Bachelor of Science in Computing Year 3

Advanced Databases Project (50%)

Cauê Duarte 2017228
Felipe Mantovani 2017192
Olga Kiseleva 2017136
Taras Boreyko 2017284

Group A

2019

Part A. Database analysis and design

Description of the Application

This application is designed to host backend information of the *Sally Forth* MMO-RPG game. The backend consists of a Mysql database management system server. The base shall store all in-game info and components such as players, monsters, items and so forth.

Assumptions

The game

Sally Forth is a game where players will be able to interact with each other, whether by cooperation, conflict, or even go on a solo adventure. The game is going to have many components that can be managed by the players and the so-called NPCs (non player character).

The players

Players will be able to possess, purchase and use any item in the game. Also, the purchases will be carried out between players or NPCs. Each NPC, or a group of it, can sell or buy items to or from its customers, that happens to be the players.

Players can attack other players or even NPCs by the use of an item category called weapon or using spells. If a player dies, it may respawn in the city where they live. At the location where the body of that player lied, there will be dropped items where other players can loot from it. So we can assume that everything that dies can drop one or more items.

Vulnerables

For the category of “things” that dies or can be harmed, it is included players, monsters and npcs. Also, this category of “things that can can die be healthy harmed” will be called *Vulnerables*.

Every *vulnerable* drops items they were wearing or carrying when they died. Also, every *vulnerable* can improve its skills. The set of skills will include the level, magic level, a bunch of physical attack skills and among others. Also, every *vulnerable* has a healthy state. The healthy state can be broken down into three metrics:

1. *Hitpoints* - points that keeps the vulnerable alive.
2. *Manapoints* - For the use of magic and spells casting.
3. *Stamina* - The ability to keep physically active for several hours without the need of resting.

When the *hitpoints* reaches to the value of 0, the vulnerable shall die. When the *manapoints* touches zero, the *vulnerable* should not be able to cast any spells.

Spells

Spells are magical elements that can change the caster or the target state. For instance, if an attack spell is cast against a target, it should subtract from its hitpoint or manapoint, of the *vulnerable* a value in order to cause they a harm. On the other end, the target could cast a healing spell to increase its hitpoint and add value to it. Some accessories such as amulets, rings, among others, can be used to increase the effect of healthy healing.

Item

An item can be anything in the game that the *vulnerables* can possess, sell, purchase or wield. Each item shall have a name and weight attribute. There will be

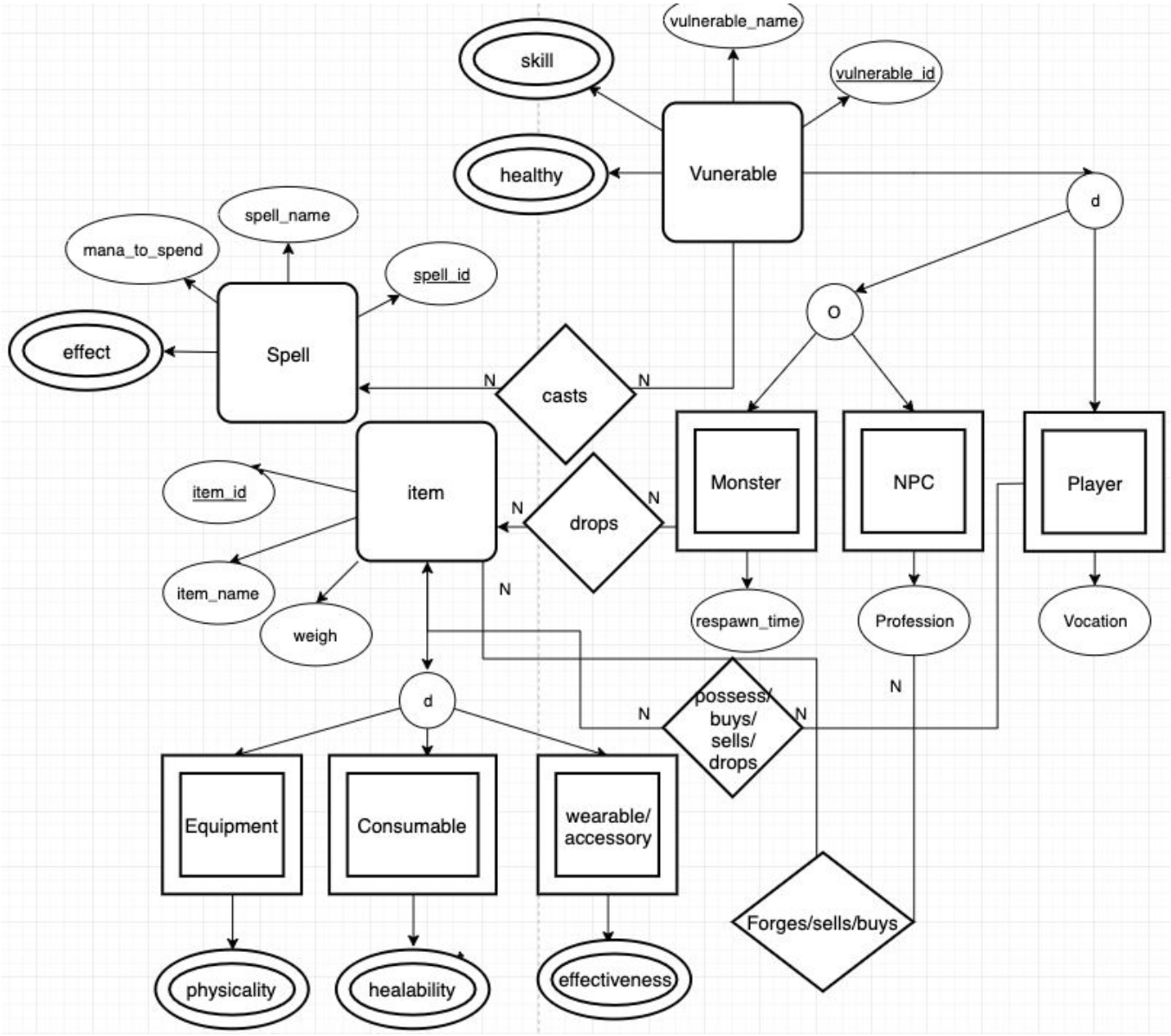
several subcategories of the item in the game. An item can consist of many forms. It may be an *equipment*, for instance, a weapon or a shield, it can be a *consumable*, like a mana potion, water, poison, health potion, and so forth. Also, the *accessories* are considered to be an item, such as a life ring, amulet of loss (for not dropping items when dying), among others. The *physicality* property of an equipment consists of a multivalued attribute that represents how much that equipment can defend or attack or both. An example of an equipment could be a weapon, a shield, armor, helmet. The *healability* of consumable is how much it can heal one of the healthy states of a *vulnerable.* For instance, a health potion may heal the consumer, and it adds to their hitpoints. Sometimes, a consumer may drink poison, but points shall be taken from the hitpoints.

NPC

An NPC is a subclass of *Vulnerable,* like the *Players.* NPCs shall be either a monster/creature in the game, at the same time, or just a person that is not player. Monsters and NPCs can overlap abilities. An example of an NPC can be a merchant that forges, sells or buys items, a lore master that assigns quests to players, or even a random human/person that can be found by chance, with no purpose pre-defined, but still can interact with the players

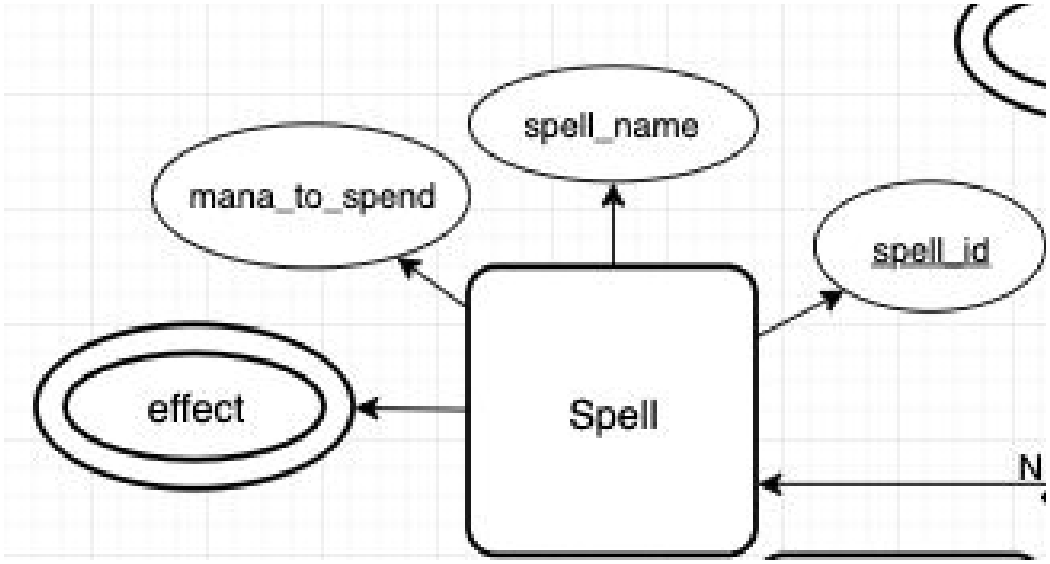
Enhanced Entity Relation Diagram

The diagram below represents all of the assumptions made above out of any relation mentioned in the previous section.



Spell properties

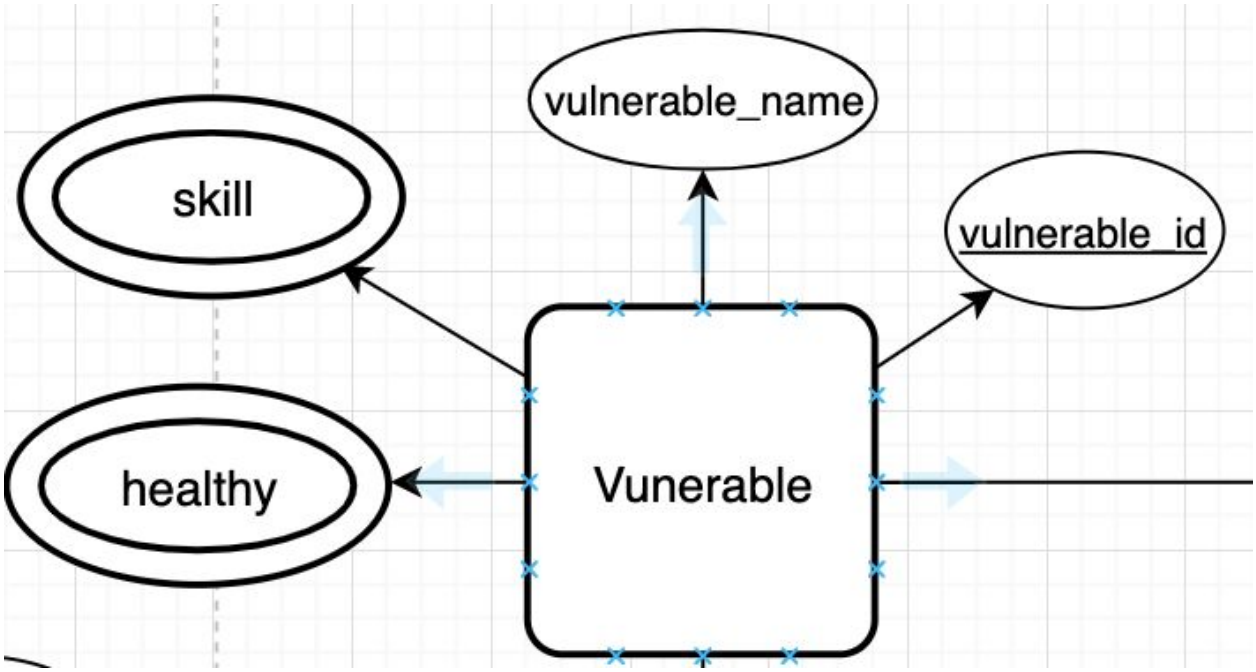
The relation spells shall have a spell_id, spell_name, mana_to_spend and the effect attribute. The effect attribute is a multivalued attribute, which is composed of effect_name, effect_efficiency and effect_description. These multivalued attributes shall be shown in the cross_foot notation.



Vulnerable properties

The vulnerables has a set of properties pre-defined that disregards whether they are players, NPCs or monsters. The properties are vulnerable_id, vulnerable_name, skill and healthy.

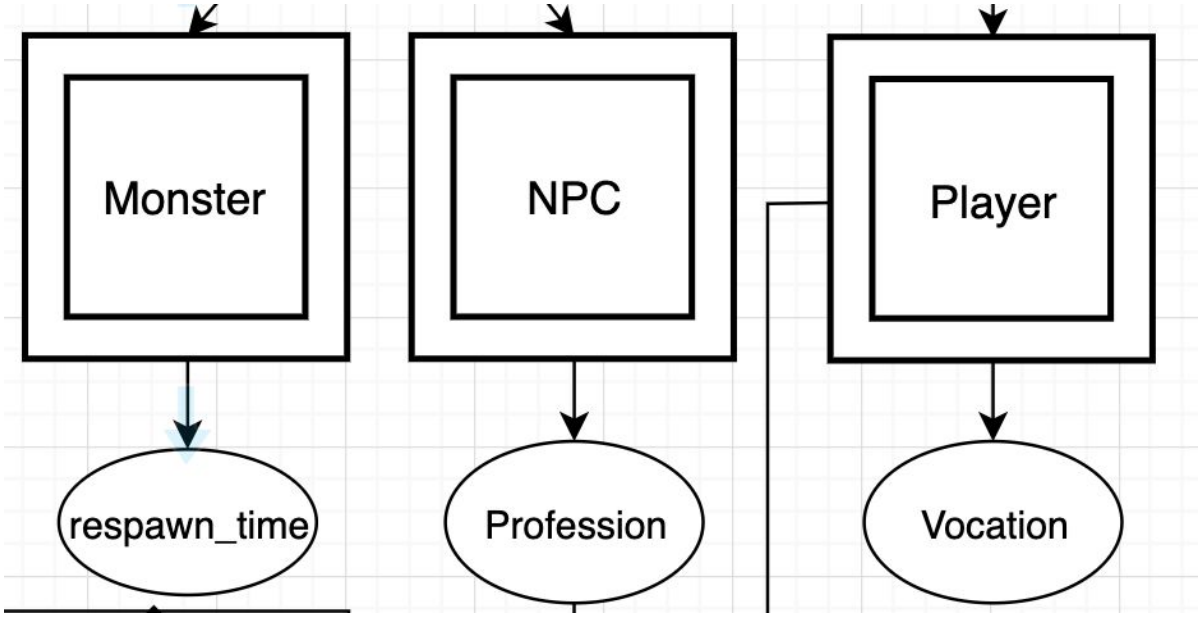
The skill and healthy properties are multivalued attributes. The healthy consists of the state of the manapoints, hitpoints and stamina the vulnerable has, whereas the skill property is composed of level, magic_level, sword_fighting, fishing, etc (See crossfoot notation for more details).



Vulnerables descendents properties

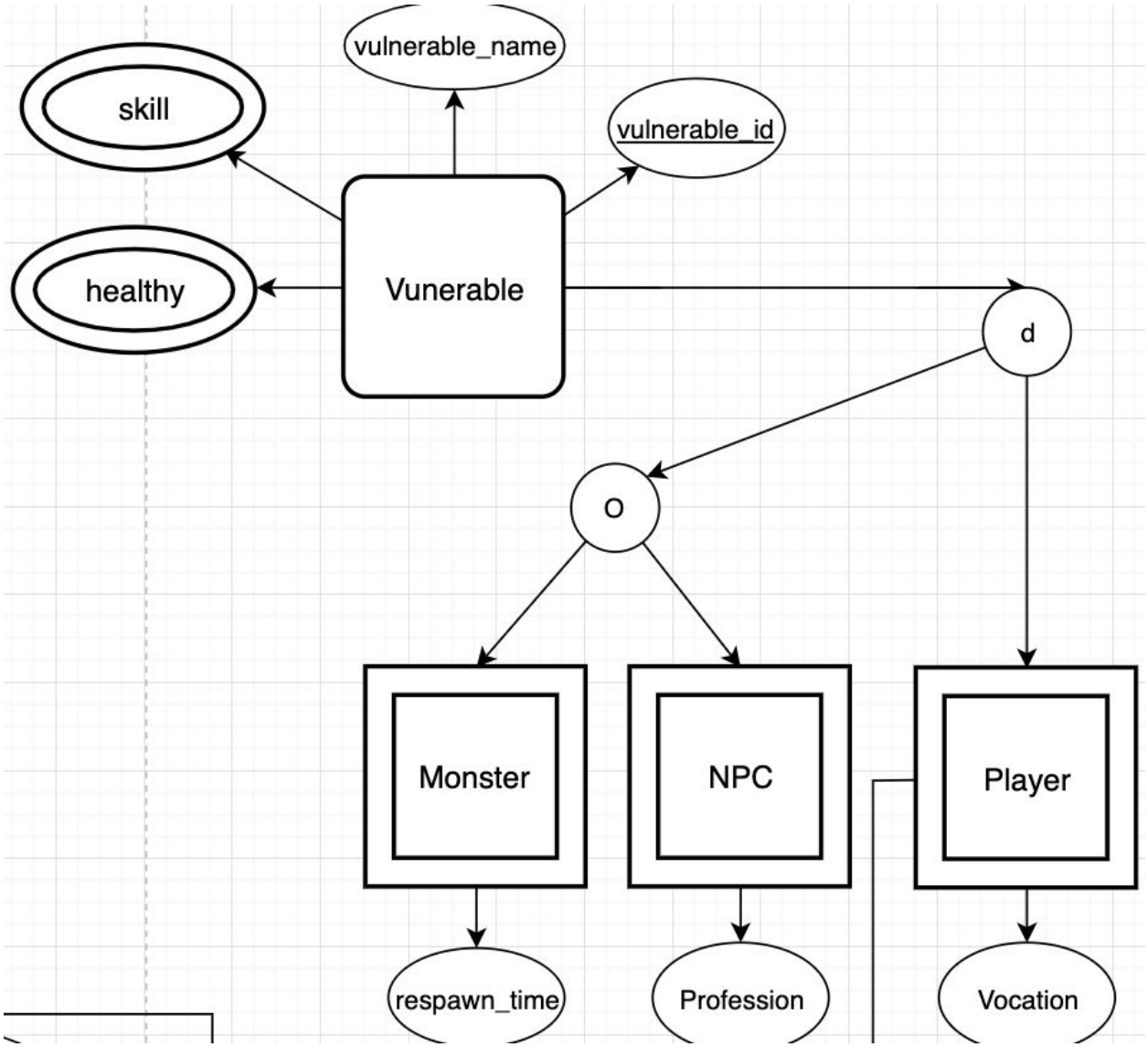
The vulnerable children are composed of three more weak entities. The monster, NPC and player. A monster can be an NPC, but never a player.

- The monster will have the respawn time attribute
- The player has the vocation attribute.
- The NPC has the profession attribute.



Vulnerables relations (overlappings and disjoints)

As previously mentioned, the Monsters and NPCs have overlapping abilities, but they are disjoint from the player children class. Therefore, the diagram will look as follows:

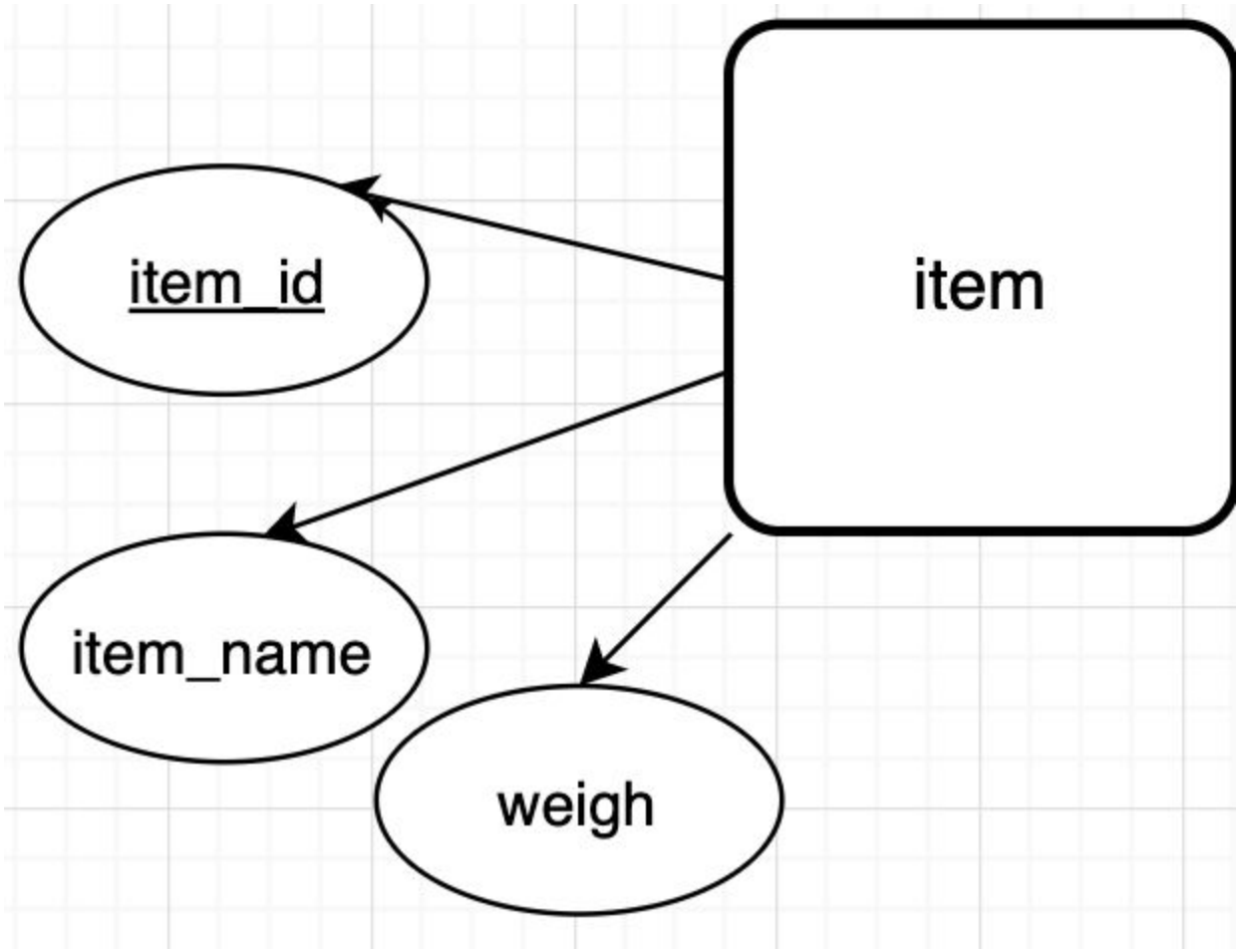


Items properties

Items are designed to have the following attributes regardless of being an equipment, consumable or an accessory:

- Item_id;
- Item_name;
- Item_weight;

The diagram below shows its attributes



Items disjoint relations (inheritance)

The item entity is the ancestor of 3 subclasses, the Equipment, Consumable and Accessories.

The Equipment entity has a multivalued physicality attribute. It consists of a set of the following predefined attributes:

- Attack;
- Defense;

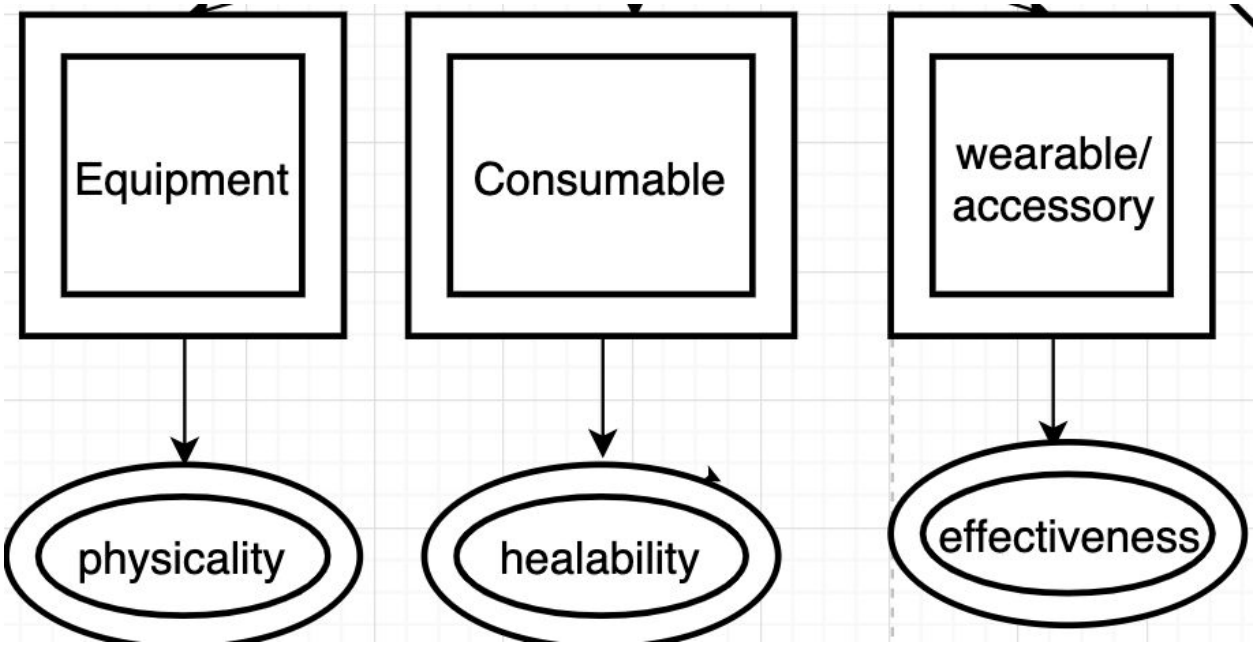
The healability is part of the Consumable item. It is concerned with how harmful or healthy it is to drink (consume) a liquid item. It also contains a multivalued attribute composed of the following properties:

- Manapoints_to_heal;
- Hitpoints_to_hel.

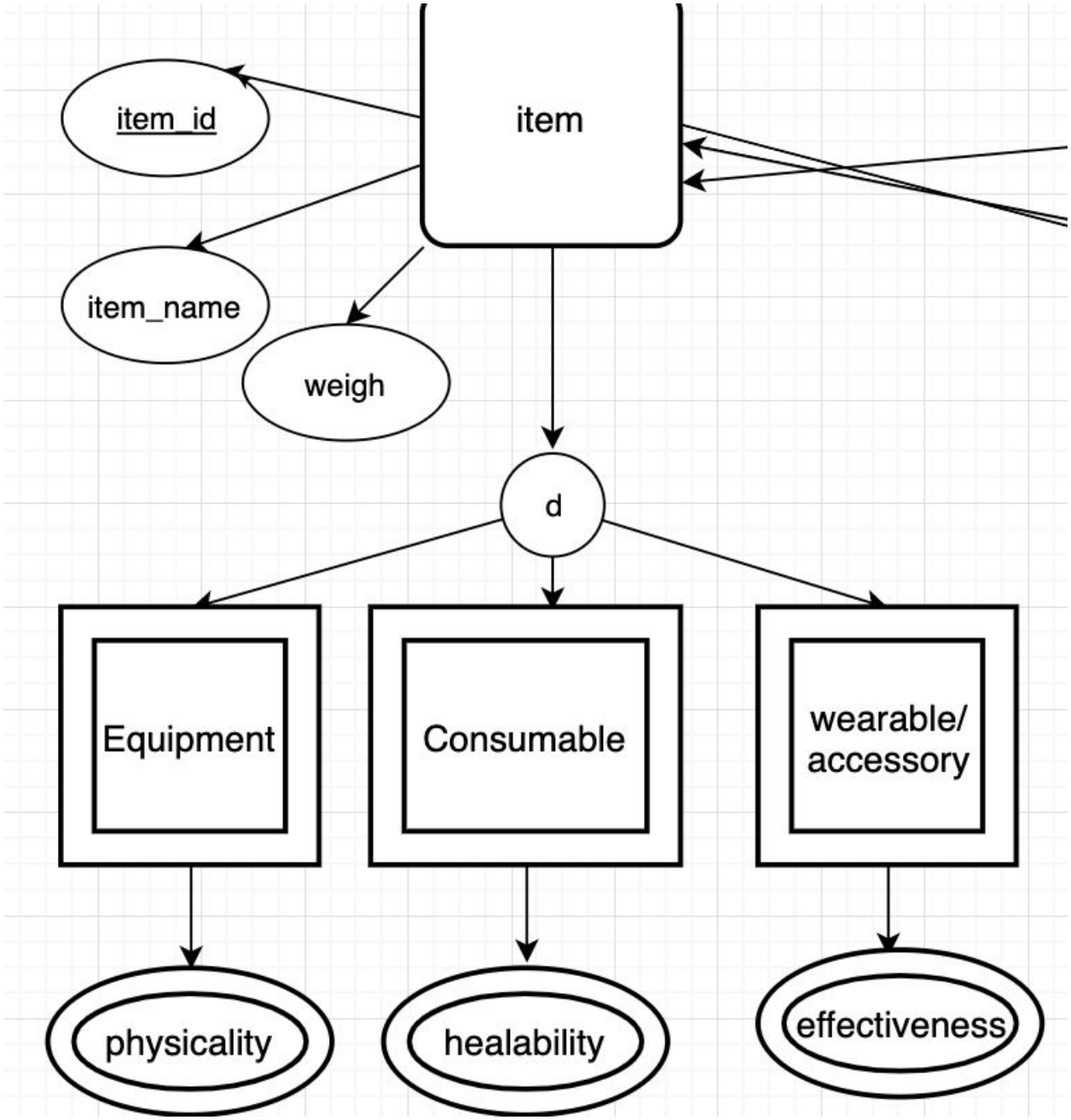
The effectiveness multivalued attribute is part of the accessory item. It is concerned with how effect that item can aid its wearer on a given task. For example, a life ring shall be very effect on increasing its wearer hitpoints (quicker heal) or an amulet of loss has 100% of effectiveness from preventing an item dropping when the vulnerable dies, etc.

The effectiveness is composed of the following attributes:

- Item_effect



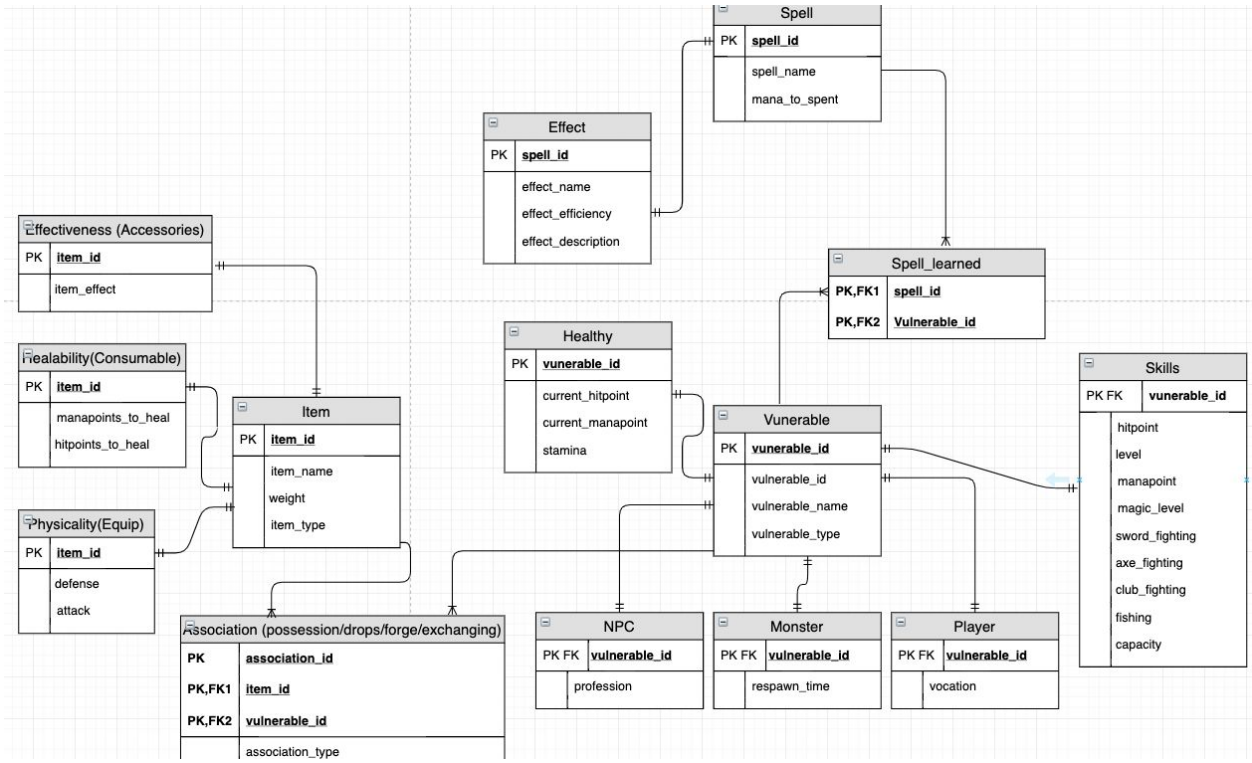
The three children item subclasses are completely disjoint, as shown in the diagram below:



Note: For more details on the multivalued attribute, check the crossfoot notation on the following chapters.

Crow's foot Notation

The relations represented below are represented in 2NF:



One view per student

Felipe Mantovani

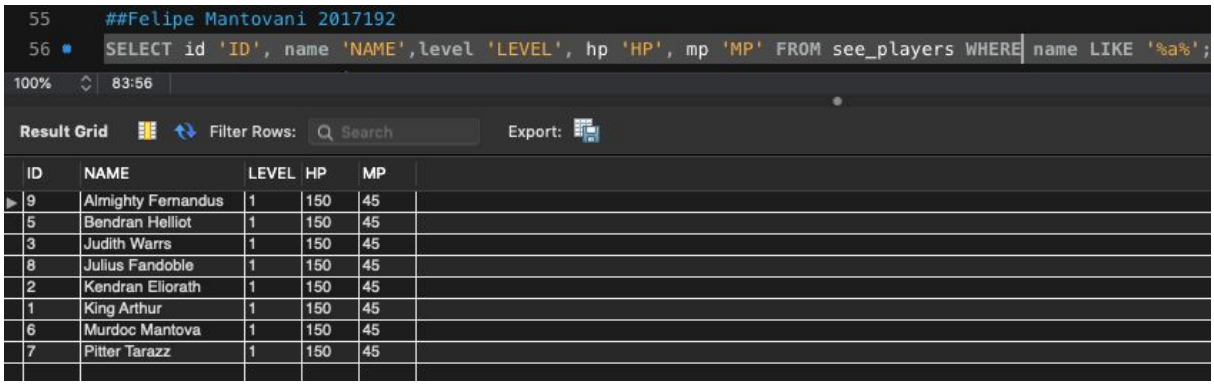
View Name	see_players
View Dependency	see_vulnerables
View Description	Shows all players registered to the database
Tables incorporated (including the dependency's tables)	Players, skills, recordable, vulnerable, healthy

Screenshot of the view declaration

```
DROP VIEW IF EXISTS see_players;
CREATE VIEW see_players AS
    SELECT
        sv.id 'ID',
        sv.name 'NAME',
        p.vocation 'VOCATION',
        s.level 'LEVEL' ,
        #t.taxonomy_type 'GROUP', Felipe Mantovani, 2017192
        sv.subgroup 'SUBGROUP',
        s.hitpoint 'HP',
        s.manapoint 'MP',
        s.capacity 'CAPACITY',
        s.magiclevel 'MAGIC LEVEL' ,
        s.swordfighting 'SWORD FIGHTING',
        s.axefighting 'AXE FIGHTING' ,
        s.clubfighting 'CLUB FIGHTING',
        s.fishing 'FISHING',
        sv.currenthp 'CURRENT HP' ,
        sv.currentmp 'CURRENT MP',
        sv.stamina 'STAMINA'
    FROM see_vulnerables sv
        JOIN players p
            ON p.vulnerable_id = sv.id
        JOIN skills s
            ON s.vulnerable_id = p.vulnerable_id;
```

Screenshot of the view selecting statement

Selecting all players with the letter “a” on name.



```
55    ##Felipe Mantovani 2017192
56 •  SELECT id 'ID', name 'NAME',level 'LEVEL', hp 'HP', mp 'MP' FROM see_players WHERE name LIKE '%a%';
```

100% 83:56

Result Grid Filter Rows: Search Export:

ID	NAME	LEVEL	HP	MP
9	Almighty Fernandus	1	150	45
5	Bendran Helliot	1	150	45
3	Judith Warrs	1	150	45
8	Julius Fandoble	1	150	45
2	Kendran Eliorath	1	150	45
1	King Arthur	1	150	45
6	Murdoc Mantova	1	150	45
7	Pitter Tarazz	1	150	45

Taras Boreyko

View Name	see_weapons
View Dependency	see_equipment
View Description	Show the weapons from equipment
Tables incorporated (including the dependency’s tables)	Equipment, weapons, physicality, recordable.

Screenshot of the view declaration

```
546
547     ## Taras Boreyko 2017284
548 •   DROP VIEW IF EXISTS see_weapons;
549 •   CREATE VIEW see_weapons AS
550         SELECT
551             se.id 'ID',
552             se.name 'NAME',
553             se.weight 'WEIGHT',
554             #se.group 'GROUP',
555             #se.subgroup 'SUB GROUP',
556             se.defense 'DEFENSE',
557             w.attack 'ATTACK'
558         FROM see_equipment se
559             JOIN weapons w
560                 ON w.physicality_id = se.ID;
```

Screenshot of the view selecting statement

```
91      ##Taras Boreyko 2017284
92 •    SELECT id 'ID',name 'NAME',weight 'WEIGHT', defense 'DEFENSE', attack 'ATTACK' FROM see_weapons;
93
94
```

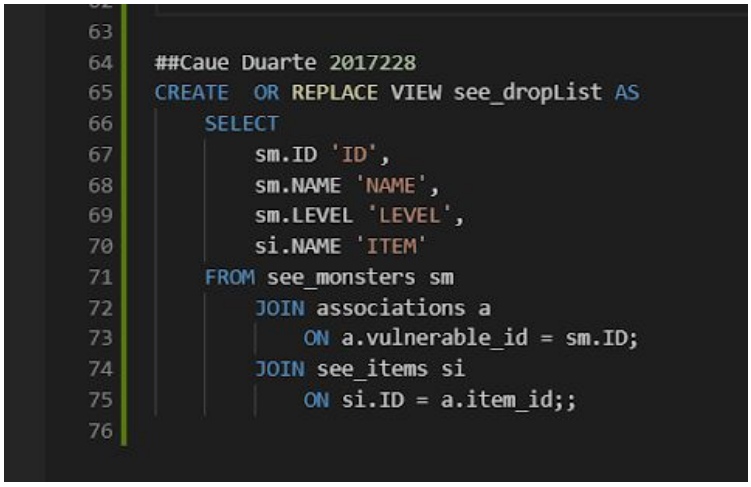
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

ID	NAME	WEIGHT	DEFENSE	ATTACK
15	Fire Sword	56	23	35
16	Sword	78	10	16
17	Knight Axe	48	15	36

Caue Duarte

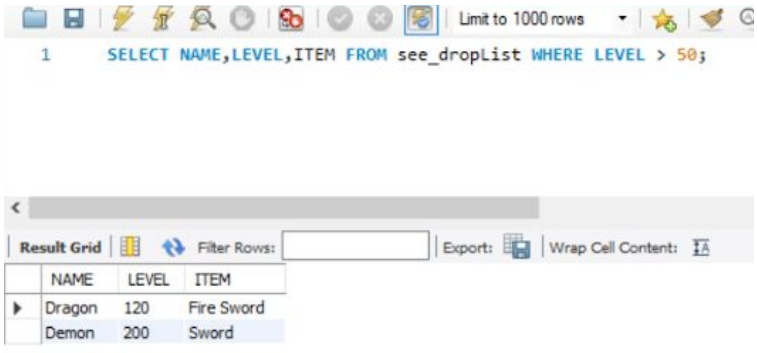
View Name	see_dropList
View Dependency	see_monsters, see_items
View Description	Shows the items dropped by all monsters
Tables incorporated (including the dependency's tables)	Item, vulnerable, recordable, skills

Screenshot of the view declaration



```
63
64  ##Caue Duarte 2017228
65  CREATE  OR REPLACE VIEW see_dropList AS
66      SELECT
67          sm.ID 'ID',
68          sm.NAME 'NAME',
69          sm.LEVEL 'LEVEL',
70          si.NAME 'ITEM'
71      FROM see_monsters sm
72          JOIN associations a
73              ON a.vulnerable_id = sm.ID;
74          JOIN see_items si
75              ON si.ID = a.item_id;;
76
```

Screenshot of the view selecting statement



```
Limit to 1000 rows
1   SELECT NAME,LEVEL,ITEM FROM see_dropList WHERE LEVEL > 50;
```

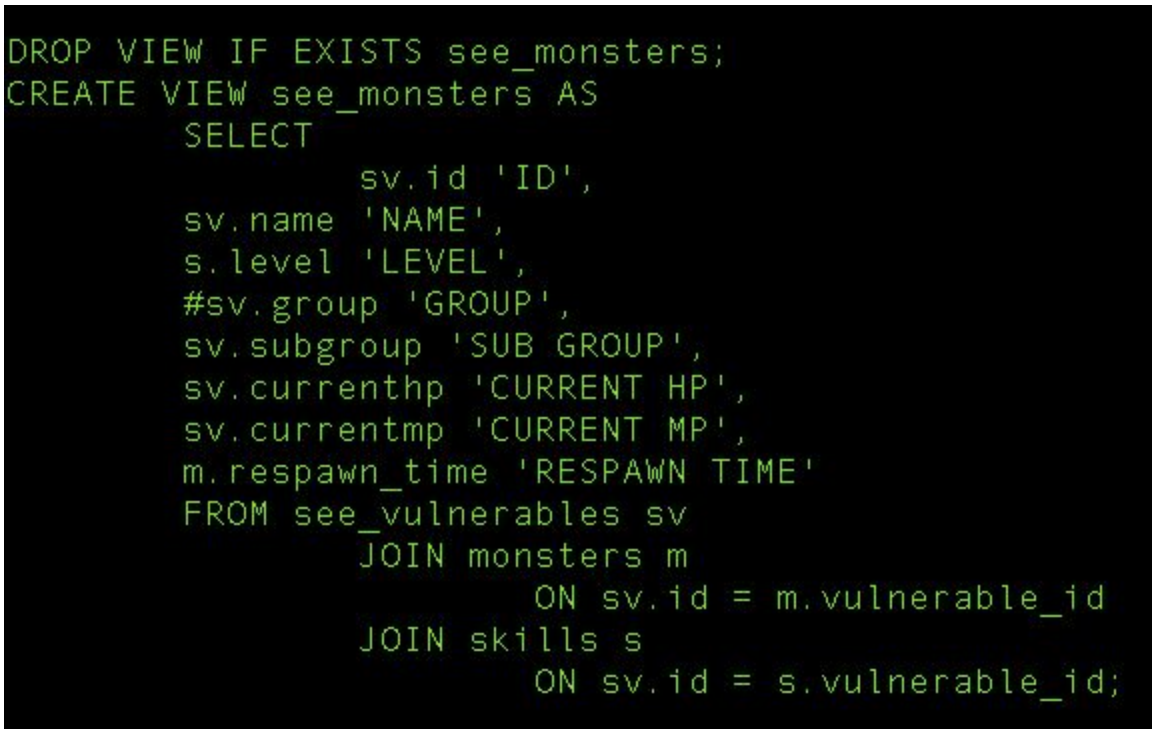
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

NAME	LEVEL	ITEM
Dragon	120	Fire Sword
Demon	200	Sword

Olga Kiseleva

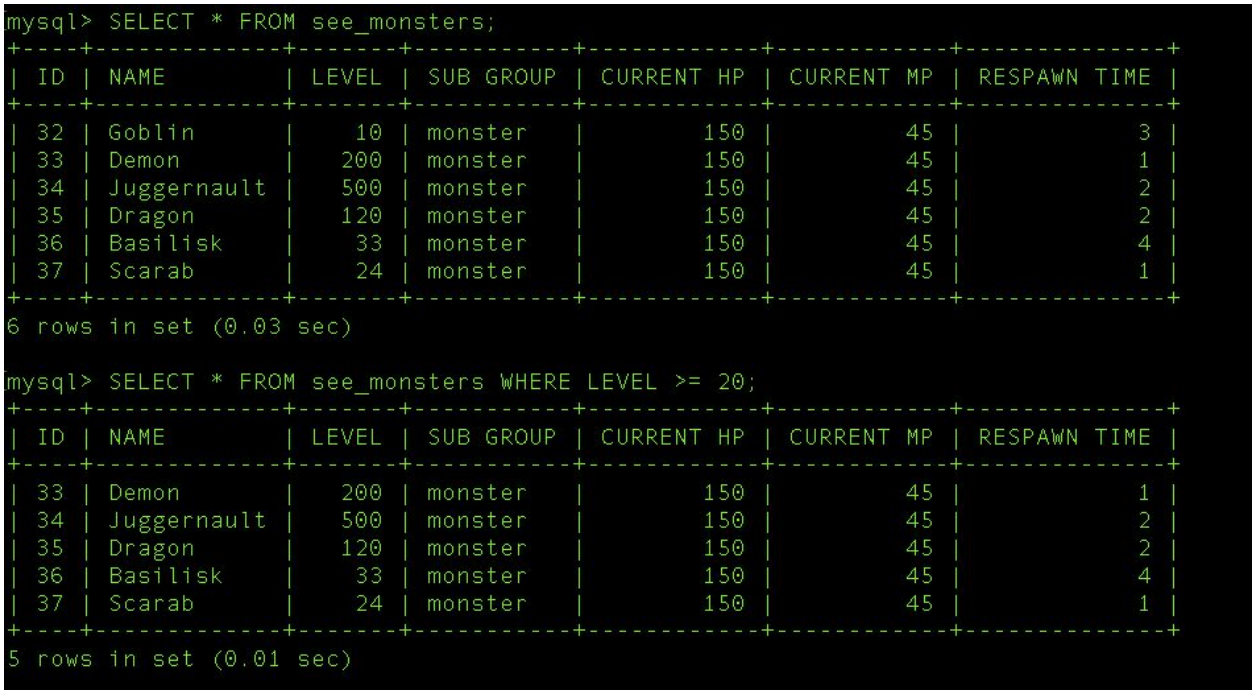
View Name	see_monsters
View Dependency	see_vulnerables
View Description	Selects all the monster saved in the database and their attributes
Tables incorporated (including the dependency's tables)	Monsters, skills, taxonomy, recordable, vulnerable, healthy.

Screenshot of the view declaration



```
DROP VIEW IF EXISTS see_monsters;
CREATE VIEW see_monsters AS
        SELECT
                sv.id 'ID',
        sv.name 'NAME',
        s.level 'LEVEL',
        #sv.group 'GROUP',
        sv.subgroup 'SUB GROUP',
        sv.currenthp 'CURRENT HP',
        sv.currentmp 'CURRENT MP',
        m.respawn_time 'RESPAWN TIME'
        FROM see_vulnerables sv
                JOIN monsters m
                        ON sv.id = m.vulnerable_id
                JOIN skills s
                        ON sv.id = s.vulnerable_id;
```

Screenshot of the view selecting statement



```
mysql> SELECT * FROM see_monsters;
+----+-------------+-------+-----------+------------+------------+--------------+
| ID | NAME        | LEVEL | SUB GROUP | CURRENT HP | CURRENT MP | RESPAWN TIME |
+----+-------------+-------+-----------+------------+------------+--------------+
| 32 | Goblin      |    10 | monster   |        150 |         45 |            3 |
| 33 | Demon       |   200 | monster   |        150 |         45 |            1 |
| 34 | Juggernault |   500 | monster   |        150 |         45 |            2 |
| 35 | Dragon      |   120 | monster   |        150 |         45 |            2 |
| 36 | Basilisk    |    33 | monster   |        150 |         45 |            4 |
| 37 | Scarab      |    24 | monster   |        150 |         45 |            1 |
+----+-------------+-------+-----------+------------+------------+--------------+
6 rows in set (0.03 sec)

mysql> SELECT * FROM see_monsters WHERE LEVEL >= 20;
+----+-------------+-------+-----------+------------+------------+--------------+
| ID | NAME        | LEVEL | SUB GROUP | CURRENT HP | CURRENT MP | RESPAWN TIME |
+----+-------------+-------+-----------+------------+------------+--------------+
| 33 | Demon       |   200 | monster   |        150 |         45 |            1 |
| 34 | Juggernault |   500 | monster   |        150 |         45 |            2 |
| 35 | Dragon      |   120 | monster   |        150 |         45 |            2 |
| 36 | Basilisk    |    33 | monster   |        150 |         45 |            4 |
| 37 | Scarab      |    24 | monster   |        150 |         45 |            1 |
+----+-------------+-------+-----------+------------+------------+--------------+
5 rows in set (0.01 sec)
```

One stored procedure per student

Felipe Mantovani

Procedure name	insert_taxonomy
Procedure dependency (helper procedures used in the declaration)	None, this procedure is the helper for many other procedures
Description	It inserts data into the recordable and taxonomy relations. It retrieves this new generated id and returns it by the use of the OUT parameter to aid the other procedures to function based on the same id..

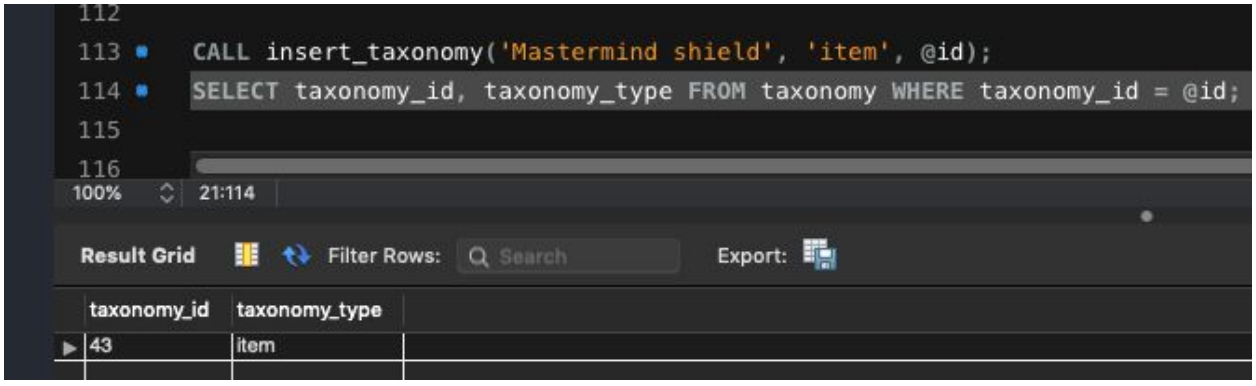
Screenshot of the procedure declaration

```
DROP PROCEDURE IF EXISTS insert_taxonomy;
##CREATION OF A PROCEDURE THAT GENERATES A TAXONOMY AND RETURNS THE ID
##CREATED BY FELIPE MANTOVANI 2017192
DELIMITER //
CREATE PROCEDURE insert_taxonomy(IN name_ text, IN type text, OUT id int)
BEGIN
    DECLARE id_ INT(11) DEFAULT(0);
    INSERT INTO recordable (name) VALUE (name_);
    set id_ = get_id(name_);
    INSERT INTO taxonomy VALUE (id_, type);
    SET id = id_;
END //
DELIMITER ;
```

Screenshot of the procedure call

The insert taxonomy procedure is never meant to be called on its own. Its purpose is only to be used as a helper for the others procedures. This means there won't ever be a type in the game called "Taxonomy", but there will be Vulnerable, Items, Spells, that have taxonomy on their structure.

For the purpose of this screenshot, I will be inserting a taxonomy record, but, again, this should not be used in this way:



Taras Boreyko

Procedure name	insert_equipment
Procedure dependency (helper procedures used in the declaration)	Insert_item
Description	Procedure inserts the equipment. name_ , weight, defense, as IN, generated new id and returns it by the use of the OUT parameter.

Screenshot of the procedure declaration

```
#CREATE PROCEDURE THAT INSERT EQUIPMENT
DROP PROCEDURE IF EXISTS insert_equipment;
DELIMITER //
CREATE PROCEDURE insert_equipment(IN name_ text, IN weight INT, IN defense TEXT, OUT id INT)
BEGIN
    CALL insert_item(name_, weight, 'equipment', @id);
    INSERT INTO physicality VALUE (@id, defense);
    SET id = @id;
END //
DELIMITER ;
```

Screenshot of the procedure call

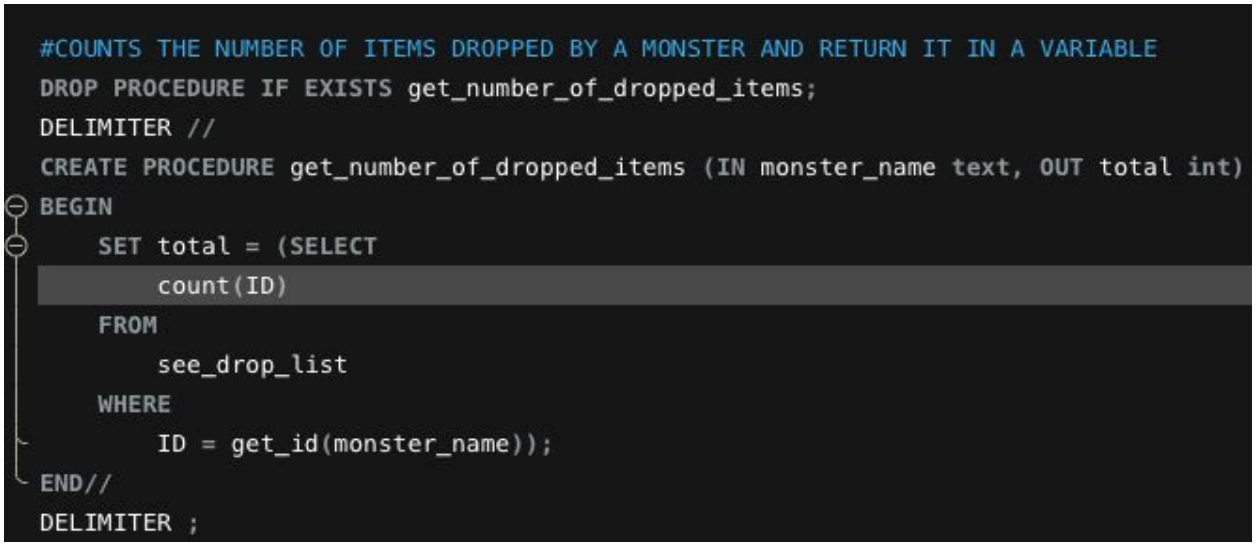
{screenshot here}

Caue Duarte

Procedure name	get_number_of_dropped_items
Procedure dependency (helper procedures used in the declaration)	None

Description	This procedure receives an int representing the monster id as IN, counts the number of tuples with that id in the see_dropList view and returns the count as the OUT.

Screenshot of the procedure declaration



```
#COUNTS THE NUMBER OF ITEMS DROPPED BY A MONSTER AND RETURN IT IN A VARIABLE
DROP PROCEDURE IF EXISTS get_number_of_dropped_items;
DELIMITER //
CREATE PROCEDURE get_number_of_dropped_items (IN monster_name text, OUT total int)
BEGIN
    SET total = (SELECT
        count(ID)
    FROM
        see_drop_list
    WHERE
        ID = get_id(monster_name));
END//
DELIMITER ;
```

Screenshot of the procedure call

{screenshot here}

Olga Kiseleva

Procedure name	insert_item
Procedure dependency (helper procedures used in the declaration)	insert_taxonomy
Description	This procedure is a helper for others items insertion procedures, such as insert_equipment and insert_equipment_weapons and consumables. The out parameter stores the id of the most inserted item.

Screenshot of the procedure declaration

```
DROP PROCEDURE IF EXISTS insert_item;
DELIMITER //
CREATE PROCEDURE insert_item(IN name_ text , IN weight INT, IN item_type TEXT, OUT id int)
BEGIN
        CALL insert_taxonomy(name_, 'item', @id);
    INSERT INTO item VALUE (@id, weight, item_type);
    SET id = @id;
END //
DELIMITER ;
```

Screenshot of the procedure call

```
mysql> CALL insert_item('Vial of blood', 10, 'consumable', @id);
Query OK, 1 row affected (0.01 sec)

mysql> select id, name, weight FROM see_items WHERE id = @id;
+----+---------------+--------+
| ID | NAME          | WEIGHT |
+----+---------------+--------+
| 45 | Vial of blood |     10 |
+----+---------------+--------+
1 row in set (0.00 sec)
```

Stored Procedure with subqueries

@TODO - I still need to do those, team. I will keep you posted when I've finished those.

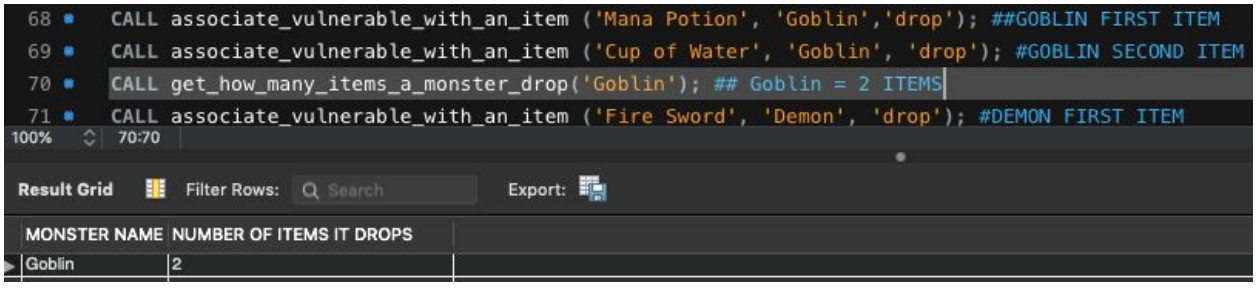
Felipe Mantovani

Procedure Declaration

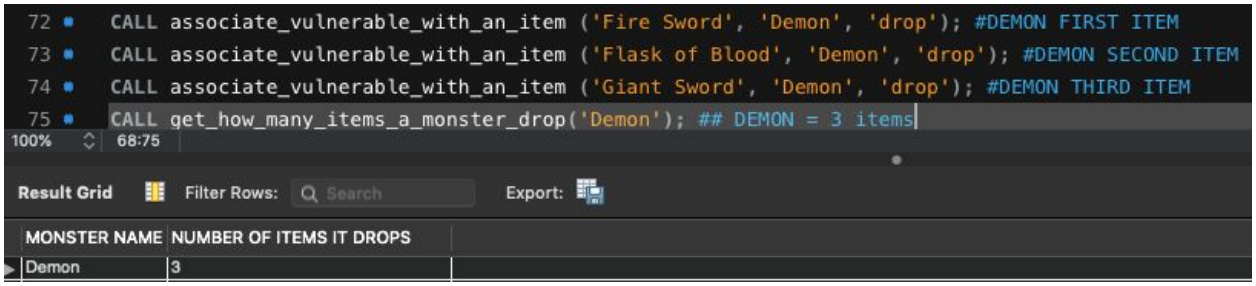
```
##FELIPE MANTOVANI 2017192
## THIS PROCEDURE selects TWO COLUMNS: THE MONSTER NAME AND ITS DROPPED ITEMS AMOUNT
DROP PROCEDURE IF EXISTS get_how_many_items_a_monster_drop;
DELIMITER //
CREATE PROCEDURE get_how_many_items_a_monster_drop (IN monster_name text)
BEGIN
    SELECT
        name 'MONSTER NAME',
        (SELECT
            count(item_id)
        FROM
            associations
        WHERE
            vulnerable_id = get_id(monster_name)) 'NUMBER OF ITEMS IT DROPS'
    FROM
        recordable
    WHERE
        id = get_id(monster_name);
END//
DELIMITER ;
```

Calling procedure

In the screenshot below, it is being assigned two items, Mana Potion and Cup of water for the monster ‘Goblin’ to drop and below is the calling procedure to check how many items the Goblin creature drops.



The same for the Demon monster. It has been assigned three items for it to drop.



```
72  CALL associate_vulnerable_with_an_item ('Fire Sword', 'Demon', 'drop'); #DEMON FIRST ITEM
73  CALL associate_vulnerable_with_an_item ('Flask of Blood', 'Demon', 'drop'); #DEMON SECOND ITEM
74  CALL associate_vulnerable_with_an_item ('Giant Sword', 'Demon', 'drop'); #DEMON THIRD ITEM
75  CALL get_how_many_items_a_monster_drop('Demon'); ## DEMON = 3 items
```

100% 68:75

Result Grid Filter Rows: Search Export:

MONSTER NAME	NUMBER OF ITEMS IT DROPS
Demon	3

Caue Duarte

@TODO - On the way

Olga Kiseleva

```
## THIS PROCEDURE LISTS HOW MANY SPELLS EACH VULNERABLE CAN CAST. THE IN PARAMETER IS
DROP PROCEDURE IF EXISTS get_learned_spells;
DELIMITER //
CREATE PROCEDURE get_learned_spells (IN vulnerable_type_ text)
BEGIN
        SELECT
                name 'VULNERABLE NAME',
        (SELECT
                        count(spell_id)
                FROM
                        spells_learned s
                WHERE
                        s.vulnerable_id = v.vulnerable_id) 'AMOUNT OF LEARNED SPELLS'
        FROM
                recordable r
                JOIN
                        vulnerable v
                ON
                        v.vulnerable_id = r.id
        WHERE v.vulnerable_type = vulnerable_type_;
END//
DELIMITER ;
```

Calling procedure

```
mysql> CALL get_learned_spells('player');
+--------------------+---------------------------+
| VULNERABLE NAME    | AMOUNT OF LEARNED SPELLS  |
+--------------------+---------------------------+
| King Arthur        |                         0 |
| Kendran Eliorath   |                         2 |
| Judith Warrs       |                         0 |
| Dimmus Borgirs     |                         0 |
| Bendran Helliot    |                         1 |
| Murdoc Mantova     |                         0 |
| Pitter Tarazz      |                         0 |
| Julius Fandoble    |                         1 |
| Almighty Fernandus |                         1 |
+--------------------+---------------------------+
9 rows in set (0.07 sec)
```

Taras Boreyki

@TODO - On the way

Stored function

This function is used inside the *insert_taxonomy* stored procedure and it is used to retrieve the id of a recordable based on its name. See the screenshot of the declaration below.

```
##CREATION OF A FUNCTION THAT MAPS AN ITEM NAME TO ITS ID
DROP FUNCTION IF EXISTS get_id;
DELIMITER //
CREATE FUNCTION get_id(name_ TEXT) RETURNS int READS sql data
BEGIN
    RETURN (SELECT id FROM recordable WHERE name = name_);
END //
DELIMITER ;
```

Trigger

@TODO - I still need to do those, team. I will keep you posted when I’ve finished those. This is just one per group, just like the stored function above

Part B - Query Optimization

Correlated subquery multi-table query examples - This query runs through the “items” table and shows which is consumable and which is not:

```
135  SELECT
136      s.id,
137      r.name,
138      IF((SELECT s.subgroup FROM see_consumables sc WHERE sc.id = s.id) is null, "No", "Yes") 'is cons
139
140  FROM
141      see_items s
142  JOIN
143      recordable r
144      ON
145          r.id = s.id;
146
147
148
```

100% 15:145

Result Grid Filter Rows: Search Export:

ID	name	is consumable
10	Royal Helment	No
11	Elven Legs	No
12	Medusa Shield	No
13	Crown Armor	No
14	Demon Shield	No
15	Fire Sword	No
16	Sword	No
17	Knight Axe	No
18	Giant Sword	No
19	War Hamer	No
20	Health Potion	Yes
21	Strong Health...	Yes
22	Mana Potion	Yes
23	Flask of Poison	Yes

Issuing the EXPLAIN command

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	i	NULL	ALL	PRIMARY	NULL	NULL	NULL	16	100.00	NULL
1	PRIMARY	r	NULL	ref	PRIMARY	PRIMARY	4	sallyforth.i.item_id	1	100.00	Using index
1	PRIMARY	r	NULL	ref	PRIMARY	PRIMARY	4	sallyforth.i.item_id	1	100.00	Using index
2	DEPENDENT SUBQU...	r	NULL	ref	PRIMARY	PRIMARY	4	func	1	100.00	Using where; Using index
2	DEPENDENT SUBQU...	h	NULL	eq_ref	PRIMARY	PRIMARY	4	sallyforth.r.id	1	100.00	Using index
2	DEPENDENT SUBQU...	i	NULL	eq_ref	PRIMARY	PRIMARY	4	sallyforth.r.id	1	100.00	Using index

No correlated subquery multi-table query - It display the total of consumable items out of the quantity of the items.

```
147    ## NO correlated subquery
148    #Qty of consumables out of the items amount.
149    SELECT
150        count(s.id)'Qty of items',
151        (SELECT count(sc.id) FROM see_consumables sc) 'Qty of consumables from items'
152
153    FROM
154        see_items s;
155
```

100% 5:152

Result Grid Filter Rows: Search Export:

Qty of items	Qty of consumables from items
16	6

Issuing the EXPLAIN query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	i	NULL	index	PRIMARY	PRIMARY	4	NULL	16	100.00	Using index
1	PRIMARY	r	NULL	ref	PRIMARY	PRIMARY	4	sallyforth.i.item_id	1	100.00	Using index
2	SUBQUERY	h	NULL	index	PRIMARY	PRIMARY	4	NULL	6	100.00	Using index
2	SUBQUERY	i	NULL	eq_ref	PRIMARY	PRIMARY	4	sallyforth.h.item_id	1	100.00	Using index
2	SUBQUERY	r	NULL	ref	PRIMARY	PRIMARY	4	sallyforth.h.item_id	1	100.00	Using index