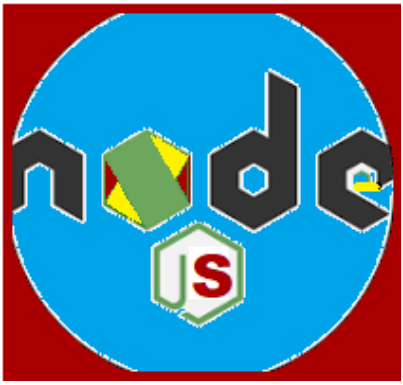
გელა ღვინეფაძე, ნინო ჩორხაული

WEB-teqnologiebi
(Node.js)



სტუ-ის „IT კონსალტინგის სამეცნიერო ცენტრი“

საქართველოს ტექნიკური უნივერსიტეტი

გელა ღვინეფაძე, ნინო ჩორხაული

Web-ტექნოლოგიები (Node.js)

დამტკიცებულია:

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრის“ სარედაქციო კოლეგიის მიერ - ოქმი N1 13.09.24

თბილისი - 2024

უაკ 004.5

აღწერილია სერვერულ გარემოში სამუშაოდ შექმნილი, Javascript ენის ბაზაზე შემუშავებული კროსპლატფორმულ პროდუქტ Node.js-ის დანიშნულება და შესაძლებლობები.

დამხმარე სახელმძღვანელო განკუთვნილია ინფორმატიკის სპეციალობის შემსწავლელი სტუდენტებისა და ამ საკითხით დაინტერესებული პირებისთვის.

რეცენზენტები:

- პროფ. თ. სუხიაშვილი – საქართველოს ტექნიკური უნივერსიტეტი,
- პროფ. თ. თოდუა – ბიზნესისა და ტექნოლოგიების უნივერსიტეტი

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), ზ. აზმაიფარაშვილი, მ. ახობაძე, გ. გოგიჩაიშვილი, ზ. ბოსიკაშვილი, ე. თურქია, რ. კაკუბავა, ვ. კვარაცხელია, თ. ლომინაძე, ნ. ლომინაძე, ჰ. მელაძე, ლ. პეტრიაშვილი, გ. სურგულაძე. ბ. შანშიაშვილი, ო. შონია, ზ. წვერაიძე

ISBN 978-9941-8-7210-5

სარჩევი

შესავალი ---------- 05
თავი 1. Node.js ---------- 07
1.1. Node.js-ის არსი ---------- 07
1.2. Node.js-ის ღირსებები ---------- 09
1.3. მოდულების თაობაზე ---------- 12
1.3.1. Node.js-ში ჩაშენებული HTTP-მოდული ---------- 12
1.3.2. მოთხოვნის სტრიქონის წაკითხვა ---------- 14
1.3.3. მოთხოვნის სტრიქონის ნაწილებად დაყოფა ---------- 15
1.4. Node.js - ფაილური სერვერი (fs მოდული) ---------- 16
1.4.1. ფაილების წაკითხვა ---------- 16
1.4.2. ფაილების შექმნა ---------- 18
1.4.3. ფაილების განახლება ---------- 20
1.4.4. ფაილების ამოგდება ---------- 21
1.4.5. ფაილისთვის სახელის გადარქმევა ---------- 21
1.4.6. ფაილების ჩატვირთვა ---------- 22
1.5. URL ჩაშენებული მოდული ---------- 22
1.6. Node.js - NPM ---------- 26
1.7. მოვლენები (Events) Node.js-ში ---------- 28
1.8. Node.js ფაილების ჩატვირთვა ---------- 30
1.9. Node.js - ელექტრონული წერილების გაგზავნა ---------- 34
თავი 2. Node.js და MySQL ---------- 36
2.1. ბაზასთან დაკავშირება ---------- 36

2.2. მოთხოვნის ფორმირება ---------- 37
2.3. მონაცემთა ბაზის შექმნა ---------- 38
2.4. ცხრილის შექმნა ---------- 39
2.5. Primary Key - ძირითადი (პირველადი) გასაღები ---------- 40
2.6. ცხრილში ჩანაწერის ჩასმა ---------- 42
2.7. ცხრილში რამდენიმე ჩანაწერის ჩასმა ---------- 43
2.8. შედეგის ამსახველი ობიექტი ---------- 45
2.9. ცხრილიდან მონაცემების ამორჩევა ---------- 46
2.10. მონაცემების ამორჩევა ფილტრაციით ---------- 53
2.11. მონაცემების დახარისხება ---------- 57
2.12. მონაცემების ამოგდება ---------- 60
2.13. ცხრილის ამოგდება ---------- 61
2.14. მონაცემების განახლება ---------- 63
2.15. სერვერიდან გადმოცემული მონაცემების ლიმიტირება --- 65
2.16. ცხრილების ერთმანეთთან მიერთება ---------- 68
2.16.1. მარცხნიდან მიერთება ---------- 69
2.16.2. მარჯვნიდან მიერთება ---------- 70
დანართი ---------- 71
ლიტერატურა ---------- 75

შესავალი

Node.js სერვერულ გარემოში უშვებს JS-ის სცენარს და წარმოადგენს JavaScript-ის ბაზაზე აგებულ მძლავრ სერვერულ კროს- ანუ მულტიპლატფორმულ პროდუქტს.

Node.js შესაძლებლობას იძლევა, რომ JavaScript ენაზე დაწერილი კოდი (სცენარი) გაშვებული იქნეს როგორც კლიენტის კომპიუტერზე ბროუზერში, ასევე მის გარეშეც და **რაც განსაკუთრებით აღსანიშნავია, - სერვერის მხარეზეც.** შესაბამისად, პროგრამისტს, რომელსაც ფრონტ-ენდ მხარისთვის ისედაც უხდება კოდის დაწერა JavaScript ენაზე, აღარ სჭირდება სერვერულ მხარეზე (ბეკ-ენდზე) ფუნქციონირებადი პროგრამის დასაწერად სხვა სერვერული ენის გამოყენება, რადგანაც Node.js-ში JavaScript ენა ამ ფუნქციასაც ითავსებს.

აქვე განვმარტოთ პროგრამების ზემოთ ხსენებული კროსპლატფორმული დამუშავების არსი:

მიდგომა გულისხმობს ისეთი კოდების დაწერას, რომელთა შესრულება შესაძლებელი იქნება რამდენიმე ოპერაციული სისტემის მიერ ამ მიზნის მისაღწევად რამდენიმე უნივერსალური ფრეიმვორკის მეშვეობით, როგორებიცაა, მაგალითად:

Flutter, React Native, Apache Cordova, Ionic და სხვ.

მათგან განსხვავებით, ე. წ. მშობლიური (native) დანართები იწერება მხოლოდ კონკრეტული ოპერაციული სისტემებისთვის, მაგალითად, iOS-სათვის Swift-ენაზე, ხოლო Android-სთვის - Kotlin-ზე, თუმცა ბოლო ხანებში მათთვისაც გახდა შესაძლებელი კროსპლატფორმულ მიდგომაზე ორიენტირებული კოდების დაწერა.

Node.js გათვალისწინებულია ინფორმაციის ინტენსიური შეტანა-გამოტანის ისეთი ვებდანართების შესაქმნელად, როგორიც

არის, მაგალითად, ნაკადური ვიდეო-საიტები, ერთფურცლოვანი და სხვა სახის კომპიუტერული პროდუქტები..

მისი პირველი ვერსია შეიმუშავა რაიან დალმა 2009 წელს.

Node.js ღია კოდის მქონე, უფასო და ფართოდ გამოყენებადი პროგრამული პროდუქტია.

მასთან მუშაობის შესწავლამდე საჭირო არის ვფლობდეთ შემდეგ ვებ-ტექნოლოგიებს:

HTML, CSS, AJAX და სხვ.

Node.js-ის ღირსებებია:

- მაღალი წარმადობა, სწრაფქმედება. განპირობებულია შეტანა-გამოტანის ოპერაციების არაბლოკირებადობით, რომელთა შესრულება ხდება ასინქრონულ რეჟიმში, მიუხედავად იმისა, რომ Node.js ოპერაციებისათვის იყენებს მათი ერთ ნაკადში მოქცევის მიდგომას. ყოველივე ეს კი ხდება კოდის გაშვებით Google Chrome ბროუზერის გარეთ გატანილი ბირთვის - JavaScript V8 ამძრავის მეშვეობით.
- ოპერაციები იმართება ხდომილობებით, რაც იძლევა ადვილად მასშტაბირებადი ქსელური დანართების შექმნის შესაძლებლობას;
- მუშაობს OS X, Microsoft Windows და Linux ოპერაციულ სისტემებში;
- ფლობს JavaScript ენაზე დაწერილი სხვადასხვა სახის მოდულებისგან შემდგარ მდიდარ ბიბლიოთეკას.

თავი 1. NODE.JS

1.1. Node.js-ის არსი

Node.js-ის არსს ამგვარად წარმოადგენენ:

Node.js = შესრულების გარემო + JavaScript-ის ბიბლიოთეკა

Node.js პროგრამული პროდუქტის ბაზაზე შექმნილი დანართების ფუნქციონირებისას მოქმედებს ასინქრონული და ხდომილობებით მართული რეჟიმი.

ავხსნათ მისი არსი:

Node.js ბაზაზე ფორმირებული სერვერი არ ელოდება, თუ როდის დააბრუნებს API (application programming interface) შედეგებს. იგი გადადის მომდევნო გამოძახებული API-ის შესრულებაზე და როდესაც მანამდე გამოძახებული API მუშაობას დაასრულებს, Node.js-ის შეტყობინებების მექანიზმით სერვერს გადაეგზავნება მის მიერ მოთხოვნილი ინფორმაცია.

ასეთი მიდგომის გამოყენების შედეგად ერთი და იგივე პროგრამა ემსახურება გაცილებით მეტ მოთხოვნას, ვიდრე ამას ახერხებენ ტრადიციული სერვერები, როგორიცაა, მაგალითად, Apache HTTP Server.

ამასთან, Node.js დანართები არ მიმართავენ მონაცემების ბუფერიზაციას, მათ მონაცემები გამოჰყავთ ეტაპობრივად.

Node.js-სთან მუშაობის შესასწავლად ჩვენი კომპიუტერი უნდა ვაქციოთ Node.js სერვერად, რის შემდეგაც შესაძლებელი იქნება სცენარებს მივმართოთ როგორც ჩვენივე კომპიუტერზე არსებული ბროუზერიდან, ასევე - გარე სივრციდანაც.

კომპიუტერზე Node.js-ის ჩამოტვირთვისათვის საჭირო ინსტრუქცია იხ. მისამართზე https://nodejs.org.

საერთოდ, Node.js სერვერულ გარემოში Javascript-ის სცენარები სრულდება არა კლიენტის კომპიუტერზე, არამედ სერვერზე. მაგრამ, რადგან ჩვენი კომპიუტერი ორივე მათგანის ფუნქციას ითავსებს, შესაძლებელია მისგან, როგორც Node.js-სერვერიდან, მოთხოვნილი ინფორმაცია გადავაგზავნოთ ამავე კომპიუტერზე არსებულ ბროუზერში.

დასაწყისისათვის დავწეროთ ასეთი მარტივი კოდი:

```
var http = require('http');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/plain'});

  res.end('მოგესალმები, სამყაროვ!');

}).listen(8080);
```

ამ კოდის ბაზაზე შევქმნათ .js გაფართოების ფაილი და განვათავსოთ იგი სერვერის ფაილურ სისტემაში.

ზემოთ მოყვანილი კოდის ბაზაზე შევქმნათ ფაილი myfirst1.js სახელით და განვათავსოთ იგი შემდეგ მისამართზე:

C:\Users\Your Name

აქვე ხაზი გავუსვამთ იმ გარემოებას, რომ Node.js-ის ფაილისათვის ადგილის არჩევაში შეზღუდული არ ვართ, მაგალითად, php სერვერული ენისაგან განსხვავებით.

ამრიგად, Node.js-ში შექმნილი სერვერული ფაილების შესრულებაზე გაშვება შესაძლებელი ხდება ჩვენსავე კომპიუტერზე

მათი განთავსების ნებისმიერი ადგილიდან, ოღონდ ეს ხორციელდება გარკვეული ხდომილობებისას, კერძოდ, საბრძანებო სტრიქონში, მაგალითად, ამ ინფორმაციის აკრეფვით:

```
C:\Users\Your Name>node myfirst1.js
```

ზემოთ მოყვანილი კოდი გვამცნობს, რომ ფაილის გამოძახებაზე რეაგირებს 8080 პორტი (ის დადარაჯებულია როგორც ჩვენი კომპიუტერიდან, ასევე გარედან მიმართვებზე).

ფაილი გაიშვება, რის შედეგადაც მოცემულ შემთხვევაში კლიენტს გადაეგზავნება და მის მონიტორზე აისახება კოდში მითითებული ეს ტექსტი:

მოგესალმები, სამყაროვ!

1.2. Node-ის ღირსებები

როგორც უკვე აღვნიშნეთ, Node.js პროგრამული პროდუქტის უდავო ღირსებაა ის თვისება, რომ იგი მუშაობს ასინქრონულ რეჟიმში.

უფრო დაწვრილებით ავხსნათ სხვა სისტემებთან შედარებით Node.js-ის ამ და სხვა უპირატესობების არსი:

1. სერვერისადმი მიმართვის შემდეგ (მაგალითად, კვლავ PHP-სგან განსხვავებით) კლიენტი არ ელოდება მისგან პასუხს - სცენარი გადადის მომდევნო ოპერატორების შესრულებაზე (ცხადია, თუკი ეს შესაძლებელია).
2. Node.js გარემოში ასინქრონული შეტანა-გამოტანის ოპერაციებისათვის განკუთვნილი სერვისები - სტანდარტულ ბიბლიოთეკაში გაერთიანებული პრიმიტივები, ფაქტობრივად (ანუ იშვიათი შემთხვევების გარდა), გამორიცხავენ კოდის შესრულე-ბისას მის ბლოკირებას (გარედან შემოსატანი ინფორმაციის

მოლოდინში). ამრიგად ხდება სისტემის მუშაობისას დაყოვნებების თავიდან აცილება.

3. გაგზავნილ მოთხოვნაზე პასუხი კლიენტს უბრუნდება მაშინ, როდესაც იგი მომზადდება.

4. Node.js-ში რეალიზებული მიდგომა ეფუძნება ერთ-ნაკადიანი, არაბლოკირებადი, ასინქრონული პროგრამირების კონცეფციას, რომელიც უზრუნველყოფს სისტემის მიერ დროისა და მეხსიერების ეფექტიანად გამოყენებას.

5. სისტემას შეუძლია ერთდროულად იმუშაოს არა ერთ, არამედ მრავალ მიმართვასთან, ამასთან ერთად, Node.js-ისთვის შექმნილი დანართები სრულდება ერთ ნაკადად, ყოველი ცალკეული მოთხოვნისათვის საკუთარის შექმნის გარეშე. შესაბამისად, სერვერს აღარ უხდება პარალელური ნაკადების შესრულების მიმდინარეობის „კურირება“ (რაც არცთუ იშვიათად შეფერხებებს იწვევდა და იწვევს!) და გამოთავისუფლებული დრო შეიძლება დაეთმოს გაცილებით მეტი კლიენტის მომსახურებას.

6. Node.js-ის მნიშვნელოვანი ღირსებაა მასშტაბირებადი ქსელური დანართების შექმნის შესაძლებლობაც.

7. მასშტაბურობის ქვეშ იგულისხმება Node.js-ის შემდეგი უნარი - სისტემაში რესურსების (მონაცემთა ბაზების, მარშრუტიზატორების და ქსელების ინფრასტრუქტურის) დამატება-გაძლიერების შემთხვევაში მას შეუძლია პროპორ-ციულად (ან თითქმის პროპორციულად) გაზარდოს თავისი წარმადობა.

პირველი, რასაც ვაკეთებთ კომპიუტერზე Node.js-სისტემის ჩატვირთვის შემდეგ (კიდევ ერთხელ - ჩამოტვირთვისათვის საჭირო ინსტრუქცია იხ. მისამართზე https://nodejs.org.), ესაა

Javascript ენაზე დაწერილი, შესაბამისად, js-გაფართოების მქონე ფაილის სერვერზე განთავსება.

ასეთი ფაილების შესრულებაზე გაშვება ხდება გარკვეული ხდომილობებისას.

ტიპური ხდომილება გახლავთ კლიენტის კომპიუტერიდან სერვერის პორტისადმი მიმართვა (ჩვეულებრივ, ამ პორტის ნომერია 8080).

კლიენტს შეუძლია სერვერზე როგორც ფაილებთან, ასევე მონაცემთა ბაზებთან ურთიერთობის წარმართვა (ამ ბაზების შექმნა და მათზე სხვადასხვა ოპერაციის ჩატარება).

კიდევ ერთხელ აღვნიშნოთ, რომ ჩვენ მიერ შექმნილ-დამახსოვრებული, მაგალითად, myfirst1.js ფაილის გაშვება შესაძლებელი არის საბრძანებო სტრიქონში ამ ინფორმაციის აკრეფვით:

C:\Users\Your Name>node myfirst.js

მაგრამ, რადგანაც ჩვენი კომპიუტერი უკვე როგორც სერვერი ფუნქციონირებს, შესაძლებელია მას მივმართოთ გარედანაც 8080 პორტის მეშვეობით, ანდა ამავე კომპიუტერის ბროუზერიდან სამისამართო უბანში

http://localhost:8080 მისამართის ჩვენებით.

შედეგად ამ მეორე შემთხვევაშიც კლიენტის მონიტორზე აისახება სერვერიდან გადმოცემული იგივე შეტყობინება:

მოგესალმები, სამყაროვ!

ამრიგად, სისტემისადმი ყოველი მიმართვისას ინიცირდება უკუკავშირი, ხოლო თუ დროის გარკვეული ინტერვალის

განმავლობაში მისდამი მიმართვები არ ხდება, სისტემას (ფხიზლად) სძინავს.

კლიენტისა და სერვერის ზემოთ აღწერილ ურთიერთობას წარმართავს npm (node package manager) პაკეტების მენეჯერი. იგი გახლავთ ინსტრუმენტი, რომლის მეშვეობითაც ხდება პაკეტების გადაგზავნა-გადმოგზავნა კლიენტებსა და npm ღრუბლოვან სერვერებს შორის.

აღვნიშნავთ, რომ Node.js-ში პაკეტად მიიჩნევა ერთი ან მეტი რაოდენობის JavaScript-ფაილი, წარმოდგენილი ბიბლიოთეკის ან რომელიმე სხვა ინსტრუმენტის სახით.

1.3. მოდულების თაობაზე

სერვერზე განთავსებულ ფაილში შესაძლებელი არის გამოძახებული იქნეს, როგორც ენაში **ჩაშენებული** (მათი სია იხ. დანართში), ასევე, - **ჩვენ მიერ შექმნილი** მ ო დ უ ლ ე ბ ი.

სპეციალისტები თვლიან, რომ Node.js-ში მოდულები, ფაქტობრივად, იმავე ფუნქციებს ასრულებენ, რასაც JavaScript-ენის ბიბლიოთეკები.

მოდული საშუალებას იძლევა, მონაცემები გადაიცეს შესაბამისი სახელის მქონე ოქმის მეშვეობით.

1.3.1. Node.js-ში ჩაშენებული HTTP-მოდული

მოვიყვანოთ სისტემაში ჩაშენებული **HTTP-მოდულის** გამოყენების მაგალითი. მისი დახმარებით მონაცემები გადაიგზავნება ჰიპერტექსტის გადაცემისათვის განკუთვნილი ოქმის მეშვეობით.

თავდაპირველად აღვნიშნოთ ის გარემოება, რომ ამ მიზნის მისაღწევად საჭიროა ჩვენს კომპიუტერზე Node.js გამოცხადდეს, როგორც HTTP-სერვერი.

ამის შემდეგ ვიყენებთ createServer() მეთოდს:

```
var http = require('http');

// create a server object:

http.createServer(function (req, res) {

  res.write('Hello World!'); //write a response to the client

  res.end(); //end the response

}).listen(8080); //the server object listens on port 8080
```

შევინახოთ ეს კოდი demo_http.js ფაილის სახელით.

მეთოდისათვის გადაცემული ფუნქცია მუშაობას იწყებს მაშინ, როდესაც ხდება რომელიმე კლიენტის მიერ სერვერის 8080 პორტისადმი მიმართვა.

გავუშვათ demo_http.js ფაილი შესრულებაზე:

```
C:\Users\Your Name>node demo_http.js
```

იმავე შედეგს მივიღებთ ბროუზერიდან სერვერისადმი http://localhost:8080 მიმართვისას.

თუკი გვსურს, რომ HTTP-სერვერიდან პასუხი მიღებული იქნეს HTML-ფორმატში, ასეთ შემთხვევაში შემდეგნაირად უნდა მოვახდინოთ ზემოთ მოყვანილი კოდის კორექტირება:

```
var http = require('http');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/html'});
```

```
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

მოდიფიცირებულ კოდში შეტანილ HTTP-სათაურში ასახული არის კონტენტის ტიპი - text/html.

res.writeHead() მეთოდის პირველი არგუმენტია მდგომარეობის ამსახველი კოდი. მისი მნიშვნელობა 200 გვაუწყებს, რომ ყველაფერი წესრიგშია, ხოლო მეორე არგუმენტი ასახავს პასუხის ტიპს.

1.3.2. მოთხოვნის სტრიქონის წაკითხვა

ზემოთ მოყვანილ კოდში http.createServer() მეთოდი არის ფუნქცია, რომლისთვისაც გადაცემული req არგუმენტი წარმოადგენს კლიენტისაგან სერვერისათვის http.IncomingMessage ობიექტის სახით გადაგზავნილ მოთხოვნას.

აღნიშნული ობიექტის .url თვისება შეიცავს URL-მისამართს, რომელიც ფიქსირდება დომენის სახელის შემდეგ:

```
demo_http_url.js
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

ეს კოდი შევინახოთ «demo_http_url.js» სახელის ფაილში და გავუშვათ იგი შესრულებაზე:

C:\Users\Your Name>node demo_http_url.js

ქვემოთ მოყვანილი გამოძახებების შედეგად:

http:// localhost:8080/summer

http:// localhost:8080/winter

მონიტორზე გამოვა 2 განსხვავებული შეტყობინება:

/summer

/winter

1.3.3. მოთხოვნის სტრიქონის ნაწილებად დაყოფა

ჩაშენებული მოდულების მეშვეობით შესაძლებელი არის მოთხოვნის სტრიქონი დაიყოს ცალკეულ წაკითხვად ნაწილებად.

ერთ-ერთი ასეთი მოდულია URL:

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

შევინახოთ ეს კოდი demo_querystring.js სახელის ფაილში და გავუშვათ იგი შესრულებაზე:

C:\Users\Your Name>node demo_querystring.js

მონიტორზე აისახება ასეთი შედეგი:

2023 July

1.4 Node.js - ფაილური სერვერი (fs მოდული)

შესაძლებელია Node.js-თან ვიმუშაოთ, როგორც ფაილურ სისტემასთან.

ამ მიზნის მისაღწევად განკუთვნილი შესაბამისი მოდულის ჩართვისათვის გამოიყენება require() მეთოდი:

```
var fs = require('fs');
```

აღნიშნული მოდულის ჩართვა საშუალებას იძლევა, მოვახდინოთ Node.js-ის ფაილების:

- წაკითხვა;
- შექმნა;
- განახლება;
- ამოგდება;
- სახელის გადარქმევა;
- ჩატვირთვა

1.4.1 ფაილების წაკითხვა

ფაილების წასაკითხავად ვიყენებთ `fs.readFile()` მეთოდს.

დავუშვათ, იმავე საქაღალდეში, რომელშიც განთავსებულია Node.js, ასევე ინახება demofile.html ფაილიც:

demofile.html

```
<html>
<body>
```

```
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

შევქმნათ Node.js ფაილი, რომელიც წაიკითხავს ამ HTML-ფაილს და დააბრუნებს მის შემცველობას:

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  // Open a file on the server and return its content:
  fs.readFile('demofile_1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

შევინახოთ ეს კოდი demo_readfile.js სახელის მქონე ფაილში და გავუშვათ შესრულებაზე:

```
C:\Users\Your Name>node demo_readfile.js
```

მიმართვის შედეგი იქნება:

My Header

My paragraph.

იგივე ინფორმაცია აისახება მონიტორზე ამ მიმართვის შემდეგაც:

http://localhost:8080

1.4.2. ფაილების შექმნა

ახალი ფაილების შესაქმნელად File System მოდულში გათვალისწინებული არის შემდეგი მეთოდები:

- fs.appendFile()
- fs.open()
- fs.writeFile()

1) fs.appendFile() მეთოდი არსებულ ფაილში ამატებს მითითებული კოდის შემცველობას. იმ შემთხვევაში კი, როდესაც ფაილი არ არსებობს, იგი იქმნება:

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {

 if (err) throw err;

 console.log('Saved!');

});
```

console.log-ში გამოვა შეტყობინება:

Saved!

2) fs.open() მეთოდი. იმ შემთხვევაში, როდესაც ამ მეთოდისათვის გადაცემული მეორე არგუმენტის მნიშვნელობა არის "w" (ანუ ბრძანება ჩაწერაზე), იგი გააღებს მითითებულ ფაილს და მოამზადებს მას

ჩაწერისათვის. თუ მითითებული ფაილი არ არსებობს, იქმნება ცარიელი ფაილი.

ვაჩვენოთ შესაბამისი მაგალითი ბოლო შემთხვევისათვის:

```
var fs = require('fs');

fs.open('mynewfile2.txt', 'w', function (err, file) {

 if (err) throw err;

 console.log('Saved!');

});
```

console.log-ში გამოვა შეტყობინება:

```
Saved!
```

3) fs.writeFile() მეთოდი კი, თუ მითითებული ფაილი არსებობს, მის შემცველობას შეცვლის ახლით, ხოლო არარსებობის შემთხვევაში შეიქმნება ახალი ფაილი შესაბამისი შემცველობით:

```
var fs = require('fs');

//create a file named mynewfile3.txt:

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {

 if (err) throw err;

 console.log('Saved!');

});
```

console.log_ში გამოვა შეტყობინება:

```
Saved!
```

1.4.3. ფაილების განახლება

ფაილების განახლებისათვის File System მოდულში გათვალისწინებულია შემდეგი მეთოდები:

- fs.appendFile()
- fs.writeFile()

1) fs.appendFile() მეთოდი არსებულ ფაილში ამატებს რაიმე ინფორმაციას მითითებული ფაილის ბოლოში:

```
var fs = require('fs');

//append content at the end of the file:

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {

  if (err) throw err;

  console.log('Updated!');

});
```

console.log-ში გამოვა შეტყობინება:

```
Updated!
```

2) fs.writeFile() მეთოდი კი მთლიანად განაახლებს მითითებული ფაილის შემცველობას:

```
var fs = require('fs');

//Replace the file with a new one:

fs.writeFile('mynewfile3.txt', 'This is my text.', function (err) {

  if (err) throw err;

  console.log('Replaced!');

});
```

console.log-ში გამოვა შეტყობინება:

Replaced!

1.4.4. ფაილების ამოგდება

ფაილების ამოგდება ხდება fs.unlink() მეთოდით:

```
var fs = require('fs');

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

console.log-ში გამოვა შეტყობინება:

File deleted!

1.4.5. ფაილისთვის სახელის გადარქმევა

ფაილებისათვის სახელის გადარქმევისათვის გათვალისწინებული არის fs.rename() მეთოდი:

```
var fs = require('fs');

//Rename the file "mynewfile1.txt" into "myrenamedfile.txt":

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

console.log-ში დაიწერება:

File Renamed!

1.4.6. ფაილების ჩატვირთვა

ეს საკითხი მოგვიანებით განიხილება, მანამდე კი შევისწავლით URL-მოდულის დანიშნულებისა და მასთან მუშაობის საკითხებს.

1.5. URL ჩაშენებული მოდული

URL მოდულის მეშვეობით ვებ-მისამართი იყოფა კითხვად ნაწილებად.

ფაილში მოდულის ჩართვა ხდება require() მეთოდით:

```
var url = require('url');
```

ვაჩვენოთ ამ მისამართის url.parse() მეთოდით გაანალიზებისა და მისი URL-ობიექტის თვისებების სახით ცალკეულ ნაწილებად დაყოფა-წარმოდგენის მაგალითი:

```
var url = require('url');
var adr =
'http://localhost:8080/default.htm?year=2017&month=february';

// Parse the address:

var q = url.parse(adr, true);

/* The parse method returns an object containing url properties */
console.log(q.host);

console.log(q.pathname);

console.log(q.search);
```

```
/ *The query property returns an object with all the querystring parameters as properties: */

var qdata = q.query;

console.log(qdata.month);
```

ფაილის შესრულებაზე გაშვების შემდეგ console.log_ში გამოვა შემდეგი სახის შეტყობინებები:

```
localhost:8080

/default

?year=2017&month=february

february
```

მას შემდეგ, რაც შევისწავლეთ, თუ როგორ უნდა განხორციელდეს Node.js-სათვის ფაილური სერვერის ფუნქციის დაკისრება, ასევე, - როგორ უნდა ჩატარდეს მოთხოვნის სტრიქონის ანალიზი, ვაჩვენოთ კლიენტის მიერ ამ შესაძლებლობების ერთ ფაილში გამოყენების მაგალითი.

ამ მიზნით, ვქმნით ორ html-ფაილს და განვათავსებთ მათ ჩვენ მიერ node.js-ფაილების შენახვისათვის უკვე შერჩეულ საქაღალდეში:

summer.html

```
<!DOCTYPE html>

<html>

<body>

<h1>Summer</h1>

<p>I love the sun!</p>
```

```
</body>
</html>
```

winter.html

```
DOCTYPE html>
<html>
<body>
<h1>Winter<!</h1>
<p>I love the snow!</p>
</body>
</html>
```

ვქმნით Node.js-ფაილს, რომელიც აღებს მოთხოვნილ ფაილს და მის შემცველობას უბრუნებს კლიენტს, რაიმე შეფერხების შემთხვევაში კი იძლევა შეცდომის შესახებ მაუწყებელ 404 კოდს:

demo_fileserver.js

```
var http = require('http');
var url = require('url');
var fs = require('fs');
http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
```

```
    res.writeHead(404, {'Content-Type': 'text/html'});
    return res.end("404 Not Found");
  }
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(data);
  return res.end();
 });
}).listen(8080);
```

ვუშვებთ ფაილს შესრულებაზე:

```
C:\Users\Your Name>node demo_fileserver.js
```

შემდეგ კი გავცემთ ამ ბრძანებებს:

1. http://localhost:8080/summer.html
   შედეგი იქნება:

   Summer

   I love the sun!

2. http://localhost:8080/winter.html
   კი მოგვცემს:

   Winter

   I love the snow!

1.6. Node.js - NPM

NPM წარმოადგენს მენეჯერების პაკეტს, რომლის მეშვეობითაც ხდება Node.js-ის მოდულების (პაკეტების) ჩამოტვირთვა-შესრულება.

კერძოდ, ეს ქმედებანი შესაძლებელი არის განხორციელდეს www.npmjs.com მისამართზე მდებარე საიტიდან, რომელზეც განთავსებულია ათასობით უფასო პაკეტი.

აღვნიშნავთ, რომ NPM პაკეტი თან მოყვება Node.js-ს მისი დაყენებისას და უკვე მზად არის გაცემული დავალებების შესასრულებლად.

საერთოდ, პაკეტები Node.js-ში შეიცავს მოდულის ფუნქციონირებისათვის საჭირო ყველა ფაილს. პაკეტი (მოდული), ფაქტობრივად, არის Javascript-ის ბიბლიოთეკა, რომელიც შესაძლებელია ჩავრთოთ საკუთარ პროექტში.

მოვიყვანოთ მაგალითი:

C:\Users\Your Name>npm install upper-case

ამ upper-case პაკეტის ჩამოტვირთვა-ინსტალირებისას NPM მენეჯერი ფაილის შესანახავად ქმნის node_modules საქაღალდეს, რომელშიც ჩაიტვირთება ყველა ის მოდული, რომელთაც მომავალში გამოვიძახებთ.

შედეგად ჩვენი პროექტის სტრუქტურა ასეთ სახეს მიიღებს:

C:\Users\My Name\node_modules\upper-case

ამის შემდეგ შესაძლებელი ხდება upper-case პაკეტის ჩავრთოთ ჩვენ მიერ შექმნილ Node.js ფაილებშიისეთივე წესით, რომელსაც ვიყენებდით აქამდე:

```
var uc = require('upper-case');
```

მიზნად დავისახოთ ისეთი Node.js ფაილის შექმნა, რომელიც "Hello World!" ტექსტს მონიტორზე მთავრულ ასოებში გამოიყვანს:

```
var http = require('http');

var uc = require('upper-case');

http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/html'});

  res.write(uc.upperCase("Hello World!"));

  res.end();

}).listen(8080);
```

კოდი დავიმახსოვროთ "demo_uppercase.js" სახელის ფაილში და გავუშვათ ის შესრულებაზე:

```
C:\Users\Your Name>node demo_uppercase.js
```

მონიტორზე გამოვა:

```
HELLO WORLD!
```

იმავე შედეგს მივიღებთ ამ მიმართვის შედეგადაც:

http://localhost:8080

1.7. მოვლენები (Events) Node.js-ში

როგორც უკვე აღვნიშნეთ, Node.js გამოყენებების (დანართების) მართვისათვის ეფექტიანად იყენებს ხდომილობების (მოვლენების) მექანიზმს.

შევნიშნავთ, რომ ხდომილობად მიიჩნევა ნებისმიერი ქმედება, რომელიც ხორციელდება კომპიუტერზე. მათი ინიცირება შეუძლია ობიექტებს.

მაგალითად, ვაჩვენოთ, თუ როგორ ხდება ფაილის გაღებისა და დახურვის ხდომილობების გაშვება readStream ობიექტის მიერ:

```
var fs = require('fs');
var rs = fs.createReadStream('./demofile.txt');
rs.on('open', function () {
  console.log('The file is open');
});
```

ოპერაციის წარმატებით დამთავრების შესახებ console.log-ში გამოვა ეს შეტყობინება:

```
'The file is open';
```

აღსანიშნავია, რომ Node.js-ის სტრუქტურაში შედის "Events" ჩაშენებული მოდული, რომლის მეშვეობითაც შესაძლებელია საკუთარი ხდომილობების შექმნა, გაშვება და მათი ქმედებებისათვის თვალ-ყურის მიდევნება. საკუთარ ფაილში ამ მოდულის ჩასართავად ვქმნით შესაბამის ობიექტს უკვე ნაცნობი წესით:

```
var events = require('events');
```

ამასთან, რადგანაც ხდომილობათა თვისებები და მეთოდები EventEmitter ობიექტის ეგზემპლარებია, მათთან მიდგომა-გამოყენებისათვის საჭირო ხდება კიდევ ერთი ობიექტის შექმნა:

```
var eventEmitter = new events.EventEmitter();
```

ამ ქმედებების შემდეგ შესაძლებელი ხდება ჩვენ მიერ შექმნილ ხდომილობებს დავუნიშნოთ ხდომილობათა დამმუშავებელნი - ფუნქციები, რომლებიც შესრულდება მოცემული ხდომილობის გაშვებისას.

ვაჩვენოთ შესაბამისი მაგალითი:

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
 //Create an event handler:
var myEventHandler = function () {
 console.log('I hear a scream!');
}
 //Assign the event handler to an event:
eventEmitter.on('scream', myEventHandler);
 //Fire the 'scream' event:
eventEmitter.emit('scream');
```

console.log-ში გამოვა შეტყობინება:

```
I hear a scream!
```

ზემოთ მოყვანილ მაგალითში გაიშვება ჩვენ მიერ შექმნილი myEventHandler ფუნქცია და შესრულებას დაიწყებს "scream"

ხდომილობისას. ამ ხდომილობის გასაშვებად კი გამოიყენება .emit() მეთოდი.

1.8. Node.js ფაილების ჩატვირთვა

Node.js-ში ფაილების ჩატვირთვას ემსახურება Formidable მოდული. მისი ჩამოტვირთვა-დაყენება ასევე npm-ის მეშვეობით ხდება:

```
C:\Users\Your Name>npm install formidable
```

ამის შემდეგ მოდულის ჩართვა შესაძლებელია ნებისმიერ დანართში:

```
var formidable = require('formidable');
```

შევქმნათ Node.js ვებფურცელი, რომელიც მოგვცემს ჩვენს კომპიუტერზე ფაილების ჩამოტვირთვის შესაძლებლობას.

ეს მოხდება 3 ეტაპად:

ბიჯი_1:

თავდაპირველად საჭიროა შეიქმნეს Node.js ფაილი, რომელშიც ჩაიწერება HTML-ფორმა ჩატვირთვის ველთან ერთად:

```
var http = require('http');
http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/html'});

res.write('<form action="fileupload" method="post"
  enctype="multipart/form-data">');

res.write('<input type="file"
  name="filetoupload"><br>');
```

```
res.write('<input type="submit">');

res.write('</form>');

return res.end();

}).listen(8080);
```

ბიჯი_2:

ამ ბიჯზე ხდება ჩამოტვირთული ფაილის გაანალიზება.

ჩამოტვირთული ფაილის გასაანალიზებლად ირთვება Formidable მოდული, რის შემდეგაც იგი განთავსდება ჩვენს კომპიუტერზე დროებით საქაღალდეში.

```
var http = require('http');
var formidable = require('formidable');
http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      res.write('File uploaded');
      res.end();
    });
  } else {
    res.writeHead(200, {'Content-Type': 'text/html'});
```

```
        res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
        res.write('<input type="file"
name="filetoupload"><br>');
        res.write('<input type="submit">');
        res.write('</form>');
        return res.end();
      }
    }).listen(8080);
```

ბიჯი_3:

ფაილის შენახვა

სერვერზე წარმატებით ჩატვირთული, დროებით საქაღალდეში განთავსებული ფაილის სხვა რომელიმე საქაღალდეში გადასანაცვლებლად «files» ობიექტიდან განვსაზღვრავთ საქაღალდის მისამართს (გზას მასთან მისასვლელად). ეს ობიექტი მესამე პარამეტრად გადაეცემა უკუგამოძახების მეთოდის parse() ფუნქციას.

მიზნის მისაღწევად გამოიყენება File System-ის fs მოდული და კოდში ხდება ფაილისათვის სახელის გადარქმევაც:

```
    var http = require('http');
    var formidable = require('formidable');
    var fs = require('fs');
http.createServer(function (req, res) {
```

```
    if (req.url == '/fileupload') {
      var form = new formidable.IncomingForm();
      form.parse(req, function (err, fields, files) {
        var oldpath = files.filetoupload.filepath;
        var newpath = 'C:/Users/Your Name/' +
files.filetoupload.originalFilename;
        fs.rename(oldpath, newpath, function (err) {
          if (err) throw err;
          res.write('File uploaded and moved!');
          res.end();
        });
    });
     } else {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
      res.write('<input type="file" name="filetoupload"><br>');
      res.write('<input type="submit">');
      res.write('</form>');
      return res.end();
     }
  }).listen(8080);
```

1.9. Node.js - ელექტრონული წერილების გაგზავნა

Node.js-ში ელექტრონული წერილების გაგზავნას ემსახურება Nodemailer მოდული. მისი დაყენებაც ხდება მიღებული წესებით:

```
C:\Users\Your Name>npm install nodemailer
```

ინსტალირების შემდეგ ამ მოდულის ჩართვა შესაძლებელი არის ნებისმიერ დანართში:

```
var nodemailer = require('nodemailer');
```

ინსტალირების შემდეგ ამ მოდულის ჩართვა შესაძლებელი არის ნებისმიერ დანართში. მოგვყავს წერილის გაგზავნისათვის დაწერილი კოდი:

```
var nodemailer = require('nodemailer');
var transporter = nodemailer.createTransport({
service: 'gmail',
auth: {
user: 'youremail@gmail.com',
pass: 'yourpassword'
}
});
var mailOptions = {
from: 'youremail@gmail.com',
to: 'myfriend@yahoo.com',
```

```
  subject: 'Sending Email using Node.js',
  text: 'That was easy!'
};
transporter.sendMail(mailOptions, function(error, info){
  if (error) {
    console.log(error);
  } else {
    console.log('Email sent: ' + info.response);
  }
});
```

წერილის რამდენიმე მისამართით გასაგზავნად mailOptions ობიექტის თვისებაში შეგვაქვს მიმღებების შესახებ შესაბამისი ინფორმაცია:

```
var mailOptions = {
  from: 'youremail@gmail.com',
  to: 'myfriend@yahoo.com, myotherfriend@yahoo.com',
  subject: 'Sending Email using Node.js',
  text: 'That was easy!'
}
```

ხოლო წერილის HTML ფორმატში გადასაგზავნად კოდში text-თვისების ნაცვლად ვწერთ html-ს:

```
var mailOptions = {
```

```
from: 'youremail@gmail.com',
to: 'myfriend@yahoo.com',
subject: 'Sending Email using Node.js',
html: '<h1>Welcome</h1><p>That was easy!</p>'
}
```

თავი 2. Node.js და MySQL

Node.js-ს შეუძლია ითანამშრომლოს მონაცემთა ბაზებთან, კერძოდ, ერთ-ერთ ყველაზე პოპულარულ MySql ბაზასთან.

ჩამოვტვირთოთ ეს ეს ბაზა https://www.mysql.com/downloads/ საიტიდან და ავამოქმედოთ.

Node.js-სა და ბაზას შორის კავშირის დასამყარებლად (დრაივერის დასაყენებლად) საბრძანებო საბრძანებო სტრიქონში ვკრეფთ:

C:\Users\Your Name>npm install mysql

2.1. ბაზასთან დაკავშირება

ვწერთ შემდეგ კოდს, რომელშიც ვიყენებთ ჩვენს ბაზაში დაფიქსირებულ მომხმარებლის სახელს და პაროლს:

demo_db_connection.js

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});
```

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```

კოდს ვინახავთ demo_db_connection.js სახელის ფაილში და ვუშვებთ შესრულებაზე:

```
C:\Users\Your Name>node demo_db_connection.js
```

console.log-ში გამოვა:

```
Connected!
```

2.2. მოთხოვნის ფორმირება

ამ მიზნით ვიყენებთ ზემო მაგალითში შექმნილი con მიერთების ობიექტის .query() მეთოდს:

```
con.connect(function(err) {
    if (err) throw err;
    console.log("Connected!");
    con.query(sql, function (err, result) {
      if (err) throw err;
      console.log("Result: " + result);
    });
  });
```

ამ მეთოდს პარამეტრებად გადაეცემა sql-ოპერატორები, რომელთათ შესრულების შედეგად ხდება შესაბამისი ინფორმაციის დაბრუნება.

შენიშვნა: *sql-ოპერატორების სინტაქსი განიხილება მრავალ წყაროში, მაგალითად, საიტზე:*

https://www.w3schools.com/sql/default.asp

2.3. მონაცემთა ბაზის შექმნა

მონაცემთა ბაზის სახელად ავირჩიოთ mydb და დავწეროთ კოდი, რომელშიც ბაზა იქმნება CREATE DATABASE ოპერატორით:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

კოდი, შენახული «demo_create_db.js სახელით გავუშვათ შესრულებაზე:

```
C:\Users\Your Name>node demo_create_db.js
```

console.log-ში გამოვა ეს შეტყობინებები:

```
Connected!

Database created
```

2.4. ცხრილის შექმნა

ცხრილის სახელად ვირჩევთ customers-ს და ვიყენებთ CREATE TABLE ოპერატორს:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```

demo_create_table.js სახელით შენახული ეს კოდი გავუშვათ შესრულებაზე:

```
C:\Users\Your Name>node demo_create_table.js
```

console.log-ში გამოვა შეტყობინებები:

```
Connected!

Table created
```

2.5. Primary Key - ძირითადი (პირველადი) გასაღები

ცხრილის შექმნისას ერთ-ერთი სვეტი განისაზღვრება, როგორც უნიკალური გასაღებური ველი, მისი იდენტიფიკატორისათვის ამ თვისების - «INT AUTO_INCREMENT PRIMARY KEY» მინიჭებით.

ეს ნიშნავს, რომ ცხრილში ყოველი ახალი ჩანაწერის დამატებისას, მას ავტომატურად მიენიჭება რიცხვითი ნომერი, დაწყებული 1-ით და ყოველ შემდეგ ჩანაწერს კი - ერთი ნომრით მეტი.

ვაჩვენოთ მაგალითი:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!")
  var sql = "CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
```

```
  });
});
```

demo_db_insert.js ფაილში დამახსოვრებული ამ კოდის შესრულებაზე გაშვების შემდეგ console.log-ში გამოვა ეს შეტყობინებები:

```
Connected!

Table created
```

იმ შემთხვევაში კი, თუ ცხრილი უკვე არსებობს, ამავე მიზნის მისაღწევად ვიყენებთ ALTER TABLE ოპერატორს:

```
var mysql = require('mysql');

var con = mysql.createConnection({

  host: "localhost",

  user: "yourusername",

  password: "yourpassword",

  database: "mydb"

});

con.connect(function(err) {

  if (err) throw err;

  console.log("Connected!");

  var sql = "ALTER TABLE customers ADD COLUMN id INT
AUTO_INCREMENT PRIMARY KEY";

  con.query(sql, function (err, result) {
```

```
  if (err) throw err;
  console.log("Table altered");
 });
});
```

ზემოთ მოყვანილი კოდები დავიმახსოვროთ ფაილებად და გავუშვათ შესრულებაზე.

მონიტორზე აისახება - პირველ შემთხვევაში:

```
Connected!
Table created
```

მეორე შემთხვევაში:

```
Connected
Table altered
```

2.6. ცხრილში ჩანაწერის ჩასმა

ვაჩვენოთ ცხრილში ჩანაწერის ჩასმის მაგალითი:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
```

```
  console.log("Connected!");
  var sql = "INSERT INTO customers (name, address) VALUES
('Company Inc', 'Highway 37')";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("1 record inserted");
  });
});
```

კოდი დავიმახსოვროთ demo_create_db.js სახელით და ეს ფაილი შესრულებაზე გავუშვათ:

```
C:\Users\Your Name>node demo_db_insert
```

შედეგი იქნება:

```
Connected!
1 record inserted
```

2.7. ცხრილში რამდენიმე ჩანაწერის ჩასმა

ცხრილში ერთ ჯერზე რამდენიმე ჩანაწერის ჩასასმელად უნდა შევქმნათ ჩანაწერების მასივი, ხოლო VALUES გასაღებური სიტყვის შემდეგ უნდა ფიგურირებდეს ? სიმბოლო, რომელსაც შეცვლის ქვემოთ მოყვანილი მასივი:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
```

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "INSERT INTO customers (name, address) VALUES ?";
  var values = [
     ['ბადრი', 'მისამართი_01'],
    ['გიორგი', 'მისამართი_02'],
    ['გურამი', 'მისამართი_03'],
    ['დავითი', 'მისამართი_04'],
    ['თამარი', 'მისამართი_05'],
    ['კახა', 'მისამართი_06'],
    ['ლალი', 'მისამართი_07'],
    ['ლაშა', 'მისამართი_08'],
    ['ლევანი', 'მისამართი_09'],
    ['ლია, 'მისამართი_10'],
    ['მზია', 'მისამართი_11'],
    ['მიხეილი', 'მისამართი_12'],
    ['ნანი', 'მისამართი_13'],
    ['ნოდარი', 'მისამართი_14']
  ];

    con.query(sql, [values], function (err, result) {
     if (err) throw err;
 console.log("Number of records inserted: " + result.affectedRows);
    });
  });
```

კოდი დავიმახსოვროთ «demo_db_insert_multple.js» სახელით და ფაილი შესრულებაზე გავუშვათ:

```
C:\Users\Your Name>node demo_db_insert
```

შედეგი იქნება:

```
Connected!
Number of records inserted: 14
```

2.8. შედეგის ამსახველი ობიექტი

მოთხოვნის შესრულების შედეგად გვიბრუნდება ცხრილში შეტანილი ცვლილებების ამსახველი ობიექტი.

წინა მოთხოვნის შესრულება გამოიწვევდა ამ ცვლილებებს:

```
{
 fieldCount: 0,
 affectedRows: 14,
 insertId: 0,
 serverStatus: 2,
 warningCount: 0,
 message: '\'Records:14  Duplicated: 0  Warnings: 0',
 protocol41: true,
 changedRows: 0
}
```

იმ სტრიქონების რიცხვის გასაგებად, რომელთაც ეს ცვლილებები შეეხო, გავცემთ ბრძანებას:

```
console.log(result.affectedRows)
```

მონიტორზე აისახება ეს შედეგი:

```
14
```

თუ ჩანაწერის, ამასთან, მხოლოდ ერთის, ჩამატება მოხდა ისეთ ცხრილში, რომლისთვისაც გათვალისწინებული არის იდენტიფიკატორის ველი მნიშვნელობის ავტომატური ზრდით, შესაძლებელი ხდება ახლად ჩამატებული ჩანაწერის ნომრის გაგება.

ვაჩვენოთ ეს შესაძლებელობა შემდეგ მაგალითზე:

```
var mysql = require('mysql');

var con = mysql.createConnection({
```

```
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "INSERT INTO customers (name, address) VALUES ('პეტრე', 'მისამართი_99')";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("1 record inserted, ID: " + result.insertId);
  });
});
```

დავიმახსოვროთ ეს კოდი demo_db_insert_id.js სახელის ფაილში და გავუშვათ შესრულებაზე:

```
C:\Users\Your Name>node demo_db_insert_id.js
```

მონიტორზე აისახება ეს ინფორმაცია:

```
1 record inserted, ID: 15
```

2.9. ცხრილიდან მონაცემების ამორჩევა

ცხრილიდან მონაცემების ამოირჩევა "SELECT" ოპერატორის მეშვეობით.

განვიხილოთ მაგალითი:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM customers", function (err, result, fields) {
    if (err) throw err;
    console.log(result);
  });
});
```

კოდი დავიმახსოვროთ demo_db_select.js» ფაილის სახელით და გავუშვათ შესრულებაზე:

```
C:\Users\Your Name>node demo_db_select.js
```

SELECT * ოპერატორი მითითებული ცხრილიდან ამოკრეფს ყველა ჩანაწერს:

```
[
  { id: 1, name: 'ბადრი', address: 'მისამართი_01'},
  { id: 2, name: 'გიორგი', address: 'მისამართი_02'},
  { id: 3, name: 'გურამი', address: 'მისამართი_03'},
```

```
  { id: 4, name: 'დავითი', address: 'მისამართი_04'},
  { id: 5, name: 'თამარი', address: 'მისამართი_05'},
  { id: 6, name: 'კახა', address: 'მისამართი_06'},
  { id: 7, name: 'ლალი', address: 'მისამართი_07'},
  { id: 8, name: 'ლაშა', address: 'მისამართი_08'},
  { id: 9, name: 'ლევანი', address: 'მისამართი_09'},
  { id: 10, name: 'ლია', address: 'მისამართი_10'},
  { id: 11, name: 'მზია', address: 'მისამართი_11'},
  { id: 12, name: 'მიხეილი', address: 'მისამართი_12'},
  { id: 13, name: 'ნანი', address: 'მისამართი_13'},
  { id: 14, name: 'ნოდარი', address: 'მისამართი_14'}
]
```

ცხრილიდან ამოვირჩიოთ ზოგიერთი სვეტი, მაგალითად, „კლიენტის სახელი“ და „მისამართი“:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
```

```
con.query("SELECT name, address FROM customers", function
(err, result, fields) {
  if (err) throw err;
  console.log(result);
 });
});
```

კოდი დავიმახსოვროთ "demo_db_select2.js" სახელის ფაილში და გავუშვათ შესრულებაზე:

```
C:\Users\Your Name>node demo_db_select2.js
```

მივიღებთ ასეთ შედეგს:

```
[
 { name: 'ბადრი', address: 'მისამართი_01'},
 { name: 'გიორგი', address: 'მისამართი_02'},
 { name: 'გურამი', address: 'მისამართი_03'},
 { name: 'დავითი', address: 'მისამართი_04'},
 { name: 'თამარი', address: 'მისამართი_05'},
 { name: 'კახა', address: 'მისამართი_06'},
 { name: 'ლალი', address: 'მისამართი_07'},
 { name: 'ლაშა', address: 'მისამართი_08'},
 { name: 'ლევანი', address: 'მისამართი_09'},
 { name: 'ლია', address: 'მისამართი_10'},
 { name: 'მზია', address: 'მისამართი_11'},
 { name: 'მიხეილი', address: 'მისამართი_12'},
```

```
  { name: 'ნანი', address: 'მისამართი_13'},
  { name: 'ნოდარი', address: 'მისამართი_14'}
]
```

ამჯერად მიზნად დავისახოთ ცხრილიდან რომელიმე კონკრეტული, მაგალითად, მესამე ჩანაწერის ამორჩევა.

ეს შესაძლებელი იქნება შემდეგი ოპერატორის მეშვეობით:

```
console.log(result[2].address);
```

მისი შესრულების შედეგი იქნება:

მისამართი_03

შემდეგ, უკუგამოძახების ფუნქციაში მესამე პარამეტრია მასივი, რომელიც შეიცავს ინფორმაციას result-ში თითოეული ველის (ფაქტობრივად, ობიექტის) შესახებ და თუ ზემოთ მოყვანილ კოდში console.log(result) ოპერატორის ნაცვლად გამოვიყენებთ console.log(fields) ოპერატორს:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
```

```
con.query("SELECT name, address FROM customers", function
(err, result, fields) {
  if (err) throw err;
  console.log(fields);
 });
});
```

მაშინ ახალი, "demo_db_select_fields.js" ფაილის შესრულებაზე გაშვების შედეგად მივიღებთ:

```
[
 {
  catalog: 'def',
  db: 'mydb',
  table: 'customers',
  orgTable: 'customers',
  name: 'name',
  orgName: 'name',
  charsetNr: 33,
  length: 765,
  type: 253,
  flags: 0,
  decimals: 0,
  default: undefined,
  zeroFill: false,
```

```
    protocol41: true
  },
  {
    catalog: 'def',
    db: 'mydb',
    table: 'customers',
    orgTable: 'customers',
    name: 'address',
    orgName: 'address',
    charsetNr: 33,
    length: 765,
    type: 253,
    flags: 0,
    decimals: 0,
    default: undefined,
    zeroFill: false,
    protocol41: true
  }
]
```

ჩანს, რომ `fields` ობიექტი მასივია, რომელიც ცხრილში არსებული ველების, როგორც ობიექტების, შესახებ შეიცავს ინფორმაციებს.

შესაბამისად, თუკი გვსურს, მოვიპოვოთ ინფორმაცია ცხრილში მეორე ველის სახელის თაობაზე, უნდა მივმართოთ მასივის მეორე ელემენტის შესაბამის თვისებას:

```
console.log(fields[1].name);
```

შედეგად დაგვიბრუნდება: Address

2.10. მონაცემების ამორჩევა ფილტრაციით

ცხრილიდან მონაცემების ამორჩევა შესაძლებელი არის მათი გარკვეული წესებით ფილტრირებითაც.

ამ მიზნის მისაღწევად გამოიყენება WHERE ოპერატორი.

ქვემოთ მოყვანილ კოდში ამოირჩევა მხოლოდ ის ჩანაწერები, რომლებშიც შემკვეთის მისამართია 'Park Lane 38'.

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM customers WHERE address =
'მისამართი_11'", function (err, result) {
```

```
  if (err) throw err;
  console.log(result);
 });
});
```

კოდს ვიმახსოვრებთ "demo_db_where.js" სახელის ფაილში და ვუშვებთ შესრულებაზე:

```
C:\Users\Your Name>node demo_db_where.js
```

ვიღებთ ამ შედეგს:

```
[
 { id: 11, name: 'მზია', address: 'მისამართი_11''}
]
```

ფილტრაციისას შეგვიძლია ვისარგებლოთ ჩადგმადი სიმბოლოებითაც, როდესაც გვსურს ამა თუ სიმბოლოების შემცველი ინფორმაციის მოძიება.

კერძოდ, % სიმბოლო საშუალებას იძლევა, ჩაანაცვლოს *ნული* და *ერთი* ან რამდენიმე სიმბოლო.

მაგალითად, თუ ვეძებთ ისეთ მისამართებს, რომლებიც იწყება S სიმბოლოთ, ვწერთ ასეთ კოდს:

```
var mysql = require('mysql');
var con = mysql.createConnection({
 host: "localhost",
 user: "yourusername",
 password: "yourpassword",
 database: "mydb"
```

```
});

con.connect(function(err) {
  if (err) throw err;
 con.query("SELECT * FROM customers WHERE address LIKE
     'მისამართი_1%'", function (err, result) {
   if (err) throw err;
   console.log(result);
  });
});
```

"demo_db_where_s.js" სახელით შენახული ამ კოდის ფაილი გავუშვათ შესრულებაზე:

```
C:\Users\Your Name>node demo_db_where_s.js
```

შედეგად ვღებულობთ:

```
[
  { id: 10 name: 'ლია', address: 'მისამართი_10'},
  { id: 11 name: 'მზია', address: 'მისამართი_11'},
  { id: 12, name: 'მიხეილი', address: 'მისამართი_12'},
  { id: 13, name: 'ნანი', address: 'მისამართი_13'},
  { id: 14, name: 'ნოდარი', address: 'მისამართი_14'}
]
```

შენიშვნა: *თუ მომხმარებლისგან გამოგზავნილ მოთხოვნაში გადმოცემული მნიშვნელობები ცვლადებია, ჰაკერების შესაძლო შემოტევებისაგან თავის დასაცავად უნდა*

მივმართოთ მოთხოვნის მნიშვნელობების ეკრანირებას, რაც რეალიზდება mysql.escape() მეთოდით:

```
var adr = 'Mountain 21';
var sql = 'SELECT * FROM customers WHERE address = ' +
mysql.escape(adr);
con.query(sql, function (err, result) {
  if (err) throw err;
  console.log(result);
});
```

შესაძლებელია ასევე, მნიშვნელობებისათვის, რომელთა ეკრანირებაც გვსურს, გამოვიყენოთ ? სიმბოლო, როგორც შემავსებელი.

ასეთ შემთხვევაში ცვლადი, რომელიც ფიგურირებს query() მოთხოვნაში, მასში მეორე პარამეტრად გადაიცემა:

```
var adr = 'Mountain 21';
var sql = 'SELECT * FROM customers WHERE address = ?';
con.query(sql, [adr], function (err, result) {
  if (err) throw err;
  console.log(result);
});
```

შედეგი იქნება ასეთი:

```
[
  { id: 4, name: 'Hannah', address: 'Mountain 21'}
```

```
]
```

ქვემოთ მოგვყავს კოდის დაწერის მაგალითი რამდენიმე შემავსებლის არსებობის შემთხვევაში:

```
var name = 'Amy';
var adr = 'Mountain 21';
var sql = 'SELECT * FROM customers WHERE name = ? OR address = ?';
con.query(sql, [name, adr], function (err, result) {
  if (err) throw err;
  console.log(result);
});
```

და შედეგი:

```
[
 { id: 3, name: 'Amy', address: 'Apple st 652'}
  { id: 4, name: 'Hannah', address: 'Mountain 21'}
]
```

2.11 მონაცემების დახარისხება

მონაცემების დახარისხება ხდება ORDER BY ოპერატორის მეშვეობით, დუმილით მათი მნიშვნელობების ზრდადობის მიხედვით, ხოლო DESC გასაღებური სიტყვის გამოყენებისას კი - კლებადობით.

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
```

```
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM customers ORDER BY
name", function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

შევქმნათ და შესრულებაზე გავუშვათ `demo_db_orderby.js` ფაილი:

```
C:\Users\Your Name>node demo_db_orderby.js
```

მივიღებთ ასეთ შედეგს:

```
[
 { id: 3, name: 'Amy', address: 'Apple st 652'},
 { id: 11, name: 'Ben', address: 'Park Lane 38'},
 { id: 7, name: 'Betty', address: 'Green Grass 1'},
 { id: 13, name: 'Chuck', address: 'Main Road 989'},
 { id: 4, name: 'Hannah', address: 'Mountain 21'},
 { id: 1, name: 'John', address: 'Higheay 71'},
 { id: 5, name: 'Michael', address: 'Valley 345'},
 { id: 2, name: 'Peter', address: 'Lowstreet 4'},
 { id: 8, name: 'Richard', address: 'Sky st 331'},
 { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},
 { id: 9, name: 'Susan', address: 'One way 98'},
 { id: 10, name: 'Vicky', address: 'Yellow Garden 2'},
 { id: 14, name: 'Viola', address: 'Sideway 1633'},
 { id: 12, name: 'William', address: 'Central st 954'}
]
```

მეორე შემთხვევაში კი:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM customers ORDER BY name
DESC", function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

შედეგი ასეთი იქნება:

```
[
  { id: 12, name: 'William', address: 'Central st 954'},
  { id: 14, name: 'Viola', address: 'Sideway 1633'},
  { id: 10, name: 'Vicky', address: 'Yellow Garden 2'},
  { id: 9, name: 'Susan', address: 'One way 98'},
  { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},
  { id: 8, name: 'Richard', address: 'Sky st 331'},
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},
  { id: 5, name: 'Michael', address: 'Valley 345'},
  { id: 1, name: 'John', address: 'Higheay 71'},
  { id: 4, name: 'Hannah', address: 'Mountain 21'},
  { id: 13, name: 'Chuck', address: 'Main Road 989'},
  { id: 7, name: 'Betty', address: 'Green Grass 1'},
  { id: 11, name: 'Ben', address: 'Park Lane 38'},
  { id: 3, name: 'Amy', address: 'Apple st 652'}
]
```

2.12.მონაცემების ამოგდება

ცხრილიდან მონაცემების ამოგდება ხდება "DELETE FROM" ოპერატორით.

ქვემოთ მოყვანილ მაგალითში ნადგურდება ყველა ის ჩანაწერი, რომელთა მისამართის ველის მნიშვნელობა არის "Mountain 21":

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});
con.connect(function(err) {
  if (err) throw err;
  var sql = "DELETE FROM customers WHERE address = 'Mountain
21'";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Number of records deleted: " + result.affectedRows);
  });
});
```

მოთხოვნის შესრულებას მოჰყვება შედეგის ობიექტის დაბრუნებაც, რასაც ამ შემთხვევაში ასეთი სახე აქვს:

```
{
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0,
  serverStatus: 34,
  warningCount: 0,
  message: '',
```

```
  protocol41: true,
  changedRows: 0
}
```

სტრიქონების რაოდენობა, რომლებზეც გავლენა მოახდინა ფაილის შესრულებაზე გაშვებამ, შესაძლებელია გავიგოთ ამ ბრძანების გაცემით:

```
console.log(result.affectedRows)
```

მოცემულ შემთხვევაში დაბრუნდება ეს შედეგი: 1

2.13. ცხრილის ამოგდება

ცხრილის ამოსაგდებად გამოიყენება "DROP TABLE" ოპერატორი.

ქვემოთ მოყვანილ მაგალითში ვანადგურებთ customers ცხრილს:

```
    var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "DROP TABLE customers";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table deleted");
  });
});
```

ვქმნით, ვიმახსოვრებთ და შესრულებაზე ვუშვებთ "demo_db_drop_table.js" ფაილს.

ვიღებთ შეტყობინებას:

```
Table deleted
```

თუ ცხრილი არ არსებობს ანდა უკვე ამოგდებულია, შეცდომის თავიდან ასაცილებლად ვიყენებთ IF EXISTS გასაღებურ სიტყვას:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "DROP TABLE IF EXISTS customers";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

შესაბამისი ფაილის შესრულებაზე გაშვების შემდეგ, თუ ფაილი ამ პროცედურამდე არსებობდა, ვიღებთ ასეთ შედეგს:

```
{
  fieldCount: 0,
  affectedRows: 0,
  insertId: 0,
  serverstatus: 2,
  warningCount: 0,
  message: '',
```

```
  protocol41: true,
  changedRows: 0
}
```

წინააღმდეგ შემთხვევაში შედეგი იქნება ოდნავ განსხვავებული:

```
{
  fieldCount: 0,
  affectedRows: 0,
  insertId: 0,
  serverstatus: 2,
  warningCount: 1,
  message: '',
  protocol41: true,
  changedRows: 0
}
```

2.14. მონაცემების განახლება

არსებულ ცხრილებში მონაცემების განახლება ხდება UPDATE ოპერატორით.

მოგვყავს მაგალითი, რომელშიც customers ცხრილში მისამართის მნიშვნელობა Valley 345 იცვლება ახალი, Canyon 12 მნიშვნელობით:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
```

```
  if (err) throw err;
  var sql = "UPDATE customers SET address = 'Canyon 123' WHERE
      address = 'Valley 345'";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result.affectedRows + " record(s) updated");
  });
});
```

ადრე აღწერილი წესებით ჩავატაროთ შესაბამისი ქმედებები. მათი შესრულების შემდეგ მონიტორზე გამოვა ასეთი ინფორმაცია:

```
1 record(s) updated
```

აქაც უნდა აღინიშნოს, რომ WHERE გასაღებური სიტყვის გამოტოვების შემთხვევაში მოხდება ყველა ჩანაწერის განახლება.

შედეგის ობიექტს ექნება ასეთი სახე:

```
{
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0,
  serverStatus: 34,
  warningCount: 0,
  message: '(Rows matched: 1 Changed: 1 Warnings: 0',
  protocol41: true,
  changedRows: 1
}
```

console.log(result.affectedRows) ბრძანებით კი მონიტორზე გამოისახება იმ ჩანაწერების რიცხვი, რომელთა განახლებაც მოხდა:

```
1
```

2.15. სერვერიდან გადმოცემული მონაცემების ლიმიტირება

როდესაც სერვერიდან გადმოცემული ინფორმაციის რაოდენობა მოცულობით დიდია, მის უკეთ აღსაქმელად მიმართავენ ლიმიტირებას. ამ დროს ჩანაწერები გადმოიცემა ნაწილ-ნაწილ, რაც ხდება LIMIT გასაღებური სიტყვის მეშვეობით.

მოვიყვანოთ მაგალითი:

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "SELECT * FROM customers LIMIT 5";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

კოდი დავიმახსოვროთ "demo_db_limit.js" სახელით და შესრულებაზე გავუშვათ:

```
C:\Users\Your Name>node demo_db_limit.js
```

მონიტორზე აისახება ასეთი შედეგი:

```
[
  { id: 1, name: 'John', address: 'Highway 71'},
  { id: 2, name: 'Peter', address: 'Lowstreet 4'},
```

```
  { id: 3, name: 'Amy', address: 'Apple st 652'},
  { id: 4, name: 'Hannah', address: 'Mountain 21'},
  { id: 5, name: 'Michael', address: 'Valley 345'}
]
```

დასაშვებია, ჩანაწერების სასურველი, მაგალითად, მესამე პოზიციიდან გამოყვანაც მათ რაოდენობაზე ასევე სასურველი შეზღუდვის დათქმით:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "SELECT * FROM customers LIMIT 5 OFFSET 2";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

შენიშვნა: *მიაქციეთ ყურადღება, რომ წანაცვლება "OFFSET 2" გულისხმობს სტარტის აღებას არა მეორე, არამედ სწორედ მესამე პოზიციიდან!*

შესაბამისი მანიპულაციების ჩატარების შემდეგ ვღებულობთ ასეთ შედეგს:

```
[
 { id: 3, name: 'Amy', address: 'Apple st 652'},
 { id: 4, name: 'Hannah', address: 'Mountain 21'},
```

```
  { id: 5, name: 'Michael', address: 'Valley 345'},
  { id: 6, name: 'Sandy', address: 'Ocean blvd 2'},
  { id: 7, name: 'Betty', address: 'Green Grass 1'}
]
```

შენიშვნა: *იმავე შედეგის მისაღებად დასაშვებია გამოვიყენოთ შემოკლებული სინტაქსი - გასაღებურ სიტყვებად არჩეული იქნეს "LIMIT 2, 5". აქ ყურადღება უნდა მიექცეს იმ გარემოებას, რომ რიცხვები ადგილებს ცვლიან!*

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "SELECT * FROM customers LIMIT 2, 5";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

2.16. ცხრილების ერთმანეთთან მიერთება

შესაძლებელია ორი და მეტი ცხრილის სტრიქონების გაერთიანება ამ ცხრილების დამაკავშირებელი სვეტის ბაზაზე, რისთვისაც გამოიყენება JOIN ოპერატორი.

ვაჩვენოთ მაგალითი - მიზნად დავისახოთ "users" და "products" ცხრილების გაერთიანება:

users

```
[
 { id: 1, name: 'John', favorite_product: 154},
 { id: 2, name: 'Peter', favorite_product: 154},
 { id: 3, name: 'Amy', favorite_product: 155},
 { id: 4, name: 'Hannah', favorite_product:},
 { id: 5, name: 'Michael', favorite_product:}
]
```

products

```
[
 { id: 154, name: 'Chocolate Heaven' },
 { id: 155, name: 'Tasty Lemons' },
 { id: 156, name: 'Vanilla Dreams' }
]
```

ზემოთ მოყვანილი ორი ცხრილის გაერთიანება შესაძლებელია ამგვარად - მათ დამაკავშირებლად პირველი ცხრილიდან ვიყენებთ favorite_product, ხოლო მეორედან - id ველს:

```
var mysql = require('mysql');

var con = mysql.createConnection({
 host: "localhost",
 user: "yourusername",
```

```
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  var sql = "SELECT users.name AS user, products.name AS favorite
        FROM users JOIN products ON users.favorite_product =
        products.id";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log(result);
  });
});
```

შენიშვნა: *შესაძლებელია, INNER JOIN-ის ნაცვლად JOIN-ის გამოყენებაც. შედეგები იქნება ერთნაირი:*

```
[
  { user: 'John', favorite: 'Chocolate Heaven' },
  { user: 'Peter', favorite: 'Chocolate Heaven' },
  { user: 'Amy', favorite: 'Tasty Lemons' }
]
```

ამრიგად, გვიბრუნდება მხოლოდ ისეთი ჩანაწერები, რომლებისთვისაც ორივე ცხრილში ხდება დამთხვევა.

2.16.1. მარცხნიდან მიერთება

დავუშვათ, გვსურს ინფორმაცია მოვიძიოთ ყველა მომხმარებლის შესახებ, მიუხედავად იმისა, გააჩნიათ თუ არა მათ თუნდაც ერთი ფავორიტი პროდუქტი.

ასეთი მიზნის მისაღწევად იყენებენ LEFT JOIN ოპერატორს:

```
SELECT users.name AS user,
products.name AS favorite
```

```
FROM users
LEFT JOIN products ON users.favorite_product = products.id
```

კოდის ფაილად გაფორმება და შესრულებაზე გაშვება მოგვცემს ასეთ შედეგს:

```
[
  { user: 'John', favorite: 'Chocolate Heaven' },
  { user: 'Peter', favorite: 'Chocolate Heaven' },
  { user: 'Amy', favorite: 'Tasty Lemons' },
  { user: 'Hannah', favorite: null },
  { user: 'Michael', favorite: null }
]
```

2.16.2. მარჯვნიდან მიერთება

ისეთ შემთხვევაში კი, როდესაც გვსურს ინფორმაცია მოვიძიოთ ყველა პროდუქტის და მათი მსურველების შესახებაც, თუ ასეთები არიან, ვიყენებთ RIGHT JOIN ოპერატორს:

```
SELECT users.name AS user,
products.name AS favorite
FROM users
RIGHT JOIN products ON users.favorite_product = products.id
```

ამჯერად შედეგს ექნება ასეთი სახე:

```
[
  { user: 'John', favorite: 'Chocolate Heaven' },
  { user: 'Peter', favorite: 'Chocolate Heaven' },
  { user: 'Amy', favorite: 'Tasty Lemons' },
  { user: null, favorite: 'Vanilla Dreams' }
]
```

ვხედავთ, რომ Hannah და Michael, ანუ ის მომხმარებელები, რომელთაც ფავორიტი პროდუქტი არ გააჩნიათ, გამოტანილ ინფორმაციაში არ ფიგურირებენ.

დანართი

Assert - Provides a set of assertion tests

მტკიცებების ტესტების კრებულით უზრუნველყოფა

Предоставляет набор тестов утверждений

Buffer - To handle binary data

ემსახურება ორობითი მონაცემების დამუშავებას

Для обработки двоичных данных

Child_process - To run a child process

შვილობითი პროცესის გაშვება

Для запуска дочернего процесса

Cluster - To split a single Node process into multiple processes

კვანძის ერთი პროცესი იყოფა რამდენიმე ნაწილად

Чтобы разделить один процесс узла на несколько процессов

Crypto - To handle OpenSSL cryptographic functions OpenSSL

კრიპტოგრაფიული ფუნქციების შესრულება

Для обработки криптографических функций OpenSSL

Dgram - Provides implementation of UDP datagram sockets

ახდენს UDP დეიტაგრამების სოკეტების რეალიზებას

Обеспечивает реализацию сокетов дейтаграмм UDP.

(სოკეტი - პროცესებს შორის მონაცემების მიმოცვლის ინტერფეისი)

Dns - To do DNS lookups and name resolution functions

DNS-ების მოძებნისა და სახელებზე ნებართვის ფუნქციების შესრულებისათვის

Для поиска DNS и функций разрешения имен

Events - To handle events

ხდომილობების დამუშავება

Для обработки событий

Fs - To handle the file system

ხდომილობები ფაილურ სისტემასთან მუშაობისათვის

События для работы с файловой системой

http - To make Node.js act as an HTTP server

Node.js-ის HTTP-სერვერად მუშაობის უზრუნველყოფა

Чтобы Node.js работал как HTTP-сервер

https - To make Node.js act as an HTTPS server

Node.js-ის HTTPS-სერვერად მუშაობის უზრუნველყოფა

Чтобы Node.js работал как HTTPS-сервер

Net - To create servers and clients

სერვერების და კლიენტების შესაქმნელად

Для создания серверов и клиентов

OS - Provides information about the operation system

იძლევა ინფორმაციას ოპერაციული სისტემის შესახებ

Предоставляет информацию об операционной системе

Path - To handle file paths

ფაილებისადმი გზების დამუშავება

Для обработки путей к файлам

Querystring - To handle URL query strings

URL მოთხოვნის ამსახველი სტრიქონების დამუშავება

Для обработки строк запроса URL

Readline - To handle readable streams one line at the time

ნაკადში შემავალი წაკითხვადი სტრიქონების სათითაოდ დამუშავება

Для обработки читаемых потоков по одной строке за раз

Stream - To handle streaming data

ნაკადური მონაცემების დამუშავება

Для обработки потоковых данных

String_decoder - To decode buffer objects into strings

ბუფერის ობიექტების დეკოდირება სტრიქონში გადაყვანით

Для декодирования объектов буфера в строки

Timers - To execute a function after a given number of milliseconds

ფუნქციების შესრულება მოცემული რაოდენობის მილიწამების გასვლის შემდეგ

Для выполнения функции через заданное количество миллисекунд

TlS - To implement TLS and SSL protocols

TLS და SSL პროტოკოლების (ოქმების) რეალიზება

Для реализации протоколов TLS и SSL

tty - Provides classes used by a text terminal

ტექსტური ტერმინალის უზრუნველყოფა შესაბამისი კლასებით

Предоставляет классы, используемые текстовым терминалом

url - To parse URL strings

URL-სტრიქონების განრჩევა-ანალიზისათვის

Для разбора строк URL

util - To access utility functions

სამსახურებრივ ფუნქციებთან შეღწევისათვის

Для доступа к служебным функциям

v8 - To access information about V8 (the JavaScript engine)

V8 (JavaScript-ის ამძრავი) შესახებ ინფორმაციის მიღებისათვის

Для доступа к информации о V8 (движок JavaScript)

vm - To compile JavaScript code in a virtual machine

ვირტუალურ მანქანაზე JavaScript-ის კოდის კომპილაციისათვის

Для компиляции кода JavaScript на виртуальной машине

zlib - To compress or decompress files

ფაილების შეკუმშვა-გახსნისათვის

Для сжатия или распаковки файлов

ლიტერატურა

1. https://www.w3schools.com/nodejs/default.asp
2. ღვინეფაძე გ. Javascript და მისი შესაძლებლობების განმავითარებელი თანამედროვე ტექნოლოგიები. სახელმძღვანელო. სტუ, საგამომცემლო სახლი: „ტექნიკური უნივერსიტეტი“. თბილისი, 2018.
3. ინტერაქტიურ რეჟიმში ვებ-ტექნოლოგიების საგნების შესწავლისათვის განკუთვნილი ელექტრონული სახელ-მძღვანელო - CD-7276.

Gela Ghvinepadze, Nino Chorkhauli

Web technologies (Node.js). Textbook

ISBN 978-9941-8-7219-5

გადაეცა წარმოებას 1.06 2024. ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 5,25. ტირაჟი 50 ეგზ.

(იბეჭდება ავტორის ხარჯით)

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი~,
თბილისი, მ.კოსტავას 77