DOCUMENTATION OF THE DATABASE
Designing a database of an online contestor
Epoka University

Team members:

- Kevin Kollçaku (team leader)
- Joana Jaupi
- Bruno Bajo
- Redi Ceni
- Rafaela Gjoshe
- Mirsada Halili

<1> INTRODUCTION

In this project we were asked to build the database of a “contestant” page. A page where everyone can login and solve programming problems, participate in contests, comment on a question.

In order for us to determine what kind of data our database would keep track of we gathered together to analyze such webpages. We based our research on acm.epoka.edu.al and leetcode.com. We then came up with some requirements for our database, which would better reflect what our database had to keep track of.

After analyzing our database we started to consider important aspects that needed to be reevaluated and improved. Such aspects would be data integrity in our database, the security of our information and also the optimization of our database, such that data retrieval could be performed in an efficient manner.

<2> REQUIREMENTS

There will be two types of members in our database, users and staff. To make this easier to store and read data, we will create a base entity PERSON, which saves general information about the members: the personid (the primary key for this entity, automatically incremented), birthday, firstName and lastName (which will be concatenated into one composite attribute called nameSurname), email, gender, the unique username, age derived from the birthday attribute, the region they're from and a password. In order to keep our member data safer, the passwords will be saved in another entity, called PASSWORD, with the attributes of hash and salt, used in password encryption, and a personid as a foreign key from the person table to connect our two tables. One PERSON must have exactly one one PASSWORD and one PASSWORD must belong to exactly one PERSON. We also save the region data in a different table called REGION, with the regionName and the regionID attributes, where the regionID primary key becomes a foreign key in the PERSON entity. One PERSON must have exactly one REGION, while a REGION can have zero to many PERSONs. A person can be either a USER or a STAFF. USER and STAFF must be exactly one PERSON.

The members who are staff are the people who write the questions. The STAFF entity has the attributes: the primary key is the personID, taken as a foreign key from

the PERSON entity, as well as the institution and the accessID. AccessID shows the access level of the staff member, whose accessName is saved in the ACCESSLEVEL entity. Each STAFF has exactly one ACCESSLEVEL and each ACCESSLEVEL has one or more STAFF members. STAFF is also directly connected to the QUESTIONs entity. A STAFF member can write zero or more QUESTIONs, but a QUESTION must have exactly one STAFF member that has written it.

Differently from staff members, users can make submissions, add comments and participate in contests. USER has the attributes dateCreated and rank, where the rank names are saved in a separate entity, RANK, and the user saves the rankID as a foreign key. The primary key of this entity is the personID, foreign key from the PERSON entity.

A USER can write zero or more COMMENTs, while a COMMENT must be written by exactly one USER. The attributes of the COMMENT entity are: commentID (primary key), commentText, date, upVote, downVote. The COMMENT entity is also connected to the QUESTIONs entity. A COMMENT must be in exactly one QUESTION, but a QUESTION can have zero or more COMMENts.

A USER can submit zero or more SUBMISSIONs, but a SUBMISSION must always be submitted by exactly one USER. A SUBMISSIONs attributes are as follows: dateOfSubmission, userAnswer, successOrNot, questionID (foreign key from the QUESTION entity), personID (foreign key from the USER entity) languageID, errorID and errorType. The primary key is a combination of the personID, questionID and the dateOfSubmission. SUBMISSIONs must be sent to exactly one QUESTION, while a QUESTION can have zero or more SUBMISSIONs.

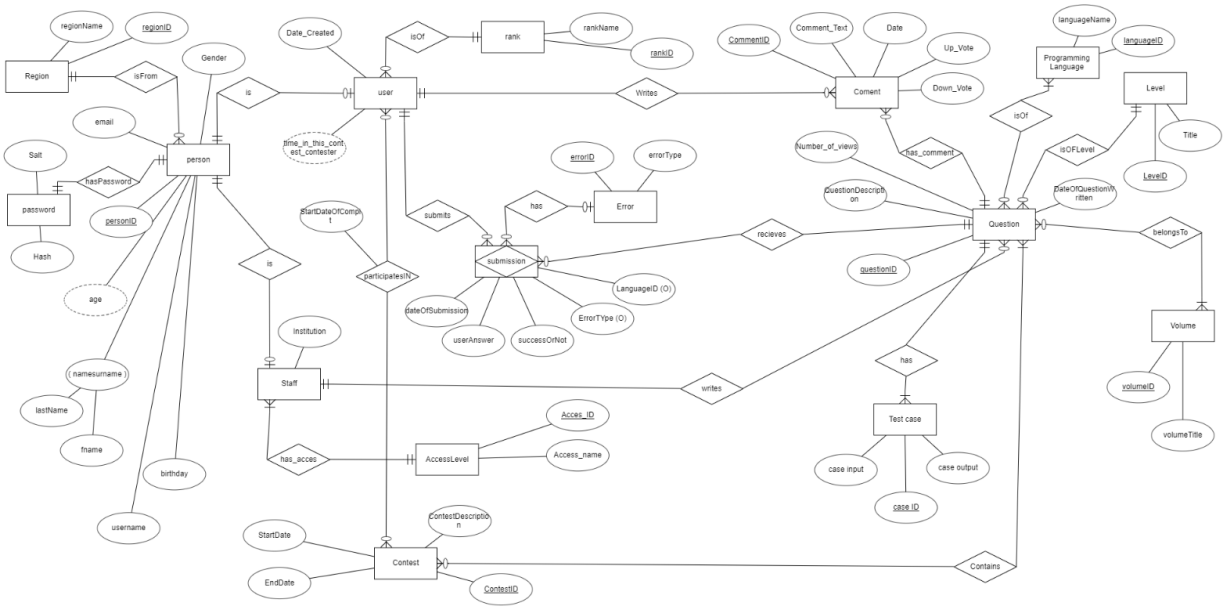
A USER can take part in zero or more CONTESTs and a CONTEST can have zero or more USERs participating in it.

A CONTEST has the following attributes: contestID (primary key), startDate, endDate and contestDescription. A CONTEST must contain one or more QUESTIONs, but a QUESTION can be part of zero or more CONTESTs.

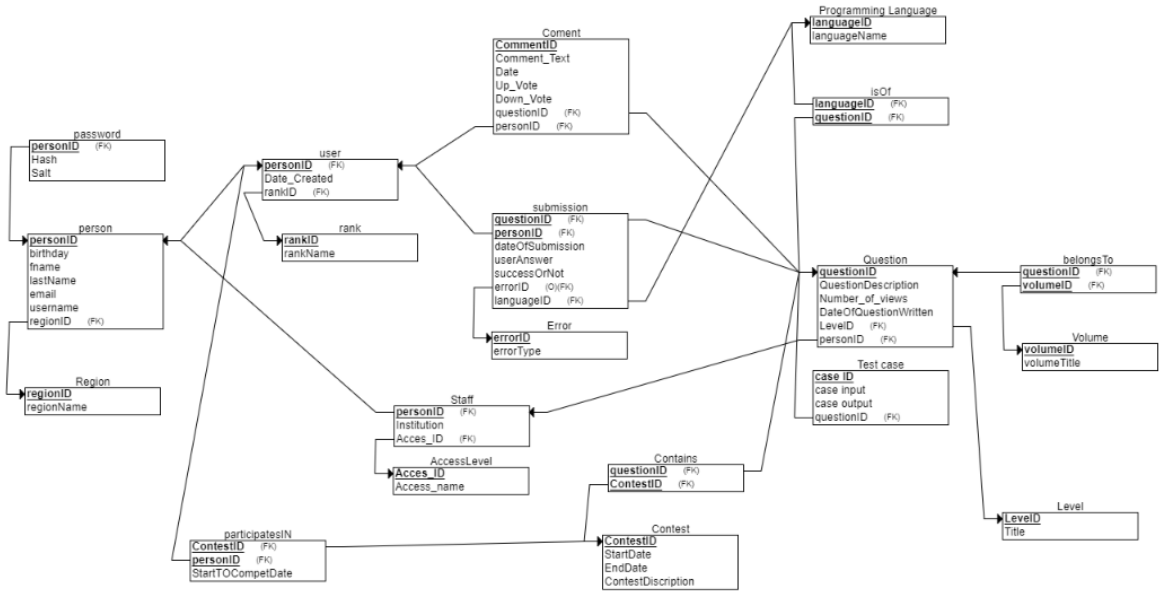
QUESTION is one of our most important entities. It has relations with the COMMENT, SUBMISSION, STAFF, CONTEST entities, which we already mentioned, as well as relations with the PROGRAMMING_LANGUAGE, LEVEL, VOLUME and TEST_CASE entities. QUESTIONs attributes are: questionID (primary key), questionTitle, questionDescription, numberOfViews, dateOfWriting, levelID (foreign key from the LEVEL entity), personID (foreign key from the STAFF entity).

<3> ERD DIAGRAM

This is the ERD diagram that we designed based on the above requirements.



<4> RS DIAGRAM



<5> NORMALIZATION PROCESS

We will first outline all the tables, their attributes and the dependencies in order to start checking in which normalization form they are:

(1) **Person**: personId(**PK**), birthday, firstName, lastName, email, username, regionId(**FK**).

Dependencies:

- **Full Dependencies**
  personId -> birthday, firstname, lastname, email, username, regionId
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Person** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Person** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Person** is in the form 3NF because there are no transitive dependencies.

(2) **User**: personId(**PK**,**FK**), DateCreated, rankId(**FK**)

Dependencies:

- **Full Dependencies**
  personId -> DateCreated, rankId(**FK**)
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **User** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **User** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **User** is in the form 3NF because there are no transitive dependencies.

(3) Staff: personId(**PK**,**FK**), Institution, Access_Id(**FK**)

Dependencies:

- **Full Dependencies**
  personId -> Institution, Access_Id(**FK**)
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Staff** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Staff** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Staff** is in the form 3NF because there are no transitive dependencies.

(4) Password: personId(**PK**,**FK**), Hash, Salt

Dependencies:

- **Full Dependencies**
  personId -> Hash, Salt
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Password** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Password** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Password** is in the form 3NF because there are no transitive dependencies.

(5) Region: regionId(**PK**), RegionName

Dependencies:

- **Full Dependencies**
  regionId -> RegionName

- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Region** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Region** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Region** is in the form 3NF because there are no transitive dependencies.

(6) Rank: rankId(**PK**), RankName

Dependencies:

- **Full Dependencies**
  rankId -> RankName
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Rank** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Rank** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Rank** is in the form 3NF because there are no transitive dependencies.

(7) AccessLevel : AccessId(**PK**), AccessName

Dependencies:

- **Full Dependencies**
  AccessId -> AccessName
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **AccessLevel** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **AccessLevel** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **AccessLevel** is in the form 3NF because there are no transitive dependencies.

(8) Comments: CommentId(**PK**), Comment_Text, Date, Up_Vote, Down_Vote, questionID(**FK)**, PersionId(**FK**)

Dependencies:

- **Full Dependencies**
  CommentId -> Comment_Text, Date, Up_Vote, Down_Vote, questionID, PersionId
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Comments** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Comments** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Comments** is in the form 3NF because there are no transitive dependencies.

(9) Submission: DateOfSubmission(**PK**), questionId(**PK,FK**), PersonId(**PK,FK**), userAnswer, Success_or_not,errorId(**FK),** LanguageId(**FK**)

Dependencies:

- **Full Dependencies**
  DateOfSubmission, questionId, PersonId -> userAnswer, Success_or_not, errorId, LanguageId
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Submission** is in the form 1NF because there are no multivalued columns within it.

✔ **Form 2NF**: The table **Submission** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Submission** is in the form 3NF because there are no transitive dependencies.

(10) **Error**: ErrorId(**PK**), errorType

Dependencies:

- **Full Dependencies**
  ErrorId -> errorType
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Error** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Error** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Error** is in the form 3NF because there are no transitive dependencies.

(11) **ParticipatesIN**: ContestId(**PK,FK**), personId(**PK,FK**), StartToCompeteDate

Dependencies:

- **Full Dependencies**
  ContestId, personId -> StartToCompeteDate
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **ParticipatesIN** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **ParticipatesIN** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **ParticipatesIN** is in the form 3NF because there are no transitive dependencies.

(12) Question: QuestionId(**PK,FK**), QuestionTitle, QuestionDescription, Number_of_views, DateOfQuestionWritten, LeveId(**FK**), PersonId(**FK**)

Dependencies:

- **Full Dependencies**
  QuestionId -> QuestionDescription, QuestionTitle, Number_of_views, DateOfQuestionWritten, LeveId, PersonId
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Question** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Question** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Question** is in the form 3NF because there are no transitive dependencies.

(13) isOf: QuestionId(**PK,FK**), languageId(**PK,FK**)

Dependencies:

- **Full Dependencies**
  There are none.
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **isOf** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **isOf** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **isOf** is in the form 3NF because there are no transitive dependencies.

(14) Programming Language: LanguageId(**PK**), languageName

Dependencies:

- **Full Dependencies**

LanguageId -> languageName

- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Programming Language** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Programming Language** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Programming Language** is in the form 3NF because there are no transitive dependencies.

(15) **Test Case**: caseId(**PK**), caseInput, caseOutput, questionId(**FK**)

Dependencies:

- **Full Dependencies**
  casedId -> caseInput, caseOutput, questionId
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Test Case** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Test Case** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Test Case** is in the form 3NF because there are no transitive dependencies.

(16) **Contest**: ContestId(**PK**), startDate, EndDate

Dependencies:

- **Full Dependencies**
  ContestId -> startDate, EndDate
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Contest** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Contest** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Contest** is in the form 3NF because there are no transitive dependencies.

(17) Contains: questionId(**PK,FK**), ContestId(**PK,FK**)

Dependencies:

- **Full Dependencies**
  There are none.
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Contains** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Contains** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Contains** is in the form 3NF because there are no transitive dependencies.

(18) Volume: VolumeId(**PK**), volumeTitle

Dependencies:

- **Full Dependencies**
  VolumeId -> volumeTitle
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Volume** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Volume** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Volume** is in the form 3NF because there are no transitive dependencies.

(19) **Level**: LevelId(**PK**), LevelTitle

Dependencies:

- **Full Dependencies**
  LevelId -> LevelTitle
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **Level** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **Level** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **Level** is in the form 3NF because there are no transitive dependencies.

(20) **belongsTo**: questionId(**PK,FK**), volumeId(**PK,FK**)

Dependencies:

- **Full Dependencies**
  There are none.
- **Partial Dependencies**
  There are none.
- **Transitive Dependencies**
  There are none.

✔ **Form 1NF**: The table **belongsTo** is in the form 1NF because there are no multivalued columns within it.
✔ **Form 2NF**: The table **belongsTo** is in the form 2NF because there are no partial dependencies found in it.
✔ **Form 3NF:** The table **belongsTo** is in the form 3NF because there are no transitive dependencies.

<6> DATA INTEGRITY

It's critical that data follow a set of rules established by the database administrator or application developer.
Data Integrity Types
This section explains how we have applied rules to table columns to enforce various sorts of data integrity.

a. **Null Rule**
   A null rule is a rule that allows or disallows inserts or updates of rows that include a null (the absence of a value) in a single column.
   In our database we applied this rule so that no data is inserted into NOT NULL columns or in the other hand no data can be inserted into columns where this rule is not applied to.
b. **Unique Column Values**
   A unique value rule defined on a column (or set of columns) allows the insert or update of a row only if it contains a unique value in that column (or set of columns).
c. **Primary Key Values**
   A primary key value rule defined on a key (a column or set of columns) specifies that each row in the table can be uniquely identified by the values in the key.
d. **Referential Integrity Rules**
   A referential integrity rule is a rule that guarantees that the values in one table's key (a column or set of columns) match the values in another table's key (the referenced value).

   The rules that govern what sorts of data manipulation are permitted on referenced values, as well as how these actions influence dependent values, are referred to as referential integrity. Referential integrity is governed by the following rules:
   D.1 *Restrict: Disallows the update or deletion of referenced data*.

   In the person table we have restricted the deletion of a region record if a region is referenced by a person.

   D.2 *Set to Null:* When referenced data is updated or deleted, all associated dependent data is set to NULL

In our database we have applied this Data integrity rule to users referring to ranks

As observed in our database, in the USER table, rankID references rankID in the rank table. IF the rank is deleted in the parent table we expect for the value of rankID in the users table to be NULL

D3. *Cascade:* When referenced data is updated, all associated dependent data is correspondingly updated. When a referenced row is deleted, all associated dependent rows are deleted.

In our database we made sure that on delete certain data will be deleted accordingly.

D4. *No Action:* Prevents referenced data from being updated or deleted. This varies from RESTRICT in that it is tested at the end of the statement or, if the constraint is delayed, at the end of the transaction.Default action is No Action in MySql.

D5. *Set to default:* When referenced data is updated or deleted, all dependant data linked with it is reset to a default value.

<7> STRUCTURE OF OUR DATABASE

1. REGION table

columnName	dataType	Description
regionId	INT	Keeps track of the ID of regions
regionName	VARCHAR(50)	regions/countries names

```
CREATE TABLE Region
(
  regionID INT NOT NULL auto_increment,
  regionName VARCHAR(50) NOT NULL DEFAULT "Unspecified",
  PRIMARY KEY (regionID)
);
```

2. VOLUME table

columnName	dataType	Description
volumeID	INT	Keeps track of the ID of question volumes (categories of questions)
volumeTitle	VARCHAR(50)	Title of the volume

```
CREATE TABLE Volume
(
  volumeID INT NOT NULL auto_increment,
  volumeTitle VARCHAR(50) NOT NULL unique,
  PRIMARY KEY (volumeID)
);
```

3. LEVEL Table

columnName	dataType	Description
LevelID	INT	Keeps track of the ID of question level(difficulty of the question)
Title	VARCHAR(50)	Title of the level

```
CREATE TABLE Level
(
  Title VARCHAR(50) NOT NULL unique,
  LevelD INT NOT NULL auto_increment,
  PRIMARY KEY (LevelD)
);
```

4. Programming_Language

columnName	dataType	Description
languageID	INT	Keeps track of the ID of question programming language
languageName	VARCHAR(50)	Language name

```
CREATE TABLE Programming_Language
```

```
(
  languageID INT NOT NULL auto_increment,
  languageName VARCHAR(50) NOT NULL,
  PRIMARY KEY (languageID)
);
```

5. CONTEST table

columnName	dataType	Description
StartDate	DATE	Start date of the contest
lEndDate	DATE	End Date of the contest
ContestID	INT	Id of the contest
ContestDescription	TEXT	A description of the contest (title and everything defining the contest

```
CREATE TABLE Contest
(
  StartDate DATE NOT NULL,
  EndDate DATE NOT NULL,
  ContestID INT NOT NULL auto_increment,
  ContestDescription TEXT NOT NULL,
  PRIMARY KEY (ContestID)
);
```

6. Errors_type table

columnName	dataType	Description
errorID	INT	Id of the error
errorType	Varchar(50)	Types of the error(runtime error etc)

```
CREATE TABLE Errors_type
(
  errorID INT NOT NULL auto_increment,
  errorType VARCHAR(50) NOT NULL unique,
  PRIMARY KEY (errorID)
);
```

7. Ranks TABLE

columnName	dataType	Description
rankID	INT	Id of the rank
rankName	Varchar(25)	Name of the person's rank (level on contester, beginner pro etc)

```
CREATE TABLE ranks
(
  rankID INT NOT NULL auto_increment,
  rankName VARCHAR(25) NOT NULL DEFAULT 'Default',
  PRIMARY KEY (rankID)
);
```

8. AccessLevel TABLE

columnName	dataType	Description
Acces_ID	INT	Id of the access level
Access_name	Varchar(25)	Name of the access (such as admin, creator, manager etc)

```
CREATE TABLE AccessLevel
(
  Acces_ID INT NOT NULL auto_increment,
  Access_name VARCHAR(25) NOT NULL unique,
  PRIMARY KEY (Acces_ID)
);
```

9. Person TABLE

columnName	dataType	Description
personID	INT	Id of the person
birthday	DATE	Birthday date of the person
fname	Varchar(25)	First name of the person

lastName	Varchar(25)	Last name of the person
email	Varchar(50)	Email of the person
username	Varchar(25)	Username that the person login(unique)
Gender	Char(1)	Gender of the person
regionID (fk)	INT	Id of the region references the region table

```
CREATE TABLE person
(
  personID INT NOT NULL auto_increment,
  birthday DATE NOT NULL,
  fname VARCHAR(25) NOT NULL,
  lastName VARCHAR(25) NOT NULL,
  email VARCHAR(50) NOT NULL,
  username VARCHAR(25) NOT NULL unique,
  Gender CHAR(1) NOT NULL CHECK(Gender IN ('M', 'F')),
  regionID INT NOT NULL,
  PRIMARY KEY (personID),
  FOREIGN KEY (regionID) REFERENCES Region(regionID)
  ON DELETE RESTRICT
  ON UPDATE CASCADE
);
```

10. Password TABLE

columnName	dataType	Description
Hash	Varchar(64)	Hash of user password+salt using SHA256 algorithm
Salt	varchar(16)	Random generated String
personID (fk)	INT	Id of the person references the person table

```
CREATE TABLE password
(
  Hash VARCHAR(256) NOT NULL,
  Salt VARCHAR(16) NOT NULL,
  personID INT NOT NULL,
```

```
  PRIMARY KEY (personID),
  FOREIGN KEY (personID) REFERENCES person(personID)
  ON DELETE CASCADE
);
```

11. User TABLE

columnName	dataType	Description
Date_Created	DATE	Date when it is created as an user(sign in date)
personID (fk)	INT	Id of the person references the person table
rankID (fk)	INT	Id of the rank (references the ranks table)

```
CREATE TABLE user
(
  Date_Created DATE NOT NULL,
  personID INT NOT NULL,
  rankID INT,
  PRIMARY KEY (personID),
  FOREIGN KEY (personID) REFERENCES person(personID)
  ON DELETE CASCADE,
  FOREIGN KEY (rankID) REFERENCES ranks(rankID) ON DELETE SET NULL
);
```

12. Staff TABLE

columnName	dataType	Description
Institution	Varchar(50)	Name of the institution
personId (fk)	INT	Id of the person references person id of the person table
Acces_ID (fk)	INT	Id of the accessor references access_level table

```
CREATE TABLE Staff
(
  Institution VARCHAR(50) NOT NULL,
  personID INT NOT NULL,
```

```
  Acces_ID INT ,
  PRIMARY KEY (personID),
  FOREIGN KEY (personID) REFERENCES person(personID) ON DELETE CASCADE,
   FOREIGN KEY (Acces_ID) REFERENCES AccessLevel(Acces_ID) ON DELETE SET
NULL
);
```

13. ParticipatesIN TABLE

columnName	dataType	Description
ContestID (fk)	INT	Id of the contest
personID (fk)	INT	Id of the person
StartTOCompetDate	DATE	Date when the person start participating on the contest

```
CREATE TABLE participatesIN
(
  ContestID INT NOT NULL,
  personID INT NOT NULL,
  StartTOCompetDate  DATE NOT NULL,
  PRIMARY KEY (ContestID, personID),
     FOREIGN KEY (ContestID) REFERENCES Contest(ContestID) ON DELETE
CASCADE,
  FOREIGN KEY (personID) REFERENCES user(personID) ON DELETE CASCADE
);
```

14. Question TABLE

columnName	dataType	Description
QuestionDescription	TEXT	Description of the given question
QuestionTitle	TINYTEXT	Title of the question
questionID	INT	Id of the question
Number_of_view	INT	Number of the views that view the question

Date of Questions Written	DATE	Data when the question is written
LevelID	INT	Id level of the question
PersonId (fk)	INT	Id of the person who has written the question. References the person id in the staff table.

```
CREATE TABLE Question
(
  QuestionTitle TINYTEXT NOT NULL,
  QuestionDescription TEXT NOT NULL,
  questionID INT NOT NULL auto_increment,
  Number_of_views INT NOT NULL DEFAULT 0,
  DateOfQuestionWritten DATE NOT NULL ,
  LevelID INT,
  personID INT,
  PRIMARY KEY (questionID),
  FOREIGN KEY (LevelID) REFERENCES Level(LevelID) ON DELETE SET NULL,
  FOREIGN KEY (personID) REFERENCES Staff(personID) ON DELETE SET NULL
);
```

15. Test_case TABLE

columnName	dataType	Description
case_input	TEXT	Input of the question
case_output	TEXT	Output of the question
case_ID	INT	Id of the case
questionID (fk)	INT	Id of the question for the test case. References the question table

```
CREATE TABLE Test_case
(
  case_input TEXT NOT NULL,
  case_output TEXT NOT NULL,
  case_ID INT NOT NULL auto_increment,
  questionID INT NOT NULL,
  PRIMARY KEY (case_ID),
```

FOREIGN KEY (questionID) REFERENCES Question(questionID) ON DELETE CASCADE
);

16. Submission TABLE

columnName	dataType	Description
dateOfSubmission	DATE	Data of the question submission
userAnswer	TEXT	Answer that the user give
successOrNot	BOOLEAN	If the answer is success or not
questionID (fk)	INT	Id of the question
errorID (fk)	INT	Id of the error.
personID (fk)	INT	Id of the person

```
CREATE TABLE submission
(
  dateOfSubmission DATE NOT NULL,
  userAnswer TEXT NOT NULL,
  successOrNot BOOLEAN NOT NULL,
  questionID INT NOT NULL,
  errorID INT,
  personID INT NOT NULL,
  PRIMARY KEY (questionID, personID, dateOfSubmission),
    FOREIGN KEY (questionID) REFERENCES Question(questionID) ON DELETE CASCADE,
  FOREIGN KEY (errorID) REFERENCES Errors_type(errorID),
  FOREIGN KEY (personID) REFERENCES user(personID) ON DELETE CASCADE
);
```

17. Comment TABLE

columnName	dataType	Description
commentId	INT	Id of the comment
comment_Text	TEXT	Comments

DateOfComment	DATE	Date that comment is made
Up_Vote	INT	Number of upvotes that this comment has received
Down_Vote	INT	Number of downvotes that this comment has received
questionID (fk)	INT	Id of the question
personID (fk)	INT	Id of the person

```
CREATE TABLE Comment
(
  CommentID INT NOT NULL auto_increment,
  Comment_Text TEXT NOT NULL,
  DateOfComment DATE NOT NULL,
  Up_Vote INT NOT NULL DEFAULT 0,
  Down_Vote INT NOT NULL DEFAULT 0,
  questionID INT NOT NULL,
  personID INT,
  PRIMARY KEY (CommentID),
    FOREIGN KEY (questionID) REFERENCES Question(questionID) ON DELETE CASCADE,
  FOREIGN KEY (personID) REFERENCES user(personID) ON DELETE SET NULL
);
```

18. BelongsTo TABLE

columnName	dataType	Description
questionID (fk)	INT	Id of the question refers to questionId from question table
volumeID (fk)	INT	Id of the volume refers to the volume id in the volume table

```
CREATE TABLE belongsTo
(
  questionID INT NOT NULL,
  volumeID INT NOT NULL,
  PRIMARY KEY (questionID, volumeID),
```

FOREIGN KEY (questionID) REFERENCES Question(questionID) ON DELETE CASCADE,
FOREIGN KEY (volumeID) REFERENCES Volume(volumeID) ON DELETE CASCADE
);

19. isOf TABLE

columnName	dataType	Description
languageID	INT	Id of the programming language
questionID (fk)	INT	Id of the question

```
CREATE TABLE isOf
(
 languageID INT NOT NULL,
 questionID INT NOT NULL,
 PRIMARY KEY (languageID, questionID),
  FOREIGN KEY (languageID) REFERENCES Programming_Language(languageID) ON DELETE CASCADE,
  FOREIGN KEY (questionID) REFERENCES Question(questionID) ON DELETE CASCADE
);
```

20. Contains TABLE

columnName	dataType	Description
questionID(fk)	INT	Id of the question
ContestID (fk)	INT	Id of the contest

```
CREATE TABLE Contains
(
 questionID INT NOT NULL,
 ContestID INT NOT NULL,
 PRIMARY KEY (questionID, ContestID),
  FOREIGN KEY (questionID) REFERENCES Question(questionID) ON DELETE CASCADE,
```

FOREIGN KEY (ContestID) REFERENCES Contest(ContestID) ON DELETE CASCADE
);

<8> OPTIMIZATION – INDEXING

The best way to improve the performance of SELECT operations in our database is to create indexes on one or more of the columns that are tested in the queries. The index entries act like pointers to the table rows, allowing the query to quickly determine which rows match a condition in the WHERE clause, and retrieve the other column values for those rows.

Primary Key Optimization

The primary key for a table represents the column or set of columns that you use in your most vital queries. Using MySQL, our primary keys have an associated index, for fast query performance. Query performance benefits from the NOT NULL optimization, because it cannot include any NULL values. In this way, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key column or columns.

Primary keys we have used (IDs) serve as pointers to corresponding rows in other tables when you join tables using foreign keys.

Foreign Key Optimization

If a table has many columns, and we query many different combinations of columns, it is efficient to split the less-frequently used data into separate tables with a few columns each, and relate them back to the main table by duplicating the numeric ID column from the main table.

That way, each small table can have a primary key for fast lookups of its data, and we can query just the set of columns that we need using a join operation. Depending on how the data is distributed, the queries might perform less I/O and take up less cache memory because the relevant columns are packed together on disk. (To maximize performance, queries try to read as few data blocks as possible from disk; tables with only a few columns can fit more rows in each data block.)

Unique Field Optimization

MySQL automatically creates a unique index when a unique constraint or primary key is defined for a table. The index covers the columns that make up the primary key or unique constraint and is the mechanism that enforces the constraint.

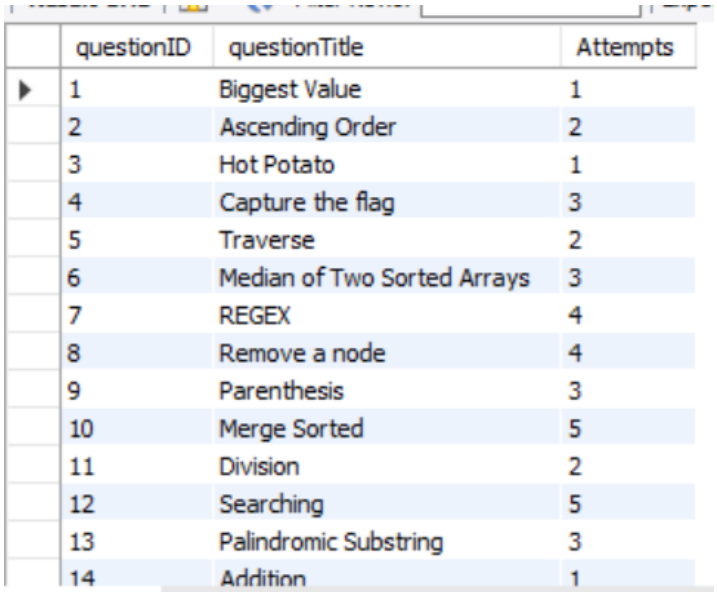
In our database we have declared the username as UNIQUE in the table person. In this way, we have optimized the performance of our queries whenever we

include searching or sorting the username field. Also, the name of each level and error types have a unique constraint.

<9> MANAGERIAL QUERIES AND THEIR RESULTS

1. Show the number of attempts for each question

```
SELECT submission.questionID ,question.questionTitle, COUNT(*) Attempts
FROM submission, question
WHERE submission.questionID = question.questionID
GROUP BY questionid;
```



questionID	questionTitle	Attempts
1	Biggest Value	1
2	Ascending Order	2
3	Hot Potato	1
4	Capture the flag	3
5	Traverse	2
6	Median of Two Sorted Arrays	3
7	REGEX	4
8	Remove a node	4
9	Parenthesis	3
10	Merge Sorted	5
11	Division	2
12	Searching	5
13	Palindromic Substring	3
14	Addition	1

2. Number of successful attempts for all questions

```
SELECT submission.questionID, question.QuestionTitle, COUNT(*)
FROM submission, question
WHERE successOrNot = 1 and submission.questionID = question.questionID
GROUP BY submission.questionid;
```

questionID	QuestionTitle	COUNT(*)
3	Hot Potato	1
4	Capture the flag	2
5	Traverse	1
6	Median of Two Sorted Arrays	2
7	REGEX	2
8	Remove a node	1
9	Parenthesis	1
10	Merge Sorted	2
11	Division	2
12	Searching	1
13	Palindromic Substring	1
15	Conversion from Roman	3
17	Donors	1
18	Balanced parentheses	2

Result 8

3. Number of unsuccessful attempts for a question

```
SELECT submission.questionID, question.QuestionTitle, COUNT(*) unsuccesful
FROM submission, question
WHERE successOrNot = 0 and submission.questionID = question.questionID
GROUP BY submission.questionid;
```

questionID	QuestionTitle	unsuccesful
1	Biggest Value	1
2	Ascending Order	2
4	Capture the flag	1
5	Traverse	1
6	Median of Two Sorted Arrays	1
7	REGEX	2
8	Remove a node	3
9	Parenthesis	2
10	Merge Sorted	3
12	Searching	4
13	Palindromic Substring	2
14	Addition	1
15	Conversion from Roman	1
16	Post-Fix	1

4. Log in of a person (checking the correctness of its password)

```
SELECT *
FROM (person,password)
WHERE person.personID=password.personID
AND  person.username="KingKevin";
```

personID	birthday	fname	lastName	email	username	Gender	regionID	Hash	Salt	
1	2000-01-02	Kevin	Kollcaku	kkollcaku20@epoka.edu.al	KingKevin	M	1	65CEAAB8D57F80D734E21814B0DEDB390CDE...	VDgzzjkTdgAhZDcf	1

5. Select submissions from every specific question from a user with id = 14

```
SELECT questionID, COUNT(*)
FROM submission
WHERE personID=14
GROUP BY questionid;
```

questionID	COUNT(*)
4	1
5	1
10	1
21	1
26	1
27	2
40	1
44	2
50	1
57	1

6. List of students with best score

```
SELECT person.fname, person.personid, COUNT(*)
FROM (submission, user, person)
WHERE(submission.personID = user.personID)
      AND (user.personid = person.personid)
      AND submission.successOrNot = 1
      AND submission.questionid
GROUP BY person.personid
ORDER BY COUNT(*) DESC;
```

fname	personid	COUNT(*)
Michael	14	7
Emily	22	6
Melisa	25	6
Jessica	20	5
James	12	4
Amber	11	4
Diane	31	4
Melisa	24	4
Laura	26	4
Helen	28	4
Robert	37	4
Melisa	23	3
Kyle	33	3
Jessica	21	3

Result 16

7. Show all beginner programmers

```
SELECT user.*, person.fname, ranks.rankName
FROM user, person, ranks
WHERE person.personid = user.personid AND user.rankid = 1 AND ranks.rankID
= user.rankID;
```

Date_Created	personID	rankID	fname	rankName
2021-01-07	12	1	James	Beginner Programmer
2021-10-16	25	1	Melisa	Beginner Programmer
2021-10-20	35	1	John	Beginner Programmer

8. Show all comments for question with id 1

```
SELECT *
FROM comment
WHERE questionID = 1;
```

CommentID	Comment_Text	DateOfComment	Up_Vote	Down_Vote	questionID	personID
1	COMMENT_TEXT	2022-06-10	0	0	1	18
2	COMMENT_TEXT	2022-03-19	0	3	1	17
3	COMMENT_TEXT	2022-02-02	9	5	1	23
4	COMMENT_TEXT	2022-03-24	6	4	1	28
NULL	NULL	NULL	NULL	NULL	NULL	NULL

9. List of questions with the most amount of comments

```
SELECT comment.questionId, question.QuestionTitle,  COUNT(*)
countOfComments
FROM comment, question
WHERE question.questionid = comment.questionid
GROUP BY comment.questionid
ORDER BY COUNT(*) DESC;
```

questionId	QuestionTitle	countOfComments
42	Divisible by 11	6
37	Pythagorean triplets	6
31	Square drawer	6
25	Decimal form of numbers	6
36	Calculator	6
35	Payed amount of a product.	6
50	Pascal triangle	6
28	Show odd numbers between n and m.	6
13	Palindromic Substring	6
47	Encryption	6

Result 30

10. List the regions from highest to lowest number of accepted solutions

```
SELECT r.regionName, COUNT(*)
FROM region r,
    user u,
```

regionName	count(*)
Albania	14
Iran	13
Croatia	11
Bangladesh	8
Israel	7
Denmark	6
Indonesia	6
Brazil	6
Iraq	5
Ethiopia	3
Bulgaria	2
China	1

```
    person p,
    submission s
WHERE s.successOrNot = 1
      AND p.regionID = r.regionID
      AND u.personID = p.personID
      AND u.personID = s.personID
GROUP BY r.regionID
ORDER BY COUNT(*) DESC;
```

11. Show the best contesters from each region

```
CREATE VIEW top_players2 AS SELECT p.username, r.regionName, COUNT(*)
AS cnt
FROM    person p,
        region r,
        submission s,
        user u
WHERE
        s.successorNot = 1
           AND p.regionID = r.regionID
           AND u.personID = p.personID
           AND u.personID = s.personID
    GROUP BY u.personid
    ORDER BY COUNT(*) DESC;

SELECT username, regionName, cnt from
(select username,
regionName,
cnt,
row_number() over
(partition by  regionName ORDER BY top_players2.cnt desc) as player_rank
from top_players2 ) ranks
WHERE player_rank <= 1;
```

username	regionName	cnt
harrismelisa_m	Albania	6
roberto.d31	Bangladesh	4
laurad2	Brazil	4
walkermichael	Bulgaria	1
lindah1	China	1
jessica.j	Croatia	5
Liar	Denmark	4
robert.w	Ethiopia	3
emily.12	Indonesia	6
James	Iran	4
kylec15	Iraq	3
michael_1	Israel	7

12. Show the number of participants which are in a specific contest

```
SELECT ContestDescription, COUNT(*)
FROM contest, participatesIn
WHERE contest.contestID = 1 AND participatesin.contestID = contest.contestID;
```

ContestDescription	count(*)
Smart Solutions	4

13. List all the questions for a specific language

```
SELECT languageName, question.questionTitle, question.QuestionDescription
FROM programming_language, isof, question
WHERE programming_language.languageID = isof.languageID AND
question.questionid = isof.questionid
      AND programming_language.languageID = 3;
```

languageName	questionTitle	QuestionDescription
C++	REGEX	Regular Expression Matching
C++	Remove a node	Remove Nth Node From End of List
C++	Division	Divide Two Integers
C++	Palindromic Substring	Find the Longest Palindromic Substring
C++	Donors	Show the list of the top N donors.
C++	Ticket Time	Calculate the min. total time for all the people to buy tickets
C++	Dvisable numbers	Show numbers divisible from 1 to N.
C++	Calculate the mass of a molecule	Given the number of protons, electrons and neutrons and also their respective mass calculate ...
C++	AVL tree	Write the methods for AVL tree rotation
C++	Calculator	Create a caluclator program. Given N operations output their result.
C++	Pythagorean triplets	Show the number of pythagorean triplets up to N.
C++	Euclidean Distance	Show the max euclidean distance between 2 islands.
C++	Multiplier	Multiply a number with 1-10. Show counter of num with all digits different
C++	Decryption	Decrypt the given text.
C++	Numbers with closest difference	Print the two numbers whose difference is the closest
C++	KMap letter finder	Find which letter A,B,C or D is different in Modified Karnaugh-Map

14. List all the questions for a specific volume.

```
SELECT * FROM belongsto,question
WHERE belongsto.questionID=question.questionID
AND belongsto.volumeID=1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

questionID	volumeID	QuestionTitle	QuestionDescription	questionID	Number_of_views	DateOfQuestionWrit
1	1	Biggest Value	FInd biggest value in the list of integers	1	0	2020-10-07
4	1	Capture the flag	Capture the flag	4	0	2021-04-20
33	1	Profit evaluation	Choose the most profitable share according to equity values	33	0	2020-08-09
47	1	Encryption	Encrypt a password entered by a user	47	0	2020-07-01
49	1	Decryption	Decrypt the given text.	49	0	2020-10-21

15. Show the number of questions for each volume

```
SELECT COUNT(*),volumeTitle
FROM belongsto,question,volume
WHERE belongsto.questionID=question.questionID
AND volume.volumeID=belongsto.volumeID
GROUP BY belongsto.volumeID;
```

count(*)	volumeTitle
5	Arrays
8	AVL Trees
7	Binary Search Trees
4	Competitive Programing
6	Functions
8	Matrices
4	Object Oriented Programming
11	Recursion
6	Sorting Problems

16. List all the questions for a specific question level

```
SELECT  question.QuestionTitle,level.title
FROM level,question
WHERE level.levelID=question.levelID
AND level.levelID=1;
```

	QuestionTitle	title
▶	Biggest Value	easy
	Ascending Order	easy
	Division	easy
	Palindromic Substring	easy
	Addition	easy
	Conversion from Roman	easy
	Weighted average	easy
	N letter	easy
	Decimal form of numbers	easy
	Dvisable numbers	easy
	Square drawer	easy
	Pizzashop	easy
	Numbers with closest di...	ea
	Wait time for a client	easy

17. Show the number of questions for each level

```
SELECT  level.title,count(*)
FROM level,question
WHERE level.levelID=question.levelID
GROUP BY level.LevelID;
```

	title	count(*)
▶	easy	14
	Intermediate	15
	hard	14
	expert	16

18. Most common error in a specific question

```
SELECT e.errorType,count(*)
FROM submission s, errors_type e
WHERE s.successornot=0
AND s.errorID=e.errorID
AND s.questionID=8
GROUP BY e.errorID
ORDER BY COUNT(*) desc
limit 1;
```

	errorType	count(*)
▶	Logic error	2

19. The most common error in all failed submissions

```
select e.errorType,count(*) from submission s, errors_type e
where s.successornot=0
and s.errorID=e.errorID
group by e.errorID
order by count(*) desc
limit 1;
```

errorType	count(*)
Logic error	14

20. List all the users with the highest rank

```
SELECT person.username, ranks.rankname FROM user,person,ranks
WHERE user.rankID=7
and ranks.rankid=user.rankid
and person.personID=user.personID;
```

username	rankname
Liar	Senior Programmer
michael_1	Senior Programmer
taylor.j	Senior Programmer

21. Show the most talkative person (who has commented the most)

```
SELECT person.username,count(*) FROM user,person,comment
WHERE user.personID=person.personID
AND comment.personID=user.personID
GROUP BY person.personID
ORDER BY count(*) desc
limit 1;
```

username	count(*)
lewisd	14

22. Show the comment with most UpVotes for a specific question

```
SELECT comment.comment_text, comment.up_vote FROM comment
WHERE questionID=1
ORDER BY comment.Up_Vote desc
limit 1;
```

	comment_text	up_vote
▶	COMMENT_TEXT	9

23. Show the most active person in contests (with the most participations)

```
SELECT  person.username,  count(*) from participatesin,person
WHERE person.personID=participatesin.personID
GROUP BY participatesin.personID
ORDER BY count(*) desc
limit 1;
```

	username	count(*)
▶	Liar	4

24. List the staff members in descending order with the corresponding number of questions created and having more than 5 questions created.

```
SELECT person.username ,count(*)
FROM question, staff,person
WHERE  person.personid=staff.personid
AND staff.personId=question.personID
GROUP BY  staff.personID having count(*) >5
ORDER BY  count(*) desc;
```

	username	count(*)
▶	Tomas3	13
	KingKevin	7
	b_bajo21	6
	m.halili 21	6

25. Time of membership for each user

```
SELECT u.personid, p.username,
TIMESTAMPDIFF(MONTH, u.date_created ,
CURDATE()) membershipTime
FROM user u, person p
WHERE u.personID = p.personID;
```

personid	username	membershipTime
11	Liar	10
12	James	17
13	rPatt	16
14	michael_1	15
15	miller.m	9
16	chars.s	14
17	lee_mary	5
18	lindah1	13
19	susans.s	17
20	jessica.j	5
21	taylor.j	17
22	emily.12	12
23	melisaw.9	13
24	melissa_rodrigues	6
25	harrismelisa_m	7
26	laurad2	5
27	emmaB.2	14
28	helene	6
29	diane.robinson	8
30	green_diane	14
31	lewisd	6
32	juliej4	15
33	kylec15	6
34	robert.w	6
35	johnh8	7
36	walkermichael	11
37	roberto.d31	17

26. Ages of members

```
SELECT fname, person.lastName,
TIMESTAMPDIFF(YEAR, person.birthday ,
CURDATE()) age
FROM PERSON;
```

fname	lastName	age
Kevin	Kollcaku	22
Joana	Jaupi	19
Jack	Sparrow	21
Rafaela	Gjoshe	21
Bruno	Bajo	22
Mirsada	Halili	20
John	Doe	23
Tomas	Wilson	21
Johnny	Depp	21
Katerina	Grauber	22
Amber	Heard	42
James	Steward	43
Robert	Pattinson	21
Michael	Smith	22
Mattew	Miller	20
Chars	Thomas	21

27. Acceptance rate by country

```
CREATE OR REPLACE VIEW success_points_per_country
AS
SELECT  r.regionname, count(*) as successful_submissions
FROM region as r, submission as s,person as p, user as u

WHERE  r.regionID=p.regionID
AND  s.personID=u.personID
AND  p.personID=u.personID
AND  s.successOrNot=1
GROUP BY  r.regionID
ORDER BY  count(*) desc;

CREATE OR REPLACE view total_submission_by_country
AS
SELECT  r.regionName,count(*) AS total_submissions
FROM person as p
 JOIN region as r
 ON r.regionID=p.regionID
 JOIN user as u
 ON p.personId=u.personID
 JOIN submission as s
 On s.personID=u.personID

 GROUP BY  r.regionID
 ORDER BY count(*) desc;
SELECT  s.regionname,
s.successful_submissions,
t.total_submissions,
s.successful_submissions/t.total_submissions AS average

FROM total_submission_by_country as t, success_points_per_country AS s
WHERE s.regionname=t.regionname
order by average desc;
```

regionname	successful_submissions	total_submissions	acceptance_rate
Indonesia	6	8	0.7500
Albania	14	22	0.6364
Iraq	5	8	0.6250
Israel	7	12	0.5833
Croatia	11	20	0.5500
Bangladesh	8	15	0.5333
Iran	13	25	0.5200
Denmark	6	12	0.5000
Brazil	6	12	0.5000
Ethiopia	3	7	0.4286
Bulgaria	2	5	0.4000
China	1	4	0.2500

28. Top 10 Hardest questions

```
CREATE OR REPLACE VIEW total_submission_to_questions AS
SELECT q.QuestionTitle, COUNT(*) AS total_submissions
FROM question AS q JOIN submission AS s ON q.questionID = s.questionID
GROUP BY q.questionID;
CREATE OR REPLACE VIEW successful_submission_to_questions AS
SELECT q.QuestionTitle, COUNT(*) AS successful_submissions
FROM question AS q  JOIN  submission AS s ON q.questionID = s.questionID
WHERE
      s.successOrNot = 1
  GROUP BY q.questionID;

SELECT
   s.QuestionTitle,
   s.successful_submissions,
   t.total_submissions,
   s.successful_submissions / t.total_submissions AS acceptance_rate
FROM
   successful_submission_to_questions AS s,
   total_submission_to_questions AS t
WHERE
   s.QuestionTitle = t.QuestionTitle
```

```
ORDER BY acceptance_rate
LIMIT 10;
```

QuestionTitle	successful_submissions	total_submissions	acceptance_rate
Searching	1	5	0.2000
Remove a node	1	4	0.2500
Dvisable numbers	1	4	0.2500
Palindromic Substring	1	3	0.3333
Parenthesis	1	3	0.3333
Pascal triangle	1	3	0.3333
Encryption	1	3	0.3333
0s Border	1	3	0.3333
nth digit	1	3	0.3333
Merge Sorted	2	5	0.4000

29. Average age of members

```
SELECT
   AVG(TIMESTAMPDIFF(YEAR,
      person.birthday,
      CURDATE())) AS test
FROM
   person
```

test
22.7838

30. People above average age

```
SELECT person.fname,
   TIMESTAMPDIFF(YEAR,
      person.birthday,
      CURDATE()) above_avg_age
FROM
   person
WHERE
   TIMESTAMPDIFF(YEAR,
      person.birthday,
      CURDATE()) > (SELECT
         AVG(TIMESTAMPDIFF(YEAR,
               person.birthday,

               CURDATE())) AS test
      FROM
         person);
```

fname	above_avg_age
John	23
Amber	42
James	43
Susan	45
Melisa	23
Emma	23
Robert	23

<10> DATABASE SECURITY

Database security refers to the collective measures used to protect and secure a database or database management software from illegitimate use and malicious cyber threats and attacks. Database security procedures are aimed at protecting not just the data inside the database, but the database management system and all the applications that access it from intrusion, misuse of data, and damage.

1. Database security covers and enforces security on all aspects and components of databases. This includes:

- Data stored in the database.
- Database server.
- Database management system (DBMS).
- Other database workflow applications.

Database security is generally planned, implemented and maintained by a database administrator and or other information security professional.

2. Some of the ways database security is analyzed and implemented include:

- Restricting unauthorized access and use by implementing strong and multifactor access and data management controls.
- Load/stress testing and capacity testing of a database to ensure it does not crash in a distributed denial of service (DDoS) attack or user overload.
- Physical security of the database server and backup equipment from theft and natural disasters. Regular data backups can be planned as part of a database security protocol, and multiple copies can be stored off-site to provide redundancy and emergency recovery.
- Reviewing the existing system for any known or unknown vulnerabilities and defining and implementing a road map/plan to mitigate them.
- Data encryption can provide an additional layer of security to protect the integrity and confidentiality of data.

A. Common threats:

Insider Dangers

An insider threat can be an attack on security from any three sources having an access privilege to the database.

- A malicious insider who wants to cause harm
- An insider who is negligent and makes mistakes that expose the database to attack. vulnerable to attacks

- An infiltrator is an outsider who acquires credentials by using a method like phishing or accessing the database of credential information in the database itself.

Human Error

The unintentional mistakes, weak passwords or sharing passwords, and other negligent or uninformed behaviors of users remain the root causes of almost half (49 percent) of all data security breaches

Attacks on Backups

Companies that do not protect backup data using the same rigorous controls employed to protect databases themselves are at risk of cyberattacks on backups.

Buffer overflow exploitations

Buffer overflow occurs when a process attempts to write more data to a fixed-length block of memory than it is allowed to hold. Attackers may use the excess data, stored in adjacent memory addresses, as a foundation from which to launch attacks.

B. Ways to improve your security:

i. Pay attention to insider threats
ii. Train you employees
iii. Limit Employee access to data
iv. Encrypt all devices
v. Testing your security
vi. Delete redundant data
vii. Establish strong passwords
viii. Update your computers and programs regularly
ix. Back-up your data regularly

C. How Can I Deploy Database Security?

There are three layers of database security: the database level, the access level, and the perimeter level. Security at the database level occurs within the database itself, where the data live. Access layer security focuses on controlling who is allowed to access certain data or systems containing it. Database security at the perimeter level determines who can and cannot get into databases. Each level requires unique security solutions.

Security Level	Database Security Solutions
Database Level	• Masking • Tokenization • Encryption
Access Level	• Access Control Lists • Permissions
Perimeter Level	• Firewalls • Virtual Private Networks