데이터베이스

- **데이터베이스**
  - 여러 사람이 데이터를 공유해서 사용하기 위해 체계적으로 통합해서 묶어놓은 데이터 집합
  - 컴퓨터 시스템에서 데이터베이스는 데이터 집합을 관리하고 접근할 수 있게 하는 프로그램을 의미함
- **데이터베이스의 장점**
  - 데이터 중복 최소화 및 저장공간 절약
  - 효율적인 데이터 접근 및 빠른 접근 속도
  - 일관성, 무결성, 보안성 제공
  - 데이터 표준화
- **관계형 데이터모델(Relational Data Model)**
  - 데이터를 테이블에 대입하여 관계를 묘사하는 이론적 모델
  - 관계형 데이터모델을 다루는 데이터베이스 시스템을 관계형 데이터베이스 관리시스템(Relational DataBase Management System, RDBMS)이라고 함
- **SQL(Structured Query Language)**
  - SQL은 관계형 데이터베이스를 관리하기 위해 사용하는 **질의** 언어
  - 데이터베이스 스키마(Schema) 생성
  - 데이터 생성(Create)
  - 데이터 읽기(Retrieve, Read)
  - 데이터 수정(Update)
  - 데이터 삭제(Delete)
  - CRUD는 데이터를 관리하기 위해 기본적으로 필요한 기능

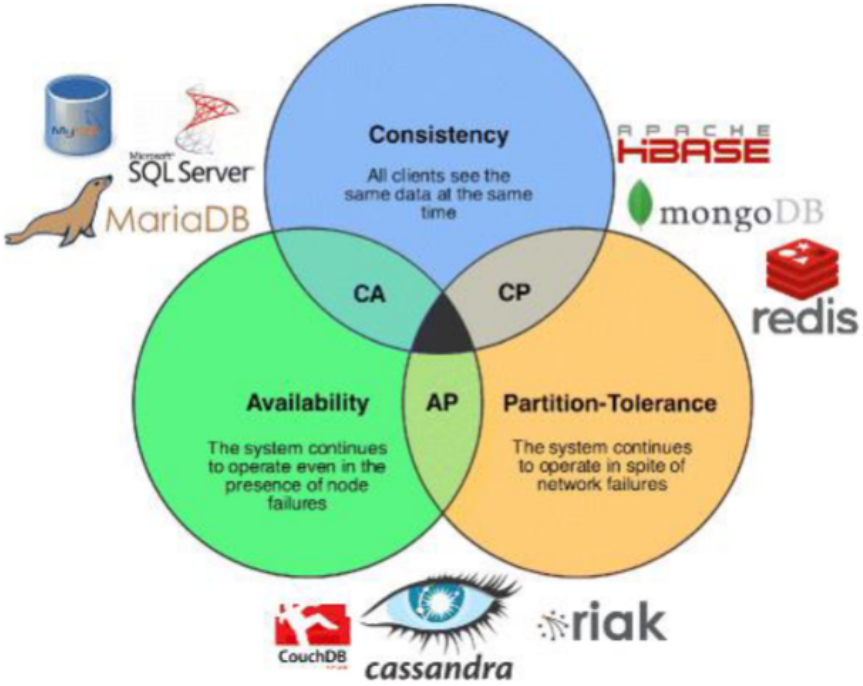
NoSQL

- NoSQL
  - SQL이 아니라는 의미에서 NoSQL이라고 하기도 하고 “Not Only SQL” 이라는 의미에서 전통적인 관계형 데이터베이스의 한계를 극복하기 위한 다양한 저장 방식을 지원함
- NoSQL 주요 개념
  - 스키마 유연성
    - RDBMS와 달리 사전에 정의된 스키마가 필요하지 않아 데이터를 자유롭게 저장할 수 있음
    - 구조가 자주 변경 되는 경우에 유리함
  - 수평적 확장
    - 데이터를 여러 서버에 분산하여 저장할 수 있음
    - 대규모 서비스에 유리함
  - 다양한 데이터 모델
    - 키-값: Redis, DynamoDB
    - 문서: MongoDB, CouchDB
    - 그래프: Neo4

CAP 이론

- **CAP Theory**
  - CAP 이론은 Consistency(일관성), Availability(가용성), Partition Tolerance(분할허용)의 세가지 속성을 **분산 시스템**에서 동시에 완벽히 만족할 수 없다는 이론
- **CAP 이론 세가지 요소**
  - **일관성(Consistency)**
    - 누가 데이터를 조회하더라도 같은 데이터를 얻을 수 있어야 함
    - 즉, 한 노드에서 데이터가 업데이트 되면 모든 노드가 동일한 값을 가져야함
  - **가용성(Availability)**
    - 시스템이 항상 응답할 수 있는 상태를 유지
  - **분할 허용성(Partition Tolerance)**
    - 노드간 연결이 끊어져서 네트워크 분할이 발생해도 시스템은 계속 동작해야함
- **CAP가 동시에 만족하지 못하는 이유**

    - CAP 이론은 분산 시스템을 가정하고 있으므로, 네트워크 장애로 인해 분할이 발생한 것을 가정함
    - 일관성과 가용성의 충돌
        - 네트워크 분할이 발생하면 노드 간에 데이터를 동기화 하지 못함
        - **일관성**을 유지하려면 네트워크 분할 문제가 해결되어 동기화 될 때까지 **가용성**을 제공 못함
        - **가용성**을 제공하려면 해당 데이터가 최신 데이터가 아닐지라도 응답을 해야하므로 **일관성**이 희생됨
- CAP-RDBMS
    - 전통적으로 단일 서버환경(소규모)을 기반으로 설계되었기 때문에 분산 처리보다는 트랜잭션의 일관성 과 데이터 무결성을 더 중요시 여기는 경향
    - 일관성과 가용성을 중시함 (CA)
- CAP-NoSQL
    - 분할 허용성을 기본으로 하고 일관성과 가용성 중 하나를 선택함



MySQL

MySQL 기본 명령어

- 셸에서 mysql에 접속하여 mysql 콘솔에서 명령어를 작업
- 모든 명령어는 세미콜론으로 끝남
- (참고) 관리자 계정으로 접속 가능한 DB는 일괄 제공함
    - mysql 계정 = mysql 비밀번호 = linux 계정 = DB_NAME

명령어	내용
$ mysql –u *ACCOUNT* –p	shell에서 mysql에 ACCOUNT 계정으로 접속
show databases;	데이터베이스 목록 출력
create database DB_NAME;	데이터베이스 DB_NAME 생성
use DB_NAME;	사용할 데이터베이스로 DB_NAME을 선택

내용	예시 코드
데이터베이스 생성	`CREATE DATABASE example_db;`

테이블 생성	`CREATE TABLE users (`
	`  id INT AUTO_INCREMENT PRIMARY KEY,`
	`  name VARCHAR(100),`
	`  email VARCHAR(100)`
	`);`

데이터 삽입	`INSERT INTO users (name, email) VALUES ('Jonggyu', 'pjk5401@dau.ac.kr');`
데이터 조회	`SELECT * FROM users;`
데이터 조회(조건)	`SELECT * FROM users WHERE name = 'Jonggyu';`

데이터 업데이트	UPDATE users SET email = 'pjk5401@gmail.com' WHERE name = 'jonggyu';
데이터 삭제	DELETE FROM users WHERE name = 'jonggyu';
테이블 수정	ALTER TABLE users ADD COLUMN age INT;
테이블 삭제	DROP TABLE users;
데이터베이스 삭제	DROP DATABASE example_db;

기본명령어 코드 실습

- shell에서 mysql에 ACCOUNT 계정으로 접속

```
$ mysqll -u [ACCOUNT] -p
```

- 데이터베이스 목록 출력

```
show databases;
```

- 데이터베이스 DB_NAME 생성

```
create database DB_NAME;
```

- 사용할 데이터베이스로 DB_NAME을 선택

```
use DB_NAME;
```

- 데이터베이스 생성

```
CREATE DATABASE example_DB;
```

- 테이블 생성

```
CREATE TABLE users (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100);
        email VARCHAR(100);
);
```

- 데이터 삽입

```
INSERT INTO users(name, email)
VALUES('Umyunsang','uys1705817');
```

- 데이터 조회

```
SELECT * FROM users;
SELECT * FROM users WHERE [조건식] ;
```

- 데이터 업데이트

```
UPDATE users SET email='uys1705817' WHERE name='umyunsang';
```

- 데이터 삭제

```
DELETE FROM users WHERE name='umyunsang';
```

- 데이터 수정

```
ALTER TABLE users ADD COLUMN age INT;
```

- 테이블 삭제

```
DROP TABLE users;
```

- 데이터베이스 삭제

```
DROP DATABASE example_DB;
```

12주차 소스코드

- 가상환경에서 라이브러리 설치

```
(venv) $ pip install mysql-connector-python
```

```
CREATE TABLE courses (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    instructor VARCHAR(255),
    location VARCHAR(255),
    time VARCHAR(255)
);

-- Students 테이블
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    student_id VARCHAR(50) UNIQUE NOT NULL
);

-- Enrollments 테이블 (수업과 학생 간 Many-to-Many 관계)
CREATE TABLE enrollments (
    course_id INT,
    student_id INT,
    PRIMARY KEY (course_id, student_id),
    FOREIGN KEY (course_id) REFERENCES courses(id) ON DELETE CASCADE,
    FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE
);
```

```
from flask import Flask, jsonify, request, abort
import mysql.connector
from mysql.connector import Error

app = Flask(__name__)

# MySQL 데이터베이스 연결 설정
db_config = {
    'host': 'localhost',
    'user': 'uys_1705817',
    'password': 'uys_1705817',
    'database': 'uys_1705817'
}

# MySQL 연결 함수
def get_db_connection():
    try:
        conn = mysql.connector.connect(**db_config)
        return conn
    except Error as e:
        print(f"Error: {e}")
        return None

# 수업 CRUD
@app.route('/courses', methods=['GET'])
```

```python
def get_courses():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM courses")
    courses = cursor.fetchall()
    conn.close()
    return jsonify(courses)

@app.route('/courses', methods=['POST'])
def add_course():
    data = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "INSERT INTO courses (name, instructor, location, time) VALUES (%s, %s, %s, %s)"
    values = (data['name'], data['instructor'], data['location'], data['time'])
    cursor.execute(query, values)
    conn.commit()
    conn.close()
    return jsonify({'message': 'Course added successfully'}), 201

@app.route('/courses/<int:course_id>', methods=['PUT'])
def update_course(course_id):
    data = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "UPDATE courses SET name=%s, instructor=%s, location=%s, time=%s WHERE id=%s"
    values = (data['name'], data['instructor'], data['location'], data['time'], course_id)
    cursor.execute(query, values)
    conn.commit()
    conn.close()
    return jsonify({'message': 'Course updated successfully'})

@app.route('/courses/<int:course_id>', methods=['DELETE'])
def delete_course(course_id):
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "DELETE FROM courses WHERE id=%s"
    cursor.execute(query, (course_id,))
    conn.commit()
    conn.close()
    return jsonify({'message': 'Course deleted successfully'})

# 학생 CRUD
@app.route('/students', methods=['GET'])
def get_students():
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("SELECT * FROM students")
    students = cursor.fetchall()
    conn.close()
    return jsonify(students)

@app.route('/students', methods=['POST'])
def add_student():
    data = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "INSERT INTO students (name, student_id) VALUES (%s, %s)"
    values = (data['name'], data['student_id'])
    cursor.execute(query, values)
    conn.commit()
    conn.close()
    return jsonify({'message': 'Student added successfully'}), 201

@app.route('/students/<int:student_id>', methods=['PUT'])
def update_student(student_id):
    data = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "UPDATE students SET name=%s, student_id=%s WHERE id=%s"
```

```
    values = (data['name'], data['student_id'], student_id)
    cursor.execute(query, values)
    conn.commit()
    conn.close()
    return jsonify({'message': 'Student updated successfully'})

@app.route('/students/<int:student_id>', methods=['DELETE'])
def delete_student(student_id):
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "DELETE FROM students WHERE id=%s"
    cursor.execute(query, (student_id,))
    conn.commit()
    conn.close()
    return jsonify({'message': 'Student deleted successfully'})

# 수업 신청/취소
@app.route('/enrollments', methods=['POST'])
def enroll_student():
    data = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "INSERT INTO enrollments (course_id, student_id) VALUES (%s, %s)"
    values = (data['course_id'], data['student_id'])
    cursor.execute(query, values)
    conn.commit()
    conn.close()
    return jsonify({'message': 'Student enrolled successfully'})

@app.route('/enrollments', methods=['DELETE'])
def unenroll_student():
    data = request.json
    conn = get_db_connection()
    cursor = conn.cursor()
    query = "DELETE FROM enrollments WHERE course_id=%s AND student_id=%s"
    values = (data['course_id'], data['student_id'])
    cursor.execute(query, values)
    conn.commit()
    conn.close()
    return jsonify({'message': 'Student unenrolled successfully'})

if __name__ == '__main__':
    app.run(debug=True, host="0.0.0.0", port=15022)
```