Managing Your RDBMS Lifecycle with Python

March 14, 2019

TORCH

What we’ll cover

- Defining your DB schema in Python with `SQLAlchemy`
- Quickly adding versioning to your database with `SQLAlchemy-Continuum`
- Managing database changes with `Alembic`

Assumptions

- This is a beginner-friendly talk, but I’m assuming you have:
  - Basic knowledge of relational databases
  - Basic knowledge of Python
  - An idea of what an ORM (Object Relational Mapper) is

A thought on RDBMS

- Relational DBs have seen a slight downturn in usage as NoSQL as gained prominence

- One reason for the popularity NoSQL DBs is that they are less overhead to manage, as Schema Migrations (or any schema at all) are not required.

- While document based record storage is fantastic for many use cases, there are still several cases where transactional, integrity constrained data storage is optimal (Transactional systems, Highly Structured Reporting Systems)

What is schema migration?

- Schema Migration is the act mutating the structure of a database over time
  - adding/dropping tables
  - adding/dropping fields
  - Changing data types
  - manipulating indices, data integrity constraints etc

SQLAlchemy

- Before we can get into schema migrations, we'll need an initial schema - our initial Data Model.

- `SQLAlchemy` is good at *a lot* of things, in this case we'll be using it's ORM (Object Relational Mapper) feature to define our DB schema from plain old Python classes.

Exercise #1 - ORM

- `git checkout exercise-1`
- In our editor, we will create a couple of simple models and relationships, and observe how these models can be used to generate our DB schema in postgreSQL using `create_all()` (see it in models.py)
- Similarly, we can drop the schema using `drop_all()`
- Insert a few records into the DB using the `SQLAlchemy session` (see this happening in run.py)

ORM Tips

- You can use your models with any relational database that has DBAPI compatible driver for Python (pretty much everything)
- Let's you avoid using proprietary or non-standard SQL Dialects
- You can still use proprietary function of specific DBs when you need them
- Even though it adds another layer to your application, ORMs do provide opportunities for memoization and other performance tuning techniques (would be a whole other presentation)
- Not working on a new DB? `sqlacodegen` can be used to generate model classes from existing DB schemas.

Continuum

- Capturing and storing changes to the data itself in your DB is very useful for audit/compliance, debugging and general peace of mind

- It may not be as important for reporting DBs, where data get refreshed often and in bulk

- It does add some performance overhead

- `SQLAlchemy-continuum` is an easy add on to `SQLAlchemy` and `Alembic` to create a complete solution for a well managed database

Exercise #2 - Audit Tables

- `git checkout exercise-1`
- We'll go back to our original models, and apply the `__versioned__` attribute to our model.
- We'll use `make_versioned()` and `configure_mappers()` to generate the models and associated versioning schema from scratch.
- We'll perform a few CRUD ops with our models…then view the history

Continuum Tips

- Access previous versions of a models using the `versions[]` attribute
- For many reasons (performance, data security), it may be good to come up with a data retention policy, and create an automated process to prune your version tables
- *Caveat:* as the schema evolves, migrations will need to be used to manage the associated versioning tables as well.
- I was having trouble getting continuum to work with SQLA 1.3+ (reported), so I am pinned at 1.2.x

Now What?

- Now that you have a schema (including the audit tables), you’re going to want to change it and add to it over time.

- But, `create_all` doesn’t diff your schema, it just generates and executes DDL for whatever is defined in your model classes at build time (the whole thing), and only if you have an empty DB to start with.

- It doesn’t make sense to drop and rebuild your database every time you need to change the schema. Even if you cooked up a way to do that, it would certainly mean downtime ☹️

Alembic

- Alembic is a tool for versioning changes to your database schema. Initialize it by running `alembic init alembic`

- ‘Versions’ are a linked-list of python scripts with distinct IDs, each having an `upgrade()` and `downgrade()` function.

- The `autogenerate` command can be used to generate migration scripts from changes detected in your models.

- A table called `alembic_version` is created in the DB to track which version your database conforms to.

Exercise #3 - Migrations

- Drop your DB & `git checkout exercise-3`
- Generate initial models: `alembic revision --autogenerate -m 'initial DB’`
- Upgrade - `alembic upgrade head`
- Make a model change, generate another migration, and upgrade again
  - `alembic revision --autogenerate -m ‘added col…’`
- Try a downgrade - `alembic downgrade -1`

Now What?

- Use unit tests to validate that your migrations upgrade and downgrade smoothly.

- Make schema migration part of your release and rollback processes, executed via your CI/CD pipeline.

Things to watch out for

- Stick to schema, leave data transformation out (unless it's needed for schema change)
- Running inside a VPC vs. from outside a VPC (for AWS users)
- Testing - unit test your migrations, but….
    - Don't re-migrate your DB from scratch for ***every*** test, use `create_all()`
    - Consider swapping in SQL lite
- Conflicts (two HEADs) happen now and again
- Slow deploys (due to long running ops, such as indexing - deploy alone)

Wrap Up

- The evolution of your relational DB schema over time should be managed programmatically to ensure consistency, traceability and ease of replication
- This used to be a lot of work, but thanks to these great OSS tools, all it takes is a little discipline and some RTFM to make this job a minor footnote as you bootstrap your next project.

Thanks!

- The Torch Engineering team, for figuring all this out.
- Rick Galbo for the docker help.
- Our sponsors:
  - Delaware North - for the awesome space and refreshments.
- You!

Learn More

- SQLAlchemy - https://www.sqlalchemy.org/
- Alembic - https://alembic.sqlalchemy.org
- Continuum - https://sqlalchemy-continuum.readthedocs.io
- Engineering @Torch - https://torch.io/careers/engineering