SQL and Map-Reduce-Filter

We'll see how the map, reduce and filter methods prove useful in SQL queries.

Exercise:

Suppose the following tables are given in the form of arrays of objects:

```
var inventory = [
    {
        id: 1,
        owner: 1,
        name: 'Sword',
        weight: 10,
        value: 100
    },
    {
        id: 2,
        owner: 1,
        name: 'Shield',
        weight: 20,
        value: 50
    },
    {
        id: 3,
        owner: 2,
        name: 'Sword',
        weight: 9,
        value: 150
    }
];

var characters = [
    {
        id: 1,
        name: 'Zsolt',
        points: 500,
        level: 5
    },
    {
        id: 2,
        name: 'Ron',
        points: 200,
        level: 2
    },
    {
        id: 3,
        name: 'Jack',
        points: 0,
            level: 1
        }
]


console.log(characters.map( c => c.name ));
```

Translate the following SQL query using `map`, `reduce`, and `filter`:

```
SELECT characters.name, SUM( inventory.value ) AS totalValue
FROM characters, inventory
WHERE characters.id = inventory.owner
GROUP BY characters.name
```

You are not allowed to use loops, if statements, logical operators, or the ternary operator.

Solution:

We need to list all character names and the sum of the value of their items.

As `characters` and `inventory` are arrays, we can use the `map`, `reduce`, and `filter`.

The following projection corresponds to a simple `map` operation:

```
SELECT name
FROM characters
```

The next step is to join the `characters` and the `inventory` tables by filtering items belonging to a given character. We can also return the number of items belonging to a character in this step.

```
SELECT characters.name, COUNT( inventory.value ) AS itemCount
FROM characters, inventory
WHERE characters.id = inventory.owner
GROUP BY characters.name
```

We have to return the name like before, but we also have to find the corresponding items.

```
console.table(
    characters.map( c => {
        let items = inventory.filter( item => item.owner === c.id );
        return {
            name: c.name,
            itemCount: items.length
        }
    } )
);
```

If you execute this code in the console, you can see that all three characters are there with their correct item count:

(index)	name	itemCount
0	“Zsolt”	2
1	“Ron”	1
2	“Jack”	0

We only have one step left: summing the value of the items.

```
SELECT characters.name, SUM( inventory.value ) AS totalValue
FROM characters, inventory
WHERE characters.id = inventory.owner
GROUP BY characters.name
```

The `SUM` function is implemented by a `reduce` call in JavaScript:

```
console.table(
    characters.map( c => {
        let totalValue =
            inventory
                .filter( item => item.owner === c.id )
                .reduce( (sum, item) => item.value + sum, 0 );
        return {
            name: c.name,
            totalValue
        }
    } )
);
```

The result is:

(index)	name	itemCount
0	“Zsolt”	150
1	“Ron”	150
2	“Jack”	0

During the task, it is worth looking up the signature of `reduce` if you forgot it. Alternatively, you can reverse engineer it using a simple example if you get stuck and your interviewers insist in you not opening other tabs.

Sometimes you might have to use a testing platform to submit an answer, and the software might monitor when you leave a tab.

This is a stupid way of evaluating candidates because it puts you in a big advantage if you have two computers side-by-side. You can execute any google searches you want on one computer, and solve the task on another. This is one reason why I believe all tests should be open book tests. If you want to browse the Internet, go for it! You have limited time, so you won’t be able to learn the basics of JavaScript while solving a task anyway.

This exercise is solved. Notice the analogy between SQL SELECT statements and `map`, `reduce`, `filter`. These are common questions not only in JavaScript.