

# Retrieval

In this lesson, we will discover how to retrieve objects from the database using the "query" object of the model class.

## WE'LL COVER THE FOLLOWING

- Common methods of retrieval in the `query` object
  - `query.all()`
    - Explanation
  - `query.first()`
    - Explanation
  - `query.get()`
    - Explanation
  - `query.filter_by()`
    - Explanation
  - `query.filter()`
    - Explanation

In the previous lesson, we inserted elements in the database by creating objects of the **models**. In this lesson, we will retrieve these objects from the database using the **SQLAlchemy ORM**.

## Common methods of retrieval in the `query` object

The `query` object is a member variable of the `Model` class. This object provides us with a method to execute the `SELECT` statement of `SQL`. We can find details on all the methods that can be used on this object in the [SQLAlchemy docs](#).

`query.all()` #

This method will retrieve all entries of the Model class on which it is called. A

demonstration is given below.

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class User(db.Model):
    email = db.Column(db.String, primary_key=True, unique=True, nullable=False)
    password = db.Column(db.String, nullable=False)

db.create_all()
archie = User(email = "archie.andrews@email.com", password = "football4life")
veronica = User(email = "veronica.lodge@email.com", password = "fashiondiva")
db.session.add(archie)
db.session.add(veronica)

try:
    db.session.commit()
except Exception as e:
    db.session.rollback()

print(User.query.all())
```



Retrieval Using "query.all()"

## Explanation #

- In **lines 14 - 22**, we have performed the steps for insertion in the database (as explained in the previous lesson).
- Then, in **line 24**, we have called the `query.all()` function on the `User` model class.
- We can observe that **all** the objects we inserted have been retrieved and printed on the console.

✎ **Note:** the `print()` function shows the following output:

```
[<User archie.andrews@email.com>, <User veronica.lodge@email.com>]
```

This output shows a list of `User` objects. The `email` is also shown because it is the `primary_key` column of the `User` model.

## query.first() #

This method will retrieve the first record of the Model class on which it is called. A demonstration is given below.

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class User(db.Model):
    email = db.Column(db.String, primary_key=True, unique=True, nullable=False)
    password = db.Column(db.String, nullable=False)

db.create_all()
archie = User(email = "archie.andrews@email.com", password = "football4life")
veronica = User(email = "veronica.lodge@email.com", password = "fashiondiva")
db.session.add(archie)
db.session.add(veronica)

try:
    db.session.commit()
except Exception as e:
    db.session.rollback()


print(User.query.first())
```



Retrieval Using "query.first()"

## Explanation #

- In the program given above, in **line 24**, we have called the `query.first()` method and printed the output.
- We can observe that the *first* record that we added, in **line 16**, is printed when the function is called.

 **Do it yourself:** *reverse* the order of **insertion** for the objects and observe the new output!

(Hint: change the order of lines 16 and 17)

## query.get() #

This method takes one argument: the value of the `primary_key` column. The record corresponding to this value will be retrieved from the database. A demonstration is given below:

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class User(db.Model):
    email = db.Column(db.String, primary_key=True, unique=True, nullable=False)
    password = db.Column(db.String, nullable=False)

db.create_all()
db.session.add(User(email = "archie.andrews@email.com", password = "football4life"))
db.session.add(User(email = "veronica.lodge@email.com", password = "fashiondiva"))

try:
    db.session.commit()
except Exception as e:
    db.session.rollback()


user = User.query.get("veronica.lodge@email.com")
print(user)
print(user.email)
print(user.password)
```



Retrieval Using "query.get()"

## Explanation #

In **line 22**, the `query.get()` method is called on the `User` model with a parameter. In the `User` model, the `primary_key` column is the `email`. Therefore, the `get()` method looks for an `email` corresponding to the parameter value and fetches the corresponding record. It returns a `NoneType` object if that `email` does not exist.

 **Do it yourself:** call the `query.get()` method on `email` value that does not exist in the database and observe the output!

## `query.filter_by()` #

The `filter_by()` method is somewhat similar to the `get()` method. They are

The `filter_by()` method is somewhat similar to the `get()` method. They are similar in that `filter_by()` also has the value of columns as parameters. This method is more useful as it can take the value of **multiple** columns as keyword arguments, i.e. the name of these arguments has to be provided. Another difference is that it does not return records. Instead, it returns a `BaseQuery` object.

This object represents the `SQL` query to be executed. Then, we can call `first()` or `all()` on this query to obtain the result(s). A demonstration of this is given below.

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class User(db.Model):
    email = db.Column(db.String, primary_key=True, unique=True, nullable=False)
    password = db.Column(db.String, nullable=False)

db.create_all()
db.session.add(User(email = "archie.andrews@email.com", password = "football4life"))
db.session.add(User(email = "veronica.lodge@email.com", password = "fashiondiva"))

try:
    db.session.commit()
except Exception as e:
    db.session.rollback()

user = User.query.filter_by(password = "football4life")
print(user) #Base Query Object
print('*'*20)

## Add first() Method to Retrieve the First Result from Query
user = User.query.filter_by(password = "football4life").first()
print(user)
print(user.email)
print(user.password)
```



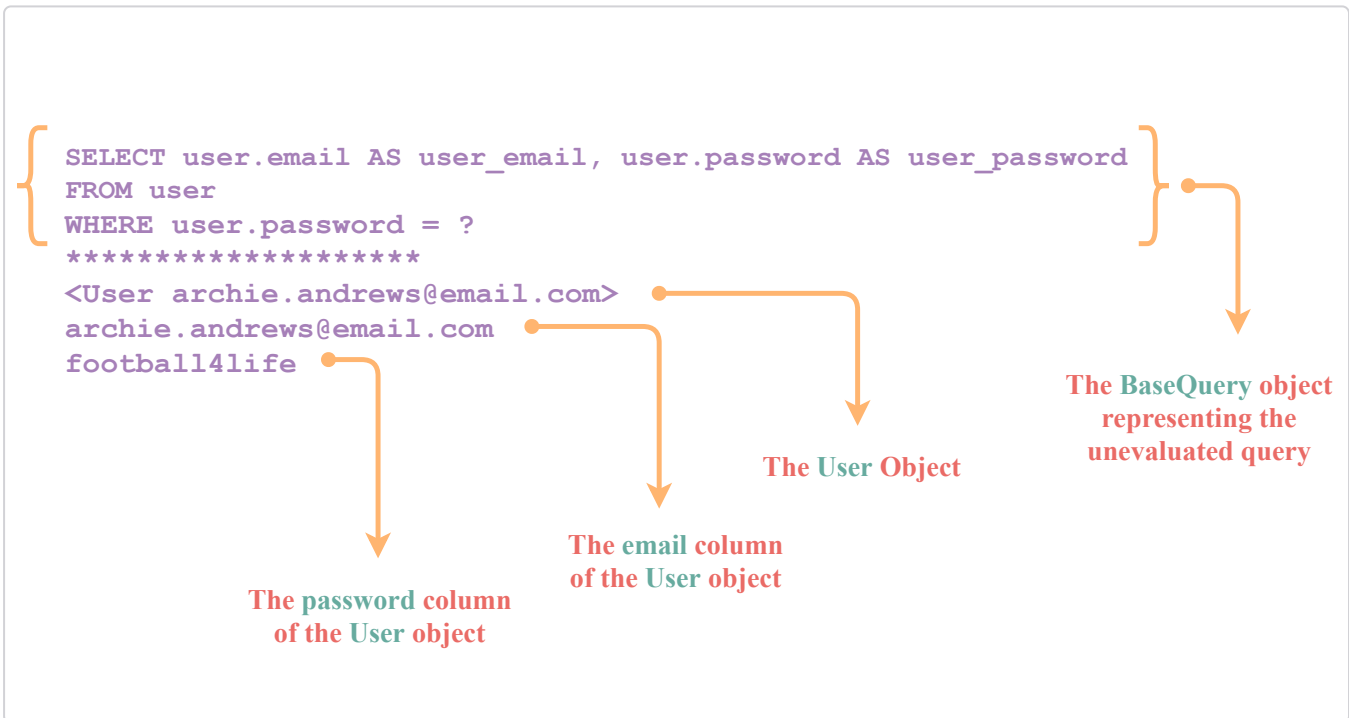
Retrieval Using "query.filter\_by()"

## Explanation #

- In **line 22**, the function `filter_by()` is called and the `password` is given as a keyword argument.

The result of this function is displayed in **line 23**. We can see that it is

- The result of this function is then printed in **line 23**. We can see that it is an `SQL` query having a `WHERE` clause with respect to `user.password`.
- We then call this function at **line 27** again but this time chain the `first()` method at the end.
- The output of this is an object of the `User` model which is the first record satisfying the query conditions. We can observe the printed output in **lines 28 - 30**.



## `query.filter()` #

The `filter()` and `filter_by()` methods are quite similar and often get confused with one another. However, the `filter()` method takes expressions as parameters. This method takes expressions as parameters. Moreover, these expressions can be applied to the columns of the model or else

`ColumnOperators` can also be used. Some of the most commonly used `ColumnOperators` are:

- `contains()`
- `endswith()`
- `startswith()`
- `like()`

Please refer to the `ColumnOperators` [documentation](#) for the complete list.

⚡ Note:

1. The `filter()` function does not take keyword arguments.  
Therefore, the following code will give an error because it should be `User.email` instead of `email`.

```
User.query.filter(email == "veronica.lodge@email.com")
```

2. The correct use of this example is given below:

```
User.query.filter(User.email == "veronica.lodge@email.com")
```

3. As shown in the snippet above, the arguments of the `filter()` function use `==` instead of `=` because it takes **expressions** as parameters.
4. The `filter()` function also returns a `BaseQuery` object, not a record.  
Therefore, `first()` or `all()` is chained after the query to obtain (a) `User` object(s).

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class User(db.Model):
    email = db.Column(db.String, primary_key=True, unique=True, nullable=False)
    password = db.Column(db.String, nullable=False)

db.create_all()
db.session.add(User(email = "archie.andrews@email.com", password = "football4life"))
db.session.add(User(email = "veronica.lodge@email.com", password = "fashiondiva"))

try:
    db.session.commit()
except Exception as e:
    db.session.rollback()

user = User.query.filter(User.email == "veronica.lodge@email.com")
print(user) #Base Query Object
print('*'*20)

# Add first() Method to Retrieve the First Result from Query
user = User.query.filter(User.email == "veronica.lodge@email.com").first()
print(user)
print('*'*20)

# A Query with Multiple Expressions
```

```
# A Query with Multiple Expressions
user = User.query.filter(User.email == "archie.andrews@email.com", User.password.contains("ar"))
print(user)

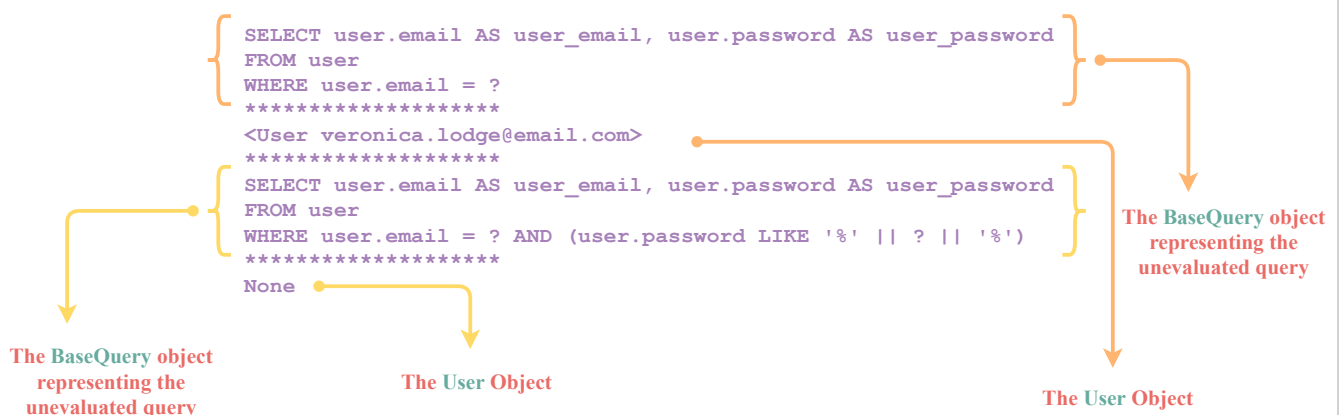
print('*'*20)
print(user.first())
```



Retrieval Using "query.filter()"

## Explanation #

1. In **line 22**, we call the `filter()` function comparing the values for `User.email`. Then, we print the `BaseQuery` object returned from **line 23**. Moreover, we can observe from the output that the `SQL` query contains a `WHERE` clause on the `user.email` value as well.
2. We then again call this function in **line 27** but this time chain the `first()` method at the end and observe the output printed in **line 28**.
3. In **line 32**, we have a query with multiple arguments: `User.email` and `User.password`. We can observe the output through the `print` statement in **line 33** that the output from the `SQL` query contains a `WHERE` clause on the `user.email` as well as `user.password`.
4. Notice that we used the `endswith()` method from the `ColumnOperators` class on the `User.password` column.



Quick Quiz!



Q

You are given the following **model** class:

```
class Student(db.Model):  
    roll_number = db.Column(db.Integer, primary_key = True, unique = True)  
    name = db.Column(db.String, nullable = False)  
    batch = db.Column(db.String, nullable = False)
```

Which of these queries will return all elements of the **Student** model having the **batch** column equal to **"2015"** and a **name** that ends with **"ah"**.

COMPLETED 0%

1 of 1



In the next lesson, we will learn how to **update** and **delete** data from the database.