Report

Assignment 3.

Author: Ahmad ABDILRAHIM
Author: Cristian BABES
code: aa224fn
code: cb223ai
Semester: HT2019
Area: Computer Science
Course code: 2DV513

Contents

1 Task 1

The system we will build is a PC building website. Users are gamers that want to showcase their computers. They will have the option to select from different components. We also planned performance scores, voting and so on, however these features have most of the implementation in the programming side rather than the database so we scrapped them since they would add complications that have nothing to do with the course.

2 Task 2

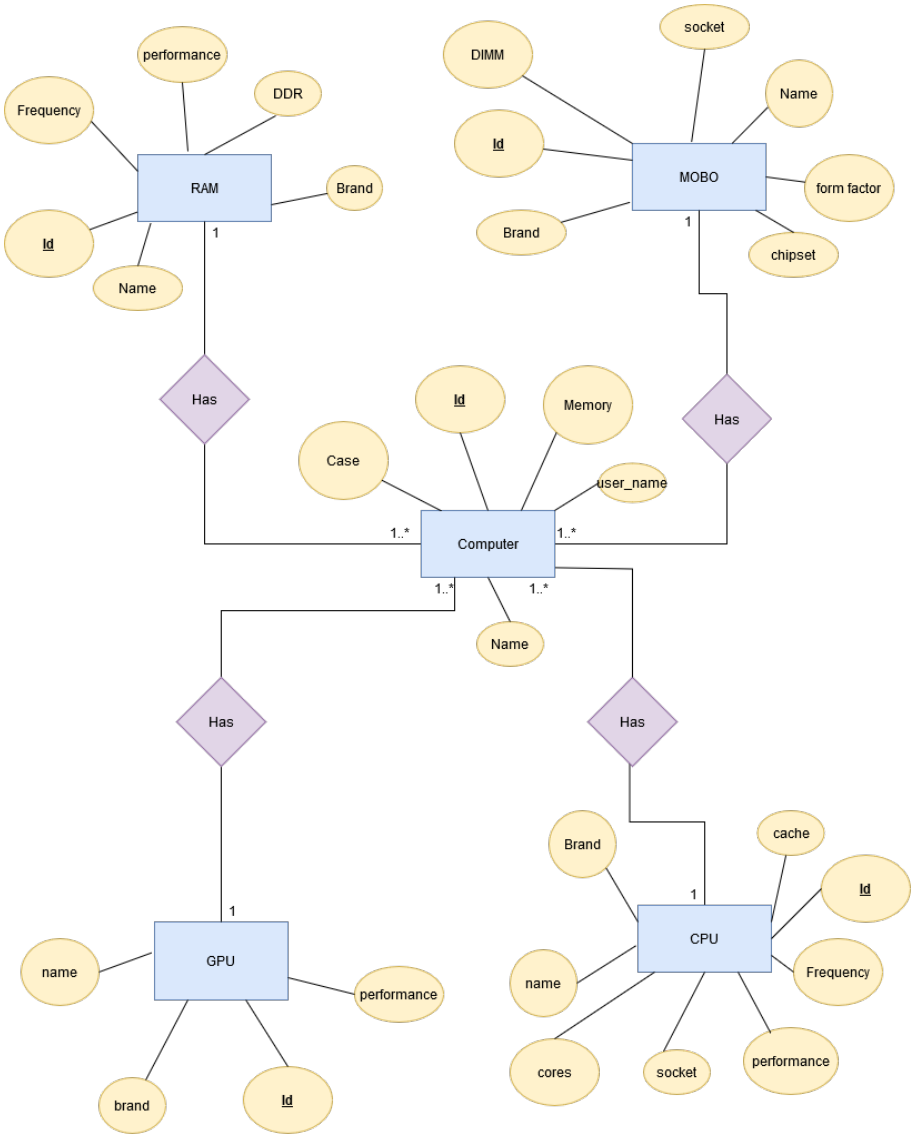

Figure 1: E/R Diagram of the computer build system

We chose to go with 5 tables. One central table which represents a computer which holds the information about it. The CPU, GPU, RAM and motherboard are held as IDs that link to their respective tables. We know there are more components in an computer, however adding the other components would add just repetition instead of any helpful situations.

Bellow you can see how the tables look:

This table represents a computer. The green keys are the foreign keys that point towards the other tables.

#	Name	Datatype	Length/Set	Unsigned	Allow NULL	Zerofill	Default	Co
1	**id**	**INT**	**11**	☐	☐	☐	**AUTO_INCREMENT**	
2	name	VARCHAR	50	☐	☐	☐	No default	
3	brand	VARCHAR	50	☐	☐	☐	No default	
4	cache	INT	11	☐	☐	☐	No default	
5	frequency	VARCHAR	50	☐	☐	☐	No default	
6	performance	FLOAT		☐	☐	☐	No default	
7	socket	VARCHAR	50	☐	☐	☐	No default	
8	cores	INT	11	☐	☐	☐	No default	

This table represents the processors.

#	Name	Datatype	Length/Set	Unsigned	Allow NULL	Zerofill	Default
1	**id**	**INT**	**11**	☐	☐	☐	**AUTO_INCREMENT**
2	name	VARCHAR	50	☐	☐	☐	No default
3	brand	VARCHAR	50	☐	☐	☐	No default
4	performance	FLOAT		☐	☐	☐	No default

This table holds the graphics card.

#	Name	Datatype	Length/Set	Unsigned	Allow NULL	Zerofill	Default
1	**id**	**INT**	**11**	☐	☐	☐	**AUTO_INCREMENT**
2	name	VARCHAR	50	☐	☐	☐	No default
3	brand	VARCHAR	50	☐	☐	☐	No default
4	socket	VARCHAR	50	☐	☐	☐	No default
5	formFactor	VARCHAR	50	☐	☐	☐	No default
6	chipset	VARCHAR	50	☐	☐	☐	No default
7	DIMM	VARCHAR	50	☐	☐	☐	No default

This table is the motherboard table.

#	Name	Datatype	Length/Set	Unsigned	Allow NULL	Zerofill	Default	C
1	**id**	**INT**	**11**	☐	☐	☐	**AUTO_INCREMENT**	
2	name	VARCHAR	50	☐	☐	☐	No default	
3	brand	VARCHAR	50	☐	☐	☐	No default	
4	DDR	INT	11	☐	☐	☐	No default	
5	frequency	CHAR	50	☐	☐	☐	No default	
6	performance	FLOAT		☐	☐	☐	No default	

This table holds the RAM.

NOTE: a read key means unique and an orange key represents the primary key.

3 Task 3

Instead of deciding on the queries from the start, we took a more top-down approach to the implementation. So we designed the front-end, the back-end and then we added the queries as we needed them. Thus, apart from the queries to insert in the database we used all of them. In total there are around 15, 3 Queries that get data from 2 tables, 1 query which uses aggregate and grouping, 2 queries which create view and use join. All the other queries are simple INSERT or SELECT queries. However please note that the database holds more information than what we used in the front end and in the queries.

4 Task 4

We will only cover the more interesting queries and get all, get one and create for CPU. Motherboard, GPU and RAM tables are not covered since the have the same queries as CPU, just a different set of columns.

```
const string = 'CREATE VIEW computers AS ' +
  'SELECT computer.name, computer.user_name, computer.case, computer.memory, cpu.name AS cpu_name, ' +
  'gpu.name AS gpu_name, ram.name AS ram_name, mobo.name AS mobo_name ' +
  'FROM computer INNER JOIN ' +
  'cpu ON computer.cpu_id = cpu.id INNER JOIN ' +
  'gpu ON computer.gpu_id = gpu.id INNER JOIN ' +
  'ram ON computer.ram_id = ram.id INNER JOIN ' +
  'mobo ON computer.mobo_id = mobo.id; ' +
  'SELECT * FROM computers; ' +
  'DROP VIEW computers'
return db.sequelize.query(string)
```

This query creates a virtual new table (a view) very similar with computer, however, using JOIN, it replaces the cpu_id with the cpu name and does the same for gpu, ram and motherboard (mobo). Then it selects all the computers in the newly created view.

```
  const string = 'CREATE VIEW computersUsers AS ' +
'SELECT computer.name, computer.user_name, computer.case, computer.memory, cpu.name AS cpu_name, ' +
'gpu.name AS gpu_name, ram.name AS ram_name, mobo.name AS mobo_name ' +    "mobo": Unknown word.
'FROM computer INNER JOIN ' +
'cpu ON computer.cpu_id = cpu.id INNER JOIN ' +
'gpu ON computer.gpu_id = gpu.id INNER JOIN ' +      You, a day ago • fixes
'ram ON computer.ram_id = ram.id INNER JOIN ' +
'mobo ON computer.mobo_id = mobo.id; ' +    "mobo": Unknown word.
'SELECT * FROM computersUsers ' +
`WHERE user_name = '${data}'; ` +
'DROP VIEW computersUsers'
  // console.log('data', data)
  return db.sequelize.query(string)
```

Very similar with the query above, however instead of selecting all of the computers, it only selects the computers that have the owner (user_name) equal with the given value

```
const string = 'CREATE VIEW computers AS ' +
  'SELECT computer.id AS id, gpu.brand AS gpu_brand ' +
  'FROM computer INNER JOIN ' +
  'gpu ON computer.gpu_id = gpu.id; ' +
  'SELECT COUNT(id) as number, gpu_brand FROM computers GROUP BY gpu_brand; ' +
  'DROP VIEW computers'
```

This query creates a new view with the id of the computers and the gpu brands, Then using COUNT and GROUP BY it counts how many computers there are using each GPU brand.

```
return db.sequelize.query(`SELECT * FROM cpu where name = '${data}'`, { type: db.sequelize.QueryTypes.SELECT })
```

This query selects all the processors which have a certain name. However since the name is unique it will only return one.

```
return db.sequelize.query('SELECT * FROM cpu', { type: db.sequelize.QueryTypes.SELECT })
```

This query selects all processors.

```
return db.connection.query('INSERT INTO cpu set ?', data, function (err, results)
```

This query inserts the data into the database, This query doesn't show the column names and the values because the library uses a model of what the cpu table looks to completes the query with the table columns and the values we insert.

5 Task 5

The source code is in the .zip file. Run it like any nodejs application using the following two scripts: npm install then npm start. Then in the web browser navigate to localhost:4000. The front-end is already build and prepared in the ExpressJS application, however if you wanna look at the source code for the front-end check the github repository bellow. The code lies under the client folder.
`https://github.com/cristianmtb/2dv513-Assignment-3`.

We have connected the database on DigitalOcean which is a remote server hosting service for ease of access(the database is already connected to it and by running NPM start it should connect automatically) and Instead of providing a database dump you can access the database through an application by connecting to the remote server using the following:
host: '139.59.137.181'
user: 'dbadmin',
password: 'admin',
database: 'computer',

6 Task 6

Here is the link to our video which is uploaded on Youtube:
`https://youtu.be/ZUvLG3jrPzc`.