NoSQL & Python ORM 기초

> - *NoSQL 구조 이해*
> - *Python을 통한 데이터 연동 학습*
> - *ORM을 활용한 **의료 데이터(환자·진료·검사)** 처리 실습*

GOAL

NoSQL 구조 이해, Python 연동 학습, 의료 데이터 처리 실습

- NoSQL 소개
  - 문서형(MongoDB), 키-값형, 그래프형 DB
  - 장단점 및 활용 사례
- Python과 DB 연동
  - Tortoise ORM (비동기 ORM)
- 실습
  - Python으로 CRUD(삽입·조회·수정·삭제)
  - ORM 모델링: 환자, 진료, 검사 테이블 구성
  - 간단한 쿼리 및 필터링

NoSQL이란?

> *“Not Only SQL” —*
> *SQL만으로는 표현하기 어려운 유연한 데이터를 다루기 위한 새로운 방식*

특징	설명
스키마 없음	구조가 고정되지 않아 필드를 자유롭게 추가 가능
확장성 높음	서버 여러 대로 쉽게 확장 (수평 확장)
유연한 저장	JSON, Key-Value, 그래프 등 다양한 구조 지원
빠른 속도	관계 계산보다 읽기/쓰기 중심의 속도에 강점

> *RDB는 “정리된 서고가 있는 도서관 ”*
> *NoSQL은 “라벨만 맞으면 뭐든 넣을 수 있는 창고 ”*

NoSQL의 대표 유형

유형	특징	예시 DB
문서형(Document)	JSON 문서 구조로 저장	MongoDB
키-값형(Key-Value)	단순한 Key와 Value 저장	Redis, DynamoDB
열 지향형(Column Family)	열 단위로 데이터 저장	Cassandra
그래프형(Graph)	관계를 노드/간선으로 표현	Neo4j

MongoDB란?

- 문서(Document) 단위로 데이터를 저장하는 비관계형 데이터베이스
- JSON 구조로 데이터를 저장 (BSON 형식)
- 하나의 문서(document)가 곧 “하나의 행(row)” 역할
- 스키마를 자유롭게 변경 가능 (필드 추가, 제거 자유)

컬렉션(Collection)

컬렉션 이름	저장 예시
`patients`	환자 정보 (기본 인적사항, 질병 목록 등)
`visits`	진료 기록 (의사, 날짜, 진단)
`tests`	검사 데이터 (혈압, 혈당 등)

> *문서마다 필드 구성이 달라도 허용*
> *SQL의 “테이블” 대신 **컬렉션(Collection)**,*
> *“행(Row)” 대신 **문서(Document)** 개념을 사용*

환자(patients) 컬렉션

```
{
  "_id": "671f84a82a01",
  "name": "아동원",
  "birth_date": "1993-01-01",
  "gender": "M",
  "phone": "010-1234-5678",
  "diseases": ["고혈압", "당뇨"],
  "created_at": "2025-10-24T09:00:00"
}
```

의사(doctors) 컬렉션

```
{
  "_id": "671f84b42e1a",
  "name": "정우성",
  "specialty": "내과",
  "available_days": ["월", "수", "금"],
  "created_at": "2025-10-22T14:30:00"
}
```

예약(appointments) 컬렉션


```
{
  "_id": "671f84c3a913",
  "patient": {
    "id": "671f84a82a01",
    "name": "이동원"
  },
  "doctor": {
    "id": "671f84b42e1a",
    "name": "정우성"
  },
  "reserved_at": "2025-10-26T10:00:00",
  "note": "초진: 기침 및 발열 증상",
  "status": "예약완료"
}
```

1:N 중첩 구조 확장형 (한 환자에 여러 예약 내장)

```
{
  "name": "이동원",
  "birth_date": "1993-01-01",
  "appointments": [
    {
      "doctor": "정우성",
      "reserved_at": "2025-10-21T14:00:00",
      "note": "감기 진료"
    },
    {
      "doctor": "하정우",
      "reserved_at": "2025-10-22T10:30:00",
      "note": "정기검진"
    }
  ]
}
```

> *JOIN 없이 “한 환자 + 모든 예약 내역”을 즉시 확인 가능.*

Python과 DB 연동

> *Python에서는 다양한 DB 연동 도구가 존재합니다.*

구분	도구	설명
RDB용 ORM	SQLAlchemy	가장 널리 쓰이는 ORM (관계형 DB용)
비동기 RDB용	Tortoise ORM	비동기 FastAPI와 잘 어울리는 ORM
NoSQL용	PyMongo / Beanie	MongoDB 같은 문서형 DB와 연동

ORM(Object Relational Mapping)이란?

> *ORM = 객체(Object) 와 테이블(Relational) 을 매핑하는 도구*

- SQL을 몰라도 Python 객체만으로 DB를 다룰 수 있음
- 데이터를 **"코드로 정의하고 다루는"** 방식

비유: 번역기

> *개발자는 Python으로 말하고, ORM이 SQL로 번역*

ORM의 장점과 단점

장점	단점
SQL 없이도 데이터 조작 가능	SQL 구조를 완전히 이해하지 못하면 한계
유지보수 편리	복잡한 쿼리 성능은 직접 튜닝 필요
코드로 DB 구조 정의	내부 동작을 이해해야 효율적 사용 가능

> *즉, ORM은 "편하지만 완전한 마법은 아님!"*
> *SQL 기초는 결국 알아야 함!!*

Tortoise ORM 소개

- Django ORM 스타일의 **간결한 문법**
- **비동기(async)** 지원 → FastAPI와 궁합 최고
- SQLite, MySQL, PostgreSQL 등 지원
- 마이그레이션 도구: **Aerich**

설치

```
pip install tortoise-orm aiomysql aerich
```

> *aiomysql = MySQL 비동기 드라이버*
>
> *aerich = 마이그레이션 도구*

프로젝트 구조

```
tortoise_mysql_demo/
 ├─ app.py
 ├─ models.py
 ├─ config.py
```

Tortoise ORM의 기본 구조

단계	구성 요소	설명
모델(Model) 정의	`class Patient(models.Model)`	Python 클래스로 테이블 구조 정의
초기화(Init)	`await Tortoise.init(config=...)`	DB 연결 및 모델 로딩
스키마 생성 (Schema)	`await Tortoise.generate_schemas()`	모델 기반으로 DB 테이블 생성
데이터 조작 (CRUD)	`await Model.create()`, `.get()`, `.filter()`	ORM 메서드로 데이터 입출력
마이그레이션 (Optional)	`aerich migrate / upgrade`	모델 변경 사항을 DB에 반영

필드명	설명	DB 컬럼 타입 예시	기본값/옵션
`IntField`	정수형	`INT`	`pk=True` 로 기본키 가능
`SmallIntField`	작은 정수형	`SMALLINT`	
`BigIntField`	큰 정수형	`BIGINT`	
`FloatField`	실수형 (소수점)	`FLOAT`	
`DecimalField`	고정 소수점	`DECIMAL(10,2)`	`max_digits`
`CharField`	짧은 문자열	`VARCHAR(n)`	`max_length` 필수
`TextField`	긴 문자열	`TEXT`	길이 제한 없음
`BooleanField`	참/거짓	`BOOLEAN`	기본값: `False`
`UUIDField`	UUID 저장	`UUID`	자동 생성 가능
`JSONField`	JSON 구조 저장	`JSON` / `JSONB`	dict, list 등 직렬화

필드명	설명	DB 컬럼 타입 예시	비고
`DateField`	날짜 (연, 월, 일)	`DATE`	`datetime.date` 객체 사용
`DatetimeField`	날짜 + 시간	`DATETIME` / `TIMESTAMP`	`auto_now_add`, `auto_now` 옵션 가능
`TimeField`	시간만	`TIME`	
`TimedeltaField`	시간 차(기간)	`INTERVAL` (지원 DB 한정)	

관계 설정

옵션	동작	설명
`CASCADE`	부모 삭제 시 자식도 삭제	“연쇄 삭제”
`PROTECT`	부모 삭제 막기 (에러 발생)	“보호”
`SET_NULL`	부모 삭제 시 FK → NULL	“기록은 남기되 참조 끊기”
`SET_DEFAULT`	부모 삭제 시 FK → 기본값	“기본 담당자에게 넘김”
`SET(func)`	부모 삭제 시 FK → 함수 결과	“동적 기본값 설정”
`RESTRICT`	참조 중이면 삭제 제한	“삭제 불가”

ForeignKeyField

다른 테이블(모델)을 참조하는 외래키

```
doctor = fields.ForeignKeyField(
    "models.Doctor",
    related_name="appointments",
    on_delete=fields.CASCADE
)
```

> *하나의 객체가 다른 테이블의 한 레코드에 연결될 때 사용*
>
> *(예: Appointment → Doctor)*

OneToOneField

두 모델이 1:1 관계를 가질 때 사용

```
profile = fields.OneToOneField(
    "models.Profile",
    on_delete=fields.CASCADE
)
```

> *한 유저는 하나의 프로필만,*
> *프로필도 하나의 유저만 가질 수 있을 때 사용*
> *(예: User → Profile)*

ManyToManyField

두 모델이 서로 다대다 관계를 가질 때 사용 (중간 테이블 자동 생성)

```
tags = fields.ManyToManyField(
    "models.Tag",
    related_name="articles"
)
```

> *하나의 게시글이 여러 태그를 가질 수 있고,*
> *하나의 태그도 여러 게시글에 붙을 수 있을 때 사용*
> *(예: Article ↔️ Tag)*

모델 정의 - Patient

```
from tortoise import fields, models

class Patient(models.Model):
    id = fields.IntField(pk=True)
    name = fields.CharField(max_length=100, index=True)
    birth_date = fields.DateField(null=True)

    def __str__(self):
        return self.name
```

> *환자(Patient)는 기본 정보만 저장*
>
> *예약(Appointment)과 1:N 관계로 연결*

모델 정의 - Doctor

```
class Doctor(models.Model):
    id = fields.IntField(pk=True)
    name = fields.CharField(max_length=100, index=True)
    specialty = fields.CharField(max_length=100, null=True)

    def __str__(self):
        return f"{self.name} ({self.specialty})"
```

> *의사(Doctor)는 여러 예약(Appointment)을 가질 수 있음*

모델 정의 - Appointment

```
class Appointment(models.Model):
    id = fields.IntField(pk=True)

    patient = fields.ForeignKeyField(
        "models.Patient",
        related_name="appointments",
        on_delete=fields.CASCADE,
    )

    doctor = fields.ForeignKeyField(
        "models.Doctor",
        related_name="appointments",
        on_delete=fields.PROTECT,
    )

    reserved_at = fields.DatetimeField(index=True)
    note = fields.CharField(max_length=255, null=True)
```

> *예약(Appointment)은 **환자와 의사 모두에 대한 FK**를 가짐.*

관계 구조

모델	관계	설명
Patient ↔️ Appointment	1:N	한 환자는 여러 예약을 가질 수 있음
Doctor ↔️ Appointment	1:N	한 의사는 여러 예약을 가질 수 있음

> *Appointment가 **중간 연결 테이블** 역할*

config.py 설정

```
TORTOISE_ORM = {
    "connections": {
        "default": "mysql://USER:PASSWORD@HOST:3306/DBNAME?charset=utf8mb4"
    },
    "apps": {
        "models": {
            "models": ["models", "aerich.models"],
            "default_connection": "default",
        },
    },
}
```

> `"aerich.models"` *는 마이그레이션 관리용 내부 테이블*

비동기(Async)란?

“하나 끝날 때까지 기다리지 말고, 다른 일도 같이 하자!”

동기(Synchronous)

```
import time

def make_coffee():
    print("커피 주문")
    time.sleep(3)
    print("커피 완료")

def eat_snack():
    print("과자 먹기")

make_coffee()
eat_snack()
```

- 코드도 순서대로 실행되고, 앞의 작업이 끝나야 다음으로 넘어감

```python
import asyncio

async def make_coffee():
    print("커피 주문")
    await asyncio.sleep(3)
    print("커피 완료")

async def eat_snack():
    print("과자 먹기")

async def main():
    await asyncio.gather(make_coffee(), eat_snack())

await main()
```

CREATE — 데이터 삽입

```
from datetime import datetime, timedelta

dr_lee = await Doctor.create(name="이병헌", specialty="내과")
dr_park = await Doctor.create(name="박명수", specialty="소아과")

p_lee = await Patient.create(name="이동원", birth_date="1993-01-01")
p_kim = await Patient.create(name="김민찬", birth_date="2000-05-10")

appt1 = await Appointment.create(
    patient=p_lee, doctor=dr_lee,
    reserved_at=datetime.now() + timedelta(days=1),
    note="초진 예약"
)
appt2 = await Appointment.create(
    patient=p_lee, doctor=dr_park,
    reserved_at=datetime.now() + timedelta(days=2),
    note="소아과 상담"
)
```

READ — 조회(Get/Filter)

```
one_doctor = await Doctor.get(id=dr_lee.id)
print("단건 의사:", one_doctor)

recent_patients = await Patient.filter(birth_date__gte="1990-01-01").order_by("name")
print("조건 환자:", [str(p) for p in recent_patients])

rows = await Appointment.all().select_related("patient", "doctor")
for a in rows:
    print(f"[예약 {a.id}] {a.patient.name} → {a.doctor.name} ({a.note})")

lee_with_appts = await Patient.get(id=p_lee.id).prefetch_related("appointments")
print("예약 수:", len(lee_with_appts.appointments))
```

UPDATE — 수정 후 저장

```
dr_lee.specialty = "가정의학과"
await dr_lee.save()

p_lee.birth_date = "1994-02-02"
await p_lee.save()

appt1.note = "증상 업데이트: 기침/두통/몸살"
appt1.reserved_at = appt1.reserved_at + timedelta(days=1)
await appt1.save()

updated = await Appointment.get(id=appt1.id).select_related("patient", "doctor")
print(f"수정 후: [예약 {updated.id}] {updated.patient.name} → {updated.doctor.name} ({updated.note})")
```

> *객체를 가져와서 필드 값을 바꾸고 await obj.save()를 꼭 호출*

DELETE — 삭제 (순서 주의)

```
await appt2.delete()
print("삭제한 예약:", appt2.id)

await p_lee.delete()
print("삭제한 환자:", "이동원")

await dr_park.delete()
print("삭제한 의사:", "박의사")
```

app.py - CRUD 예제

```python
from datetime import datetime, timedelta
from tortoise import Tortoise, run_async
from models import Patient, Doctor, Appointment
from config import TORTOISE_ORM

async def main():
    await Tortoise.init(db_url=DB_URL, modules={"models": ["__main__"]})
    await Tortoise.generate_schemas()

    # --- CREATE ---
    dr = await Doctor.create(name="이병헌", specialty="내과")
    p = await Patient.create(name="이동원", birth_date="1993-01-01")
    a = await Appointment.create(patient=p, doctor=dr, reserved_at=datetime.now(), note="초진")

    # --- READ ---
    rows = await Appointment.all().select_related("patient", "doctor")
    for r in rows: print(r.id, r.patient.name, "→", r.doctor.name)

    # --- UPDATE ---
    a.note = "증상 업데이트"
    await a.save()

    # --- DELETE ---
    await a.delete()
    await p.delete()
    await dr.delete()

    await Tortoise.close_connections()

if __name__ == "__main__":
    asyncio.run(main())
```

CRUD 정리

동작	코드	비고
Create	`await Model.create()`	새 레코드
Read	`await Model.get()` / `.filter()`	조건 검색
Update	`obj.field = val; await obj.save()`	값 변경
Delete	`await obj.delete()`	삭제

실습

- Python tortoise로 ORM 만들기
- 환자, 의사, 진료 모델(테이블) 만들기
- crud 로직 구현