< 16/03/2017>

ReviUL

Deliverable 2

Week 7/8

Group 10

Orla Bonar - 14031833

Kate Hennessy - 11108517

Mary Annie Vijula Ashok Kumar - 16136861

Contents

1. Introduction

This report details the database schema as created by Group 10 for the CS4065 Web Infrastructures project. The document consists of a single section, containing;

- Database Table Synopsis – a list of the attributes and datatypes for each table in the database. In some instances, this list will be followed by a brief note explaining the reasoning and logic behind the grouping of attributes in a particular table.
- Table SQL Creation Declarations – this will consist of the SQL statements used to create each table on the group10 MySQL database used for the project. This includes the use of foreign key constraints, auto-incrementation, default values as well as setting unique attributes.
- Database Security Measures – Secure Salted Password Hashing.

- Link to Git Repository - https://github.com/KateHennessy/CS4014_TermProject.git

2. Database Tables

This section will contain database tables, their attributes and datatypes. It will also contain SQL create table statements in order to display foreign keys, auto-increments, default values and unique attributes.

Symbols:
🔑: Primary Key
🗝️: Foreign Key

User

🔑user_id	f_name	l_name	email	pass
INT (11)	VARCHAR (100)	VARCHAR (100)	VARCHAR (128)	CHAR (64)

🗝️discipline_id	reputation	signup_date
INT (11)	INT (11)	DATETIME

NOTE: Reputation score will determine if a user if a moderator, or a general user. Banned users are distinguished through a linked 'Banned_User' table. Emails will be flagged as UNIQUE in the table which ensures they cannot be entered twice. Discipline_id is a foreign key linking with the Discipline table.

```
CREATE TABLE IF NOT EXISTS  `user`(
      `user_id` INT(11) unsigned NOT NULL AUTO_INCREMENT,
      `f_name` VARCHAR(100) NOT NULL,
      `l_name` VARCHAR(100) NOT NULL,
      `email` VARCHAR(128) NOT NULL,
      `pass` CHAR(64) NOT NULL,
      `discipline_id` INT(11) unsigned NOT NULL,
      `reputation` INT(11) NOT NULL,
      `signup_date` DATETIME NOT NULL,
      PRIMARY KEY(`user_id`),
      FOREIGN KEY(`discipline_id`) REFERENCES discipline(`discipline_id`)
          ON DELETE CASCADE ON UPDATE CASCADE,
      UNIQUE(`email`)
);
```

Discipline

🔑 discipline_id	discipline_name
INT(11)	VARCHAR(128)

NOTE: Discipline_id corresponds to that of the same name in the user table. This table will store the names of the different disciplines available.

```
CREATE TABLE IF NOT EXISTS `discipline`(
      `discipline_id` INT(11) unsigned NOT NULL AUTO_INCREMENT,
      `discipline_name` VARCHAR(128) NOT NULL,
      PRIMARY KEY(`discipline_id`),
      UNIQUE(`discipline_name`)
);
```

Banned_User

🔑🔑user_id	timestamp
INT(11)	TIMEDATE

NOTE: This tables contains information on users who have been banned and time and date on which they were banned.

```
CREATE TABLE IF NOT EXISTS `banned_user`(
      `user_id` INT(11) unsigned NOT NULL,
      `timestamp` DATETIME,
      PRIMARY KEY(`user_id`),
      FOREIGN KEY(`user_id`) REFERENCES user(`user_id`)
          ON DELETE CASCADE ON UPDATE CASCADE
);
```

Task

task_id	creator_id	task_title	task_type	description
BIGINT (20)	INT (11)	VARCHAR(128)	VARCHAR (128)	VARCHAR (200)

claim_deadline	completion_deadline	no_pages	no_words	format
DATETIME	DATETIME	INT (11)	INT(11)	VARCHAR (5)

storage_address
VARCHAR(200)

NOTE: Task information will be stored in this table which will allow for query efficiency. Flagged tasks are distinguished through a linked table named Flagged_Tasks.

```
CREATE TABLE IF NOT EXISTS `task`(
      `task_id` BIGINT(20) unsigned NOT NULL AUTO_INCREMENT,
      `creator_id` INT(11) unsigned NOT NULL,
      `task_title` VARCHAR(128) NOT NULL,
      `task_type` VARCHAR(128) NOT NULL,
      `description` VARCHAR(200) NOT NULL,
      `claim_deadline` DATETIME NOT NULL,
      `completion_deadline` DATETIME NOT NULL,
      `no_pages` INT(11) NOT NULL,
      `no_words` INT(11) NOT NULL,
      `format` VARCHAR(5) NOT NULL,
      `storage_address` VARCHAR(200) NOT NULL,
      PRIMARY KEY(`task_id`),
      FOREIGN KEY (`creator_id`) REFERENCES user(`user_id`)
          ON DELETE CASCADE ON UPDATE CASCADE
);
```

Claimed_Task

task_id	claimer_id	score
BIGINT(20)	INT(11)	INT(11)

NOTE: A claimed task can only be claimed one user. A score will also be stored and will be 0 until the claimant has completed the task. At this point the creator will be able to give the claimant a review of “happy” or “unhappy” which corresponds to score changing to 5 or -5 respectively.

```
CREATE TABLE IF NOT EXISTS `claimed_task`(
      `claimer_id` INT(11) unsigned NOT NULL,
      `task_id` BIGINT(20) unsigned NOT NULL,
      `score` INT(11) unsigned NOT NULL DEFAULT 0,
      PRIMARY KEY(`task_id`),
      FOREIGN KEY (`claimer_id`) REFERENCES user (`user_id`) ON DELETE
      CASCADE  ON UPDATE CASCADE,
      FOREIGN KEY(`task_id`) REFERENCES task(`task_id`)  ON DELETE CASCADE
      ON UPDATE CASCADE
);
```

Flagged Task

🔑🔑task_id	🔑flagger_id	timestamp
BIGINT(20)	INT(11)	DATETIME

NOTE: This table contains information on tasks which have been flagged. It contains the user id of the person who has flagged the task as well as a timestamp from when it was reported.

```
CREATE TABLE IF NOT EXISTS `flagged_task`(
      `task_id` BIGINT(20) unsigned NOT NULL,
      `flagger_id` INT(11) unsigned NOT NULL,
      `timestamp` DATETIME NOT NULL,
      PRIMARY KEY(`task_id`),
      FOREIGN KEY(`task_id`) REFERENCES task(`task_id`)
          ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY(`flagger_id`) REFERENCES user(`user_id`)
          ON DELETE CASCADE ON UPDATE CASCADE
);
```

Status

🔑status_id	status_name
BIGINT(20)	VARCHAR(40)

NOTE: This table contains information on the name of the status and it's corresponding is which would then be used in the task_status table. It allows for easy changing of a status name in future without needing to update a large quantity of data in tables.

```
CREATE TABLE IF NOT EXISTS `status`(
      `status_id` INT(11) unsigned NOT NULL AUTO_INCREMENT,
      `status_name` VARCHAR(40) NOT NULL,
      PRIMARY KEY(`status_id`),
      UNIQUE(`status_name`)
);
```

Task_Status

🔑🔑task_id	🔑🔑status_id	🔑timestamp
BIGINT(20)	BIGINT(20)	DATETIME

NOTE: This table contains information on the status of a task and the time and date in which it was updated. It would allow for a history of a task's statuses (which could be introduced in a future version of the website) as well as access to the most recent status update of a task.

```
CREATE TABLE IF NOT EXISTS `task_status`(
      `task_id` BIGINT(20) unsigned NOT NULL,
      `status_id` INT(11) unsigned NOT NULL,
      `timestamp` DATETIME,
      PRIMARY KEY(`task_id`,`status_id`,`timestamp`),
      FOREIGN KEY(`task_id`) REFERENCES task(`task_id`)
          ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY(`status_id`) REFERENCES status(`status_id`)
          ON DELETE CASCADE ON UPDATE CASCADE
);
```

Tag

🔑tag_id	tag_name
INT (11)	VARCHAR(128)

NOTE: Tag name holds the name of the tag that would be available to the user. This tag name is then associated with a tag_id. Tag names cannot be entered twice into this table. Tags will later be linked with users and tasks through User_Tag and Task_Tag tables respectively.

```
CREATE TABLE IF NOT EXISTS `tag`(
      `tag_id` INT(11) unsigned NOT NULL AUTO_INCREMENT,
      `tag_name` VARCHAR(128) NOT NULL,
      PRIMARY KEY(`tag_id`),
      UNIQUE(`tag_name`)
);
```

User_Tag

🔑🔑user_id	🔑🔑tag_id	clicks
INT (11)	INT(11)	INT(11)

NOTE: Both user_id and tag_id are primary and foreign keys in this table. Clicks relates to the number of times a user has "clicked" on a certain tag (which would allow for future implementations of the site to order available tasks based on browsing habits). This allows for greater efficiency when showing available tasks with those associated tags.

```
CREATE TABLE IF NOT EXISTS `user_tag`(
      `user_id` INT(11) unsigned NOT NULL,
      `tag_id` INT(11) unsigned NOT NULL,
      `clicks` INT(11) unsigned NOT NULL DEFAULT 0,
      PRIMARY KEY(`user_id`, `tag_id`),
      FOREIGN KEY(`user_id`) REFERENCES user(`user_id`)
         ON DELETE CASCADE  ON UPDATE CASCADE,
      FOREIGN KEY(`tag_id`) REFERENCES tag(`tag_id`)
         ON DELETE CASCADE ON UPDATE CASCADE
);
```

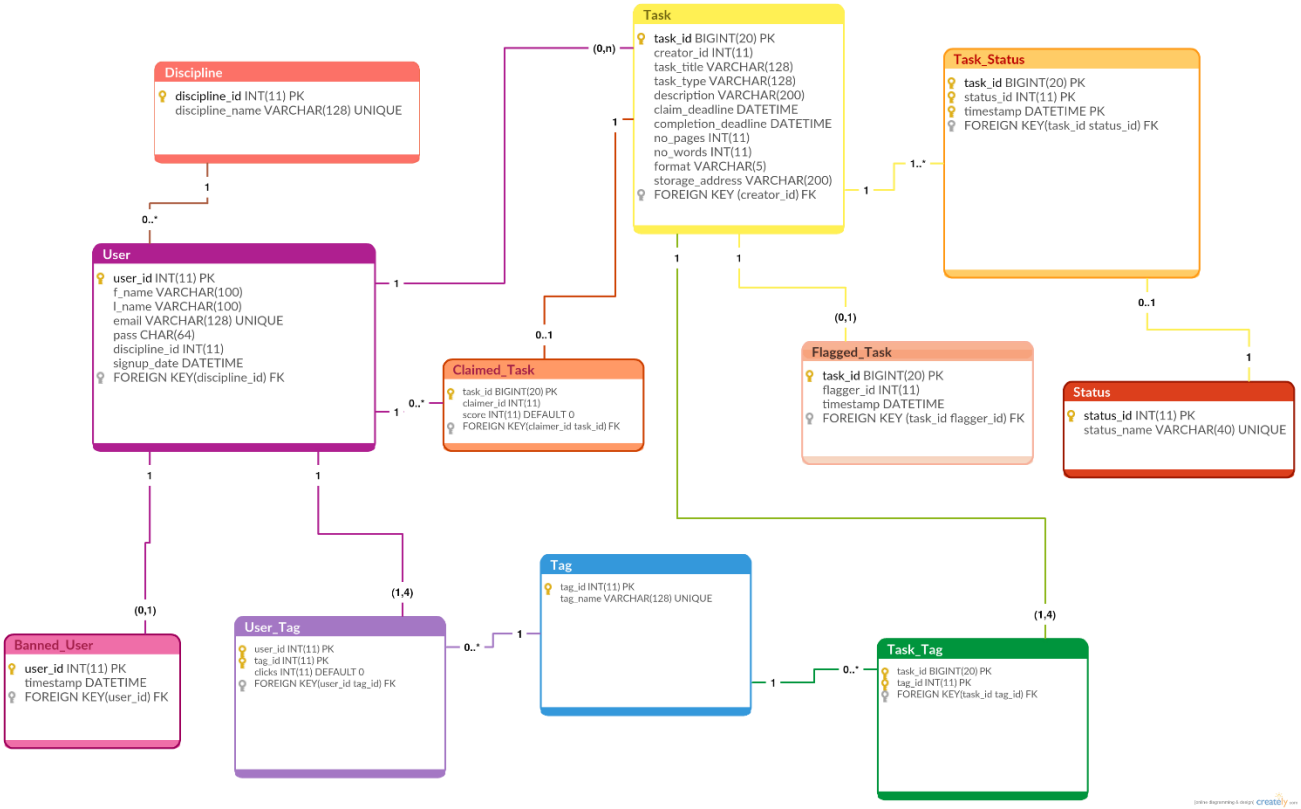
Task_Tag

🔑🔑task_id	🔑🔑tag_id
BIGINT (20)	INT(11)

NOTE: Both task_id and tag_id are foreign keys and primary keys. This associates the tag_id from the tag table with the task_id, mapping it to the task table.

```
CREATE TABLE IF NOT EXISTS `task_tag`(
      `task_id` BIGINT(20) unsigned NOT NULL,
      `tag_id` INT(11) unsigned NOT NULL,
      PRIMARY KEY(`task_id`, `tag_id`),
      FOREIGN KEY(`task_id`) REFERENCES task(`task_id`)
         ON DELETE CASCADE  ON UPDATE CASCADE,
      FOREIGN KEY(`tag_id`) REFERENCES tag(`tag_id`)
          ON DELETE CASCADE ON UPDATE CASCADE
);
```

3. Database Tables Diagram



***created using creately

4. Database Security Measures

Secure Salted Password Hashing:

We are using a randomly generated salt and SHA256 hashing algorithm on inputted passwords entering the database.

"Salting" is a security practice of adding random data (a "salt") to a password before hashing it and storing the hashed value in a database. The salt is stored in plaintext. Salts are used to add randomness in the actual data. If the salt is not appended in the plain text password every time when the hash of the password is generated, you will get the same hash value (provided same hashing algorithm has been used). Normally people use identical passwords in multiple websites for the ease of remembering it. If password of one website is compromised, most likely it will be tried in another website by attacker/hacker. If the salt is used every website will have different hash value in their database so attacker won't succeed.