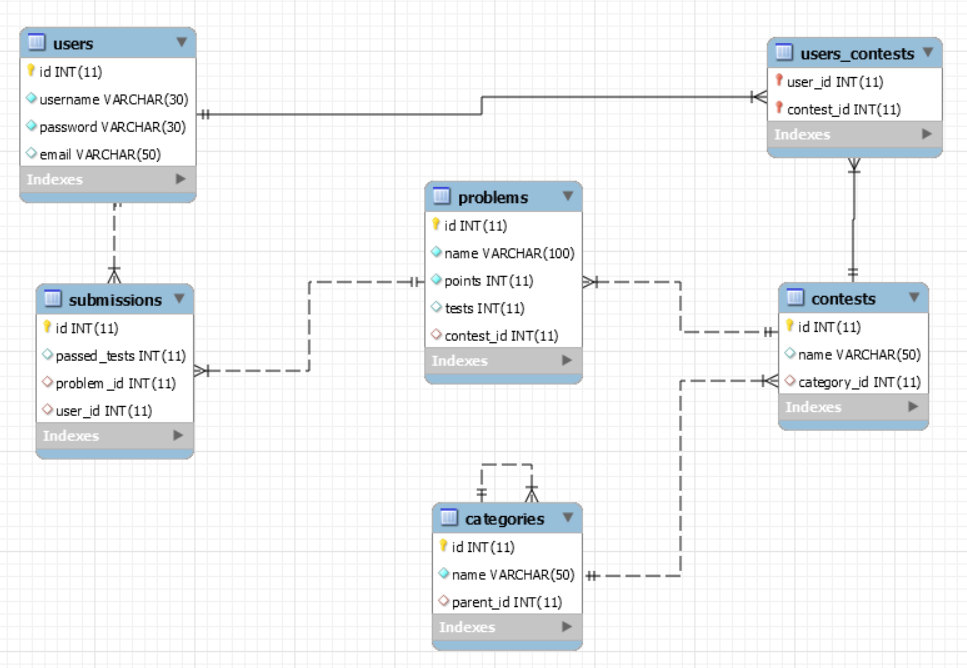
# Database Basics (MySQL) Practical Exam Closed Judge System

Ivo is very creative and he likes to create problems for exercises and exams for the students. But his problems are often too hard and complex so they are not really ready for the Open Judge System, and so he created the Closed Judge System. The Closed Judge System needs a database, but Ivo cannot do that, so you’ll have to help him.

## Section 0: Database Overview

You have been given an Entity / Relationship Diagram of the CJS Database:

****

The CJS Database needs to hold information about users, categories, contests, problems and submissions.

Your task is to create a database called closed\_judge\_system. Then you will have to create several **tables**.

* users – contains information about the **users**.
* categories – contains information about the **categories**.
  + Each category may have a parent category.
* contests – contains information about the **contests**.
  + Each contest has a category.
* problems – contains information about the **problems**.
  + Each problem has a contest.
* submissions – contains information about the **submissions**.
  + Each submission has a problem.
  + Each submission has a user.
* users\_contests – a mapping table between the **users** and the **contests**.

## Section 1: Data Definition Language (DDL) – 40 pts

Make sure you implement the whole database correctly on your local machine, so that you could work with it.

The instructions you’ll be given will be the minimal needed for you to implement the database.

### Table Design

You have been tasked to create the tables in the database by the following models:

**users**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | **Integer,** from **1** to **2,147,483,647.** | **Primary Key** |
| username | A **string** containing a maximum of **30 characters**. Unicode is **NOT** needed. | **NULL** is **NOT** permitted**. UNIQUE** values. |
| password | A **string** containing a maximum of **30 characters**. Unicode is **NOT** needed. | **NULL** is **NOT** permitted**.** |
| email | A **string** containing a maximum of **50 characters**. Unicode is **NOT** needed. |  |

**categories**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | **Integer**, from **1** to **2,147,483,647.** | **Primary Key** |
| name | A **string** containing a maximum of **50 characters**. Unicode is **NOT** needed. | **NULL** is **NOT** permitted. |
| parent\_id | **Integer**, from **1** to **2,147,483,647.** | Relationship with table categories. |

**contests**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | **Integer**, from **1** to **2,147,483,647.** | **Primary Key** |
| name | A **string** containing a maximum of **50 characters**. Unicode is **NOT** needed. | **NULL** is **NOT** permitted. |
| category\_id | **Integer**, from **1** to **2,147,483,647.** | Relationship with table categories. |

**problems**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | **Integer**, from **1** to **2,147,483,647.** | **Primary Key** |
| name | A **string** containing a maximum of **100 characters**. Unicode is **NOT** needed. | **NULL** is **NOT** permitted. |
| points | **Integer**, from **1** to **2,147,483,647.** | **NULL** is **NOT** permitted. |
| tests | **Integer**, from **1** to **2,147,483,647.** | Has a **DEFAULT** value of **0**. |
| contest\_id | **Integer**, from **1** to **2,147,483,647.** | Relationship with table contests. |

**submissions**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | **Integer**, from **1** to **2,147,483,647.** | **Primary Key**  **AUTO INCREMENT** |
| passed\_tests | **Integer**, from **1** to **2,147,483,647.** | **NULL** is **NOT** permitted. |
| problem\_id | **Integer**, from **1** to **2,147,483,647.** | Relationship with table problems. |
| user\_id | **Integer**, from **1** to **2,147,483,647.** | Relationship with table users. |

**users\_contests**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| user\_id | **Integer**, from **1** to **2,147,483,647.** | **Primary Key**  Relationship with table users. |
| contest\_id | **Integer**, from **1** to **2,147,483,647.** | **Primary Key**  Relationship with table contests. |

Submit your solutions in Judge on the first task. Submit **all** SQL table creation statements.

You will also be given a data.sql file. It will contain a **dataset** with random data which you will need to **store** in your **local database**. This data will be given to you so you will not have to think of data and lose essential time in the process. The data is in the form of **INSERT** statement queries.

## Section 2: Data Manipulation Language (DML) – 30 pts

Here we need to do several manipulations in the database, like changing data, adding data etc.

### Data Insertion

You will have to **INSERT** records of data into the submissions table, based on the problems table. For problem with id between **1** and **10**, insert data in the submissions table with the following values:

* passed\_tests – **POWER** the **length** of the name of the problem by **3** and take the **SQUARE ROOT** of it. **SUBTRACT** from the **resulting value** the **LENGTH** of the problem’s name.
  + **ROUND** the resulting value **UP**.
* problem\_id – the problem’s id.
* user\_id – **MULTIPLY** the id of the problem by **3** and **DIVIDE** it by **2**.
  + **ROUND** the resulting value **UP**.

### Data Update

**UPDATE** all problems which have tests equal to **0**. **DIVIDE** the problem’s id by **3** and take the **REMAINDER**:

* If the **remainder** is **0** set the column tests to the **LENGTH** of the problem’s contest’s category’s name.
* If the **remainder** is **1** set the column tests to the **SUM** of the ids of the problem’s submissions.
* If the **remainder** is **2** set the column tests to the **LENGTH** of the problem’s contest’s name.

### Date Deletion

Ivo doesn’t like people who just stay and watch. **DELETE** all users which do **NOT** participate in any contest.

## Section 3: Querying – 100 pts

And now we need to do some data extraction. **Note** that the **example results** from **this section** use a **fresh database**. It is **highly recommended** that you **clear** the **database** that has been **manipulated** by the **previous problems** from the **DML** **section** and **insert again** the **dataset** you’ve been given, to ensure **maximum consistency** with the **examples** given in this section.

### Users

Extract from the database, all of the users.

**ORDER** the results **ascending** by user id.

#### Required Columns

* id (users)
* username
* email

#### Example

|  |  |  |
| --- | --- | --- |
| **id** | **username** | **email** |
| 1 | density | default@gmail.com |
| ... | ... | ... |

### Root Categories

Extract from the database, all **root** categories. A root category is a category which has **NO** **parent**.

**ORDER** the results **ascending** by category id.

#### Required Columns

* id (categories)
* name (categories)

#### Example

|  |  |
| --- | --- |
| **id** | **name** |
| 1 | JavaScript Development |
| ... | ... |

### Well Tested Problems

**Extract** from the database, all problems, which have **MORE** tests **THAN** points, and their name consists of **two words**.  
The **two words** will always be **separated** by a **single space**.

**ORDER** the results **descending** by problem id.

#### Required Columns

* id (problems)
* name (problems)
* tests

#### Example

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **tests** |
| 43 | Test Problem | 99 |
| ... | ... | ... |

### Full Path Problems

**Extract** from the database, all problems and the full path to them. The full path is the **concatenation** of the category’s name, contest’s name and problem’s name in the following format:

{category.name} – {contest.name} – {problem.name}

You will have to take the contest in which the problem is and the category in which the contest is.

**ORDER** the results **ascending** by problem id.

#### Required Columns

* id (problems)
* full\_path

#### Example

|  |  |
| --- | --- |
| **id** | **full\_path** |
| 1 | C# OOP Advanced - C# OOP Exercises - NeedForSpeed |
| ... | ... |

### Leaf Categories

**Extract** from the database, all leaf categories. A leaf category is a category which has **NO** **children**. In other words, there is no category which has the corresponding category as its parent.

**ORDER** the results in **alphabetically** by category name, and **ascending** by category id.

#### Required Columns

* id (categories)
* name (categories)

#### Example

|  |  |
| --- | --- |
| **id** | **name** |
| 15 | C# Advanced |
| ... | ... |

### Mainstream Passwords

**Extract** from the database, all users, which have exactly the **SAME** password as another user.

**ORDER** the results **alphabetically** by username, and **ascending** by user id.

#### Required Columns

* id (users)
* username
* password

#### Example

|  |  |  |
| --- | --- | --- |
| **id** | **username** | **password** |
| 5 | andrew | karuusel |
| ... | ... | ... |

### Most Participated Contests

**Extract** from the database, the **top 5** contests in terms of **count** of **their contestants**.

**ORDER** the result **ascending** by count of contestants, and **ascending** by contest id.

#### Required Columns

* id (contests)
* name (contests)
* contestants

#### Example

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **contestants** |
| 1 | JS Exercises | 6 |
| ... | ... | ... |

### Most Valuable Person

Extract from the database, all submissions of the user that participates in the most contests.

For each submission (**if there are any**) extract the problem’s name.   
Concatenate passed\_tests from the submission and tests from the problem in the following format: passed\_tests / tests as result.

**ORDER** the results **descending** by submission id.

#### Required Columns

* id (submissions)
* username (users)
* name (problems)
* result

#### Example

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **username** | **name** | **result** |
| 81 | BabyYouAreMine | Mass Effect | 6 / 10 |
| ... | ... | ... | ... |

### Contests Maximum Points

Extract from the database, all contests. For each contest, extract the maximum possible points (**sum** of all points of its problems **IF THERE ARE ANY**).

**ORDER** the results **descending** by maximum possible points, and **ascending** by contest id.

#### Required Columns

* id (contests)
* name (contests)
* maximum\_points

#### Example

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **maximum\_points** |
| 30 | Khaalesi Dies | 10129 |
| ... | ... | ... |

### Contestants’ Submissions

**Extract** from the database, all of the contests and the **COUNT** of submissions in their problems   
(**IF THERE ARE ANY**).   
Take **ONLY** the submissions which’s users are still **PARTICIPATING** in the contest.

**ORDER** the results **descending** by **count** of submissions, and **ascending** by contest id.

#### Required Columns

* id (contests)
* name (contests)
* submissions

#### Example

|  |  |  |
| --- | --- | --- |
| **id** | **name** | **submissions** |
| 22 | When you are comming up with tha data | 3 |
| ... | ... | ... |

## Section 4: Programmability – 30 pts

Ivo wants to make the server database a bit more … dynamic. So he wants you to write several procedures.

You will be **given an extra table** which works as a cache for the **logged** in users. The table has the **same** **structure** as the users table and is called logged\_in\_users.

You will also be given **another extra table** which works as a cache for the **evaluated expressions**. The table’s structure is shown below:

**evaluated\_submissions**

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| id | **Integer**, from **1** to **2,147,483,647.** | **Primary Key**  **AUTO INCREMENT** |
| problem | A **string** containing a maximum of **100 characters**. Unicode is **NOT** needed. | **NULL** is **NOT** permitted. |
| user | A **string** containing a maximum of **30 characters**. Unicode is **NOT** needed. | **NULL** is **NOT** permitted. |
| result | **Integer**, from **1** to **2,147,483,647.** | **NULL** is **NOT** permitted. |

You will be given the SQL CREATE statements for both tables.

### Login

Create a stored procedure udp\_login which accepts the following parameters:

* username
* password

And checks the following things:

If the username does **NOT** exist in the users table:

Throw an exception with error code ‘45000’ and message ‘Username does not exist!’.

If the username exists, but the password is **NOT** the same:

Throw an exception with error code ‘45000’ and message ‘Password is incorrect!’.

If the username already exists in the logged\_in\_users table:

Throw an exception with error code ‘45000’ and message ‘User is already logged in!’.

If **all checks pass**, extract the id and the email of the corresponding user, from the users table and **INSERT** it into the logged\_in\_users table.



#### Result

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **username** | **password** | **email** |
| 27 | doge | doge | doge@doge.doge |
| ... | ... |  | ... |

### Evaluate Submission

Create a stored procedure udp\_evaluate which accepts the following parameters:

* id (submissions)

And evaluates the points which a submission should earn based on its passed\_tests and the tests and points of the problem it is submitted to.

If there is **NO** submission with the **given id**:

Throw an exception with error code ‘45000’ and message ‘Submission does not exist!’.

If everything passes you should **INSERT** into the evaluated\_submissions table the submission, with its given id, the problem’s name, the user’s username and the result of the submission.

The result of the submission is calculated by the following formula:

{problem.points} / {problem.tests} \* {submission.passed\_tests}

**ROUND** the resulting value **UP**.

The problem being the corresponding problem to which the submission is submitted.



#### Result

|  |  |  |  |
| --- | --- | --- | --- |
| **id** | **problem** | **username** | **result** |
| 1 | Emergency | QuandoImmersia | 2 |
| ... | ... |  | ... |

## Section 5: Bonus – 20 pts

### Check Constraint

Create a trigger, that will help you **INSERT** more adequate values into the problems table.

Upon the **insertion** of a record into the problems table, the trigger should check the following things:

If the given name does **NOT** contain at least **ONE space** (‘ ’):

Throw an exception with error code ‘45000’ and message ‘The given name is invalid!’.

If the given points are **LESS THAN** or **EQUAL** to **0:**

Throw an exception with error code ‘45000’ and message ‘The problem’s points cannot be less or equal to 0!’.

If the given tests are **LESS THAN** or **EQUAL** to **0**:

Throw an exception with error code ‘45000’ and message ‘The problem’s tests cannot be less or equal to 0!’.

If all checks pass successfully, the record should be successfully inserted into the table problems.

Submit **ONLY** the **trigger code**.

The name of the trigger can be anything.