SQLAlchemy 2.0 Quick Reference

A companion to the SQLAlchemy for Python module in DataTweets' SQL & Databases course. Every pattern here is verified against a real SQLite database.

---

1. Connect and reflect

```
from sqlalchemy import create_engine, MetaData, Table, inspect

engine = create_engine("sqlite:///company.db")
inspector = inspect(engine)
inspector.get_table_names()

metadata = MetaData()
employees = Table("employees", metadata, autoload_with=engine)
```

2. Define and create tables

```
from sqlalchemy import Column, Integer, String, ForeignKey

departments = Table("departments", metadata,
Column("id", Integer, primary_key=True),
Column("name", String(50), nullable=False, unique=True))

employees = Table("employees", metadata,
Column("id", Integer, primary_key=True),
Column("department_id", Integer, ForeignKey("departments.id")))

metadata.create_all(engine)
```

3. Insert, select, filter

```
from sqlalchemy import insert, select, and_

with engine.connect() as conn:
conn.execute(insert(employees), [{"id": 1, "name": "Alice"}])
conn.commit()

stmt = select(employees).where(
and_(employees.c.salary.between(30000, 60000),
employees.c.department_id == 2))
rows = conn.execute(stmt).fetchall()
```

4. Update, delete, transactions

```
from sqlalchemy import update, delete

with engine.connect() as conn:
with conn.begin(): # commits, or rolls back on error
conn.execute(update(employees)
.where(employees.c.id == 1).values(salary=65000))
conn.execute(delete(employees).where(employees.c.id == 99))
```

5. Joins, aggregates, self-joins

```
from sqlalchemy import func
```

```
stmt = (
select(departments.c.name, func.avg(employees.c.salary))
.select_from(employees.join(departments))
.group_by(departments.c.name)
)

managers = employees.alias() # self-join
stmt = select(employees.c.name, managers.c.name).select_from(
employees.join(managers, employees.c.mgr_id == managers.c.id))
```

6. The ORM: models and sessions

```
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column, relationship, Session

class Base(DeclarativeBase): pass

class Department(Base):
__tablename__ = "departments"
id: Mapped[int] = mapped_column(primary_key=True)
name: Mapped[str]
employees: Mapped[list["Employee"]] = relationship(back_populates="department")

class Employee(Base):
__tablename__ = "employees"
id: Mapped[int] = mapped_column(primary_key=True)
department_id: Mapped[int] = mapped_column(ForeignKey("departments.id"))
department: Mapped["Department"] = relationship(back_populates="employees")

with Session(engine) as session:
rows = session.scalars(select(Employee).where(Employee.salary > 30000))
```

7. Pagination, bulk ops, performance

```
from sqlalchemy import Index

stmt = select(employees).order_by(employees.c.id).limit(10).offset(20)

Index("ix_employees_department_id", employees.c.department_id).create(engine)

with engine.connect() as conn:
conn.execute(insert(employees), list_of_row_dicts) # one round trip
result = conn.execute(select(employees)).yield_per(500) # stream in batches
```

Full lessons, runnable examples, and a guided project analyzing real U.S. Census data with SQLAlchemy and pandas: datatweets.com/courses/sql/sqlalchemy-for-python/