SQLAlchemy Python SQL toolkit

Engine URL + pool → *connect* → **Connection** Core, raw SQL → *wraps* → **Session** ORM unit of work → *tracks* → **Mapped objects** identity map

CONCEPTS

Core is SQL as Python expressions (Engine, Connection, Table). The ORM adds mapped classes and the Session on top.

Nothing is saved until commit(). Leaving the with-block without commit rolls back.

ENGINE

```
create_engine("sqlite://
/app.db")
```
SQLite file engine

```
create_engine(
"postgresql+psycopg://
<user>:<pw>@<host>/
<db>")
```
PostgreSQL engine

```
create_engine("sqlite://
", echo=True)
```
In-memory DB, log all SQL

```
with engine.connect() as
conn:
```
Core connection block

```
conn.execute(text("... :id"),
{"id": 5})
```
Bind parameters by name

DECLARE MODELS

```
class
Base(DeclarativeBase):
pass
```
Declarative base class

```
__tablename__ = "user"
```
Table name for a model

```
id: Mapped[int] =
mapped_column(
primary_key=True)
```
Primary key column

```
name: Mapped[str] =
mapped_column(String(
50))
```
Typed string column

```
email:
Mapped[Optional[str]]
```
Nullable column

```
mapped_column(unique=
True, index=True)
```
Unique constraint / index

```
mapped_column(
ForeignKey("user.id"))
```
Foreign key column

```
Base.metadata.
create_all(engine)
```
Create missing tables

SESSION

```
with Session(engine) as
session:
```
Open a session

SESSION (cont.)

```
session.add(<obj>)
```
Add one object

```
session.commit()
```
Write changes and end transaction

```
session.rollback()
```
Discard pending changes

```
session.get(User, <pk>)
```
Fetch by primary key

```
sessionmaker(engine)
```
Session factory for an app

QUERY

```
session.scalars(select(
User)).all()
```
All rows as objects

```
select(User).where(User.
name == "ann")
```
Filter rows

```
session.scalars(stmt).
first()
```
First object or None

```
session.execute(select(
User.id, User.name))
```
Rows of column tuples

```
select(User).order_by(
User.name.desc())
```
Sort rows

```
.where(or_(<a>, <b>))
```
OR conditions

QUERY (cont.)

```
select(Order.user_id,
func.sum(Order.total)).
group_by(Order.user_id)
```
Aggregate per group

INSERT, UPDATE, DELETE

```
user.name = "bob"
```
Update via attribute, then commit

```
update(User).where(
<cond>).values(name="x")
```
Bulk UPDATE

RELATIONSHIPS

```
posts:
Mapped[list["Post"]] =
relationship(
back_populates="author")
```
One-to-many side

```
author: Mapped["User"] =
relationship(
back_populates="posts")
```
Many-to-one side

```
relationship(cascade=
"all, delete-orphan")
```
Delete children with parent

```
select(User).options(
selectinload(User.
posts))
```
Eager-load collection

RELATIONSHIPS (cont.)

```
select(Post).options(
joinedload(Post.author))
```
Eager-load by JOIN

JOINS

```
select(User,
Post).join(Post)
```
Inner join on the foreign key

```
select(User).join(User.
posts)
```
Join along a relationship

ALEMBIC MIGRATIONS

```
alembic init <dir>
```
Create migration environment

```
target_metadata =
Base.metadata
```
Set in env.py for autogenerate

```
alembic revision --
autogenerate -m "<msg>"
```
Diff models against DB

```
alembic upgrade head
```
Apply all migrations

```
alembic downgrade -1
```
Undo the last migration