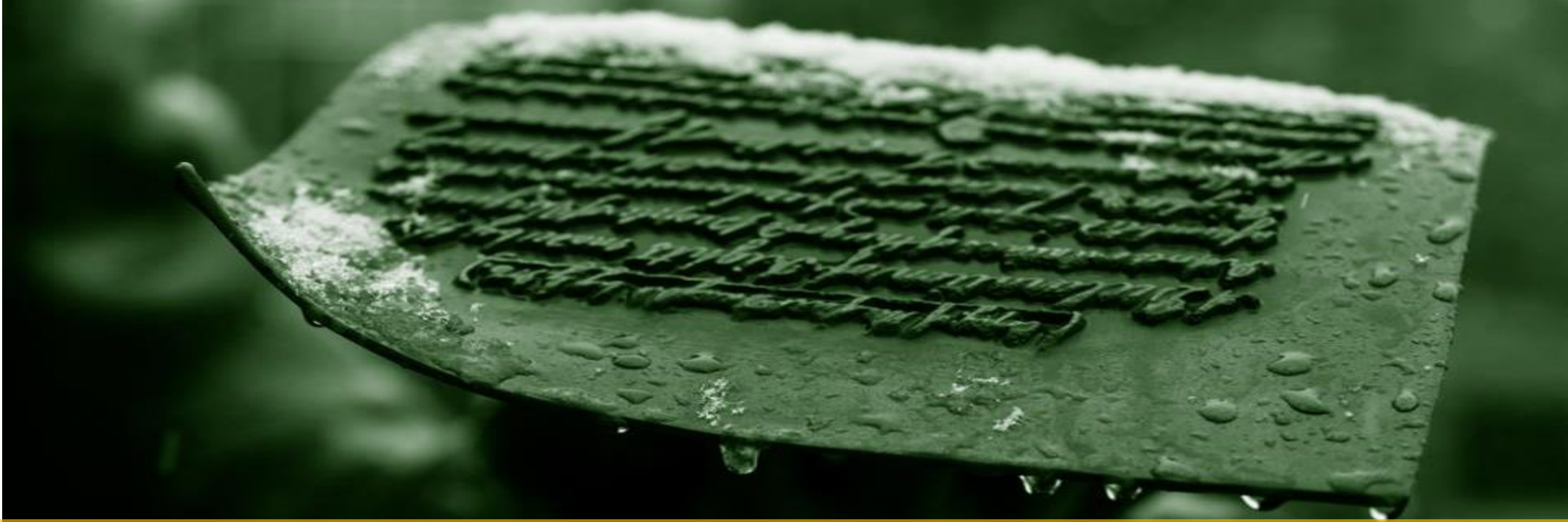


MODULE 8 – CONNECTING MYSQL SERVER TO NODEJS

IT 207 – IT Programming

GEORGE MASON UNIVERSITY

Lecture Outline

- ❖ Modules in Nodejs
- ❖ MySQL driver in Nodejs
- ❖ Parameterized Queries

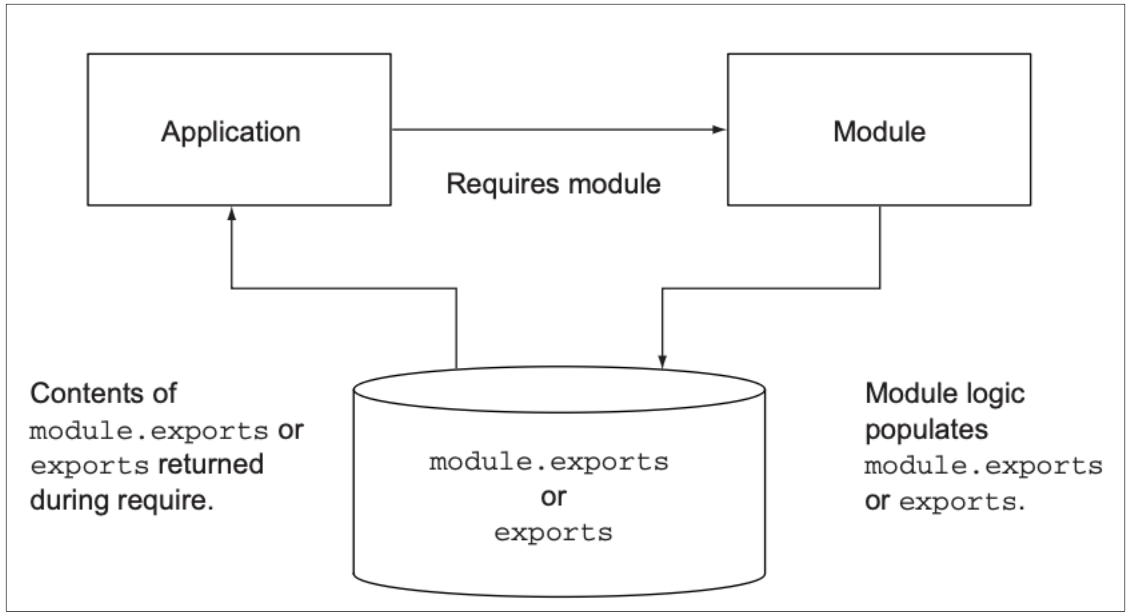
Modules in Nodejs

- Modules in Nodejs are equivalent to modules in Python or APIs in JAVA
  - JS files bundled together to provide functionalities to developers
- There are three types of Modules in Nodejs
  - Core modules
    - Built-in modules installed with Nodejs
  - Local modules
    - User defined modules
  - Third-party modules
    - Modules available online and can be added to Nodejs using the Node Package Manager(NPM), e.g., MySQL module

Advantages of NodeJS Modules

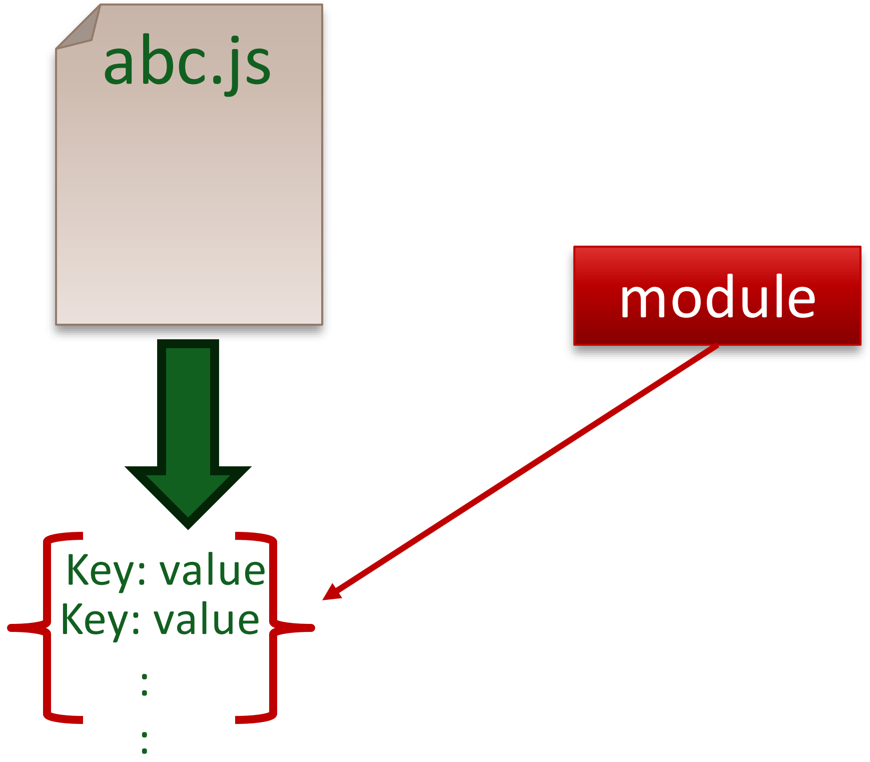
- Node modules bundle up code for reuse
  - Code implementing related logic can be grouped into separate files and imported into the application script using the require keyword
- Node modules do not pollute the global scope
  - No naming collisions
  - Developers can select which functions and/or variables to be exported into their application

Exporting functions, objects or variables is done through the *module* object


Application
Requires module
Module
Contents of module.exports or exports returned during require.
module.exports or exports
Module logic populates module.exports or exports.

MODULE OBJECT IN NODEJS

- Node.js treats each JavaScript file as a separate module and assigns to it an object that can be referenced by a variable called *module*.
- The module object has several key: value pairs
- One of its keys is called *exports* the value corresponding to this key is {} (an empty object).
- module.exports is used for defining what can be exported by a module.
  - Whatever is exported from a module can, in turn, be made available to other modules.



Example – module Object

```
% node tryme.js
Module {
  id: '.',
  path: '/Users/hhassa2/Documents/IT207/code',
  exports: {},
  filename: '/Users/hhassa2/Documents/IT207/code/tryme.js',
  loaded: false,
  children: [],
  paths: [
    '/Users/hhassa2/Documents/IT207/code/node_modules',
    '/Users/hhassa2/Documents/IT207/node_modules',
    '/Users/hhassa2/Documents/node_modules',
    '/Users/hhassa2/node_modules',
    '/Users/node_modules',
    '/node_modules'
  ]
}
```

```
/*tryme.js */

console.log(module);
```

Example - Exporting from a Module

```
/*currency.js */
//private variable - not exposed out of the module
let canadianDollar = 0.73;
//private method
function roundTwoDecimals(amount) {
return Math.round(amount * 100) / 100;
}
//Exported methods
module.exports.canadianToUS = (canadian)=> {
return roundTwoDecimals(canadian * canadianDollar);
}
module.exports.USToCanadian = function(us) {
return roundTwoDecimals(us / canadianDollar);
}
```

```
/* testCurrency.js */
const currency = require('./currency');
console.log(`50 Canadian dollars equals ${currency.canadianToUS(50)} US dollars`);
console.log(`30 US dollars equals ${currency.USToCanadian(30)} Canadian dollars:`);
```

MODULE.EXPORTS VS EXPORTS

- As stated in Nodejs documentation.

For convenience, module.exports is also accessible via the exports key

```
% node currency.js
Module {
  id: '.',
  path: '/Users/hmhassan/Main/IT207/CourseDevelopment/SternCenter/Module 8/currency-app',
  exports: {
    canadianToUS: [Function (anonymous)],
    USToCanadian: [Function (anonymous)]
  },
  ……
  ……
]
}
```

The value for the exports key in the module object for the currency application includes the exported functions

Example – Using exports

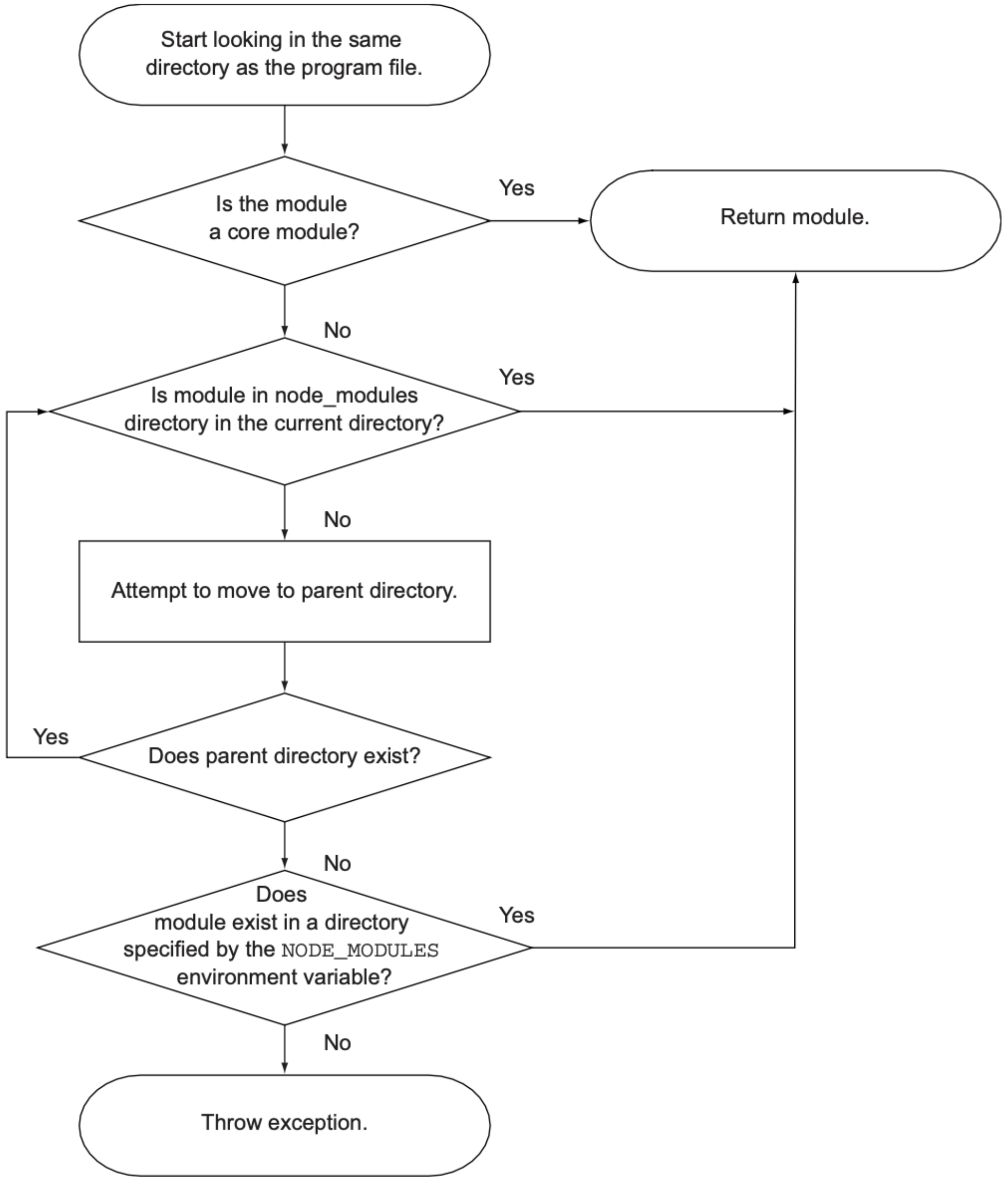
```
/*currency.js */
//private variable - not exposed out of the module
let canadianDollar = 0.73;
//private method
function roundTwoDecimals(amount) {
return Math.round(amount * 100) / 100;
}
//Exported methods
exports.canadianToUS = (canadian)=> {
return roundTwoDecimals(canadian * canadianDollar);
}
exports.USToCanadian = function(us) {
return roundTwoDecimals(us / canadianDollar);
}
```

However, there are some cases where you MUST use module.exports

```
/* testCurrency.js */
const currency = require('./currency');
console.log(`50 Canadian dollars equals ${currency.canadianToUS(50)} US dollars`);
console.log(`30 US dollars equals ${currency.USToCanadian(30)} Canadian dollars:`);
```

Locating Modules

- When adding modules to your script, their location should be specified.
- If the module is not found in the specified location, then Node will try to locate it by following along the directories listed in the *paths* property in the module object.
- Node_modules is the name of the folder where custom defined modules should be placed.


Start looking in the same directory as the program file.
Is the module a core module?
Yes
Return module.
No
Is module in node_modules directory in the current directory?
Yes
No
Attempt to move to parent directory.
Yes
Does parent directory exist?
No
Does module exist in a directory specified by the NODE_MODULES environment variable?
Yes
No
Throw exception.

Caveats when Creating Modules

- Modules can either be single files or directories containing one or more files.
- If a module is a directory, the file in the module directory that will be evaluated is normally named *index.js*.
- To specify an alternative to index.js, the *package.json* file must contain JavaScript Object Notation (JSON) data defining an object with a key named *main* that specifies the path, within the module directory, to the main file.

INSTALLING & CONNECTING TO MYSQL DRIVER

MySQL Driver

- mysql2 is a driver for connecting to MySQL database from Nodejs
  - Nodejs script will act as a client to the MySQL server
- mysql2 is installed using npm (Node Package Management)
- Steps
  1. Create a folder for your mysql project
  2. Run `npm init` to create package.json file
  3. Install mysql2 driver using the command `npm install mysql2`
  4. Write your application logic in the same folder that you have created in step 1

Connecting to MySQL Server from within Nodejs - 1

- Import mysql2 to the Nodejs script

```
//import the mysql2 module
const mysql = require('mysql2');
```

- Create a connection to the MySQL database by calling the createConnection() method and passing the connection options

```
let connection = mysql.createConnection({
host: 'localhost',
user: 'root',
password: '',
database: 'todoapp' //database has already been created on the server
});
```

  - More options can be found at https://github.com/mysqljs/mysql#connection-options
- Call the connect() method on the connection object to connect to the MySQL database server

Connecting to MySQL Server from within NodeJs - 2

- Call the connect() method on the connection object to connect to the MySQL database server

```
connection.connect((err)=>{
  if (err) {
              return console.error('error: ' + err.message);
              }
  console.log('Connected to the MySQL server.');
});
```

- Gracefully end the connection by calling the end() method on the connection object.
  - The end() method ensures that all remaining queries are executed before the database connection is closed.

```
connection.end( (err) =>{
  if (err) {
      return console.log('error:' + err.message);
   }
   console.log('Close the database connection.');});
```

Querying the MySQL Server from within NodeJS

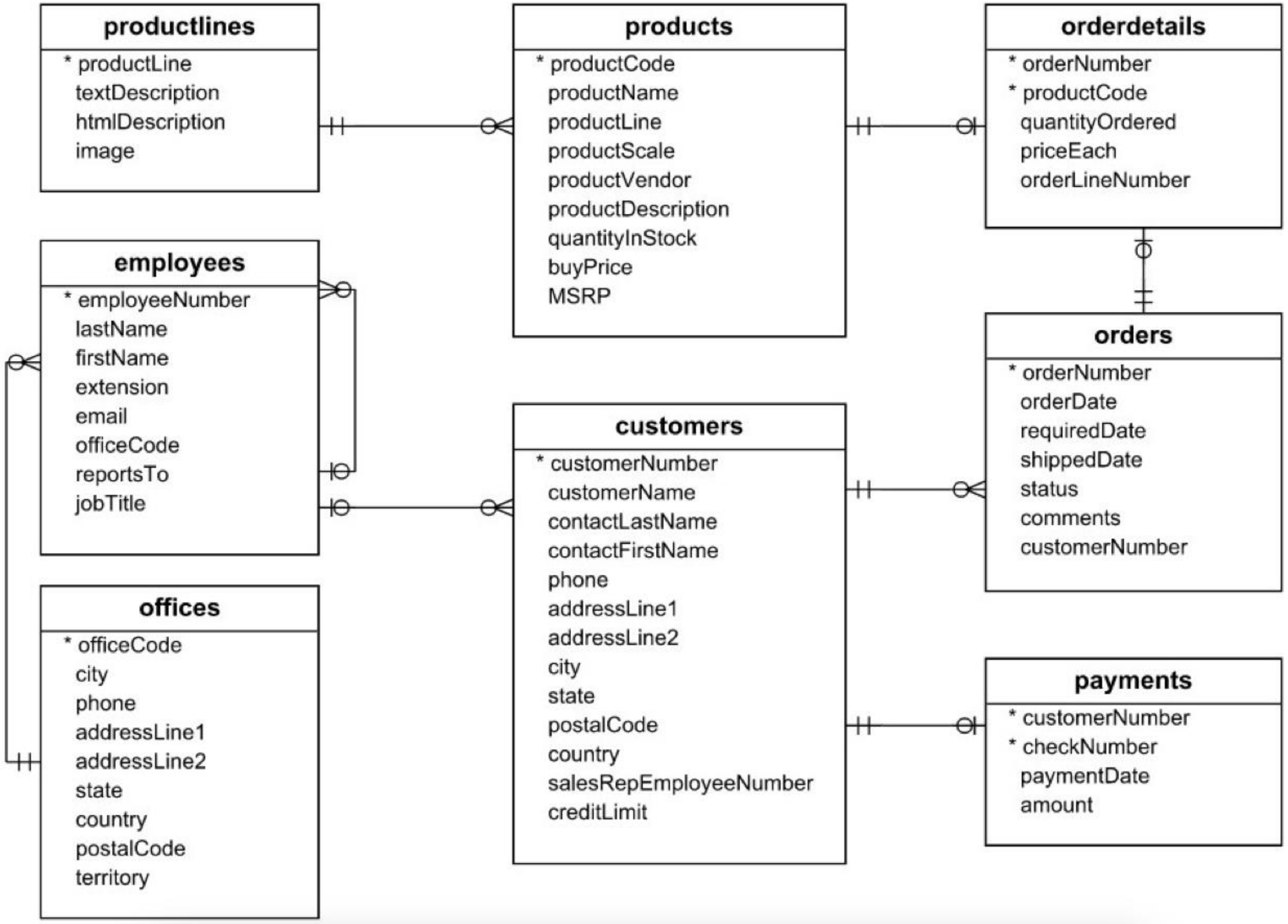
- The query() method is defined on the connection object
- It is used to send SQL statements to the MySQL server from within a NodeJS script.
- The syntax of the query method

```
connectionObject.query(sql, [values], (error, result, fields)=>{…})
```

  - sql: the SQL statement to be executed on the database
  - [values]: an array holding the values to be used in the SQL statement
  - Callback function: defines the logic to be executed on the returned results
    1. error will be an Error if one occurred during the query
    2. results will contain the results of the query
    3. fields will contain information about the returned results fields (if any)

EXAMPLE – CLASSIC MODEL SAMPLE DATABASE

- Classicmodel is a sample database found on the MySQL tutorial site



Example – Query results & fields Parameters

- Select from the employees table the first name and last name of the employees reporting to 1002

```
db.query("select firstName, lastName From classicmodels.employees where reportsTo = 1002;", (err, results, fields)=>{
                console.log(results);
      console.log(fields);
});
```

```
[
  { firstName: 'Mary', lastName: 'Patterson' },
  { firstName: 'Jeff', lastName: 'Firrelli' }
]
[ `firstName` VARCHAR(50) NOT NULL, `lastName` VARCHAR(50) NOT NULL ]
```

PARAMETERIZED QUERIES

SQL Queries

- SQL queries contain values that identify the records to be retrieved from the database.

```
SELECT OrderNumber, CustomerNumber, OrderDate
FROM Order
WHERE CustomerNumber = 382 AND OrderDate > "2004 – 07 – 16";
```

  - 382 and 2004-07-16 are values entered by the user
- Inserting the value input by the user directly in the SQL query makes the database vulnerable to SQL injection attacks.
  - An attacker can enter executable SQL code as data and get unauthorized access to a database.

SQL Parameterized Query

- ❖ Parameterized queries are SQL queries that contain parameters that can be set at runtime.

```
SELECT OrderNumber, CustomerNumber, OrderDate
FROM Order
WHERE CustomerNumber = ? AND OrderDate > ?
```

  - ❖ The query and the values are sent as parameters to the SQL server separately and will be executed separately.

```
sql = "SELECT OrderNumber, CustomerNumber, OrderDate From Orders where CustomerNumber= ?"
     + "AND OrderDate > ?";
values = ["382", "2004-07-16"];
db.query(sql,values, (err, results, fields)=>{});
```

- ❖ Parameterized queries protect against SQL Injection attacks.

SUMMARY

- Nodejs provides a systematic way for locating and exporting new modules added by developer to its codebase.
- Modules are exported using a require statement.
- Only functions/objects/variables that are exported using module.exports are exposed from Node modules.
- mysql2 is a third-party module that allows connecting to the MySQL server from Nodejs.
- Parameterized queries are SQL queries that take users' input as parameters
- Parameterized queries mitigate SQL injection attacks