

Raport 2 - Project Databases

Raphael Assa (s0102981)
Bruno De Deken (s0080968)
Armin Halilovic (s0122210)
Fouad Kichauat (s0121821)

April 28, 2015

Contents

1	Status	4
1.1	Fase 1	4
1.2	Fase 2	5
2	Design	6
2.1	Keuzes	6
2.1.1	Programmeertaal	6
2.1.2	Database communicatie	6
2.1.3	Python interpreter	6
2.1.4	Oefeningen flow-control	7
2.1.5	Score bepaling	7
2.1.6	Sociaal	7
2.2	Database schema	8
2.2.1	Fase 1	8
2.2.2	Fase 2	9
2.3	UML Schema	10
2.3.1	Fase 1	10
2.3.2	Fase 2	11
3	Product	14
3.1	User Interface	14
3.1.1	Homepage	14
3.1.2	Navigatie	14
3.1.3	Vertalingen	14
3.2	Extra functionaliteit	15
4	Planning	16
5	Appendix	17
5.1	Tables	17
5.2	Triggers	20
5.3	Helpers	23
5.3.1	Exercises	23
5.3.2	Friends	26
5.3.3	Groups	28
5.3.4	Messages	29

5.3.5	Notifications	32
5.3.6	Random	33
5.3.7	Ratings	33
5.3.8	Recommendations	34
5.3.9	Series	35
5.3.10	Statistics	38
5.3.11	Types	45
5.3.12	Users	46

1 Status

1.1 Fase 1

Taken	Afgewerkt	Verantwoordelijke
Ontwerp database (ER-diagram)	X	Volledige groep
Database	X	Raphael
Design database verbeteren	X	Raphael
SQL queries	X	Bruno + Raphael + Armin
Algemeen ontwerp	X	Bruno
Grafisch design	X	Bruno + Armin
Series	X	Raphael
Exercises	X	Bruno
User login	X	Armin
Authorization Control	X	Armin
Python Simulatie	X	Bruno + Fouad
Error systeem	X	Armin
Groups + Friends	X	Raphael + Armin
Answers doorgeven	X	Raphael
Answers controleren	X	Bruno
Messages	X	Armin
Statistieken opvragen	X	Raphael
Grafieken	X	Bruno
Vertalingen	X	Fouad

1.2 Fase 2

Taken	Afgewerkt	Verantwoordelijke
Filteren	X	Armin
Sorteren	X	Armin
Grafieken uitbreiden	X	Bruno + Raphael
Basis aanbevelingen maken	X	Fouad
Notifications/Requests systeem	X	Armin
Database update	X	Raphael + Bruno
Exercises copy	X	Bruno
Exercises reference	X	Bruno
Exercise flow-control	X	Raphael
Exercise UI	X	Bruno
General UI	X	Armin
Homepage design	X	Fouad
Social login	X	Fouad
RegEx checks	X	Bruno
Hide irrelevant data	X	Raphael
Gevorderde aanbevelingen	X	Fouad
Search bar	X	Armin
Debuggen	X	Armin, Bruno, Fouad, Raphael
Testen	X	Armin, Bruno, Fouad, Raphael

2 Design

2.1 Keuzes

2.1.1 Programmeertaal

We zijn begonnen met het PHP framework *Laravel (5.0)*. Aangezien niemand van ons enige ervaring had met webdevelopment of webdesign was het een enorme hulp om een soort template te hebben waar we op konden voortgaan. Een bijkomende reden om Laravel te gebruiken is dat het een zeer uitgebreide en actieve community bevat, evenals veel en duidelijke tutorials. Laravel gebruikt MVC design, wat we dan ook gevolgd hebben. Voor alle grote klassen werd een Controller aangemaakt. Deze Controller bevat de voornaamste php code. De resultaten die in de Controller behaald worden, worden dan gebruikt om Views te creëren die de content geformatteerd renderen (als html code).

2.1.2 Database communicatie

Om met objecten te kunnen werken worden zogenaamde Models aangemaakt. Dit zijn php klassen die gebruikt worden als container voor de data die via SQL queries worden opgevraagd. Bovendien worden deze gebruikt om een zekere mate van beveiliging toe te voegen doordat in deze models gespecificeerd kan worden welke data wel of niet kan aangepast worden.

SQL queries worden verzameld in een enkele file, en worden doorheen het programma gebruikt om te communiceren met de database.

2.1.3 Python interpreter

Als python interpreter hebben we gekozen voor *Skulpt*. Skulpt is een 'in-browser' implementatie van python. De voornaamste voordelen hiervan zijn dat beveiliging veel minder belangrijk is, omdat het kwetsbare systeem voornamelijk de eigen computer van de gebruiker is. Een tweede voordeel is dat de interpretatie van python code geen rekenkracht vergt van de servers. Dit maakt dat het systeem voorbereid is voor gelijktijdig gebruik door een groot aantal gebruikers. Een nadeel van dit systeem is dan weer dat Skulpt een implementatie is van Python, en dus niet hetzelfde is als de werkelijke python interpreter. Hierdoor is skulpt niet hetzelfde als de officiële python interpreter. Na afweging van voor- en nadelen hebben we besloten dat de

voordelen zwaarder doorwegen. Voornamelijk omdat de applicatie dient om te leren programmeren. De nadruk ligt dus op relatief eenvoudige problemen, waardoor de meer gevorderde zaken die eventueel zouden ontbreken niet van toepassing zijn.

2.1.4 Oefeningen flow-control

Reeksen Bij het maken van een reeks, kan een gebruiker slechts in een vaste volgorde de oefeningen oplossen. Op deze manier is er een mate van controle over de volgorde van oplossen. Het is echter niet vereist dat een gebruiker de oefening correct oplost om verder te kunnen gaan, een enkele poging is voldoende.

Individuele oefeningen Bij het oplossen van individuele oefeningen wordt een gelijkaardig systeem toegepast. Om hier een mate van controle te behouden, krijgt een gebruiker slechts de oefeningen te zien die hij mag oplossen. Indien een gebruiker nog geen enkele oefening van een willekeurige reeks gemaakt heeft, zal hij in de lijst van beschikbare oefeningen alleen de eerste oefening van die reeks te zien krijgen.

2.1.5 Score bepaling

Om een zekere uniformiteit en objectiviteit te garanderen wordt iedere oefening binnen een reeks even zwaar gequoteerd. De score die een gebruiker op een reeks behaald is afhankelijk van het percentage van oefeningen binnen die reeks die hij correct heeft opgelost. Door moeilijke reeksen en makkelijke reeksen gelijk te quoteren, is het voor een beginner minder demotiverend om lage scores te halen (vergeleken met zeer gevorderde vrienden bijvoorbeeld).

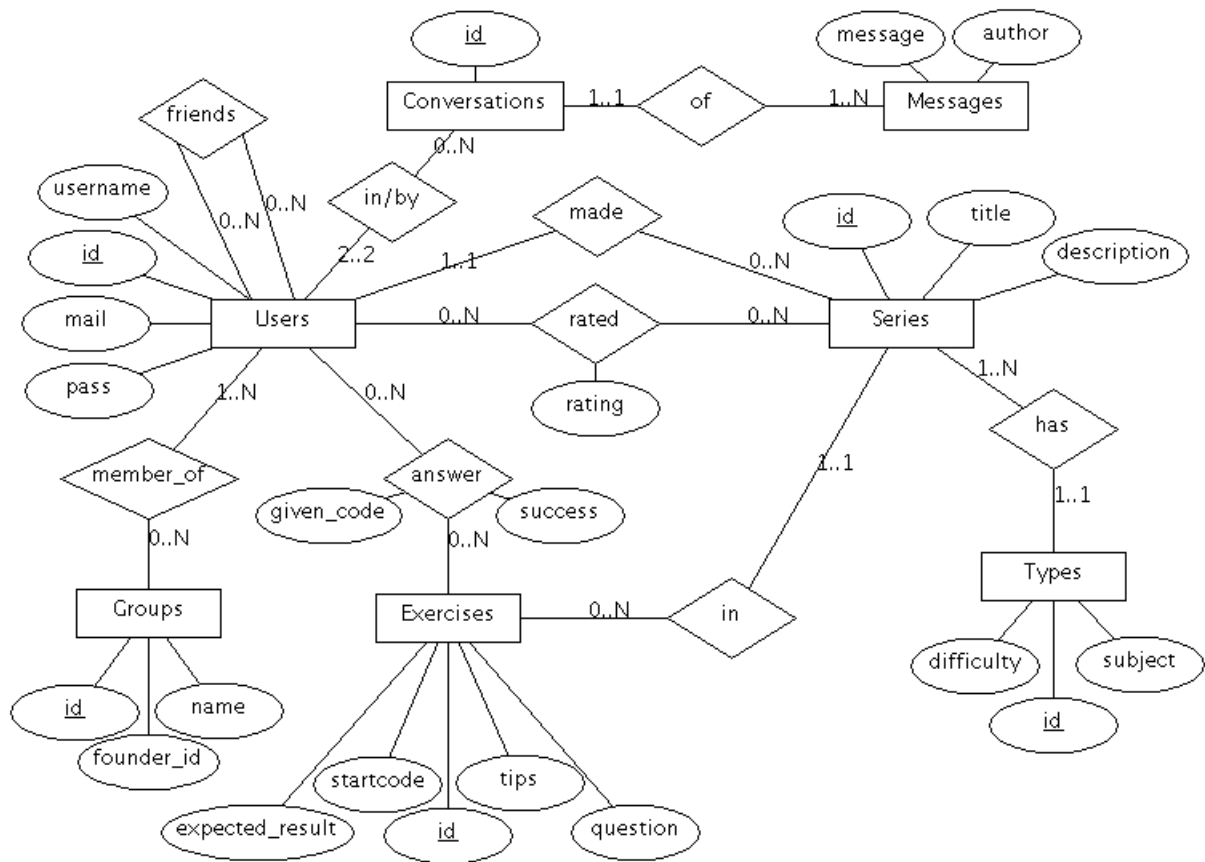
2.1.6 Sociaal

Vrienden Een gebruiker kan vrienden maken. Dit doet hij door een vriendschapsverzoek te verzenden. Dit verzoek moet dan weer door de potentiële vriend aanvaard worden. Iedere gebruiker heeft volledige controle over wie hij/zij bevriend. Nadat een vriendschapsverzoek geaccepteerd werd, kan iedere vriend dit annuleren. Een vriendschap is nu eenmaal niet definitief.

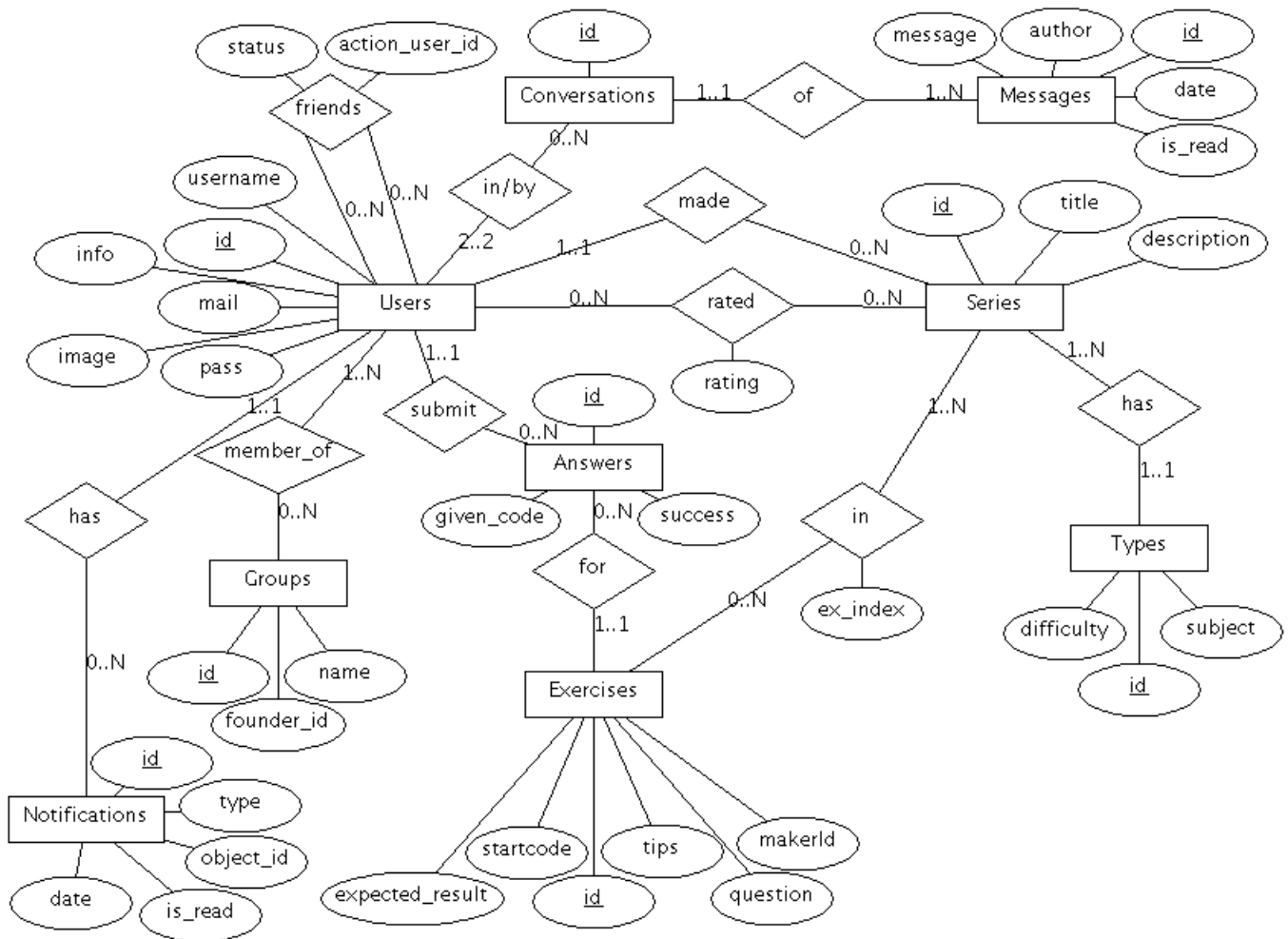
Groepen Een gebruiker kan een groep aanmaken of lid worden van een bestaande groep. Het voornaamste verschil met vrienden is dat een gebruiker geen controle heeft over wie er mede-lid wordt van een groep. Uiteraard kan iedere gebruiker, met uitzondering van de oprichter, ook de groep weer verlaten.

2.2 Database schema

2.2.1 Fase 1

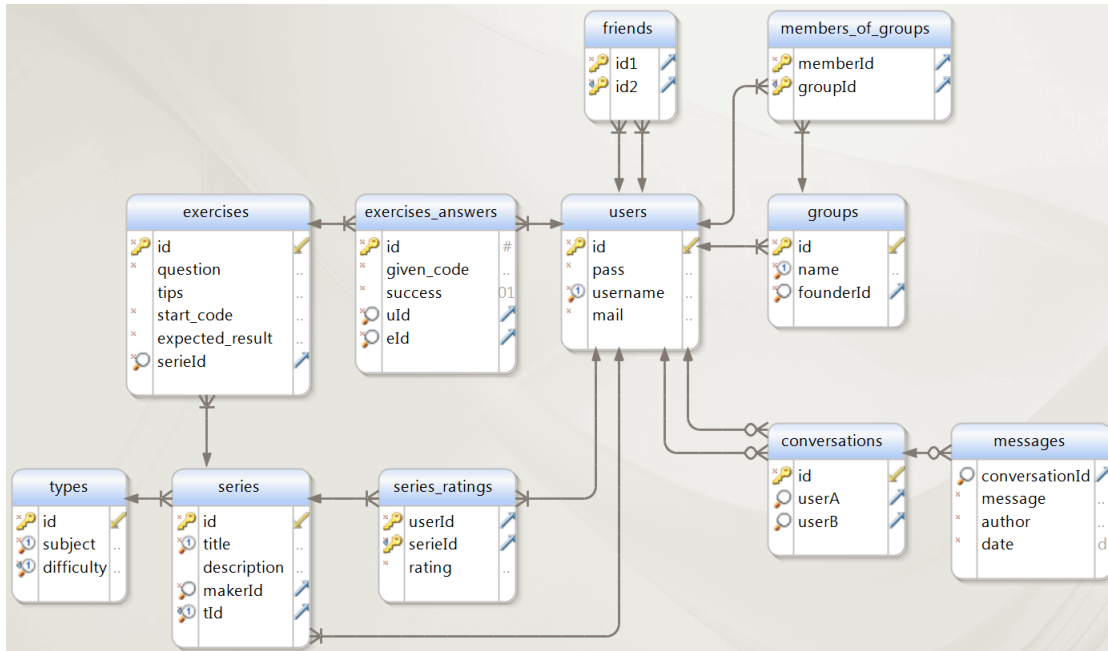


2.2.2 Fase 2

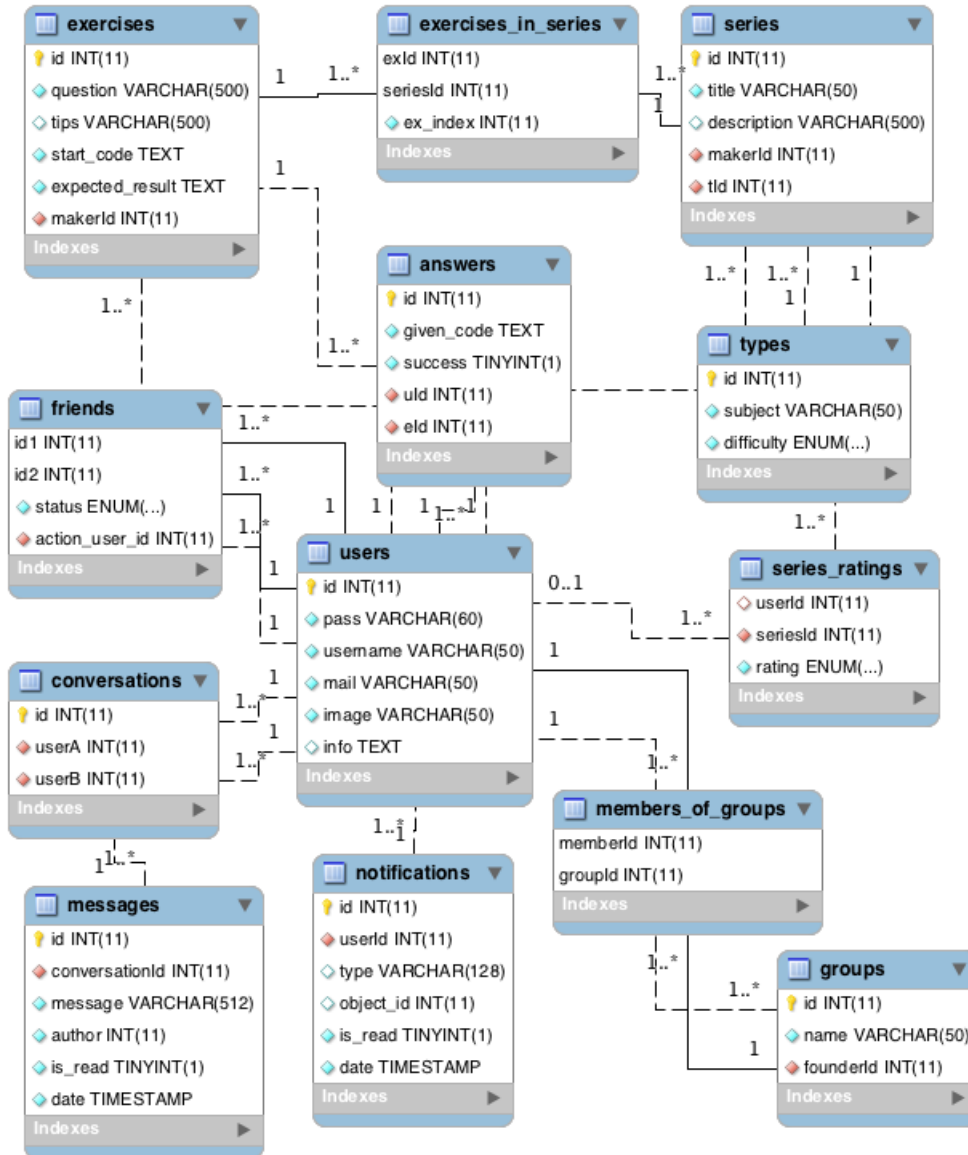


2.3 UML Schema

2.3.1 Fase 1



2.3.2 Fase 2



Hieronder wordt een korte uitleg gegeven over bovenstaande entiteiten. De uitbreiding van fase 2 t.o.v. fase 1 worden *licht cursief* weergegeven.

Users zijn de essentie van de website. Zij maken uit hoe succesvol de website kan zijn. Een User is 'een geregistreerd bezoeker'. *Users zijn uitgebreid om nu ook een afbeelding te bevatten, alsook een korte uitleg over de*

gebruiker. *Beide zijn optionele attributen en mogen dus leeg blijven.*

Friends is een 1-to-1 relatie. Om vrienden te worden moet een request verzonden worden. Indien de andere gebruiker dit aanvaard, is de Friends-relatie gecreëerd.

Groups zijn verzamelingen van Users. Een groep wordt opgericht door een gebruiker, de 'founder' van de groep. Meerdere Users kunnen dan zonder meer lid worden van de groep. De gebruikers kunnen nadien de groep verlaten (met uitzondering van de 'founder'). Dit maakt dat een groep altijd minstens 1 lid heeft.

Members of groups zijn de gebruikers die lid zijn van een groep.

Conversations zijn verzamelingen van berichten die tussen 2 gebruikers heen en weer gestuurd werden.

Messages zijn de enkelvoudige berichten die van 1 gebruiker naar de andere gestuurd wordt. Alle messages van de ene gebruiker naar de andere en eventuele antwoorden vormen samen een conversation. *Een zeer kleine uitbreiding hierop is een controle of een bericht al dan niet gelezen is. Hierdoor kan de zender zien of zijn bericht aangekomen/gelezen is.*

Series zijn het tweede essentiële deel van de applicatie. Een serie bestaat uit een set van 'Exercises'. Iedere serie krijgt ook een type en een rating.

Series rating zijn de ratings die gebruikers aan een serie kunnen geven. Deze rating zal gebruikt worden om voorstellen te doen aan andere gebruikers.

Types zijn tuples van een onderwerp en een moeilijkheidsgraad. Deze tuple vormt een unieke key van het type.

Exercises vormen samen een serie. Iedere exercise bevat een vraag, (optionele) tips voor het oplossen van de oefening, start code die de gebruiker een beginpunt geeft en een verwacht resultaat. Dit verwacht resultaat wordt vergeleken met de gegenereerde output van de interpreter. *Exercises zijn niet langer verbonden aan een enkele serie. Om dit mogelijk te maken is een extra relatie toegevoegd: exercises_in_series. Een bijkomende uitbreiding is de mogelijkheid om als verwacht antwoord*

een reguliere expressie te geven. Deze wordt dan gematched met de gegenereerde oplossing.

Exercises_in_series is een relatie tussen oefeningen en series. Hierin wordt gespecificeerd welke oefening in welke serie staat. Als bijkomend attribuut wordt de index van de oefening binnen de serie gespecificeerd. Hierdoor wordt de volgorde van oefeningen gecontroleerd. De beperking is opgesteld dat iedere oefening slechts 1x in een serie mag staan.

Answers zijn niet meer dan de aangepaste start code, samen met het resultaat van de interpreter. Zo kan een gebruiker de code later opnieuw opvragen en kan tegelijk snel opgevraagd worden of een gebruiker de oefening correct had opgelost.

Notifications zijn meldingen die een gebruiker kan krijgen. Denk hierbij aan bijvoorbeeld een ontvangen vriendschapsverzoek.

3 Product

3.1 User Interface

3.1.1 Homepage

De homepage is strak maar toch speels ontworpen. Met duidelijke, korte titels trekken we de aandacht van de bezoeker. Langere, geanimeerde teksten zorgen ervoor dat de gebruiker langer blijft kijken. Op de homepage lichten we kort het concept van de website toe. De homepage is m.a.w. volledig ontworpen om bezoekers te trekken en bij te houden. Het grafische ontwerp leent er bovendien toe om extra componenten toe te voegen. Indien er in de toekomst delen toegevoegd moeten worden, kan dit zonder het grafische concept te verstoren. Een gestreamlinede gebruikservaring primeert!

3.1.2 Navigatie

Om navigatie te vereenvoudigen is er een navigatiebalk bovenaan geplaatst om snel naar de voornaamste delen te navigeren. Eenmaal daar kunnen meer specifieke zaken opgevraagd worden op de webpage zelf. Zaken die niet van toepassing zijn op een gegeven moment worden verborgen. Een niet-ingelogde bezoeker krijgt bijvoorbeeld geen optie te zien om een oefeningenreeks te maken. Zoals doorheen heel de website is het ook voor de navigatie belangrijk dat een gebruiker niet moet zoeken. We werken daarom ook niet met onnodige, verwarrende animaties zoals dynamische dropdown menu's.

3.1.3 Vertalingen

Om op ieder gewenst ogenblik te kunnen wisselen van taal, is een dropdown-menu voorzien dat steeds zichtbaar is. Indien een gebruiker per ongeluk een taal aanduidt die hem compleet vreemd is, kan hij nog steeds op een vlag klikken om een courantere taal aan te duiden. Dit is relevant indien een gebruiker bijvoorbeeld op 'chinese' klikt ipv 'dutch'. Het spreekt voor zich dat de meeste gebruikers die 'dutch' willen aanduiden, niets kunnen met chinese tekens.

3.2 Extra functionaliteit

Turtles Aangezien "Skulpt" de mogelijkheid biedt om op een makkelijke manier "turtle graphics" te ondersteunen, zullen we deze uitbreiding alvast ter beschikking stellen.

Vertalingen Buiten de verplichte Engelse en Nederlandse taal, ondersteunen we enkele tientallen talen.

Messages Daarnaast werd ook gekozen om een "messaging" systeem te implementeren zodat gebruikers onderling met elkaar kunnen communiceren. Dit is uitgebreid met notifications voor belangrijke of interessante mededelingen.

Grafieken Een uitbreiding van de grafieken is dat deze individueel geprint of gedownload kunnen worden in allerlei formaten.

Groepsleden controle Een extra functionaliteit dat aan groepen nog toegevoegd moet worden is de mogelijkheid voor de founder van de groep om leden te weigeren/uit de groep te verwijderen.

Lessen Om beginners op weg te helpen zullen we de cursus Python uit de 1e bachelor gebruiken om echte lessen aan te bieden.

Login Deels geïmplementeerd is het aanmelden via verscheidene sociale media. Voorlopig is Facebook al ondersteund, maar dit wordt ongetwijfeld nog uitgebreid.

Reguliere expressies Het verwachte antwoord aanvaard ook reguliere expressies die gematched worden aan de gegenereerde output.

Python shell Deze optie is nog niet geïmplementeerd, maar staat wel op de lijst van toekomstige uitbreidingen.

4 Planning

Nu de basisvereisten en al enkele uitbreidingen afgewerkt zijn kunnen we ons richten op de dingen die we zelf zeer graag willen doen. De focus zal vooral liggen op het stroomlijnen van alle functionaliteit met als doel om de user experience zo vlot mogelijk te maken. Uiteraard zullen we ook extra zaken toevoegen. De voornaamste zaken staan hieronder opgelijst.

Taken	Datum	Verantwoordelijke
Python shell	03/05	Bruno
Gevorderde antwoord checks	03/05	Bruno
Social media login	03/05	Fouad

5 Appendix

In this section you find all the sql queries that are used throughout the project. If the query is used as part of a function, only the sql query is shown. This is done as not to clutter the appendix, since php code is not relevant for this section.

5.1 Tables

Below is a listing of all queries that were used to create the database. It lists the tables of the database and the specifications of each column.

```
/* In case a database is still loaded, remove it */
DROP DATABASE if exists learn2program_db;

/* Load a new database */
CREATE DATABASE learn2program_db;
USE learn2program_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT,
    pass VARCHAR(60) NOT NULL,
    username VARCHAR(50) NOT NULL UNIQUE,
    mail VARCHAR(50) NOT NULL,
    image VARCHAR(50) NOT NULL DEFAULT 'NoProfileImage.jpg',
    info TEXT,
    PRIMARY KEY (id)
);

/* id1 < id2 */
CREATE TABLE friends (
    id1 INT NOT NULL,
    id2 INT NOT NULL,
    status ENUM('pending', 'accepted', 'declined') NOT NULL,
    action_user_id INT NOT NULL,
    PRIMARY KEY (id1, id2),
    FOREIGN KEY (id1) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (id2) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (action_user_id) REFERENCES users(id) ON DELETE
    CASCADE
```

```

);

CREATE TABLE conversations (
    id INT AUTO_INCREMENT,
    userA INT NOT NULL,
    userB INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (userA) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (userB) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE messages (
    id INT AUTO_INCREMENT,
    conversationId INT NOT NULL,
    message VARCHAR(512) NOT NULL,
    author int NOT NULL,
    is_read BOOL NOT NULL DEFAULT 0,
    date TIMESTAMP, /* 'YYYY-MM-DD HH:MM:SS' format */
    PRIMARY KEY (id),
    FOREIGN KEY (conversationId) REFERENCES conversations(id) ON
        DELETE CASCADE
);

CREATE TABLE groups (
    id INT AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL UNIQUE,
    founderId INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (founderId) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE members_of_groups (
    memberId INT NOT NULL,
    groupId INT NOT NULL,
    PRIMARY KEY (memberId, groupId),
    FOREIGN KEY (memberId) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (groupId) REFERENCES groups(id) ON DELETE CASCADE
);

CREATE TABLE types (

```

```

    id INT AUTO_INCREMENT,
    subject VARCHAR(50) NOT NULL,
    difficulty ENUM('Easy', 'Intermediate', 'Hard', 'Insane') NOT
        NULL,
    PRIMARY KEY (id),
    UNIQUE (subject, difficulty)
);

CREATE TABLE series (
    id INT AUTO_INCREMENT,
    title VARCHAR(50) NOT NULL,
    description VARCHAR(500),
    makerId INT NOT NULL,
    tId INT NOT NULL,
    UNIQUE (title, tId),
    PRIMARY KEY (id),
    FOREIGN KEY (makerId) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (tId) REFERENCES types(id) ON DELETE CASCADE
);

CREATE TABLE series_ratings (
    userId INT,
    seriesId INT NOT NULL,
    rating ENUM('0', '1', '2', '3', '4', '5') NOT NULL,
    UNIQUE (userId, seriesId),
    FOREIGN KEY (userId) REFERENCES users(id) ON DELETE SET NULL,
    FOREIGN KEY (seriesId) REFERENCES series(id) ON DELETE CASCADE
);

CREATE TABLE exercises (
    id INT AUTO_INCREMENT,
    question VARCHAR(500) NOT NULL,
    tips VARCHAR(500),
    start_code TEXT NOT NULL,
    expected_result TEXT NOT NULL,
    makerId INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (makerId) REFERENCES users(id) ON DELETE CASCADE
);

```

```

CREATE TABLE exercises_in_series (
    exId INT NOT NULL,
    seriesId INT NOT NULL,
    ex_index INT NOT NULL,
    PRIMARY KEY (exId, seriesId),
    FOREIGN KEY (exId) REFERENCES exercises(id) ON DELETE CASCADE,
    FOREIGN KEY (seriesId) REFERENCES series(id) ON DELETE CASCADE
);

CREATE TABLE answers (
    id INT AUTO_INCREMENT,
    given_code TEXT NOT NULL,
    success BOOL NOT NULL,
    uId INT NOT NULL,
    eId INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (uId) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (eId) REFERENCES exercises(id) ON DELETE CASCADE
);

CREATE TABLE notifications (
    id INT AUTO_INCREMENT,
    userId INT NOT NULL,
    type VARCHAR(128),
    object_id INT,
    is_read BOOL NOT NULL DEFAULT 0,
    date TIMESTAMP,
    PRIMARY KEY (id),
    FOREIGN KEY (userId) REFERENCES users(id) ON DELETE CASCADE
);

```

5.2 Triggers

Below is a list of all the triggers used to guarantee proper use of the database. Most triggers are strictly speaking not necessary as they were also implemented in the application itself. The triggers were added nonetheless since they are part of the database design and add an extra level of protection.

```

/* Make sure password, username and mail are not empty. */

```

```

delimiter //
CREATE TRIGGER check_users
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    IF NEW.pass = "" THEN
        SET NEW.pass = Null;
    END IF;
    IF NEW.username = "" THEN
        SET NEW.username = Null;
    END IF;
    IF NEW.mail = "" THEN
        SET NEW.pass = Null;
    END IF;
END;

delimiter ;

/* Make sure the answer is not empty. */
delimiter //
CREATE TRIGGER check_answer
BEFORE INSERT ON answers
FOR EACH ROW
BEGIN
    IF NEW.given_code = "" THEN
        SET NEW.given_code = Null;
    END IF;
END;

delimiter ;

/* Make sure the title from serie is not empty. */
delimiter //
CREATE TRIGGER check_title
BEFORE INSERT ON series
FOR EACH ROW
BEGIN
    IF NEW.title = "" THEN
        SET NEW.title = Null;
    END IF;

```

```

END;

delimiter ;

/* Make sure the group name is not empty. */
delimiter //
CREATE TRIGGER check_group
BEFORE INSERT ON groups
FOR EACH ROW
BEGIN
    IF NEW.name = "" THEN
        SET NEW.name = Null;
    END IF;
END;

delimiter ;

/* Make sure the subject and difficulty are not empty. */
delimiter //
CREATE TRIGGER check_type
BEFORE INSERT ON types
FOR EACH ROW
BEGIN
    IF NEW.subject = "" THEN
        SET NEW.subject = Null;
    END IF;
    IF NEW.difficulty = "" THEN
        SET NEW.difficulty = Null;
    END IF;
END;

delimiter ;

/* Make sure question, start_code and expected_result are not
empty. */
delimiter //
CREATE TRIGGER check_exercise
BEFORE INSERT ON exercises
FOR EACH ROW
BEGIN

```

```

IF NEW.question = "" THEN
    SET NEW.question = Null;
END IF;
IF NEW.start_code = "" THEN
    SET NEW.start_code = Null;
END IF;
IF NEW.expected_result = "" THEN
    SET NEW.expected_result = Null;
END IF;
END;

```

5.3 Helpers

This section lists all queries that were used throughout the program. It contains queries such as 'inserts', 'updates', 'selects', etc. Above each query is the function name in which it was used. Obviously some functions also contain php code, but this was omitted as stated at the top of this page.

5.3.1 Exercises

```

/* function loadExercisesFromSerie($sId) */
select * from exercises, (select * from exercises_in_series where
    seriesId = ?) eps
where exercises.id = eps.exId order by ex_index,
[$sId];

/* function storeExercise($exercise) */
insert into exercises (question, tips, start_code,
    expected_result, makerId) VALUES (?, ?, ?, ?, ?),
[$exercise->question, $exercise->tips, $exercise->start_code,
    $exercise->expected_result, $exercise->makerId];

/* function storeReference($exercise) */
insert into exercises_in_series (exId, seriesId, ex_index)
select max(exercises.id), ?, (agg.count + 1)
from exercises, (select count(seriesId) as count from
    exercises_in_series where exercises_in_series.seriesId = ?)
agg, [$exercise->seriesId, $exercise->seriesId];

```

```

/* function addToSeries($exId, $seriesId) */
insert into exercises_in_series(exId, seriesId, ex_index)
select ?, ?, (agg.count + 1)
from (select count(seriesId) as count
      from exercises_in_series
      where exercises_in_series.seriesId = ?) agg,
[$exId, $seriesId, $seriesId];

/* function loadAllExercises() */
select * from exercises;

/* function loadAllAccessibleExercises($uId) */
select * from exercises
where id in (select exId as id
             from exercises_in_series
             where ex_index = 1)

group by id
union
select * from exercises where makerId = ?
union
select * from exercises
where id in
  (select exId as id
   from exercises_in_series join answers on exId = eId
   where uId = ? and success = 1)
or id in
  (select exId as id from exercises_in_series
   where (ex_index-1) in
     (select ex_index
      from exercises_in_series join answers on exId = eId
      where uId = ? and success = 1))
group by id, [$uId, $uId, $uId];

/* function loadAllFirstExercises() */
select * from exercises
where id in (select exId as id
             from exercises_in_series
             where ex_index = 1)

group by id;

```



```

/* function loadExercise($id) */
select * from exercises where id = ?, [$id];

/* function isMakerOfExercise($eId, $uId) */
select * from exercises where makerId = ? and id = ?, [$uId, $eId];

/* function nextExerciseOfSerie($eId, $sId) */
select * from exercises
where id in (select exId as id from exercises_in_series
             where (ex_index-1) in (select ex_index from
                                   exercises_in_series
                                   where exId=? and seriesId=?)
             and seriesId = ?)
group by id, [$eId, $sId, $sId];

/* function nextExerciseInLine($eId, $uId, $sId) */
select * from exercises_in_series
where seriesId = ? and ex_index not in
  (select ex_index from (exercises_in_series eis) join answers on
   eis.exId = eId
   where eis.seriesId in (select seriesId from
                         exercises_in_series where exId=?) and uId=?
   group by exId)
group by exId
order by ex_index, [$sId, $eId, $uId];

/* function firstExerciseOfSerie($eId) */
select * from exercises_in_series where exId = ? and ex_index = 1,
[$eId];

/* function userCompletedExercise($eId, $uId) */
select * from answers where eId = ? and uId = ?, [$eId, $uId];

/* function storeAnswer($ans) */
insert into answers (given_code, success, uId, eId) values (?, ?,
?, ?), [$ans->given_code, $ans->success, $ans->uId, $ans->eId];

/* function loadMyExercises() */
select *

```

```

from exercises
where makerId = ?,
[\Auth::id()];

/* function loadExercisesSearch($s) */
select *
from exercises
where question like ?,
[%. $s. %];

```

5.3.2 Friends

```

/* function loadFriend($id) */
select *
from friends
where id1 = ? and id2 = ? and status = \accepted\,
[min($id, \Auth::id()), max($id, \Auth::id())];

/* function canSendFriendRequest($id) */
select *
from friends
where id1 = ? and id2 = ? and (status = \accepted\ or status =
    \pending\) and action_user_id = ?
union
select *
from friends
where id1 = ? and id2 = ? and action_user_id = ?,
[$a, $b, \Auth::id(), $a, $b, $id];

/* function storeFriendRequest($id) */
/* insert if not exists: */
insert into friends (id1, id2, status, action_user_id)
values (?, ?, ?, ?),
[min(\auth::id(), $id), max(\auth::id(), $id), pending,
    \auth::id()];
/* update if exists: */
update friends
set status = \pending\, action_user_id = ?
where id1 = ? and id2 = ?,

```

```

[\Auth::id(), min($id, \Auth::id()), max($id, \Auth::id())];

/* function deleteFriend($id) */
update friends
set    status = \declined\, action_user_id = ?
where  id1 = ? and id2 = ?,
[\Auth::id(), min($id, \Auth::id()), max($id, \Auth::id())];

/* function isFriendRequestPending($id) */
select *
from   friends
where  id1 = ? and id2 = ? and status = \pending\ and
      action_user_id = ?,
[\min($id, \Auth::id()), max($id, \Auth::id()), $id]];

/* function isSentFriendRequestPending($id) */
select *
from   friends
where  id1 = ? and id2 = ? and status = \pending\ and
      action_user_id = ?,
[\min($id, \Auth::id()), max($id, \Auth::id()), \Auth::id()];

/* function acceptFriend($id) */
update friends
set    status = \accepted\, action_user_id = ?
where  id1 = ? and id2 = ?,
[\Auth::id(), min(\Auth::id(), $id), max(\Auth::id(), $id)];

}

/* function declineFriend($id) */
update friends
set    status = \declined\, action_user_id = ?
where  id1 = ? and id2 = ?,
[\Auth::id(), min(\Auth::id(), $id), max(\Auth::id(), $id)];

/* function loadMyFriends() */
select *
from   users
where  id <> ?

```

```

and id in (select case when id1 = ? then id2 else id1 end
          from friends
          where (id1 = ? or id2 = ?) and status = \accepted\),
[\Auth::id(), \Auth::id(), \Auth::id(), \Auth::id()];

```

5.3.3 Groups

```

/* function storeGroup($group) */
insert into groups (name, founderId) VALUES (?, ?), [$group->name,
    $group->founderId];

/* function loadAllGroups() */
select * from groups ;

/* function loadGroup($group) */
select * from groups where id = ? or name = ?, [$group, $group];

/* function isFounderOfGroup($groupId, $founderId) */
select * from groups where (id = ? or name = ?) and founderId =
    ?,[$groupId, $groupId, $founderId];

/* function addMember2Group($uId, $gId) */
insert into members_of_groups (memberid, groupid) values (?, ?),
    [$uId, $group[0]->id];

/* function deleteMemberFromGroup($uId, $gId) */
delete from members_of_groups where memberId = ? and groupId = ?,
    [$uId, $group[0]->id];

/* function noMemberYet($uId, $gId) */
DB::select(select * from members_of_groups where memberId = ? and
    groupId = ?,[$uId, $group[0]->id];

/* function updateGroup($id, $groupname) */
update groups SET name = ? where id = ?, [$groupname,
    $group[0]->id];

/* function listGroupsOfUser($id) */

```

```

select groupname from groups join (select distinct(groupId) from
    members_of_groups where memberId = ?) agg on id=groupId, [$id];

/* function listUsersOfGroup($id) */
select * from users join (select memberId from members_of_groups
    where groupId = ?) agg on id=memberId, [$id];

/* function loadAllGroupsSortedByNameASC() */
select * from groups order by name asc;

/* function loadAllGroupsSortedByNameDESC() */
select * from groups order by name desc;

/* function loadAllGroupsSortedByFounderASC() */
select * from groups join users on founderid = users.id order by
    username asc;

/* function loadAllGroupsSortedByFounderDESC() */
select * from groups join users on founderid = users.id order by
    username desc;

/* function loadMyGroups() */
select *
from groups
where id in (select distinct groupid
    from members_of_groups
    where memberid = ?),
[\Auth::id()];

/* function loadGroupsSeach($s) */
SELECT *
FROM groups
WHERE name LIKE ?,
[%. $s. %];

```

5.3.4 Messages

```

/* function storeConversation($id) */
insert into conversations (usera, userb)

```

```

value (?, ?),
[min(\Auth::id(), $userId), max(\Auth::id(), $userId)];

/* function loadConversation($id) */
select *
from   conversations
where  usera = ? and userb = ?,
[min(\Auth::id(), $id), max(\Auth::id(), $id)];

/* function loadLatestConversation() */
select c.usera, c.userb
from   conversations c
join   messages m on c.id = m.conversationid
where  c.usera = ? or c.userb = ?
order by date desc
limit  1,
[\Auth::id(), \Auth::id()];

/* function storeMessage($cId, $message) */
insert into messages (conversationid, author, message)
value (?, ?, ?),
[$cId, \Auth::id(), $message];

/* function loadAllMessagesInDB() */
select c.usera, c.userb, m.message, m.date, u.username
from   conversations c
join   messages m on c.id = m.conversationid
join   users u on m.author = u.id
order by date;

/* function loadAllMessages($id) */
select u.username, m.message, m.date
from   conversations c
join   messages m on c.id = m.conversationid
join   users u on u.id = m.author
where  c.usera = ? and c.userb = ?,
[min(\Auth::id(), $id2), max(\Auth::id(), $id2)];

/* function loadConversationsWithMessage() */

```

```

/* //Get all conversations for the logged in user, then only
   select the latest message for each of them */
select  usera, userb, message, date
from    (select c.id, c.usera, c.userb, m.message, m.date
        from    conversations c
        join    messages m on c.id = m.conversationid
        join    users u on u.id = m.author
        where   c.usera = ? or c.userb = ?
        order by date desc) as x
group by id
order by date desc,
[\Auth::id(), \Auth::id()];

/* function loadLastNConversationsWithMessage($n) */
/* //Get all conversations for the logged in user, then only
   select the latest message for each of them */
select  *
from    (select  case when c.usera = ? then c.userb else c.usera
                end as otheruser, image, message, is_read, author, date
        from    conversations c
        join    messages m on c.id = m.conversationid
        join    users u on u.id = m.author
        where   c.usera = ? or c.userb = ?
        order by m.id desc) as x
group by otheruser
limit   ?,
[\Auth::id(), \Auth::id(), \Auth::id(), $n];

/* function loadUnreadMessages() */
select  *
from    conversations c
join    messages m on c.id = m.conversationid
where   m.author <> ? and m.is_read = 0 and (c.usera = ? or c.userb
= ?),
[\Auth::id(), \Auth::id(), \Auth::id()];

//get the last message that was user $id has read
/* function loadLastReadMessage($id) */
select  *
from    conversations c

```

```

join    messages m on c.id = m.conversationid
where   m.is_read = 1 and c.usera = ? and c.userb = ? and m.author
       = ?
order by m.id desc,
[$a, $b, \Auth::id()];

/* function loadLastNMessages($n) */
select *
from    conversations c
join    messages m on c.id = m.conversationid
where   c.usera = ? or c.userb = ?
order by m.id desc
limit   ?,
[\Auth::id(), \Auth::id(), $n];

/* function UpdateMessagesToSeen($id) */
update  messages m
join    (select *
        from    conversations c
        where   c.usera = ? and c.userb = ?) cc
on      cc.id = m.conversationid
set     m.is_read = true
where   m.author <> ?,
[$a, $b, \Auth::id()];

```

5.3.5 Notifications

```

/* function loadAllNotifications() */
SELECT *
FROM notifications
WHERE  userId = ?,
[\Auth::id()];

/* function loadUnreadNotifications() */
return \db::select(select *
from notifications
where userid = ? and is_read = 0,
[\Auth::id()];

```



```

/* function loadLastNNotifications($n) */
select *
from notifications
where userid = ?
order by id desc
limit ?,
[\Auth::id(), $n];

/* function updateNotificationsToSeen() { */
update notifications
set      is_read = true
where    userid = ?,
[\Auth::id()];

/* function storeNotification($id, $type, $objectId) */
insert into notifications (userid, type, object_id)
value (?, ?, ?),
[$id, $type, $objectId];

```

5.3.6 Random

```

/* function ExNrOfSerie($eId, $sId) */
select * from exercises_in_series where exId = ? and seriesId = ?,
[$eId, $sId];

```

5.3.7 Ratings

```

/* function notRatedYet($uId, $sId) */
select * from series_ratings where userId = ? and seriesId = ?,
[$uId, $sId];

/* function addRating($nr) */
insert into series_ratings (rating, userId, seriesId) VALUES (?,
?, ?), [$nr->rating, $nr->userId, $nr->seriesId];

/* function unratedSeries($sId) */
select * from series_ratings where seriesId = ?, [$sId];

```

```

/* function averageRating($sId) */
select * from series_ratings where seriesId = ?, [$sId];

```

5.3.8 Recommendations

```

/* function returnSeriesSameMaker($serie) */
select *
from series
where series.title != ? and makerid = ?,
[$serie->title, $serie->makerId];

/* function returnSeriesSameDifficulty($serie) */
$difficulty = select *
    from types
    where types.id = ?, [$serie->tId]);

select *
from series, types
where series.title != ?
and series.tid = types.id
and types.difficulty = ?,
[$serie->title, $difficulty[0]->difficulty];

/* function returnSeriesSameRating($serie) { */
/* $rating = */
select *
from series, series_ratings
where series.id = series_ratings.seriesid and series.id = ?,
[$serie->id];
/* if (!empty($rating)): */
select *
from series, series_ratings
where series.id = series_ratings.seriesid
and series.id != ?
and series_ratings.rating = ? ,
[$serie->id, $rating[0]->rating];
/* return: */
select *

```

```

from series
where series.id != series.id;

/* function isEmptySeries($serie) */
select *
from series, exercises, exercises_in_series
where series.id = exercises_in_series.seriesid
and exercises.id = exercises_in_series.exid
and series.id = ?, [$serie->id];

```

5.3.9 Series

```

/* function storeSerie($serie) */
insert into series (title, description, makerId, tId)
values (?, ?, ?, ?),
[$serie->title, $serie->description, $serie->makerId, $serie->tId];

/* function loadSerieWithId($id) */
select *
from series
where id = ?,
[$id];

/* function loadSerieWithIdOrTitle($id) */
select *
from series
where id = ?
or title = ?,
[$id, $id];

/* function loadSerieWithIdOrTitleAndExercise($sId, $eId) */
select *
from series
where (id = ? or title = ?)
and id in (select seriesId as id
          from exercises_in_series
          where exId = ?),
[$sId, $sId, $eId];

```

```

/* function loadSerie($title, $tId) */
select *
from series
where title = ?
and tId = ?,
[$title, $tId];

/* function loadAllSeries() */
select *
from series;

/* function loadAllDistinctSeries() */
select *
from series
group by title;

/* function updateSerie($id, $serie, $typeId) */
update series
set title = ?, description = ?, tid = ?
where id = ?,
[$serie->title, $serie->description, $typeId, $id];

/* function isMakerOfSeries($sId, $mId) */
select *
from series
where (id = ? or title = ?)
and makerId = ?,
[$sId, $sId, $mId];

/* function SerieContainsExercises($sId) */
select *
from exercises, (select *
    from exercises_in_series
    where seriesId = ?) eps
where exercises.id = eps.exId,
[$sId];

```

```

/* function loadSeriesSortedByNameASC() */
select *
from series
group by title
order by title ASC;

/* function loadSeriesSortedByDiffASC() */
select *
from series
join types on tId = types.id
order by difficulty ASC;

/* function loadSeriesSortedBySubASC() */
select *
from series
join types on tId = types.id
order by subject ASC;

/* function loadSeriesSortedByNameDESC() */
select *
from series
group by title
order by title DESC;

/* function loadSeriesSortedByRatingDESC() */
/* $avgs = averageRatingsBySeries(); */
select *
from series
where id = ?,
$avg[0];

/* function loadSeriesSortedByDiffDESC() */
select *
from series
join types on tId = types.id
order by difficulty DESC;

```

```

/* function loadSeriesSortedBySubDESC() */
select *
from series
join types on tId = types.id
order by subject DESC;

/* function loadSeriesWithExercise($eId) */
select *
from series
join exercises_in_series on id = seriesId
where exId = ?,
[$eId];

/* function loadMySeries() */
select *
from series
where makerid = ?,
[\Auth::id()];

/* function loadSeriesSearch($s) */
select *
from series
where title like ?
or description like ?,
[%. $s.%, %. $s.%];

```

5.3.10 Statistics

```

/* function averageRatingByUser($id) */
select * from series_ratings where userId = ?, [$id];

/* function averageRatingsByTypes() */
select types.id, avg(rating) as a
from series_ratings
join series
join types
where series.id = seriesid

```

```

and tid = types.id
group by tid;

/* function countSeriesByMakers() */
select makerId, count(id) as c
from series
group by makerId;

/* function countExercisesByMakers() */
select makerId, count(exercises.id) as c
from exercises
group by makerId;

/* function countSeriesCompletedByUsers() */
select uId, count(uId) as c
from (select uId, count(distinct(eId)) as c1, agg.seriesId, agg.c2
from answers
join exercises
join (select seriesId, count(id) as c2
      from exercises
      group by seriesId) agg
on eId = exercises.id
and exercises.seriesId = agg.seriesId
group by uId, agg.seriesId having c1 = c2) agg
group by uId;

/* function countSeriesSucceededByUsers() */
select uId, count(uId) as c
from (select uId, count(distinct(eId)) as c1, agg.seriesId, agg.c2
from answers
join exercises
join (select seriesId, count(id) as c2
      from exercises
      group by seriesId) agg
on eId = exercises.id
and exercises.seriesId = agg.seriesId
where success = 1
group by uId, agg.seriesId having c1 = c2) agg
group by uId;

```

```

/* function countSeriesInProgressByUsers() */
select uId, (agg2.c-count(uId)) as c
from (select uId, count(distinct(eId)) as c1, agg.seriesId, agg.c2
      from answers
      join exercises
      join (select seriesId, count(id) as c2
            from exercises group by seriesId) agg
      on eId = exercises.id
      and exercises.seriesId = agg.seriesId
      group by uId, agg.seriesId having c1 = c2) agg,
      (select count(distinct(seriesId)) as c
       from exercises) agg2
group by uId;

/* function countSeriesUnstartedByUsers() */
select *
from (select users.id, agg.c as c
      from users, (select count(id) as c
                   from series
                   where id in (select distinct(seriesId)
                                from exercises)) agg
      where users.id not in (select uId from answers)
      union (select uId as id, (agg.c-count(distinct(seriesId)))
             from (answers join exercises on eId = exercises.id),
                  (select count(id) as c
                   from series
                   where id in (select distinct(seriesId)
                                from exercises)) agg
             group by uId)) agg
group by id;

/* function countExercisesCompletedByUsers() */
select uId,
count(distinct(eId)) as c
from answers
group by uId;

/* function countExercisesSucceededByUsers() */
select uId,
count(distinct(eId)) as c

```



```

from answers
where success = 1
group by uId;

/* function countExercisesFailedByUsers() */
select uId, count(distinct(eId)) as c
from answers
where success = 0
group by uId;

/* function countExercisesUnstartedByUsers() */
select *
from (select users.id, agg.c as c
      from users, (select count(id) as c
                   from exercises) agg
      where users.id not in (select uId
                             from answers)
      union (select uId as id, (agg.c-count(distinct(eId)))
             from answers, (select count(id) as c
                             from exercises) agg
             group by uId)) agg
group by id;

/* function countTypesCompletedByUsers() */
select uId, count(agg1.tId) as c
from (select agg.uId, count(agg.eId) as c, series.tId
      from (select uId, eId
            from answers group by uId, eId) agg
      join exercises
      join series on agg.eId = exercises.id
                  and seriesId = series.id
      group by agg.uId, tId) agg1
join (select tId, count(exercises.id) as c
      from exercises
      join series on seriesId = series.id
      group by tId) agg2
on agg1.tId = agg2.tId
where agg1.c = agg2.c
group by uId;

```

```

/* function countTypesSucceededByUsers() */
select uId, count(agg1.tId) as c
from (select agg.uId, count(agg.eId) as c, series.tId
      from (select uId, eId
            from answers
            where success = 1
            group by uId, eId) agg
      join exercises
      join series on agg.eId = exercises.id
      and seriesId = series.id
      group by agg.uId, tId) agg1
join (select tId, count(exercises.id) as c
      from exercises
      join series on seriesId = series.id group by tId) agg2
on agg1.tId = agg2.tId
where agg1.c = agg2.c
group by uId;

/* function countTypesInProgressByUsers() */
select uId, (agg2.c-agg1.c) as c
from (select agg.uId, count(agg.eId) as c, series.tId
      from (select uId, eId
            from answers group by uId, eId) agg
      join exercises
      join series on agg.eId = exercises.id
      and seriesId = series.id group by agg.uId, tId) agg1
join (select tId, count(exercises.id) as c
      from exercises join series on seriesId = series.id
      group by tId) agg2
on agg1.tId = agg2.tId
group by uId;

/* function countTypesUnstartedByUsers() */
select *
from (select users.id, agg.c as c
      from users, (select count(id) as c
                  from types
                  where id in (select tId from series
                              where id in (select distinct(seriesId)
                                          from exercises))) agg

```

```

where users.id not in (select uId from answers)
union (select uId as id, (agg.c-count(distinct(tId)))
      from (answers join exercises join series join types on eId
            = exercises.id and seriesId = series.id and tId =
            types.id), (select count(id) as c
            from types
            where id in (select tId
                          from series
                          where id in (select distinct(seriesId)
                                        from exercises))) agg
      )) agg
group by id;

/* function countSeriesByTypes() */
select tId, count(id) as c
from series
group by tId;

/* function countUsersByGroups() */
select groupId, count(memberId) as c
from members_of_groups
group by groupId;

/* function countGroupsByUsers() */
select memberId, count(groupId) as c
from members_of_groups
group by memberId;

/* function countExercisesBySeries() */
select * from
( (select id as seriesId, 0 as c
  from series
  where id not in
    (select seriesId
     from exercises_in_series
     group by seriesId) )
  union
  (select seriesId, count(exId) as c
   from exercises_in_series

```

```

        group by seriesId) )
agg
group by seriesId;

/* function countUsersSucceededSeries() */
select * from
    (select seriesId, count(distinct(uId)) as c from
        (select count(exId) as c, seriesId, uId
            from exercises_in_series join answers on exId = eId
            where success = 1
            group by seriesId, uId) agg1
        join
        (select count(exId) as exCount, seriesId as sId
            from exercises_in_series
            group by sId) agg2
        on (seriesId = sId and c = exCount)
        group by seriesId
    union
        select seriesId, 0 as c from exercises_in_series
        where exId not in
        (select eId as exId
            from answers
            where success = 1
            group by seriesId)
    union
        select id as seriesId, 0 as c from series
        where id not in (select seriesId as id
            from exercises_in_series
            group by seriesId)) agg
group by seriesId;

/* function countExercisesInSeries($seriesId) */
select count(distinct(exId)) as c
from exercises_in_series eis
where eis.seriesId = ?,
[$seriesId];

/* function countUserSucceededExercisesBySeries($uId) */
select seriesId, c from
    (select seriesId, count(distinct(exId)) as c

```

```

        from exercises_in_series join answers on eId=exId
        where success = 1 and uId = ?
        group by seriesId
union
    select seriesId, 0 as c from exercises_in_series
    where exId not in
        (select eId as exId from answers
         where success=1 and eId=exId and uId=?)
union
    select id as seriesId, 0 as c from series
    where id not in (select seriesId as id from
                     exercises_in_series)
    ) agg
group by seriesId,
[$uId, $uId];

```

5.3.11 Types

```

/* function removeUnusedTypes() */
delete from types
where id NOT IN (select distinct(tId)
                 from series);

/* function loadTypeId($type) */
select id
from types
where subject = ?
and difficulty = ?,
[$type->subject, $type->difficulty];

/* function loadType1($type) */
select *
from types
where subject = ?
and difficulty = ?,
[$type->subject, $type->difficulty];

/* function loadType2($id) */
select *

```

```

from types
where id = ?,
[$id];

/* function loadAllTypes() */
select *
from types;

/* function storeType($type) */
insert into types (subject, difficulty)
values (?, ?),
[$type->subject, $type->difficulty];

/* function loadDifficultyAsInt($id) */
select difficulty
from types
where id = ?,
[$id];

```

5.3.12 Users

```

/* function loadusers() */
select *
from users;

/* function loadUser($name) */
select *
from users
where username = ?
or id = ?,
[$name, $name];

/* function loadName($id) */
select username
from users
where id = ?,
[$id];

/* function loadId($name) */

```

```

select id
from users
where username = ?,
[$name];

/* function storeUser($user) */
insert into users (pass, username, mail, info)
VALUES (?, ?, ?, ?),
[$user->pass, $user->username, $user->mail, $user->info];

/* function updateUser($id, $data) */
update users
set username = ?, mail = ?, pass = ?, image = ? , info = ?
where id = ?
or username = ?,
[$data->username, $data->mail, $data->pass, $data->image,
    $data->info, $id, $id];

/* function loadUsersSearch($s) */
select *
from users
where username like ?,
[%. $s. %];

```
