



ORM
Gold - Chapter 5 - Topic 5

**Selamat datang di Chapter 5 Topik 5 online
course Fullstack Web dari Binar Academy!**





Yeay, sampai juga di topik terakhir!

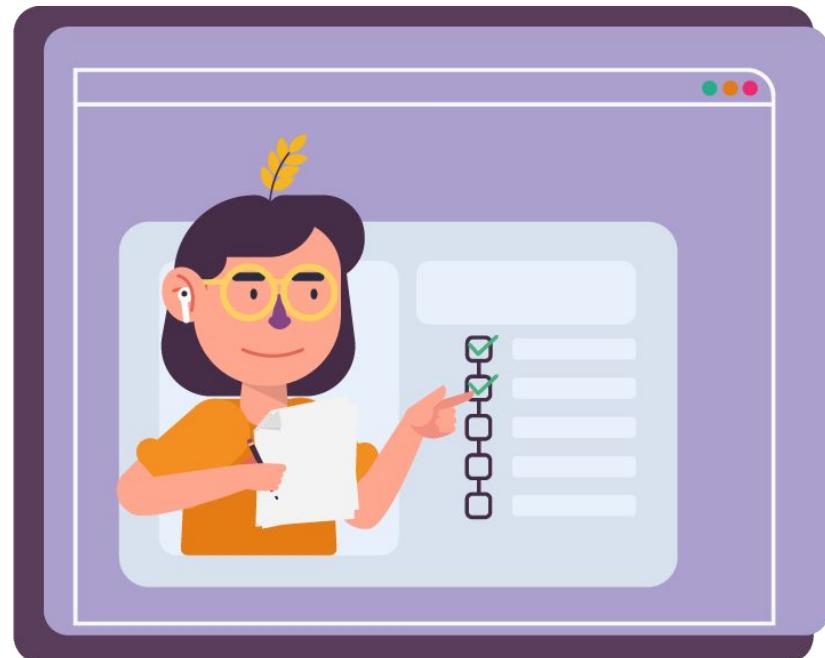
Di topik sebelumnya kita bahas tentang database dan juga penggunaan database SQL. Sekarang kita akan bahas tentang ORM.

Hmmm... benda apa ya itu ORM? Kita langsung bahas yuk!



Detailnya, kita bakal bahas hal-hal berikut:

- Mengenal ORM (Object Relational Mapping)
- Cara melakukan instalasi ORM
- Defnisi model
- Implementasi CRUD pada REST API dengan model





Di topik sebelumnya, kita udah tahu nih tentang jenis-jenis query dan pemaikaian query di SQL.

Sebenarnya ada cara lain lho untuk membuat database relasional menjadi lebih efisien. Nah, **ORM ini tools yang bikin data menjadi lebih efisien, yuk~**



Setelah belajar SQL mungkin kalian bertanya-tanya. Apa hubungannya materi SQL dengan materi NodeJS dan Express yang udah dibahas?

Sebenarnya **DBMS itu nantinya akan kita sambungkan dengan aplikasi NodeJS**. Nah, untuk menyambungkan dua aplikasi itu, kita butuh yang namanya **adapter**. Adapter ini common di bahasa pemrograman, ibaratnya kayak Google Translate gitu deh, karena adapter punya kemampuan untuk **menerjemahkan perintah JavaScript ke dalam bahasa SQL yang dapat dimengerti DBMS**.

Terus, cara kerjanya gimana? Simpelnya gini, kode JavaScript kita nantinya akan didefinisikan oleh fungsi/method untuk melakukan query.

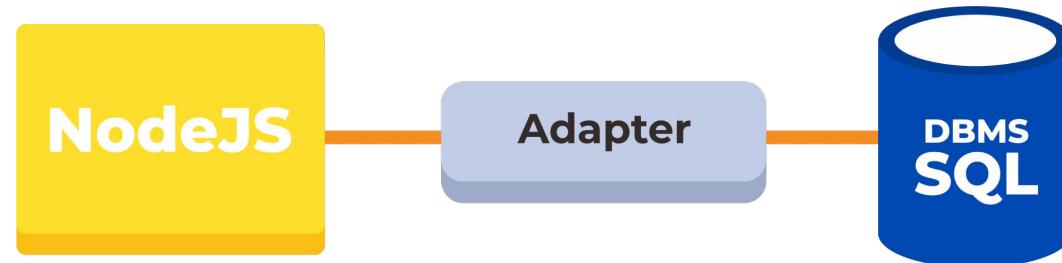


Adapter

Adapter yang sangat populer digunakan untuk menghubungkan NodeJS ke DBMS SQL, yaitu **Knex** dan **Sequelize**. Tapi, kita hanya akan bahas tentang Sequelize aja di kursus ini ya.

Sebenarnya Sequelize memiliki dependensi ke adapter yang sesungguhnya untuk menyambungkan NodeJS ke dalam DBMS. Berikut ini dependensi yang digunakan Sequelize dalam menyambungkan database:

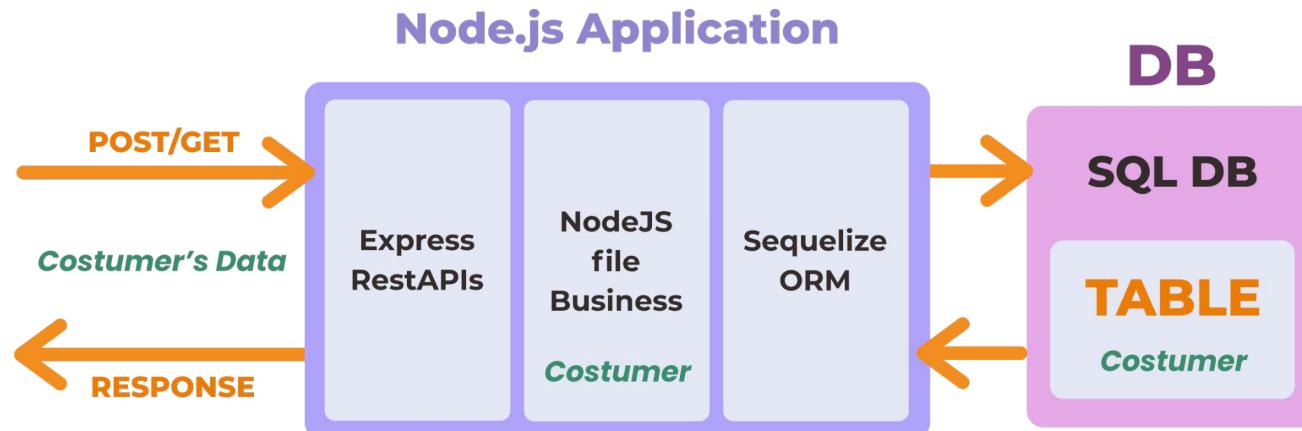
- **pg & pg-hstore** untuk PostgreSQL
- **mysql2** untuk MySQL
- **mariadb** untuk MariaDB
- **sqlite3** untuk SQLite3
- **tedious** untuk Microsoft SQL Server

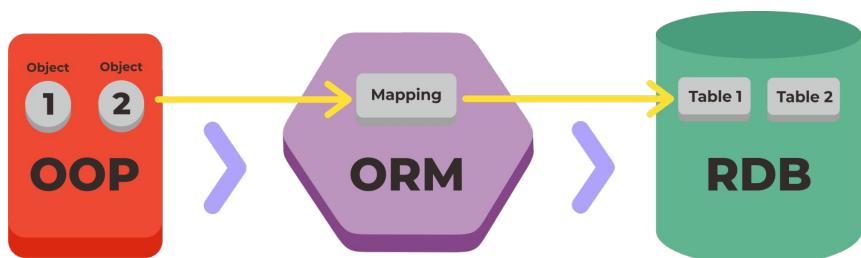


Nah, Sequelize selain berguna untuk menyambungkan ke DBMS, juga berguna untuk melakukan **mapping ke dalam setiap table di dalam database tersebut.**

Itulah alasan kenapa **Sequelize disebut sebagai ORM (Object Relational Mapping)**. Sequelize bisa dikatakan sebagai suatu package atau modul ORM yang dapat memetakan (mapping) setiap baris data di dalam suatu table database yang nantinya akan direpresentasikan sebagai objek di dalam NodeJS.

Oke, jadi sebenarnya **ORM (Object Relational Mapping)** itu apa ya?





ORM

ORM adalah suatu **metode atau teknik pemrograman yang digunakan untuk mengubah (konversi) data dari lingkungan bahasa pemrograman berorientasi objek (OOP) dengan lingkungan database relasional**. Hal ini berkaitan juga kan dengan pembahasan di awal tentang peran adapter sebagai penerjemah.

Nah, **peran ORM ini pada intinya sebagai penghubung atau dengan kata lain menjembatani kedua sistem yang berbeda dalam membuat aplikasi**.

ORM juga diciptakan untuk memudahkan developer dalam membuat aplikasi yang menggunakan *database relasional*, sehingga tugas kita jadi lebih efisien.



Kita ambil contoh dari topik sebelumnya ya, kalian masih ingat table articles kan?

```
CREATE TABLE articles (
    id BIGSERIAL PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    body TEXT NOT NULL,
    approved BOOLEAN NOT NULL DEFAULT FALSE,
);

// Untuk melihat table articles
SELECT * FROM articles;
```

Nah, jika kita query data table tersebut dari Sequelize, maka output-nya akan seperti ini:

```
// Contoh query
Article.find();
// output-nya berupa array karena kita SELECT * FROM
articles;
[
  // Baris data di dalam table articles direpresentasikan
  // sebagai objek
  {
    id: 1,
    title: "Hello World",
    body: "Lorem Ipsum Dolor Sit Amet"
  }
]
```



Selanjutnya, ada beberapa fitur atau interface yang akan kita pelajari di ORM ini, yaitu :

1. Migration
2. Model
3. Seeders





1. Migration

Berhubung kita menggunakan adapter Sequelize, aplikasi kita akan jadi lebih portable dan berjalan dengan baik di mesin/ komputer lain. Karena itu membuat kita jadi lebih efisien daripada melakukan setup manual untuk membuat aplikasi jalan di server.

Maksudnya begini, mungkin setup manual untuk 1-2 table masih oke. Tapi, kalau table-nya udah 15 atau lebih, apa kita harus buat table-nya satu per satu? Lalu, apa kita bisa menjamin kalau setup table di laptop kita udah sama persis dengan yang di server? Enggak, kan!? Nah, karena itulah untuk mengatasinya, kita bisa menggunakan **migrasi (migration)**.





Migration ini dapat kita pakai **untuk melakukan setup database aplikasi kita, dimanapun aplikasi itu akan dijalankan.** Nah, bagian yang kita setup dengan migration adalah yang berkaitan dengan DDL. Jadi, semua query yang berhubungan dengan DDL, kita definisikan melalui migration.

Kita bisa menggunakan migration ini untuk:

- Create Table (biasanya bersamaan ketika kita membuat model)
- Alter Table
- Drop Table





Bisa kita simpulkan bahwa migrations ini sebuah fitur atau paket schema builder yang berguna untuk **memudahkan kita dalam membuat skema database ketika terdapat perubahan dalam database kita.**

Nah, migrasi ini akan dijalankan secara terpisah dengan API kita. Kita dapat menjalankan migrasi dengan script: **npm run migrate**. Jadi, sebelum kita menyalakan API kita untuk listen ke HTTP Request, kita jalankan script tersebut terlebih dahulu.

```
root@MIC:/vagrant/vhost/learn-node# npm run migrate
> learn-node@1.0.0 migrate /vagrant/vhost/learn-node
> node_modules/sequelize-cli/bin/sequelize db:migrate --config tutorial/migrate/index.js --migrations-path tutorial/migrate/mysql/migrations

Sequelize (Node: 6.9.5, CLI: 2.7.0, ORM: 3.30.4)

Loaded configuration file "tutorial/migrate/index.js".
Using environment "development".
== 20170512091951-create-books: migrating =====
== 20170512091951-create-books: migrated (0.027s)
root@MIC:/vagrant/vhost/learn-node# 
```

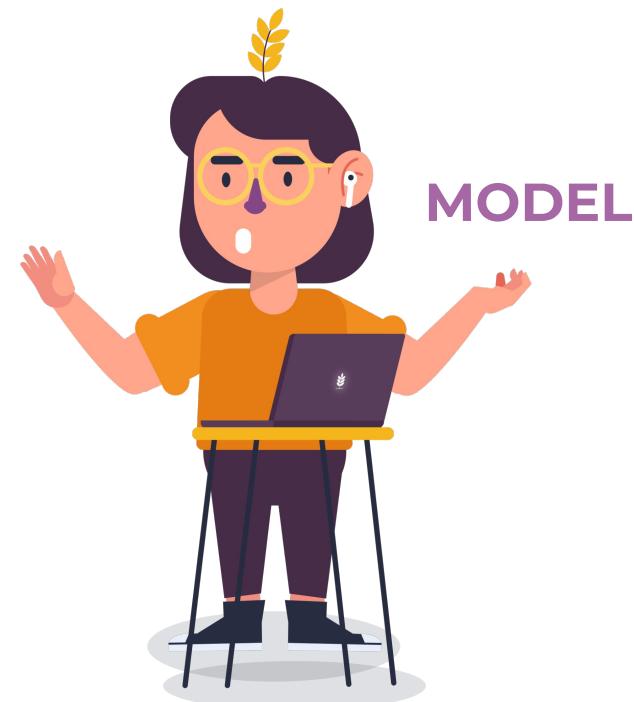


2. Model

Model ini bisa dikatakan sebuah fitur yang bertugas untuk **menghubungkan antara database dengan logic dalam mengambil data**. Jadi, dengan menggunakan model, kita bisa merepresentasikan suatu table di dalam database yang ada di aplikasi NodeJS.

Misalnya table articles yang udah kita buat sebelumnya, ketika kita mau menyambungkan aplikasi ke database tersebut, maka setiap transaksi atau query yang berkaitan dengan table tersebut akan selalu melalui model bernama **Article**.

Nah, biasanya ketika kita membuat API baru yang menggunakan ORM, maka kita akan membuat database baru yang benar-benar fresh untuk menghindari konflik yang nggak diinginkan. Model ini akan memiliki akses ke struktur table dan aksi-aksi yang berhubungan dengan DML.



3. Seeder

Seeder adalah **sebuah fitur yang digunakan untuk membuat data dummy atau data sembarang yang bersifat sementara**. Selain bisa kita implementasikan untuk tahap development, seeder juga bisa kita gunakan untuk membuat data-data yang bersifat tetap

Misalnya, data yang berisi daftar provinsi dan kota, karena data tersebut akan tetap sama. Seeder akan sangat memudahkan kita untuk menguji aplikasi dengan data awal tersebut.

Seeder ini mirip dengan migration, tapi di dalamnya kita melakukan query dengan jenis DML. Biasanya untuk insert, update, maupun delete suatu data. Seeder akan dijalankan secara terpisah dengan API kita. Contoh penggunaan seeder, biasanya ketika kita membuat user admin.





**Setelah kita paham tentang konsep ORM,
selanjutnya kita akan bahas gimana cara
instalasi ORM.**





Instalasi ORM dengan Sequelize

Yuk kita coba **instal ORM dengan Sequelize** di dalam aplikasi NodeJS! Nah, untungnya Sequelize ini udah menyediakan generator yang memudahkan kita untuk menggunakan interface atau fiturnya, baik itu untuk Migration, Model, dan Seeder.

Sequelize akan membuat package bernama **sequelize-cli**. Karena ada kata cli sudah pasti generator ini bekerja di terminal. Nah, kita bisa install generator ini dengan mudah, cukup ketikkan perintah disamping ini di terminal, bisa pakai package manager **npm atau yarn**. Penggunaan keyword **sudo** diperlukan karena kita butuh hak akses sebagai super user (root) untuk install package tersebut.

```
npm install sequelize-cli -g
```

```
isumi@isumizumi:~/Documents/orm$ sudo npm install sequelize-cli -g
[sudo] password for isumi:
/usr/bin/sequelize-cli -> /usr/lib/node_modules/sequelize-cli/lib/sequelize
/usr/bin/sequelize -> /usr/lib/node_modules/sequelize-cli/lib/sequelize
+ sequelize-cli@6.2.0
added 80 packages from 43 contributors in 3.174s
```

```
yarn add sequelize-cli -g
```

```
isumi@isumizumi:~/Documents/orm$ sudo yarn add sequelize-cli -g
yarn add v1.22.4
info No lockfile found.
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
success Saved lockfile.
success Saved 70 new dependencies.
info Direct dependencies
└─ sequelize-cli@6.2.0
info All dependencies
```



Setelah instal generator-nya, lalu kita harus melakukan beberapa *setup*. Ikuti petunjuk di gambar ini.

1. Inisialisasi *npm project*: **npm init -y**

```
isumi@isumizumi:~/Documents/orm$ npm init -y
Wrote to /home/isumi/Documents/orm/package.json:

{
  "name": "orm",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\" Error: no test specified \\ && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

2. Instal modul sequelize: **npm install sequelize**

```
isumi@isumizumi:~/Documents/orm$ npm install sequelize
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN orm@1.0.0 No description
npm WARN orm@1.0.0 No repository field.

+ sequelize@6.3.3
added 17 packages from 80 contributors and audited 17 packages in 2.576s
found 0 vulnerabilities
```



3. Inisialisasi adapter sequelize: **sequelize init**

```
isumi@isumizumi:~/Documents/orm$ sequelize init
Sequelize CLI [Node: 13.14.0, CLI: 6.2.0, ORM: 6.3.3]
Created "config/config.json"
Successfully created models folder at "/home/isumi/Documents/orm/models".
Successfully created migrations folder at "/home/isumi/Documents/orm/migrations".
Successfully created seeders folder at "/home/isumi/Documents/orm/seeders".
```

Ketika kita menjalankan perintah tersebut, maka sequelize akan secara otomatis melakukan generate atau membuat 4 folder sekaligus, yaitu:

- config
- models
- migrations
- seeders



Selanjutnya, kita lakukan konfigurasi untuk sequelize dengan cara mengedit file **config/config.json**.

```
{\n  \"development\": {\n    \"username\": \"namauerkamu\", \n    \"password\": \"passwordkamu\", \n    \"database\": \"namadatabasekamu_development\", \n    \"host\": \"localhost\", \n    \"dialect\": \"postgres\" \n  },\n  \"test\": {\n    \"username\": \"namauerkamu\", \n    \"password\": \"passwordkamu\", \n    \"database\": \"namadatabasekamu_staging\", \n    \"host\": \"localhost\", \n    \"dialect\": \"postgres\" \n  },\n  \"production\": {\n    \"username\": \"namauerkamu\", \n    \"password\": \"passwordkamu\", \n    \"database\": \"namadatabasekamu_production\", \n    \"host\": \"localhost\", \n    \"dialect\": \"postgres\" \n  } \n}
```



Setelah kita edit file **config.json**, sekarang kita buat database untuk aplikasi kita dengan menjalankan perintah **sequelize db:create**.

```
isumi@isumizumi:~/Documents/orm$ sequelize db:create
Sequelize CLI [Node: 13.14.0, CLI: 6.2.0, ORM: 6.3.3]
Loaded configuration file "config/config.json".
Using environment "development".
ERROR: Please install pg package manually
```

Tapi, kita akan mendapatkan error, karena harus *install adapter* dulu untuk database kita yaitu **pg**.

```
isumi@isumizumi:~/Documents/orm$ npm install pg
npm [WARN] orm@1.0.0 No description
npm [WARN] orm@1.0.0 No repository field.

+ pg@8.3.0
added 17 packages from 9 contributors and audited 34 packages in 2.329s
found 0 vulnerabilities
```



Kemudian, kita bisa coba jalankan lagi perintah **sequelize db:create**. Tapi, kalau terjadi error seperti gambar di bawah ini, maka kita perlu ingat-ingat lagi materi di *Chapter 6 Topic 1: Database*.

```
isumi@isumizumi:~/Documents/orm$ sequelize db:create
Sequelize CLI [Node: 13.14.0, CLI: 6.2.0, ORM: 6.3.3]
Loaded configuration file "config/config.json".
Using environment "development".
ERROR: connect ECONNREFUSED 127.0.0.1:5432
```

Nah, error itu terjadi karena kita belum *install* postgresql. Kita juga perlu masuk ke konsol postgres untuk membuat *user* dan mengatur *user* yang kita buat itu agar menjadi SUPERUSER. Jika hal itu udah di *setting*, maka kita bisa coba jalankan lagi perintah **sequelize db:create**. Kalau udah gak ada error, maka kita berhasil membuat *database* untuk aplikasi dan kita siap untuk memakai ORM-nya.

```
isumi@isumizumi:~/Documents/orm$ sequelize db:create
Sequelize CLI [Node: 13.14.0, CLI: 6.2.0, ORM: 6.3.3]
Loaded configuration file "config/config.json".
Using environment "development".
Database database_development created.
```



Oke lah kita udah berhasil install ORM, tapi itu belum cukup yaa gaes.

Kita masih perlu tahu juga nih tentang **model definition**, baik itu cara membuat model dan berbagai method yang bisa kita pakai. Yuk kita lanjutkan~



Kalau kita mau **membuat data artikel di dalam database, maka hal pertama yang harus kita lakukan adalah membuat migrasi, lalu membuat model-nya.** Nah, untungnya Sequelize baik banget deh, generator-nya udah disiapkan sedemikian rupa, spesial buat kita pakai. Untuk menggunakannya dengan perintah:

```
sequelize model:generate --name Article --attributes title:string, body:text, approved:boolean
```

```
isumi@isumizumi:~/Documents/orm$ sequelize model:generate --name Article --attributes title:string, body:text, approved:boolean
Sequelize CLI [Node: 13.14.0, CLI: 6.2.0, ORM: 6.3.3]
New model was created at /home/isumi/Documents/orm/models/article.js .
New migration was created at /home/isumi/Documents/orm/migrations/20200716111837-create-article.js .
```

Untuk nama *model*, konvensi-nya atau aturan penulisan yang digunakan adalah PascalCase. Nah, kalau pada contoh tersebut, nama *model* yang kita buat adalah **Article**, yang mana memiliki beberapa atribut yang menyertainya.



Setelah kita berhasil generate model, selanjutnya kita harus jalankan migrasi untuk setup database, agar bisa dipakai oleh model. Caranya dengan mengetikkan perintah: `sequelize db:migrate`.

```
tsumi@tsumizumi:~/Documents/orm$ sequelize db:migrate
Sequelize CLI [Node: 13.14.0, CLI: 6.2.0, ORM: 6.3.3]
Loaded configuration file "config/config.json".
Using environment "development".
== 20200716111837-create-article: migrating =====
== 20200716111837-create-article: migrated (0.019s)
```

Sebenarnya model memiliki beberapa method yang bisa kita pakai loh! Untuk mengetahui cara pakainya, bisa baca dokumentasi lengkapnya melalui link [ini](#). Nah, kalau di topik ini kita hanya akan bahas beberapa method yang sering banget dipakai, misalnya **create, update, find, dan destroy**.



1. Create

Kita bisa **melakukan insert data artikel dengan model yang baru aja kita buat**. Gimana caranya? Kita buat satu file bernama **create.js**, lalu kita import model-nya dan gunakan method create. Contoh kodennya bisa dilihat di samping ini.

Setelah itu, kita bisa jalankan file tersebut dengan mengetikkan perintah ini di terminal:

node create.js

```
const { Article } = require('./models')

Article.create({
  title: 'Hello World',
  body: 'Lorem Ipsum Dolor Sit Amet',
  approved: true
})
  .then(article => {
    console.log(article)
})
```



Output-nya nanti akan seperti gambar di bawah ini, yang berarti kita berhasil menambahkan data artikel dengan model yang baru aja kita buat di file **create.js**.

```
isumi@isumizumi:~/Documents/orm$ node create.js
Executing (default): INSERT INTO "Articles" ("id","title","body","approved","createdAt","updatedAt")
VALUES ($1,$2,$3,$4,$5) RETURNING "id","title","body","approved","createdAt","updatedAt";
Article {
  dataValues: {
    id: 1,
    title: 'Hello World',
    body: 'Lorem Ipsum Dolor Sit Amet',
    approved: true,
    updatedAt: 2020-07-17T06:19:23.532Z,
    createdAt: 2020-07-17T06:19:23.532Z
  },
  _previousDataValues: {
    title: 'Hello World',
    body: 'Lorem Ipsum Dolor Sit Amet',
    approved: true,
    id: 1,
    createdAt: 2020-07-17T06:19:23.532Z,
    updatedAt: 2020-07-17T06:19:23.532Z
  },
  _changed: Set(0) {},
  _options: {
    isNewRecord: true,
    _schema: null,
    _schemaDelimiter: '',
    attributes: undefined,
    include: undefined,
    raw: undefined,
    silent: undefined
  },
  isNewRecord: false
}
```



Kita bisa mengecek datanya di *database* kita, caranya dengan perintah : `SELECT * FROM "Articles";`
Output-nya nanti akan seperti gambar di bawah ini.

```
isumi@isumizumi:~/Documents/orm$ psql database_development
psql (10.12 (Ubuntu 10.12-0ubuntu0.18.04.1))
Type "help" for help.

database_development=# SELECT * FROM "Articles";
 id |      title       |          body          | approved |      createdAt       |      updatedAt
---+----------------+-----+-----+-----+
  1 | Hello World   | Lorem Ipsum Dolor Sit Amet | t        | 2020-07-17 13:19:23.532+07 | 2020-07-17 16:52:05.535+07
(1 row)
```



2. Update

Kita juga bisa **melakukan update dengan model**. Caranya cukup mudah, kita bisa pakai method bernama **.update**. Kita coba buat satu file bernama **update.js**, lalu kita import model-nya dan gunakan method update. Contoh kodennya bisa dilihat di samping ini. Setelah itu, kita bisa jalankan file tersebut dengan mengetikkan perintah ini di terminal:

node update.js



```
const { Article } = require('./models')
// Kita lakukan query terhadap artikel
// Artikel tersebut memiliki id yang bernilai
1
const query = {
  where: { id: 1 }
}

Article.update({
  approved: false
}, query)
  .then(() => {
    console.log("Artikel berhasil diupdate")
    process.exit()
  })
  .catch(err => {
    console.error("Gagal mengupdate artikel!")
  })
```



Output-nya nanti akan seperti gambar di bawah ini, yang berarti kita berhasil mengubah data artikel dengan *model* yang baru aja kita buat di file **update.js**.

```
isumi@isumizumi:~/Documents/orm$ node update.js
Executing (default): UPDATE "Articles" SET "approved"=$1, "updatedAt"=$2 WHERE "id" = $3
Artikel berhasil diupdate
```





3. Find

Model bisa kita gunakan juga untuk **mencari data di dalam database (DML)**. Ada banyak hal yang bisa kita bahas dari method find ini, tapi di materi ini kita hanya bahas 2 method, yaitu:

1. **Find All**

Untuk menampilkan semua data, kita bisa lakukan dengan method **.findAll** dan di dalamnya kita gak perlu menambahkan query apapun. Perintahnya seperti ini:

```
Article.findAll().then(article => console.log(article))
```

2. **Find by ID.**

Untuk *method* ini, kita harus tambahkan *query* secara spesifik seperti ini:

```
Article.findOne({  
    where: { id: 1 }  
})  
.then(article => console.log(article))
```

Nah, di dalam *method* *find* ini, kita bisa *explore* lebih dalam tentang berbagai operator (OP). OP ini membantu kita dalam melakukan *query* di dalam Sequelize. Referensinya ada di link [ini](#).



4. Destroy

Untuk melakukan *delete* data, kita bisa menggunakan *method destroy*.

```
Article.destroy({
  where: {
    approved: false
  }
})
  .then(() => console.log("Artikel yang
belum di approve sudah dihapus"))
```

Destroy





Setelah penggunaan model udah kita ketahui, berarti sekarang kita bisa coba implementasi CRUD di REST API dengan model.

Tapi, gimana ya caranya?

Hmmm.. Mikir dulu deh biar otaknya bisa olahpikir~





CRUD di REST API

Untuk membuat CRUD di Express yang sudah mengimplementasikan model, caranya bisa dikatakan sangat mudah. Nah, contoh API-nya bisa mengikuti kode disamping ini.

Pada contoh ini, kita membuat file bernama **index.js**. Kita hanya perlu meng-handle request-nya dan arahkan ke method yang tepat.

Misalnya, kita gunakan method API **get** untuk mengarahkan ke route yang diinginkan yaitu /articles, lalu kita perlu menggunakan method **findAll** untuk mendapatkan semua artikel. Sedangkan untuk mendapatkan artikel berdasarkan **id**, kita arahkan dulu route-nya ke /articles/:id, lalu gunakan method **findOne** untuk mendapatkan parameter id-nya.

```
// index.js
const express = require('express')
const app = express()
const { Article } = require('../models')

app.use(express.json())

// GET all articles
app.get('/articles', (req, res) => {
  Article.findAll()
    .then(articles => {
      res.status(200).json(articles)
    })
})

// GET article by ID
app.get('/articles/:id', (req, res) => {
  Article.findOne({
    where: { id: req.params.id }
  })
    .then(article => {
      res.status(200).json(article)
    })
})
```



```
● ● ●  
  
// POST an article  
app.post('/articles', (req, res) => {  
  Article.create({  
    title: req.body.title,  
    body: req.body.body,  
    approved: req.body.approved  
  })  
  .then(article => {  
    res.status(201).json(article)  
  }) .catch(err => {  
    res.status(422).json("Can't create  
article")  
  })
```

Jika kita ingin menambah data, kita bisa gunakan method API **post** untuk mengarahkan ke route yang diinginkan yaitu /articles, lalu kita perlu menggunakan method **create** untuk insert data artikel yang diinginkan, misalnya title, body, dan approved.



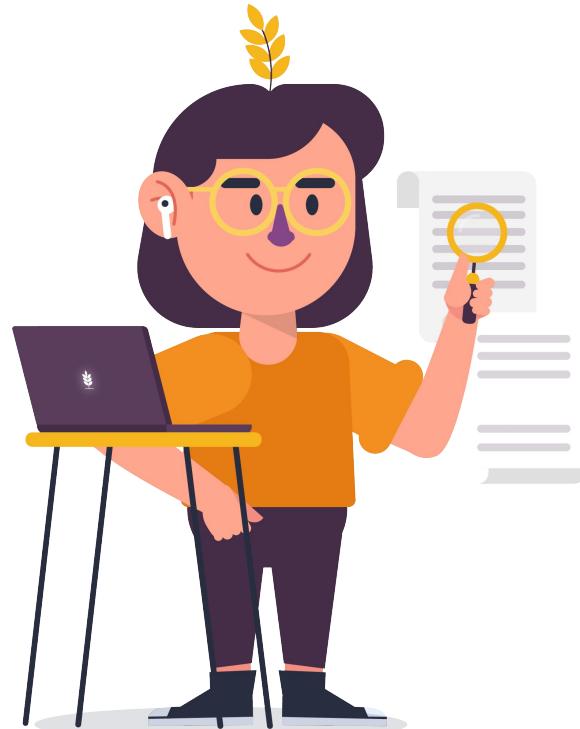
Lalu, gimana kalau kita mau edit suatu data?

Kita bisa gunakan method API **put** untuk mengarahkan ke route yang diinginkan yaitu /articles/:id, lalu kita perlu menggunakan method **update** untuk memperbarui data artikel berdasarkan id artikel.

```
// PUT an article
app.put('/articles/:id', (req, res) => {
  Article.update({
    title: req.body.title,
    body: req.body.body,
    approved: req.body.approved
  }, {
    where: { id: req.params.id }
  })
  .then(article => {
    res.status(201).json(article)
  }) .catch(err => {
    res.status(422).json("Can't create
article")
  })
})
```

Referensi buat kamu yang suka kepo-kepo~

- <https://sequelize.org/>
- <https://www.codepolitan.com/berkenalan-dengan-object-relational-document-graph-mapper>
- <https://www.codepolitan.com/mengapa-menggunakan-orm-dalam-membuat-aplikasi-android-59d5a3928887e>
- <https://www.crudpro.com/2020/02/belajar-membuat-web-menggunakan-framework-express-js-part-3.html>
- <https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a>
- <https://www.educba.com/what-is-orm/>
- <https://medium.com/skyshidigital/membuat-restful-api-menggunakan-express-dan-sequelize-ef0e10da36ff>



Naahhh....

Beres semua pembahasan chapter ini,
mulai dari pembahasan Express.Js, Restful
API, Database dan topik terakhir ini
adalah ORM

Kita istirahat dulu sebelum masuk chapter
berikutnya.

Sayonara bersama Sabrina ~



Terima Kasih!



Chapter ✓

completed