

데브옵스(DevOps)

2. DBMS 2

❖ 수업 목표

- SQL 문장의 문법을 이해하고 코드를 작성할 수 있다.
- 조인 문법을 이해하고 코드를 작성할 수 있다.
- MySQL을 사용하여 데이터베이스에 연결하고 테이블을 생성할 수 있다.
- MySQL을 사용하여 데이터를 삽입, 조회, 수정, 삭제 할 수 있다.

❖ 세부 목표

- 2.1 SQL 문장
- 2.2 조인
- 2.3 파이썬 DB 연동 (1)
- 2.4 파이썬 DB 연동 (2)

❖ SQL 문장

■ DML

- **SELECT** : 테이블이나 뷰에 있는 데이터를 조회
- **INSERT** : 데이터를 신규로 생성
- **UPDATE** : 기 생성된 데이터를 수정
- **DELETE** : 데이터 삭제

■ TCL

- **COMMIT** : 변경된 데이터를 최종 적용
- **ROLLBACK** : 변경된 데이터를 적용하지 않고 이전 상태로 되돌림

❖ INSERT 문

- 새 데이터를 입력해 넣을 때 사용하는 문장

- INSERT 구문1 (기본형태)

- INSERT INTO 테이블명 (컬럼1, 컬럼2, ...)
- VALUES (값1, 값2, ...);

- 각 절 설명

- INSERT INTO : 새로운 데이터를 추가하기 위한 테이블의 컬럼 지정
 - 컬럼 목록을 명시적으로 지정하는 것을 권장 (DEFAULT 값 활용 가능)
- VALUES : 테이블 각 컬럼에 추가할 데이터 값
 - 괄호 안의 값 순서와 타입이 테이블명 괄호 안 컬럼의 순서, 타입과 일치해야 함

❖ INSERT 문

■ 예제

- **영업팀 연봉 5000 김철수 입사**
 - `INSERT INTO employees (name, department_name, salary)`
 - `VALUES ('김철수', '영업팀', 5000);`
- **팀(미정) 연봉 4000 이영희 입사**
 - `INSERT INTO Employees (employee_id, name, salary)`
 - `VALUES (2, '이영희', 4000);`
- **팀(미정) 연봉(미정) 박민수 입사**
 - `INSERT INTO Employees (employee_id, name, hiredate)`
 - `VALUES (3, '박민수', '2025-02-05');`

❖ INSERT 문

■ INSERT 구문2 (컬럼명 기술 생략 형태)

- INSERT INTO 테이블명
- VALUES (값1, 값2, ...);

■ 각 절 설명

- INSERT INTO : 새로운 데이터를 추가하기 위한 테이블지정
 - 컬럼 목록을 명시하지 않았으므로 모든 컬럼에 값을 넣는다는 의미
- VALUES : 테이블 각 컬럼에 추가할 데이터 값
 - 괄호 안의 값 순서와 타입이 테이블에 있는 모든 컬럼 순서, 타입과 일치해야 함
 - 컬럼 순서는 테이블 생성 시 명시한 것과 일치

❖ INSERT 문

■ 예제

- **영업팀 연봉 4500 박명수 입사**
 - INSERT INTO employees
 - VALUES (4, '박명수', '영업팀', 4500, '2025-02-06');
- **개발팀 연봉 7000 유재석 입사**
 - INSERT INTO Employees
 - VALUES (5, '유재석', '개발팀', 7000, '2025-02-06');

❖ INSERT 문

■ INSERT 구문3 (INSERT~SELECT 형태)

- INSERT INTO 테이블명 (컬럼1, 컬럼2, ...)
- SELECT 문;

■ 각 절 설명

- INSERT INTO : 새로운 데이터를 추가하기 위한 테이블지정
 - 컬럼 목록을 명시하지 않았으므로 모든 컬럼에 값을 넣는다는 의미
- SELECT 문 : VALUES 절에 값을 각각 명시하는 대신 조회된 내용을 사용
 - 테이블명 괄호 안의 컬럼 순서와 타입이 SELECT 문장의 컬럼 순서, 타입과 일치해야 함
 - 테이블명 다음 컬럼 목록을 생략 가능. 이 경우, 테이블의 모든 컬럼에 값을 삽입
- 특정 조건을 만족하는 데이터를 새로운 테이블에 복사할 때 유용

❖ INSERT 문

■ 예제

- 새 테이블 생성
 - `CREATE TABLE new_employees (`
 - `employee_id INT PRIMARY KEY AUTO_INCREMENT,`
 - `name VARCHAR(50) NOT NULL,`
 - `department_name VARCHAR(50),`
 - `salary DECIMAL(10,2) CHECK (salary > 0),`
 - `hiredate DATE DEFAULT (CURRENT_DATE)`
 - `);`
- 연봉이 5000 미만인 사원 목록을 새 테이블에 삽입
 - `INSERT INTO new_employees`
 - `SELECT * FROM employees WHERE salary < 5000;`

1. SQL 문장

❖ SELECT 문

- 테이블이나 뷰에 있는 데이터를 선택(조회) 시, 사용

- SELECT 구문

- SELECT 컬럼명
- FROM 테이블명
- [WHERE 조건]
- [ORDER BY 컬럼명];

- 각 절 설명

- SELECT : 선택하고자 하는 컬럼명, 모든 컬럼을 조회하고 싶다면 *
- FROM : 선택할 테이블이나 뷰 명
- WHERE : 선택 조건, 여러 조건 기술 시에는 AND, OR로 연결
- ORDER BY : 조회 데이터 정렬 시, 정렬하고자 하는 컬럼명 기술

❖ SELECT 문

■ 예제

- **사원 테이블로부터 모든 데이터 조회**
 - `SELECT * FROM employees;`
- **사원 테이블로부터 name, salary 컬럼 데이터 조회**
 - `SELECT name, salary FROM employees;`
- **사원 테이블로부터 연봉이 6000 초과인 name, salary 컬럼 데이터 조회**
 - `SELECT name, salary`
 - `FROM employees`
 - `WHERE salary > 6000;`
- **사원 테이블로부터 연봉 기준 내림차순으로 모든 데이터 조회**
 - `SELECT *`
 - `FROM employees`
 - `ORDER BY Salary DESC;`

❖ UPDATE 문

- 테이블에 있는 기존 데이터를 수정하는 문장

■ UPDATE 구문

- UPDATE 테이블명
- SET 컬럼1 = 변경값1,
- 컬럼2 = 변경값2,
- ...
- [WHERE 조건];

■ 각 절 설명

- UPDATE : 업데이트 대상 테이블명 명시
- SET : 변경하고자 하는 컬럼과 그 값을 명시
 - 여러 컬럼을 갱신할 때는 콤마로 분리
- WHERE : 데이터 갱신 조건. 이 조건에 맞는 데이터만 변경됨
 - WHERE 조건을 생략하면 테이블에 있는 모든 데이터가 변경

❖ UPDATE 문

■ 예제

- **사원 테이블에서 사원 번호가 2인 데이터의 부서명을 영업지원팀으로 수정**
 - UPDATE employees
 - SET department_name = '영업지원팀'
 - WHERE employee_id = 2;
- **사원 테이블에서 사원 번호가 3인 데이터의 부서명을 연구소로, 연봉을 8000 으로 수정**
 - UPDATE employees
 - SET department_name = '연구소', salary = 8000
 - WHERE employee_id = 3;
- **사원 테이블에서 연봉이 5000 미만인 데이터의 연봉을 5% 인상하도록 수정**
 - UPDATE employees
 - SET salary = salary + (salary * 0.05)
 - WHERE salary < 5000;

❖ DELETE 문

- 테이블에 있는 데이터를 삭제하는 문장
- DELETE 구문
 - DELETE FROM 테이블명
 - [WHERE delete조건];
- 각 절 설명
 - DELETE FROM : 삭제하고 싶은 데이터가 포함된 대상 테이블명 명시
 - WHERE : 데이터 삭제 조건. 이 조건에 맞는 데이터만 삭제됨
 - WHERE 조건을 생략하면 테이블에 있는 모든 데이터가 삭제

❖ DELETE 문

■ 예제

- **사원 테이블에서 연봉이 7000 이상인 데이터를 삭제**
 - **DELETE FROM employees**
 - **WHERE salary >= 7000;**
- **사원 테이블에서 사원 번호가 4인 데이터를 삭제**
 - **DELETE FROM employees**
 - **WHERE employee_id = 4;**

❖ COMMIT 문

- 변경한 데이터를 데이터베이스에 최종적으로 반영
- 기본적으로 자동 커밋(AUTOCOMMIT) 설정

■ COMMIT 구문

- COMMIT;
- AUTOCOMMIT 여부 확인
 - SELECT @@AUTOCOMMIT;
- AUTOCOMMIT 설정 및 해제
 - SET AUTOCOMMIT = 1; -- 설정 (기본 설정)
 - SET AUTOCOMMIT = 0; -- 해제 (세션 단위, 일시적)
 - 일시적 해제가 아닌 완전히 끄려면 다음 경로의 my.ini 파일을 수정
 - » C:\ProgramData\MySQL\MySQL Server X.X\my.ini
 - » [mysqld] 섹션에 autocommit = 0 추가
 - » **서비스에서 MySQL 서버 재시작**

❖ COMMIT 문

■ 예제

- 현재까지 SQL 문장을 사용하여 변경된 사항을 저장
 - COMMIT;
- 개발팀 연봉 6000 조세호 입사 데이터 삽입 및 조회
 - INSERT INTO Employees
 - VALUES (3, '조세호', '개발팀', 6000, '2025-02-07');
 - SELECT * FROM employees;
- MySQL Command Line Client 종료 및 재실행
 - 실행 결과

```
mysql> select * from employees;
```

employee_id	name	department_name	salary	hiredate
1	김철수	영업팀	5000.00	2025-02-02
2	이영희	영업지원팀	4200.00	2025-02-02

2 rows in set (0.00 sec)

❖ ROLLBACK 문

- 변경한 데이터를 변경 전 상태로 되돌림
- ROLLBACK 구문
 - ROLLBACK;

❖ ROLLBACK 문

■ 예제

- 개발팀 연봉 6000 조세호 입사 데이터 삽입 및 조회
 - » INSERT INTO Employees
 - » VALUES (3, '조세호', '개발팀', 6000, '2025-02-07');
 - » SELECT * FROM employees;
- 현재까지 SQL 문장을 사용하여 변경된 사항을 저장
 - COMMIT;
- employees 테이블에서 사원 번호가 3인 데이터를 삭제
 - » DELETE FROM employees WHERE employee_id = 3;
- 변경한 데이터를 변경 전 상태로 되돌림
 - ROLLBACK;
- 사원 번호가 3인 데이터가 복구 되었는지 확인
 - » SELECT * FROM employees;

❖ SAVEPOINT 문

- 특정 시점으로 롤백할 수 있도록 저장점을 설정
- SAVEPOINT 구문
 - SAVEPOINT 저장점_이름;

❖ SAVEPOINT 문

■ 예제

- 현재까지 SQL 문장을 사용하여 변경된 사항이 적용된 저장점 생성
 - `SAVEPOINT sp1;`
 - 개발팀 연봉 7000 박나래 입사 데이터 삽입 및 조회
 - » `INSERT INTO Employees`
 - » `VALUES (4, '박나래', '기획팀', 7000, '2025-02-08');`
 - » `SELECT * FROM employees;`
- 현재까지 SQL 문장을 사용하여 변경된 사항이 적용된 저장점 생성
 - `SAVEPOINT sp2;`
 - 개발팀 연봉 8000 전현무 입사 데이터 삽입 및 조회
 - » `INSERT INTO Employees`
 - » `VALUES (5, '전현무', '개발팀', 6000, '2025-02-07');`
 - » `SELECT * FROM employees;`

❖ SAVEPOINT 문

■ 예제

- 변경한 데이터를 변경 전 sp2 저장점 상태로 되돌림
 - ROLLBACK TO sp2;
 - 사원 번호가 5인 데이터가 삽입 전 상태로 복구 되었는지 확인
 - » SELECT * FROM employees;
- 변경한 데이터를 변경 전 sp1 저장점 상태로 되돌림
 - ROLLBACK TO sp1;
 - 사원 번호가 4인 데이터가 삽입 전 상태로 복구 되었는지 확인
 - » SELECT * FROM employees;

❖ 조인(JOIN) 개요

■ 조인 소개

- 테이블 간의 관계를 맺는 방법
- 두 개 이상의 테이블에서 데이터를 결합하여 조회할 때 사용

■ 조인의 종류 및 개념

- INNER JOIN: 두 테이블에서 일치하는 데이터만 반환
- LEFT JOIN (LEFT OUTER JOIN): 왼쪽 테이블의 모든 행과 일치하는 오른쪽 테이블의 데이터를 반환
- RIGHT JOIN (RIGHT OUTER JOIN): 오른쪽 테이블의 모든 행과 일치하는 왼쪽 테이블의 데이터를 반환
- FULL JOIN (FULL OUTER JOIN): 두 테이블의 모든 행을 반환 (MySQL에서는 지원하지 않으며 UNION을 사용함)
- CROSS JOIN: 두 테이블의 모든 행을 조합한 결과를 반환
- SELF JOIN: 같은 테이블을 자기 자신과 조인

❖ INNER JOIN (내부 조인)

- 두 테이블에서 일치하는 데이터만 반환
- 문법
 - SELECT A.컬럼명, B.컬럼명
 - FROM 테이블A AS A
 - INNER JOIN 테이블B AS B
 - ON A.공통컬럼 = B.공통컬럼;

❖ INNER JOIN (내부 조인)

■ 예제에서 사용할 테이블 생성

- **CREATE TABLE Departments (**
- **DeptID INT PRIMARY KEY,**
- **DeptName VARCHAR(50) NOT NULL**
- **);**

- **CREATE TABLE Employees (**
- **ID INT PRIMARY KEY,**
- **Name VARCHAR(100) NOT NULL,**
- **DeptID INT,**
- **ManagerID INT**
- **);**

❖ INNER JOIN (내부 조인)

■ 예제에서 사용할 데이터 삽입

- -- Departments 테이블에 데이터 삽입
 - INSERT INTO Departments (DeptID, DeptName) VALUES
 - (1, 'HR'),
 - (2, 'Engineering'),
 - (3, 'Marketing'),
 - (4, 'planning');
- -- Employees 테이블에 데이터 삽입
 - INSERT INTO Employees (ID, Name, DeptID, ManagerID) VALUES
 - (101, 'Alice', 1, NULL),
 - (102, 'Bob', 2, 101),
 - (103, 'Charlie', 2, 101),
 - (104, 'David', 3, 102),
 - (105, 'Eva', NULL, NULL);

❖ INNER JOIN (내부 조인)

■ 예제

- **Employees 테이블과 Departments 테이블을 조인하여 사원의 이름과 부서명을 조회**
 - **SELECT Employees.Name, Departments.DeptName**
 - **FROM Employees**
 - **INNER JOIN Departments ON Employees.DeptID = Departments.DeptID;**
- **실행 결과**

```
mysql> SELECT Employees.Name, Departments.DeptName
-> FROM Employees
-> INNER JOIN Departments ON Employees.DeptID = Departments.DeptID;
```

Name	DeptName
Alice	HR
Bob	Engineering
Charlie	Engineering
David	Marketing

```
4 rows in set (0.00 sec)
```

❖ LEFT JOIN

- 좌측 테이블을 기준으로 일치하는 데이터만 반환
- 문법
 - SELECT A.컬럼명, B.컬럼명
 - FROM 테이블A AS A
 - LEFT JOIN 테이블B AS B
 - ON A.공통컬럼 = B.공통컬럼;

❖ LEFT JOIN

■ 예제

- **Employees 테이블의 모든 직원 데이터를 가져오되, 부서 정보가 없는 직원도 포함하여 조회**
 - **SELECT Employees.Name, Departments.DeptName**
 - **FROM Employees**
 - **LEFT JOIN Departments ON Employees.DeptID = Departments.DeptID;**
- **실행 결과**

```
mysql> SELECT Employees.Name, Departments.DeptName
-> FROM Employees
-> LEFT JOIN Departments ON Employees.DeptID = Departments.DeptID;
```

Name	DeptName
Alice	HR
Bob	Engineering
Charlie	Engineering
David	Marketing
Eva	NULL

5 rows in set (0.00 sec)

❖ RIGHT JOIN

- 우측 테이블을 기준으로 일치하는 데이터만 반환
- 문법
 - SELECT A.컬럼명, B.컬럼명
 - FROM 테이블A AS A
 - RIGHT JOIN 테이블B AS B
 - ON A.공통컬럼 = B.공통컬럼;

❖ RIGHT JOIN

■ 예제

- Departments 테이블의 모든 부서를 포함하여 직원 정보를 조회
 - SELECT Employees.Name, Departments.DeptName
 - FROM Employees
 - RIGHT JOIN Departments ON Employees.DeptID = Departments.DeptID;
- 실행 결과

```
mysql> SELECT Employees.Name, Departments.DeptName
-> FROM Employees
-> RIGHT JOIN Departments ON Employees.DeptID = Departments.DeptID;
+-----+-----+
| Name   | DeptName |
+-----+-----+
| Alice  | HR       |
| Charlie| Engineering |
| Bob    | Engineering |
| David  | Marketing |
| NULL   | planning |
+-----+-----+
5 rows in set (0.00 sec)
```

❖ SELF JOIN

- 같은 테이블에서 ON절의 조건과 일치하는 데이터만 반환
- 문법
 - SELECT A.컬럼명, B.컬럼명
 - FROM 테이블명 AS A
 - JOIN 테이블명 AS B
 - ON A.공통컬럼 = B.공통컬럼;

❖ SELF JOIN

■ 예제

- **Employees 테이블에서 직원과 해당 직원의 매니저 정보를 조회**

- **SELECT E1.Name AS Employee, E2.Name AS Manager**
- **FROM Employees AS E1**
- **JOIN Employees AS E2 ON E1.ManagerID = E2.ID;**

- **실행 결과**

```
mysql> SELECT E1.Name AS Employee, E2.Name AS Manager
-> FROM Employees AS E1
-> JOIN Employees AS E2 ON E1.ManagerID = E2.ID;
+-----+-----+
| Employee | Manager |
+-----+-----+
| Bob      | Alice   |
| Charlie  | Alice   |
| David    | Bob     |
+-----+-----+
3 rows in set (0.00 sec)
```

3. 파이썬 DB 연동(1)

❖ 파이썬 DB 연동

■ MySQL Connector 모듈 설치

- `pip install pymysql`

■ 임포트 설정

- `import pymysql`

3. 파이썬 DB 연동(1)

❖ 파이썬 DB 연동

■ MySQL 연결 및 데이터베이스 선택

- # MySQL 서버에 연결
- `conn = pymysql.connect(` # Connection class 객체 생성
- `host="localhost",` # MySQL 서버 주소 (로컬인 경우 'localhost')
- `user="root",` # MySQL 사용자명
- `password="password",` # MySQL 비밀번호
- `database="exampledb",` # 사용할 데이터베이스 명
- `charset='utf8mb4',` # UTF-8의 확장 버전
- # DictCursor 클래스: sql 구문 실행 후, 조회된 결과를 딕셔너리 형태로 반환
- `cursorclass=pymysql.cursors.DictCursor`
- `)`
- `cursor = conn.cursor()` # cursor 함수 객체 생성

- # 현재 선택된 데이터베이스 확인
- `cursor.execute("SELECT DATABASE()")`
- `print("현재 데이터베이스:", cursor.fetchone())`
- `conn.close()`

3. 파이썬 DB 연동(1)

❖ 파이썬 DB 연동

■ 테이블 생성 및 데이터 조작

• 테이블 생성

- `conn = pymysql.connect(host="localhost", user="root", password="1234", database="exampledb")`
- `cursor = conn.cursor()`
- `# 테이블 생성 (users 테이블)`
- `cursor.execute("""`
- `CREATE TABLE IF NOT EXISTS users (`
- `id INT AUTO_INCREMENT PRIMARY KEY,`
- `name VARCHAR(100) NOT NULL,`
- `age INT,`
- `email VARCHAR(100) UNIQUE`
- `)`
- `""")`
- `print("테이블 생성 완료")`
- `conn.close()`

4. 파이썬 DB 연동(2)

❖ 파이썬 DB 연동

■ 테이블 생성 및 데이터 조작

• 데이터 삽입

- `conn = pymysql.connect(host="localhost", user="root", password="1234", database="exampledb")`
- `cursor = conn.cursor()`
- `# 데이터 삽입`
- `cursor.execute("INSERT INTO users (name, age, email) VALUES ('Alice', 25, 'alice@example.com');")`
- `conn.commit() # 변경 사항 저장`
- `print("데이터 삽입 완료")`
- `conn.close()`

4. 파이썬 DB 연동(2)

❖ 파이썬 DB 연동

■ 테이블 생성 및 데이터 조작

• 데이터 조회

- `conn = pymysql.connect(host="localhost", user="root", password="1234", database="exampledb")`
- `cursor = conn.cursor()`

- `# 데이터 조회`
- `cursor.execute("SELECT * FROM users")`
- `users = cursor.fetchall()`

- `for user in users:`
- `print(user)`

- `conn.close()`

4. 파이썬 DB 연동(2)

❖ 파이썬 DB 연동

■ 테이블 생성 및 데이터 조작

• 데이터 수정

- `conn = pymysql.connect(host="localhost", user="root", password="1234", database="exampledb")`
- `cursor = conn.cursor()`
- `# 사용자 정보 수정 (이름 변경)`
- `cursor.execute("UPDATE users SET name = %s WHERE id = %s", ("Bob", 1))`
- `conn.commit()`
- `print("데이터 수정 완료")`
- `conn.close()`

4. 파이썬 DB 연동(2)

❖ 파이썬 DB 연동

■ 테이블 생성 및 데이터 조작

• 데이터 삭제

- `conn = pymysql.connect(host="localhost", user="root", password="1234", database="exampledb")`
- `cursor = conn.cursor()`

- `# 특정 사용자 삭제`
- `cursor.execute("DELETE FROM users WHERE id = 3")`
- `conn.commit()`
- `print("데이터 삭제 완료")`

- `conn.close()`

❖ 과제

- 1. SQL 문장 코드 작성하기
- 2. 조인 코드 작성하기
- 3. MySQL을 사용하여 데이터베이스에 연결하고 테이블을 생성하기
- 4. MySQL을 사용하여 데이터를 삽입, 조회, 수정, 삭제 하기

❖ 다음 수업 내용

- GIT 형상 관리
 - GIT 개요
 - VSCode 연동 및 활용
 - 명령어 기반 GIT 활용
 - 버전 관리