

이화여대 공학관 강의실 **DB** 설계

과목	데이터베이스		
담당교수	김연정 교수님		
제출일	2024.06.07		
팀이름	팀5 - 공강		
조원 (학과/학번/이름)	컴퓨터공학과	2271007	강민서
	컴퓨터공학과	2276095	김희진
	컴퓨터공학과	2276114	박민서
	컴퓨터공학과	2276242	이주원
	컴퓨터공학과	2276309	조수현

[목차]

1. 선택한 기관의 데이터베이스 필요성 설명 및 요구 분석
2. ER 다이어그램과 간단한 설명
3. 데이터베이스 스키마 다이어그램
4. 자바 코드의 클래스(class)와 메서드(method)에 대한 설명
 - (1) 학생 코드
 - (2) 교수 코드
 - (3) 관리자 코드
5. 응용프로그램 설치 및 사용 방법, 연결을 위한 구성(configuration)에 대한 지시 사항
6. 제시한 17개의 요구 사항을 만족함을 보이는 자세한 설명
 - (1) 5개 이상의 테이블을 가지고 있어야 하고 각 테이블들의 컬럼(Attribute)의 수를 합하면 20개 이상이어야 한다.
 - (2) 초기화를 위해 적어도 30개의 레코드(투플)를 가지고 있어야 한다. (모든 테이블들의 레코 드 수의 총합이 30개 이상)
 - (3) 기본 키(primary key), 외래 키(foreign key), not null 제약조건(not null constraints)를 포함해야 한다
 - (4) 적어도 2개의 뷰를 정의해야 한다. (레포트에 view를 사용한 이유를 설명)
 - (5) 모든 테이블과 뷰의 이름들은 "DB2024_"라는 접두어를 가지고 있어야 한다.
 - (6) 적어도 4개의 인덱스를 정의해야 한다.
 - (7) 인덱스를 사용하는 쿼리들을 포함해야 한다.
 - (8) 뷰를 사용하는 쿼리를 포함해야 한다.
 - (9) 트랜잭션(transaction)을 포함해야 한다.
 - (10) 중첩된 쿼리(nested query)들을 가지는 쿼리들을 포함해야 한다.
 - (11) 조인 쿼리(join query)들을 가지는 쿼리들은 포함해야 한다.
 - (12) 매개 변수를 가지면서 동적으로 만드는 쿼리를 포함해야 한다.
 - (13) 그래픽 또는 문자 기반의 사용자 인터페이스를 사용해야 한다. 사용하기 쉽고 사용하기에 도움이 되는 정보를 가지고 있는 메뉴를 제공해야 한다.
 - (14) 데이터베이스에 삽입(insert)을 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.
 - (15) 데이터베이스에 갱신(update)을 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.
 - (16) 데이터베이스에 삭제(delete)를 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.
 - (17) 데이터베이스에 검색(select)을 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.
7. 요구조건 외의 팀만의 강점
8. 팀의 구성원의 담당 부분

1. 선택한 기관의 데이터베이스 필요성 설명 및 요구 분석

(1) 데이터베이스 필요성

이화여대 공대생으로서, 공강 시간에 팀 프로젝트, 스터디 등을 효율적으로 진행할 수 있는 장소를 찾는 것이 쉽지 않다. 그러나 강의실 위치나 빈 강의실 정보, 텀플이나 스터디 장소의 유무 등은 학업과 관련된 여러 활동을 원활하게 진행하는 데 중요한 요소이다. 따라서, 이화여대 공대의 강의실과 공간, 수업 등에 대한 정보를 체계적으로 관리하고 제공할 수 있는 데이터베이스 시스템의 필요하다. 그래서 우리 팀은 공대의 여러 강의실과 공간, 수업들을 데이터베이스 대상으로 선정하고, 이화여대 소속 학생과 교수님들께 필요한 강의실 정보를 제공하기 위한 DB 시스템을 설계하고 구현하기로 했다.

(2) 요구 분석

이를 통해 학생 사용자, 교수 사용자의 요구사항과 이를 관리하는 관리자의 요구사항을 분석했다.

[학생 사용자 요구사항]

- 강의실 검색
 - 휴식 및 취식, 회의, 공부 등의 목적에 따라 강의실을 검색할 수 있다.
- 교수님 검색
 - 교수님의 성함으로 교수님의 연구실 위치, 전화번호, 이메일, 담당 강의실 등을 검색할 수 있다.
 - 교수님의 성함을 모르는 경우에는 강의 이름이나 학수번호로 검색할 수 있다.
- 강의실 및 강의 검색
 - 강의의 학수번호를 입력하거나 강의실 이름, 강의 시간, 교수님 성함으로 검색하여 강의실의 이름과 위치 정보를 얻을 수 있다.

이를 통해 학생들은 공강 시간에 활용할 수 있는 자습실이나 스터디 공간을 쉽게 찾을 수 있게 된다. 또한 수업 장소를 찾는데 드는 시간을 절약할 수 있다. 이는 학생들의 학습 환경을 개선하고, 공강 시간을 보다 생산적으로 활용할 수 있도록 돕는다. 그리고 팀 프로젝트나 스터디를 위한 장소 정보를 제공함으로써, 학생들이 보다 원활하게 협업 활동을 진행할 수 있게 된다.

[교수 사용자 요구사항]

- 강의실 검색
 - 시험 공간 대여 및 수업을 위해 원하는 강의실을 좌석 수와 원하는 시간 등의 조건으로 검색할 수 있다.
- 자신이 수업하는 강의실 찾기
 - 교수님의 성함과 수업 유형(지금 수업, 오늘 수업, 전체 수업)을 선택하여 수업 시간, 수업 이름, 강의실 이름을 얻을 수 있다.
- 다른 교수님의 정보 검색
 - 강의실 위치와 시간을 입력하여 다른 교수님의 성함, 이메일, 전화번호, 연구실 위치정보를 얻을 수 있다.

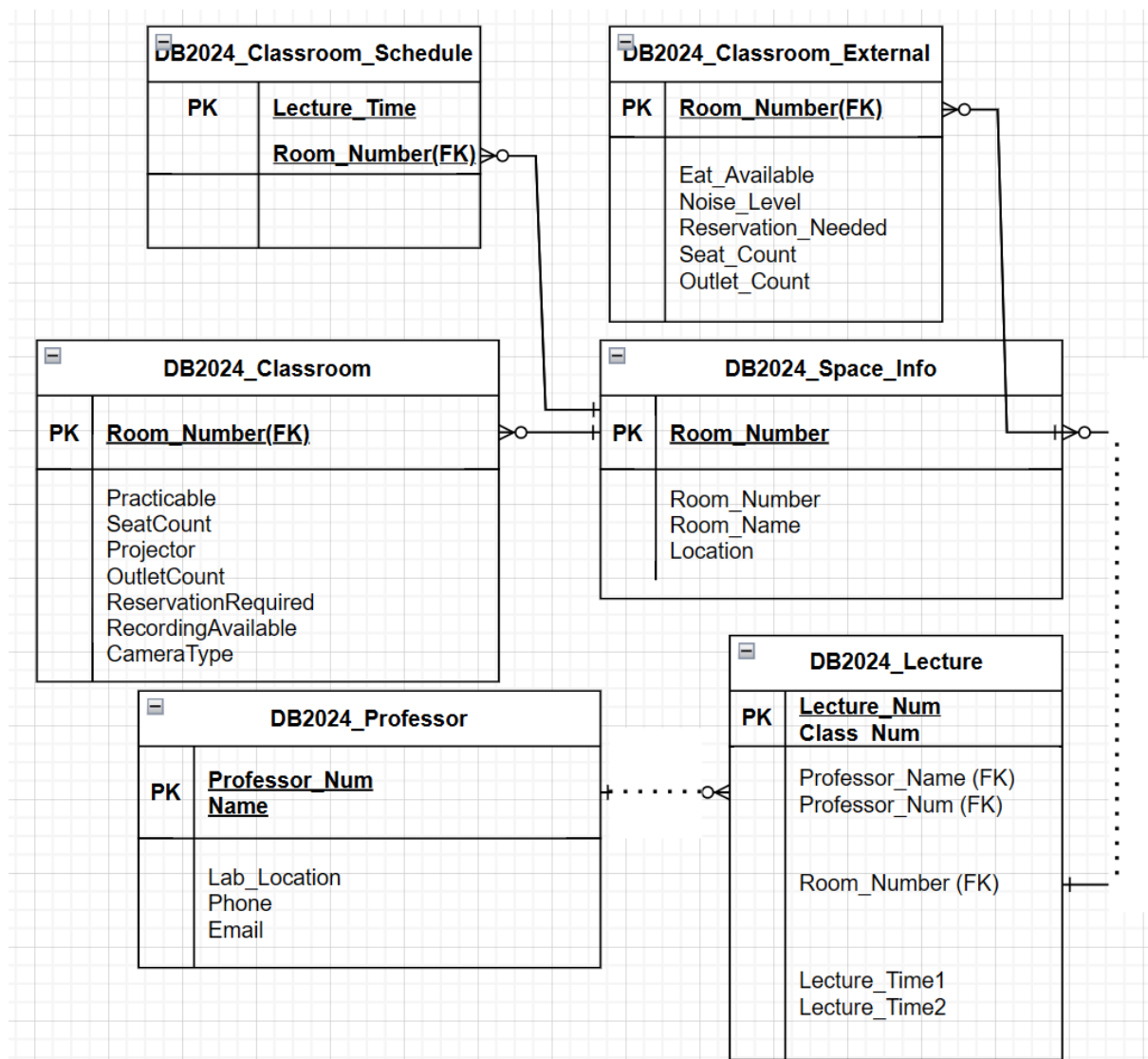
각 강의실의 위치, 자리 수, 설비 등 세부 정보를 제공함으로써, 교수님들은 강의 계획을 세울 때 필요한 정보를 쉽게 얻을 수 있다.

[관리자 요구사항]

- 강의 정보 등록 및 관리
 - 강의 정보를 등록한다.
 - 강의시간표, 강의 정보 등을 관리하고 운영한다.

이를 통해 학교 관리자는 강의실 배정 및 사용 현황을 체계적으로 관리할 수 있게 되며, 이를 통해 효율적인 공간 관리가 가능하다.

2. ER 다이어그램과 간단한 설명



객체로는 DB2024_Space_Info, DB2024_Classroom, DB2024_Classroom_External, DB2024_Professor, DB2024_Lecture, DB2024_Schedule가 있다.

DB2024_Space_Info와 DB2024_Classroom은 1:N의 관계로 DB2024_Classroom은 전체 참여하나 DB2024_Space_Info는 부분참여한다. DB2024_Classroom은 약한 개체로 DB2024_Space_Info의 키를 통해 식별된다.

DB2024_Space_Info와 DB2024_Classroom_Externa은 1:N의 관계로 DB2024_Classroom_Externa은 전체 참여하나 DB2024_Space_Info는 부분참여한다. DB2024_Classroom_Externa은 약한 개체로 DB2024_Space_Info의 키를 통해 식별된다.

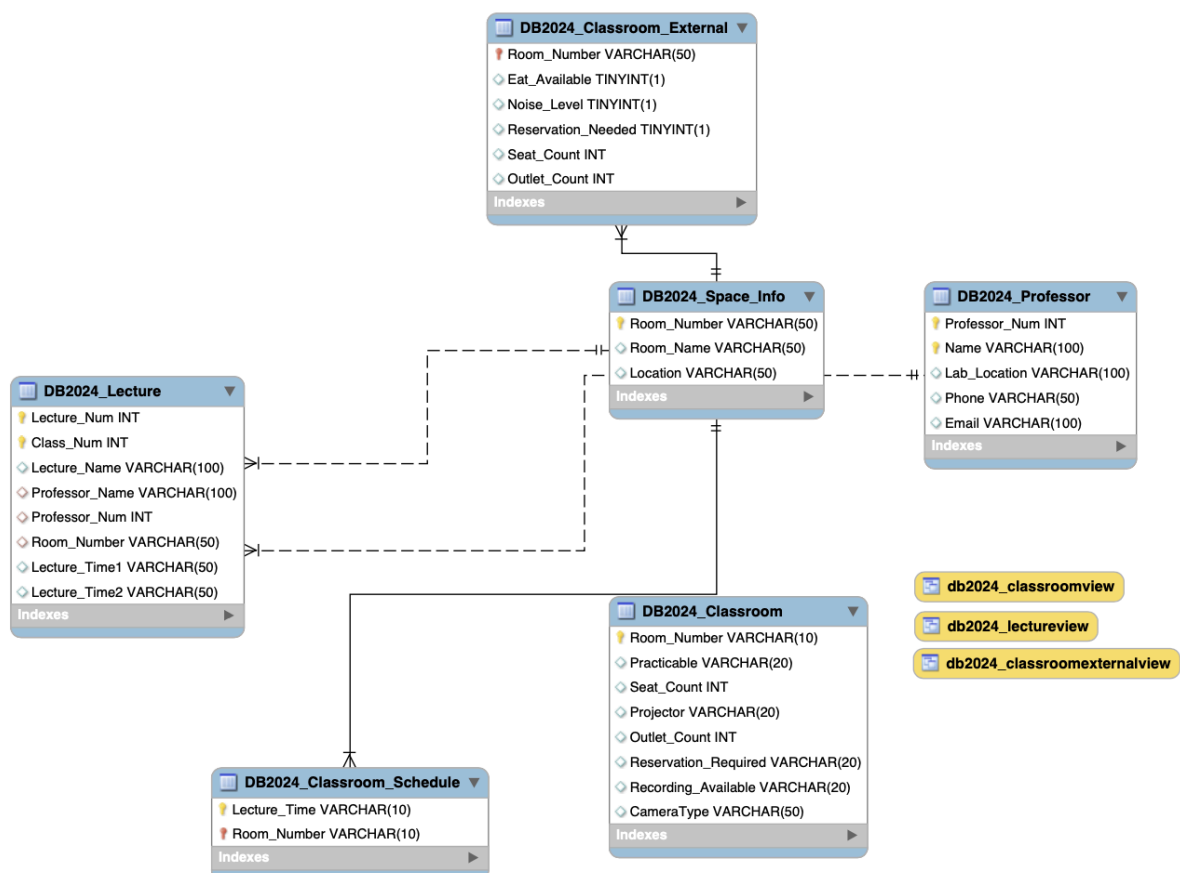
DB2024_Space_Info와 DB2024_Schedule은 1:N의 관계로 DB2024_Schedule은 전체 참여하나 DB2024_Space_Info는 부분참여한다. DB2024_Schedule은 약한 개체로 DB2024_Space_Info의 키를 통해 식별된다.

DB2024_Space_Info와 DB2024_Lecture은 1:N의 관계로 DB2024_Lecture은 전체 참여하나 DB2024_Space_Info는 부분참여한다.

DB2024_Professor와 DB2024_Lecture는 1:N의 관계로 DB2024_Lecture는 전체 참여하나 DB2024_Professor는 부분참여한다.

3. 데이터베이스 스키마 다이어그램

MySQL Workbench를 사용해서 데이터베이스 스키마 다이어그램을 작성했다.



4. 자바 코드의 클래스(class)와 메서드(method)에 대한 설명

(1) 학생 코드

<student_button1 클래스>

```
public class student_button1 extends JFrame{

    private JPanel contentPane;

    String usage;           // 사용 유형
    String seats;           // 좌석 수
    boolean content;        // 콘센트 유무
    boolean project;        // 빔프로젝트 유무
    boolean eat;            // 식사 가능 유무
    boolean computer;       // 컴퓨터 유무
    Map<String, Boolean> timeDictionary=new HashMap<>();
    private JTextArea infoArea; // 결과 가져오는 텍스트
    String message = "검색된 교실의 번호: \n";

    public student_button1() {
        // main frame 설정
        // main JPanel 생성
        // 로고 붙이는 Panel
        // GONG-GANG 로고 label 생성
        // - 원하는 강의실 찾기 - 로고 label 생성
        // option & time table 붙이는 mainPanel 생성
        //combobox와 checkbox 붙이는 subMainPanel
        //checkbox Panel
        //checkbox 생성
        //dropdown Panel 생성
        // ComboBox1 - 공간유형
        //찾고자하는 공간의 유형 선택
        // ComboBox2 - 좌석 수
        //좌석수를 JComboBox로 받아오기
        // 검색 원하는 시간 받아오기
        // 시간표 패널 생성
        // 시간표 라벨 및 체크박스 생성
        // resultButton & homeButton 붙이는 Panel 생성
        // result 버튼 붙이는 Panel 생성
        // 디자인을 위한 빈 패널 생성
        // result Button에 ActionListener 부착해 사용자 클릭시 함수 호출

        resultButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                //사용자가 선택한 항목들을 받아와 변수에 저장하는 코드
                //교실 정보를 선택한 경우
                if(usage.equals("교실")) searchClassroomInfo(seats, content, project, eat, computer);
                //교실 외의 정보를 선택한 경우
                if(usage.equals("교실 외")) searchClassroom_ExternalInfo(seats, content, project, eat,
computer);

                // 새 창을 여는 로직
                // 로고 붙이는 Panel
                // GONG-GANG 로고 label 생성
                //- 결과 - 로고 label 생성
            }
        });

        //homebutton
        // ActionListener 를 homeButton에 달아 홈버튼 클릭시 메인 화면으로 갈 수 있도록 함
    }

    //교실 정보를 찾는 함수
    private void searchClassroomInfo(String seats, boolean content, boolean project, boolean eat, boolean
```

```

computer) {
}

//교실 외 정보를 찾는 함수
private void searchClassroom_ExternalInfo(String seats, boolean content, boolean project, boolean eat,
boolean computer) {

}
//쿼리를 생성하는 코드
private String buildQuery(boolean content, boolean project, boolean eat, boolean computer, int type) {
}
//쿼리를 input으로 받아서 해당 쿼리를 실행하는 메소드 이후에 해당하는 쿼리를 실행한 결과를 받아 infoArea에
불이기
private void executeQuery(String seats, String query, int type) {
}
//원하는 시간표에 맞는지 해당 쿼리 결과를 확인하는 코드
private boolean processResultSet(String seats, ResultSet rs) throws SQLException {
}
//검색된 결과에서 Room_number를 가져오고 출력할 수 있도록 만들어주는 코드
private String formatRoomInfo(ResultSet rs) throws SQLException {
}
// 좌석 수를 사용자에게 범위로 받았기에 이를 테이블의 값과 비교하기 위한 코드
public static boolean isNumberInRange(String range, int number) {
}
}

```

1. searchClassroomInfo 메서드

```

//교실 정보를 찾는 함수
private void searchClassroomInfo(String seats, boolean content, boolean project, boolean eat, boolean
computer) {
    if (eat) {
        //교실에서 식사를 원할 경우 아래와 같은 문자를 화면에 보이게 한다.
        infoArea.setText("교실에서는 취식이 불가능합니다. 재선택 해주세요");
        return;
    }

    String query = buildQuery(content, project, eat, computer, 0); // 쿼리를 생성하는 함수를 실행하고
    executeQuery(seats, query, 0); // 해당 쿼리를 실행시키는 함수를 호출한다. 여기서 0은 교실외가 아니라 교실을
    선택했음을 알려주기 위한 숫자.
}

```

입력

- **seats** (String): 좌석 정보
- **content** (boolean): 콘텐츠 유무
- **project** (boolean): 프로젝트 가능 여부
- **eat** (boolean): 취식 가능 여부
- **computer** (boolean): 컴퓨터 사용 가능 여부

출력

- **infoArea.setText()**: 교실에서 취식이 불가능하다는 메시지 출력 (취식 가능 여부에 따라)
- **executeQuery()** 호출: 생성된 쿼리를 실행하여 교실 정보를 검색

핵심 기능

- 취식 여부 확인: 취식 불가능 시 메시지 출력 후 종료
- 쿼리 생성: 입력된 조건을 기반으로 쿼리 생성
- 쿼리 실행: 좌석 정보와 생성된 쿼리를 사용하여 교실 정보를 검색

2. searchClassroom_ExternalInfo 메서드

```
//교실 외 정보를 찾는 함수
private void searchClassroom_ExternalInfo(String seats, boolean content, boolean project, boolean eat,
boolean computer) {
    String warnMessage="";
    if (computer) {
        //컴퓨터를 사용하길 원하는 경우
        warnMessage+="컴퓨터 실습을 원한다면 교실을 선택해 주세요.";
        if(project)
            //빔 프로젝트도 사용하길 원하는 경우
            warnMessage+="\n빔프로젝트를 원한다면 교실을 선택해주세요.";
        infoArea.setText(warnMessage);
        return;
    } else if (project) {
        //빔 프로젝트를 원하는 경우
        warnMessage+="빔프로젝트 원한다면 교실을 선택해 주세요.";
        infoArea.setText(warnMessage);
        return;
    }
    String query = buildQuery(content, project, eat, computer, 1); //빔프로젝트나 컴퓨터 실습을 원하는 상황이
    아닌 경우 쿼리를 만드는 함수를 호출
    executeQuery(seats, query, 1); //쿼리 실행
}
```

입력

- **seats** (String): 좌석 정보
- **content** (boolean): 콘텐츠 유무
- **project** (boolean): 프로젝트 가능 여부
- **eat** (boolean): 취식 가능 여부
- **computer** (boolean): 컴퓨터 사용 가능 여부

출력

- **infoArea.setText()**: 교실에서 취식이 불가능하다는 메시지 출력 (취식 가능 여부에 따라)
- **executeQuery()**: 생성된 쿼리를 실행하여 교실 정보를 검색

핵심 기능

- 취식 여부 확인: **eat**가 **false**인 경우, 메시지 "교실에서 취식이 불가능합니다." 출력 후 메서드 종료.
- 쿼리 생성: 입력된 조건(**content**, **project**, **eat**, **computer**)을 기반으로 쿼리 생성.
- 쿼리 실행: **seats** 정보와 생성된 쿼리를 사용하여 **executeQuery** 메서드를 호출해 교실 정보를 검색.

3. buildQuery 메서드


```
//쿼리를 생성하는 코드
private String buildQuery(boolean content, boolean project, boolean eat, boolean computer, int type) {

    String query = "SELECT * FROM ";
    //type은 사용자가 찾기를 원하는 것이 교실(0)인지 교실 외(1)인지를 구분
    if(type==0){
        query+="DB2024_ClassroomView";
        if (computer || project) {
            //교실에서 컴퓨터와 빔프로젝트를 원하는 경우에는 조건을 붙여준다. 교실에서는 식사가 불가하고 모든
            //교실에 콘센트가 있으므로 해당하는 조건은 추가하지 않아도 됨
            query += " WHERE";
            if (computer) {
                query += " Practicable='실습가능'";
                if (project) {
                    query += " AND";
                }
            }
            if (project) {
                query += " Projector='빔 있음'";
            }
        }
    }

    if (type==1) {
        query += "DB2024_ClassroomExternalView";
        //교실 외에서는 콘센트 필요유무, 식사 가능 여부에 대해서 검색 조건을 걸어줄 수 있음.
        if(content) {
            query+="WHERE Outlet_Count > 0 ";
            if(eat) query+="AND Room_Number IN (SELECT Room_Number FROM DB2024_ClassroomExternalView WHERE
Eat_Available = 1)";
            query+="";
        }
        else{
            if(eat) query+="WHERE Eat_Available = True;";
        }
    }

    return query;
}
```

입력

- **content** (boolean): 콘텐츠 유무
- **project** (boolean): 프로젝트 가능 여부
- **eat** (boolean): 취식 가능 여부
- **computer** (boolean): 컴퓨터 사용 가능 여부
- **type** (int): 검색 유형 (교실: 0, 교실 외: 1)

출력

- **query** (String): 생성된 SQL 쿼리 문자열

핵심 기능

- 교실 쿼리 생성 (**type == 0**):
 - 기본 테이블 DB2024_ClassroomView에서 조회.
 - computer**나 **project**가 true인 경우 조건 추가:
 - computer**: Practicable='실습가능'

- ii. `project:Projector='빔 있음'`
- 교실 외 쿼리 생성 (`type == 1`):
 - c. 기본 테이블 `DB2024_ClassroomExternalView`에서 조회.
 - d. `content`가 `true`인 경우:
 - i. `Outlet_Count > 0` 조건 추가.
 - ii. `eat`가 `true`인 경우, `Eat_Available = 1` 조건 추가.
 - e. `content`가 `false`인 경우:
 - i. `eat`가 `true`인 경우, `Eat_Available = True` 조건 추가.

4. executeQuery 메서드

```
//쿼리를 input으로 받아서 해당 쿼리를 실행하는 메소드 이후에 해당하는 쿼리를 실행한 결과를 받아 infoArea에 붙이기
private void executeQuery(String seats, String query, int type) {
    final String url = "jdbc:mysql://localhost/DB2024Team05";
    final String user = "DB2024Team05";
    final String password = "DB2024Team05";

    try (Connection conn = DriverManager.getConnection(url, user, password);
        Statement stmt = conn.createStatement();
    ) {
        ResultSet rs = stmt.executeQuery(query);//연결 맺고 실행한 결과를 받아오기

        boolean found = false;
        while (rs.next()) {
            if (type==0 && processResultSet(seats, rs, conn)) {
                found = true;
            }else if((type == 1) && isNumberInRange(seats, rs.getInt("Seat_Count"))){
                message += formatRoomInfo(rs);
                found = true;
            }
        }

        infoArea.setText(found ? message : "원하는 교실이 없습니다. 조건을 재선택하세요.");//결과값이 없는 경우

    } catch (SQLException e) {
        infoArea.setText("데이터를 불러오는 과정에서 오류가 있습니다 \n" + e.getMessage());//DB 연결이 실패한
        경우
    }
}
```

5. processResultSet 메서드

```
private boolean processResultSet(String seats, ResultSet rs, Connection conn) throws SQLException {
    boolean hasFalse = false; // 각 시간대에 대해 결과가 false인지 확인하는 변수

    // 각 시간대에 대해 반복
    for (Map.Entry<String, Boolean> entry : timeDictionary.entrySet()) {
        String key = entry.getKey();
        Boolean value = entry.getValue();

        // 만약 현재 시간대에 대한 값이 true이고 좌석 수가 조건에 맞다면
        if (value && isNumberInRange(seats, rs.getInt("Seat_Count"))) {
```

```

String query2 = "SELECT * FROM DB2024_Classroom_Schedule WHERE Room_Number=? AND
Lecture_Time=?";

PreparedStatement stmt = conn.prepareStatement(query2);
stmt.setString(1, rs.getString("Room_Number"));
stmt.setString(2, key);

ResultSet rs2 = stmt.executeQuery();

// 현재 시간대에 대한 결과가 없는 경우
if (!rs2.next()) {
    // 결과가 없으면 hasFalse를 true로 설정
    message+=rs.getString("Room_Number");
    message+=" "+key+"가능\n";
    hasFalse=true;
} else {
    // 결과가 있으면 바로 false 반환
    //앞에서 한번만 true로 바꿔주면 있기는 한거니까 hasFalse를 다시 false로 바꿀 필요는 없음.
}
}
// 각 시간대에 대해 결과가 없는 경우에만 true 반환
return hasFalse;
}

```

입력

- **seats** (String): 좌석 수 조건
- **rs** (ResultSet): 첫 번째 쿼리 결과 집합
- **conn** (Connection): 데이터베이스 연결 객체

출력

- **hasFalse** (boolean): 각 시간대에 대해 결과가 없는 경우 **true**, 있는 경우 **false**

핵심 기능

- 초기화: **hasFalse** 변수를 **false**로 초기화.
- 시간대 반복: **timeDictionary**의 각 시간대를 반복하여 확인.
- 조건 확인:
현재 시간대(**value**가 **true**인 경우)와 좌석 수(**isNumberInRange**로 확인)가 조건에 맞는 경우:
 - 해당 시간대와 강의실 번호로 두 번째 쿼리(**DB2024_Classroom_Schedule**)를 실행.
 - Room_Number**와 **Lecture_Time**을 사용하여 쿼리 준비 및 실행.
- 결과 처리:
두 번째 쿼리 결과가 없는 경우 (**!rs2.next()**):
 - hasFalse**를 **true**로 설정.
 - message**에 강의실 번호와 가능한 시간대 추가.

두 번째 쿼리 결과가 있는 경우:

 - hasFalse** 값을 변경하지 않고 **false** 반환.

6. formatRoomInfo 메서드

```
//검색된 결과에서 Room_number를 가져오고 출력할 수 있도록 만들어주는 코드
private String formatRoomInfo(ResultSet rs) throws SQLException {
    return rs.getString("Room_number") + " 가능\n";
}
```

입력

- **rs (ResultSet)**: 데이터베이스 쿼리 결과 집합

출력

- **String**: 포맷된 강의실 정보 문자열

핵심 기능

- 강의실 번호 추출: **rs.getString("Room_number")**를 사용하여 **Room_number** 열에서 강의실 번호를 추출.
- 문자열 포맷: 강의실 번호에 " 가능\n" 문자열을 추가하여 반환.

7. isNumberInRange 메서드

```
// 좌석 수를 사용자에게 범위로 받았기에 이를 테이블의 값과 비교하기 위한 코드
public static boolean isNumberInRange(String range, int number) {
    String[] parts = range.split("-");

    int start = Integer.parseInt(parts[0].trim());
    int end = Integer.parseInt(parts[1].trim());

    return number >= start && number <= end;
}
```

입력

- **range (String)**: 사용자가 입력한 좌석 수 범위 (예: "10-20")
- **number (int)**: 비교할 좌석 수

출력

- **boolean**: 범위 내에 있으면 **true**, 아니면 **false**

핵심 기능

- 범위 분할:
range.split("-")를 사용하여 입력된 범위를 시작과 끝 부분으로 분할. 분할된 값을 **parts** 배열에 저장.
- 범위 값 변환:
Integer.parseInt(parts[0].trim())를 사용하여 시작 값을 정수로 변환하여 **start**에

저장.

`Integer.parseInt(parts[1].trim())`를 사용하여 끝 값을 정수로 변환하여 `end`에 저장.

- 범위 확인:

`number`가 `start` 이상이고 `end` 이하인 경우 `true`를 반환.

그렇지 않으면 `false`를 반환.

<student_button2 클래스>

```
public student_button2() {
    // main frame 설정
    // main JPanel 생성
    // 로고 붙이는 Panel
    // GONG-GANG 로고 label 생성
    // inputPanel & resultPanel 포함하는 mainPanel
    // 입력 받는 inputPanel
    // 학수번호나 강의이름과 분반 번호를 입력하세요' Label 생성
    // 결과 보여주는 TextField 생성
    // 결과 출력을 위한 searchButton 생성
    //입력후 search 버튼을 눌렀을 경우
    searchButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            String lectureNumber = lectureNumberField.getText();//텍스트를 받아온 뒤에
            if (!lectureNumber.isEmpty()) {
                showRoomNumber(lectureNumber);//해당 강의의 공간번호를 보여주는 함수 호출
            } else {
                JOptionPane.showMessageDialog(null, "학수 번호를 입력하세요.");
            }
        }
    });
    //JText Area로 결과 표시
    //home button
    // Add ActionListener to homeButton }
```

1. showRoomNumber 메서드

```
//강의번호나 강의이름에 해당하는 공간번호를 보여주는 코드
private void showRoomNumber(String lectureNumber) {
    String dbUrl = "jdbc:mysql://localhost/DB2024Team05";
    String dbUser = "DB2024Team05";
    String dbPass = "DB2024Team05";

    // 학수번호와 함께 받아와서 분리
    String[] parts = lectureNumber.split("-");
    if (parts.length != 2) {
        JOptionPane.showMessageDialog(this, "입력값을 확인하세요");
        return; // 입력이 올바르게 분리되지 않은 경우
    }

    // 서브 쿼리를 사용하여 해당 강의실의 추가적인 특성 정보를 조회하는 쿼리
    String query = "SELECT * FROM DB2024_ClassroomView " +
        "WHERE Room_Number = (SELECT Room_Number FROM DB2024_LectureView " +
        "if (isInteger(parts[0])) {
            query += "WHERE Lecture_Num = ? AND Class_Num = ?)";
        } else {
            query += "WHERE Lecture_Name = ? AND Class_Num = ?)";
        }
    }
```

```

try (Connection conn = DriverManager.getConnection(dbUrl, dbUser, dbPass);
    PreparedStatement stmt = conn.prepareStatement(query)) {
    stmt.setString(1, parts[0]); // Lecture_Num or Lecture_Name
    stmt.setString(2, parts[1]); // Class_Num

    try (ResultSet rs = stmt.executeQuery()) {
        if (rs.next()) {
            String roomNumber = rs.getString("Room_Number");
            String roomName = rs.getString("Room_Name");
            String location = rs.getString("Location");
            int seatCount = rs.getInt("Seat_Count");
            String projector = rs.getString("Projector");
            String reservationRequired = rs.getString("Reservation_Required");
            int outletCount = rs.getInt("Outlet_Count");

            if (isInteger(parts[0])) {
                infoArea.setText("강의실 번호: " + roomNumber + " \n강의실 이름: " + roomName +
                    "\n강의실 위치: " + location +
                    "\n좌석 수: " + seatCount +
                    "\n프로젝터: " + projector +
                    "\n예약 필요: " + reservationRequired +
                    "\n콘센트 개수: " + outletCount);
            } else {
                infoArea.setText("강의실 번호: " + roomNumber +
                    "\n강의실 이름: " + roomName +
                    "\n강의실 위치: " + location +
                    "\n좌석 수: " + seatCount +
                    "\n프로젝터: " + projector +
                    "\n예약 필요: " + reservationRequired +
                    "\n콘센트 개수: " + outletCount);
            }

        } else {
            JOptionPane.showMessageDialog(this, "해당 강의실의 추가 정보를 찾을 수 없습니다.");
        }
    }
} catch (SQLException ex) {
    ex.printStackTrace();
    JOptionPane.showMessageDialog(this, "Database error: " + ex.getMessage());
}
}

```

입력

lectureNumber: 사용자로부터 입력받은 문자열, 강의 번호와 분반 번호를 포함. 예: "20471-3" 또는 "데이터베이스-3".

출력

- 정보 출력: 강의실의 상세 정보(번호, 이름, 위치, 좌석 수 등)를 사용자 인터페이스에 표시.
- 오류 메시지: 입력 오류 또는 데이터베이스 문제가 있을 경우 사용자에게 오류 메시지를 제공.

핵심 기능

- 입력 검증: 사용자 입력을 검사하여 형식이 올바른지 확인.
- 중첩 쿼리 실행: 서브 쿼리를 사용하여 강의 번호나 이름에 따른 강의실 번호를 찾고, 이를 기반으로 강의실의 상세 정보를 조회.
- 결과 표시: 조회된 정보를 화면에 출력하거나, 정보가 없을 경우 오류 메시지를 표시.

2. isInteger 메서드

```
//학수번호를 입력한 것인지 확인하기 위한 문자열의 숫자 구성 판단 함수
public boolean isInteger(String strValue){
    try {
        Integer.parseInt(strValue);
        return true;
    } catch (NumberFormatException ex) {
        return false;
    }
}
```

입력

- **strValue** (String): 판단할 문자열

출력

- **boolean**: 문자열이 숫자 구성인지 여부 (**true**이면 숫자, **false**이면 숫자가 아님)

핵심 기능

- 문자열 숫자 구성 판단:
 - **Integer.parseInt(strValue)**를 사용하여 문자열을 정수로 변환 시도.
 - 변환이 성공하면 **true** 반환.
 - **NumberFormatException**이 발생하면 **false** 반환.

<student_button3 클래스>

```
public class student_button3 extends JFrame{

    public student_button3() {

        // main frame 설정
        // main Panel 생성 및 설정
        // 로고 붙이는 Panel
        // GONG-GANG 로고 label 생성
        // - 교수님 찾기 - 로고 label 생성 및 설정
        //inputPanel & Info display area 붙이는 mainPanel
        // Input Panel
        // '교수님 이름을 입력하세요' label 생성 및 설정
        // 검색 버튼 생성
        searchButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (!nameField.getText().trim().isEmpty()) {
                    connectToDatabase(); // 데이터베이스 연결
                    searchProfessorInfo(nameField.getText().trim());
                    searchProfessor();
                } else {
                    JOptionPane.showMessageDialog(null, "교수님 이름을 입력하세요.");
                }
            }
        });
        //output Panel
        // Info display area
        // 결과 영역 (강의 테이블)
        //home button 붙이는 Panel 생성 및 설정
        //home button 생성 및 설정
        // 디자인을 위한 empty Panel
```

```
// Add ActionListener to homeButton

}
```

1. searchProfessorInfo 메서드

```
private void searchProfessorInfo(String name) {

    //JDBC driver name and database URL
    final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    final String url = "jdbc:mysql://localhost/DB2024Team05";
    //Database credentials
    // MySQL 계정과 암호 입력
    final String user = "DB2024Team05";
    final String password = "DB2024Team05";

    String query = "SELECT Email, Lab_Location, Phone FROM DB2024_Professor WHERE Name = ?";
    try (Connection conn = DriverManager.getConnection(url, user, password);
        PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, name);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            String email = rs.getString("Email");
            String Lab_Location = rs.getString("Lab_Location");
            String Phone = rs.getString("Phone");
            infoArea.setText("Email: " + email + "\nLab_Location: " + Lab_Location + "\nPhone Number: "
+ Phone);
        } else {
            infoArea.setText("No information found.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
        infoArea.setText("Error retrieving data.");
    }
}
```

입력

- **name (String):** 검색할 교수의 이름

출력

- 정보 출력: 검색된 교수의 이메일, 연구실 위치, 전화번호를 화면에 표시
- 오류 메시지 또는 정보 부재: 해당 교수의 정보가 데이터베이스에 없는 경우 "No information found." 메시지를 출력하고, 데이터베이스 접근 중 오류가 발생한 경우 "Error retrieving data." 메시지를 출력

핵심 기능

- 데이터베이스 연결 설정: **JDBC** 드라이버와 데이터베이스 **URL**, 사용자 이름, 비밀번호를 사용하여 데이터베이스 연결을 설정
- **SQL** 쿼리 실행: 준비된 **SQL 쿼리(PreparedStatement)**를 사용하여 입력받은 이름에 해당하는 교수의 정보를 **DB2024_Professor** 테이블에서 조회

- 결과 처리 및 출력: 조회 결과를 받아, 해당 결과가 존재하면 정보를 화면에 출력하고, 결과가 없으면 적절한 메시지를 화면에 표시

2. connectToDatabase 메서드

```
//DB연동 준비하는 코드
private void connectToDatabase() {
    try {
        Class.forName("com.mysql.jdbc.Driver"); // MySQL 드라이버 로드
        connection = DriverManager.getConnection("jdbc:mysql://localhost/DB2024Team05", "DB2024Team05",
"DB2024Team05"); // 데이터베이스 연결
        System.out.println("Database connected successfully.");
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Database connection failed: " + e.getMessage());
    }
}
```

입력

- **name (String)**: 교수님의 이름

출력

- 교수님의 이메일, 연구실 위치, 전화번호 정보를 **infoArea**에 출력

핵심 기능

- 데이터베이스 연결 설정: **JDBC** 드라이버와 데이터베이스 **URL**, 사용자 인증 정보를 설정.
- 쿼리 준비: 입력된 이름을 기준으로 교수님의 이메일, 연구실 위치, 전화번호를 검색하는 쿼리를 준비.
- 쿼리 실행:
데이터베이스에 연결하고 준비된 쿼리를 실행.
PreparedStatement를 사용하여 사용자 입력값을 쿼리에 설정.
- 결과 처리:
ResultSet에서 결과를 가져와 이메일, 연구실 위치, 전화번호를 **infoArea**에 출력.
결과가 없는 경우, "No information found." 메시지를 출력.
- 예외 처리:
데이터베이스 연결 또는 쿼리 실행 중 발생하는 예외를 처리하고, 오류 메시지를 **infoArea**에 출력.

3. searchProfessor 메서드

```
//사용자에게 입력받은 교수님의 성함을 찾고 테이블을 업데이트 하는 코드
private void searchProfessor() {
    String professorName = nameField.getText().trim(); // 검색어 가져오기
    if (professorName.isEmpty()) {
        JOptionPane.showMessageDialog(this, "교수님 이름을 입력하세요.", "경고", JOptionPane.WARNING_MESSAGE);
    } // 검색어 없을 시 경고 메시지
    return;
}

try {
```

```

ProfessorInfo professorInfo = getProfessorInfo(professorName); // 교수 정보 가져오기
if (professorInfo != null) {

    updateLectureTable(professorName); // 테이블 업데이트
} else {
    JOptionPane.showMessageDialog(this, "해당 교수님의 정보를 찾을 수 없습니다.", "경고",
JOptionPane.WARNING_MESSAGE); // 교수

}
} catch (SQLException e) {
    e.printStackTrace(); // SQL 예외 발생 시 스택 트레이스 출력
}
}
}

```

입력

- 없음 (사용자로부터 교수님의 이름을 **nameField** 텍스트 필드에서 가져옴)

출력

- 없음 (교수님의 정보를 기반으로 테이블을 업데이트하거나 경고 메시지 출력)

핵심 기능

- 교수님 이름 가져오기: **nameField** 텍스트 필드에서 교수님의 이름을 가져와 공백을 제거.
- 유효성 검사: 입력된 이름이 비어있는지 확인하고, 비어있으면 경고 메시지 출력 후 메서드 종료.
- 교수 정보 검색:
getProfessorInfo 메서드를 호출하여 교수님의 정보를 검색.
 검색된 정보가 존재하면 테이블을 업데이트 (**updateLectureTable** 메서드 호출).
 검색된 정보가 없으면 경고 메시지 출력.
- 예외 처리: SQL 예외 발생 시 스택 트레이스 출력.

4. getProfessorInfo 메서드

```

//교수님 성함을 입력받아 DB에서 검색한 다음 검색되면 관련정보를 professorInfo에 보여주기
private ProfessorInfo getProfessorInfo(String professorName) throws SQLException {
    ProfessorInfo professorInfo = null;

    String professorQuery = "SELECT Name, Lab_Location, Phone, Email FROM DB2024_Professor WHERE Name = ?";
    // 교수 정보
    // 쿼리
    PreparedStatement professorStmt = connection.prepareStatement(professorQuery); // 쿼리 준비
    professorStmt.setString(1, professorName); // 검색어 설정
    ResultSet professorRs = professorStmt.executeQuery(); // 쿼리 실행

    if (professorRs.next()) {
        professorInfo = new ProfessorInfo();
        professorInfo.setName(professorRs.getString("Name")); // 이름 설정
    }

    return professorInfo; // 교수 정보 반환
}

```

입력

- **professorName** (String): 교수님의 이름

출력

- **ProfessorInfo**: 검색된 교수님의 정보를 담은 객체, 검색되지 않으면 **null**

핵심 기능

- 초기화: **ProfessorInfo** 객체를 **null**로 초기화.
- 쿼리 준비 및 실행:
교수님의 정보를 검색하는 **SQL** 쿼리 문자열 준비.
PreparedStatement를 사용하여 쿼리를 준비하고, 입력받은 교수님의 이름을 쿼리에 설정.
쿼리를 실행하여 결과를 **ResultSet**으로 받음.
- 결과 처리:
ResultSet에서 결과가 있는 경우, **ProfessorInfo** 객체를 생성하고, 검색된 정보를 객체에 설정.
설정된 **ProfessorInfo** 객체를 반환.
- 예외 처리: **SQL** 예외가 발생하면 메서드를 호출한 곳에서 처리.

5. updateLectureTable 메서드

```
//시간표 모양의 테이블에 교수님의 강의하고 계시는 위치를 표시
private void updateLectureTable(String professorName) {
    // 테이블 초기화

    for (int i = 0; i < 9; i++) {
        for (int j = 1; j < 6; j++) {

            lectureTable.setValueAt("", i, j);
            // 테이블 셀 초기화
        }
    }

    try {
        String lectureQuery = "SELECT Room_Number, Lecture_Time1, Lecture_Time2 FROM DB2024_Lecture WHERE
Professor_Name = ?";
        PreparedStatement lectureStmt = connection.prepareStatement(lectureQuery);
        lectureStmt.setString(1, professorName);
        ResultSet lectureRs = lectureStmt.executeQuery();
        //강의테이블에서 교수님이 수업하시는 강의 정보를 받아와서 강의시간과 위치를 알아온 뒤 이를 시간표 테이블에
        입력해준다.
        while (lectureRs.next()) {
            String RoomNumber = lectureRs.getString("Room_Number");

            String lectureTime1 = lectureRs.getString("Lecture_Time1");
            String lectureTime2 = lectureRs.getString("Lecture_Time2");

            int dayIndex = getDayIndex(lectureTime1);
            if (dayIndex != -1) {
                int period = Integer.parseInt(lectureTime1.substring(1)) - 1;
                lectureTable.setValueAt(RoomNumber, period, dayIndex); //해당하는 시간에 강의실 번호 입력
            }

            if (lectureTime2 != null) {
                dayIndex = getDayIndex(lectureTime2);
                if (dayIndex != -1) {
                    int period = Integer.parseInt(lectureTime2.substring(1)) - 1;
                    lectureTable.setValueAt(RoomNumber, period, dayIndex);
                }
            }
        }
    }
}
```

```

    }
} catch (SQLException e) {
    e.printStackTrace();
}

// 현재 요일과 교시 강조 표시
highlightCurrentDayAndPeriod();
}

```

입력

- **professorName** (String): 교수님의 이름

출력

- 없음 (교수님의 강의 위치를 시간표 모양의 테이블에 표시)

핵심 기능

- 테이블 초기화: 9행 6열의 테이블을 순회하며 각 셀을 빈 문자열로 초기화.
- 강의 정보 검색:
입력된 교수님의 이름을 기준으로 강의실 번호와 강의 시간을 검색하는 **SQL** 쿼리를 준비.
PreparedStatement를 사용하여 쿼리를 준비하고, 교수님의 이름을 쿼리에 설정.
쿼리를 실행하여 결과를 **ResultSet**으로 받음.
- 결과 처리 및 테이블 업데이트:
ResultSet을 순회하며 각 강의 정보를 테이블에 입력.
Lecture_Time1과 **Lecture_Time2**에 따라 해당 시간대와 요일의 셀에 강의실 번호를 입력.
- 현재 요일과 교시 강조 표시:
highlightCurrentDayAndPeriod 메서드를 호출하여 현재 요일과 교시를 강조.

6. **getDayIndex** 메서드

```

//날짜 인덱스 만드는 함수
private int getDayIndex(String time) {
    switch (time.charAt(0)) {
        case '월':
            return 1;
        case '화':
            return 2;
        case '수':
            return 3;
        case '목':
            return 4;
        case '금':
            return 5;
        default:
            return -1;
    }
}

```

입력

- **time** (String): 요일을 포함한 문자열 (예: "월1", "화2")

출력

- **int**: 요일에 해당하는 인덱스 (월요일: 1, 화요일: 2, 수요일: 3, 목요일: 4, 금요일: 5, 잘못된 값: -1)

핵심 기능

- 요일 인덱스 반환:
time 문자열의 첫 번째 문자를 검사하여 해당 요일의 인덱스를 반환.
월, 화, 수, 목, 금에 따라 각각 1, 2, 3, 4, 5를 반환.
잘못된 값이 입력된 경우 -1을 반환.

7. highlightCurrentDayAndPeriod 메서드

```
private void highlightCurrentDayAndPeriod() {
    Calendar calendar = Calendar.getInstance(); // 현재 날짜와 시간을 가져오는 캘린더 인스턴스 생성
    int dayOfWeek = calendar.get(Calendar.DAY_OF_WEEK); // 현재 요일 가져오기
    int hourOfDay = calendar.get(Calendar.HOUR_OF_DAY); // 현재 시간 가져오기

    int currentDayIndex = -1; // 현재 요일 인덱스 초기화
    switch (dayOfWeek) { // 요일에 따른 인덱스 설정
        case Calendar.MONDAY:
            currentDayIndex = 1;
            break;
        case Calendar.TUESDAY:
            currentDayIndex = 2;
            break;
        case Calendar.WEDNESDAY:
            currentDayIndex = 3;
            break;
        case Calendar.THURSDAY:
            currentDayIndex = 4;
            break;
        case Calendar.FRIDAY:
            currentDayIndex = 5;
            break;
    }

    int currentPeriod = -1; // 현재 교시 인덱스 초기화
    if (hourOfDay >= 8 && hourOfDay < 21) { // 8시부터 21시까지 시간 범위 확인
        currentPeriod = (int) Math.floor((hourOfDay - 8) / 1.5); // 8시부터 1.5시간 단위 교시 계산
    }

    // 새로운 렌더러 적용
    for (int i = 0; i < lectureTable.getRowCount(); i++) {
        for (int j = 0; j < lectureTable.getColumnCount(); j++) {
            lectureTable.getColumnModel().getColumn(j).setCellRenderer(new DefaultTableCellRenderer()); // 기본 셀 렌더러로 초기화
        }
    }

    if (currentDayIndex != -1) { // 현재 요일이 유효한 경우
        lectureTable.getColumnModel().getColumn(currentDayIndex)
            .setCellRenderer(new CustomRenderer(Color.PINK, -1)); // 현재 요일 컬럼 강조
    }

    if (currentPeriod != -1 && currentDayIndex != -1) { // 현재 교시와 요일이 유효한 경우
        lectureTable.getColumnModel().getColumn(currentDayIndex)
            .setCellRenderer(new CustomRenderer(Color.PINK, currentPeriod));
        lectureTable.getColumnModel().getColumn(currentDayIndex)
```

```

        .setCellRenderer(new CustomRenderer(Color.RED, currentPeriod, currentDayIndex)); // 현재
교시와 요일 셀 강조
    }

    if (currentDayIndex != -1 && currentPeriod != -1) {
        for (int i = 0; i < lectureTable.getColumnCount(); i++) {
            if (i == currentDayIndex) {
                lectureTable.getColumnModel().getColumn(i).setCellRenderer(new CustomRenderer(Color.RED,
currentPeriod));
            } else {
                lectureTable.getColumnModel().getColumn(i).setCellRenderer(new DefaultTableCellRenderer());
            }
        }
    }
    lectureTable.repaint();

    lectureTable.repaint(); // 테이블 다시 그리기
}

```

입력

- 없음

출력

- 없음 (현재 요일과 교시를 테이블에 강조 표시)

핵심 기능

- 현재 날짜와 시간 가져오기:
Calendar 인스턴스를 생성하여 현재 날짜와 시간을 가져옴.
현재 요일과 시간을 `dayOfWeek`와 `hourOfDay` 변수에 저장.
- 현재 요일 인덱스 설정:
`dayOfWeek` 값을 기반으로 현재 요일 인덱스를 `currentDayIndex`에 설정 (월요일부터
금요일까지).
- 현재 교시 인덱스 설정:
`hourOfDay` 값을 기반으로 현재 교시를 `currentPeriod`에 설정.
8시부터 21시 사이의 시간을 1.5시간 단위로 계산하여 교시 인덱스 설정.
- 테이블 셀 렌더러 초기화:
모든 셀의 렌더러를 기본 `DefaultTableCellRenderer`로 초기화.
- 현재 요일 및 교시 강조 표시:
`currentDayIndex`가 유효한 경우 해당 요일 컬럼을 강조 (`Color.PINK`).
`currentPeriod`와 `currentDayIndex`가 유효한 경우 해당 교시와 요일 셀을 강조
(`Color.RED`).
- 테이블 다시 그리기:
`lectureTable.repaint()`를 호출하여 테이블을 다시 그림.

8. CustomRenderer 클래스

```

public class CustomRenderer extends DefaultTableCellRenderer {
    private Color backgroundColor; // 배경색
    private int highlightedRow = -1; // 강조할 행
}

```

```

private int highlightedColumn = -1; // 강조할 열

public CustomRenderer(Color backgroundColor) {
    this.backgroundColor = backgroundColor;
}

public CustomRenderer(Color backgroundColor, int highlightedRow) {
    this.backgroundColor = backgroundColor;
    this.highlightedRow = highlightedRow;
}

public CustomRenderer(Color backgroundColor, int highlightedRow, int highlightedColumn) {
    this.backgroundColor = backgroundColor;
    this.highlightedRow = highlightedRow;
    this.highlightedColumn = highlightedColumn;
}

@Override
public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean
hasFocus,
                                         int row, int column) {
    Component c = super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);
    if ((highlightedRow == -1 || row == highlightedRow)
        && (highlightedColumn == -1 || column == highlightedColumn)) {
        c.setBackground(backgroundColor); // 강조된 행과 열의 셀 배경색 설정
    } else if (column == highlightedColumn) {
        c.setBackground(Color.PINK); // 강조된 열의 셀 배경색 설정
    } else {
        c.setBackground(Color.WHITE); // 기본 셀 배경색 설정
    }
    return c;
}
}

```

`CustomRenderer` 클래스는 `DefaultTableCellRenderer`를 상속받아 특정 행, 열 또는 둘 다 강조 표시하기 위해 셀의 배경색을 커스터마이징 했다.

- 필드: `backgroundColor`(강조 색상), `highlightedRow`(강조 행), `highlightedColumn`(강조 열).
- 생성자: 강조할 배경색과 선택적으로 강조할 행 및 열을 설정.
- 메서드: `getTableCellRendererComponent`는 강조할 셀의 배경색을 설정하며, 기본 셀 배경색은 흰색으로 설정.

9. ProfessorInfo 클래스

```

class ProfessorInfo {
    private String name; // 교수 이름

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

ProfessorInfo 클래스는 교수님의 정보를 저장하기 위한 간단한 데이터 클래스 이다.

- 1) 필드: **name** (교수 이름).
- 2) 메서드:
 - **getName()**: 교수 이름을 반환.
 - **setName(String name)**: 교수 이름을 설정.

10. Lecture 클래스

```
class Lecture {
    private String lectureName; // 강의 이름
    private String lectureTime1; // 강의 시간1
    private String lectureTime2; // 강의 시간2

    public String getLectureName() {
        return lectureName;
    }

    public void setLectureName(String lectureName) {
        this.lectureName = lectureName;
    }

    public String getLectureTime1() {
        return lectureTime1;
    }

    public void setLectureTime1(String lectureTime1) {
        this.lectureTime1 = lectureTime1;
    }

    public String getLectureTime2() {
        return lectureTime2;
    }

    public void setLectureTime2(String lectureTime2) {
        this.lectureTime2 = lectureTime2;
    }
}
```

Lecture 클래스는 강의의 정보를 저장하기 위한 데이터 클래스

- 1) 필드: **lectureName** (강의 이름), **lectureTime1** (강의 시간1), **lectureTime2** (강의 시간2).
- 2) 메서드:
 - **getLectureName()**: 강의 이름을 반환.
 - **setLectureName(String lectureName)**: 강의 이름을 설정.
 - **getLectureTime1()**: 강의 시간1을 반환.
 - **setLectureTime1(String lectureTime1)**: 강의 시간1을 설정.
 - **getLectureTime2()**: 강의 시간2를 반환.
 - **setLectureTime2(String lectureTime2)**: 강의 시간2를 설정.

(2) 교수님 코드

<professor_button1 클래스>

이 클래스는 교수님 사용자를 대상으로 원하는 강의실을 찾는 기능을 제공한다.

- usage(공간 유형: 교실/교실 외),
- seats(좌석 수 : 10 단위로 구분됨),
- cameraType(카메라 유형: 고정식/추적식 카메라),
- content(콘센트 유무),
- project(빔 프로젝터 유무),
- reservation(예약 필요 여부),
- recording(녹화 가능 여부),
- practicable(실습 가능 여부)

위 변수들을 설정하여 교수님 사용자가 필요로 할 만한 조건들을 설정했다. 이 변수들을 통해 사용자가 입력한 조건에 맞는 교실/교실 외 공간들을 쿼리문을 통해 **sql**에서 검색해 온다.

1. searchClassroomInfo 메서드

좌석수, 카메라 타입, 예약 필요 여부 등 사용자가 입력한 강의실 조건에 따라 검색된 '교실' 을 출력하는 메소드이다.

2. searchClassroom_ExternalInfo 메서드

사용자가 녹화 가능/빔프로젝터/실습 가능 여부 와 함께 교실 외 를 선택하면 에러 메시지를 출력하는 과정이 포함되어 있다. (해당 기능은 교실에서만 가능하기 때문에)

3. buildQuery 메서드

sql에서 조건에 따른 Room_Number를 가져올 수 있도록 검색을 위한 query를 만든 부분으로, type을 통해 교실 (type==0), 교실 외 (type==1) 강의실에 따라 조건을 달리하여 검색이 가능하도록 한다.

- 교실의 경우 : ClassroomView 테이블에서 조건에 따라 검색을 실행한 후 해당되는 교실 번호를 가져온다.
- 교실 외의 경우 : ClassroomExternalView 에서 콘센트 유무 (개수가 1이상인) 에 따라 검색 실행 후 해당되는 교실의 번호를 가져온다.

4. executeQuery 메서드

buildquery에서 만들어진 query들이 실행되는 메서드

- 데이터베이스에 연결
- 주어진 조건에 따라 SQL 쿼리 실행 : PreparedStatement 객체를 생성해서 buildQuery 메소드에서 반환된 쿼리가 실행된다.
- 결과 집합(ResultSet)에서 각 강의실의 정보를 검사 : type에 따라 (0인 경우 : processResultSet 메소드 호출해서 교실 조건 확인, 1인 경우 : IsNumberRange 메소드 호출해서 교실 외 공간 조건 확인)
- 조건에 부합하고 사용자가 선택한 시간에 강의가 없는 강의실을 메시지로 출력

5. processResultSet 메서드

교실('DB2024_ClassroomView')의 강의실 정보 확인, 사용자가 선택한 시간에 강의가 없는지 확인하는 메서드.

- 좌석 수 확인 : isNumberInRange 메소드 활용해서 현 강의실 좌석 수가 유저가 선택한 좌석 수 범위에 있는지 확인하고 맞지 않을 시 false 반환
- 강의실 번호 가져오기 : 도출된 결과 집합해서 Room_Number를 가져온다.
- 시간표 조회 : 2번에서 가져온 강의실 번호를 사용해서 해당 강의실의 시간표를 조회하는 쿼리를 준비, 시행
- 사용자가 선택한 시간과 비교 : 시간표 결과를 반복해 각 시간표 항목의 'Lecture_Time'과 유저가 선택한 시간 ('TimeDictionary')를 비교하고, 유저가 선택한 시간에 강의가 있는 경우 'false'반환하고, 모든 조건을 만족하면 'true'를 반환하게 된다.

6. processExternalRoom 메서드

교실 외(DB2024_ClassroomExternalView)의 강의실 정보를 확인하고, 사용자가 선택한 시간에 강의가 없는지 확인한다. 위에서 설명한 processResultSet 메소드와 동일한 절차를 따른다.

7. formatRoomInfo 메서드

'ResultSet' 에서 강의실 번호를 가져와서 출력 가능한 형식(Room_Number+"가능")으로 반환하는 메서드

8. isNumberInRange 메서드

좌석 수가 유저가 입력한 범위 내에 있는지 확인하는 메서드

- range 문자열을 '-'로 분리하여 시작&종료 값을 추출한다
- 'number'가 시작 값 이상이고, 종료 값 이하인지 확인한다.

<professor_button2 클래스>

교수님 유저가 자신의 강의실을 찾는 기능을 구현하는 자바 코드

1. **connectToDatabase** 메서드

MySQL 데이터베이스에 연결하는데 사용되는 메서드

- MySQL 드라이버 로드: **Class.forName**을 사용하여 MySQL JDBC 드라이버를 로드한다.
- 데이터베이스 연결: **DriverManager.getConnection**을 사용하여 지정된 URL, 사용자명 및 비밀번호를 사용하여 데이터베이스에 연결한다.
- 연결 성공 메시지 출력: 데이터베이스 연결에 성공하면 **"Database connected successfully."** 메시지를 출력한다.
- 예외 처리: 예외가 발생하면 **e.printStackTrace()**를 호출하여 예외 정보를 출력한다.

2. **searchProfessor** 메서드

교수님의 이름을 입력받아 해당 교수님의 정보를 검색하고, 강의 테이블을 업데이트하는 메서드

[상수/변수/쿼리]

- **professorName**: 검색된 교수님의 이름을 나타내는 문자열 변수
- **professorInfo**: 검색된 교수님의 정보를 담은 **ProfessorInfo** 객체

[메서드 설명]

- 검색어 가져오기: 검색 창에서 입력된 교수님의 이름을 가져온다.
- 검색어 유효성 확인: 검색어가 비어 있는지 확인하고, 비어 있으면 사용자에게 경고 메시지를 표시하고 메서드 실행을 중지한다.
- 교수 정보 가져오기: **getProfessorInfo** 메서드를 호출하여 입력된 교수님의 정보를 가져온다.
- 만약 해당 교수님의 정보가 존재하지 않는다면, 사용자에게 해당 메시지를 표시한다.
- 테이블 업데이트: 검색된 교수님의 정보가 존재한다면, **updateLectureTable** 메서드를 호출하여 강의 테이블을 업데이트한다.
- 예외 처리: SQL 예외가 발생하면 **e.printStackTrace()**를 호출하여 예외 정보를 출력한다.

3. **getProfessorInfo(String professorName)** 메서드

교수님 이름을 받아서 db에서 찾은 해당 교수님의 정보를 개체에 저장하여 반환하는 메서드

[상수/변수/쿼리 설명]

- **professorName**: 교수님의 이름을 나타내는 문자열 변수
- **professorQuery**: 교수 정보를 가져오기 위한 SQL 쿼리 문자열
- **professorStmt**: 교수 정보 쿼리를 실행하기 위한 **PreparedStatement** 객체
- **professorRs**: 교수 정보 쿼리의 결과를 저장하기 위한 **ResultSet** 객체
- **professorInfo**: 교수 정보를 담은 **ProfessorInfo** 객체

[메서드 설명]

- 교수 정보 쿼리 준비: 입력된 교수님의 이름을 기반으로 교수 정보를 가져오는 SQL 쿼리를 준비한다.
- 검색어 설정: **PreparedStatement** 객체에 검색어를 설정한다.
- 쿼리 실행: 쿼리를 실행하여 **ResultSet**을 가져온다.
- 결과 처리: **ResultSet**에서 다음 행이 존재하는지 확인하고, 존재한다면 새로운 **ProfessorInfo** 객체를 생성하고 이름을 설정한다.
- 교수 정보 반환: 교수 정보 객체를 반환한다.

4. **updateLectureTable(String professorName)** 메서드

입력된 교수님의 이름을 기반으로 db에서 해당 교수님의 강의 정보를 가져와서 강의 시간표 테이블을 업데이트하는 메서드

[상수/변수/쿼리 설명]

- **professorName**: 강의 정보를 가져올 교수의 이름을 나타내는 문자열 변수
- **lectureQuery**: 강의 정보를 가져오기 위한 SQL 쿼리 문자열
- **lectureStmt**: 강의 정보 쿼리를 실행하기 위한 **PreparedStatement** 객체
- **lectureRs**: 강의 정보 쿼리의 결과를 저장하기 위한 **ResultSet** 객체
- **lectureName**: 강의의 이름을 나타내는 문자열 변수
- **lectureTime1**, **lectureTime2**: 강의의 시간을 나타내는 문자열 변수
- **dayIndex**: 요일의 인덱스를 나타내는 정수 변수
- **period**: 교시를 나타내는 정수 변수

[메서드 설명]

- 테이블 초기화: 강의 시간표 테이블의 모든 셀을 초기화한다.
- 강의 정보 쿼리 실행: 입력된 교수의 이름을 기반으로 강의 정보를 데이터베이스에서 가져온다.
- 결과 처리: 강의 정보를 하나씩 순회하면서 강의 이름과 강의 시간을 가져와 테이블에 업데이트한다.
- 현재 요일과 교시 강조 표시: 테이블에서 현재 요일과 교시를 강조하여 표시한다.

5. **getDayIndex(String time)** 메서드

강의 시간 문자열 (**time** 문자열) 에서 첫번째 문자를 가져와서 요일에 따른 인덱스를 반환하는 메서드.

[메서드 설명]

- 문자열 첫 글자 분석: 주어진 시간 문자열의 첫 번째 문자를 분석하여 해당하는 요일 인덱스를 반환한다.
- **switch** 문 분기: 시간 문자열의 첫 글자에 따라 다음과 같이 처리한다.
 - '월': 1을 반환 (월요일)
 - '화': 2를 반환 (화요일)
 - '수': 3을 반환 (수요일)
 - '목': 4를 반환 (목요일)
 - '금': 5를 반환 (금요일)

- 기타: 유효하지 않은 경우 -1을 반환

6. highlightCurrentDayAndPeriod() 메서드

현재 요일과 교시를 확인해서 해당되는 셀을 테이블에서 강조해 표시하는 메서드

[상수/변수/쿼리 설명]

- **calendar**: Calendar 클래스의 인스턴스로 현재 날짜와 시간을 저장한다.
- **dayOfWeek**: 현재 요일을 나타내는 변수로서 1(일요일)부터 7(토요일) 사이의 값을 가진다.
- **hourOfDay**: 현재 시간을 나타내는 변수로서 0부터 23까지의 값을 가진다.
- **currentDayIndex**: 현재 요일의 인덱스를 나타내는 변수로서 1(월요일)부터 5(금요일) 사이의 값을 가진다.
- **currentPeriod**: 현재 교시를 나타내는 변수로서 0부터 12까지의 값을 가진다.
- **currentClassInfo**: 시스템상 현재 시간에 수업이 있다면, 해당 수업이 저장되는 문자열 default는 '수업 없음'
- **currentClass**: 시스템상 현재 시간에 해당되는 수업이 저장되는 오브젝트

[메서드 설명]

- 현재 날짜와 시간 확인: **Calendar.getInstance()** 메서드를 사용하여 현재 날짜와 시간을 가져온다.
- 현재 요일 및 시간 추출: **Calendar** 인스턴스에서 현재 요일과 시간을 추출한다.
- 요일 및 교시 인덱스 설정: 추출된 요일과 시간을 기반으로 현재 요일과 교시의 인덱스를 설정한다.
- 테이블 렌더러 초기화: 테이블의 모든 셀 렌더러를 기본값으로 초기화한다.
- 현재 요일 강조 표시: 현재 요일의 컬럼을 핑크색으로 강조한다.
- 현재 교시 강조 표시: 현재 교시와 요일의 셀을 빨간색으로 강조한다.
- 수업 정보 표시: **currentClass**가 존재한다면 **currentClassInfo**가 옆에 표시된다. 없다면 '수업 없음'이 표시된다.
- 테이블 다시 그리기: 변경된 셀 색 및 수업 정보를 적용하여 테이블을 다시 그린다.

7. ClassRenderer 클래스

DefaultTableCellRenderer 클래스 확장해서 강의 테이블의 시스템 시간에 해당되는 요일 열을 분홍색, (요일 열, 시간 행)에 해당되는 셀을 붉은색으로 표시되게끔 하는 클래스.

[상수/변수/생성자 설명]

- 변수
 - **backgroundColor**: 배경색을 나타내는 변수다.
 - **highlightedRow**: 강조할 행을 나타내는 변수이다. 기본값은 -1이며, 이 값이 -1이면 행을 강조하지 않는다.
 - **highlightedColumn**: 강조할 열을 나타내는 변수이다. 기본값은 -1이며, 이 값이 -1이면 열을 강조하지 않는다.
- 생성자:
 - **CustomRenderer(Color backgroundColor)**: 배경색만을 인자로 받는 생성자다.

- CustomRenderer(Color backgroundColor, int highlightedRow): 배경색과 강조할 행을 인자로 받는 생성자다.
- CustomRenderer(Color backgroundColor, int highlightedRow, int highlightedColumn): 배경색과 강조할 행과 열을 인자로 받는 생성자다.

[메서드 설명]

- **getTableCellRendererComponent**: 테이블의 각 셀을 렌더링하는 메서드로, 강조된 열은 핑크색, 강조된 셀은 붉은색, 그 외의 셀은 기본 배경색인 흰색으로 설정된다.

8. ProfessorInfo 클래스

교수님 이름을 정의하는 클래스

[상수/변수/쿼리 설명]

- **Name** : 교수님 이름을 저장하는 문자열 변수

[메서드 설명]

- **getName** : 문자열 **name**을 가져오는 메소드
- **setName** : **name** 문자열에 교수님 이름을 넣는 메소드

9. Lecture 클래스

강의 정보를 정의하는 클래스

[상수/변수/쿼리 설명]

- **lectureName** : 강의 이름을 저장하는 문자열
- **lectureTime1, 2** : 강의 시간1, 2를 저장하는 문자열

[메서드 설명]

- **getLectureName, setLectureName** : 강의 이름을 가져오고 문자열에 부여하는 메소드
- **getLectureTime1/2, setLectureTime1/2** : 강의 시간을 가져오고 문자열에 부여하는 메소드

<professor_button3>

강의 시간과 강의실 위치를 입력하면 해당 강의를 진행하는 교수님 정보를 출력하는 자바 코드

1. searchProfessorByRoomAndTime() 메서드

[상수/변수/쿼리 설명]

- Room : 입력된 강의실을 저장하는 문자열 변수

[메서드 설명]

- 강의실 위치와 선택된 강의 시간을 가져온다.
- 강의실 위치나 강의 시간이 입력되지 않은 경우 경고 메시지를 표시한다.
- 입력된 정보를 바탕으로 `getProfessorsByRoomAndTime(room, selectedTimes)` 메서드를 호출하여 교수님 정보를 가져온다.
- 검색된 교수 정보가 있으면 `updateProfessorInfoArea(professors)` 메서드를 호출하여 출력 영역을 업데이트하고, 없으면 경고 메시지를 표시한다.

2. `getProfessorsByRoomAndTime(String room, List<String> times)` 메서드

[상수/변수/쿼리 설명]

```
StringBuilder queryBuilder = new StringBuilder("SELECT DISTINCT p.Name,
p.Lab_Location, p.Phone, p.Email "
        + "FROM DB2024_Professor p "
        + "JOIN DB2024_Lecture l ON p.Professor_Num = l.Professor_Num "
        + "WHERE l.Room_Number = ? AND (");
```

주어진 강의실 위치, 선택된 시간에 강의를 진행하는 교수님 정보를 조회하기 위한 쿼리.

DB2024_Professor와 DB2024_Lecture 테이블을 조인하여 교수님 정보를 가져오는데, 이 때 강의 시간은 Lecture_Time1 또는 Lecture_Time2가 선택된 시간과 일치해야한다.

[메서드 설명]

- 주어진 강의실 위치와 시간에 해당하는 교수 정보를 데이터베이스에서 조회한다.
- SQL 쿼리를 동적으로 생성하여 준비된 명령문(PreparedStatement)을 사용해 실행한다.
- 조회된 결과를 ProfessorInfo 객체에 저장하고 리스트에 추가한다.
- 교수 정보 리스트를 반환한다.

3. `updateProfessorInfoArea(List<ProfessorInfo> professors)` 메서드:

[메서드 설명]

- 주어진 교수 정보 리스트를 텍스트 형식으로 변환하여 `professorInfoArea`에 표시한다.
- 교수 이름, 연구실 위치, 연락처, 이메일 정보를 포함하여 출력한다.

(3) 관리자 코드

<관리자 클래스>

1. DatabaseManager 메서드

이 메서드는 데이터베이스에 대한 연결을 설정하고 **Statement** 객체를 생성하여 **SQL** 문을 실행할 수 있는 상태로 만듭니다. 이렇게 하면 데이터베이스 관련 작업을 수행할 때마다 매번 연결 객체와 **Statement** 객체를 새로 생성할 필요가 없다.

```
public Connection conn;
public Statement stmt;

public void DatabaseManager() { // conn와 stmt를 매번 호출하지 않아도 실행되도록 함
    try {
        conn = DriverManager.getConnection(DB_URL, USER, PASS); // 데이터베이스 연결을 설정
        stmt = conn.createStatement(); // Statement 객체를 생성하여 SQL 문을 실행할 준비를 함
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

- **DriverManager.getConnection()** 메서드를 사용하여 데이터베이스에 연결한다. **DB_URL**, **USER**, **PASS**는 데이터베이스 연결을 위한 **URL**, 사용자 이름, 암호를 나타내는 상수이다. 연결이 성공하면 **conn** 변수에 연결 객체가 할당된다.
- **conn.createStatement()** 메서드를 호출하여 **Statement** 객체를 생성한다. 이 객체는 데이터베이스에 대한 **SQL** 문을 실행할 준비를 한다. 이후 **SQL** 문을 실행할 때 이 **Statement** 객체를 사용한다.

2. password 메서드

이 코드는 사용자로부터 **MySQL** 계정과 암호를 입력 받아 사용자 인증 기능을 갖춘 로그인 화면을 구현했다.

입력

- **TextField**를 사용하여 사용자로부터 **ID**와 **Password**를 입력 받는다.
- **Password**는 입력 시 '*' 문자로 표시된다.

출력(Output):

- 사용자가 입력한 **ID**와 **Password**가 콘솔에 출력된다.
- 로그인이 성공하면 다음 단계로 넘어감을 나타내는 메시지가 콘솔에 출력된다.

핵심 기능

- 사용자가 **"ENTER"** 버튼을 클릭하면 입력된 **ID**와 **Password**를 확인하여 로그인 여부를 판단한다.

```
// Enter 버튼
JButton submitButton = new JButton("ENTER");
```



```

submitButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == submitButton) {
            String id = idField.getText(); // 입력 받은 id 저장
            String password = passField.getText(); // 입력 받은 password 저장
            System.out.println("ID: " + id);
            System.out.println("Password: " + password);
            if (id.equals(USER) && password.equals(PASS)) {
                // MySQL 계정과 암호가 입력한 내용과 같으면 메뉴를 호출
                DatabaseManager(); // 한번만 실행한다
                menu1();
            }
        }
    }
});

```

- 입력된 ID와 Password가 정해진 상수인 **USER**와 **PASS**와 일치하는지 확인한다.
- 일치할 경우 **DatabaseManager()**와 **menu1()** 메서드를 호출하여 다음 단계로 진행한다.

3. menu1 메서드

menu1 구성

- 강의 등록 버튼 생성: "강의등록" 버튼을 생성하고, 버튼에 액션 리스너를 추가하여 클릭 이벤트를 강의 등록 창으로 넘어갈 수 있도록 처리 한다.
- 기존에 어떤 강의들이 있는지 관리자가 관리 전에 전체를 볼 수 있도록 한다. 결과보기 메서드를 따로 만들어서 넣는다.
- 화면 하단에는 "HOME" 버튼이 위치하며, 이 버튼을 클릭하면 홈 화면으로 돌아간다.

4. initInputVal 메서드

```

private void initInputVal() {
    inLecture_Num.setText(""); // 입력 필드 초기화
    inClass_Num.setText(""); // 입력 필드 초기화
    inLecture_Name.setText(""); // 입력 필드 초기화
    inProfessor_Num.setText(""); // 입력 필드 초기화
    inProfessor_Name.setText(""); // 입력 필드 초기화
    inRoom_Number.setText(""); // 입력 필드 초기화
    inLecture_Time1.setText(""); // 입력 필드 초기화
    inLecture_Time2.setText(""); // 입력 필드 초기화
}

```

- 모든 입력 필드를 초기화하는 메서드이다. 계속 사용되기에 따로 빼서 메서드로 만들었다.

5. editWin(int mode) 메서드

이 메서드는 GUI를 초기화하고, 로고와 메시지를 추가한 후, 강의 정보를 입력할 수 있는 텍스트 필드와 등록/수정/삭제 버튼을 추가한다. **mode** 값에 따라 등록 또는 편집 모드를 결정한다. 등록 모드에서는 등록 버튼과 뒤로가기 버튼을 제공하며, 수정 모드에서는 수정, 삭제 및 뒤로가기

버튼을 제공한다. 각 컴포넌트의 위치와 크기를 직접 설정하고, 최종적으로 컨테이너를 재배치하고 다시 그린다.

```
private void editWin(int mode) { // 튜플 삽입 로직 실행
    // 기존 컴포넌트 초기화
    // 로고 생성

    // insert 임을 알려주는 글 생성
    JLabel logo2;
    if (mode == 0) {
        logo2 = new JLabel("새로운 강의를 등록하세요");
    } else {
        logo2 = new JLabel("강의를 편집하세요");
    }

    // 결과 텍스트와 입력 필드를 생성하여 컨테이너에 추가
```

- 이 메서드는 **mode** 매개변수를 사용하여 실행됩니다. **mode**가 0이면 등록 모드, 그렇지 않으면 편집 모드이다.
- 삽입/편집 안내 문구 생성: **mode**가 0이면 "새로운 강의를 등록하세요"라는 안내 문구를, 그렇지 않으면 "강의를 편집하세요"라는 안내 문구를 생성한다.

```
if (mode == 0) {
    JButton confirmButton = new JButton("등록");
    confirmButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { // 확인 버튼 누르면 실행
            insertTuple();
        }
    });
    ct.add(confirmButton);

    // 뒤로가기 버튼
    input_back.addActionListener(this);
    ct.add(input_back);
```

- 등록모드일 경우, 등록 버튼을 생성하고 클릭시 **insertTuple()** 메서드가 호출한다.
- 뒤로가기 버튼을 생성하여 등록모드에서 나갈 수 있도록 한다.

```
} else {
    // 수정
    JButton confirmButton = new JButton("수정");
    confirmButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { // 수정 버튼 누르면 실행
            updateTuple();
        }
    });
    ct.add(confirmButton);
    // 삭제
    JButton delButton = new JButton("삭제");
    delButton.addActionListener(new ActionListener() {
        @Override
```

```

        public void actionPerformed(ActionEvent e) { // 삭제 버튼 누르면 실행
            deleteTuple();
        }
    });

    ct.add(delButton);

    // 뒤로가기 버튼
    input_back.addActionListener(this);
    ct.add(input_back);
}

// 결과 텍스트와 입력 필드의 배치
ct.revalidate();
ct.repaint();
}

```

- 편집모드일 경우, 수정 버튼을 생성하고 클릭시 `updateTuple()` 메서드가 호출되도록 한다
- 편집모드일 경우, 삭제 버튼을 생성하고 클릭시 `deleteTuple()` 메서드가 호출되도록 한다.
- 뒤로가기 버튼을 생성하여 편집모드에서 나갈 수 있도록 한다.

6. insertTuple 메서드

이 메서드는 강의 정보와 관련된 여러 데이터를 데이터베이스의 두 테이블에 삽입한다. 데이터의 일관성을 유지하기 위해 강의 정보를 `DB2024_Lecture` 테이블에 삽입한 후에 `DB2024_Classroom_Schedule` 테이블에도 삽입한다. 트랜잭션을 사용하여 여러 삽입 작업이 모두 성공하거나 모두 실패하도록 보장하며, 디버깅을 위한 메시지가 콘솔창에 출력되어 오류를 수정할 수 있도록 한다.

```

private void insertTuple() {
    String Lecture_Num = inLecture_Num.getText();
    String Class_Num = inClass_Num.getText();
    String Lecture_Name = inLecture_Name.getText();
    String Professor_Num = inProfessor_Num.getText();
    String Professor_Name = inProfessor_Name.getText();
    String Room_Number = inRoom_Number.getText();
    String Lecture_Time1 = inLecture_Time1.getText();
    String Lecture_Time2 = inLecture_Time2.getText();
    String query1 = "INSERT INTO DB2024_Lecture
(Lecture_Num,Class_Num,Lecture_Name,Professor_Num,Professor_Name,Room_Number,Lecture_Time1,Lecture_Time2)"
        + " VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

    String query2 = "INSERT INTO DB2024_Classroom_Schedule (Room_Number, Lecture_Time) VALUES (?, ?)";
    // 연결된 DB2024_Classroom_Schedule 테이블에서 튜플 추가하기
    String query3 = "INSERT INTO DB2024_Classroom_Schedule (Room_Number, Lecture_Time) VALUES (?, ?)";
    // 연결된 DB2024_Classroom_Schedule 테이블에서 튜플 추가하기
    try (PreparedStatement pstmt1 = conn.prepareStatement(query1);
        PreparedStatement pstmt2 = conn.prepareStatement(query2);
        PreparedStatement pstmt3 = conn.prepareStatement(query3);) {

        try {
            conn.setAutoCommit(false); // 트랜잭션 시작
            pstmt1.setInt(1, Integer.parseInt(Lecture_Num));
            pstmt1.setInt(2, Integer.parseInt(Class_Num));
            pstmt1.setString(3, Lecture_Name);
            pstmt1.setInt(4, Integer.parseInt(Professor_Num));

```

```

pStmt1.setString(5, Professor_Name);
pStmt1.setString(6, Room_Number);
pStmt1.setString(7, Lecture_Time1);
pStmt1.setString(8, Lecture_Time2);
System.out.println("강의 정보 삽입");
System.out.println(pStmt1); // 디버깅 위해 출력
pStmt1.executeUpdate();
try {
    pStmt2.setString(1, Room_Number);
    pStmt2.setString(2, Lecture_Time1);
    System.out.println(pStmt2); // 디버깅 위해 출력
    pStmt2.executeUpdate();
} catch (SQLException se) {
    se.printStackTrace();
    System.out.println("강의일정1 추가 실패 -> Rolling back data");
    try {
        if (conn != null)
            conn.rollback(); // 실패 시 롤백
    } catch (SQLException se2) {
        se2.printStackTrace();
    } // end try
}
try {
    pStmt3.setString(1, Room_Number);
    pStmt3.setString(2, Lecture_Time2);
    System.out.println(pStmt3); // 디버깅 위해 출력
    pStmt3.executeUpdate();
} catch (SQLException se) {
    se.printStackTrace();
    System.out.println("강의일정2 추가 실패 -> Rolling back data");
    try {
        if (conn != null)
            conn.rollback(); // 실패 시 롤백
    } catch (SQLException se2) {
        se2.printStackTrace();
    } // end try
}
initInputVal();
conn.commit(); // 트랜잭션 진행되었다
System.out.println("transaction succeeds");
} catch (SQLException se) {
    se.printStackTrace();
    System.out.println("Rolling back data here....");
    try {
        if (conn != null)
            conn.rollback(); // 실패 시 롤백
    } catch (SQLException se2) {
        se2.printStackTrace();
    } // end try
}
// 한 문장씩 디비에 반영되도록!
conn.setAutoCommit(true);
} catch (SQLException se) {
    se.printStackTrace();
}
}

```

- 각 입력 필드에서 사용자로부터 입력받은 강의 정보를 가져온다.
- **DB2024_Lecture** 테이블과 **DB2024_Classroom_Schedule** 테이블에 삽입할 쿼리를 준비한다.
- **conn.setAutoCommit(false)**를 사용하여 트랜잭션을 시작한다.
- 각 **PreparedStatement** 객체는 트랜잭션 내에서 실행된다. 이는 트랜잭션 범위 내에서 쿼리 실행이 보장됨을 의미한다.

- pStmt1을 사용하여 강의 정보를 DB2024_Lecture 테이블에 삽입하고 executeUpdate()를 호출하여 SQL 문을 실행한다.
- pStmt2와 pStmt3을 사용하여 해당하는 강의실과 강의시간을 DB2024_Classroom_Schedule 테이블에 삽입한다.
- executeUpdate()를 호출하여 SQL 문을 실행한다.
- pStmt2와 pStmt3작업이 실패 시 롤백하여 트랜잭션을 이전 상태로 복원한다.
- 모든 작업이 성공적으로 완료되면 트랜잭션을 커밋하여 데이터를 영구적으로 저장한다.
- 실패 시 롤백하여 이전 상태로 되돌린다.

7. updateTuple 메서드

이 메서드는 트랜잭션을 사용하여 데이터베이스에 있는 DB2024_Lecture 테이블내의 튜플을 안전하게 수정한다. 데이터의 일관성을 유지하기 위해 강의 정보를 DB2024_Lecture 테이블에서 수정한 후에 DB2024_Classroom_Schedule 테이블도 수정한다. 트랜잭션을 사용하여 여러 삽입 작업이 모두 성공하거나 모두 실패하도록 보장하며, 디버깅을 위한 메세지가 콘솔창에 출력되어 오류를 수정할 수 있도록 한다.

```
private void updateTuple() {
    String Lecture_Num = inLecture_Num.getText();
    String Class_Num = inClass_Num.getText();
    String Lecture_Name = inLecture_Name.getText();
    String Professor_Num = inProfessor_Num.getText();
    String Professor_Name = inProfessor_Name.getText();
    String Room_Number = inRoom_Number.getText();
    String Lecture_Time1 = inLecture_Time1.getText();
    String Lecture_Time2 = inLecture_Time2.getText();
    String query1 = "UPDATE DB2024_Lecture "
        + "SET Lecture_Num = ?, Class_Num = ?, Lecture_Name = ?, Professor_Num
= ?, Professor_Name = ?, Room_Number = ?, Lecture_Time1 = ?, Lecture_Time2 = ? "
        + "WHERE Lecture_Num = ? AND Class_Num = ? ";

    String query2 = "UPDATE DB2024_Classroom_Schedule " + "SET Room_Number = ?, Lecture_Time = ?" +
"WHERE Room_Number = ? AND Lecture_Time = ?";
    // 연결된 DB2024_Classroom_Schedule 테이블에서 튜플 수정하기
    String query3 = "UPDATE DB2024_Classroom_Schedule " + "SET Room_Number = ?, Lecture_Time = ?" +
"WHERE Room_Number = ? AND Lecture_Time = ?";
    // 연결된 DB2024_Classroom_Schedule 테이블에서 튜플 수정하기
    try (PreparedStatement pStmt1 = conn.prepareStatement(query1);
        PreparedStatement pStmt2 = conn.prepareStatement(query2);
        PreparedStatement pStmt3 = conn.prepareStatement(query3);) {

        try {
            conn.setAutoCommit(false); // 트랜잭션 시작
            pStmt1.setInt(1, Integer.parseInt(Lecture_Num));
            pStmt1.setInt(2, Integer.parseInt(Class_Num));
            pStmt1.setString(3, Lecture_Name);
            pStmt1.setInt(4, Integer.parseInt(Professor_Num));
            pStmt1.setString(5, Professor_Name);
            pStmt1.setString(6, Room_Number);
            pStmt1.setString(7, Lecture_Time1);
            pStmt1.setString(8, Lecture_Time2);
            pStmt1.setInt(9, Integer.parseInt(Lecture_Num_ori)); // 수정 전의 원본 값 사용
            pStmt1.setInt(10, Integer.parseInt(Class_Num_ori)); // 수정 전의 원본 값 사용
            System.out.println(pStmt1); // 디버깅 위해 출력
            pStmt1.executeUpdate();
            try {
                pStmt2.setString(1, Room_Number);
                pStmt2.setString(2, Lecture_Time1);
                pStmt2.setString(3, Room_Number_ori); // 원본 값을 사용
```

```

        pstmt2.setString(4, Lecture_Time1_ori); // 원본 값을 사용
        System.out.println(pstmt2); // 디버깅 위해 출력
        pstmt2.executeUpdate();
    } catch (SQLException se) {
        se.printStackTrace();
        System.out.println("강의일정1 수정 실패 -> Rolling back data");
        try {
            if (conn != null)
                conn.rollback(); // 실패 시 롤백
        } catch (SQLException se2) {
            se2.printStackTrace();
        } // end try
    }
    try {
        pstmt3.setString(1, Room_Number);
        pstmt3.setString(2, Lecture_Time2);
        pstmt3.setString(3, Room_Number_ori); // 원본 값을 사용
        pstmt3.setString(4, Lecture_Time2_ori); // 원본 값을 사용
        System.out.println(pstmt3); // 디버깅 위해 출력
        pstmt3.executeUpdate();
    } catch (SQLException se) {
        se.printStackTrace();
        System.out.println("강의일정2 수정 실패 -> Rolling back data");
        try {
            if (conn != null)
                conn.rollback(); // 실패 시 롤백
        } catch (SQLException se2) {
            se2.printStackTrace();
        } // end try
    }
    conn.commit(); // 트랜잭션 진행되었다
    System.out.println("transaction succeeds");
} catch (SQLException se) {
    se.printStackTrace();
    System.out.println("Rolling back data here....");
    try {
        if (conn != null)
            conn.rollback(); // 실패 시 롤백
    } catch (SQLException se2) {
        se2.printStackTrace();
    }
}
conn.setAutoCommit(true);
} catch (SQLException se) {
    se.printStackTrace();
}
}

```

- 각 입력 필드에서 수정된 강의 정보를 가져온다.
- **DB2024_Lecture** 테이블과 **DB2024_Classroom_Schedule** 테이블에 수정할 쿼리를 준비한다.
- **conn.setAutoCommit(false)**를 사용하여 트랜잭션을 시작한다.
- 각 **PreparedStatement** 객체는 트랜잭션 내에서 실행된다. 이는 트랜잭션 범위 내에서 쿼리 실행이 보장됨을 의미한다.
- **pstmt1**을 사용하여 **DB2024_Lecture** 테이블에서 강의 정보를 삭제한다. 이때 **WHERE** 절에서는 수정전의 기본키 값을 사용한다.
- **pstmt2**와 **pstmt3**을 사용하여 **DB2024_Classroom_Schedule** 테이블에서 바뀐 강의실과 강의시간을 수정한다. 이때 **WHERE** 절에서는 수정전의 기본키 값을 사용한다.
- **executeUpdate()**를 호출하여 SQL 문을 실행한다.
- **pstmt2**와 **pstmt3**작업이 실패 시 롤백하여 트랜잭션을 이전 상태로 복원한다.
- 모든 작업이 성공적으로 완료되면 트랜잭션을 커밋하여 데이터를 영구적으로 저장한다.

- 실패 시 롤백하여 이전 상태로 되돌린다.

8. deleteTuple 메서드

이 메서드는 트랜잭션을 사용하여 데이터베이스에 있는 **DB2024_Lecture** 테이블내의 튜플을 안전하게 삭제한다. 데이터의 일관성을 유지하기 위해 강의 정보를 **DB2024_Lecture** 테이블에서 삭제한 후에 **DB2024_Classroom_Schedule** 테이블에서도 삭제한다. 트랜잭션을 사용하여 여러 삽입 작업이 모두 성공하거나 모두 실패하도록 보장하며, 디버깅을 위한 메시지가 콘솔창에 출력되어 오류를 수정할 수 있도록 한다.

```
private void deleteTuple() {
    String Lecture_Num = inLecture_Num.getText();
    String Class_Num = inClass_Num.getText();
    String Lecture_Name = inLecture_Name.getText();
    String Professor_Num = inProfessor_Num.getText();
    String Professor_Name = inProfessor_Name.getText();
    String Room_Number = inRoom_Number.getText();
    String Lecture_Time1 = inLecture_Time1.getText();
    String Lecture_Time2 = inLecture_Time2.getText();

    String query1 = "DELETE FROM DB2024_Lecture WHERE Lecture_Num = ? AND Class_Num = ?";
    // Lecture_Num, Class_Num, Lecture_Name, Professor_Num, Professor_Name,
    String query2 = "DELETE FROM DB2024_Classroom_Schedule WHERE Room_Number = ? AND
Lecture_Time = ?";
    // 연결된 DB2024_Classroom_Schedule 테이블에서 삭제된 강의의 강의실 시간 정보도 삭제
    String query3 = "DELETE FROM DB2024_Classroom_Schedule WHERE Room_Number = ? AND Lecture_Time =
?";
    // 연결된 DB2024_Classroom_Schedule 테이블에서 삭제된 강의의 강의실 시간 정보도 삭제

    try (PreparedStatement pstmt1 = conn.prepareStatement(query1)) {
        conn.setAutoCommit(false); // 트랜잭션 시작
        // 첫 번째 쿼리 실행
        pstmt1.setInt(1, Integer.parseInt(Lecture_Num));
        pstmt1.setInt(2, Integer.parseInt(Class_Num));
        System.out.println(pstmt1);
        pstmt1.executeUpdate();

        // 두 번째 쿼리 실행
        try (PreparedStatement pstmt2 = conn.prepareStatement(query2)) {
            pstmt2.setString(1, Room_Number_ori);
            pstmt2.setString(2, Lecture_Time1_ori);
            System.out.println(pstmt2);
            pstmt2.executeUpdate();
        } catch (SQLException se) {
            se.printStackTrace();
            System.out.println("강의일정1 삭제 실패 -> Rolling back data");
            try {
                if (conn != null)
                    conn.rollback(); // 실패 시 롤백
            } catch (SQLException se2) {
                se2.printStackTrace();
            } // end try
        }
    }
    // 세 번째 쿼리 실행
    try (PreparedStatement pstmt3 = conn.prepareStatement(query3)) {
        pstmt3.setString(1, Room_Number_ori);
        pstmt3.setString(2, Lecture_Time2_ori);
        pstmt3.executeUpdate();
        System.out.println(pstmt3);
    }
}
```

```

        } catch (SQLException se) {
            se.printStackTrace();
            System.out.println("강의일정2 삭제 실패 -> Rolling back data");
            try {
                if (conn != null)
                    conn.rollback(); // 실패 시 롤백
            } catch (SQLException se2) {
                se2.printStackTrace();
            } // end try
        }
        initInputVal(); // 입력필드 초기화
        conn.commit(); // 트랜잭션 진행되었다
        System.out.println("transaction succeeds");
    } catch (SQLException se) {
        se.printStackTrace();
        System.out.println("Rolling back data here....");
        try {
            if (conn != null)
                conn.rollback(); // 실패 시 롤백
        } catch (SQLException se2) {
            se2.printStackTrace();
        } // end try
    } finally {
        // Auto-commit 설정을 원래대로 돌림
        try {
            conn.setAutoCommit(true);
        } catch (SQLException se) {
            se.printStackTrace();
        }
    }
}
}

```

- 각 입력 필드에서 사용자로부터 삭제할 강의 정보를 가져온다.
- **DB2024_Lecture** 테이블과 **DB2024_Classroom_Schedule** 테이블에 대한 삭제 쿼리를 준비한다.
- **conn.setAutoCommit(false)**를 사용하여 트랜잭션을 시작한다.
- 각 **PreparedStatement** 객체는 트랜잭션 내에서 실행된다. 이는 트랜잭션 범위 내에서 쿼리 실행이 보장됨을 의미한다.
- **pStmt1**을 사용하여 **DB2024_Lecture** 테이블에서 강의 정보를 삭제한다. 이때 **WHERE** 절에서는 수정전의 기본키 값을 사용한다.
- **pStmt2**와 **pStmt3**을 사용하여 **DB2024_Classroom_Schedule** 테이블에서 삭제한다.
- **executeUpdate()**를 호출하여 SQL 문을 실행한다.
- **pStmt2**와 **pStmt3**작업이 실패 시 롤백하여 트랜잭션을 이전 상태로 복원한다.
- 모든 작업이 성공적으로 완료되면 트랜잭션을 커밋하여 데이터를 영구적으로 저장한다.
- 실패 시 롤백하여 이전 상태로 되돌린다.
- **finally** 블록에서 **conn.setAutoCommit(true)**을 호출하여 트랜잭션 상태를 원래대로 복원한다.

9. 결과보기 메서드

이 메서드는 데이터베이스에서 검색된 정보를 메타데이터를 사용하여 처리하고, 데이터베이스에서 데이터를 가져와 **JTable**에 표시하고, 창 크기 변경 시 **JScrollPane**을 조정하며 이를 사용자에게 표시하는 역할을 한다.


```

public void 결과보기() { // 선택한 테이블 이름을 매개변수로 받아 해당 테이블을 보여주는 함수
    String query2 = "SELECT * FROM DB2024_Lecture";
    System.out.println(query2); // 디버깅을 위해 출력
    // JTable에 데이터를 넣기 위한 기본 테이블 모델을 생성합니다.
    DefaultTableModel tableModel = new DefaultTableModel();
    try (Statement stmt1 = conn.createStatement();
        ResultSet rs = stmt1.executeQuery(query2)) {
        // ResultSet의 메타데이터를 가져오기
        ResultSetMetaData metaData = rs.getMetaData();
        // 열의 수를 가져오기

        // 모든 열 이름을 가져와 벡터에 추가

        // Create JTable with the table model

        // resize event -> resize ScrollPane
        ct.addComponentListener(new ComponentAdapter() {
            // 마우스가 가르키는 테이블의 값을 입력값으로 만드는 함수
            table.addMouseListener(this); // --> mouseClicked
            scrollPane.setVisible(true);
            ct.revalidate();
            ct.repaint();
        });
    }
}

```

- **DB2024_Lecture** 테이블에서 데이터를 검색하기 위해 **SELECT * FROM DB2024_Lecture** 쿼리를 실행한다.
- 실행된 쿼리의 결과는 **ResultSet** 객체로 반환한다. 결과셋의 메타데이터를 가져오기 위해 **ResultSetMetaData** 객체를 사용하고, 열의 수와 각 열의 이름을 가져온다.
- 테이블 모델을 생성하여 테이블의 데이터를 저장한다. **ResultSetMetaData** 객체를 사용하여 테이블 모델의 열 이름을 설정한다.
- 결과셋에서 각 행의 데이터를 가져와서 테이블 모델에 추가한다. 각 행의 데이터는 열 이름에 따라 테이블 모델에 삽입된다.
- 데이터가 채워진 테이블 모델을 사용하여 스크롤 패널을 설정하고 **JTable**을 포함한다. 컨테이너에 추가하고 화면을 갱신하여 사용자에게 데이터를 보여준다.

10. mouseClicked 메서드

이 메서드는 테이블에서 튜플을 클릭했을 때 발생하는 이벤트를 처리합니다. 클릭한 튜플의 정보를 가져와서 해당 정보를 텍스트 필드에 표시하고, 편집 모드로 전환한다.

- **mouseClicked()** 메서드는 마우스 클릭 이벤트가 발생했을 때 호출된다. 클릭한 테이블의 행(row)과 열(col)을 가져온다.
- 클릭한 튜플의 정보를 가져와서 문자열로 저장한다. 각 열의 정보를 가져와서 텍스트 필드에 설정하고, 원본 값 변수에 저장한다.
- 각 열의 정보를 텍스트 필드에 설정한다. 텍스트 필드에 설정된 정보는 사용자가 수정할 수 있도록 한다.
- 편집 모드 전환: **editWin(1)**을 호출하여 편집 모드로 전환한다. 이때, **mode** 매개변수가 1로 전달된다.

11. mousePressed, mouseReleased, mouseEntered, mouseExited 메서드

- `mousePressed()`: 마우스 버튼이 눌렸을 때의 이벤트를 처리
- `mouseReleased()`: 마우스 버튼이 떼어졌을 때의 이벤트를 처리
- `mouseEntered()`: 마우스가 컴포넌트에 들어갔을 때의 이벤트를 처리
- `mouseExited()`: 마우스가 컴포넌트를 벗어났을 때의 이벤트를 처리

5. 응용프로그램 설치 및 사용 방법, 연결을 위한 구성(configuration)에 대한 지시 사항

데이터베이스를 SQL로 구축하고, 이클립스에서 자바 코드를 구현했다. 이를 통해 문제 없이 응용 프로그램을 사용할 수 있다.

DB2024Team05_code 폴더 안에 자바 프로젝트가 들어있다.

그 중 DB2024Team05src 폴더 안에 .java 파일들이 들어있다.

[실행 방법]

- (1) `create.sql` 을 통해 DB 생성 후,
- (2) `mainFrame` 폴더의 `mainGUI` 파일을 실행시킨다.

6. 제시한 17개의 요구 사항을 만족함을 보이는 자세한 설명

(1) 5개 이상의 테이블을 가지고 있어야 하고 각 테이블들의 컬럼(Attribute)의 수를 합하면 20개 이상이어야 한다.

테이블 6개, 속성 총 32개로 이루어져 있다.

TABLE_NAME
db2024_classroom
db2024_classroom_external
db2024_classroom_schedule
db2024_classroomexternalview
db2024_classroomview
db2024_lecture
db2024_lecturereview
db2024_professor
db2024_space_info

1. DB2024_Space_Info : 공대에 사용 가능한 공간들에 대해 저장할 테이블

공간 번호, 이름, 위치를 속성으로 가짐

```
Room_Number VARCHAR(50) PRIMARY KEY,  
Room_Name VARCHAR(50) not null,  
Location VARCHAR(50) not null
```

2. TABLE DB2024_Classroom : 공대에 존재하는 강의실에 대해 저장할 테이블

공간 번호, 실습 할 컴퓨터가 있는지, 좌석 수, 예약 필요 여부, 녹화 가능 여부, 녹화하는 카메라의 종류, 수업이 언제 있는지를 속성으로 가짐

```
Room_Number VARCHAR(10) PRIMARY KEY,  
Practicable VARCHAR(20),  
Seat_Count INT,  
Projector VARCHAR(20),  
OutletCount INT,  
ReservationRequired VARCHAR(20),  
RecordingAvailable VARCHAR(20),  
CameraType VARCHAR(50)
```

3. DB2024_Classroom_Schedule : 강의실에 수업이 언제 있는지에 대해 저장할 테이블

강의시간, 공간 번호를 속성으로 가짐

```
Lecture_Time VARCHAR(10),  
Room_Number VARCHAR(10),  
FOREIGN KEY (Room_Number) REFERENCES  
DB2024_Space_Info(Room_Number),  
PRIMARY KEY(Room_Number, Lecture_Time)
```

4. DB2024_Classroom_External : 강의실 외의 공간에 대해 저장할 테이블

공간 번호, 취식 가능 여부, 소음이 발생해도 되는지, 예약 필요 여부, 좌석수, 콘센트 개수를 속성으로 가짐

```
Room_Number VARCHAR(50) PRIMARY KEY,  
Eat_Available BOOLEAN,  
Noise_Level BOOLEAN,  
Reservation_Needed BOOLEAN,  
Seat_Count INT,  
Outlet_Count INT,  
FOREIGN KEY (Room_Number) REFERENCES  
DB2024_Space_Info(Room_Number)
```

5. DB2024_Professor : 교수님의 정보들에 대해 저장할 테이블

교번, 교수님 성함, 연구실 위치, 전화번호, 이메일을 속성으로 가짐

```
Professor_Num INT,  
Name VARCHAR(100),  
Lab_Location VARCHAR(100),  
Phone VARCHAR(50),  
Email VARCHAR(100),  
PRIMARY KEY (Professor_Num, Name)
```

6. DB2024_Lecture : 강의 정보에 대해 저장할 테이블

학수번호, 분반 번호, 강의 이름, 교수님 성함, 교번, 강의실 공간 번호를 속성으로 가짐

```
Lecture_Num INT,  
Class_Num INT,  
Lecture_Name VARCHAR(100) not null,  
Professor_Name VARCHAR(100), /*교수님 1명만*/  
Professor_Num INT,  
Room_Number VARCHAR(50) not null,  
Lecture_Time1 VARCHAR(50),  
Lecture_Time2 VARCHAR(50), /*수업시간 1or2개로*/  
PRIMARY KEY(Lecture_Num, Class_Num),  
FOREIGN KEY (Room_Number) REFERENCES  
DB2024_Space_Info(Room_Number),  
FOREIGN KEY (Professor_Num, Professor_Name)  
REFERENCES DB2024_Professor(Professor_Num, Name)
```

(2) 초기화를 위해 적어도 30개의 레코드(튜플)를 가지고 있어야 한다. (모든 테이블들의 레코드 수의 총합이 30개 이상)

레코드 총 664개

```
mysql> select count(*)
-> from DB2024_classroom;
+-----+
| count(*) |
+-----+
|      36 |
+-----+
```

```
mysql> select count(*)
-> from db2024_lecture;
+-----+
| count(*) |
+-----+
|     165 |
+-----+
```

```
mysql> select count(*)
-> from DB2024_classroom_external;
+-----+
| count(*) |
+-----+
|      16 |
+-----+
```

```
mysql> select count(*)
-> from db2024_professor;
+-----+
| count(*) |
+-----+
|     128 |
+-----+
```

```
mysql> select count(*)
-> from DB2024_classroom_schedule;
+-----+
| count(*) |
+-----+
|     319 |
+-----+
```

(3) 기본 키(primary key), 외래 키(foreign key), not null 제약조건(not null constraints)를 포함해야 한다.

위의 (1)번 요구사항을 보면 해당 조건의 만족함을 보인다.

(4) 적어도 2개의 뷰를 정의해야 한다.

강의실에 대한 정보를 담고 있는 DB2024_Classroom 테이블과 강의실 외 공간에 대한 정보를 담고 있는 DB2024_Classroom_External 테이블에 해당 공간의 번호에 관한

내용은 존재하지만 공간의 이름이나, 위치는 존재하지 않기에 **View**를 만들어서 공간에 대한 정보를 사용할 때 공간의 이름과 위치도 같이 사용할 수 있도록 하였다.

또한 기존 **DB2024_Lecture** 테이블에 해당 강의가 진행되는 강의실의 번호는 존재하나 해당 강의실의 이름과 위치 정보가 없기에 **View**를 만들어 해당 내용까지 모두 사용할 수 있도록 하였다.

```
CREATE OR REPLACE VIEW DB2024_LectureView AS
SELECT DB2024_Space_Info.Room_Name,DB2024_Space_Info.Location,DB2024_Lecture.*
FROM db2024_lecture, db2024_space_info
WHERE DB2024_Space_Info.Room_Number = DB2024_Lecture.Room_Number;

CREATE OR REPLACE VIEW DB2024_ClassroomView AS
SELECT DB2024_Space_Info.Room_Name,DB2024_Space_Info.Location,DB2024_Classroom.*
FROM db2024_classroom, db2024_space_info
WHERE DB2024_Space_Info.Room_Number = DB2024_Classroom.Room_Number;

CREATE OR REPLACE VIEW DB2024_ClassroomExternalView AS
SELECT DB2024_Space_Info.Room_Name,DB2024_Space_Info.Location,db2024_classroom_external.*
FROM db2024_classroom_external, db2024_space_info
WHERE DB2024_Space_Info.Room_Number = db2024_classroom_external.Room_Number;
```

(5) 모든 테이블과 뷰의 이름들은 “DB2024_”라는 접두어를 가지고 있어야 한다.

6개의 테이블 모두 “DB2024_”라는 접두어를 가진다.

- DB2024_Space_Info
- DB2024_Classroom
- DB2024_Classroom_External
- DB2024_Professor
- DB2024_Lecture
- DB2024_Classroom_Schedule

모든 뷰의 이름은 “DB2024_”라는 접두어를 가진다.

- DB2024_LectureView
- DB2024_ClassroomView
- DB2024_ClassroomExternalView

(6) 적어도 4개의 인덱스를 정의해야 한다.

```
# 학생과 교수님이 교수님 성함으로 검색하는 경우가 많기에 이를 빠르게 하기 위해 인덱스 생성
CREATE INDEX idx_name
ON DB2024_Professor (Name);

# 빈 강의실을 찾을 때 강의실 번호를 자주 사용하므로 인덱스로 생성
CREATE INDEX idx_room_number
ON DB2024_Lecture (Room_Number);

# 교수정보를 찾을 때 교수님을 찾을 때 교수 번호를 자주 사용하므로 인덱스로 생성
CREATE INDEX idx_professor_num_on_professor
ON DB2024_Professor (Professor_Num);
```

```
# 교수님의 강의정보를 찾을 때 교수 번호를 자주 사용하므로 인덱스로 생성
CREATE INDEX idx_professor_num_on_lecture
ON DB2024_Lecture (Professor_Num);
```

(7) 인덱스를 사용하는 쿼리들을 포함해야 한다.

```
private void searchProfessorInfo(String name) {

    //JDBC driver name and database URL
    // MySQL 계정과 암호 입력

    String query = "SELECT Email, Lab_Location, Phone FROM DB2024_Professor WHERE Name
= ?";
    // 나머지 처리하는 코드들
    }
} catch (SQLException e) {
    e.printStackTrace();
    infoArea.setText("Error retrieving data.");
}
}
```

DB2024_Professor 테이블에서 Name을 기준으로 한 검색 쿼리의 실행 속도를 향상시키기 위해 Name 컬럼에 인덱스를 생성했다. 이 인덱스는 Name 컬럼에 대한 데이터 접근을 빠르게 하여, 효율적인 데이터 검색과 쿼리 성능 개선을 도모했다.

```
CREATE INDEX idx_name ON DB2024_Professor (Name);
```

- Name 인덱스

: Name 컬럼에 인덱스를 생성한 이유는 DB2024_Professor 테이블에서 Name을 기준으로 한 검색 쿼리의 실행 속도를 향상시키기 위함이다, 이 인덱스는 Name 컬럼에 대한 데이터 접근을 빠르게 하여, 효율적인 데이터 검색과 쿼리 성능을 개선했다.

```
// 강의실과 시간으로 교수 정보 가져오기 메서드
private List<ProfessorInfo> getProfessorsByRoomAndTime(String room, List<String> times) throws
SQLException {
    // SQL 쿼리 생성
    StringBuilder queryBuilder = new StringBuilder("SELECT DISTINCT p.Name, p.Lab_Location,
p.Phone, p.Email "
        + "FROM DB2024_Professor p "
        + "JOIN DB2024_Lecture l ON p.Professor_Num = l.Professor_Num "
        + "WHERE l.Room_Number = ? AND (");

    for (int i = 0; i < times.size(); i++) {
        queryBuilder.append("l.Lecture_Time1 = ? OR l.Lecture_Time2 = ?");
        if (i < times.size() - 1) {
            queryBuilder.append(" OR ");
        }
    }
}
```

```
}
queryBuilder.append(")");
```

이 쿼리는 **DB2024_Professor** 테이블과 **DB2024_Lecture** 테이블을 조인하고 있으며, **Room_Number**와 **Professor_Num** 필드를 사용하고 있다. 이 필드들에 적절히 인덱스를 추가하면, 조인과 검색 성능을 향상시킨다.

Room_Number는 **WHERE** 절에 사용되고 있어, 이 컬럼에 인덱스를 추가하는 것이 검색 속도를 높이는 데 도움이 된다.

```
CREATE INDEX idx_room_number
ON DB2024_Lecture (Room_Number);
```

- **Room_Number** 인덱스

: 쿼리의 **WHERE** 절에서 **Room_Number**를 사용해 레코드를 필터링하는 경우에 데이터베이스가 전체 테이블 스캔을 수행하는 대신 인덱스를 통해 훨씬 빠르게 필요한 레코드를 찾을 수 있도록 도와준다.

Professor_Num 컬럼은 두 테이블을 **JOIN**하는 데 사용되므로, 이 컬럼에 인덱스를 추가하면 조인 연산의 효율성을 높일 수 있다. 두 테이블 모두에 인덱스를 추가하는 것이 좋다.

```
-- DB2024_Professor 테이블에 인덱스 추가
CREATE INDEX idx_professor_num_on_professor
ON DB2024_Professor (Professor_Num);

-- DB2024_Lecture 테이블에 인덱스 추가
CREATE INDEX idx_professor_num_on_lecture
ON DB2024_Lecture (Professor_Num);
```

- **Professor_Num** 인덱스

: **Professor_Num**을 기준으로 두 테이블을 조인하는 작업은 비용이 많이 드는 연산인데 인덱스를 사용하면, 조인이 필요한 레코드를 훨씬 빠르고 효율적으로 찾아 조인 비용을 줄일 수 있다. 각 테이블에서 **Professor_Num**에 대한 인덱스를 구성함으로써, 데이터베이스는 이를 활용하여 조인 성능을 향상시킬 수 있다.

(8) 뷰를 사용하는 쿼리를 포함해야 한다.

- professor_button1 클래스의 buildQuery 메소드

: 교실/교실 외 조건을 필터링하기 위해 **ClassroomView**, **ClassroomExternalView** 사용

```
private String buildQuery(boolean content, boolean project, boolean reservation, boolean
recording, boolean practicable, int type, String cameraType) {
    StringBuilder query = new StringBuilder("SELECT * FROM ");
```



```

// type은 사용자가 찾기를 원하는 것이 교실(0)인지 교실 외(1)인지를 구분
if (type == 0) {
    query.append("DB2024_ClassroomView");
    boolean hasCondition = false;
    if (practicable || project || reservation || recording ||
!cameraType.equals("선택")) {
        query.append(" WHERE");
        if (practicable) {
            query.append(" Practicable='실습가능'");
            hasCondition = true;
        }
        if (project) {
            if (hasCondition) {
                query.append(" AND");
            }
            query.append(" Projector='빔 있음'");
            hasCondition = true;
        }
        if (reservation) {
            if (hasCondition) {
                query.append(" AND");
            }
            query.append(" ReservationRequired='예약 필요'");
            hasCondition = true;
        }
        if (recording) {
            if (hasCondition) {
                query.append(" AND");
            }
            query.append(" RecordingAvailable='가능'");
            hasCondition = true;
        }
        if (!cameraType.equals("선택")) {
            if (hasCondition) {
                query.append(" AND");
            }
            query.append(" CameraType='').append(cameraType).append('");
        }
    }
} else if (type == 1) {
    query.append("DB2024_ClassroomExternalView");
    if (content) {
        query.append(" WHERE Outlet_Count > 0");
    }
}
return query.toString();
}

```

(9) 트랜잭션(transaction)을 포함해야 한다.

- 관리자 클래스의 `insertTuple`, `updateTuple`, `deleteTuple` 메서드 모두 트랜잭션이 포함된다. 그중 `deleteTuple` 메서드를 예시로 들어 조건이 만족함을 보인다.

: 어떤 강의를 삭제하면 해당 강의가 이루어지는 강의실의 강의시간 또한 모두 삭제되어야 하므로 이를 위해 트랜잭션을 사용

```
private void deleteTuple() {
```

```

String Lecture_Num = inLecture_Num.getText();
String Class_Num = inClass_Num.getText();
String Lecture_Name = inLecture_Name.getText();
String Professor_Num = inProfessor_Num.getText();
String Professor_Name = inProfessor_Name.getText();
String Room_Number = inRoom_Number.getText();
String Lecture_Time1 = inLecture_Time1.getText();
String Lecture_Time2 = inLecture_Time2.getText();

String query1 = "DELETE FROM DB2024_Lecture WHERE Lecture_Num = ? AND Class_Num = ?";
// Lecture_Num, Class_Num, Lecture_Name, Professor_Num, Professor_Name,
String query2 = "DELETE FROM DB2024_Classroom_Schedule WHERE Room_Number = ? AND
Lecture_Time = ?";
// 연결된 DB2024_Classroom_Schedule 테이블에서 삭제된 강의의 강의실 시간 정보도 삭제
String query3 = "DELETE FROM DB2024_Classroom_Schedule WHERE Room_Number = ? AND Lecture_Time =
?";
// 연결된 DB2024_Classroom_Schedule 테이블에서 삭제된 강의의 강의실 시간 정보도 삭제

try (PreparedStatement pstmt1 = conn.prepareStatement(query1)) {
    conn.setAutoCommit(false); // 트랜잭션 시작
    // 첫 번째 쿼리 실행
    pstmt1.setInt(1, Integer.parseInt(Lecture_Num));
    pstmt1.setInt(2, Integer.parseInt(Class_Num));
    System.out.println(pstmt1);
    pstmt1.executeUpdate();

    // 두 번째 쿼리 실행
    try (PreparedStatement pstmt2 = conn.prepareStatement(query2)) {
        pstmt2.setString(1, Room_Number_ori);
        pstmt2.setString(2, Lecture_Time1_ori);
        System.out.println(pstmt2);
        pstmt2.executeUpdate();
    } catch (SQLException se) {
        se.printStackTrace();
        System.out.println("강의일정1 삭제 실패 -> Rolling back data");
        try {
            if (conn != null)
                conn.rollback(); // 실패 시 롤백
        } catch (SQLException se2) {
            se2.printStackTrace();
        } // end try
    }
    // 세 번째 쿼리 실행
    try (PreparedStatement pstmt3 = conn.prepareStatement(query3)) {
        pstmt3.setString(1, Room_Number_ori);
        pstmt3.setString(2, Lecture_Time2_ori);
        pstmt3.executeUpdate();
        System.out.println(pstmt3);
    } catch (SQLException se) {
        se.printStackTrace();
        System.out.println("강의일정2 삭제 실패 -> Rolling back data");
        try {
            if (conn != null)
                conn.rollback(); // 실패 시 롤백
        } catch (SQLException se2) {
            se2.printStackTrace();
        } // end try
    }
    initInputVal(); // 입력필드 초기화
    conn.commit(); // 트랜잭션 진행되었다
    System.out.println("transaction succeeds");
} catch (SQLException se) {
    se.printStackTrace();
    System.out.println("Rolling back data here....");
    try {
        if (conn != null)
            conn.rollback(); // 실패 시 롤백
    }
}

```

```

        } catch (SQLException se2) {
            se2.printStackTrace();
        } // end try
    } finally {
        // Auto-commit 설정을 원래대로 돌림
        try {
            conn.setAutoCommit(true);
        } catch (SQLException se) {
            se.printStackTrace();
        }
    }
}

```

- `conn.setAutoCommit(false)` 자동 커밋을 비활성화하고, 수동 커밋을 사용하도록 설정하는 트랜잭션의 시작을 나타낸다.
- 각 `PreparedStatement` 객체는 트랜잭션 내에서 실행된다. 이는 트랜잭션 범위 내에서 쿼리 실행이 보장됨을 의미한다.
- `query1`에서는 `DB2024_Lecture` 테이블에서 선택한 강의를 삭제하려고 한다. 하지만 `DB2024_Lecture` 테이블의 속성에는 해당 강의가 이루어지는 강의실의 강의시간이 포함되어 있다. `DB2024_Classroom_Schedule` 테이블의 강의실과 강의시간 모두 삭제되어야 하므로 이를 위해 트랜잭션을 사용한다.
- `query2`, `query3` 두 개의 `Lecture_Time`이 같다면 콘솔창에 어느 쿼리에서 오류가 나서 롤백 했는지 알려준다.
- 모든 쿼리가 성공적으로 실행되면, `conn.commit();`을 호출하여 트랜잭션을 커밋한다. 이는 데이터베이스에 대한 모든 변경 사항을 영구적으로 적용하기 위해 사용한다.
- 예외가 발생하면, `catch` 블록에서 트랜잭션을 롤백할 수 있다. 롤백은 트랜잭션 내에서 발생한 모든 변경 사항을 취소하고 이전 상태로 되돌린다.
- 트랜잭션의 성공 여부와 관계없이 트랜잭션 종료 후에는 `conn.setAutoCommit(true);`을 호출하여 자동 커밋을 다시 활성화한다.

(10) 중첩된 쿼리(nested query)들을 가지는 쿼리들을 포함해야 한다.

```

//강의번호나 강의이름에 해당하는 공간번호를 보여주는 코드
private void showRoomNumber(String lectureNumber) {
    String dbUrl = "jdbc:mysql://localhost/DB2024Team05";
    String dbUser = "DB2024Team05";
    String dbPass = "DB2024Team05";

    // 학수번호와 함께 받아와서 분리
    String[] parts = lectureNumber.split("-");
    if (parts.length != 2) {
        JOptionPane.showMessageDialog(this, "입력값을 확인하세요");
        return; // 입력이 올바르게 분리되지 않은 경우
    }

    // 중첩 쿼리를 사용하여 해당 강의실의 추가적인 특성 정보를 조회하는 쿼리
    String query = "SELECT * FROM DB2024_ClassroomView " +
        "WHERE Room_Number = (SELECT Room_Number FROM DB2024_LectureView " +
        "WHERE Lecture_Num = ? AND Class_Num = ?)";
    if (isInteger(parts[0])) {
        query += "WHERE Lecture_Num = ? AND Class_Num = ?";
    } else {
        query += "WHERE Lecture_Name = ? AND Class_Num = ?";
    }

    try (Connection conn = DriverManager.getConnection(dbUrl, dbUser, dbPass);

```

```

        PreparedStatement stmt = conn.prepareStatement(query)) {
    stmt.setString(1, parts[0]); // Lecture_Num or Lecture_Name
    stmt.setString(2, parts[1]); // Class_Num

    try (ResultSet rs = stmt.executeQuery()) {
        if (rs.next()) {
            String roomNumber = rs.getString("Room_Number");
            String roomName = rs.getString("Room_Name");
            String location = rs.getString("Location");
            int seatCount = rs.getInt("Seat_Count");
            String projector = rs.getString("Projector");
            String reservationRequired = rs.getString("Reservation_Required");
            int outletCount = rs.getInt("Outlet_Count");

            if (isInteger(parts[0])) {
                infoArea.setText("강의실 번호: " + roomNumber + "\n강의실 이름: " + roomName +
                    "\n강의실 위치: " + location +
                    "\n좌석 수: " + seatCount +
                    "\n프로젝터: " + projector +
                    "\n예약 필요: " + reservationRequired +
                    "\n콘센트 개수: " + outletCount);
            } else {
                infoArea.setText("강의실 번호: " + roomNumber +
                    "\n강의실 이름: " + roomName +
                    "\n강의실 위치: " + location +
                    "\n좌석 수: " + seatCount +
                    "\n프로젝터: " + projector +
                    "\n예약 필요: " + reservationRequired +
                    "\n콘센트 개수: " + outletCount);
            }

            } else {
                JOptionPane.showMessageDialog(this, "해당 강의실의 추가 정보를 찾을 수 없습니다.");
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(this, "Database error: " + ex.getMessage());
    }
}

```

- 사용자가 입력한 강의 번호 또는 이름(parts[0])과 분반 번호(parts[1])를 사용하여 서버 쿼리를 실행한다. 서버 쿼리는 DB2024_LectureView 테이블에서 해당 조건에 맞는 강의실 번호를 찾아낸다.
- 찾아낸 강의실 번호는 외부 쿼리의 WHERE 절에서 사용되어 DB2024_ClassroomView에서 해당 강의실의 상세 정보를 가져온다.
- 결과적으로, 강의실의 상세 정보가 화면에 출력되거나 추가 정보가 없는 경우 사용자에게 알림이 제공된다.

(11) 조인 쿼리(join query)들을 가지는 쿼리들은 포함해야 한다.

- professor_button3 클래스의 getProfessorByRoomAndTime 메서드

:DB2024_professor 테이블과 DB2024_Lecture 테이블을 조인하여 DB2024_Lecture 테이블의 Room_Number로 교수명, 연구실위치, 전화번호, 이메일 정보를 가져온다.

```
// 강의실과 시간으로 교수 정보 가져오기 메서드
```

```

private List<ProfessorInfo> getProfessorsByRoomAndTime(String room, List<String> times)
throws SQLException {
    List<ProfessorInfo> professors = new ArrayList<>(); // 교수 정보 리스트 생성
    // SQL 쿼리 생성
    StringBuilder queryBuilder = new StringBuilder("SELECT DISTINCT p.Name,
p.Lab_Location, p.Phone, p.Email "
        + "FROM DB2024_Professor p "
        + "JOIN DB2024_Lecture l ON p.Professor_Num = l.Professor_Num "
        + "WHERE l.Room_Number = ? AND (");
    for (int i = 0; i < times.size(); i++) {
        queryBuilder.append("l.Lecture_Time1 = ? OR l.Lecture_Time2 = ?");
        if (i < times.size() - 1) {
            queryBuilder.append(" OR ");
        }
    }
    queryBuilder.append(")");
    PreparedStatement stmt = connection.prepareStatement(queryBuilder.toString()); //
준비된 명령문 생성
    stmt.setString(1, room); // 강의실 위치 설정
    int index = 2; // 시간 설정 인덱스 초기값
    for (String time : times) {
        stmt.setString(index++, time); // 강의 시간1 설정
        stmt.setString(index++, time); // 강의 시간2 설정
    }
}

```

(12) 매개 변수를 가지면서 동적으로 만드는 쿼리를 포함해야 한다. 다시 말해, 사용자로부터 입력 값을 받고 사용자가 입력한 값으로 쿼리를 생성한다.

- 관리자 클래스의 **insertTuple**, **updateTuple**, **deleteTuple** 메서드 모두 매개 변수를 가지면서 동적으로 만드는 쿼리를 포함한다. 그중 **insertTuple** 메서드를 예시로 들어 조건이 만족함을 보인다.

```

private void insertTuple() {
    String Lecture_Num = inLecture_Num.getText();
    String Class_Num = inClass_Num.getText();
    String Lecture_Name = inLecture_Name.getText();
    String Professor_Num = inProfessor_Num.getText();
    String Professor_Name = inProfessor_Name.getText();
    String Room_Number = inRoom_Number.getText();
    String Lecture_Time1 = inLecture_Time1.getText();
    String Lecture_Time2 = inLecture_Time2.getText();
    String query1 = "INSERT INTO DB2024_Lecture
(Lecture_Num,Class_Num,Lecture_Name,Professor_Num,Professor_Name,Room_Number,Lecture_Time1,
Lecture_Time2)" + " VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
    try (PreparedStatement pStmt1 = conn.prepareStatement(query1);) {
        try {
            conn.setAutoCommit(false); // 트랜잭션 시작
            pStmt1.setInt(1, Integer.parseInt(Lecture_Num));
            pStmt1.setInt(2, Integer.parseInt(Class_Num));
            pStmt1.setString(3, Lecture_Name);
            pStmt1.setInt(4, Integer.parseInt(Professor_Num));
            pStmt1.setString(5, Professor_Name);
            pStmt1.setString(6, Room_Number);
            pStmt1.setString(7, Lecture_Time1);
            pStmt1.setString(8, Lecture_Time2);
            System.out.println("강의 정보 삽입");
        }
    }
}

```

```
System.out.println(pStmt1); // 디버깅 위해 출력
pStmt1.executeUpdate();
```

- 사용자로부터 여러 입력 값을 받아 동적으로 쿼리를 생성하는 경우에는 사용자 입력을 안전하게 처리하고 각각의 ?는 **PreparedStatement** 에서 매개 변수로 사용된다.

(13) 그래픽 또는 문자 기반의 사용자 인터페이스를 사용해야 한다. 사용하기 쉽고 사용하기에 도움이 되는 정보를 가지고 있는 메뉴를 제공해야 한다.

공강 DB는 **Java GUI** 를 사용하여 그래픽 인터페이스를 제공한다.

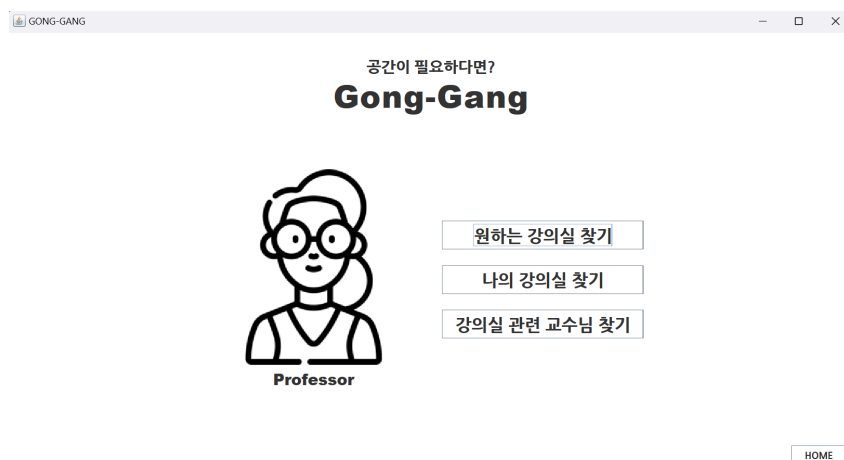
1. 메인화면 mainFame.java

사용자를 선택하는 인터페이스를 제공한다.



사용자는 교수, 학생, 관리자로 구성되고 각 버튼으로 구현하였다.

2. professor_choiceGUI.java / student_choiceGUI.java





교수 / 학생 목적에 맞는 버튼 선택 화면

3. professor_button1.java

사용자의 목적에 맞게 조건을 선택할 수 있도록 인터페이스 구현



시간 선택을 편리하게 해줄 수 있도록 타임 테이블 구현

4. professor_button2.java

사용자에게 더 편리한 인터페이스 제공을 위해 시간표 구현

공간이 필요하다면?

Gong-Gang

교수님 이름:

검색

현재 교시 정보

현재 교시 정보가 여기에 표시됩니다.

	월	화	수	목	금
1					
2					
3					
4					
5					
6					
7					
8					
9					

HOME

search했을 때, 현재 요일을 핑크색, 현재 시간을 빨간색으로 표시

공간이 필요하다면?

Gong-Gang

교수님 이름: 이병훈

검색

현재 교시 정보

수업 없음

	월	화	수	목	금
1				공학기기분석	고분자전자재료
2					
3		공학기기분석			고분자전자재료
4					
5					
6					
7					
8					
9					

HOME

5. professor_button3.java

GONG-GANG

Gong-Gang

- 교수님 찾기 -

강의 시간 선택

<input type="checkbox"/> 월1	<input type="checkbox"/> 화1	<input type="checkbox"/> 수1	<input type="checkbox"/> 목1	<input type="checkbox"/> 금1
<input type="checkbox"/> 월2	<input type="checkbox"/> 화2	<input type="checkbox"/> 수2	<input type="checkbox"/> 목2	<input type="checkbox"/> 금2
<input type="checkbox"/> 월3	<input type="checkbox"/> 화3	<input type="checkbox"/> 수3	<input type="checkbox"/> 목3	<input type="checkbox"/> 금3
<input type="checkbox"/> 월4	<input type="checkbox"/> 화4	<input type="checkbox"/> 수4	<input type="checkbox"/> 목4	<input type="checkbox"/> 금4
<input type="checkbox"/> 월5	<input type="checkbox"/> 화5	<input type="checkbox"/> 수5	<input type="checkbox"/> 목5	<input type="checkbox"/> 금5
<input type="checkbox"/> 월6	<input type="checkbox"/> 화6	<input type="checkbox"/> 수6	<input type="checkbox"/> 목6	<input type="checkbox"/> 금6
<input type="checkbox"/> 월7	<input type="checkbox"/> 화7	<input type="checkbox"/> 수7	<input type="checkbox"/> 목7	<input type="checkbox"/> 금7
<input type="checkbox"/> 월8	<input type="checkbox"/> 화8	<input type="checkbox"/> 수8	<input type="checkbox"/> 목8	<input type="checkbox"/> 금8
<input type="checkbox"/> 월9	<input type="checkbox"/> 화9	<input type="checkbox"/> 수9	<input type="checkbox"/> 목9	<input type="checkbox"/> 금9

강의실 위치 :

Search

HOME

GONG-GANG

Gong-Gang

- 수업 강의실 찾기 -

학수번호나 강의이름과 분반 번호를 입력하세요 (ex 20471-3, 데이터베이스-3)

Search

강의실 번호: B152

강의실 이름: B152

강의실 위치: 산공 1

좌석 수: 70

프로젝터: 있음

예약 필요: 예약 필요

콘센트 개수: 12

HOME

6. student_button1.java

GONG-GANG

Gong-Gang

- 원하는 강의실 찾기 -

☐ 콘센트

☐ 식사

☐ 빔프로젝터

☐ 컴퓨터

공간 유형 :

선택

좌석 수 :

선택

원하는 교시 선택

<input type="checkbox"/> 월 1	<input type="checkbox"/> 화 1	<input type="checkbox"/> 수 1	<input type="checkbox"/> 목 1	<input type="checkbox"/> 금 1
<input type="checkbox"/> 월 2	<input type="checkbox"/> 화 2	<input type="checkbox"/> 수 2	<input type="checkbox"/> 목 2	<input type="checkbox"/> 금 2
<input type="checkbox"/> 월 3	<input type="checkbox"/> 화 3	<input type="checkbox"/> 수 3	<input type="checkbox"/> 목 3	<input type="checkbox"/> 금 3
<input type="checkbox"/> 월 4	<input type="checkbox"/> 화 4	<input type="checkbox"/> 수 4	<input type="checkbox"/> 목 4	<input type="checkbox"/> 금 4
<input type="checkbox"/> 월 5	<input type="checkbox"/> 화 5	<input type="checkbox"/> 수 5	<input type="checkbox"/> 목 5	<input type="checkbox"/> 금 5
<input type="checkbox"/> 월 6	<input type="checkbox"/> 화 6	<input type="checkbox"/> 수 6	<input type="checkbox"/> 목 6	<input type="checkbox"/> 금 6
<input type="checkbox"/> 월 7	<input type="checkbox"/> 화 7	<input type="checkbox"/> 수 7	<input type="checkbox"/> 목 7	<input type="checkbox"/> 금 7
<input type="checkbox"/> 월 8	<input type="checkbox"/> 화 8	<input type="checkbox"/> 수 8	<input type="checkbox"/> 목 8	<input type="checkbox"/> 금 8

검색

HOME

검색된 교실의 번호:
A125-2 월4가능

Gong-Gang

- 결과 -

7. student_button2.java

GONG-GANG

Gong-Gang

- 수업 강의실 찾기 -

학수번호나 강의이름과 분반 번호를 입력하세요 (ex 20471-3, 데이터베이스-3)

Search

HOME

GONG-GANG

Gong-Gang

- 수업 강의실 찾기 -

학수번호나 강의이름과 분반 번호를 입력하세요 (ex 20471-3, 데이터베이스-3)

데이터베이스-3

Search

강의 이름: 데이터베이스

강의실 번호: B102

강의실 이름: B102

강의실 위치: 신공 1

HOME

8. student_button3.java

시간표를 구현하여 현재 요일, 시간에 대한 정보 제공

GONG-GANG

Gong-Gang

- 교수님 찾기 -

교수님 이름을 입력하세요

1					
2					
3					
4					
5					
6					

HOME

GONG-GANG

Gong-Gang

- 교수님 찾기 -

교수님 이름을 입력하세요

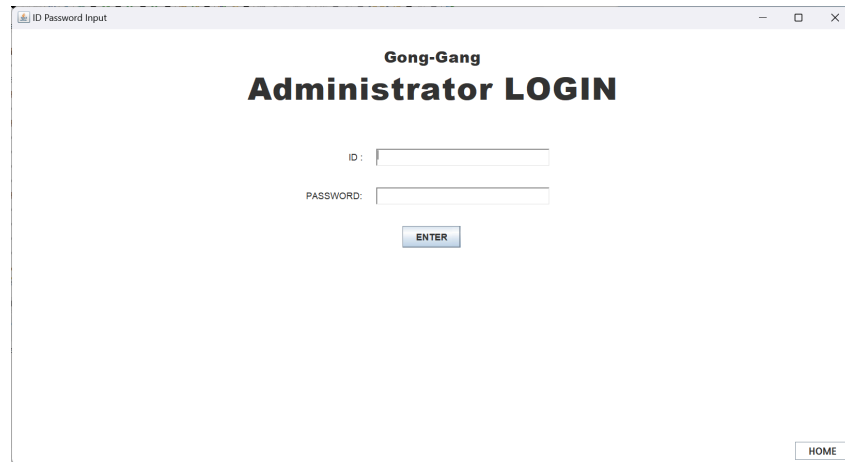
Email: yj.kim@ewha.ac.kr
Lab_Location: 동강201-2
Phone Number: null

1					
2					
3					
4		B152			
5	B152				
6					

HOME

9. Administrator.java

아이디, 비밀번호 입력할 수 있는 그래픽 인터페이스 제공



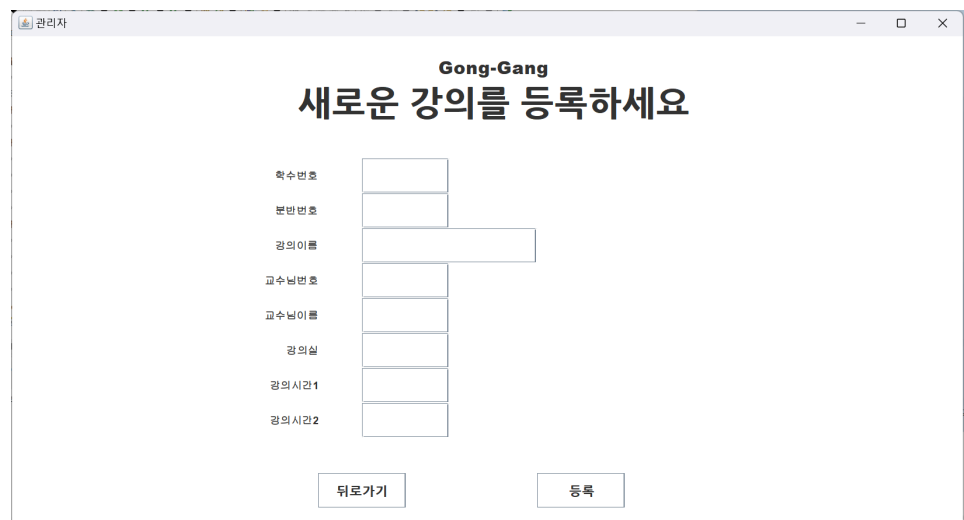
A screenshot of a Java Swing window titled "ID Password Input". The window has a title bar with standard OS controls. The main content area has a background image of a person. At the top, it says "Gong-Gang" and "Administrator LOGIN" in large, bold, black letters. Below this, there are two input fields: "ID:" and "PASSWORD:". Below the password field is a blue button labeled "ENTER". At the bottom right, there is a small button labeled "HOME".



A screenshot of a Java Swing window titled "관리자" (Administrator). The window has a title bar with standard OS controls. The main content area has a background image of a person. At the top, it says "Gong-Gang" and "강의등록" (Lecture Registration) in large, bold, black letters. Below this, there is a section titled "강의 전체 보기" (View All Lectures). Below this section is a table with 8 columns: Lecture_Num, Class_Num, Lecture_Name, Professor_Name, Professor_Num, Room_Number, Lecture_Time1, and Lecture_Time2. The table contains 20 rows of data. At the bottom right, there is a small button labeled "HOME".

Lecture_Num	Class_Num	Lecture_Name	Professor_Name	Professor_Num	Room_Number	Lecture_Time1	Lecture_Time2
11155	1	과학과학구	손아정	76	B159	목3	월2
14336	1	운정제재특론	박종경	32	B158	목6	목7
14349	1	메이타웨어하위징	이민수	46	B154	수6	수7
14674	1	영상부호환	장재원	39	B155	수6	수7
14735	1	양자소자	조성재	34	B158	화7	화8
14892	1	시스템수학			B156	화2	화3
15161	1	고급연출설계1	권민영	38	B155	목6	목6
15185	1	천더와공간연구	김미선	48	B154	화2	화3
15203	1	친환경하이테크건축	이진영	37	B155	목8	목9
15268	1	환경화학특론			B156	월6	월7
16159	1	토보학개론			B156	금6	금7
16453	1	원전설개론	이준성	65	B154	목5	목6
16460	1	구름과강수분석	최용상	35	B158	월4	월5
17609	1	첨단ICT와도시시스템			B156	수2	수3
17652	1	도시개발분석	김단아	49	B154	월4	월5
17706	1	정보보호론	배호	61	B154	수2	수3
17710	1	차세대보안특론	김종길	60	B158	목4	목5
17868	1	사이버보안특론3	양대한	110	B158	목2	목3
17985	1	양자계산과양자학			B156	금4	금5
18143	1	비서열수치해석	김철구	100	B155	화5	화6
18166	1	로봇의물리적상호작용			B156	목4	목5
18236	2	전력시장설계및해석	허진	36	B158	월2	월3
18271	1	고분자전자재료	이병훈	121	B155	금2	금3
18272	1	유전공학특론	박시재	120	B155	목2	목3

MySQL을 모르는 관리자도 쉽게 데이터베이스를 관리할 수 있도록 구현



A screenshot of a Java Swing window titled "관리자" (Administrator). The window has a title bar with standard OS controls. The main content area has a background image of a person. At the top, it says "Gong-Gang" and "새로운 강의를 등록하세요" (Register New Lecture) in large, bold, black letters. Below this, there are several input fields for registration: "학수번호" (Course Number), "분반번호" (Section Number), "강의이름" (Lecture Name), "교수님번호" (Professor Number), "교수님이름" (Professor Name), "강의실" (Lecture Room), "강의시간1" (Lecture Time 1), and "강의시간2" (Lecture Time 2). At the bottom, there are two buttons: "뒤로가기" (Go Back) and "등록" (Register).

(14) 데이터베이스에 삽입(insert)을 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.

Java GUI를 사용하여 삽입(insert)을 하기 위한 입력창 인터페이스를 제공한다.

Java GUI에서 입력창에서 입력받은 값을 매개변수로 INSERT문에 넣어 삽입(insert)을 하기 위한 쿼리를 생성하고 실행한다. 아래는 코드 일부이다.

```
private void insertTuple() {
    String Lecture_Num = inLecture_Num.getText();
    String Class_Num = inClass_Num.getText();
    String Lecture_Name = inLecture_Name.getText();
    String Professor_Num = inProfessor_Num.getText();
    String Professor_Name = inProfessor_Name.getText();
    String Room_Number = inRoom_Number.getText();
    String Lecture_Time1 = inLecture_Time1.getText();
    String Lecture_Time2 = inLecture_Time2.getText();
    String query1 = "INSERT INTO DB2024_Lecture (Lecture_Num, Class_Num,
Lecture_Name, Professor_Num, Professor_Name, Room_Number, Lecture_Time1, Lecture_Time2)"
+ " VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt1 = conn.prepareStatement(query1);) {
        try {
            conn.setAutoCommit(false); // 트랜잭션 시작
```

```

pStmt1.setInt(1, Integer.parseInt(Lecture_Num));
pStmt1.setInt(2, Integer.parseInt(Class_Num));
pStmt1.setString(3, Lecture_Name);
pStmt1.setInt(4, Integer.parseInt(Professor_Num));
pStmt1.setString(5, Professor_Name);
pStmt1.setString(6, Room_Number);
pStmt1.setString(7, Lecture_Time1);
pStmt1.setString(8, Lecture_Time2);
System.out.println("강의 정보 삽입");
System.out.println(pStmt1); // 디버깅 위해 출력
pStmt1.executeUpdate();

```

(15) 데이터베이스에 갱신(update)을 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.

Java GUI를 사용하여 테이블을 클릭하여 입력받은 값을 제공하고 갱신(update)을 하기 위한 인터페이스와 버튼을 제공한다.

Java GUI에서 테이블을 클릭하여 입력받은 값을 매개변수로 UPDATE문에 넣어 갱신(update)을 하기 위한 쿼리를 생성하고 실행한다. 아래는 코드 일부이다.

```

private void updateTuple() {
    String Lecture_Num = inLecture_Num.getText();
    String Class_Num = inClass_Num.getText();
    String Lecture_Name = inLecture_Name.getText();
    String Professor_Num = inProfessor_Num.getText();
    String Professor_Name = inProfessor_Name.getText();
    String Room_Number = inRoom_Number.getText();
    String Lecture_Time1 = inLecture_Time1.getText();
    String Lecture_Time2 = inLecture_Time2.getText();
    String query1 = "UPDATE DB2024_Lecture " + "SET Lecture_Num = ?, Class_Num = ?,
Lecture_Name = ?, Professor_Num = ?, Professor_Name = ?, Room_Number = ?, Lecture_Time1 =
?, Lecture_Time2 = ? " + "WHERE Lecture_Num = ? AND Class_Num = ?";

    try (PreparedStatement pStmt1 = conn.prepareStatement(query1);
        PreparedStatement pStmt2 = conn.prepareStatement(query2);
        PreparedStatement pStmt3 = conn.prepareStatement(query3);) {
        try {
            conn.setAutoCommit(false); // 트랜잭션 시작
            pStmt1.setInt(1, Integer.parseInt(Lecture_Num));

```

```

pStmt1.setInt(2, Integer.parseInt(Class_Num));
pStmt1.setString(3, Lecture_Name);
pStmt1.setInt(4, Integer.parseInt(Professor_Num));
pStmt1.setString(5, Professor_Name);
pStmt1.setString(6, Room_Number);
pStmt1.setString(7, Lecture_Time1);
pStmt1.setString(8, Lecture_Time2);
pStmt1.setInt(9, Integer.parseInt(Lecture_Num_ori));
pStmt1.setInt(10, Integer.parseInt(Class_Num_ori));
System.out.println(pStmt1); // 디버깅 위해 출력
pStmt1.executeUpdate();

```

(16) 데이터베이스에 삭제(delete)를 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.

Java GUI를 사용하여 테이블을 클릭하여 입력받은 값을 제공하고 삭제(delete)을 하기 위한 인터페이스와 버튼을 제공한다.

Java GUI에서 테이블을 클릭하여 입력받은 값을 매개변수로 DELETE문에 넣어 삭제(delete)을 하기 위한 쿼리를 생성하고 실행한다. 아래는 코드 일부이다.

```

private void deleteTuple() {
    String Lecture_Num = inLecture_Num.getText();
    String Class_Num = inClass_Num.getText();
    String Lecture_Name = inLecture_Name.getText();
    String Professor_Num = inProfessor_Num.getText();
    String Professor_Name = inProfessor_Name.getText();
    String Room_Number = inRoom_Number.getText();
    String Lecture_Time1 = inLecture_Time1.getText();
    String Lecture_Time2 = inLecture_Time2.getText();
    String query1 = "DELETE FROM DB2024_Lecture WHERE Lecture_Num = ? AND
Class_Num = ?";
    try (PreparedStatement pStmt1 = conn.prepareStatement(query1)) {
        conn.setAutoCommit(false); // 트랜잭션 시작
        // 첫 번째 쿼리 실행
        pStmt1.setInt(1, Integer.parseInt(Lecture_Num_ori));
        pStmt1.setInt(2, Integer.parseInt(Class_Num_ori));
        System.out.println(pStmt1);
        pStmt1.executeUpdate();
    }
}

```

(17) 데이터베이스에 검색(select)을 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.

- professor_button2클래스의 getProfessorInfo 메서드

: DB2024_Professor 에서 정보 검색

```
//교수 정보 가져오기 메서드
private ProfessorInfo getProfessorInfo(String professorName) throws SQLException{
    ProfessorInfo professorInfo = null;
    String professorQuery = "SELECT Name, Lab_Location, Phone, Email FROM DB2024_Professor
WHERE Name = ?"; // 교수 정보 쿼리
    PreparedStatement professorStmt = connection.prepareStatement(professorQuery); // 쿼리
    준비
    professorStmt.setString(1, professorName); // 검색어 설정
    ResultSet professorRs = professorStmt.executeQuery(); // 쿼리 실행
    if (professorRs.next()) {
        professorInfo = new ProfessorInfo(); // 교수 정보 객체 생성
        professorInfo.setName(professorRs.getString("Name")); // 이름 설정
    }
    return professorInfo; // 교수 정보 반환
}
```

: 교수님 검색 인터페이스 : 유저에게 교수님 이름을 입력할 수 있는 검색창을 제공한다.

공간이 필요하다면?

Gong-Gang

교수님 이름:

	월	화	수	목	금
1					
2					
3					
4			데이터베이스		
5	데이터베이스				
6					
7					
8					
9					

7. 요구조건 외의 팀만의 강점

- 이화여대 공학관에 개설된 실제 **24-1**학기 수업을 직접 조사해서 만들었기 때문에 설계한 **DB**와 인터페이스는 실제 사용 가능하다.
- DB**를 수정하여 다음 학기 재사용이 가능하다.
- DB**구축 시 정규화 과정을 진행했으며 공간적 효율성을 높였다.
- 중간고사 때 시험 볼 적절한 빈 강의실을 찾고 계시는 교수님의 모습을 보았는데, 우리 팀의 **DB**와 인터페이스를 사용하면 해당 문제를 해결할 수 있다.
- 사용자 친화적인 인터페이스 구현으로 편리한 사용이 가능하다.
 - 사용자 이미지를 넣어 직관적 선택을 가능하게 함
 - **DB**를 모르는 사용자가 사용할 수 있도록 버튼과 검색 등을 구성함
 - 사용자별로 필요할 수 있는 기능들을 구상하여 나누었기에 필요한 기능을 바로 찾을 수 있게 함. (어디에 어떤 기능이 있나 고민하지 않아도 됨)
 - 관리자가 테이블에서 선택으로 바로 삭제 가능하도록 하였음.
 - 교수님이나 학생들이 현재 자신의 시간을 기반과 찾은 정보를 비교할 수 있도록 현재 시간과 요일에 색을 칠해 표시해 주었으며 정보를 시각화 하여 빠르게 처리할 수 있도록 테이블 형태를 사용함.

8. 팀의 구성원의 담당 부분

팀원 이름	SQL	Java Code	Report	발표	데모 비디오	기타
이주원	- create, insert문 작성 - index생성	교수님 대상 - 강의공간 찾기 - 강의실찾기 - 교수님 검색	- 스키마 다이어그램 - 학생 코드 설명 - Index 설명 -요구사항 (6),(7),(10)			- 신공학관 2층 강의실 조사 -요구사항명 세서(교수) 작성
조수현	-create, insert문 작성	관리자 대상 -보안용 코드 -테이블 관리 -강의 정보 화면	-요구분석 -관리자 코드 설명 -요구사항 (9), (12), (14), (15), (16)	- 중간발표 : DB목적과 요구명세서		-아산공학관 1층 강의실 조사
박민서	-create, insert문 작성	교수님 대상 - 강의실 찾기 - 강의 찾기 - 교수 찾기	- 교수님 코드 설명 - 요구사항 (8), (17)	- 중간발표 : 스키마와 개발일정 - 기말발표 : 프로그램 시연	- 데모 비디오 제작	- 신공학관 B1, B2층 강의실 조사
강민서	-create, insert문 작성 - view 생성	GUI 구현 - 관리자 버튼 연결 및 디자인	-ER다이어그램 -View 설명	-중간발표 : ppt 제작		- 신공학관 1층 강의실 조사
김희진	-create, insert문 작성	GUI 구현 - 메인 화면 - choice 화면 - 학생, 교수 버튼 연결 및 디자인	- GUI - 요구사항(1), (2), (13)	- 기말발표 : ppt 제작, DB목적과 프로그램 기능		- 아산공학관 B1, 2층 강의실 조사 -요구사항명 세서(학생) 작성