New technology stack in Invenio v2.0

SQLAlchemy, Flask and Jinja2

Jiří Kunčar

jiri.kuncar@cern.ch

CERN

www.invenio-software.org

May 9th, 2012

Outline

SQLAlchemy

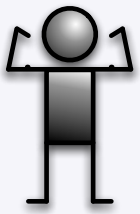
Flask

web development,
one drop at a time

Jinja

Database layer

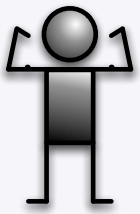
SQLAlchemy



- Library providing a Python-ic interface to relational databases
- Includes powerful SQL expression language
- "Simple things simple, and complex things possible."

Database layer

SQLAlchemy

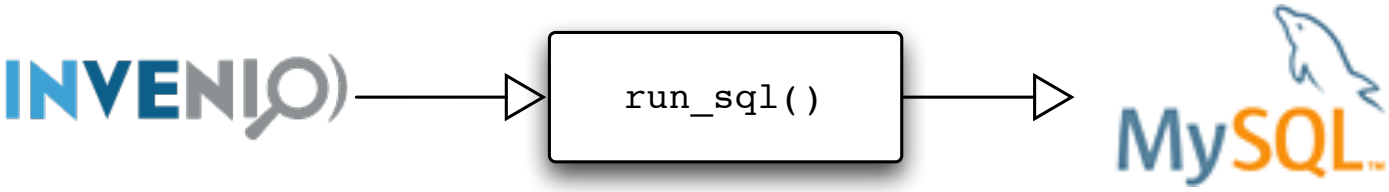


- Library providing a Python-ic interface to relational databases
- Includes powerful SQL expression language
- "Simple things simple, and complex things possible."

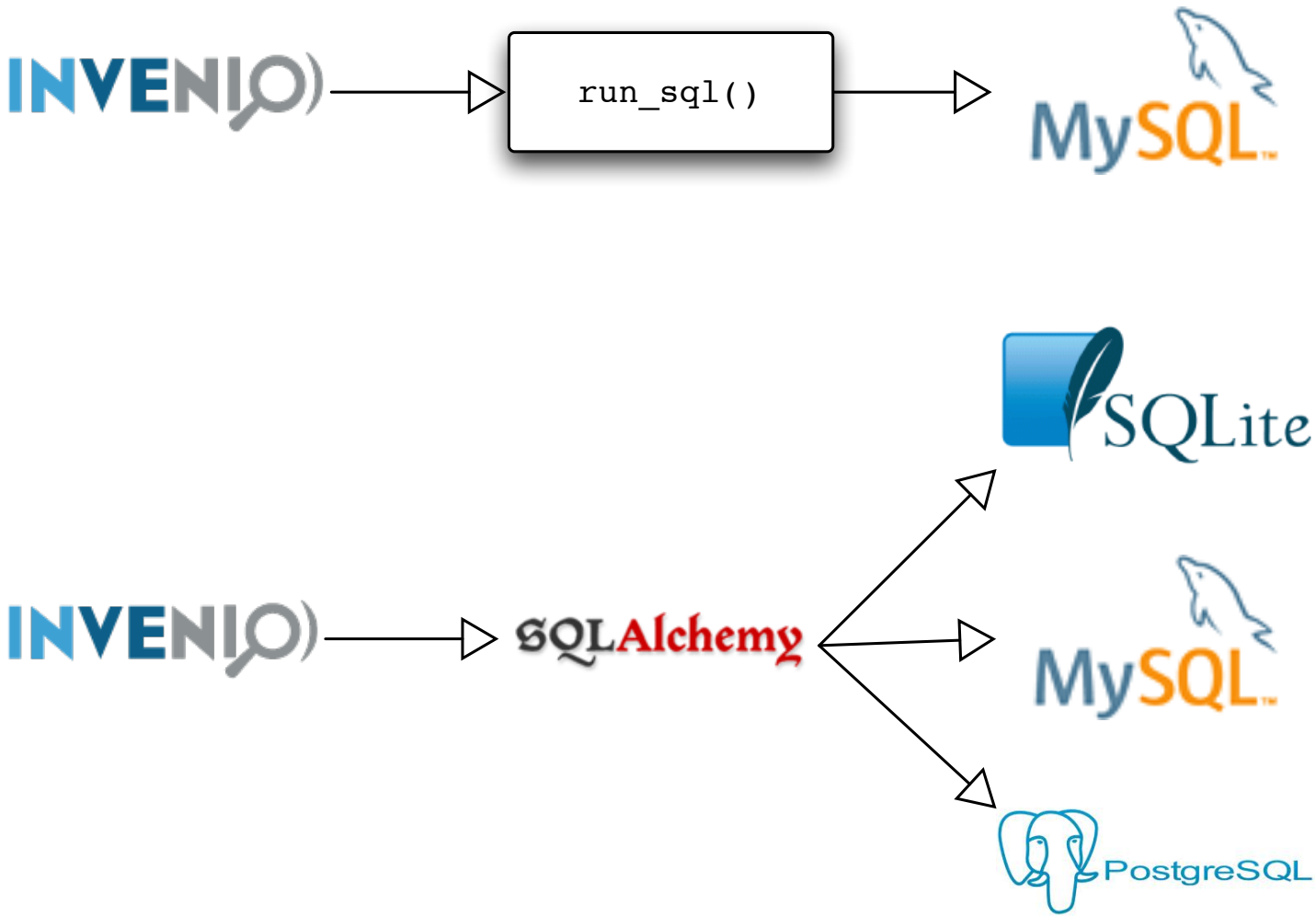
How is it going to be useful in Invenio?

- Database independence
- Easier query building
- Advanced features: Horizontal Sharding, Versioned Objects

Database Independence



Database Independence



Object Relation Mapping (ORM)

Why SQLAlchemy?

- Support for wide variety of database and architectural designs by flexible model definitions
- Mature, High Performing Architecture
- Composite PK and FK represented as sets of columns
- Simple definition of relations
- `db.metadata.create_all()`

SQLAlchemy


sourceforge
mozilla
Firefox
Seagate
lolapps
fedora
openstack
CLOUD SOFTWARE

Flask

Description

- Lightweight
- Build on top of `Werkzeug`
- Wide range of extensions: `assets`, `cache`, `testing`, `sqlalchemy`
- `Blueprints`: pluggable architecture
- Powerful URL routing system: multiple rules per function, remapping

Jinja2

Description

- Unicode support
- Template inheritance
- Sandboxed execution mode, (auto)escaping
- Loop controls: `index`, `first`, `last`, `odd`
- Template designer helpers: `batch`, `split`, `filter`
- `{% css 'css/invenio.css' %}`
- `{% js 'css/jquery.js' %}`

Outline

Invenio Model Features I

Declarative model

- Table name
- Basic types (`Integer`, `String`, `DateTime`, `Binary`)
- Database specific options (`Enum`, `Autoincrement`)
- Definitions in db object

```
class MsgMESSAGE(db.Model):
    __tablename__ = 'msgMESSAGE'
    id = db.Column(db.Integer(15, unsigned=True), nullable=False,
                   primary_key=True, autoincrement=True)
    subject = db.Column(db.Text, nullable=False)
    body = db.Column(db.Text, nullable=True)
    sent_date = db.Column(db.DateTime, nullable=False,
                   server_default='0001-01-01 00:00:00')
    received_date = db.Column(db.DateTime, server_default='0001-01-01 00:00:00')
```

Invenio Model Features II

Relations

- Foreign key definition
- Relation with back-reference
- Simplified One-to-One, One-to-Many, and Many-to-Many relations (`AssociationProxy`)

```
id_user_from = db.Column(db.Integer(15, unsigned=True),
            db.ForeignKey(User.id), nullable=True, server_default='0')
user_from = db.relationship(User, backref='sent_messages')
recipients = association_proxy('sent_to_users', 'user_to',
          creator=lambda u:UserMsgMESSAGE(user_to=u))
```

Invenio Model Features III

Properties, Validations

- Column aliases – `@property`, `@db.hybrid_property`
- Validators – `@db.validates`

```
_sent_to_user_nicks = db.Column(db.Text, name='sent_to_user_nicks', nullable=False)
@db.hybrid_property # @property
def user_nicks(self):
    if not self._sent_to_user_nicks:
        return []
    return filter(len, map(strip,
        self._sent_to_user_nicks.split(CFG_WEBMESSAGE_SEPARATOR)))
@db.validates('_sent_to_user_nicks')
def validate_sent_to_user_nicks(self, key, value):
    user_nicks = filter(len, map(strip, value.split(CFG_WEBMESSAGE_SEPARATOR)))
    assert len(user_nicks) == len(set(user_nicks))
    assert len(user_nicks) == User.query.filter(User.nickname.in_(user_nicks)).count()
    return CFG_WEBMESSAGE_SEPARATOR.join(user_nicks)
```

Invenio Model Features IV

```
@sent_to_user_nicks.setter
def sent_to_user_nicks(self, value):
    old_user_nicks = self.user_nicks
    self._sent_to_user_nicks = value
    to_add = set(self.user_nicks)-set(old_user_nicks)
    to_del = set(old_user_nicks)-set(self.user_nicks)
    to_del = to_del-set([u.nickname for u in User.query.\
                join(User.usergroups).filter(
                Usergroup.name.in_(self.group_names)).\
                all()])
    if len(to_del):
        is_to_del = lambda u: u.nickname in to_del
        remove_old = filter(is_to_del, self.recipients)
        for u in remove_old:
            self.recipients.remove(u)
    if len(to_add):
        for u in User.query.filter(User.nickname.\
           in_(to_add)).all():
            if u not in self.recipients:
                self.recipients.append(u)
```

Database Independence Testing

- Using alternative databases alongside production database.
- `MySQL`, `PostgreSQL` and `SQLite`
- `Fixtures`: specific data for each test case, possible universal replacement for `tablefill.sql`.

```
from invenio.websession_model import User
class UserData(DataSet):
    class admin:
        nickname = 'admin'
        email=nickname+"@example.com"
    class romeo(admin):
        nickname = 'romeo'
    class juliet(admin):
        nickname = 'juliet'
```

```
fixture = FlaskSQLAlchemyFixture(
        env={'UserData': User},
        engine=db.metadata.bind,
        session=db.session
        )

@fixture.with_data(UserData)
def test_index(data, self):
     # test code
```

```
> sudo -u www-data /opt/invenio/bin/inveniocfg --run-flask-tests
```

Performance

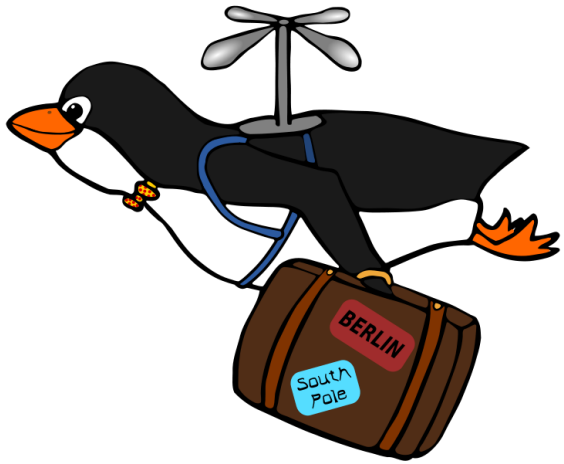
Common problems

- Too many queries.
- Gathering too much data.
- ORM is sometimes simply too slow.

There are solutions how to speed it up:

`http://invenio-software.org/wiki/Tools/SQLAlchemy/Performance`

Migration



Common use-cases

- Schema upgrade (downgrade)
- Data transfer between instances (`dev`, `prod`)
- Data format: `Fixtures` python classes, `ini` text format
- `sqlalchemy-migrate`, `Alembic`, or own solution

Filter and Sort Decorators

Features

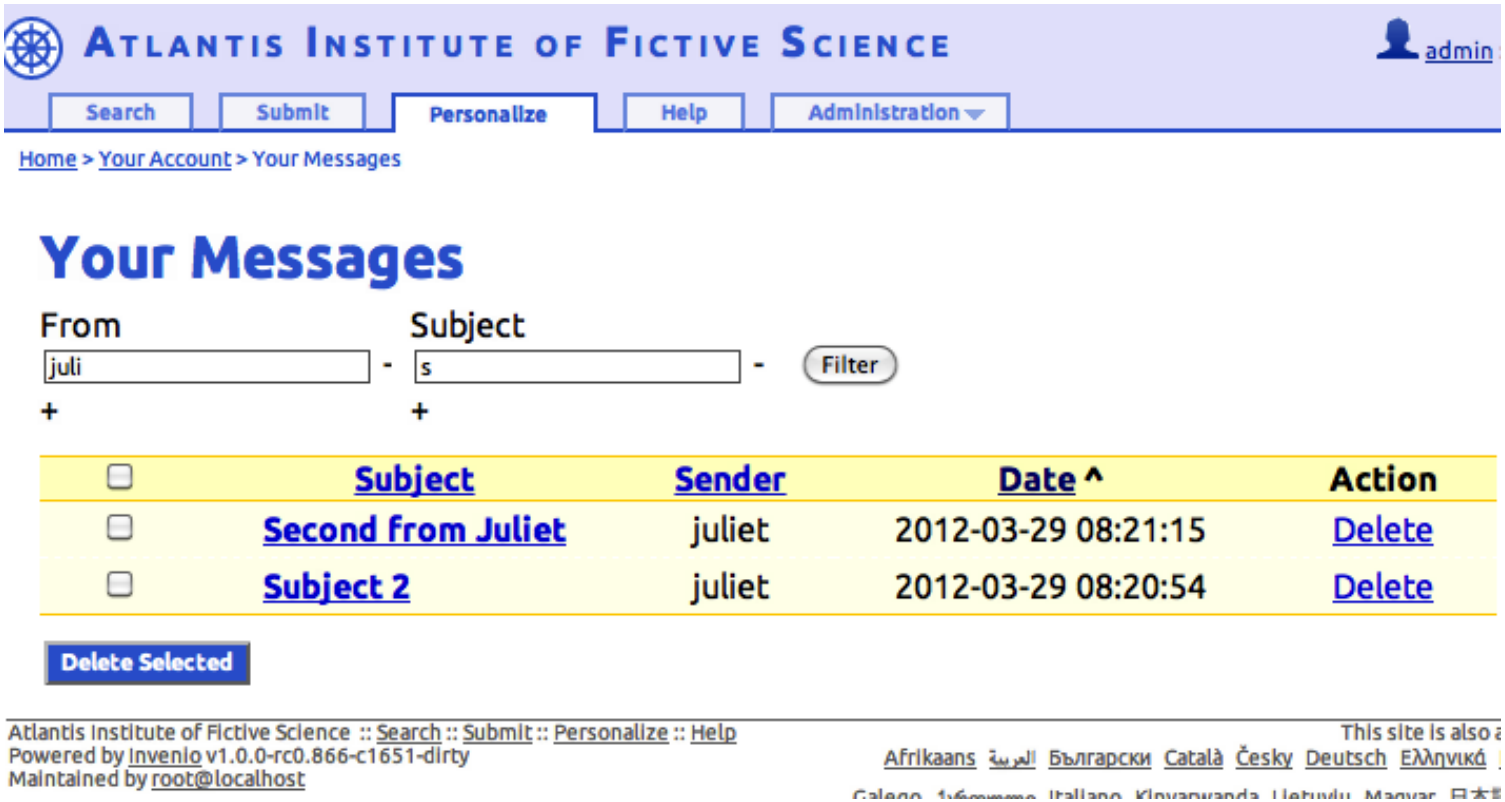
- generate **WHERE** and **ORDER BY** statements
- specify different compare functions: equal, startswith, like %...%
- keep filter while sorting and vice versa

Decorators

```
@blueprint.invenio_sorted(MsgMESSAGE)
@blueprint.invenio_filtered(MsgMESSAGE, columns={
                    'subject':operators.startswith_op,
                    'user_from.nickname':operators.contains_op},
                    form=FilterMsgMESSAGEForm)
def index(sort=None, filter=None):
    # ...
```

Web Messages

http://localhost:8080/yourmessages/display?sort_by=sent_date&order=desc&user_from.nickname=juli&subject=s



REST API

Templated

- Automatically calls `render_template()` with appropriate template
- Guess name of template if it is not defined
- On XHR returns JSON

Decorator

```
@blueprint.route('/index')
@blueprint.invenio_templated()
def index():
    return dict(users=User.query.all())
```

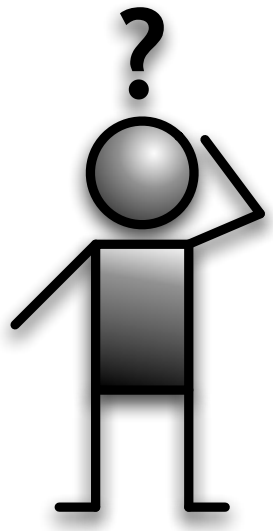
Outline

Summary

- Created all models for tables in master branch.
- Implemented WebMessage module as a proof of concept showing some of the advanced SQLAlchemy features.
- Prepared for the future generalized database table creater and updater tool.
- Tested DB independence with MySQL, SQLite, PostgreSQL for WebMessage module.
- Enriched Flask framework with SQLAlchemy friendly decorators for sorting and filtering.

Share your ideas

http://invenio-software.org/wiki/Talk



Share your ideas

http://invenio-software.org/wiki/Talk