WEB322 Test 4 Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Instructions: Complete all questions in the spaces provided. This quiz is worth 10% of your final mark and you will have exactly 30 minutes to complete it.

# Question 1 (10 Marks):

For each multiple-choice question, identify the correct answer for the given question. There is only **one** correct answer provided for each question:

|  |  |
| --- | --- |
| 1. The relational database system used in WEB322 is:  * Oracle * Postgres * postSQL * MySQL | 1. The popular GUI tool for working with the DB system (from 'a'), used in WEB322 is :  * pgAdmin 4 * pgQuery 3 * pgGUI 4 * pgAdmin 3 |
| 1. To ensure that all of our Models are represented in the database as Tables (using **Sequelize**), we use the following method:  * .synchronize() * .apply() * .updateTables() * .sync() | 1. If not explicitly specified, **Sequelize** will automatically add the following columns to a table:  * \_id, created, modified * id, createdAt, updatedAt * \_id, createdAt, modifiedAt * id, created, updated |
| 1. To return all the data for a specific Model (using **Sequelize**) we use the method:  * .find(…) * .getAll(…) * .findAll(…) * .get(…) | 1. MongoDB stores its data in the format of:  * NonSQL * JSON * BSON * Object(s) |
| 1. After "mongoDB" database engine running on you PC, the command which is used to start the mongo (db) shell from a terminal window:  * mongoose * mongod * mongodb * mongo | 1. The term “Collection” in MongoDB is corresponding to what in traditional RDMDS :  * Database file * Table * Database log * Record |
| 1. The call that is added after a query (using the framework module Mongoose) to make it return a promise:  * .catch(…) * . synchronize (…) * .exec(…) * .sync(…) | 1. If not explicitly specified, the framework module Mongoose will automatically add the following fields when store a created object:  * \_id, \_\_v * id, createdAt, updatedAt * \_id, createdAt, modifiedAt * id, \_v |

# Question 2 (4 Marks):

Using the ORM framework Sequelize, write code to specify a new **"Student" Model** that will correctly capture the following data (represented as a **JavaScript Object**). **NOTE:** Any special details about the data will be listed as **comments** beside the **related property**:

let data = {

studentNum: 012345678, // **primary key**

fullName: 'Christina Jeffers',

semester: 3,

registrationDate: 01/09/2015,

fullTime: true,

grantTotal: 8200.00

};

const Sequelize = require('sequelize');

var sequelize = new Sequelize('…', '…', '…', {… …});

**let Student = sequelize.define('Student', {**

**studentNum: {**

**type: Sequelize.INTEGER,**

**primaryKey: true**

**},**

**fullName: Sequelize.STRING, semester: Sequelize.INTEGER, registrationDate: Sequelize.DATE, fullTime: Sequelize.BOOLEAN, grantTotal: Sequelize.FLOAT // Or Sequelize.DOUBLE**

**});**

# Question 3 (8 Marks):

Using the ORM framework Sequelize, write valid code (method call only) to work with the "**Student**" Model defined in **Question 2** according to the following specifications:

1. Read all Student Numbers (studentNum) and Full Names (fullName) of the data in the "Student" model whose current Semester is 2. Once the operation completes successfully, write the full result to the console: [4 marks]

**Student.findAll({**

**attributes: ['studentNum', 'fullName'],**

**where: {**

**semester: 2**

**}**

**}).then((data)=>{**

**console.log(data);**

**});**

1. Update all **Full Time** (fullTime) Students whose current **Semester** is **1** and set their Grant Total (**grantTotal**) to **300.00**. Once this operation completes successfully, write the confirmation message "**update complete**" to the console: [4 marks]

**Student.update({**

**grantTotal: 300**

**}, {**

**where: { fullTime: true, semester: 1 }**

**}).then(()=>{**

**console.log("update Complete");**

**});**

# Question 4 (5 Marks):

Using the framework Mongoose in Node.js, write code to specify a new **"Vehicle" schema** and create a **model** that will correctly capture the following data (represented as a **JavaScript Object**). **NOTE:** Any special details about the data will be listed as **comments** beside the **related property**:

let data = {

VIN: 1HGBN45JXMN585, // **Unique indexes**

make: 'Honda',

model: 'Civic',

year: 2016,

used: true,

price: 9800.00

};

**const mongoose = require("mongoose");**

**let Schema = mongoose.Schema;**

**// Defining the Vehicle schema**

**let vehicleSchema = new Schema({**

**"VIN": {**

**"type": String,**

**"unique": true**

**},**

**"make": String,**

**model: String,**

**year: Number,**

**used: Boolean,**

**price: Number**

**});**

**// Creating the Vehicle model**

**var vehicle = mongoose.model('****vehicle', vehicleSchema);**

# Question 5 (8 Marks):

Using the framework Mongoose to work with the "**Vehicle**" Model defined in **Question 4** based on the following specifications.

1. Create a new Vehicle with the property values of the Vehicle object “data” in the Question 4. Save the created Vehicle and console log error message if error happens. [3 marks]

**var civicObj = new** **Vehicle ({**

**VIN: '1HGBN45JXMN585',**

**make: 'Honda',**

**model: 'Civic',**

**year: 2016,**

**used: true,**

**price: 9800.00**

**});**

**civicObj.save((err)=>{if (err) console.log("Error:" + err);});**

1. Retrieve all used car and console log the results or error message. [3 marks]

**Vehicle.find({ used: true })**

**.exec()**

**.then((cars) => {**

**console.log(cars);**

**}).catch((err) => {**

**console.log("Error: " + err);**

**});**

1. Update the created Vehicle’s price to 8800.00. [2 marks]

**Vehicle.update(**

**{ VIN: '1HGBN45JXMN585'},**

**{ $set: { price: 8800.00 } } ).exec();**