Eric Tsai (ekt7be), David Tominaga (dyt2yv)

Section 002

PROJECT REPORT

Introduction

We are developing a system that models a self coaching guide for an online game. The game DotA2 (Defense of the Ancients 2) is a MOBA (multiplayer online battle arena) where teams fight against each other to destroy the base of the opposing team. Each player themselves chooses a hero that they are in control of for the duration of the game. Heroes come in all shapes and sizes and each has their own strengths and weaknesses. What we want to do is to develop an application that will display all of this information to the user as a sort of library and to allow a user to define their own personal list of preferred heroes and create what is essentially a sort of cheat sheet as a guide for the game that will show the user the best items to take up on a certain hero. We also want to allow the user to define any list they want. A database is a device used to store large quantities of information. However the data itself is unimportant unless it has some meaning to the user.

The significance of the data can often be derived from the relationship between different parts of the data. However there are certain cases where what is important to a certain user will not be the same as another user. One of our goals with the project is to develop a system where the user's preferences are taken into account as we form a personalized application platform that will allow the particular user to see the information that is most important to them by defining particular relationships within the data to fit their needs. Naturally this means that we will be implementing a user interface within the application when accessing information from the database and also requires that we store information pertaining to each user. This will allow us to specify access to a user's personal information and to enforce authorization protocols. This information includes the user's personalized lists of heroes and general or specific item builds for characters. Furthermore the application will host information related to many

aspects of the game in general including the background knowledge behind different heroes with their individual stats and their abilities and the different in game items and their effects.

Requirements Document

- The user will be able to login to their own account using a personal username and password
- Users can create an account with a username and password
- User's passwords will be encrypted using salt and hash
- When using the website the a general user will have only certain access to some relevant tables in the database (not a super user)
- Application will have an admin user that updates the information if it becomes necessary
- User will be able to export data in JSON format by opening a new page
- User will be able to view lists of all heroes in the game and display the related information of their stats, skills, etc for each
- Users will be able to select their favorite heroes and generate different personalized lists
- User can remove heroes for either their lists
- The application will allow the user to see all their lists
- Users can remove heroes from lists
- Users can rename their lists
- The user can specify certain items to go into builds
- The user can delete builds
- Users can remove items from builds
- User can specify a build is for a certain hero that a build is designed to go with
- Application will have a list of items, their costs, benefits, and effects
- Application will show the items required to make other items and also the items that make up that item as well

- Users will not be able to delete hero or item data or otherwise manipulate this data
- Users will also be unable to alter other user's lists and builds in any way
- Items will indicate what items are necessary to construct it and which recipe is required

Part II - The Design Process

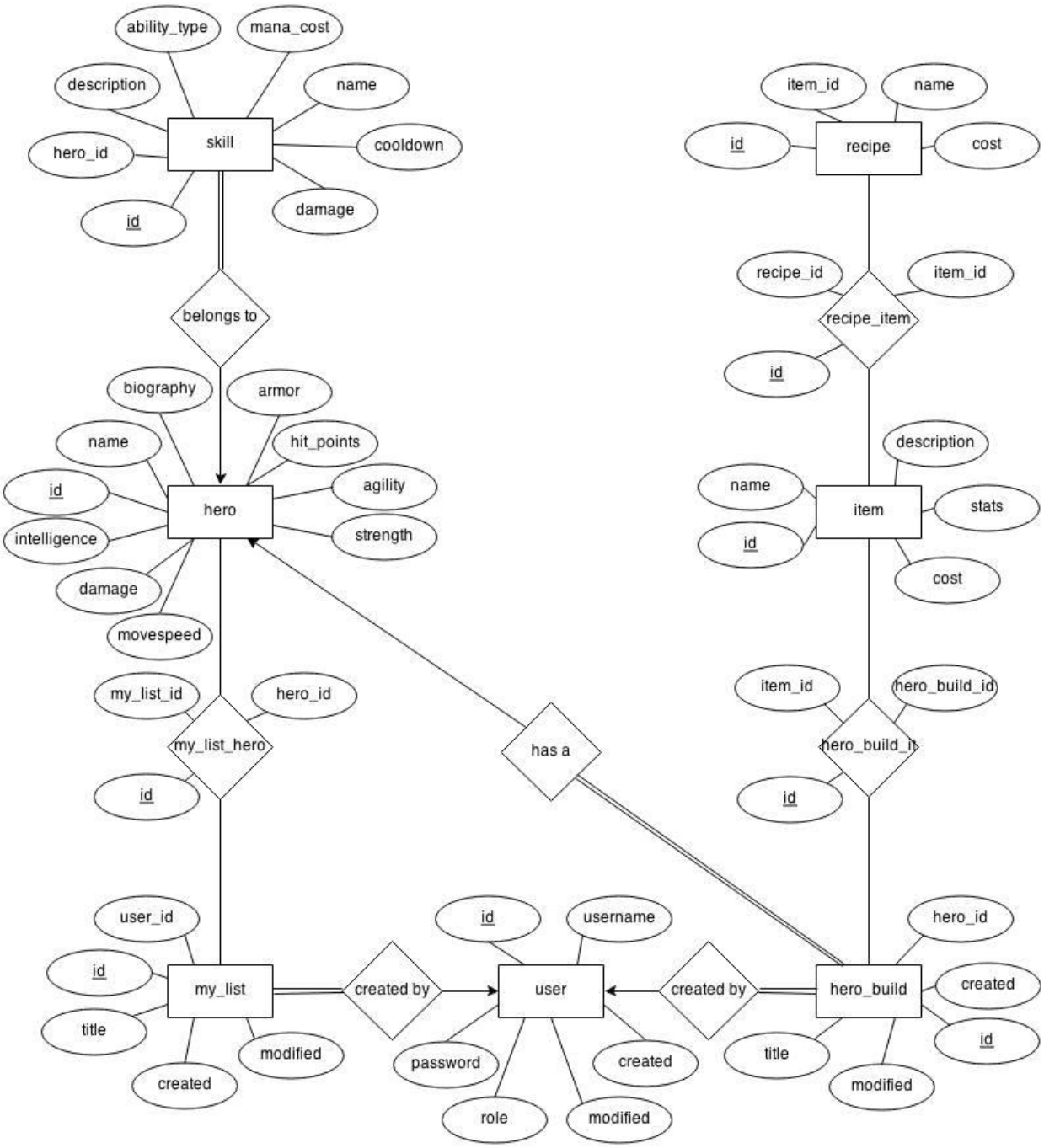
We used PHP because PHP interfaces well with MySQL. We also used the CakePHP framework because CakePHP provides a lot of support for PHP to MySQL integration.

Our database is secure in several ways. Both in application and database level, we have varying levels of access for normal users or admin users. Admin users can only be created through PHPMyAdmin or directly interfacing with the database. We salted and hashed passwords for every user for additional security and we also prevent normal users from editing or deleting Recipes, Items, Heroes, and Skills because those are items determined by the game DOTA 2. We also connect to the database with cs4750ekt7bea which is a subaccount with only INSERT, UPDATE, DELETE, and SELECT rights.

In our database, we used two special mysql commands, check and trigger. We used check to ensure that our Hero was data not below the minimum values. Similarly, we also checked skill cooldowns to make sure they weren't below zero. MySQL Triggers were placed to check for additional constraints such as maintaining statistics are above 0 such as agility, intelligence and strength. They also serve to provide error messages as MySQL doesn't provide error messages with check constraints. Constraints and triggers are shown in red within the database schema.

In proving our database, we proved using BCNF because 3NF doesn't deal well with relations that have multiple candidate keys where each candidate keys have overlap. There were several cases of that within our database schema so we proved with BCNF for better clarity.

ER Diagram



Database Schema

```
//uses sha-1, hence char(40) for password
CREATE TABLE users
(
id int NOT NULL AUTO_INCREMENT,
username varchar(255) NOT NULL UNIQUE,
password char(40) NOT NULL,
```

```
role varchar(20) DEFAULT NULL,
created DATETIME,
modified DATETIME,
PRIMARY KEY (id)
);

//none of the int fields should be greater than 8 units long
//no one has negative hp, and no one is slower than 150 units
CREATE TABLE heroes
(
id int NOT NULL AUTO_INCREMENT,
name varchar(255) NOT NULL UNIQUE,
biography text NOT NULL,
hit_points INT(8) NOT NULL,
damage INT(8),
armor INT(8) NOT NULL,
movespeed INT(8) NOT NULL,
strength INT(8) NOT NULL,
agility INT(8) NOT NULL,
intelligence INT(8) NOT NULL,
PRIMARY KEY (id),
CONSTRAINT chk_Hero CHECK (hit_points>0 AND movespeed>150)
);

DELIMITER //
CREATE TRIGGER `chk_attribute_trig` BEFORE INSERT ON `heroes`
FOR EACH ROW
BEGIN
    IF NEW.strength < 0 OR NEW.agility < 0 OR NEW.intelligence < 0 THEN
        SIGNAL SQLSTATE '45000'
             SET MESSAGE_TEXT = 'An error occurred. Attributes must be
greater than 0.';
    END IF;
END;//
DELIMITER ;


//cooldowns can't be negative
CREATE TABLE skills
(
id int NOT NULL AUTO_INCREMENT,
hero_id int NOT NULL,
name varchar(60),
description text NOT NULL,
ability_type varchar(10) NOT NULL,
damage int(8),
mana_cost int(8),
cooldown INT(8),
PRIMARY KEY (id),
FOREIGN KEY (hero_id) REFERENCES heroes(id) ON DELETE CASCADE,
CONSTRAINT chk_skills CHECK (cooldown>=0)
)engine=innodb;

DELIMITER //
CREATE TRIGGER `chk_skill_trig` BEFORE INSERT ON `skills`
FOR EACH ROW
BEGIN
```

```
    IF NEW.cooldown < 0 THEN
        SIGNAL SQLSTATE '45000'
             SET MESSAGE_TEXT = 'An error occurred. Cooldown can't be
negative.';
    END IF;
END;//
DELIMITER ;

CREATE TABLE my_lists
(
id int NOT NULL AUTO_INCREMENT,
user_id int NOT NULL,
title varchar(255) NOT NULL,
created DATETIME,
modified DATETIME,
PRIMARY KEY (id),
FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
)engine=innodb


CREATE TABLE my_list_heroes
(
id int NOT NULL AUTO_INCREMENT,
my_list_id int NOT NULL,
hero_id int NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (my_list_id) REFERENCES my_lists(id) ON DELETE CASCADE,
FOREIGN KEY (hero_id) REFERENCES heroes(id) ON DELETE CASCADE
)engine=innodb


CREATE TABLE items
(
id int NOT NULL AUTO_INCREMENT,
name varchar(255) NOT NULL,
description text NOT NULL,
stats text NOT NULL,
cost int(8),
PRIMARY KEY (id)
)


CREATE TABLE recipes
(
id int NOT NULL AUTO_INCREMENT,
item_id int NOT NULL,
name varchar(255) NOT NULL,
cost int(8),
PRIMARY KEY (id),
FOREIGN KEY (item_id) REFERENCES items(id) ON DELETE CASCADE
)engine=innodb

CREATE TABLE recipe_items
(
id int NOT NULL AUTO_INCREMENT,
recipe_id int NOT NULL,
item_id int NOT NULL,
```

```
PRIMARY KEY (id),
FOREIGN KEY (recipe_id) REFERENCES recipes(id) ON DELETE CASCADE,
FOREIGN KEY (item_id) REFERENCES items(id) ON DELETE CASCADE
)engine=innodb


CREATE TABLE hero_builds
(
id int NOT NULL AUTO_INCREMENT,
hero_id int NOT NULL,
title varchar(255) NOT NULL,
created DATETIME,
modified DATETIME,
PRIMARY KEY (id),
FOREIGN KEY (hero_id) REFERENCES heroes(id) ON DELETE CASCADE
)engine=innodb

CREATE TABLE hero_build_items
(
id int NOT NULL AUTO_INCREMENT,
hero_build_id int NOT NULL,
item_id int NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY (hero_build_id) REFERENCES hero_builds(id) ON DELETE CASCADE,
FOREIGN KEY (item_id) REFERENCES items(id) ON DELETE CASCADE
)engine=innodb
```

BCNF Proofs

User (id, username, password, role, created, modified) = User (A, B, C, D, E, F)

Functional Dependencies = {A->CDEF, B->CDEF, A->B, B->A}

F+:
A -> ABCDEF
B -> ABCDEF

Fc:
A -> B
B -> ACDE

BCNF:
A is a superkey, B is a superkey, and C to F are trivial. BCNF table is ABCDEF.

Hero (id, name, biography, hit_points, damage, armor, movespeed, strength, agility, intelligence) = Hero (A, B, C, D, E, F, G, H, I, J)

Functional Dependencies = {A->BCDEFGHIJ, B -> ACDEFGHIJ, C -> ABDEFGHIJ}

F+:
A -> ABCDEFGHIJ
B -> ABCDEFGHIJ
C -> ABCDEFGHIJ

Fc:
A -> B
B -> C
C -> ABDEFGHIJ

BCNF:
A is a superkey, B is a superkey, C is a superkey, and D to J are trivial. BCNF table is ABCDEFGHI.

Skills (id, hero_id, name, description, ability_type, damage, mana_cost, cooldown) = Skills (A, B, C, D, E, F, G, H, I, J)

Functional Dependencies = {A -> B, A -> CDEFGHIJ}

F+:
A -> ABCDEFGHIJ

Fc:
A -> BCDEFGHIJ

BCNF:
A is a superkey and B to J are trivial. BCNF table is ABDEFGHIJ.

Lists (id, user_id, title, created, modified) = Lists (A, B, C, D, E)

Functional Dependencies = {A -> B, A -> C, A -> BC, A -> DE}

F+:

A -> ABC

Fc:
A -> BC

BCNF:
A is a superkey. B and C are trivial. BCNF table is ABC.

List_heroes (id, list_id, hero_id) = List_heroes (A, B, C)

Functional Dependencies = {A -> BC, BC -> A}

F+:
A -> ABC
BC -> ABC

Fc:
A -> BC
BC -> A

BCNF:
A is a superkey and BC is a superkey. B and C are trivial. BCNF table is ABC.

Items (id, name, description, cost) = Items (A, B, C, D)

Functional Dependencies = {A -> BCD, B -> ACD, C ->ABD}

F+:
A -> ABCD
B -> ABCD
C -> ABCD

Fc:
A -> BCD
B -> ACD
C -> ABD

BCNF:
A is a superkey , B is a superkey, C is a superkey, and D is trivial. BCNF table is ABCD.

Recipes (id, item_id, name, cost) = Recipes (A, B, C, D)

Functional Dependencies = {A -> BCD, C ->ABD}

F+:
A -> ABCD
C -> ABCD

Fc:
A -> BCD
C -> ABD

BCNF:
A is a superkey, B is trivial, C is a superkey, and D is trivial. BCNF table is ABCD.

Recipe_items (id, recipe_id, item_id) = Recipe_items (A, B, C)

Functional Dependencies = {A -> BC, BC -> A}

F+:
A -> ABC
BC -> ABC

Fc:
A -> BC
BC -> A

BCNF:
A is a superkey and BC is a superkey. B and C are trivial. BCNF table is ABC.

Hero_builds (id, hero_id, title, created, modified) = Hero_builds (A, B, C, D, E)

Functional Dependencies = {A -> BCDE, BC->A}

F+:
A -> ABCDE
BC -> ABCDE

Fc:
A -> BC
BC -> DE

BCNF:
A is a superkey and BC is a superkey. B to E are trivial otherwise. BCNF table is ABCDE.

Hero_build_items (id, hero_build_id, item_id) = Hero_build_items (A, B, C)

Functional Dependencies = {A -> BC, BC -> A}

F+:
A -> ABC
BC -> ABC

Fc:
A -> BC
BC -> A

BCNF:
A is a superkey and BC is a superkey. B and C are trivial. BCNF table is ABC.

Part III - Evaluation of Product

We tested the system to make sure that it would fulfill all of the desired functionality by actually using the application. We did a number of operations to test what would work with all the basic CRUD operations for the database in any way we thought that the system might be used. To ensure that foreign key constraints worked we did inserts with valid and invalid inputs to check if the database was responding to invalid key space entries appropriately. We also tested the cascading delete for dependencies between certain entities. We did many tests where we would remove specific entries from the database to see if it would cause issues with data input and output. We also tested the user access to certain functions in the database to make sure that user types in the database were functioning as we

intended so that not every user was being treated as a super user and there were the appropriate permissions granted to sub users when they access the database.

In order to make sure the application was working properly we used a similar approach in testing by using the actual system as we felt a user would interact with the system. It would be difficult to do a full comprehensive test of this as the user is unpredictable at best however we tried to do a pretty extensive survey of all behaviors that we believed to be pretty standard for the user. Most of the testing dealt with making sure users could only access what they were supposed to and unable to make any changes where they were not granted permissions. We also made sure that the application would correctly be able to handle the users input if it was invalid by using form validation to the database so that there would not be any errors in querying the database from the application side and that the application would not suddenly break on invalid input to the forms. We also tested the workflow of the application to make sure it made sense from the users perspective as best we could.

Sample Data

(7," Bloodseeker", "bio text which is too long for example", 587, 53, 3, 300, 23, 24, 18)

This would be a hero named Bloodseeker with the hero id 7 some sort of accompanying bio text to tell the background of the character and the other numbers represent the different stats of health, damage, armor, movespeed, strength, agility, and intelligence respectively.

(1,1, "Fissure", "Slams the ground with a mighty totem, fissuring the earth while stunning and damaging enemy units in a line. Creates an impassable ridge of stone.", "magical",125,125,15)

This is an example of one of the skill entries in the database. The first number represents the skill id and the second number is the hero id indicating which hero the skill belongs to. This is used to get the name of the hero and display it. There is a text description of what the ability does followed by an

indication of what type of damage the ability can do either physical, magical, or pure, or in the case where the skill does no damage this field says passive instead. The next three numbers represent the damage, mana cost, and cool down of the skill.

(8, “Band of Elvenskin”, “A tensile fabric often used for its light weight and ease of movement”, “+6 Agility”, 450)

This represents an item in the database. The item is called the Band of Elvenskin and is identified in the database with item id 8. The description is a copy of the in-game description of the item. This is followed by another text area which represents the stat bonuses of the item. In this case the item indicates that the character with the item receives a bonus of 6 agility. The last number is the cost in gold of the item.

Sample Queries

SELECT * FROM skills WHERE hero_id = 4

This query will return every skill belonging to the hero with id 4.

INSERT INTO my_lists_heroes(my_list_id,hero_id) VALUES(3,4)

This query will add hero with id 4 to list with id 3

UPDATE items SET cost=500 WHERE Name = “Circlet”

This query will change the cost of the item with the name “Circlet” from its proper value of 185 to 500

DELETE FROM heroes WHERE id = 2

This query will delete the entry from the heroes table that has id value equal to 2. Because of cascade specification when we delete this entry it will also remove any skills that are associated with the hero that was removed.

These are just a few examples of the queries we ran. They are not very complex but none of the queries that are used to make the application functional are any more complicated than any of these are. There are just many more forms of each that are used in different situations for the application but which follow a similar form.