## Quiz: Lesson 1 Solution

1. Why do they call it a relational database?

   We call it a relational database, because the tables within the database are associated with each other. This association can be created with primary/foreign keys and various syntax.

2. What is SQL?

   SQL stands for "Structured Query Language" and it is used to manage the operations of a relational database.

3. There are two predominant views into a relational database. What are they, and how are they different?

   The two predominant views are the data and schema views. Data view displays like a spreadsheet, with the table columns at the top and rows of data per each object instance.

   A schema view shows us the column names and the value type of each column.

4. In a table, what do we call the column that serves as the main identifier for a row of data? We're looking for the general database term, not the column name.

   We call this the "primary key".

5. What is a foreign key, and how is it used?

   A foreign key is the identifier that connects an association with the models involved. The foreign key is always on the "many" side and is always in an integer type.

6. At a high level, describe the ActiveRecord pattern. This has nothing to do with Rails, but the actual pattern that ActiveRecord uses to perform its ORM duties.

   ActiveRecord is a way to access the database. A database table is related to a class. An object of that class is created as a row in the table. This object can have different attribute values shown as the columns in the table. We can create, retrieve, update, and delete the object instances by altering the database table.

7. If there's an ActiveRecord model called "CrazyMonkey", what should the table name be?

   By Rails convention, the table name should be "crazy_monkeys" by using `"CrazyMonkey".tableize`.

8. If I'm building a 1:M association between `Project` and `Issue`, what will the model associations and foreign key be?

   In the `Project` model:

   ```
   class Project < ActiveRecord::Base
     has_many :issues, foreign_key: :project_id
   end
   ```

   In the `Issue` model:

   ```
   class Issue < ActiveRecord::Base
     belongs_to :project, foreign_key: :project_id
   end
   ```

   The foreign key will be `project_id`.

9. Given this code

   ```
   class Zoo
     has_many :animals
   end
   ```

   **What do you expect the other model to be and what does database schema look like?**

   The Animal model:

   ```
   class Animal
     belongs_to :zoo
   end
   ```

   The database schema will have tables named:

   ```
   - `zoos` table with a primary key of `id`
   - `animals` table with a primary key of `id` and foreign key of `zoo_id`
   ```

   **What are the methods that are now available to a zoo to call related to animals?**

   ```
   - `zoo.animals` will return all a list of all of the animals.
   - `zoo.animals.first` will return the first row of data in the `animals` table.
   ```

   You can also iterate through the list of a zoo's animals and display certain properties of the animals.

   **How do I create an animal called "jumpster" in a zoo called "San Diego Zoo"?**

   In the Rails console, enter the following commands:

   ```
   zoo = Zoo.create(name: 'San Diego Zoo')         # => Sets the primary id of 1 for San Diego Zoo
   animal = Animal.create(name: 'jumpster', zoo_id: 1)
   ```

10. What is mass assignment? What's the non-mass assignment way of setting values?

    Mass assignment allows us to assign multiple values to attributes with only a single assignment operator.

Mass assignment:

```
Post.new(title: 'My first post', topic: 'Life')
Post.create(title: 'My first post', topic: 'Life')
```

Non-mass assignment:

```
post = Post.new
post.title = 'My first post'
post.topic = 'Life'
```

11. What does this code do? `Animal.first`

    This will return the first row of data for the first Animal instance object in the `animals` table.

12. If I have a table called "animals" with columns called "name", and a model called `Animal`, how do I instantiate an animal object with name set to "Joe". Which methods makes sure it saves to the database?

```
animal = Animal.create(name: 'Joe')         # => Using "create" will hit the database and automatically saves it.

animal = Animal.new(name: 'Joe')            # => Using "new" will require you to use the save method afterwards to save it to the databa
animal.save
```

13. How does a M:M association work at the database level?

    On the database level of a M:M association, we use a join table to support it. Both of the primary models will each have a 1:M association with the join table.

    By using the `has_many :through` technique, we are able to create an indirect M:M association with the two primary models.

14. What are the two ways to support a M:M association at the ActiveRecord model level? Pros and cons of each approach?

    The two ways to support a M:M association are the `has_many :through` and `has_and_belongs_to_many` methods.

    `has_many :through` requires an explicit join model and a join table, but it is more flexible and we can add additional attributes to the join table.

    `has_and_belongs_to_many` doesn't require a join model but still requires a join table, but it is less flexible and we cannot add additional attributes to the join table.

    Note: always use `has many :through`, as `has_and_belongs_to_many` will be deprecated in the future.

15. Suppose we have a `User` model and a `Group` model, and we have a M:M association all set up. How do we associate the two?

    We will need to use a join model(UserGroup) and table(user_groups) in this situation.

    The `User` model:

```
class User < ActiveRecord::Base
    has_many :user_groups, foreign_key: :user_id
    has_many :groups, through: :user_groups
end
```

    The `User_Group` model:

```
class UserGroup < ActiveRecord::Base
    belongs_to :user, foreign_key: :user_id
    belongs_to :group, foreign_key: :group_id
end
```

    The `Group` model:

```
class Group < ActiveRecord::Base
    has_many :user_groups, foreign_key: :group_id
    has_many :users, through: :user_groups
end
```

You marked this topic or exercise as completed.
Take me to the Next Lesson
Quiz: Lesson 1

# Formatting Help

### Normal Markdown & GFM

Github Flavored Markdown

---

****bold****

---

~~\~\~strike through\~\~~

---

`` `single line of code` ``

---

Supports Emoji

### Code Blocks