# SQLAlchemy Python SQL toolkit

## CONCEPTS

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

## ENGINE

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

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

```
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

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

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

## SESSION (cont.)

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

## 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

## INSERT, UPDATE, DELETE

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

## RELATIONSHIPS

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

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

## 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