**Bamazon Homework Assignment**

Create 2 files in Visual code:

1. bamazon.sql
2. bamazonCustomer.js



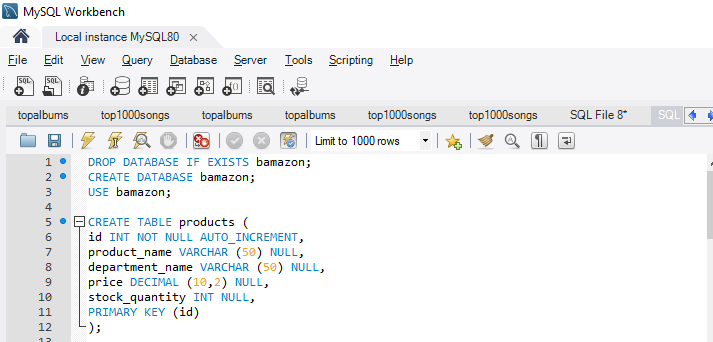
**Part I- Creating the SQL database & Table**

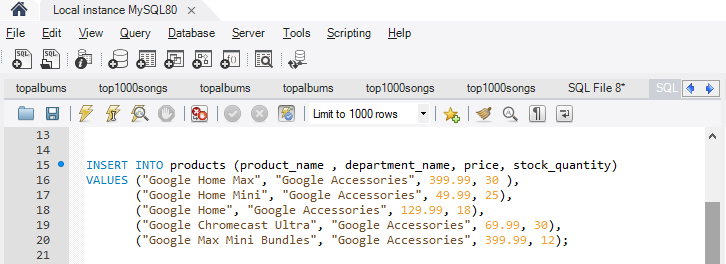


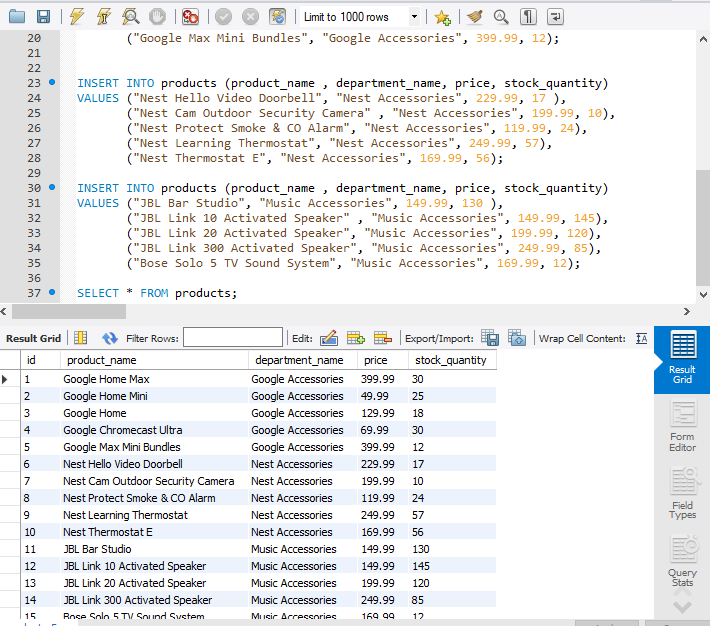
1. Create a MySQL Database called bamazon.
2. Then create a Table inside of that database called products.
3. The products table should have each of the following columns:

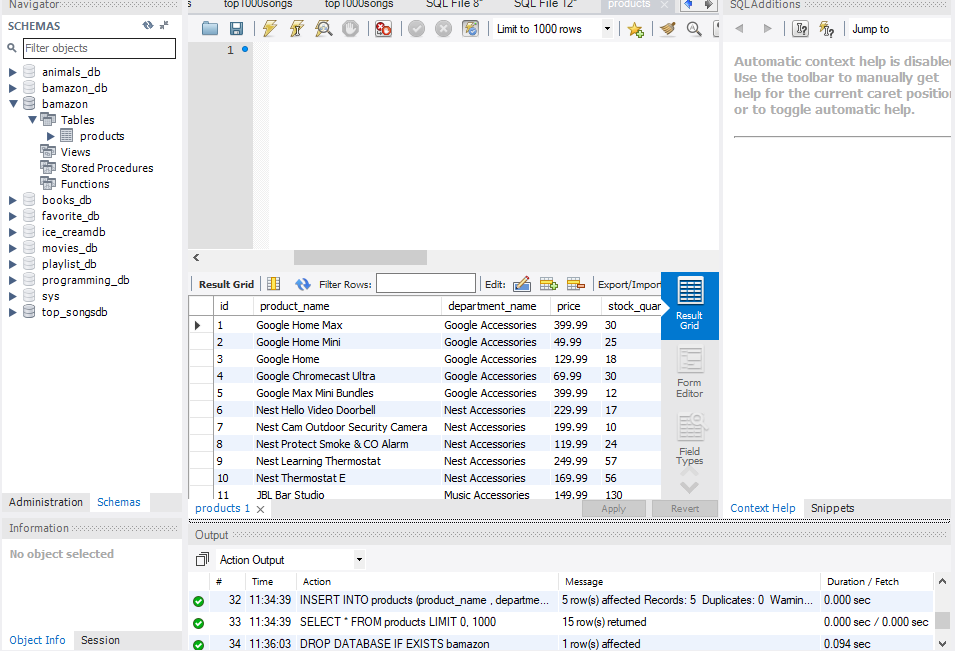
* item\_id (unique id for each product)
* product\_name (Name of product)
* department\_name
* price (cost to customer)
* stock\_quantity (how much of the product is available in stores)

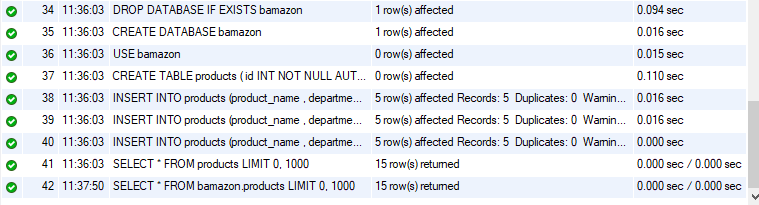
1. Populate this database with around 10 different products. (i.e. Insert "mock" data rows into this database and table).
2. Result of steps 1-5 is shown below











**SQL CODE & EXPLANATION:**



--Creating a database, ensuring the if there is already a db existing with the same name to drop it--

DROP DATABASE IF EXISTS bamazon;

CREATE DATABASE bamazon;

USE bamazon;

--Creating a table called Products inside the database--

CREATE TABLE products (

id INT NOT NULL AUTO\_INCREMENT,

product\_name VARCHAR (50) NULL,

department\_name VARCHAR (50) NULL,

price DECIMAL (10,2) NULL,

stock\_quantity INT NULL,

PRIMARY KEY (id)

);

--Populating values inside the table--

--1st category used Google Accessories--

INSERT INTO products (product\_name , department\_name, price, stock\_quantity)

VALUES ("Google Home Max", "Google Accessories", 399.99, 30 ),

("Google Home Mini", "Google Accessories", 49.99, 25),

("Google Home", "Google Accessories", 129.99, 18),

("Google Chromecast Ultra", "Google Accessories", 69.99, 30),

("Google Max Mini Bundles", "Google Accessories", 399.99, 12);

--2nd category used is Nest Accessories--

INSERT INTO products (product\_name , department\_name, price, stock\_quantity)

VALUES ("Nest Hello Video Doorbell", "Nest Accessories", 229.99, 17 ),

("Nest Cam Outdoor Security Camera" , "Nest Accessories", 199.99, 10),

("Nest Protect Smoke & CO Alarm", "Nest Accessories", 119.99, 24),

("Nest Learning Thermostat", "Nest Accessories", 249.99, 57),

("Nest Thermostat E", "Nest Accessories", 169.99, 56);

--3rd category used is Music Accessories--

INSERT INTO products (product\_name , department\_name, price, stock\_quantity)

VALUES ("JBL Bar Studio", "Music Accessories", 149.99, 130 ),

("JBL Link 10 Activated Speaker" , "Music Accessories", 149.99, 145),

("JBL Link 20 Activated Speaker", "Music Accessories", 199.99, 120),

("JBL Link 300 Activated Speaker", "Music Accessories", 249.99, 85),

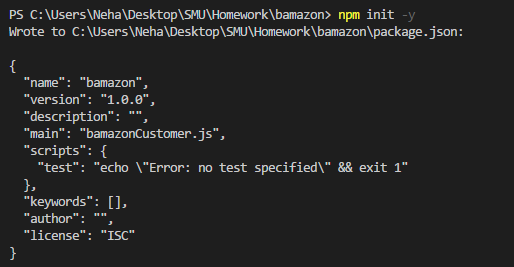
("Bose Solo 5 TV Sound System", "Music Accessories", 169.99, 12);

SELECT \* FROM products;

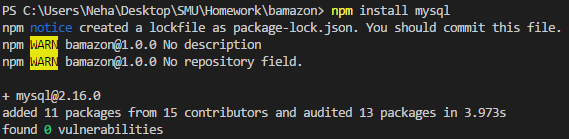
**Part II- Integration of Nodejs & SQL**



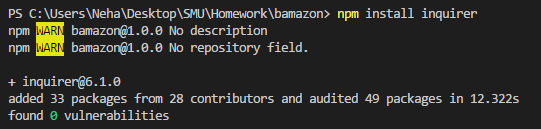
1. Setting up the javascript file
2. **npm init –y** to set up the package.json and the dependancies



1. **npm install mysql** -setting up mysql to integrate our SQL file i.e. bamazon.sql



1. **npm install inquirer** so we can use prompt commands and interact with the user



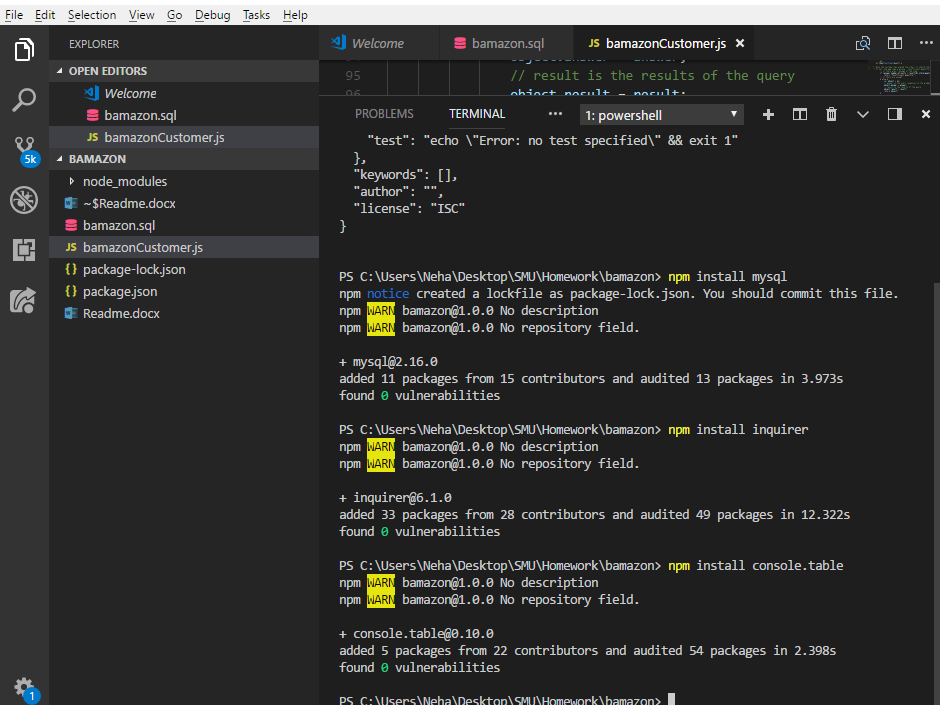
1. **npm install console.table**

You can pass multiple strings and arrays in a single console.table call. Each argument will be formatted and printed separately on new line

Example:



1. After installing the above mentioned node packages mention above visual studio bamazon folder will look similar to the screen below



**Part III- Writing Javascript**

Part 1: Needed for SQL set up

//1. require mysql grabs the mysql from wherever its installed//

var mysql = require('mysql');

Part 2: Set up the Inquirer connection for Prompts

//2. Required for prompts

var inquirer = require('inquirer');

Part 3: console.table

//3. Required for logging tables

require('console.table');

Part 4: Setting up connection with MYSQLDB

//4. Create connection between mysql and SQLdb

var connection = mysql.createConnection({

host: "localhost",

// 5. Your port; if not 3306

port: 3306,

//6. Your username

user: "root",

//7. Your password

password: "root", //DONOT push password on github

database: "bamazon"

});

Part 5:

Remaining Javascript code is in the js file on github

//1. require mysql grabs the mysql from wherever its installed//

var mysql = require('mysql');

//2. Required for prompts

var inquirer = require('inquirer');

//3. Required for logging tables

require('console.table');

//4. Create connection between mysql and SQLdb

var connection = mysql.createConnection({

host: "localhost",

// 5. Your port; if not 3306

port: 3306,

//6. Your username

user: "root",

//7. Your password

password: "", //DONOT push password on github

database: "bamazon"

});

//8. connect to mySQL database, connection should only be long enough so we can get the data

connection.connect(function(err) {

if (err) throw err;

//9. get all products

allProducts().then(function(result) {

//10. then list them

console.log("Please select a product.")

result.forEach(function(item) {

console.log('Item ID: ' + item.id + ' | Product Name: ' + item.product\_name + ' | Price: ' + item.price);

});

//11. Asking user what they would like to do

}).then(function() {

return whatWouldYouLike();

});

})

function allProducts(){

return new Promise(function(resolve, reject) {

//12. query for all items in products table

connection.query("SELECT \* FROM products", function(err, res) {

if (err) reject(err);

resolve(res);

});

});

}

//13. The app should then prompt users with two messages.

// The first prompt should ask them the ID of the product they would like to buy.

function whatWouldYouLike() {

return inquirer.prompt([{

name: 'product\_id',

message: 'What is the ID of the product you would like to buy?',

type: 'input',

validate: function(value) {

if (isNaN(value) === false) {

return true;

} else {

console.log('\nPlease enter a valid ID.');

return false;

}

}

},

//2nd message is asking the user how many units of the product they would like to buy.

{

name: 'number\_of\_units',

message: 'How many units would you like to buy?',

type: 'input',

validate: function(value) {

if (isNaN(value) === false) {

return true;

} else {

console.log('\nPlease enter a valid quantity.');

return false;

}

}

//14. Storing the result

}]).then(function(answer) {

return new Promise(function(resolve, reject) {

//15. query for all items in products table where the item\_id is what was chosen

connection.query("SELECT \* FROM products WHERE id=?", answer.product\_id, function(err, res) {

if (err) reject(err);

resolve(res);

});

}).then(function(result) {

//16. Once the customer has placed the order, it should check if the store has enough of the product to meet the customer's request.

//If quantity is not sufficient, then app logs a phrase `Insufficient quantity!`, and also prevents the order from going through.

if (answer.number\_of\_units > result[0].stock\_quantity) {

return "Insufficient quantity!";

// if there are enough

} else {

var object = {};

// answer is the users responses to the prompts

object.answer = answer;

// result is the results of the query

object.result = result;

return object;

}

}).catch(function(err) {

console.log(err);

connection.destroy();

}).then(function(object) {

//17. If there is sufficient stock then

//a. order should go throw successfully

//b. update the sql db to reflect the remaining quantity

//c. Once the order goes thru show the total cost to the user

if (object.answer) {

var newQuantity = object.result[0].stock\_quantity - object.answer.number\_of\_units;

var product = object.answer.product\_id;

var totalCost = (object.result[0].price \* object.answer.number\_of\_units).toFixed(2);

// query that updates the quantity of the item

connection.query("UPDATE products SET stock\_quantity=? WHERE id=?", [newQuantity, product], function(err, res) {

if (err) reject(err);

console.log('Your total cost is $' + totalCost);

// destroy connection

connection.destroy();

});

} else {

console.log(object);

// destroy connection

connection.destroy();

}

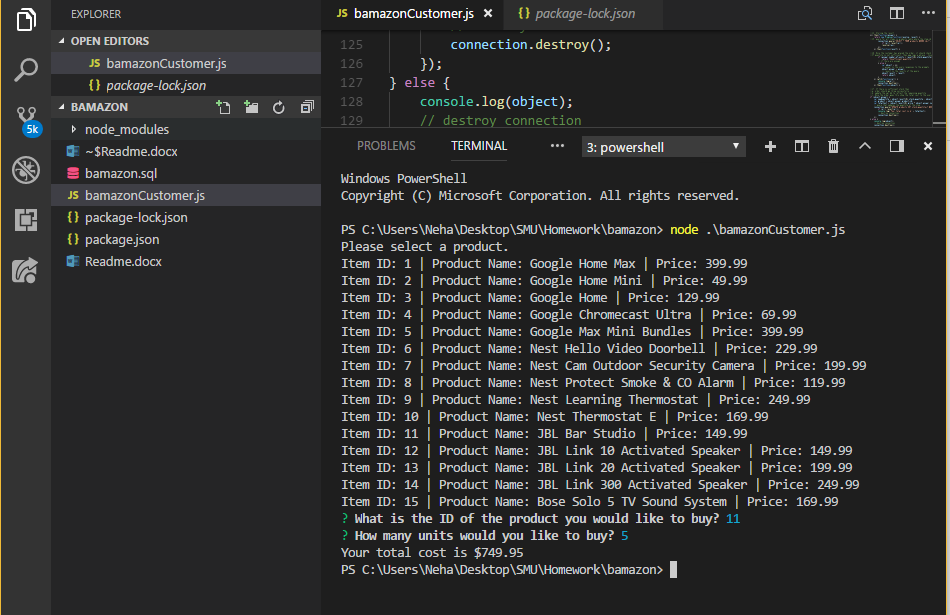
});

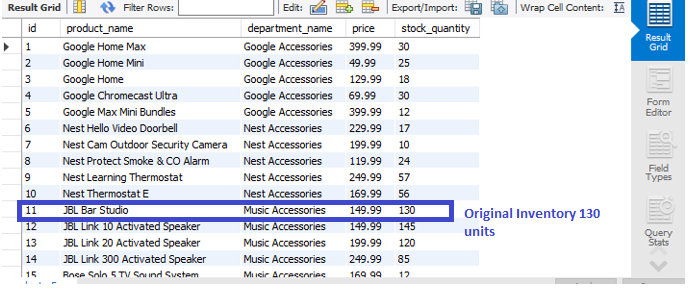
});

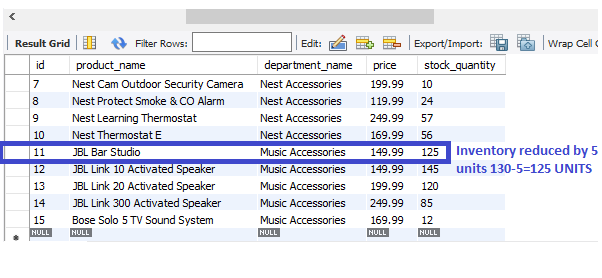
}

Part 6: Result

**Result 1 (Sufficient Quantity)-** Item 11 has a stock qty of 130 units. User input entered as 5 units therefore since there is sufficient quantity result returned the total cost which is 5\*149.99=$749.95. Also, inventory table has been updated to reduce quantity by 5 units (see table below)







**Result 2 (Insufficient quantity)-** Item 7 has a stock qty of 10 units. User input entered as 13 units therefore since there is insufficient quantity result returned as insufficient quantity

