

SEQUELIZE

CSE5006 – LAB 6

REPOSITORY:

[HTTPS://GITHUB.COM/CSE5006/LAB-6](https://github.com/CSE5006/LAB-6)

TABLE OF CONTENTS

Outline	2
1. Get Everything Running	3
1.1. Docker Compose Up	3
1.2. Enter Container Terminal	4
1.3. Login to Database	4
1.4. Show Tables	4
1.4. SEQUELIZE CONFIGURATION	5
2. Working with a Single Table/Model Database	6
2.1. Foreword.....	6
2.2. Running the JavaScript with Node	6
2.3. Asynchronous Errors	7
2.4. Articles Table.....	9
2.5. Table Information.....	9
2.6. Selecting Data	10
2.7. Run the Queries	10
2.8. Exercise 1	11
2.9. Exercise 2	11
3. One-to-many (1:N) Relationships	12
3.1. Foreword.....	12
3.2. Exercise 3	13
3.3. Exercise 4	14
3.4. Exercise 5	14
4. Cleaning Up.....	15
4.1. Removing The Container.....	15
5. Change RDBMS	15

OUTLINE

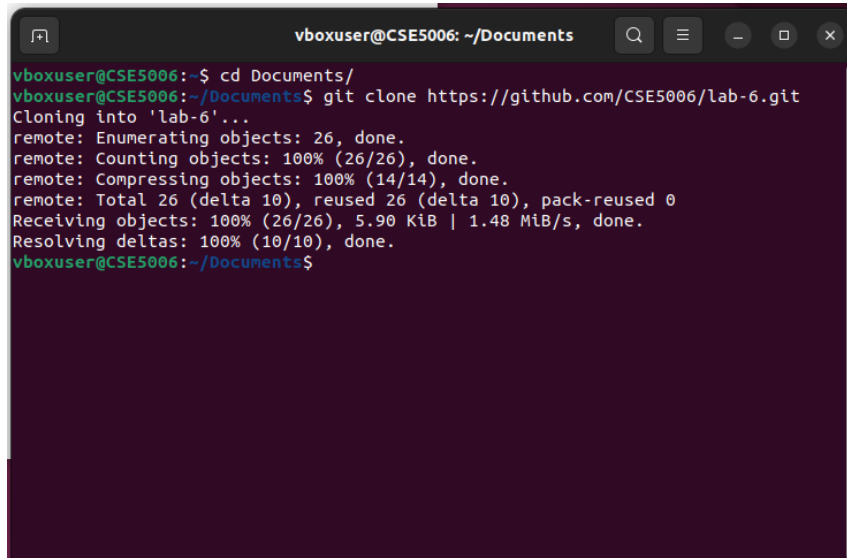
In this lab you will practice the following:

1. Creating database models
2. Querying models
3. Specifying associations between models

1. GET EVERYTHING RUNNING

1.1. DOCKER COMPOSE UP

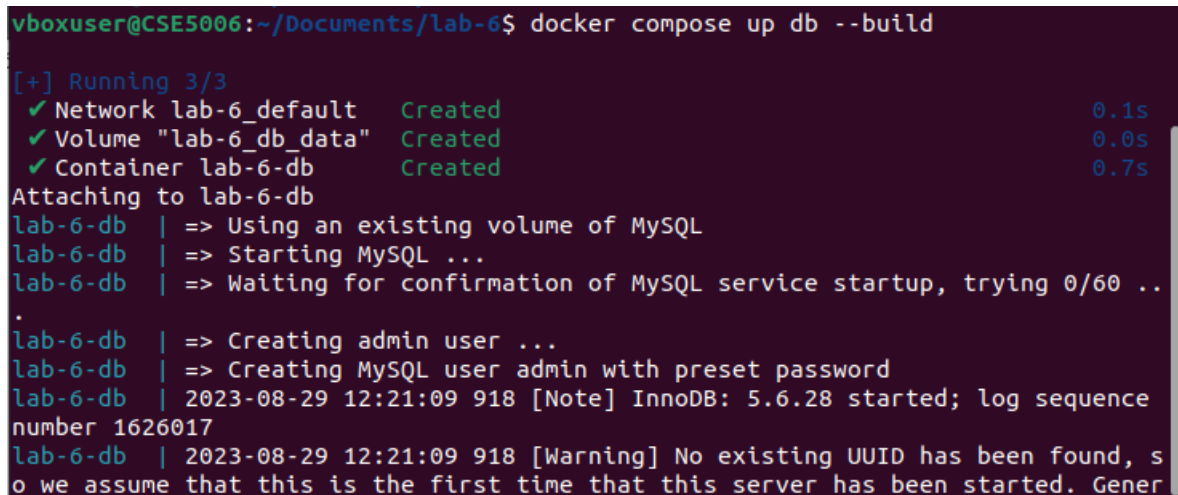
Clone the repository locally before you start working with this lab.

A terminal window titled 'vboxuser@CSE5006: ~/Documents' showing the command 'git clone https://github.com/CSE5006/lab-6.git' and its output. The output shows the cloning process, including enumerating, counting, and compressing objects, and resolving deltas. The terminal text is as follows:

```
vboxuser@CSE5006:~/Documents$ git clone https://github.com/CSE5006/lab-6.git
Cloning into 'lab-6'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 26 (delta 10), reused 26 (delta 10), pack-reused 0
Receiving objects: 100% (26/26), 5.90 KiB | 1.48 MiB/s, done.
Resolving deltas: 100% (10/10), done.
vboxuser@CSE5006:~/Documents$
```

Go into the **lab-6/** directory (where the **docker-compose.yml** file can be found). Type the following command to spin up the MySQL database Docker container:

```
docker compose up db --build
```

A terminal window titled 'vboxuser@CSE5006:~/Documents/lab-6\$' showing the command 'docker compose up db --build' and its output. The output shows the creation of the network, volume, and container, and the startup of the MySQL service. The terminal text is as follows:

```
vboxuser@CSE5006:~/Documents/lab-6$ docker compose up db --build

[+] Running 3/3
 ✓ Network lab-6_default      Created          0.1s
 ✓ Volume "lab-6_db_data"     Created          0.0s
 ✓ Container lab-6-db         Created          0.7s
Attaching to lab-6-db
lab-6-db | => Using an existing volume of MySQL
lab-6-db | => Starting MySQL ...
lab-6-db | => Waiting for confirmation of MySQL service startup, trying 0/60 ..
.
lab-6-db | => Creating admin user ...
lab-6-db | => Creating MySQL user with preset password
lab-6-db | 2023-08-29 12:21:09 918 [Note] InnoDB: 5.6.28 started; log sequence
number 1626017
lab-6-db | 2023-08-29 12:21:09 918 [Warning] No existing UUID has been found, s
o we assume that this is the first time that this server has been started. Gener
```

Wait until you see the following output

```

lab-6-db | 
lab-6-db | Please remember to change the above password as soon as possible!
lab-6-db | MySQL user 'root' has no password but only allows local connections
lab-6-db | =====
====
lab-6-db | Creating MySQL database development_db
lab-6-db | Database created!
lab-6-db | tail -F $LOG

```

1.2. ENTER CONTAINER TERMINAL

Start a new terminal where we are going to log into the MySQL database so we can see what is happening as we start inserting data and querying the database via Sequelize. Type the following (you can do this in any directory):

```
docker exec -it lab-6-db bash
```

```

vboxuser@CSE5006:~/Documents/lab-6$ docker exec -it lab-6-db bash
root@d53e877f96d6:/# 

```

1.3. LOGIN TO DATABASE

Now you are inside the container that is running the MySQL database! Let's log into the MySQL database. Type the following:

```
mysql --user=$MYSQL_USER --password=$MYSQL_PASS development_db
```

```

root@d53e877f96d6:/# mysql --user=$MYSQL_USER --password=$MYSQL_PASS development
_db
Warning: Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.6.28-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 

```

1.4. SHOW TABLES

Now you are in the MySQL database. Try to see if there are currently any tables in the database by typing in the following:

```
SHOW TABLES;
```

```
mysql> show tables;
Empty set (0.00 sec)

mysql> █
```

Not surprisingly there are currently no tables there, since we just created a brand new database. The following exercises will put tables and records in the database. We can monitor what is being inserted as we go along, so leave this terminal open for the rest of the lab.

1.4. SEQUELIZE CONFIGURATION

The Sequelize configuration is stored in `\backend\connect_db.js` file. This file contains the host and port of the database server, username and password. As you can see, the database server configurations are taken from the environment variable, where the data was loaded from `mysql.env` file.

```
JS connect_db.js X
backend > JS connect_db.js > ...
1  /**
2   * Requiring this file will connect to the database and return the
3   * Sequelize database connection object.
4   */
5
6  const Sequelize = require('sequelize');
7
8  const dbConfig = {
9    host: process.env.MYSQL_REMOTE_HOST,
10   port: process.env.MYSQL_REMOTE_PORT,
11   // here we are selecting mysql as the database type we will be using
12   dialect: 'mysql'
13 };
14
15 // Here we connect to the database
16 const db = new Sequelize(
17   'development_db',
18   process.env.MYSQL_USER,
19   process.env.MYSQL_PASS,
20   dbConfig);
21
22 module.exports = db;
23
```

The screenshot shows the VS Code interface. On the left, the Explorer panel displays the project structure under the 'SEQUELIZE' workspace. The 'backend' folder is expanded, showing files like `connect_db.js`, `Dockerfile`, `package.json`, `populate_data1.js`, `populate_data2.js`, `querying1.js`, and `querying2.js`. The 'env' folder is also expanded, showing `mysql.env` (selected) and `docker-compose.yml`. On the right, the Environment Variable panel for `mysql.env` is open, displaying the following configuration:

```
env > mysql.env
1  MYSQL_USER=admin
2  MYSQL_PASS=af84de4d1b8a247e8ef9925d273d3ec549782c76be096217
3  MYSQL_REMOTE_HOST=db
4  MYSQL_REMOTE_PORT=3306
5
```

The database environment variable file is loaded through the docker-compose

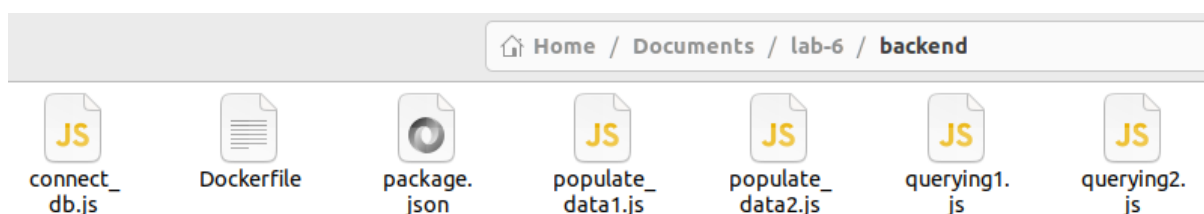
```
docker-compose.yml X
docker-compose.yml
1  version: "2"
2  services:
3    backend:
4      build: backend
5      environment:
6        - NODE_ENV=development
7      volumes:
8        - "./backend:/app"
9      env_file:
10       - ./env/mysql.env
11      links:
12       - db
13    db:
14      image: tutum/mysql:5.6
15      environment:
16        - ON_CREATE_DB=development_db
17      env_file:
18        - ./env/mysql.env
19      volumes:
20        - "db_data:/var/lib/mysql"
21  volumes:
22    db_data:
23      external: false
24
```

2. WORKING WITH A SINGLE TABLE/MODEL DATABASE

2.1. FOREWORD

We have divided this lab into three sections. Each section has one file that populates the database and another one that queries it. This section uses the files **populate_data1.js** and **querying1.js**.

The files can be found in the "**backend**" folder.



2.2. RUNNING THE JAVASCRIPT WITH NODE

Let's get started by running the first JavaScript file with Node (this will cause the code inside the **populate_data1.js** file to run). Start a new terminal and go into the **backend/** directory and type the following in it:

```
docker compose run --rm backend node populate_data1.js
```

```
vboxuser@CSE5006:~/Documents/lab-6$ cd backend/
vboxuser@CSE5006:~/Documents/lab-6/backend$ docker compose run --rm backend node
populate_data1.js
[+] Creating 1/0
✓ Container lab-6-db Running 0.0s
[+] Building 26.7s (9/14)
=> => transferring context: 2B 0.0s
=> [backend internal] load metadata for docker.io/library/node:18.16.0-a 2.8s
=> [backend 1/10] FROM docker.io/library/node:18.16.0-alpine@sha256:90 16.0s
=> => resolve docker.io/library/node:18.16.0-alpine@sha256:9036ddb8252ba 0.1s
=> => sha256:9036ddb8252ba7089c2c83eb2b0dcdf74ff1069e8dd 1.43kB / 1.43kB 0.0s
=> => sha256:5e7724fe034d5693ffd3a2f925d6b1feea3f86096a4 1.16kB / 1.16kB 0.0s
=> => sha256:6dcce61929307b8f11b3a7c67feb4e000987f5f3810 6.73kB / 6.73kB 0.0s
=> => sha256:31e352740f534f9ad170f75378a84fe453d6156e407 3.40MB / 3.40MB 2.3s
=> => sha256:c017600940c61ad955e20e7dacd301feb287ddddd 47.48MB / 47.48MB 13.4s
=> => sha256:c9f8586f07bd2ca2364211d26e40d5bb881cf547be7 2.34MB / 2.34MB 2.9s
=> => extracting sha256:31e352740f534f9ad170f75378a84fe453d6156e40700b88 0.2s
=> => sha256:ee16df044bfcf0a8d28e8a4488c5ee7013f2c49e8d061dd 450B / 450B 2.6s
=> => extracting sha256:c017600940c61ad955e20e7dacd301feb287ddddd442f63d9 2.1s
=> => extracting sha256:c9f8586f07bd2ca2364211d26e40d5bb881cf547be76e7e0 0.1s
```

2.3. ASYNCHRONOUS ERRORS

Oh no, it doesn't work!

```
},
original: Error: Table 'development_db.articles' doesn't exist
  at Packet.asError (/deps/node_modules/mysql2/lib/packets/packet.js:728:17)
  at Prepare.execute (/deps/node_modules/mysql2/lib/commands/command.js:29:2
6)
  at Connection.handlePacket (/deps/node_modules/mysql2/lib/connection.js:47
8:34)
  at PacketParser.onPacket (/deps/node_modules/mysql2/lib/connection.js:97:1
2)
  at PacketParser.executeStart (/deps/node_modules/mysql2/lib/packet_parser.
js:75:16)
  at Socket.<anonymous> (/deps/node_modules/mysql2/lib/connection.js:104:25)
  at Socket.emit (node:events:513:28)
  at addChunk (node:internal/streams/readable:324:12)
  at readableAddChunk (node:internal/streams/readable:297:9)
  at Readable.push (node:internal/streams/readable:234:10) {
  code: 'ER_NO_SUCH_TABLE',
  errno: 1146,
  sqlState: '42S02',
  sqlMessage: "Table 'development_db.articles' doesn't exist",
  sql: 'INSERT INTO `articles` (`id`,`title`,`content`,`createdAt`,`updatedAt`
) VALUES (DEFAULT,?,?,?,?);',
  parameters: [
```

Read the comments in **populate_data1.js**. The problem is that the code is attempting to insert data before the tables have finished being created! Not a good idea. This is because a common pattern in JavaScript is to execute code asynchronously (meaning the next line runs without having to wait for the previous one to finish).

Let's now fix it by uncommenting the ".then" function call and the close brackets at the end of the file.

```
23 // Currently the then function is commented out. This does not work properly,
24 // and you will get an error message. Uncomment the then function here and the
25 // at end of the program (the closing braces) and see what happens.
26 db.sync({ force: true }).then(() => {
27
28   // When creating a record the id attribute is automatically generated and put into the database.
29   Article.create({
30     title: 'War and Peace',
31     content: 'A book about fighting and then making up.'
32   });
33
34   Article.create({
35     title: 'Sequelize for dummies',
36     content: 'Writing lots of cool javascript code that get turned into SQL.'
37   });
38
39   Article.create({
40     title: 'I like tomatoes',
41     content: 'The story about the adventures of a tomato lover.'
42   });
43
44   Article.create({
45     title: 'PHP for dummies',
46     content: 'Why PHP is so so so bad at backend stuff. Why you should use express node.'
47   });
48
49   Article.create({
50     title: 'The lovely car',
51     content: 'How a car changed his life forever.'
52   });
53
54 });
55
56 // NOTE: To keep this particular example simple, we don't close the database
57 // connection before exiting. See populate_data2.js for an example which
58 // exits gracefully.
59
```

This will force the **Article.create** lines to execute after sync has completed. Save the file and run it again.

```
vboxuser@CSE5006:~/Documents/lab-6/backend$ docker compose run --rm backend node populate_data1.js
[+] Creating 1/0
  ✓ Container lab-6-db Running 0.0s
(node:7) [SEQUELIZE0006] DeprecationWarning: This database engine version is not supported, please update your database server. More information https://github.com/sequelize/sequelize/blob/main/ENGINE.md
(Use 'node --trace-deprecation ...' to show where the warning was created)
Executing (default): DROP TABLE IF EXISTS `articles`;
Executing (default): SELECT CONSTRAINT_NAME as constraint_name,CONSTRAINT_NAME as constraintName,CONSTRAINT_SCHEMA as constraintSchema,CONSTRAINT_SCHEMA as constraintCatalog,TABLE_NAME as tableName,TABLE_SCHEMA as tableSchema,TABLE_SCHEMA as tableCatalog,COLUMN_NAME as columnName,REFERENCED_TABLE_SCHEMA as referencedTableSchema,REFERENCED_TABLE_SCHEMA as referencedTableCatalog,REFERENCED_TABLE_NAME as referencedTableName,REFERENCED_COLUMN_NAME as referencedColumnName FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE where TABLE_NAME = 'articles' AND CONSTRAINT_NAME!='PRIMARY' AND CONSTRAINT_SCHEMA='development_db' AND REFERENCED_TABLE_NAME IS NOT NULL;
Executing (default): DROP TABLE IF EXISTS `articles`;
Executing (default): DROP TABLE IF EXISTS `articles`;
Executing (default): CREATE TABLE IF NOT EXISTS `articles` (`id` INTEGER NOT NULL auto_increment, `title` VARCHAR(255), `content` TEXT, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `articles`;
Executing (default): INSERT INTO `articles` (`id`,`title`,`content`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?,?,?,?);
Executing (default): INSERT INTO `articles` (`id`,`title`,`content`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?,?,?,?);
Executing (default): INSERT INTO `articles` (`id`,`title`,`content`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?,?,?,?);
Executing (default): INSERT INTO `articles` (`id`,`title`,`content`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?,?,?,?);
Executing (default): INSERT INTO `articles` (`id`,`title`,`content`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?,?,?,?);
vboxuser@CSE5006:~/Documents/lab-6/backend$
```


Note : If you only remove the then function, you will end up with this error:

```
$ docker-compose run --rm backend node populate_data1.js
Starting sequelize_db_1 ... done
/app/populate_data1.js:60
});
^
SyntaxError: Unexpected end of input
    at createScript (vm.js:74:10)
    at Object.runInThisContext (vm.js:116:10)
    at Module._compile (module.js:533:28)
    at Object.Module._extensions..js (module.js:580:10)
    at Module.load (module.js:503:32)
    at tryModuleLoad (module.js:466:12)
    at Function.Module._load (module.js:458:3)
    at Function.Module.runMain (module.js:605:10)
    at startup (bootstrap_node.js:158:16)
    at bootstrap_node.js:575:3
```

2.4. ARTICLES TABLE

Now go to the MySQL terminal that you logged into earlier. Type the following command:

```
SHOW TABLES;
```

```
mysql> show tables;
+-----+
| Tables_in_development_db |
+-----+
| articles                  |
+-----+
1 row in set (0.00 sec)

mysql>
```

You should now see a table called **articles**.

2.5. TABLE INFORMATION

You can get more information about the columns in a particular table very easily:

```
DESCRIBE articles;
```

```
mysql> describe articles;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(11) | NO | PRI | NULL | auto_increment |
| title | varchar(255) | YES | | NULL | |
| content | text | YES | | NULL | |
| createdAt | datetime | NO | | NULL | |
| updatedAt | datetime | NO | | NULL | |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> █
```

You should be able to see that the articles table has columns for **id**, **title**, and **content**, as well as creation and update timestamps.

2.6. SELECTING DATA

Let's take a look at what is actually stored in the articles table. Type the following in the MySQL terminal:

```
SELECT * FROM articles;
```

```
mysql> SELECT * FROM articles;
+-----+-----+-----+-----+-----+
| id | title | content | createdAt | updatedAt |
+-----+-----+-----+-----+-----+
| 1 | War and Peace | A book about fighting and then making up. | 2023-08-31 11:27:05 | 2023-08-31 11:27:05 |
| 2 | Sequelize for dummies | Writing lots of cool javascript code that get turned into SQL. | 2023-08-31 11:27:05 | 2023-08-31 11:27:05 |
| 3 | I like tomatoes | The story about the adventures of a tomato lover. | 2023-08-31 11:27:05 | 2023-08-31 11:27:05 |
| 4 | PHP for dummies | Why PHP is so so so bad at backend stuff. Why you should use express node. | 2023-08-31 11:27:05 | 2023-08-31 11:27:05 |
| 5 | The lovely car | How a car changed his life forever. | 2023-08-31 11:27:05 | 2023-08-31 11:27:05 |
+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

You should see 5 records in the database. Now look at the code inside the **populate_data1.js** file and see the correspondence between the code there and what you see in the table.

2.7. RUN THE QUERIES

Look at the comments and code in the querying1.js file, then actually run the queries:

```
docker compose run --rm backend node querying1.js
```

```
Executing (default): SELECT 'id', 'title', 'content', 'createdAt', 'updatedAt' FROM 'articles' AS 'articles' WHERE 'ar
ticles'.id = '[object Object]';
Executing (default): SELECT 'id', 'title', 'content', 'createdAt', 'updatedAt' FROM 'articles' AS 'articles';
# All articles after destroying
{
  id: 1,
  title: 'War and Peace',
  content: 'A book about fighting and then making up.',
  createdAt: 2023-08-31T11:27:05.000Z,
  updatedAt: 2023-08-31T11:27:05.000Z
}
{
  id: 2,
  title: 'Sequelize for dummies',
  content: 'Writing lots of cool javascript code that get turned into SQL.',
  createdAt: 2023-08-31T11:27:05.000Z,
  updatedAt: 2023-08-31T11:27:05.000Z
}
{
  id: 3,
  title: 'I like tomatoes',
  content: 'The story about the adventures of a tomato lover.',
  createdAt: 2023-08-31T11:27:05.000Z,
  updatedAt: 2023-08-31T11:27:05.000Z
}
```

Verify that there are only 3 rows left in the articles table.

```
mysql> mysql> select id, title, content from articles;
+-----+-----+-----+
| id | title | content |
+-----+-----+-----+
| 1 | War and Peace | A book about fighting and then making up. |
| 2 | Sequelize is the Worst ORM ever! | The story about the adventures of a tomato lover. |
| 3 | PHP for dummies | Why PHP is so so so bad at backend stuff. Why you should use express node. |
+-----+-----+-----+
3 rows in set (0.01 sec)
```

Now that you have gained some familiarity with the setup, let's test your knowledge. Complete the following exercises by writing queries at the end of the **querying1.js** file.

Notes: The developer of online libraries such as Sequelize will constantly update their features and syntax and operators. The latest sequelize syntax and operators can be seen in this link.

<https://sequelize.org/docs/v6/core-concepts/model-querying-basics/>

Please refer to this link to solve your Sequelize exercise.

2.8. EXERCISE 1

Print out the contents of articles whose **id** is either 1 or 3.

*Hint: use the **[Op.in]** operator, like **id: {[Op.in]: [1,3]}**.*

Expected output

```
Executing (default): SELECT `id`, `title`, `content`, `createdAt`, `updatedAt` FROM `articles` AS `articles` WHERE `articles`.`id` IN (1, 3);
ID = 1 or 3
{
  id: 1,
  title: 'War and Peace',
  content: 'A book about fighting and then making up.',
  createdAt: 2023-08-31T11:57:38.000Z,
  updatedAt: 2023-08-31T11:57:38.000Z
}
{
  id: 3,
  title: 'I like tomatoes',
  content: 'The story about the adventures of a tomato lover.',
  createdAt: 2023-08-31T11:57:38.000Z,
  updatedAt: 2023-08-31T11:57:38.000Z
}
```

2.9. EXERCISE 2

Retrieve the article record which has an **id** of 2 and update it so that its content is now "Sequelize is the worst ORM ever!".

*Hint: this is how you would update the **title** of an article: **article.update({title: "Blah"})**.*

This is the expected outcome:

```
Executing (default): SELECT `id`, `title`, `content`, `createdAt`, `updatedAt` FROM `articles` AS `articles` WHERE `articles`.`id` = 2;
Executing (default): SELECT `id`, `title`, `content`, `createdAt`, `updatedAt` FROM `articles` AS `articles` WHERE `articles`.`id` = 2;
# Article with id=2
{
  id: 2,
  title: 'Sequelize for dummies',
  content: 'Sequelize is the worst ORM ever!',
  createdAt: 2023-08-31T11:57:38.000Z,
  updatedAt: 2023-08-31T12:21:56.000Z
}
```

```
mysql> select * from articles where id=2;
+-----+-----+-----+-----+-----+
| id | title | content | createdAt | updatedAt |
+-----+-----+-----+-----+-----+
| 2 | Sequelize for dummies | Sequelize is the worst ORM ever! | 2022-08-22 10:22:42 | 2022-08-23 13:46:28 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

3. ONE-TO-MANY (1:N) RELATIONSHIPS

3.1. FOREWORD

Run the code in **populate_data2.js** and **querying2.js** (refer to the instructions in section §2).

```
vboxuser@CSE5006:~/Documents/lab-6/backend$ docker compose run --rm backend node populate_data2.js
[+] Creating 1/0
  ✓ Container lab-6-db Running 0.0s
(node:7) [SEQUELIZE0006] DeprecationWarning: This database engine version is not supported, please update your database server. More information https://github.com/sequelize/sequelize/blob/main/ENGINE.md
(Use 'node --trace-deprecation ...' to show where the warning was created)
Executing (default): DROP TABLE IF EXISTS 'employees';
Executing (default): DROP TABLE IF EXISTS 'companies';
Executing (default): SELECT CONSTRAINT_NAME as constraintName,CONSTRAINT_NAME as constraintName,CONSTRAINT_SCHEMA as constraintSchema,CONSTRAINT_SCHEMA as constraintCatalog,TABLE_NAME as tableName,TABLE_SCHEMA as tableSchema,TABLE_SCHEMA as tableCatalog,COLUMN_NAME as columnName,REFERENCED_TABLE_SCHEMA as referencedTableSchema,REFERENCED_TABLE_SCHEMA as referencedTableCatalog,REFERENCED_TABLE_NAME as referencedTableName,REFERENCED_COLUMN_NAME as referencedColumnName FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE where TABLE_NAME = 'companies' AND CONSTRAINT_NAME!='PRIMARY' AND CONSTRAINT_SCHEMA='development_db' AND REFERENCED_TABLE_NAME IS NOT NULL;
Executing (default): SELECT CONSTRAINT_NAME as constraintName,CONSTRAINT_NAME as constraintName,CONSTRAINT_SCHEMA as constraintSchema,CONSTRAINT_SCHEMA as constraintCatalog,TABLE_NAME as tableName,TABLE_SCHEMA as tableSchema,TABLE_SCHEMA as tableCatalog,COLUMN_NAME as columnName,REFERENCED_TABLE_SCHEMA as referencedTableSchema,REFERENCED_TABLE_SCHEMA as referencedTableCatalog,REFERENCED_TABLE_NAME as referencedTableName,REFERENCED_COLUMN_NAME as referencedColumnName FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE where TABLE_NAME = 'employees' AND CONSTRAINT_NAME!='PRIMARY' AND CONSTRAINT_SCHEMA='development_db' AND REFERENCED_TABLE_NAME IS NOT NULL;
Executing (default): DROP TABLE IF EXISTS 'companies';
Executing (default): DROP TABLE IF EXISTS 'employees';
Executing (default): DROP TABLE IF EXISTS 'companies';
Executing (default): CREATE TABLE IF NOT EXISTS 'companies' ('id' INTEGER NOT NULL auto_increment , 'name' VARCHAR(255), 'profit' FLOAT, 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL, PRIMARY KEY ('id')) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM 'companies';
Executing (default): DROP TABLE IF EXISTS 'employees';
Executing (default): CREATE TABLE IF NOT EXISTS 'employees' ('id' INTEGER NOT NULL auto_increment , 'name' VARCHAR(255), 'age' INTEGER, 'createdAt' DATETIME NOT NULL, 'updatedAt' DATETIME NOT NULL, 'companyId' INTEGER, PRIMARY KEY ('id'), FOREIGN KEY ('companyId') REFERENCES 'companies' ('id') ON DELETE SET NULL ON UPDATE CASCADE) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM 'employees';
Executing (default): INSERT INTO 'companies' ('id','name','profit','createdAt','updatedAt') VALUES (DEFAULT,?,?,?,?);
Executing (default): INSERT INTO 'companies' ('id','name','profit','createdAt','updatedAt') VALUES (DEFAULT,?,?,?,?);
Executing (default): INSERT INTO 'employees' ('id','name','age','createdAt','updatedAt','companyId') VALUES (DEFAULT,?,?,?,?,?);
Executing (default): INSERT INTO 'employees' ('id','name','age','createdAt','updatedAt','companyId') VALUES (DEFAULT,?,?,?,?,?);
Executing (default): INSERT INTO 'employees' ('id','name','age','createdAt','updatedAt','companyId') VALUES (DEFAULT,?,?,?,?,?);
vboxuser@CSE5006:~/Documents/lab-6/backend$
```

```
mysql> show tables;
+-----+-----+
| Tables_in_development_db |
+-----+-----+
| articles |
| companies |
| employees |
+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> select * from companies;
+-----+-----+-----+-----+-----+
| id | name | profit | createdAt | updatedAt |
+-----+-----+-----+-----+-----+
| 1 | Apple | 20202.1 | 2023-08-31 12:28:08 | 2023-08-31 12:28:08 |
| 2 | Google | 32 | 2023-08-31 12:28:08 | 2023-08-31 12:28:08 |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> select * from employees;
```

id	name	age	createdAt	updatedAt	companyId
1	John Smith	20	2023-08-31 12:28:08	2023-08-31 12:28:08	1
2	Peter Senior	10	2023-08-31 12:28:08	2023-08-31 12:28:08	1
3	Peter Rabbit	3	2023-08-31 12:28:08	2023-08-31 12:28:08	2

```
3 rows in set (0.00 sec)
```

```
vboxuser@CSE5006:~/Documents/lab-6/backend$ docker compose run --rm backend node querying2.js
[+] Creating 1/0
  ✓ Container lab-6-db Running 0.0s
(node:7) [SEQUELIZE0006] DeprecationWarning: This database engine version is not supported, please update your database server. More information https://github.com/sequelize/sequelize/blob/main/ENGINE.md
(Use `node --trace-deprecation ...` to show where the warning was created)
Executing (default): SELECT `employees`.`id`, `employees`.`name`, `employees`.`age`, `employees`.`createdAt`, `employees`.`updatedAt`, `employees`.`companyId`, `company`.`id` AS `company.id`, `company`.`name` AS `company.name`, `company`.`profit` AS `company.profit`, `company`.`createdAt` AS `company.createdAt`, `company`.`updatedAt` AS `company.updatedAt` FROM `employees` AS `employees` LEFT OUTER JOIN `companies` AS `company` ON `employees`.`companyId` = `company`.`id` LIMIT 1;
John Smith works at Apple
Executing (default): INSERT INTO `companies` (`id`,`name`,`profit`,`createdAt`,`updatedAt`) VALUES (DEFAULT,?,?,?,?);
Executing (default): SELECT `id`, `name`, `age`, `createdAt`, `updatedAt`, `companyId` FROM `employees` AS `employees` WHERE `employees`.`id` = 1;
Executing (default): UPDATE `employees` SET `companyId`=?,`updatedAt`=? WHERE `id` = ?
Executing (default): SELECT `id`, `name`, `age`, `createdAt`, `updatedAt`, `companyId` FROM `employees` AS `employees` WHERE `employees`.`id` = 1;
{
  id: 1,
  name: 'John Smith',
  age: 20,
  createdAt: 2023-08-31T12:28:08.000Z,
  updatedAt: 2023-08-31T12:32:01.000Z,
  companyId: 3
}
Executing (default): SELECT `id`, `name`, `age`, `createdAt`, `updatedAt`, `companyId` FROM `employees` AS `employees` ORDER BY `employees`.`age` DESC;
{
  id: 1,
  name: 'John Smith',
  age: 20,
  createdAt: 2023-08-31T12:28:08.000Z,
  updatedAt: 2023-08-31T12:32:01.000Z,
  companyId: 3
}
{
  id: 2,
  name: 'Peter Senior',
  age: 10,
  createdAt: 2023-08-31T12:28:08.000Z,
  updatedAt: 2023-08-31T12:28:08.000Z,
  companyId: 1
}
{
  id: 3,
  name: 'Peter Rabbit',
  age: 3,
  createdAt: 2023-08-31T12:28:08.000Z,
  updatedAt: 2023-08-31T12:28:08.000Z,
  companyId: 2
}
vboxuser@CSE5006:~/Documents/lab-6/backend$
```

3.2. EXERCISE 3

Find the name of the company that "Peter Rabbit" works for by searching for the name "Peter Rabbit" in the employees table.

*Hint: construct your query via the employees model (**Employees.findOne**), using the appropriate "where" clause and "include" option.*

Expected Outcome:

```
Executing (default): SELECT `employees`.`id`, `employees`.`name`, `employees`.`age`, `employees`.`createdAt`, `employees`.`updatedAt`, `employees`.`companyId`, `company`.`id` AS `company.id`, `company`.`name` AS `company.name`, `company`.`profit` AS `company.profit`, `company`.`createdAt` AS `company.createdAt`, `company`.`updatedAt` AS `company.updatedAt` FROM `employees` AS `employees` LEFT OUTER JOIN `companies` AS `company` ON `employees`.`companyId` = `company`.`id` WHERE `employees`.`name` = 'Peter Rabbit' LIMIT 1;
{
  id: 2,
  name: 'Google',
  profit: 32,
  createdAt: 2023-08-31T12:28:08.000Z,
  updatedAt: 2023-08-31T12:28:08.000Z
}
```

3.3. EXERCISE 4

Find the company with the highest profit and list its employees.

Hint: if you use `.findOne()` with an "order" clause, the sorting occurs before limiting the result to one record.

Expected outcome:

```
Executing (default): SELECT `companies`.*, `employees`.`id` AS `employees.id`, `employees`.`name` AS `employees.name`, `employees`.`age` AS `employees.age`, `employees`.`createdAt` AS `employees.createdAt`, `employees`.`updatedAt` AS `employees.updatedAt`, `employees`.`companyId` AS `employees.companyId` FROM (SELECT `companies`.`id`, `companies`.`name`, `companies`.`profit`, `companies`.`createdAt`, `companies`.`updatedAt` FROM `companies` AS `companies` ORDER BY `companies`.`profit` DESC LIMIT 1) AS `companies` LEFT OUTER JOIN `employees` AS `employees` ON `companies`.`id` = `employees`.`companyId` ORDER BY `companies`.`profit` DESC;
{
  id: 2,
  name: 'Peter Senior',
  age: 10,
  createdAt: 2023-08-31T12:28:08.000Z,
  updatedAt: 2023-08-31T12:28:08.000Z,
  companyId: 1
}
```

3.4. EXERCISE 5

Try inserting a new employee into the employees table.

For example, inserting a new employee in company id=1

Expected outcome:

```
Inserting new employee to company ID=1
Executing (default): INSERT INTO `employees` (`id`,`name`,`age`,`createdAt`,`updatedAt`,`companyId`) VALUES (DEFAULT,?,?,?,?);
vboxuser@CSE5006:~/Documents/lab-6$
```

```
mysql> select * from employees;
+----+-----+-----+-----+-----+-----+
| id | name       | age | createdAt           | updatedAt           | companyId |
+----+-----+-----+-----+-----+-----+
| 1  | John Smith | 20  | 2023-08-31 12:28:08 | 2023-08-31 12:36:50 | 5         |
| 2  | Peter Senior | 10  | 2023-08-31 12:28:08 | 2023-08-31 12:28:08 | 1         |
| 3  | Peter Rabbit | 3   | 2023-08-31 12:28:08 | 2023-08-31 12:28:08 | 2         |
| 5  | Peter Junior | 2   | 2023-08-31 13:09:06 | 2023-08-31 13:09:06 | 1         |
+----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Hint: you can always consult the Sequelize documentation for help <https://sequelize.org/docs/v6/>

4. CLEANING UP

4.1. REMOVING THE CONTAINER

At the end of the lab you can stop and remove the database by using the following command. Do it in the **sequelize/** directory (where the **docker-compose.yml** file can be found). This command removes the Docker container for the database and also the volume used to store the database.

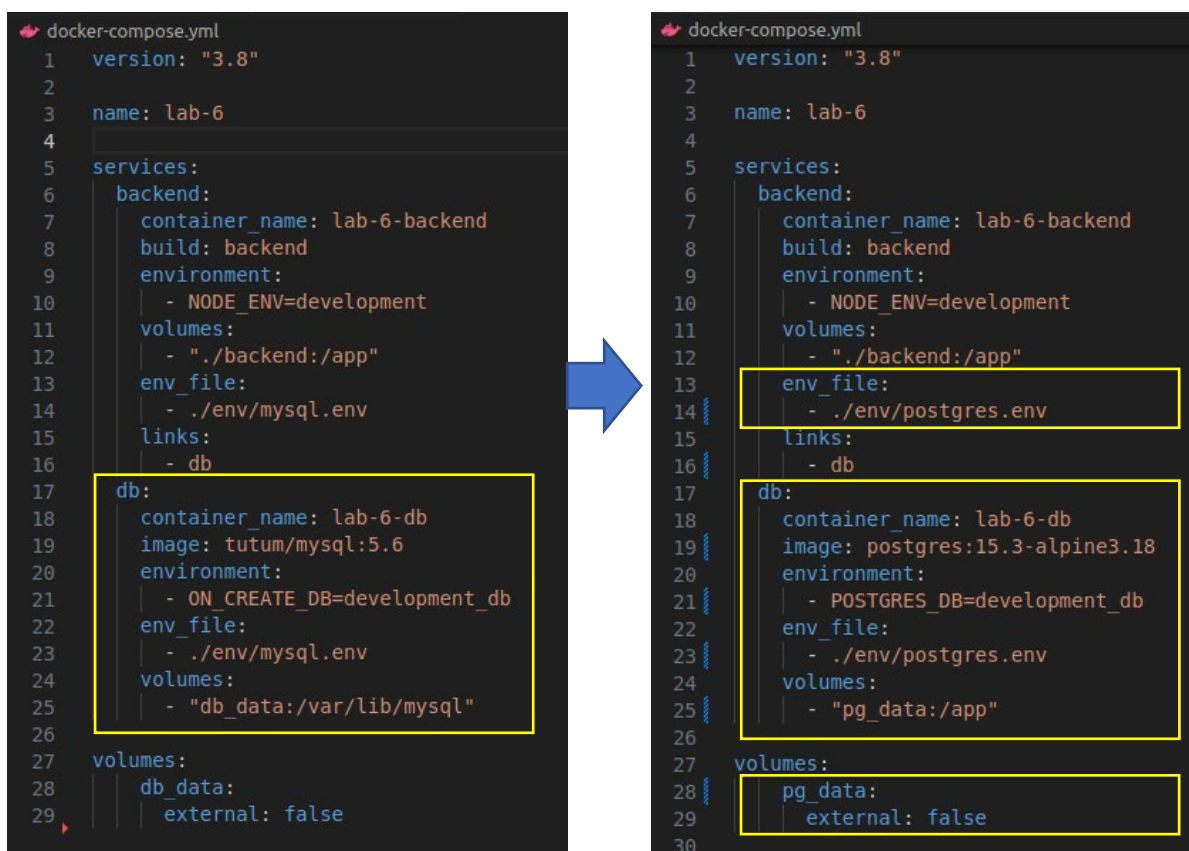
```
docker compose down -v
```

5. CHANGE RDBMS

Sequelize is the middleman between your application and RDBMS. By using the middleman, you don't have to be worried with the RDBMS specific languages. It means that all of your sequelize script will work with any RDBMS engine.

In this section, we will demonstrate how to change your RDBMS engine from MySQL to PostgreSQL. Before you start changing the RDBMS configuration, make sure to **stop your container**.

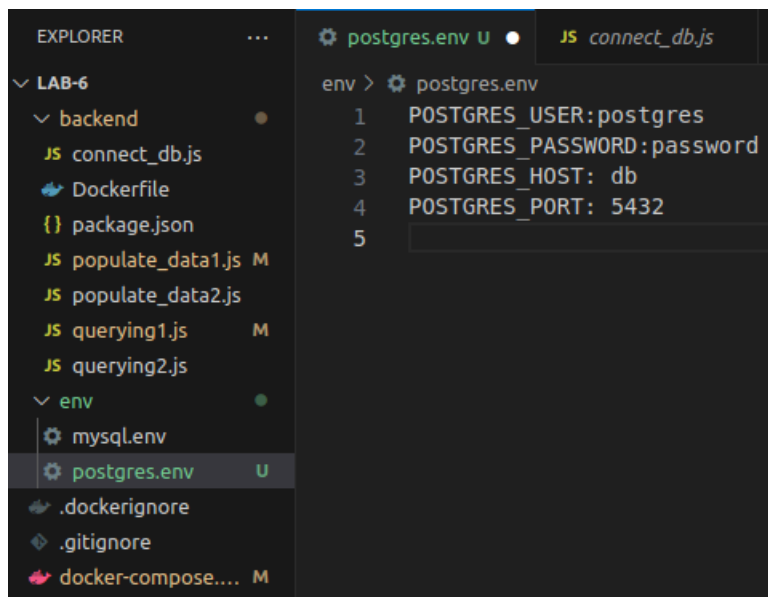
First, we must replace the database service in "docker-compose.yml" from MySQL to PostgreSQL by replacing the "db" service. In this example, we will use postgres:15.3-alpine3.18 image.



We need to replace the database image from MySQL to PostgreSQL, environment variable and the volumes to store the data location.

```
db:
  container_name: lab-6-db
  image: postgres:15.3-alpine3.18
  environment:
    - POSTGRES_DB=development_db
  env_file:
    - ./env/postgres.env
  volumes:
    - "pg_data:/app"
```

Note, that we must create a file /env/postgres.env to keep the username and password to access the database.



The Sequelize configuration is stored in `\backend\connect_db.js` file. This file contains the host and port of the database server, username and password. As you can see, the database server configurations are taken from the environment variable, where the data was loaded from postgres.env file.

JS connect_db.js

```
mysql > lab-6 > backend > JS connect_db.js > ...
1  /**
2   * Requiring this file will connect to the
3   * database and return the
4   * Sequelize database connection object.
5   */
6
7  const Sequelize = require('sequelize');
8
9  const dbConfig = {
10    host: process.env.MYSQL_REMOTE_HOST,
11    port: process.env.MYSQL_REMOTE_PORT,
12    // here we are selecting mysql as the
13    // database type we will be using
14    dialect: 'mysql'
15  };
16
17  // Here we connect to the database
18  const db = new Sequelize(
19    'development_db',
20    process.env.MYSQL_USER,
21    process.env.MYSQL_PASS,
22    dbConfig);
23
24  module.exports = db;
25
```

JS connect_db.js M

```
backend > JS connect_db.js > ...
1  /**
2   * Requiring this file will connect to the
3   * database and return the
4   * Sequelize database connection object.
5   */
6
7  const Sequelize = require('sequelize');
8
9  const dbConfig = {
10    host: process.env.POSTGRES_HOST,
11    port: process.env.POSTGRES_PORT,
12    // here we are selecting postgres as the
13    // database type we will be using
14    dialect: 'postgres'
15  };
16
17  // Here we connect to the database
18  const db = new Sequelize(
19    'development_db',
20    process.env.POSTGRES_USER,
21    process.env.POSTGRES_PASSWORD,
22    dbConfig);
23
24  module.exports = db;
25
```

Replace the npm package dependency from MySQL to PostgreSQL in /backend/package.json

EXPLORER

LAB-6

backend

JS connect_db.js M

Dockerfile M

{} package.json M

JS populate_data1.js M

JS populate_data2.js M

JS querying1.js M

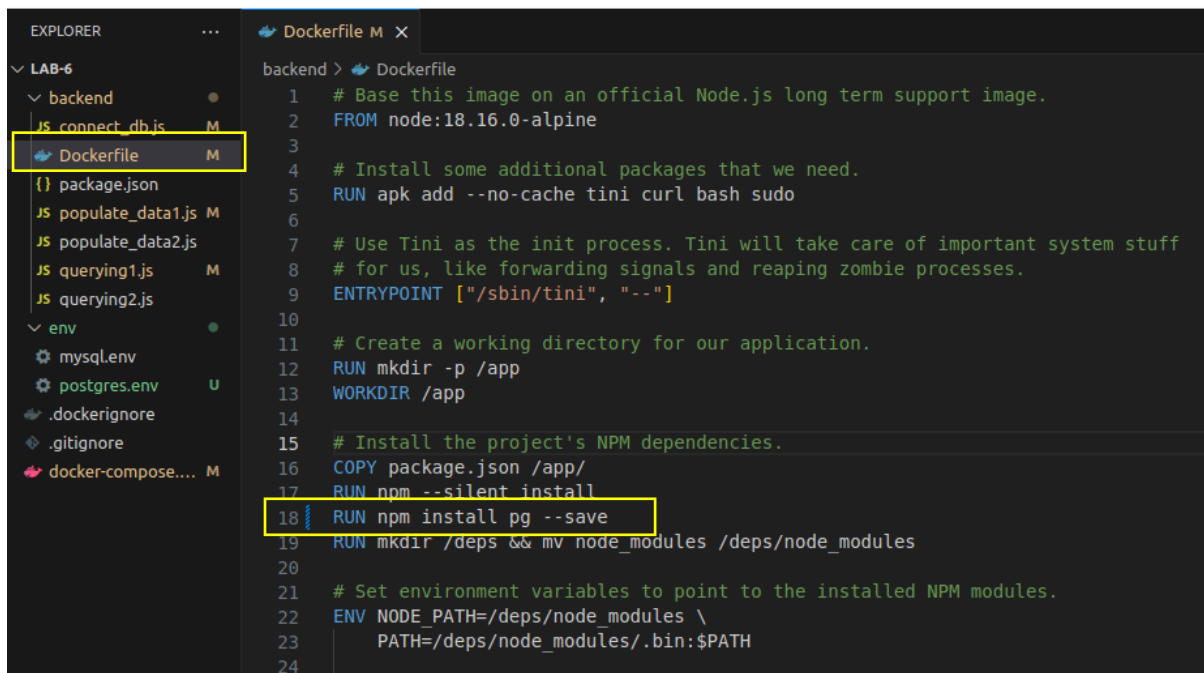
JS querying2.js M

> env

{} package.json M

```
backend > {} package.json > ...
1  {
2    "name": "backend",
3    "version": "0.0.1",
4    "private": true,
5    "dependencies": {
6      "postgres": "3.3.5",
7      "sequelize": "6.32.0"
8    }
9  }
10
```

Then, you have to replace the MySQL library during npm installation to PostgreSQL in **/backend/Dockerfile**.



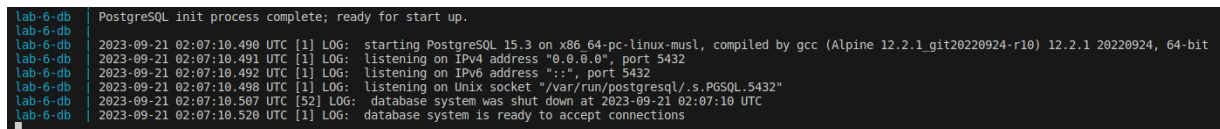
```
EXPLORER
LAB-6
├── backend
│   ├── connect_db.js
│   ├── Dockerfile
│   ├── package.json
│   ├── populate_data1.js
│   ├── populate_data2.js
│   ├── querying1.js
│   └── querying2.js
├── env
│   ├── mysql.env
│   └── postgres.env
├── .dockerignore
├── .gitignore
└── docker-compose....

backend > Dockerfile
1  # Base this image on an official Node.js long term support image.
2  FROM node:18.16.0-alpine
3
4  # Install some additional packages that we need.
5  RUN apk add --no-cache tini curl bash sudo
6
7  # Use Tini as the init process. Tini will take care of important system stuff
8  # for us, like forwarding signals and reaping zombie processes.
9  ENTRYPOINT ["/sbin/tini", "--"]
10
11 # Create a working directory for our application.
12 RUN mkdir -p /app
13 WORKDIR /app
14
15 # Install the project's NPM dependencies.
16 COPY package.json /app/
17 RUN npm --silent install
18 RUN npm install pg --save
19 RUN mkdir /deps && mv node_modules /deps/node_modules
20
21 # Set environment variables to point to the installed NPM modules.
22 ENV NODE_PATH=/deps/node_modules \
23     PATH=/deps/node_modules/.bin:$PATH
24
```

Once you have configured everything correctly, start up your PostgreSQL engine

```
docker compose up db --build
```

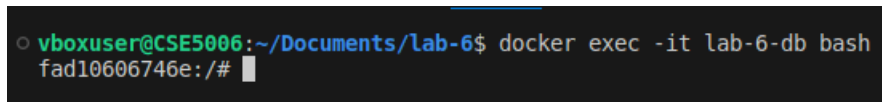
Your database is ready when you see this output



```
lab-6-db | PostgreSQL init process complete; ready for start up.
lab-6-db | 2023-09-21 02:07:10.490 UTC [1] LOG:  starting PostgreSQL 15.3 on x86_64-pc-linux-musl, compiled by gcc (Alpine 12.2.1_git20220924-r10) 12.2.1 20220924, 64-bit
lab-6-db | 2023-09-21 02:07:10.491 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
lab-6-db | 2023-09-21 02:07:10.492 UTC [1] LOG:  listening on IPv6 address "::", port 5432
lab-6-db | 2023-09-21 02:07:10.498 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
lab-6-db | 2023-09-21 02:07:10.507 UTC [52] LOG:  database system was shut down at 2023-09-21 02:07:10 UTC
lab-6-db | 2023-09-21 02:07:10.520 UTC [1] LOG:  database system is ready to accept connections
```

Now, you can connect to the database using the following command

```
docker exec -it lab-6-db bash
```



```
o vboxuser@CSE5006:~/Documents/lab-6$ docker exec -it lab-6-db bash
fad10606746e:/#
```

Use the following script to open the PostgreSQL console and check the table in it

```
psql -U $POSTGRES_USER -h $POSTGRES_HOST -d $POSTGRES_DB
```

```

fad10606746e:/# psql -U $POSTGRES_USER -h $POSTGRES_HOST -d $POSTGRES_DB
Password for user postgres:
psql (15.3)
Type "help" for help.

development_db=# \dt
Did not find any relations.
development_db=#

```

As you can see, there are no tables in the database. Try to redo the labs using PostgreSQL as your RDBMS.

Below is the example of running populate_data1.js

```

vboxuser@CSE5006:~/Documents/lab-6/backend$ docker compose run --rm backend node populate_data1.js
[+] Creating 1/0
✓ Container lab-6-db Running
0.0s
Executing (default): DROP TABLE IF EXISTS "articles" CASCADE;
Executing (default): SELECT DISTINCT tc.constraint_name as constraint_name, tc.constraint_schema as constraint_schema, tc.constraint_catalog as constraint_catalog, tc.table_name as table_name, tc.table_schema as table_schema, tc.table_catalog as table_catalog, tc.initially_deferred as initially_deferred, tc.is_deferrable as is_deferrable, kcu.column_name as column_name, ccu.table_schema AS referenced_table_schema, ccu.table_catalog AS referenced_table_catalog, ccu.table_name AS referenced_table_name, ccu.column_name AS referenced_column_name FROM information_schema.table_constraints AS tc JOIN information_schema.key_column_usage AS kcu ON tc.constraint_name = kcu.constraint_name JOIN information_schema.constraint_column_usage AS ccu ON ccu.constraint_name = tc.constraint_name WHERE constraint_type = 'FOREIGN KEY' AND tc.table_name = 'articles' AND tc.table_catalog = 'development_db';
Executing (default): DROP TABLE IF EXISTS "articles" CASCADE;
Executing (default): DROP TABLE IF EXISTS "articles" CASCADE;
Executing (default): CREATE TABLE IF NOT EXISTS "articles" ("id" SERIAL, "title" VARCHAR(255), "content" TEXT, "createdAt" TIMESTAMP WITH TIME ZONE NOT NULL, "updatedAt" TIMESTAMP WITH TIME ZONE NOT NULL, PRIMARY KEY ("id"));
Executing (default): SELECT i.relname AS name, ix.indisprimary AS primary, ix.indisunique AS unique, ix.indkey AS indkey, array_agg(a.attname) AS column_names, pg_get_indexdef(ix.indexrelid) AS definition FROM pg_class t, pg_class i, pg_index ix, pg_attribute a WHERE t.oid = ix.indrelid AND i.oid = ix.indexrelid AND a.attrelid = t.oid AND t.relkind = 'r' and t.relname = 'articles' GROUP BY i.relname, ix.indexrelid, ix.indisprimary, ix.indisunique, ix.indkey ORDER BY i.relname;
Executing (default): INSERT INTO "articles" ("id","title","content","createdAt","updatedAt") VALUES (DEFAULT,$1,$2,$3,$4) RETURNING "id","title","content","createdAt","updatedAt";
Executing (default): INSERT INTO "articles" ("id","title","content","createdAt","updatedAt") VALUES (DEFAULT,$1,$2,$3,$4) RETURNING "id","title","content","createdAt","updatedAt";
Executing (default): INSERT INTO "articles" ("id","title","content","createdAt","updatedAt") VALUES (DEFAULT,$1,$2,$3,$4) RETURNING "id","title","content","createdAt","updatedAt";
Executing (default): INSERT INTO "articles" ("id","title","content","createdAt","updatedAt") VALUES (DEFAULT,$1,$2,$3,$4) RETURNING "id","title","content","createdAt","updatedAt";
Executing (default): INSERT INTO "articles" ("id","title","content","createdAt","updatedAt") VALUES (DEFAULT,$1,$2,$3,$4) RETURNING "id","title","content","createdAt","updatedAt";
vboxuser@CSE5006:~/Documents/lab-6/backend$

```

```

development_db=# \dt
List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | articles | table | postgres
(1 row)

development_db=# select * from articles;
 id | title | content | createdAt | updatedAt |
-----+-----+-----+-----+-----+
  1 | War and Peace | A book about fighting and then making up. | 2023-09-21 02:36:17.256+00 | 2023-09-21 02:36:17.256+00 |
  2 | Sequelizeize for dummies | Writing lots of cool javascript code that get turned into SQL. | 2023-09-21 02:36:17.256+00 | 2023-09-21 02:36:17.256+00 |
  3 | I like tomatoes | The story about the adventures of a tomato lover. | 2023-09-21 02:36:17.256+00 | 2023-09-21 02:36:17.256+00 |
  4 | PHP for dummies | Why PHP is so so so bad at backend stuff. Why you should use express node. | 2023-09-21 02:36:17.256+00 | 2023-09-21 02:36:17.256+00 |
  5 | The lovely car | How a car changed his life forever. | 2023-09-21 02:36:17.257+00 | 2023-09-21 02:36:17.257+00 |
(5 rows)

development_db=#

```

Please note that the connection setting can be configured in many ways. Therefore, you might see different ways of configuration from one app to another. Our labs use several configurations, so you have more experience in handling multiple settings.

Answers:

Exercise 1:

```
.then(()=>Article.findAll({
  where:{
    id:[Op.in]:[1,3]}
}))
.then(articles=>{
  console.log('ID = 1 or 3');
  articles.forEach(article=>{
    console.log(article.dataValues);
  })
  console.log();
})
```

Exercise 2:

```
.then(()=>Article.findByPk(2))
.then(article => article.update({ content: 'Sequelize is the worst
ORM ever!' }))
.then(() => Article.findByPk(2))
.then(article => {
  console.log('# Article with id=2');
  console.log(article.dataValues);
  console.log();
})
```

Exercise 3:

```
.then(() => Employees.findOne({ where: { name: 'Peter Rabbit' },
include: [Companies] }))
.then(employee => console.log(employee.company.dataValues))
```

Exercise 4:

```
.then(() => Companies.findOne({ order: [['profit', 'DESC']], include:
[Employees] }))
.then(company => {
  company.employees.forEach(employee => {
    console.log(employee.dataValues);
    console.log()
  });
})
```


Exercise 5:

```
.then(()=>Companies.findOne({id:1}))
.then(c=>{
  console.log("Inserting new employee to company ID="+c.id);
  const e3 = Employees.create({
    name: 'Peter Junior',
    age: 2,
    companyId: c.id
  })
  return e3
})
```