Quizlet

Week 10

Study online at quizlet.com/_8w2r3v

1. **Aside from using the very direct .findByPk() method, what is the most common method we use for querying a database?**
 	
How can we make that method more realistically useable?
	We save and await the output of the Model.findAll() method.
	
	Utilizing a where clause as an argument in the findAll method can allow us to get so specific with a query that we return one/just a few records.
	
	const cats = await Cat.findAll({
	where: {
	firstName: "Markov"
	}
	});
	console.log(JSON.stringify(cats, null, 2));

2. **Assuming we have two tables in our database connected via foreign keys, how can we pull in data from another table? How about a third related table?**	We would use the include key in our options object:
	const pet = Pet.findByPk(1, { include: [PetType, Owner] });
	console.log(
	pet.id,
	pet.name,
	pet.age,
	pet.petTypeId,
	pet.PetType.type,
	petOwners
	)
	
	To pull in multiple tables, we just nest more includes keys:
	const owner = Owner.findByPk(1, { include: { model: Pet, include: PetType } });
	console.log(
	same as above
	)

3. **Does the order of the flags we use in command line matter?**	Yes and no! So when we use the -U flag, it must be immediately followed by a username. However, where we place those two items, as long as they're together, doesn't matter.
4. **How do I give connect privileges to a specific user for a given database?** **How about for all users on a given instance of postgres?**	GRANT CONNECT ON DATABASE database FROM user; GRANT CONNECT ON DATABASE database FROM PUBLIC;
5. **How do we add the 'node-postgres' library to our application?**	`npm install pg`. From there we will typically use the Pool class associated with this library. That way we can run many SQL queries with one database connection (as opposed to Client, which closes the connection after a query)
6. **How do we create a database that we want a specific user to own?**	CREATE DATABASE database WITH OWNER user;
7. **How do we create the values in our database using a seed file?** **How do we rollback that change?**	- Running `npx sequelize-cli db:seed:all` will run all of our seeder files. - `npx sequelize-cli db:seed:undo:all` will undo all of our seeding.
8. **How do we identify if we're logged in as a regular user or a superuser?**	What follows the = when we're logged in: - If we're logged in as a super user, like we are when we just use the terminal command "psql", we'll see "=>". - However, logged in as a normal user we see "=#"
9. **How do we login to a specific database in psql?**	psql database name

10. **How do we use a color palette effectively when designing a web page?**	We'll focus on using our primary & secondary colors most of the time, with sparing use of the accent colors in our palette.
11. **How do we view all the tables within a given database, via the terminal?**	\dt
12. **How many fonts are optimal for a well-designed web page?**	At most 2.
13. **How would we create a seed file called add-cats?**	npx sequelize-cli seed:generate --name add-cats

14. **How would we go about deleting records from our DB using sequelize methods?**	We could either query for the record using our standard methods, i.e. findByPk, then use the .destroy() method, or we could use a where conditional as the argument of .destroy({ where: { specialSkill: 'jumping' } }):
	
	const cat = await Cat.findByPk(1);
	await cat.destroy();
	
	await Cat.destroy({ where: { specialSkill: 'jumping' } });

15. **If I want to create a user with a password in Postgres via the terminal (assume we're logged in as a superuser), how do we do that?**	CREATE USER user WITH PASSWORD 'password';
16. **If our project has 4 DB's and 11 tables, how many instances of our RDBMS do we need?**	Just one! All RDBMS can handle as many databases & tables as we need.
17. **If we a field to represent date/time in a table, what field type would we use?** **What about for a true/false field?**	TIMESTAMP BOOLEAN
18. **If we didn't want to create our databases for a new project via the sequelize-cli, how would we do that?**	If we take this approach, we need to make sure our user that we created has the `CREATEDB` option when we make them, since sequelize will attempt to make the databases with this user. This other approach would look like: - In psql: `CREATE USER example_user WITH PASSWORD 'badpassword' CREATEDB` - In terminal: `npx sequelize-cli db:create`

19. **If we have to create a many to many association, what is the specific command we will use in our model file?**
 	
What is the other part of building that relationship? And the final part?
	Owner.belongsToMany(models.Pet, columnMapping);
	
	We've got to create a column mapping:
	const columnMapping = {
	through: 'PetOwner', // joins table
	otherKey: 'petId', // key that connects to other table we have a many association with
	foreignKey: 'ownerId' // our foreign key in the joins table
	}
	
	Build a corresponding belongsToMany for the other file (in the above example, it was Pet).

20. **If we only want a single value returned, not as an array, what method would we use on a model to do that?**
What keys can we use therein?
	const cat = await Cat.findOne({
	order: [["age", "DESC"]],
	});
	console.log(JSON.stringify(cat, null, 2));
	
	We can use all the keys we used in findAll, i.e. where, includes, order, limit

21. **If we're doing a select join statement, how do we show columns from both tables?**	We would specify the table & field in the statement: SELECT friends.name AS friend_name , puppies.name AS puppy_name FROM friends JOIN puppies ON friends.puppy_id = puppies.id
22. **If we've performed a migration how do we undo it? How do we rollback all migrations?**	npx sequelize-cli db:migrate:undo npx sequelize-cli db:migrate:undo:all
23. **If we wanted to destroy all the records that have a specific value in the "specialSkill" field, how would we do that?**	We can also call `destroy` on the model itself. By passing in an object that specifies a where clause, we can destroy all records that match that query await Cat.destroy({ where: { specialSkill: 'jumping' } });

24. **If we want our down from the seed file to destroy all the records it created, how do we do that?**	down: (queryInterface, Sequelize) => {
	return queryInterface.bulkDelete('<<TableName>>', null, {});
	}

25. **If we want to destroy a record in a table, how would we do that via the Postgres terminal interface?**	DELETE FROM pets WHERE name IN ('Floofy', 'Doggo') OR id = 3;
26. **If we want to understand how SQL is optimizing the queries we run, what are the commands we would use?**	EXPLAIN ANALYZE

27. **If we want to update a record via javascript what are the methods & the syntax for doing so?**	First, we would need to either create the record or query for it, at which point we can begin to modify it:
	// Get a reference to the cat record that we want to update (here just the cat with primary key of 1)
	const cat = await Cat.findByPk(1);
	// Change cat's attributes.
	cat.firstName = "Curie";
	cat.specialSkill = "jumping";
	cat.age = 123;
	// Save the new name to the database.
	await cat.save();

28. **If we want to use create a pool in a function we're writing, how would we do that?**
 	
What would have to have been done first?
	const { Pool } = require('pg');
	// If we need to specify a username, password, or database, we can do so when we create a Pool instance, otherwise the default values for logging in to psql are used:
	const pool = new Pool({ username: '<<username>>', password: '<<password>>', database: '<<database>>'})
	
	We would have to have npm installed the 'pg' library.

29.	**If we want to validate a new record or updated record before it hits the database, how do we do that?**
	
	How do we do that?
		We would want to make changes to our model file, adding validations:
		specification: {
		type: DataTypes.TEXT,
		validate: {
		notEmpty: {
		msg: 'The specification cannot be empty'
		},
		len: {
		args: [10, 100]
		msg: 'The specifcation must be between 10 and 100 characters'
		}
		}
		}

30.	**In terms of designing our projects & portfolio for recruiters, what features should we includde?**	- Plentiful seed data - A favicon - A demo login (they won't give you an email address/create a user) - Remove all console logs & errors - Include a score card
31.	**Once we've finished altering a migration file, how do we push that migration?**	npx sequelize-cli db:migrate
32.	**Say we have a record we need to change, how would we do that via the Postgres terminal interface?**	UPDATE pets SET (name, breed) = ('Floofy', 'Fluffy Dog Breed') WHERE id = 4;
33.	**Say we've created a database, what's the next step?** **How do we do that?**	Once a database has been created, we need tables to hold actual data/records. The syntax for creating a table is: CREATE TABLE {table name} ({columnA} {typeA}, {columnB} {typeB}, etc...);

34. **Say we want more than a basic "or" among values, or "and" pulling in multiple fields for a "where" clause in a findAll - what would we do?**	Take advantage of the Op comparison operators! To do that we would first need to import Op using:
	`const { Op } = require("sequelize");`
	
	From there we can take advantage of multiple complex operators in our where clauses:
	where: {
	firstName: {
	// All cats where the name is not equal to "Markov"
	// We use brackets in order to evaluate Op.ne and use the value as the key
	[Op.ne]: "Markov"
	},
	},
	
	where: {
	// The array that Op.and points to must all be true
	// Here, we find cats where the name is not "Markov" and the age is 4
	[Op.and]: [
	{ firstName: { [Op.ne]: "Markov" } },
	{ age: 4 },
	],
	},
	
	where: {
	// One condition in the array that Op.or points to must be true
	// Here, we find cats where the name is "Markov" or where the age is 4
	[Op.or]: [
	{ firstName: "Markov" },
	{ age: 4 },
	],
	},

35. **So we've defined our key relationships of one to one or one to many - are we done with creating associations?**	Nope! We still gotta do our associations in our model file: - `Instruction.belongsTo(models.Recipe, { foreignKey: 'recipeId' });` - `Recipe.hasMany(models.Instruction, { foreignKey: 'recipeId' });`
36. **Using the OOP approach is most appropriate with which step of database design?**	The first step, designing the entities: - If you wanted to model this information using classes, what classes would you make? Those are generally going to be the tables that are created in your database. - The attributes of your classes are generally going to be the fields/columns that we need for each table.
37. **We've created a database with a table - how do we create records inside that table using the Postgres terminal interface?**	INSERT INTO table_name VALUES (column1_value, colum2_value, column3_value), (column1_value, colum2_value, column3_value), (column1_value, colum2_value, column3_value);
38. **What are the arguments of the JSON method .stringify?**	1: the value we want to stringify 2: a replacer function or array (we don't use that often.. or ever) 3: a spacer, which we can provide as a string of spaces we want to includes, a number of spaces we want to include, or another spacer of our choice (i.e. "\n")

39. **What are the basic parts of a simple query?**	SELECT */fields we want FROM tableName
40. **What are the big, overarching steps of designing a database?**	1. Define the entities. What data are are you storing, what are the fields for each entity? 2. Identify primary keys. Most of the time these will be ids that you can generate as a serial field, incrementing with each addition to the database. 3. Establish table relationships. Connect related data together with foreign keys. Know how we store these keys in a one-to-one, one-to-many, or many-to-many relationship.

41. **What are the commands we need to use to create a bunch of new records in a seed file?**	up: (queryInterface, Sequelize) => {
	return queryInterface.bulkInsert('<<TableName>>', [
	{ field1: value1a, field2: value2a },
	{ field1: value1b, field2: value2b },
	{ field1: value1c, field2: value2c }
	]);
	}

42. **What are the common keys we would use in the object we pass in as a part of a findAll? Include thinking through what the values for each would be.**	where: conditional clause order: an array with the fields that we want to order by i.e. order: [["age", "DESC"], "firstName"] limit: and integer of the number of results we want - even if it's 1, we'll still get an array returned
43. **What are the common text fields we would use when creating a table?**	VARCHAR(num of characters) TEXT
44. **What are the important considerations for text on a page?**	Use large bold headers Smaller length of body text is easier to digest No more than 2 font sizes Text should have padding between it and any border
45. **What are the important CSS styles & features to use in websites we design?**	- Rounding the corners of buttons and modals - Applying CSS transitions when elements appear on or disappear from the page - Using shadows to make elements stand out - Making sure page elements feel interactive, such as changing background colors or cursors on hover
46. **What are the main advantages of using a transaction?**	Unless every operation succeeds, the entire transaction fails, and no changes are made to the database While we're altering records with the transaction, those records are locked - no other software/function may alter them
47. **What are the major commands we use in a transaction?**	BEGIN - starts a transaction ROLLBACK - rollsback the changes in the current transaction COMMIT - assuming all parts of the transaction were successful, it's committed to the database
48. **What are the parts of good code hygiene?**	- Readable code - Use comments - Standardize your naming conventions - Make sure code is correctly indented - Refactor
49. **What are the parts of the seed data file?**	`up` indicates what to create when we seed our database, `down` indicates what to delete if we want to unseed the database.

50. **What are the rules for naming something in PostgreSQL?**	Names within postgres should generally consist of only lowercase letters, numbers, and underscores. Tables are always plural.
51. **What are the steps of setting Sequelize for a new project?**	1. To start a new project we use our standard npm initialize statement - `npm init -y` 2. Add in the packages we will need (sequelize, sequelize-cli, and pg) - `npm install sequelize@^5.0.0 sequelize-cli@^5.0.0 pg@^8.0.0` 3. Initialize sequelize in our project - `npx sequelize-cli init` 4. Create a database user with credentials we will use for the project - `psql` - `CREATE USER example_user WITH PASSWORD 'badpassword'` 5. Here we can also create databases since we are already in postgres - `CREATE DATABASE example_app_development WITH OWNER example_user` - `CREATE DATABASE example_app_test WITH OWNER example_user` - `CREATE DATABASE example_app_production WITH OWNER example_user` 6. Double check that our configuration file matches our username, password, database, dialect, and seederStorage (these will be filled out for you in an assessment scenario)
52. **What are the two ways we can insert a seed file into a database via the terminal?**	Both use built in command line tools, either the < or the	: - `psql -d {database} < {sql filepath}` - `cat {sql filepath} | psql -d {database}`

53. **What are the two way we can make new data for our database?**	We can use the method .build & .save or using the method .create, which combines .build & .save into one method. For both, the method is performed on the model:
	const newCat = Cat.build({
	firstName: 'Markov',
	specialSkill: 'sleeping',
	age: 5
	});
	await newCat.save();
	
	const newerCat = await Cat.create({
	firstName: 'Whiskers',
	specialSkill: 'sleeping',
	age: 2
	})

54. **What commands do we use to create an index?** **How would we destroy it?**	CREATE INDEX index_name ON table_name (column_name); DROP INDEX index_name
55. **What does adding the command ANALYZE in addition to EXPLAIN do?**	We can also use the ANALYZE command with EXPLAIN, which will actually run the specified query. Doing so gives us more detailed information, such as the milliseconds it took our query to execute as well as specifics like the exact number of rows filtered and returned.
56. **What does using the EXPLAIN command do?**	- EXPLAIN gives us information about how a query will run (the query plan) - It gives us an idea of how our database will search for data as well as a qualitative comparitor for how expensive that operation will be. Comparing the cost of two queries will tell us which one is more efficient (lower cost).

57. **What field type would we use building a table if we needed a number?**	There are a few: - `SMALLINT`: signed two-byte integer (-32768 to 32767) - `INTEGER`: signed four-byte integer (standard) - `BIGINT`: signed eight-byte integer (very large numbers) - `NUMERIC`: or `DECIMAL`, can store exact decimal values
58. **What is a primary key? How does it relate to a foreign key?**	A primary key is a unique identifier for a table row (aka a record). A foreign key uses that primary key to create a referential relationship between two pieces of data. We might reference a primary key from our Pets table when making our Pet_Owners table, where we will utilize that primary key in a foreign key field.
59. **What is a query plan?**	Information about how a query will run
60. **What is a RDBMS?**	Relational Database Management System - a software application that you run that your programs can connect to so that they can store, modify, and retrieve data.
61. **What is relational data?**	Relational data is information that is connected to other pieces of information.
62. **What is returned by the findAll method? How can we make that more usable?**	Sequelize is returning an array of instance objects in users. We can make that more usable in a few ways, but the most common way we'll do it so stringify the results: console.log(JSON.stringify(cats, null, 2));
63. **What is the basic syntax of a select statement utilizing a condition?**	We use a WHERE statement: SELECT name, breed, weight_lbs FROM puppies WHERE weight_lbs > 50;
64. **What is the command to view all the databases we have access to in our Postgres instance?**	\l or \list
65. **What is the field type we would use building a table to create an autoincrementing field? i.e. for an ID**	SERIAL
66. **What is the primary method we make relationships among data?** **How does it work?**	The use of foreign keys! Foreign keys allow us to reference other pieces data through a unique key that corresponds to that data - their primary key.
67. **What is the syntax for a command to create a model using sequelize's cli?**	npx sequelize-cli model:generate --name Cat --attributes firstName:string,specialSkill:string

68. **What is the syntax for a findAll where clause? How do we specify "or"? How about "and"?**	We create an object, in which the first key is a "where:", with a value of another object, in which we input the fields we want to use in our conditional statement:
	const cats = await Cat.findAll({
	where: {
	firstName: "Markov"
	}
	});
	If we want to specify an "or" relationship between values, we would input an array for the key of the field we're using in our conditional:
	firstName: ["Markov", "Curie"]
	For an "and" relationship, where we want to specify more than one field, we just put commas between key fields:
	where: {
	firstName: "Markov",
	age: 4
	}

69. **What is the syntax for creating a transaction?**	...
70. **What is the syntax for seeding a database with an sql file via the terminal with "<"?**	psql -d {database} < {sql filepath}
71. **What is the syntax for seeding a database with an sql file via the terminal with "|"?**	cat {sql filepath} | psql -d {database}

72. **What is the syntax for using an Op complex operator?**	Brackets around our Op dot operator, colon, another set of brackets, and a set of operands:
	[Op.or]: [
	{ firstName: "Markov" },
	{ age: 4 },
	],
	
	OR
	where: {
	// Find all cats where the age is greater than 4
	age: { [Op.gt]: 4 },
	}

73. **What is the syntax of creating a sequelize transaction?**

Inside of a try block, we create await a method "sequelize.transaction", where we pass in an async function, with transaction or tx as the parameter. Inside that function we perform whatever actions we need to, then open a catch block - don't forget you can access the error message via dot notation. Finally we close the database connection with await seqeulize.close():

```
async function main() {
try {
// Do all database access within the transaction.
await sequelize.transaction(async (tx) => {
// Fetch Markov and Curie's accounts.
const markovAccount = await BankAccount.findByPk(
1, { transaction: tx },
);
const curieAccount = await BankAccount.findByPk(
2, { transaction: tx }
);
// No one can mess with Markov or Curie's accounts until the
// transaction completes! The account data has been locked!
// Increment Curie's balance by $5,000.
curieAccount.balance += 5000;
await curieAccount.save({ transaction: tx });
// Decrement Markov's balance by $5,000.
markovAccount.balance -= 5000;
await markovAccount.save({ transaction: tx });
});
} catch (err) {
// Report if anything goes wrong.
console.log("Error!");
for (const e of err.errors) {
console.log(
`${e.instance.clientName}: ${e.message}`
);
}
}
await sequelize.close();
}
```

74. **What is the unit of measurement when judging if the margin between a set of elements is adequate?**

Would a lowercase letter "a" of the same font size as the body text of the web page.

75. **What is the way to login into Postgres via the command line?**

Where does it take us?

psql

The `psql` command by default will try to connect to a database and username that matches your system's username

76. **What is wrong with the following table creation?**

CREATE TABLE {table name} (
{columnA} {typeA},
{columnB} {typeB},
);

It ends with a comma - the last value in a comma separated list should not have a comma.

77. **What must our database schema accommodate when creating a one-to-one or one-to-many data relationship?**	It's easy, we've just got to create foreign keys on our tables so we can create them.
78. **What must we do before using methods like .save, .findByPk, or .create?**	...
79. **What's the downside of creating an index?**	Making an index is not always the best approach. Indices allow for faster lookup, but slow down record insertion and the updating of associated fields, since we not only have to add the information to the table, but also manipulate the index.
80. **What's the flag we use when we want a prompt to input a user password?**	-W
81. **What's the syntax for removing connection privileges from the user for a given database?**	REVOKE CONNECT ON DATABASE database FROM PUBLIC;
82. **What type of data relationship requires a join table?**	Many-to-many: With a many-to-many relationship, each record could be connected to multiple records, so we have to create a join table to connect these entities. A record on this join table connects a record from one table to a record from another table.

83. **When building the down of a seed file, how do we tell it to remove specific files?**	down: (queryInterface, Sequelize) => {
	return queryInterface.bulkDelete('<<TableName>>', {
	field1: [value1a, value1b, value1c] //...etc.
	});
	}

84. **When creating a user that we need to be able to create a database, how do I do that?**	CREATE USER user WITH PASSWORD 'password' CREATEDB;
85. **When creating a user that we need to be a super user, how do I do that?**	CREATE USER user WITH PASSWORD 'password' SUPERUSER;

86. **When creating validations in our migration files, how do we pass along an error to our users if the validation is not met?**	The "msg:" key:
	validate: {
	notEmpty: {
	msg: 'The specification cannot be empty'
	}
	}

87. **When designing a web page, how do we ensure a consistent theme? How do we avoid color clashes?**	For both we utilize a color palette.
88. **When using a text field, what is the character we would use for a wildcard?**	%
89. **When we use model:generate, what is created?**	A migration file & a model file.

90. **When we want to make sure that a field is not nullable, or that it's unique, how would we do that?**	We would open the migration file, find the field we want to edit, and open curly braces, then add the different validations:
	name: {
	allowNull: false,
	type: Sequelize.STRING(20),
	unique: true
	},
	createdAt: {
	allowNull: false,
	type: Sequelize.DATE
	},

91. **When would probably want to avoid an index?**	- The tables are small (then it's not necessary, it'll be fast without an index) - We are updating the table frequently, especially the associated columns - The column has many NULL values
92. **When would we use an index?**	If we are constantly looking up records in a table by a particular field (such as username or phone number), we would want to create an index
93. **When writing a join statement, how do we tell the JOIN to connect records from different tables?**	We specify the field where the primary key of one table matches the foreign key on the other table: SELECT * FROM friends JOIN puppies ON friends.puppy_id = puppies.id
94. **Why would we use an index?**	- An index can help optimize queries that we have to run regularly. If we are constantly looking up records in a table by a particular field (such as username or phone number), we can add an index in order to speed up this process. - An index maintains a sorted version of the field with a reference to the record that it points to in the table (via primary key). If we want to find a record based on a field that we have an index for, we can look through this index in a more efficient manner than having to scan through the entire table (generally O(log n) since the index is sorted, instead of O(n) for a sequential scan).
95. **Why would we want to create a pool in a javascript function?**	By creating a pool we can run SQL queries on our DB via the pg library: const airportsByNameSql = ` SELECT name, faa_id FROM airports WHERE UPPER(name) LIKE UPPER($1) `; async function selectAirportsByName(name) { const results = await pool.query(airportsByNameSql, [`%${name}%`]); console.log(results.rows); pool.end();
96. **Will extra comma's in a table creation statement or values insertion statement cause issues?**	Yes - SQL does not like trailing commas, because the language takes that as an indicator that more data is coming.