# Python Flask and Forms

# Database Models in Python Flask (ORM)

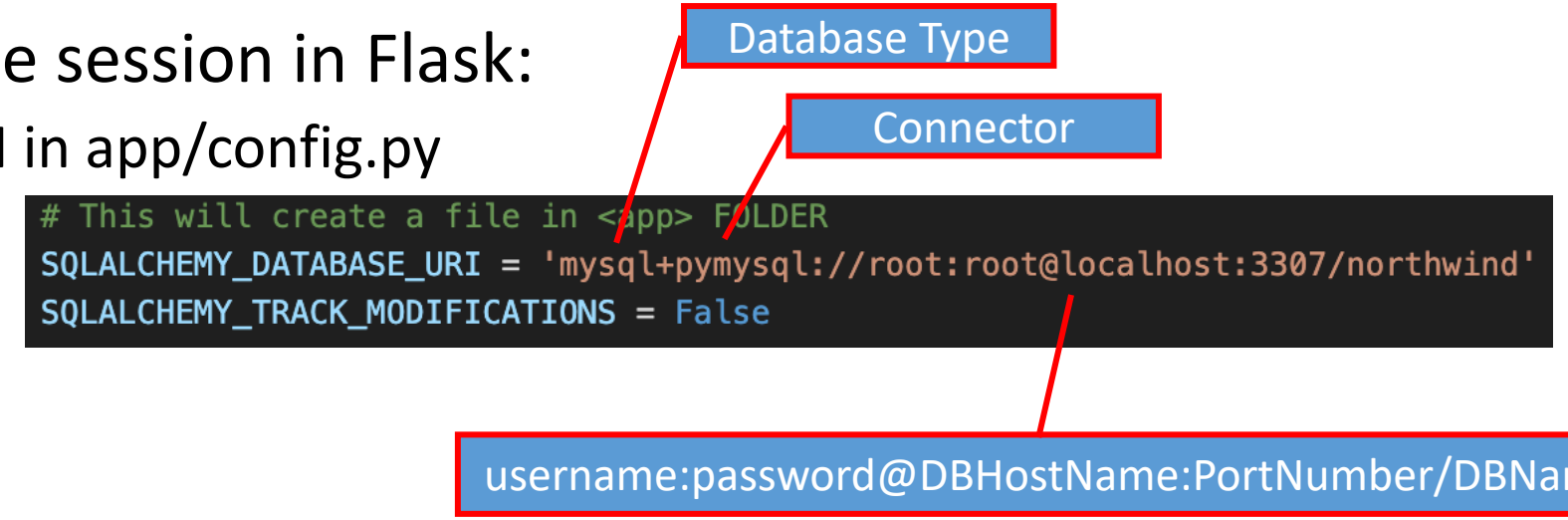
- The following Python packages are needed to represent database models in Python Flask
  - Flask_SQLAlchemy (v2.5.1)
  - SQLAlchemy (v1.4.5)
  - pyMySql (v1.0.3)
  - Flask_migrate (v2.7.0)
- ORM (Object-Relational Mapping) uses metadata descriptors to create a layer between the programming language and a relational database
- A database table is represented as a class defined in app/models.py

```
class Products(db.Model):
    ProductID = db.Column(db.Integer, primary_key=True)
    ProductName = db.Column(db.String(40), nullable=False)
    SupplierID = db.Column(db.Integer, nullable=True)
    UnitPrice = db.Column(db.Numeric, nullable=True, default=0)
```

Ex. The Products table in the northwind database

# A SQLAlchemy Database Object

- A SQLAlchemy database object must be created
- The object has a **session** property object that is a “holding zone” for all the objects which you’ve loaded or associated with it
- To establish a database session in Flask:
  - Define a database URI in app/config.py




  - Create a **db** object in app/__init__.py

```
# Construct the DB Object (SQLAlchemy interface)
db = SQLAlchemy (app)
```

# Import the ORM to Flask

- The database table models must be imported to app/views.py before accessing or modifying the table content

```
# App modules
from app import app, db
from app.models import Products
```

# Querying a Table

- Flask-SQLAlchemy makes a **query** object available in each model class (e.g., Products)
- Commonly used query methods:

| Method | Description | Example |
|---|---|---|
| all() | Returns all the results of a query as a list | Prods = Products.query.all() |
| first() | Returns the first result of a query, or None if there is no results | Prod = Products.query.first() |
| get() | Returns the row that matches the primary key, or None if no matching row is found | Prod = Products.query.get(1) |
| count() | Returns the result count of a query | numProds = Products.query.count() |
| filter_by() | Returns a new refined query result with additional query conditions | ProdByID = Products.query.filter_by(ProductID=1) |

# Querying a Table (Cont'd)

| Method | Description | Example |
|---|---|---|
| order_by() | Returns a query that sorts the result according to the given criteria | Prods = Products.query.order_by(Products.ProductName).all() |
| group_by() | Returns a query that groups the result according by the given criteria | from sqlalchemy import func numProdsPerSupplier = Products.query(SupplierID, func.count(ProductID).label("numProds")).group_by(SupplierID).having(numProds>=1).all() |
| join() | Returns a query that joins another query according to the given join criteria | ProdsSupp = db.session.query(Products.ProductName, Suppliers.CompanyName).join(Suppliers, Products.SupplierID==Suppliers.SupplierID).all() |

# Making Changes in a Table

- Inserting a row

```
#add a new supplier
new_supplier = Suppliers(CompanyName="ABC Co.")
db.session.add(new_supplier)
db.session.commit()
db.session.refresh(new_supplier)
flash("A new supplier with SupplierID " + str(new_supplier.SupplierID) + " has been added.")
```

- Modifying a row

```
#update a supplier
existing_supplier = Suppliers.query.get(1)
existing_supplier.ProductName = pname
existing_supplier.SupplierID = sid
existing_supplier.UnitPrice = pprice
existing_supplier.verified = True
db.session.commit()
flash("The supplier record with SupplierID 1 has been updated.")
```

# Making Changes in a Table

- Deleting a row

```
#delete a supplier
deleted_supplier = Suppliers.query.get(1)
db.session.delete(deleted_supplier)
db.session.commit()
```

# Set Up a Form Request in Flask

- In views.py: Set up a new route that can accept GET and POST requests

```
@app.route('/productsFormSubmit', methods=['GET', 'POST'])
def productsFormSubmit():
```

- In the html source code: Set up the form action to the new route and provide a name attribute for each input element

```
<form method="post" action="{{ url_for('productsFormSubmit') }}">
```

```
<input type="number" class="form-control" name="pid" id="inputPID" value="{{ pid }}">
```

The double curly brackets are Jinda syntax

# Set Up a Form Request in Flask (cont'd)

- Jinja syntax examples that you can use in the html template

```
{% if supplier.SupplierID == sid %}
    <option value="{{ supplier.SupplierID }}" selected>{{ supplier.CompanyName }}</option>
{% else %}
    <option value="{{ supplier.SupplierID }}">{{ supplier.CompanyName }}</option>
{% endif %}
```

```
{% for supplier in suppliers %}
        <option value="{{ supplier.SupplierID }}">{{ supplier.CompanyName }}</option>
{% endfor %}
```

```
 <p>{{ session.get('pid') }}</p>
```

```
{% with messages = get_flashed_messages() %}
    {% if messages %}
        <ul>
        {% for message in messages %}
            <li>{{ message }}</li>
        {% endfor %}
        </ul>
    {% endif %}
{% endwith %}
```

# Set Up a Form Request in Flask (cont'd)

- In views.py, you can pass those variables used in the html template in render_template

```
return render_template("products.html", suppliers=suppliers, pid=pid, pname=pname, sid=sid, pprice=pprice)
```

- In views.py, you can access the user input values provided in a form by (make sure the request package is imported from flask):

```
pid = request.form.get('pid')
```

# State Management

- What if we need to save those variable values so that they are available going from page to page?
- Solutions to the Data Persistence Problem
  - Short-term data persistence
    - **Sessions** stored on the web server (import the session package from flask)
    - Cookies stored in the client's browser
  - Long-term data persistence
    - Databases

```
#To set a session variable
session['pid'] = 1

#To get a session variable value
session.get('pid')
```

# Demo 1: Supplier Login/Register