Monday: One to Many Relationships

One to Many Relationship

Users have different levels of access in the application. We need to connect the users access with the User and the their level of access in our application.

models.py

```
# ...
class Role(db.Model):
    __tablename__ = 'roles'

    id = db.Column(db.Integer,primary_key = True)
    name = db.Column(db.String(255))

    def __repr__(self):
        return f'User {self.name}'
```

We create a Role class that will define all the different roles. We create two columns for the ID and the name.

Creating the relationship.

Relationships are between two models that allow the models to reference each other. We want to create a `One-Many` relationship between our models. This is that one role can be shared by many different users.

We need to create a connection between roles and users by using a Foreign Key. A Foreign key is a field in one table that references a primary key in another table.

models.py

```
# ...
class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer,primary_key = True)
    username = db.Column(db.String(255))
    role_id = db.Column(db.Integer,db.ForeignKey('roles.id'))


    def __repr__(self):
        return f'User {self.username}'

# .....
```

We create a new column in our `User` model `role_id` and we give it an `Integer` type. We also pass in the `db.ForeignKey` class and pass in `'roles.id'` argument. This tells SQLAlchemy that this is a foreign key and is the ID of a `Role` model.

We also need to define that relationship inside our Roles model.

app/models.py

```
# ....
class Role(db.Model):
    __tablename__ = 'roles'

    id = db.Column(db.Integer,primary_key = True)
    name = db.Column(db.String(255))
    users = db.relationship('User',backref = 'role',lazy="dynamic")


    def __repr__(self):
        return f'User {self.name}'
```

We use `db.relationship` to create a virtual column that will connect with the foreign key. We pass in 3 arguments. The first one is the class that we are referencing which is `User`.
Next `backref` allows us to access and set our `User` class. We give it the value of `role` now because when we want to get the role of a user instance we can just run `user.role`.
Lazy parameter is how SQLAlchemy will load our projects. The lazy option is our objects will be loaded on access and filtered before returning.

manage.py

```
#....
from app.models import User,Role
#....
@manager.shell
def make_shell_context():
    return dict(app = app,db = db,User = User, Role = Role )
if __name__ == '__main__':
    manager.run()
```

We import the `Role` class and pass it into our shell context. Now we can update our database schema.

```
(virtual)$ python3.6 manage.py shell
>>> db.drop_all()
>>> db.create_all()
```

We first have to drop the database and recreate it with the new model. We need to create the roles in our application

```
>>> role_admin = Role(name = 'Admin')
>>> role_user = Role (name = 'User')
>>> db.session.add_all([role_admin,role_user])
>>> db.session.commit()
```

We create two roles and save them to our database.We can confirm these items have been added to our database .

```
watchlist=# select * from roles;
 id | name
----+-------
  1 | Admin
  2 | User
(2 rows)
```

We can now create Users.

```
>>> user_james = User (username = "James Muriuki",role = role_admin)
>>> user_christine = User(username = "Christine",role = role_user)
>>> db.session.add_all([user_james,user_christine])
>>> db.session.commit()
```

We create our user instances by passing the `username`. We also pass in the `role`value and equate it to the `Role` instances we created above. We can confirm the enries have been added by checking the PostgreSQL server.

```
watchlist=# select * from users;
 id |   username    | role_id
----+---------------+---------
  1 | James Muriuki |       1
  2 | Christine     |       2
(2 rows)
```

We can query our data now.

```
>>> first_user = User.query.first()
>>> first_user.role
User Admin
```

We target the first entry and then we check what role that user has. You can find the application at this point from here https://github.com/mbuthiya/watchlist/tree/19-one-to-many-relationships