

Your to-do application was working great, but then, disaster. The file system where you were storing your saved sessions corrupted and you lost all your data. Your incident analyst recommends you use a database table to store your to-dos (as if you didn't have enough to do). Most of the application is already updated to support the database but you are tasked with completing the module, nodeTodoDb, which will connect to the database and execute queries to read to-dos for a given session id and insert new to-dos into the table.

Copy the starter files into a new directory called lab10 in your coen161 directory.

Creating the Table (20 pts)

1. Access the linux terminal and run the following commands from the [MySQL wiki](#)

```
setup mysql5
```

```
mysql -h dbserver.engr.scu.edu -p -u <username> <db_name>
```

2. To see the list of databases use the SHOW DATABASES; command (make sure to add a semi colon at the end)
3. To make changes to your primary databases you first need to select your database using the USE sdb_<username>; statement
4. To see the list of tables use the SHOW TABLES; command
5. Create a new table called todos using the following command

```
CREATE TABLE Todos (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    description VARCHAR(255) NOT NULL,
    sessionId VARCHAR(255) NOT NULL
);
```

You should now have a table called Todos that we can use to store data for the application.

Finishing the Application (80 pts)

The application you are handed is missing the code to connect to the database and retrieve and insert data to a database.

First, start by installing the mysql module in your application.

```
npm install mysql
```

Modify the file nodeTodoDb.js with your database credentials.

In the addTodo function, use the provided connection to connect to the database and execute a query to insert a new row into the table Todos with two columns, the description and session id. Use one of the methods discussed in class to escape your query. Rather than throwing an error in the callback functions, you can call the passed in callback function and return so that the server can continue listening for requests.

```
if (err) {  
  return callback(err);  
}
```

If you do have a successful database query, you still call the callback function so that the server knows nothing went wrong. After your query executes and you process it, you will want to call the method con.end() to properly close the connection.

Once you're inserting data into the database, you can verify it by selecting it in the mysql command line tool where you created the table.

```
SELECT * FROM Todos;
```

Now that we're adding records to the database, we need to retrieve them to send them back to the client. In the method getTodos, use the provided connection to connect and execute a query that selects all the rows from the Todos table where the sessionId matched the given sessionId in the parameter of the function. Once again, make sure to properly escape your query before sending it to the database. Like the addTodo function, make sure to use the callback parameter function to communicate with the server. This time, you will need to pass the results as the second parameter of the callback function.

```
callback(err, results);
```

Don't forget to close your connection. Once you've verified that you're reading and writing to the database successfully, restart your server and reload the todo application to make sure that all your todos are still saved.

Submit only your finished nodeTodoDb module.