EER-diagram to SQL

Assignment #3 - Report

Duy Dinh & Özgun Mirtchev

12/2/2014

Table of Contents

INTRODUCTION

For this assignment the students were asked to translate an EER-diagram into SQL queries to be used in SQL and DB Visualizer, to make their own database in the university's local database server. The way this was executed, was by using books and sites to help and translate the entities, relations and the attributes into the correct queries in SQL.

Throughout this report there will two sections displaying queries, codes and diagrams for each of the table that is created and modified, for the reader to be able to grasp every detail of the work.

CREATING TABLES

SERVER

The first entity SERVER was translated from the ERR-diagram to be its own table in SQL. The unique attribute in this case is the IP-address. To make it unique in SQL, the primary key command was used as seen in the code below:

```
CREATE TABLE SERVER(
IPADRESS VARCHAR(10),
PRIMARY KEY(IPADRESS));
```

ROOM

Since the room is an entity with several relations, it was translated into its own table as well. In this case the foreign key HAS_IN_ID was used to link to the primary key of SERVER (IPADRESS) and ROOM (Important thing to note here was that both keys had to have the same datatype, in this case varying character):

```
CREATE TABLE ROOM(
ID INT, NOT NULL
NAME VARCHAR(20),
DESCRIPTION VARCHAR(250),
HAS_IN_ID VARCHAR(20),
PRIMARY KEY(ID),
FOREIGN KEY (HAS_IN_ID) REFERENCES SERVER);
```

ITEM

The entity ITEM has a special attribute, in which that it is reflexive. This means that it has a relation-pointer which goes back to itself. To be able to demonstrate this in the database program, the problem was easily solved by making a foreign key, PUT_IN_ID, and make it reference to the already existing primary key ID. By doing this the entity got a relation which goes back to itself.

Another thing which makes this entity a bit special is that it has three foreign keys, including the one that is reflexive.

To create this table the code below was entered:

```
CREATE TABLE ITEM(
ID INT  NOT NULL,
DESCRIPTION VARCHAR(250),
NAME VARCHAR(20),
PUT_IN_ID INT,
HAS_IN_ID INT,
PRIMARY KEY(ID),
FOREIGN KEY(PUT_IN_ID) REFERENCES ITEM,
FOREIGN KEY(HAS_IN) REFERENCES ROOM,
FOREIGN KEY(HAS_IN_ID) REFERENCES CHARACT);
```

CHARACTER

The entity for CHARACT (Character) was also translated to its own table in this case. All the necessary attributes was added as columns and the primary key was assigned to ID. The foreign key was assigned to a new column HAS_IN_ID which references to the ROOM to connect them to each other:

```
CREATE TABLE CHARACT(
ID INT  NOT NULL,
DESCRIPTION VARCHAR(250),
NAME VARCHAR(20),
SKILL VARCHAR(20),
STRENGTH INT,
HEALTH INT,
POINTS INT,
HAS_IN_ID INT,
PRIMARY KEY (ID),
FOREIGN KEY (HAS_IN_ID) REFERENCES ROOM);
```

For the character in a specific program to be able to store its information regardless of if it's in the room or not, it needs to be connected to the server. This also enables the character to move between different servers if it so wishes, within a specific program utilizing this database.

To connect CHARACT with SERVER, the queries below was entered:

A column was first added to CHARACT to be the foreign key:

```
ALTER TABLE CHARACT ADD HAS_IN VARCHAR(10);
```

The column was made into a foreign key by this query:

```
ALTER TABLE CHARACT ADD CONSTRAINT "CHARACT TO SERVER" FOREIGN KEY
(HAS_IN) REFERENCES SERVER(IPADRESS);
```

Now CHARACT is connected to SERVER.

PLAYER & MONSTER (SUBCLASSES)

PLAYER and MONSTER are subclasses to the entity CHARACT. The way this is solved and displayed in SQL is to make foreign keys for each of them, by adding a new column, and connect them to the primary key of the super class (CHARACT). The code below demonstrates this:

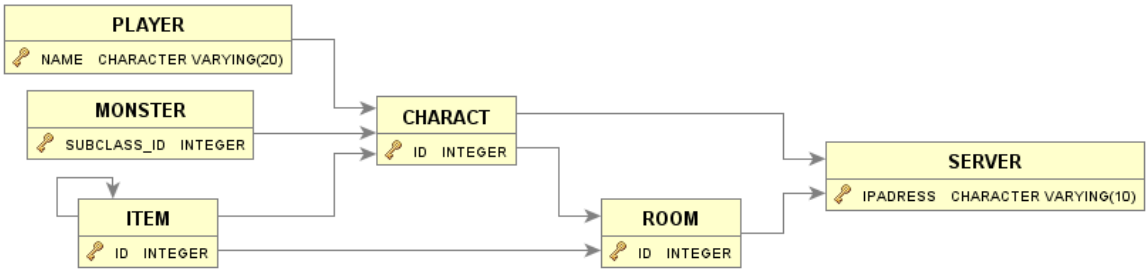
```
CREATE TABLE PLAYER(
NAME VARCHAR(20),
SUBCLASS_ID INT,
PRIMARY KEY (NAME),
FOREIGN KEY (SUBCLASS_ID) REFERENCES CHARACT);

CREATE TABLE MONSTER(
SUBCLASS_ID INT,
PRIMARY KEY(SUBCLASS_ID));
```

As the reader might have noticed, the codes that have been written have all had their key implementation in the create table command. For variation the key for the monster is going to be added through the ALTER TABLE-command:

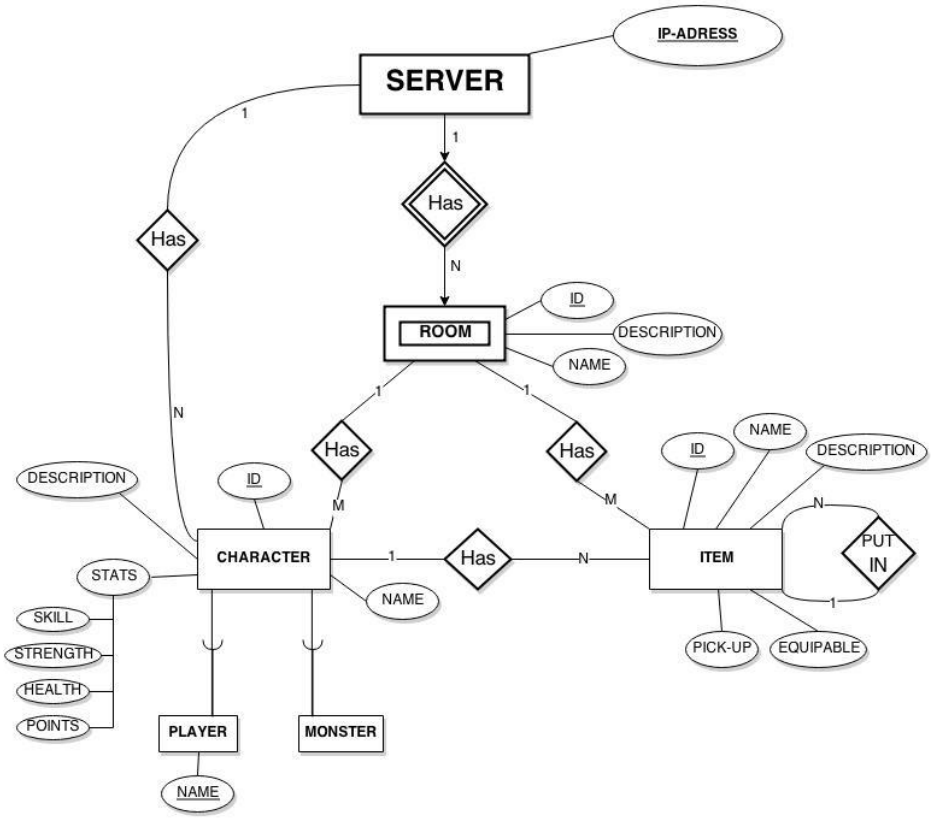
```
   ALTER TABLE MONSTER ADD CONSTRAINT MONSTER TO CHARACT FOREIGN
KEY(SUBCLASS_ID) REFERENCES CHARACT;

   ALTER TABLE MONSTER ADD CONSTRAINT FK_MONSTER_CHARACT FOREIGN KEY
(SUBCLASS_ID) REFERENCES CHARACT (ID);
```

There are two types of queries above this text, which implements a constraint to the table MONSTER. The name of the first constraint was given the name MONSTER TO CHARACT and the second constraint was given the name FK_MONSTER_CHARACT, to make the connections easier to understand. Both of the codes works the same way but the second of them is more specific and more user-friendly. For this reason, only the last query was added in the database.

The reference table now looks like this (picture below):



As it's seen, the relations are correctly displayed if compared to the SQL queries and the original EER-diagram, which can be found beneath this text:



INSERT-DATA

The reader should keep in mind that these are only simulation values and that some values might not be applicable to the real data world.

To insert values, the query `INTO`, was used. For each of the tables, appropriate values were inserted. It was also important to keep in mind that some values had to match the primary key values of the foreign key. If this was not done, the values would be false and the values wouldn't match according to the relations. The queries used will be listed here:

SERVER-INSERT

```
INSERT INTO SERVER VALUES ('150.6.8.92');
INSERT INTO SERVER VALUES ('150.6.8.91');
INSERT INTO SERVER VALUES ('150.6.8.93');
INSERT INTO SERVER VALUES ('150.6.8.94');
INSERT INTO SERVER VALUES ('150.6.8.95');
```

ROOM-INSERT

```
INSERT INTO ROOM VALUES (1, 'ROOM 1', NULL, '150.6.8.91');
INSERT INTO ROOM VALUES (2, 'ROOM 2', NULL, '150.6.8.91');
INSERT INTO ROOM VALUES (3, 'ROOM 3', NULL, '150.6.8.91');
INSERT INTO ROOM VALUES (4, 'ROOM 4', NULL, '150.6.8.94');
INSERT INTO ROOM VALUES (5, 'EMPTY ROOM', NULL, '150.6.8.95');
```

CHARACT-INSERT

```
INSERT INTO CHARACT VALUES(1, NULL, 'Haris Pilton', 'Skill#7', 1, 7, 30,
2, '150.6.8.91');
INSERT INTO CHARACT VALUES(2, NULL, 'Harrison Jones', 'Skill#8', 3, 5,
50, 5, '150.6.8.95');
INSERT INTO CHARACT VALUES(3, NULL, 'Ragnaros', 'Skill#3', 8, 8, 80, 4,
'150.6.8.94');
INSERT INTO CHARACT VALUES(4, NULL, 'Alexstraza', 'Skill#6', 8, 8, 90,
3, '150.6.8.91');
INSERT INTO CHARACT VALUES(5, NULL, 'Noob8965', 'Skill#5', 1, 4, 10, 5,
'150.6.8.95');
```

PLAYER: Noob8965
MONSTER: Ragnaros, Harrison Jones, Haris Pilton, Alexstraza

ITEM-INSERT

```
INSERT INTO ITEM VALUES (1, NULL, 'Axe', NULL, 3, 4);
INSERT INTO ITEM VALUES (2, NULL, 'Sword', NULL, 2, 5);
INSERT INTO ITEM VALUES (3, NULL, 'Bag', NULL, 1, 2);
INSERT INTO ITEM VALUES (4, NULL, 'Potion', NULL, 5, 5);
INSERT INTO ITEM VALUES (5, NULL, 'Food', NULL, 5, 5);
```

Last words

This is generally the work that is needed to create a very small basic database. From this assignment a lot of valuable knowledge was acquired for future database assignments as well as projects in SQL. Thank you for this assignment.

ATTACHMENTS

VALUES OF TABLES

SERVER

IP-ADRESS
150.6.8.91
150.6.8.92
150.6.8.93
150.6.8.94
150.6.8.95

ROOM

ID	NAME	DESCRIPTION	HAS_IN_ID
1	ROOM 1		150.6.8.91
2	ROOM 2		150.6.8.91
3	ROOM 3		150.6.8.91
4	ROOM 4		150.6.8.94
5	EMPTY ROOM		150.6.8.95

CHARACT

ID	NAME	DESCRIPTION	SKILL	STRENGTH	HEALTH	POINTS	HAS_IN_ID	HAS_IN
1	Haris Pilton		Skill#7	1	7	30	2	150.6.8.91
2	Harrison Jones		Skill#8	3	5	50	5	150.6.8.95
3	Ragnaros		Skill#3	8	8	80	4	150.6.8.94
4	Alexstraza		Skill#6	8	8	90	3	150.6.8.91
5	Noob8965		Skill#5	1	4	10	5	150.6.8.95

PLAYER

NAME	SUB_CLASS_ID
Noob8965	5
Haris Pilton	1
Harrison Jones	2

MONSTER

SUB_CLASS_ID
4
3

ITEM

ID	NAME	DESCRIPTION	PUT_IN_ID	HAS_IN_ID	HAS_IN_ROOM
1	Axe			3	4
2	Sword			2	5
3	Bag			1	2
4	Potion			5	5
5	Food			5	5

ERR-DIAGRAM - ASSIGNMENT 2 (ORIGINAL SIZE)