SEQUELIZE

CSE5006 – LAB 6

Repository:
https://github.com/choiruzain-latrobe/lab-6.git

TABLE OF CONTENTS

OUTLINE

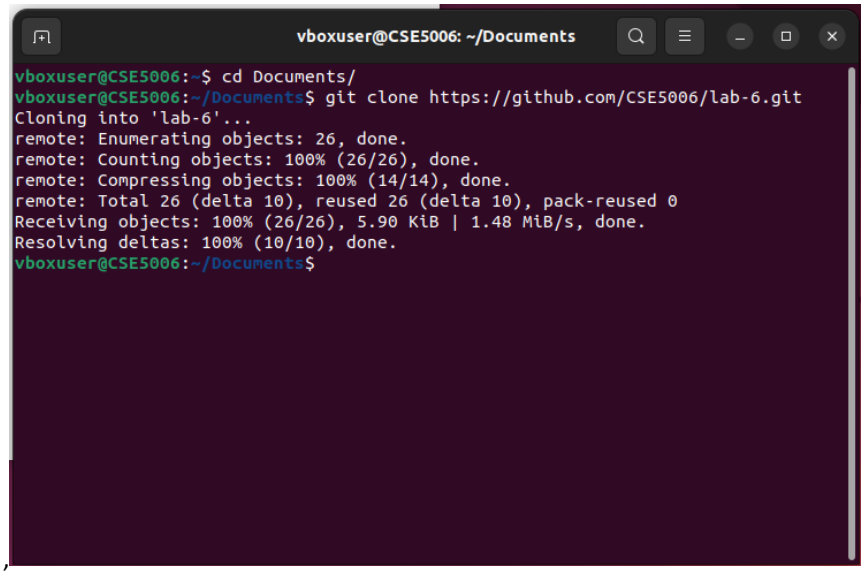
In this lab you will practice the following:

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

1. GET EVERYTHING RUNNING

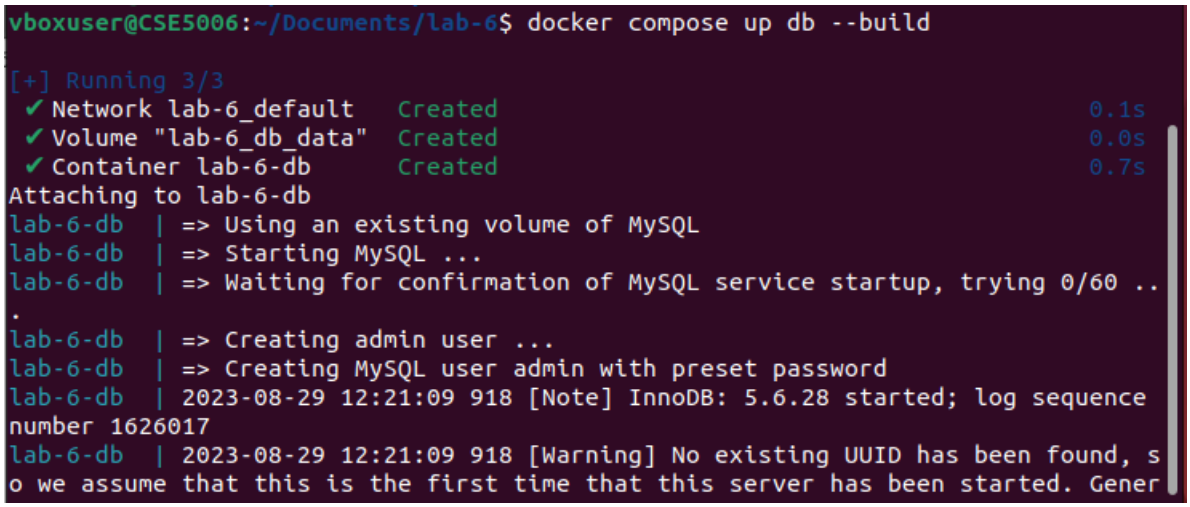
1.1. DOCKER COMPOSE UP

Fork the git address from the https://github.com/choiruzain-latrobe/lab-6.git to your local repository. Clone from your local repository, so that the command like the one below:



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
```



Wait until you see either the output shown in the screenshot below or a message saying, "**ready for connection.**" Once you see either, you can proceed to the next step.

Example output 1:

```
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
```

Example output 2:

```
Attaching to sequelize_db_1
db_1       | => Using an existing volume of MySQL
db_1       | => Starting MySQL ...
db_1       | => Waiting for confirmation of MySQL service startup, trying 0/60 ...
db_1       | => Waiting for confirmation of MySQL service startup, trying 1/60 ...
db_1       | tail -F $LOG
db_1       | 2022-08-22 09:13:08 917 [Note] InnoDB: Waiting for purge to start
db_1       | 2022-08-22 09:13:08 917 [Note] InnoDB: 5.6.28 started; log sequence number 1626037
db_1       | 2022-08-22 09:13:08 917 [Note] Server hostname (bind-address): '0.0.0.0'; port: 330
6
db_1       | 2022-08-22 09:13:08 917 [Note]   - '0.0.0.0' resolves to '0.0.0.0';
db_1       | 2022-08-22 09:13:08 917 [Note] Server socket created on IP: '0.0.0.0'.
db_1       | 2022-08-22 09:13:08 917 [Warning] 'user' entry 'root@64bf666ea130' ignored in --ski
p-name-resolve mode.
db_1       | 2022-08-22 09:13:08 917 [Warning] 'proxies_priv' entry '@ root@64bf666ea130' ignore
d in --skip-name-resolve mode.
db_1       | 2022-08-22 09:13:08 917 [Note] Event Scheduler: Loaded 0 events
db_1       | 2022-08-22 09:13:08 917 [Note] /usr/sbin/mysqld: ready for connections.
db_1       | Version: '5.6.28-0ubuntu0.14.04.1'  socket: '/var/run/mysqld/mysqld.sock'  port: 33
06  (Ubuntu)
db_1       | tail: unrecognized file system type 0x794c7630 for '/var/log/mysql/error.log'. plea
se report this to bug-coreutils@gnu.org. reverting to polling
```

1.2. ENTER CONTAINER TERMINAL

1. Start a **new terminal** where we are going to log into the MySQL database.
2. Run the docker container (lab-6-db), and run the following command:

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

You will be redirected to the container shell as shown in the next section.

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
```

You will be redirected into the mysql shell inside the container as follows:

```
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
/**
 * Requiring this file will connect to the database and return the
 * Sequelize database connection object.
 */

const Sequelize = require('sequelize');

const dbConfig = {
  host: process.env.MYSQL_REMOTE_HOST,
  port: process.env.MYSQL_REMOTE_PORT,
  // here we are selecting mysql as the database type we will be using
  dialect: 'mysql'
};

// Here we connect to the database
const db = new Sequelize(
  'development_db',
  process.env.MYSQL_USER,
  process.env.MYSQL_PASS,
  dbConfig);

module.exports = db;
```

```
MYSQL_USER=admin
MYSQL_PASS=af84de4d1b8a247e8ef9925d273d3ec549782c76be096217
MYSQL_REMOTE_HOST=db
MYSQL_REMOTE_PORT=3306
```

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

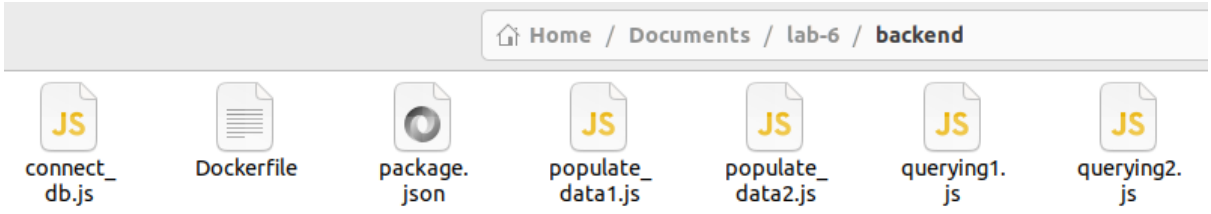
```yaml
version: "2"
services:
  backend:
    build: backend
    environment:
      - NODE_ENV=development
    volumes:
      - "./backend:/app"
    env_file:
      - ./env/mysql.env
    links:
      - db
  db:
    image: tutum/mysql:5.6
    environment:
      - ON_CREATE_DB=development_db
    env_file:
      - ./env/mysql.env
    volumes:
      - "db_data:/var/lib/mysql"
volumes:
    db_data:
      external: false
```

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

Here, we can see what is happening as we start inserting data and querying the database via Sequelize. 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).

1. 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
 => => sha256:9036ddb8252ba7089c2c83eb2b0dcaf74ff1069e8dd 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:c017600940c61ad955e20e7dacd301feb287dddd 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:c017600940c61ad955e20e7dacd301feb287dddd442f63d9  2.1s
 => => extracting sha256:c9f8586f07bd2ca2364211d26e40d5bb881cf547be76e7e0  0.1s
```

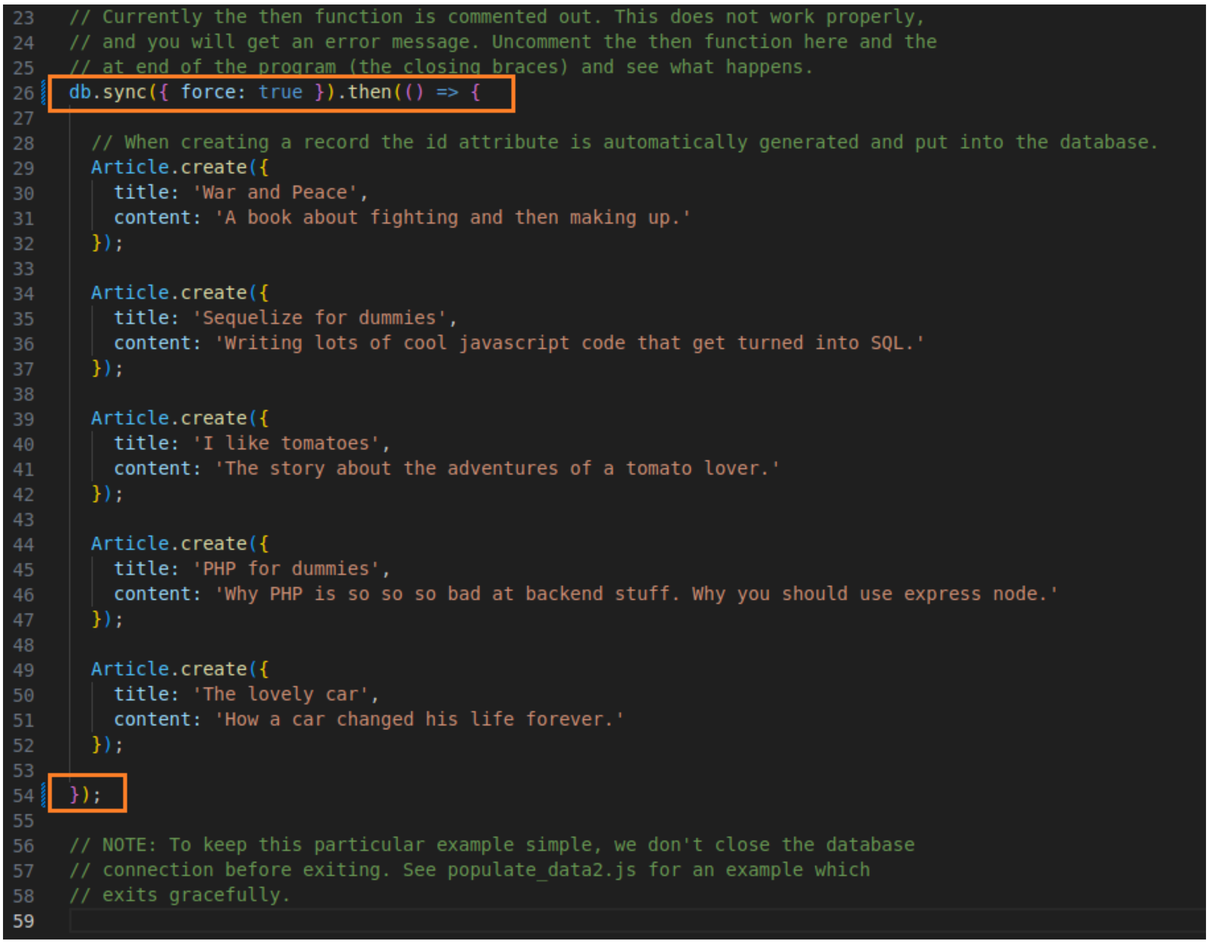
2.3. ASYNCHRONOUS ERRORS

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.



```
// Currently the then function is commented out. This does not work properly,
// and you will get an error message. Uncomment the then function here and the
// at end of the program (the closing braces) and see what happens.
db.sync({ force: true }).then(() => {

  // When creating a record the id attribute is automatically generated and put into the database.
  Article.create({
    title: 'War and Peace',
    content: 'A book about fighting and then making up.'
  });

  Article.create({
    title: 'Sequelize for dummies',
    content: 'Writing lots of cool javascript code that get turned into SQL.'
  });

  Article.create({
    title: 'I like tomatoes',
    content: 'The story about the adventures of a tomato lover.'
  });

  Article.create({
    title: 'PHP for dummies',
    content: 'Why PHP is so so so bad at backend stuff. Why you should use express node.'
  });

  Article.create({
    title: 'The lovely car',
    content: 'How a car changed his life forever.'
  });

});

// NOTE: To keep this particular example simple, we don't close the database
// connection before exiting. See populate_data2.js for an example which
// exits gracefully.

```

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 databas
e 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_SCHE
MA 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_S
CHEMA='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**, ascd cd well as creation and update timestamps.

2.6. SELECTING DATA

Let's 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

Execute the query by **running the JavaScript code (querying1.js)** below. Note: run it in a new terminal as follows:

```
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 an 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 `ar
ticles`.`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 `ar
ticles`.`id` = 2;
Executing (default): SELECT `id`, `title`, `content`, `createdAt`, `updatedAt` FROM `articles` AS `articles` WHERE `ar
ticles`.`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 databas
e 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 constraint_name,CONSTRAINT_NAME as constraintName,CONSTRAINT_SCHEMA as
constraintSchema,CONSTRAINT_SCHEMA as constraintCatalog,TABLE_NAME as tableName,TABLE_SCHEMA as tableSchema,TABLE_SCHE
MA 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 constraint_name,CONSTRAINT_NAME as constraintName,CONSTRAINT_SCHEMA as
constraintSchema,CONSTRAINT_SCHEMA as constraintCatalog,TABLE_NAME as tableName,TABLE_SCHEMA as tableSchema,TABLE_SCHE
MA 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 databas
e 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`, `employe
es`.`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.update
dAt` 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`, `employe
es`.`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.update
dAt` 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 `e
mployees.updatedAt`, `employees`.`companyId` AS `employees.companyId` FROM (SELECT `companies`.`id`, `companies`.`name
`, `companies`.`profit`, `companies`.`createdAt`, `companies`.`updatedAt` FROM `companies` AS `companies` ORDER BY `co
mpanies`.`profit` DESC LIMIT 1) AS `companies` LEFT OUTER JOIN `employees` AS `employees` ON `companies`.`id` = `emplo
yees`.`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
}
vboxuser@CSE5006:~/Documents/lab-6/backend$
```

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`,`upd
atedAt`,`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 the volume used to store the database.

```
docker compose down -v
```

5. CHANGE RDBMS INTO POSTGRESQL

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 your sequelize script will work with any RDBMS engine.

5.1. CONFIGURING THE DOCKER COMPOSE, POSTGRES ENV, CONNECT_DB, PACKAGE.JSON, AND DOCKERFILE

In this section, we will demonstrate how to change your RDBMS engine from **MySQL to PostgreSQL**.

Do the following tasks:

1. Stop the container (Ctrl + C) or Go to the root folder and execute

   ```
   docker compose down -v
   ```

2. Change the RDBMS configuration. In the **docker-compose.yml** file, make the following edits, so that the MySQL and PostgreSQL settings are as follows:

```yaml
version: "3.8"

name: lab-6

services:
  backend:
    container_name: lab-6-backend
    build: backend
    environment:
      - NODE_ENV=development
    volumes:
      - "./backend:/app"
    env_file:
      - ./env/mysql.env
    links:
      - db
  db:
    container_name: lab-6-db
    image: tutum/mysql:5.6
    environment:
      - ON_CREATE_DB=development_db
    env_file:
      - ./env/mysql.env
    volumes:
      - "db_data:/var/lib/mysql"

volumes:
    db_data:
      external: false
```

```yaml
version: "3.8"

name: lab-6

services:
  backend:
    container_name: lab-6-backend
    build: backend
    environment:
      - NODE_ENV=development
    volumes:
      - "./backend:/app"
    env_file:
      - ./env/postgres.env
    links:
      - db
  db:
    container_name: lab-6-db
    image: postgres:15.4-alpine3.18
    environment:
      - POSTGRES_DB=development_db
    env_file:
      - ./env/postgres.env
    volumes:
      - pg_data:/var/lib/postgresql/data

volumes:
    pg_data:
      external: false
```

The final configuration of the docker-compose.yml for PostgreSQL is as follows:

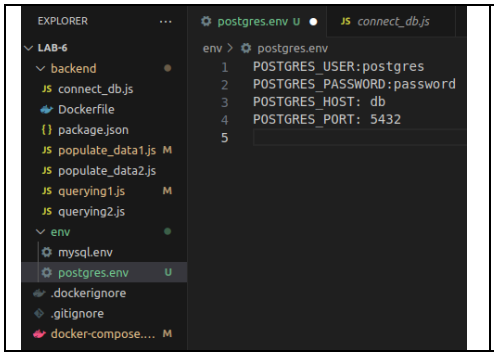
```yaml
version: "3.8"

name: lab-6-postgres

services:
  backend:
    container_name: lab-6-backend
    build: backend
    environment:
      - NODE_ENV=development
    volumes:
      - "./backend:/app"
    env_file:
      - ./env/postgres.env
    links:
      - db
  db:
    container_name: lab-6-db
    image: postgres:15.4-alpine3.18
    environment:
      - POSTGRES_DB=development_db
    env_file:
      - ./env/postgres.env
    volumes:
      - pg_data:/var/lib/postgresql/data

volumes:
    pg_data:
      external: false
```

3. Create a file named **postgres.env** in the **env** folder and configure it with the following settings to store the username and password for database access.



Final configuration of postgres.env:

```
POSTGRES_USER=postgres
POSTGRES_PASSWORD=password
POSTGRES_HOST=db
POSTGRES_PORT=5432
```

4. Change the configuration of **\backend\connect_db.js** file. This file contains the host, port, username, and password for the database server. The configurations are taken from the environment variable, where the data was loaded from postgres.env file. The configuration and final configuration is as follows:

Mysql connect_db.js

```
const Sequelize = require('sequelize');

const dbConfig :{...} = {
  host: process.env.MYSQL_REMOTE_HOST,
  port: process.env.MYSQL_REMOTE_PORT,
  // here we are selecting mysql as the database
    type we will be using
  dialect: 'mysql'
};

// Here we connect to the database
const db = new Sequelize(
  'development_db',
  process.env.MYSQL_USER,
  process.env.MYSQL_PASS,
  dbConfig);

module.exports = db;
```

PostgreSQL connect_db.js

```
const Sequelize = require('sequelize');

const dbConfig :{...} = {
  host: process.env.POSTGRES_HOST,
  port: process.env.POSTGRES_PORT,
  // here we are selecting postgresql as the
   database type we will be using
  dialect: 'postgres'
};

// Here we connect to the database
const db = new Sequelize(
    'development_db',
    process.env.POSTGRES_USER,
    process.env.POSTGRES_PASSWORD,
    dbConfig);

module.exports = db;
```

Final configuration of PostgreSQL connect_db.js:

```
/**
 * Requiring this file will connect to the database and return the
 * Sequelize database connection object.
 */

const Sequelize = require('sequelize');

const dbConfig = {
  host: process.env.POSTGRES_HOST,
  port: process.env.POSTGRES_PORT,
  // here we are selecting mysql as the database type we will be using
  dialect: 'postgres'
};

// Here we connect to the database
const db = new Sequelize(
  'development_db',
  process.env.POSTGRES_USER,
  process.env.POSTGRES_PASSWORD,
  dbConfig);

module.exports = db;
```

5. Configure of **/backend/package.json** to change the npm package dependency from MySQL to PostgreSQL in as follows:

Previous configuration *package.json* of MySQL:

```
{
  "name": "backend",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "mysql": "2.18.1",
    "sequelize": "6.32.0"
  }
}
```

Final configuration *package.json* of PostgreSQL:

```
{
  "name": "backend",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "postgres": "3.3.5",
    "sequelize": "6.32.0"
  }
}
```

6. Configure **/backend/Dockerfile**. Replace MySQL library during npm installation to PostgreSQL as follows:

```
# Base this image on an official Node.js long term support image.
FROM node:18.16.0-alpine

# Install some additional packages that we need.
RUN apk add --no-cache tini curl bash sudo
```

```
# Use Tini as the init process. Tini will take care of important system stuff
# for us, like forwarding signals and reaping zombie processes.
ENTRYPOINT ["/sbin/tini", "--"]

# Create a working directory for our application.
RUN mkdir -p /app
WORKDIR /app

# Install the project's NPM dependencies.
COPY package.json /app/
RUN npm --silent install
RUN npm install pg --save
RUN mkdir /deps && mv node_modules /deps/node_modules

# Set environment variables to point to the installed NPM modules.
ENV NODE_PATH=/deps/node_modules \
  PATH=/deps/node_modules/.bin:$PATH

# Copy our application files into the image.
COPY . /app

# Switch to a non-privileged user for running commands inside the container.
RUN chown -R node:node /app /deps \
 && echo "node ALL=(ALL) NOPASSWD:ALL" > /etc/sudoers.d/90-node
USER node

# Run an interactive shell
CMD [ "bash" ]
```

7. Once the, everything has been setup, Run the following command in the root folder:

```
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  |
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
```

database system is ready to accept connections

8. Now, you can connect to the database. Open the **new terminal**, run the following command:

```
LABS-2024-02 % docker exec -it lab-6-db bash
```

9. Inside the docker terminal, use the following script to open the PostgreSQL console:

```
ab53eeeb5f63:/# PGPASSWORD=$POSTGRES_PASSWORD psql -U $POSTGRES_USER -h $POSTGRES_HOST -d $POSTGRES_DB
```

you will see the POSTGRES terminal like the picture shown below:

```
c116a0ee5b3a:/# PGPASSWORD=$POSTGRES_PASSWORD psql -U $POSTGRES_USER -h $POSTGRES_HOST -d $POSTGRES_DB
psql (15.4)
Type "help" for help.

development_db=#
```

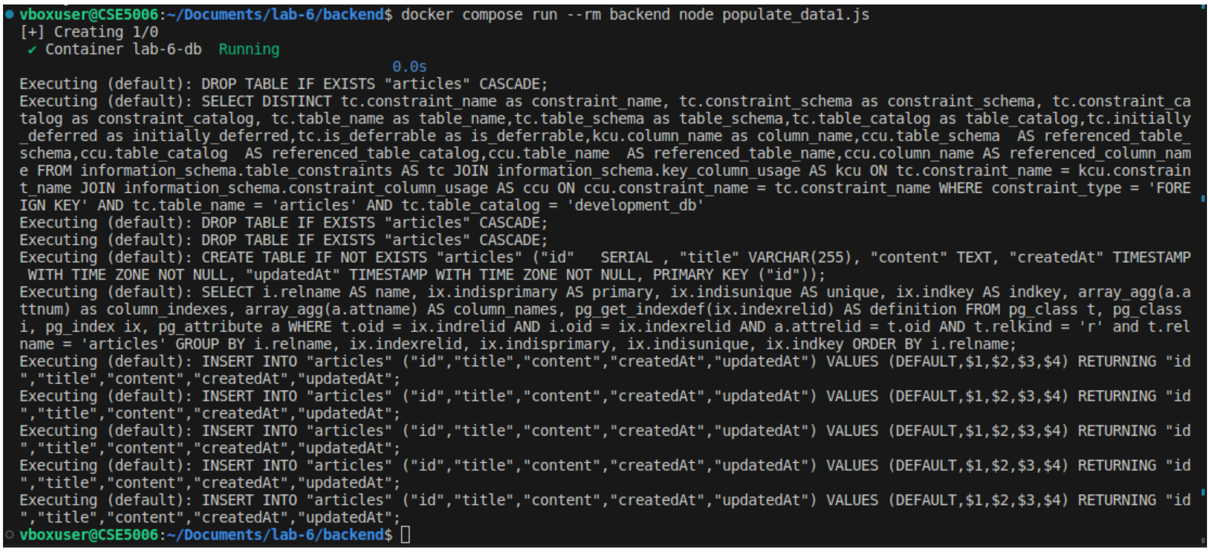
5.2. POPULATE THE DATA INTO THE DATABASE

Up to now, we have changed the environment to PostgreSQL. Now, let's focus on populating data into the database. As shown above (step 10), there are no tables in the PostgreSQL database. Use the same code as before to populate data using PostgreSQL as your RDBMS. See example: populate_data1.js. Run the following steps:

1. In the terminal, under the **backend** folder, run the following command:

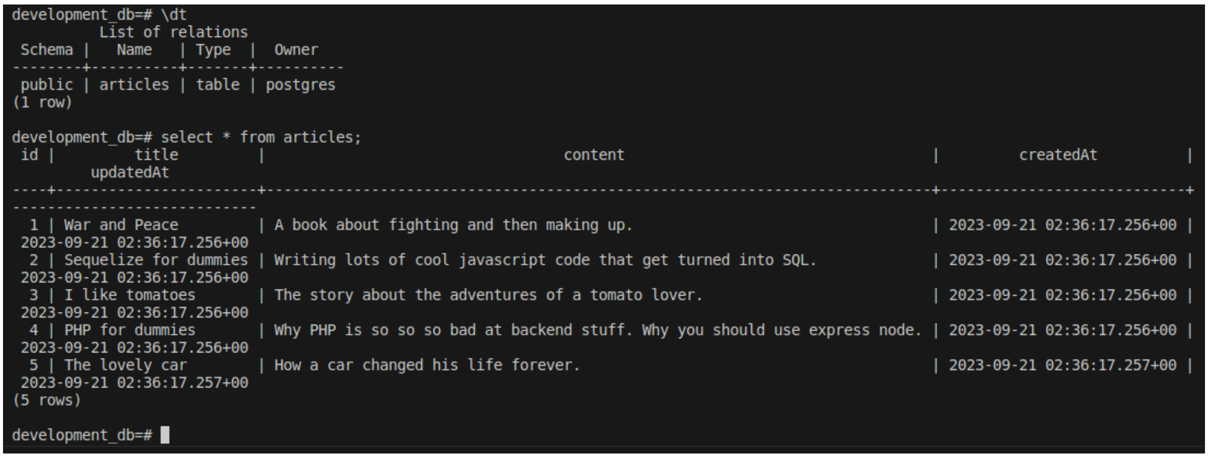
   ```
   docker compose run --rm backend node populate_data1.js
   ```

   You will get the interface presented as follows:



```
● 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_ca
 talog 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_nam
 e FROM information_schema.table_constraints AS tc JOIN information_schema.key_column_usage AS kcu ON tc.constraint_name = kcu.constrain
 t_name JOIN information_schema.constraint_column_usage AS ccu ON ccu.constraint_name = tc.constraint_name WHERE constraint_type = 'FORE
 IGN 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.a
 ttnum) as column_indexes, 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.rel
 name = '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,$4) RETURNING "id
 ","title","content","createdAt","updatedAt";
○ vboxuser@CSE5006:~/Documents/lab-6/backend$ ▯
```

2. Please focus on the PostgreSQL shell displayed below. Execute the following command under **development_db** to view the list of tables.



```
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 | Sequelize 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=# █
```

From here, you can now populate data inside the **PostgreSQL** database under **development_db**. You've created an **articles** table and inserted records, as demonstrated earlier. Similarly, follow a similar approach as outlined in section 2.2 for populating data in MySQL.

6. SOLUTION OF EXERCISES

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
})
```