

WMNBE-2203

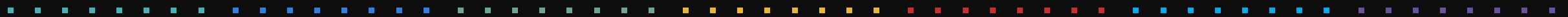
BACK-END DEVELOPMENT

Flask #8

Database Operations



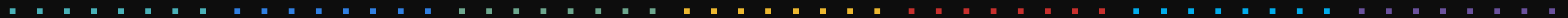
ΠΕΡΙΕΧΟΜΕΝΑ



Περιεχόμενα

- **Flask-SQLAlchemy**
 - Retrieving data
 - Manipulating data
- **Flask-Migrate**
 - Database Migrations

FLASK-SQLALCHEMY



Flask- SQLAlchemy

Retrieving data

Το **Flask-SQLAlchemy** δημιουργεί ένα *query object* σε κάθε κλάση / μοντέλο.

Μέσω αυτού μπορούν να ανακτηθούν δεδομένα από τη βάση.

```
<ModelClass>.query.<QueryFilters>.<QueryExecutor>
```

Retrieving data

Query filters

Filter	Description
filter_by	Returns a new query that adds an additional equality filter to the original query
filter	Returns a new query that adds an additional filter to the original query
limit	Returns a new query that limits the number of results of the original query to the given number
offset	Returns a new query that applies an offset into the list of results of the original query
order_by	Returns a new query that sorts the results of the original query according to the given criteria
group_by	Returns a new query that groups the results of the original query according to the given criteria

Retrieving data

Query executors

Executor	Description
all	Returns all the results of a query as a list
get	Returns the row that matches the given primary key, or None if no matching row was found
get_or_404	Returns the row that matches the given primary key or, if the key is not found, aborts the request with a 404 response
one	Returns exactly one result or raise an exception, if no or multiple results were found
one_or_none	Returns one result, or None , or raise an exception, if multiple rows were found
first	Returns the first result of a query, or None , if no rows were found
first_or_404	Returns the first result of a query, or aborts the request and sends a 404 response if there are no result

Retrieving data

Query executors (cont.)

Executor	Description
count	Returns the result count of the query
paginate	Returns a Pagination object that contains the specified range of results

Retrieving data

filter_by

Δέχεται πολλαπλά *keyword arguments* και κατασκευάζει ένα **WHERE clause** βάσει αυτών. Ελέγχει μόνο για ισότητα.

Αν υπάρχουν παραπάνω από ένα ορίσματα, αυτά ενώνονται με **AND**.

```
Student.query.filter_by(srn=srn).first_or_404()

Student.query.filter_by(firstname='John', lastname='Doe') \
    .one_or_none()
```

Retrieving data

filter

Μπορεί να κατασκευάσει πιο σύνθετα **WHERE clauses**. Δέχεται ως όρισμα, συνήθως, μια λογική συνθήκη.

```
# Note the `==` here and the `=` in `filter_by`
Student.query.filter(Student.firstname == 'Doe').all()

# Use |, & for complex queries. Parenthesis are mandatory
Student.query.filter((Student.firstname == 'Jane') & \
                     (Student.lastname == 'Doe')).one()

# Or use the `or_`, `and_` functions. Same result as above
Student.query.filter(or_(Student.firstname == 'Jane', \
                         Student.firstname == 'John')).all()

Student.query.filter(Student.email.endswith('@sae.edu')).all()
```

Retrieving data

limit & offset

Αντιστοιχούν στα **LIMIT** και **OFFSET** που συναντώνται σε αρκετές **SQL** διαλέκτους. Δέχονται από ένα αριθμητικό όρισμα και μπορούν να χρησιμοποιηθούν μαζί, αλλά και ξεχωριστά.

Δεν έχει νόημα να χρησιμοποιηθεί η **limit** μαζί με τις **first**, καθώς αυτή στην ουσία αντιστοιχεί σε **.limit(1)**.

```
page_no = ...
page_size = ...

Student.query.limit(page_size) \
    .offset((page_no - 1) * page_size).all()
```

Αν η διάλεκτος δεν υποστηρίζει **LIMIT** / **OFFSET** (π.χ. **T-SQL**), τότε κατασκευάζεται **query** που να έχει το ίδιο αποτέλεσμα.

Retrieving data

slice

Στη θέση των **limit** & **offset** μπορεί να χρησιμοποιηθεί η μέθοδος **slice**.

Δέχεται δύο αριθμητικά ορίσματα, τα οποία λειτουργούν όπως στην **range**.

```
start = (page_no - 1) * page_size
end = start + page_size

Student.query.slice(start, end).all()
```

Retrieving data

order_by

Είναι σπάνιο για ένα **query** στη βάση, να μην έχει κάποιο **ORDER BY clause**, όταν ανακτάμε περισσότερες από μία εγγραφές.

Δεν μπορούμε να βασιστούμε στη σειρά με την οποία μπήκαν οι εγγραφές στη βάση.

Δέχεται ένα ή περισσότερα ορίσματα που αντιστοιχούν σε *attributes* της κλάσης. Για φθίνουσα σειρά χρησιμοποιείται η μέθοδος **desc**.

```
Student.query.order_by(Student.lastname, Student.firstname) \
    .all()

Course.query.order_by(Course.year.desc()) \
    .limit(10) \
    .all()
```

Retrieving data

join, group_by, with_entities, count, label

Οι δυνατότητες του **(Flask-)SQLAlchemy** είναι πολύ περισσότερες από αυτές που αναφέρονται εδώ.

Στο παρακάτω παράδειγμα φαίνονται κάποιες από αυτές. Το **query** ανακτά όλους τους **Students**, μαζί με τον αριθμό των μαθημάτων που παρακολουθεί ο καθένας.

```
Student.query.join(Class) \
    .group_by(Student.id) \
    .with_entities(User, func.count(Class) \
        .label('post_count')) \
    .all()
```

Retrieving data

Lazy loading & the **N+1** problem

Ένα βασικό χαρακτηριστικό των **ORM** είναι το *lazy loading*, που αφορά στον τρόπο ανάκτησης αντικειμένων που αλληλοσχετίζονται.

Αν δεν υπήρχε η λογική του *lazy loading*, όταν ανακτούσαμε κάποιο αντικείμενο, το **ORM** θα έπρεπε να ανακτήσει και όλα τα αντικείμενα που σχετίζονται με αυτό.

Αυτό όμως δημιουργεί ένα νέο πρόβλημα. Αν δεν είμαι προσεκτικοί με τον τρόπο που ανακτάμε τα δεδομένα, τότε αναγκάζουμε το **ORM** να τα ανακτήσει **implicitly** δημιουργώντας επιπλέον (**N**) *queries*.

Retrieving data

Παράδειγμα (προς αποφυγή)

Στο παρακάτω παράδειγμα προσπαθούμε να ανακτήσουμε τα δεδομένα των **Students** μαζί με τον αριθμό των μαθημάτων τους.

Πέρα από τον λάθος τρόπο υπολογισμού του **length** / **count** (θα έπρεπε να γίνει στη βάση και όχι στον κώδικα), προκύπτει και το εξής:

Για κάθε ένα **Student** το **ORM** θα εκτελέσει ένα νέο *query*, στην προσπάθεια να ανακτήσει τα μαθήματα του. Άρα, αν ανακτήθηκαν **N Students**, θα εκτελεστούν, συνολικά, **N+1 queries**.

```
students = Student.query.all()

data = []
for s in students:
    data.append({'srn': s.srn, 'no_of_classes': len(s.classes)})
```


Retrieving data

Lazy loading & the N+1 problem (cont.)

Για να αποφύγουμε τέτοιες καταστάσεις, που προφανώς αυξάνουν δραματικά το χρόνο εκτέλεσης ενός **request**, πρέπει να φροντίζουμε να ανακτάμε τα συσχετιζόμενα δεδομένα, στο ίδιο **query**.

Ένας τρόπος για να το επιτύχουμε αυτό, είναι να δηλώσουμε στις κλάσεις / μοντέλα, άλλη "στρατηγική" ανάκτησης (*eager loading*) και όχι **lazy=True**.

Αυτός όμως μας δεσμεύει, καθώς κάθε φορά που θα ανακτάμε ένα αντικείμενο μιας κλάσης, που δεν έχει **lazy=True** στις σχέσεις της, θα φέρνουμε μαζί και όλα τα συσχετιζόμενα αντικείμενα.

Retrieving data

options, joinedload, selectinload ...

Μια καλύτερη πρακτική είναι να ορίζουμε τη "στρατηγική" αυτή ξεχωριστά σε κάθε *query*.

Αυτό μπορεί να γίνει με τη βοήθεια της μεθόδου **options** και κάποιας από τις μεθόδους που αφορά στις "στρατηγικές" ανάκτησης (αναφέρονται παρακάτω).

```
Class.query.options(joinedload(Class.course)).all()

Student.query.options(selectinload(Student.classes)).all()
```

Αν θέλουμε το **ORM** να μας ειδοποιεί όταν δεδομένα ανακτώνται *implicitly*, μπορούμε να ορίσουμε στις σχέσεις **lazy='raise'**.

Retrieving data

Relationship Loading Techniques

Στρατηγική Περιγραφή

lazy loading	available via lazy='select' or the lazyload() option, this is the form of loading that emits a SELECT statement at attribute access time to lazily load a related reference on a single object at a time.
joined loading	available via lazy='joined' or the joinedload() option, this form of loading applies a JOIN to the given SELECT statement so that related rows are loaded in the same result set.
subquery loading	available via lazy='subquery' or the subqueryload() option, this form of loading emits a second SELECT statement which re-states the original query embedded inside of a subquery, then JOINS that subquery to the related table to be loaded to load all members of related collections / scalar references at once.
select IN loading	available via lazy='selectin' or the selectinload() option, this form of loading emits a second (or more) SELECT statement which assembles the primary key identifiers of the parent objects into an IN clause, so that all members of related collections / scalar references are loaded at once by primary key.

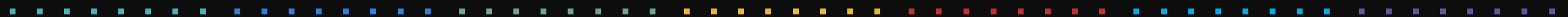
Retrieving data

Relationship Loading Techniques (cont.)

Τεχνική Περιγραφή

raise loading	available via lazy='raise' , lazy='raise_on_sql' , or the raiseload() option, this form of loading is triggered at the same time a lazy load would normally occur, except it raises an ORM exception in order to guard against the application making unwanted lazy loads.
no loading	available via lazy='noload' , or the noload() option, this loading style turns the attribute into an empty attribute (None or []) that will never load or have any loading effect.

MANIPULATING DATA



Manipulating data

Adding data

Η αποθήκευση νέας εγγραφής στη βάση είναι αρκετά απλή υπόθεση.

Δημιουργούμε ένα *object* της κλάσης που θέλουμε και το προσθέτουμε στο **db.session**, το οποίο είναι αυτόματα διαθέσιμο μέσω του **Flask-SQLAlchemy**.

Το **db.Model** από το οποίο κληρονομούν όλες οι κλάσεις, έχει φροντίσει να υπάρχει ένας *constructor* για αυτή τη δουλειά.

Manipulating data

Adding data

Για να ολοκληρωθεί το *transaction* στη βάση, πρέπει να γίνει κλήση και στη μέθοδο **db.session.commit()**.

On commit θα αποθηκευτούν όλα τα νέα αντικείμενα που έχουμε προσθέσει *explicitly* στο **db.session**, αλλά και *implicitly* όλα τα αντικείμενα που μπορεί να αναφέρονται σε αυτά.

Εκτός από τη **commit**, μπορούμε να καλέσουμε και τη μέθοδο **rollback** (π.χ. σε περίπτωση σφάλματος).

Manipulating data

Παράδειγμα

```
web = Course(title='Web', year='2021')
flask = Class(title='Flask', course=web)

s1 = Student(srn=1234, firstname='Jane', lastname='Doe')
s2 = Student(srn=2345, firstname='John', lastname='Doe')

flask.students.append(s1)
flask.students.append(s2)

db.session.add(flask)

try:
    db.session.commit()
except Error:
    db.session.rollback()
```


Manipulating data

Updating data

Ομοίως, η ενημέρωση μιας εγγραφής είναι, συνήθως, αρκετά απλή.

Όταν αλλάξουμε την τιμή μιας ιδιότητας σε ένα αντικείμενο, που έχει ανακτηθεί από τη βάση, τότε η ιδιότητα αυτή σημειώνεται ως *modified*.

Στο επόμενο **commit** το **SQLAlchemy** θα φροντίσει να παράξει το αντίστοιχο **SQL UPDATE statement**.

Για πιο σύνθετες περιπτώσεις μπορεί να χρησιμοποιηθεί το **Query API** του **SQLAlchemy**.

Manipulating data

Παράδειγμα

```
student = Student.query.filter_by(srn=1234)
# SELECT ... FROM student WHERE student.srn = ?
# (1234,)

student.fullname = 'Joan Doe'
db.session.commit()
# UPDATE student SET student.fullname = ? WHERE student.id = ?
# (Joan Doe, 1)

num_rows_updated = Class.query.filter(year='2020') \
    .update({Class.year: '2021'})
# UPDATE class SET class.year = ? WHERE class.year = ?
# (2021, 2020)

db.session.commit()
```

Manipulating data

Deleting data

Τέλος, η διαγραφή εγγραφών, είναι, ίσως, η πιο απλή ενέργεια.

Αρκεί να καλέσουμε τη μέθοδο **db.session.delete** με όρισμα το αντικείμενο που θέλουμε να διαγράψουμε.

Όπως και πριν, πρέπει στη συνέχεια να γίνει κλήση στη μέθοδο **db.session.commit**.

Και εδώ, για πιο σύνθετες περιπτώσεις, μπορεί να χρησιμοποιηθεί το **Query API** του **SQLAlchemy**.

Manipulating data

Παράδειγμα

```
wrong_entry = Student.query.filter(Student.srn == 0) \
    .one_or_none()

if wrong_entry is not None:
    db.session.delete(wrong_entry)
    db.session.commit()
    # DELETE FROM student WHERE student.id = ?
    # (1,)

Student.query.filter(Student.srn < 0).delete()
db.session.commit()
# DELETE FROM student WHERE student.srn < ?
# (0,)
```

FLASK-MIGRATE

Flask-Migrate

Database migrations

Μαζί με την ανάπτυξη του κώδικα μιας εφαρμογής, αναπτύσσεται και η βάση στην οποία βασίζεται.

Είναι σημαντικό, εκτός από το *versioning control* του κώδικα, να υπάρχει και ένας τρόπος διαχείρισης των αλλαγών της βάσης, κρατώντας στιγμιότυπα κ.λπ.

Για αυτή τη δουλειά υπάρχει το εργαλείο **Alembic** που έχει αναπτυχθεί από το δημιουργό του **SQLAlchemy**. Αντ' αυτού, σε μια εφαρμογή **Flask**, μπορεί να χρησιμοποιηθεί το **Flask-Migrate** που παρέχει κάποιες επιπλέον λειτουργίες.

Flask-Migrate

flask db

Το **Flask-Migrate** προσθέτει ένα *CLI command*, το **flask db**.

Όλες οι απαραίτητες λειτουργίες, βρίσκονται σε αυτό.

Για να ξεκινήσουμε με τη χρήση του, αρκεί να δηλωθεί το εξής:

```
from flask_migrate import Migrate
...

migrate = Migrate(app, db)

# or
# migrate = Migrate()
# migrate.init_app(app, db)
```

Flask-Migrate

Initial steps

1. **flask db init** - Αρχικοποίηση της διαδικασίας. Δημιουργείται ο φάκελος **migrations** όπου θα αποθηκευτούν όλα τα απαραίτητα *script*.
2. **flask db migrate -m 'Initial migration'** - Δημιουργία του πρώτου *migration script*. Το *script* αυτό περιέχει όλες τις **Alembic** εντολές ώστε να φτιαχτεί μία βάση που να αντιπροσωπεύει το μοντέλο του **SQLAlchemy**.
3. **flask db upgrade** - Αναβάθμιση της βάσης στην επόμενη έκδοση. Αν η βάση δεν υπάρχει ακόμα, θα δημιουργηθεί, μαζί με έναν επιπλέον πίνακα, τον **alembic_version**, που υποδεικνύει την τρέχουσα έκδοση.

Flask-Migrate

Schema changes

Ο τρόπος που διαχειριζόμαστε τις αλλαγές στη βάση είναι ο εξής:

1. Αλλαγές στις κλάσεις / μοντέλα του **SQLAlchemy**.
2. Εκτέλεση του **flask db migrate -m 'message'** για τη δημιουργία του κατάλληλου *script*.
3. Εκτέλεση του **flask db upgrade** για την εφαρμογή των αλλαγών στη βάση.

Αν, για κάποιο λόγο πρέπει να γυρίσουμε σε παλαιότερη "έκδοση" της βάσης, υπάρχει διαθέσιμη η εντολή **flask db downgrade**.

Χρήσιμα links

SQLAlchemy in Flask — Flask Documentation (2.1.x)
<https://flask.palletsprojects.com/en/2.1.x/patterns/sql...>

Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (2.x)
<https://flask-sqlalchemy.palletsprojects.com/en/2.x/>

SQLAlchemy ORM — SQLAlchemy 1.4 Documentation
<https://docs.sqlalchemy.org/en/14/orm/index.html>

Flask-Migrate — Flask-Migrate documentation
<https://flask-migrate.readthedocs.io/en/latest/>

Data Management With Python, SQLite, and SQLAlchemy – Real Python
<https://realpython.com/python-sqlite-sqlalchemy/#cr...>

Extra info


📄 Application Factories — Flask Documentation (2.1.x)
<https://flask.palletsprojects.com/en/2.1.x/patterns/ap...>

📄 Application Setup — Flask Documentation (2.1.x)
<https://flask.palletsprojects.com/en/2.1.x/tutorial/fact...>

📄 python - How to update SQLAlchemy row entry? - Stack Overflow
<https://stackoverflow.com/questions/9667138/how-to...>

📄 python - SQLAlchemy - Is there a way to see what is currently in the session? - Stack Overflow
<https://stackoverflow.com/questions/31928520/sqlalc...>

📄 SQLAlchemy: get object instance state - Stack Overflow
<https://stackoverflow.com/questions/3885601/sqlalch...>

 Query Planning
<https://sqlite.org/queryplanner.html>

🌐 Squeezing Performance from SQLite: Indexes? Indexes! | by Jason Feinstein | Medium
<https://medium.com/@JasonWyatt/squeezing-perfor...>

THANK YOU!