# Workshop: Creating large DB Part 1

This exercise is part of the [“Databases Frameworks” course @ SoftUni](https://softuni.bg/courses/databases-advanced-hibernate).

## Judge System

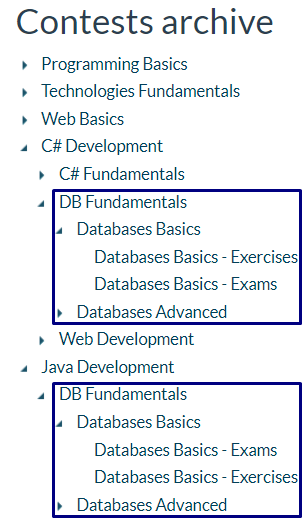
For the following assignment**,** we are going to simulate the Judge System database. Its concept is most probably familiar to you: we have **contests** grouped in **categories** and **subcategories**. Each contest has different amount of **problems** to be solved. Solving is done by users **submitting** solutions, which go under **tests** to check if they are correct or not. Each user gets different amount of **points** according to the correctness of his solution. Based on this, a **maximum points submission** is formed – user’s best submission.

In the first part of the assignment we are going to **model the database**. Follow the instructions to properly set table relations and design the entities, which will be persisted.

## Categories

Currently the contests category hierarchy in our well-known Judge System looks like this:

Each category can be **nested** in another to achieve better organization. Therefore, when we implement the category entity we will have a **self-relationship**. Categories will be mandatorily named, but not necessarily unique. For example, the DB Fundamentals in both C# and Java Development categories have the **same subcategories**, but **different contests and problems**:



Each category has:

* Id
* Category to which it **belongs**
* Collection of **subcategories**
* Collection of **contests**

Design the category entity as described.

## Contests

Contests are placed in certain categories. We consider that **the same contest cannot be placed in various categories** regardless name duplication. For example, the **Databases Basics** subcategory in both **C#** or **Java Development categories** has same contests – Data Definition and Data Types, Basic CRUD and so on, but **the problems in them differ**, therefore we consider them **different**. Each contest has:

* **Id**
* **Name**
* **Category**
* Collection of **problems** – many problems can belong to one contest and vice versa
* Collection of **contestants** – many users can practice same contests
* Collection of **maximum results per contest**

The maximum results per contest will be a collection containing the maximum results for the users who have practiced the contest. A **maximum result for a user per contest will be calculated by the sum of all problems score for each user.** It will be an entity keeping information about the user, who made the result, the contest in which the result is made, the average performance of submissions and the points sum.

E.g.: For a contest containing 3 problems, a user has submitted 10 times. The maximum result is formed by **getting the maximum result submission for each problem**, **summing the points scored** from those submissions and **calculating the average performance** from the same submissions.

## Problems

The problems which will be solved are the main part in our mini application. Each problem has:

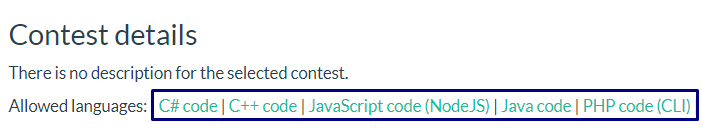
* **Id**
* **Name**
* Collection of **submissions**
* Collection of **contestants** – the user who have practiced the problem
* **Contest** to which the problem belongs
* Collection of **user maximum results for a problem**

The **maximum results per user for a problem** are a very important optimization for us. The entity can be, of course, skipped but let’s look after the following scenario: A user submits his solution for the current problem. If we want to calculate the maximum results for the problem, we must traverse every single submission for the problem in the database and check if the current score is the maximum. This will not only be **slow**, but **very slow**. Therefore, we can solve the situation by keeping another entity – **user maximum result per problem** and skip the whole traversing of the database. This entity will give us information about the **user**, the **problem** and the **submission**, which gave maximum score. With that, we will check for information in smaller table in our data source, and not go through all submissions – we will always have one that will be the maximum we need.

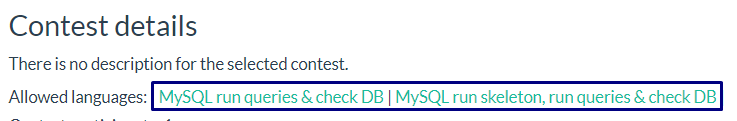
## Strategies

The strategies for a contest is the ways for solving problems in it. For example:

In a programming basics exam we can submit code with by:



And during the MySQL course we could submit solutions by:



Each strategy has a **name** and **contests** which use it. Many contests can use the same strategy.

## Submissions

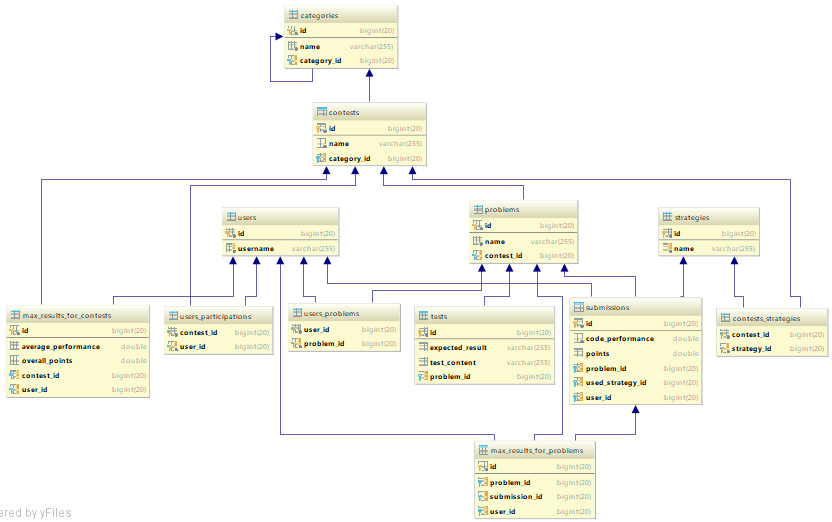
Submissions hold the solutions results that the users in the system submit. Each submission keeps information about the **user** who has submitted, the **used strategy** to solve the problem, the **performance** of the solution in milliseconds, **points** which have been achieved by the solution and the **problem**.

## Users

A used is the person registered in the Judge System. Each user has an **id**, **unique username**, collections of **submissions**, **maximum results for problems**, **maximum results for contests**, **contests** in which the user has participated and **problems** solved.

## Database Schema

If you’ve modelled the entities and their relations correctly, the database schema should look like this:



In the next part of the workshop we will add validation to our entities, implement services and controllers and add some functionalities.