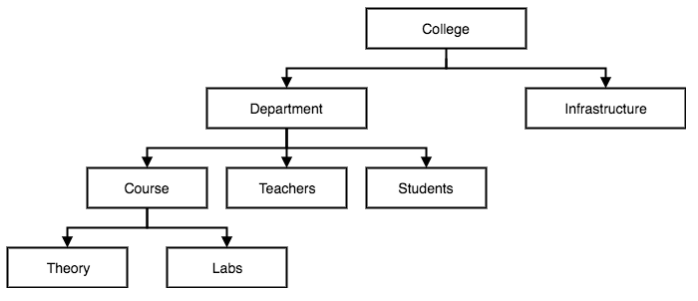
**What is Object-Relational mapping?**

With regard to widely used relational databases, which practically are very fast, stable and safe, it turned out that a serious problem is to introduce object oriented system structure to relational database. Then Object-Relational mapping comes to help. It is a process of changing data representation from relational to object oriented form. ORM tool lets to project database resources to objects and to execute different operations on the database that are performed in the same way like in every object oriented programming language operations on regular objects. Every class is mapped to a table. Each instance of the class is mapped to a row of the specified table with corresponding identifier. The identifier usually appears as an auto incremental integer that allows to directly refer to a given row in the database. In practice work with the ORM is based on creating a data model in object oriented language and setting up a database upon that model. After that mapping of the database to a relational model is performed.

**Database model**

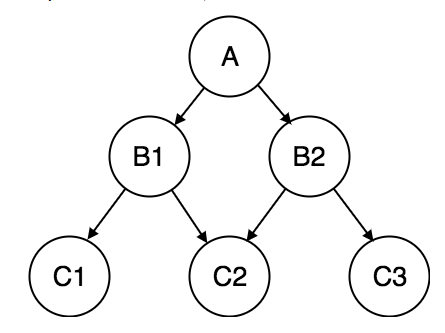
Database model is a structure that organizes data elements within their relationships. It is more like a set of concepts that can be used while describing database structure. One of the most popular database models is the relational one, but there are also other models like:

* **Hierarchical Model** - in this model data is organized in the tree-like-structure. There is a single root element and its expanding child nodes.



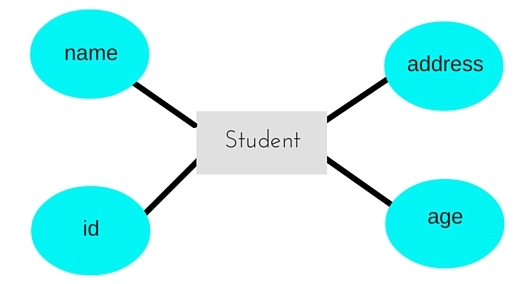
*Photo 5. Example of hierarchical model*

* **Network Model** - in this case data is organized in the graph structure. It is possible for a child node to have more than one parent node. Network model is kind of the extended version of the hierarchical model.



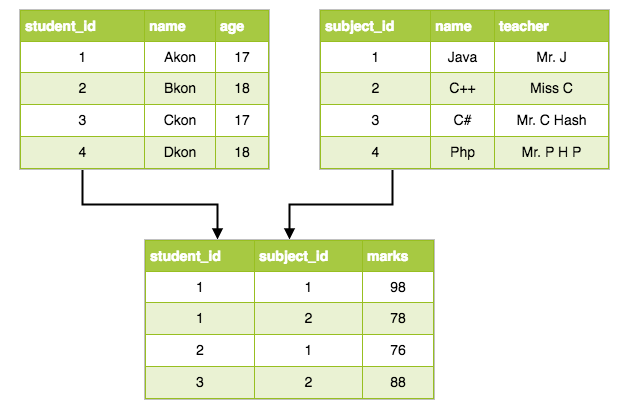
*Photo 6. Example of network model*

* **Entity-relationship Model** - it is based on dividing data objects into entities and their data into attributes. This type of model is very often applied while designing a database. It can be easily transformed into the relational model.



*Photo 7. Example of entity-relationship model*

* **Relational Model** - in this model data is organized in tables.Common attributes set the relationship between considering entities. Data information is stored in rows. Columns determine all information associated to the specific table entry.



*Photo 8. Example of relational model*

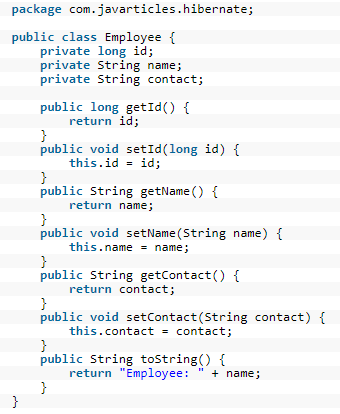
The data model is an integral part of the **Database Management System (DBMS)** - a system that provides two the most important database mechanisms. One of them is based on defining database schemas. The other one delivers a bunch of operations manipulating the data with the same schema at all times. It should be specified that DBMS is responsible for:

* Managing files
* Searching for information
* Managing the whole a database i.e. it is like a shell that surrounds the database where all the operations are performed

**ORM in examples**

Nowadays, there are many popular object-relational mapping frameworks. As it was said at the beginning of this chapter, the main idea of the ORM tool is to make it easier to execute different operations on the database structure instead of writing long SQL queries. In this thesis two JavaScript ORM frameworks will be used in order to check their efficiency and execution times in comparison to implemented library tool. However, Hibernate framework for Java will be described in this chapter to demonstrate how ORM works.

First of all a persistence class is created to represent database entity. As it is pictured below, class contains some attributes. For each entity class an unique identifier property must be defined. In discussing example ***id***property acts as a unique key. The entity class has also some getters, setters and other methods.



*Photo 8. Example of persistence class*

For Hibernate to work properly a configuration file to configure properties like database settings, metadata and using dialect is required. That file is purposely omitted in this discussion and it will not be described in details. The most crucial thing is to demonstrate the specification of the ORM itself.

In the next few examples all required steps to list, delete and create a new employee in the database will be performed.



*Photo 9. Example of creating session factory*

Initially a new configuration instance is created. Based on that configuration a new factory session object is built. SessionFactory is a factory for Session object mostly created at the start of the application and acting as a safe thread object. To start running any database operations a transaction on that session must be initialized.

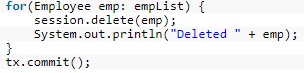
****

*Photo 10. Example of listing existing employee*

In the example above, employee with nameJoeisfetched from the database. In this case Hibernate ***createQuery*** and ***list*** functions have been used. While not using any ORM, to perform such an operation a following SQL query had to be executed:

**SELECT \* FROM Employee WHERE name='Joe;**

Each employee can be also deleted from the database:

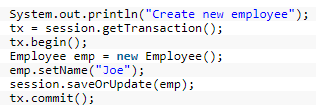


*Photo 11. Example of deleting employees*

The alternative SQL query may look like:

**DELETE FROM Employee**

To create a new employee the following actions should be taken:



*Photo 11. Example of creating new employee*

After a transaction begins, a new instance of Employee persistence class is created. It is an object which will be sent to the database. Then the employee name is set on that object. In order to create or update a new employee simple ***saveOrUpdate*** method is used. Some may feel a little bit confused at the first time. What ***saveOrUpdate*** reallydoes is:

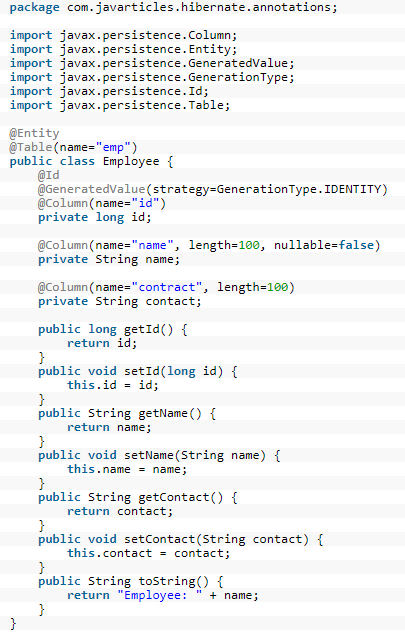
* Saving an object if it has no identifier property.
* Updating an object if it has an identifier property.

For comparison the following SQL query had to be done while creating new employee:

**INSERT INTO Employee (name, contract) VALUES ('Joe', null)**

Hibernate framework also supports using annotations. The annotations start with @ sign and can include the following information:

* Specifying details of the table in the database.
* Defining which field or property will be mapped.
* Specifying relations for tables.
* Determining which generator type to use.



*Photo 12. Exampleof using Hibernate annotations*

**ORM - pros and cons**

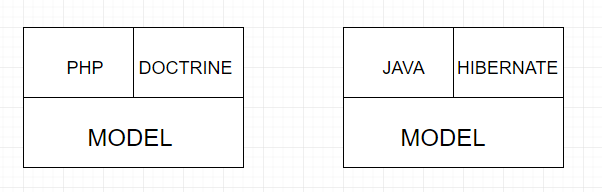
One of the biggest advantages of using ORM is that it follows the DRY rule. DRY stands for - Don't repeat yourself. This principle reduces the repetition of different patterns during a software development process.

Another point worth to stress is connected to the model. There is only a one database model allocated in one place in the project and thanks to it, this model is easier to maintain. What is more, it is possible to use considering piece of code in any place in the project. This rule actually comes to one of the most popular design patterns, namely MVC (Model- View- Controller). Many things are being performed automatically while using ORM, so developers can focus more precisely on the other aspects.

Flexibility is another big pros of using this kind of tool. For example, there is a possibility to easily change a database management system without any complications. Additionally, the beginning developers are not obliged to know how to create more advanced SQL queries.

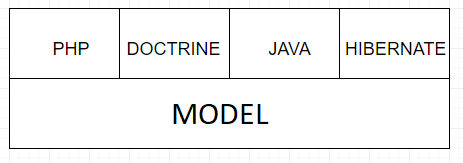
Regards to the disadvantages of an ORM tool, it has to be said that there is a lot of manual configuration while starting using such a framework. Referring to the efficiency, it has been examined that more complicated queries are executed faster by the plain SQL.

There are different ORM frameworks and libraries on the market for various programming languages. Each ORM proposes its own way of describing the database model. The problem comes when there are many programming teams working on the same project. It is supposed that each team use different programming languages and concrete technologies. In that case considering models will differ. The problem is clearly presented on the picture below.



*Photo 13. Example of different ORM models*

The idea of the implemented library by the author is to create a unified text base representation for describing the database model. Based on that a common database model will be created. The main goal is to prevent the necessity of making changes in many places. Then all considering platforms can share the same model. This approach is illustrated in the scheme below.



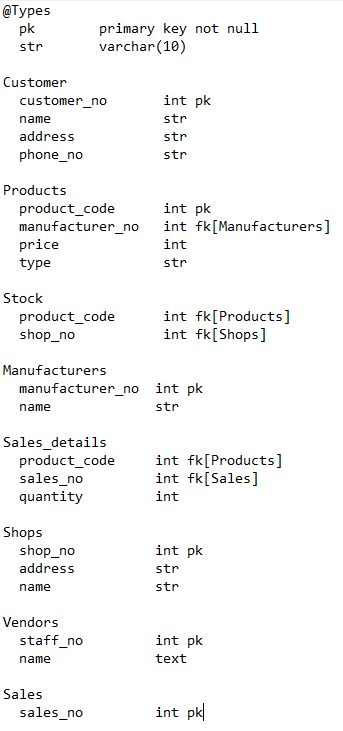
*Photo 13. Example of sharing the same model*

To sum up, it should be mentioned that using ORM may not always be useful, especially while working with massive databases because it can require a lot of work. On the other hand, it is particularly a good solution if programmers want to avoid writing long SQL queries.

**Model representation**

In this chapter the exemplary database model representation used in the project library will be introduced. All required constraints will be explained.

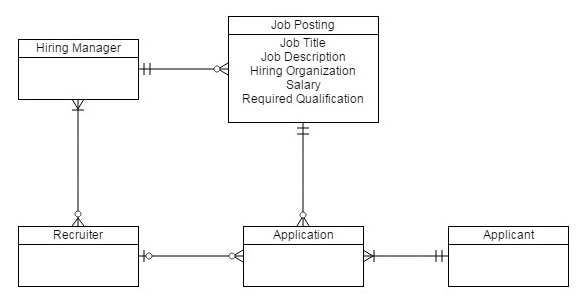
The proposed solution by the author of this thesis is based on the input text file. Such a file contains all entities within its attributes and constraints describing the database schema. It must be mentioned that the PostgreSQL database engine has been used in this project.



*Photo 14. Description of the database model*

As it can be seen, each entity name starts with a capital letter. Such an entity contains all its attributes and specified constraints which are parsed into a PostgreSQL data types later on. What is more, it is possible to define custom data types in order to save user's time. For example primary key for entities as a *pk* and varchar type as a *str* variables have been declared. All custom elements are stored in the section named *@Types*. Relations between tables are defined just by using a *fk* identifierand the name of related table in square brackets. JavaScript library implemented by the author of this thesis is responsible for reading such a text file, then parsing its content and finally creating a corresponding database schema.

This solution can be an alternative for famous Entity-Relational Diagram (ERD). ERD is a kind of graphical representation of relationships between entities, used during designing complex systems and database structures.



*Photo 14. Example of Entity-Relational Diagram*

**4. Badania**

**Model consistency**

Model can constantly change, so the mechanism checking the consistency of the database structure has been implemented. The main role of that library's feature is to catch errors if any inconsistencies between model and database schema are detected. After that, user should be informed about the differences. What is more, possible solution including SQL query required to fix the problem should be displayed. Tables described in *Photo 14* has been used during the tests.

**Changing the entity name**

First of all, the case where the name of the entity has been changed is performed. Changes may occur accidentally or deliberately both in the database or in the defined for database model. In the considering example table name in database has been changed from ***customers*** to ***customer***. Library without any problem finds the difference and responds with the correct SQL query to fix the following problem.



*Photo 15. Library response for entity name change*

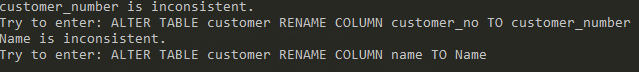
It may be assumed that someone dropped a table directly in the database. For instance, table customer has been removed. While checking the cohesion, the following solution is proposed.

(wkleić nowe zdjęcie)

**Changing the attribute name**

Subsequent example is about to change the entity's attribute name. In order to check the behaviour of the library in such an incident, multiple changes in the model has been applied. Column named ***customer\_no*** is replaced by the ***customer\_number***.

Another modification is changing attribute ***name*** to ***Name*** . Response is shown in the Photo 17.



*Photo 17. Library response for attribute name change*

**Adding the attribute**

Database structure can constantly get bigger and bigger. So, there may be a need of adding a new columns. In this scenario column named ***nip*** has been added in the database model. If there considering column is missing in the database the succeeding alert is displayed.



*Photo 18. Library response for adding new column*

**Removing the attribute**

Contrary to the previous case, removing the column from the model should be presented. Sometimes columns which are no more needed can be removed. It may also happen that one of them has been accidentally deleted. In the following example user is informed that the column ***address*** is redundant.



*Photo 19. Library response for removing column*

**Changing the attribute type**

There are many cases that you want to convert one data type into another for the given entity. It might appear that currently used data type no more meets the newest requirements. Firstly the type for column ***price*** in ***Products*** table is changed from ***float*** of double precision to ***integer***. The library detects the inconsistency and alerts with the following query.



*Photo 20. Library response for changing column type*

The specific situation is when there is an attempt of changing the data type from ***varchar*** to any other type. The character varying type cannot be implicitly transformed and a special cast has to be applied. I that example ***varchar***data type has been changed into the numeric one.



*Photo 21. Library response for changing column type with cast*