

# 중간 결과 보고서

명지대학교 - 클라이언트 서버 프로그래밍

이름: 노태영

학번: 60211660



## 보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁 받지도 않겠습니다.

나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.



과목 : 클라이언트 서버 프로그래밍

담당 교수 : 김정호

작성자: 노태영

## - 목차 -

1. 프로젝트 개요 -----
  - 1-1 프로젝트 배경
  - 1-2 기능 구현 범위 및 사용 기술
2. 시스템 설계 -----
  - 2-1 UseCaseDiagram 과 시스템의 동작방식
  - 2-2 Entity-Relationship Diagram 과 데이터 베이스의 구조
3. 코드 설명 -----
  - 3-1 로그인, 토큰발행과 저장, 로깅
  - 3-2 학생 Entity CRUD
  - 3-3 강좌 Entity CRUD
  - 3-4 학생 계정 Entity CRUD
  - 3-5 수강신청, 삭제
4. 프로젝트 결과와 소감 -----

# 1. 프로젝트 개요

## 1-1 프로젝트 배경

클라이언트-서버 프로그램은 Inter Process Communication 의 일종으로 자원을 공유하지 않는 독립적인 Independent Context 가 서로 통신한다. 둘 이상의 Process 가 마치 하나의 Process 가 동작하는 것처럼 작동하는 프로그램이다. 클라이언트와 서버는 서로 다른 물리적 위치에서 동작하는 Independent Context 이며 클라이언트는 서버에게 서비스를 요청하고 서버는 클라이언트에게 서비스를 제공한다. 두 프로그램이 통신하기 위해서는 물리적으로 서로 다른 위치에 존재하는 두 프로그램을 연결해야 한다. 두 프로그램 즉 시스템이 연결되어 서비스를 요청하고 제공하는 형태를 네트워크라고 부르며 서버와 클라이언트 모두 OSI 7 Layer 라는 논리적인 네트워크의 표준화된 개념을 따른다. OSI 7 Layer 의 상위 계층이라고 불리는 Applicaiton, Presentation, Session 계층은 응용 프로그램에서 동작하고 하위 계층이라고 불리는 Transport, Network, DataLink, Physical 계층은 OS 를 기반으로 동작한다. 이러한 개념에 기반해 서버와 클라이언트 프로그램은 통신을 위하여 정해진 규칙인 프로토콜을 따라야 하고 두 시스템이 오류없이 서비스를 요청하고 응답하는 통신을 통해 클라이언트-서버 프로그램을 구현해야 한다.

두 시스템이 연결되어 통신하기 위해선 상위 계층의 연결과 하위 계층의 연결이 필요하다. 일반적으로 하위 계층의 연결은 TCP/IP 라고 하는 IP 주소와 Port 넘버를 통하여 인터넷 상의 유일무이한 주소를 기반으로 연결되며 하위계층의 연결은 OS 가 담당한다. 상위 계층의 연결은 클라이언트의 응용 프로그램과 서버의 응용 프로그램을 연결하는 것이고 이를 위해서 사용되는 기술이 Broker 이다. Broker 기술을 기반으로 서버와 클라이언트의 응용프로그램을 연결하기 위한 다양한 라이브러리와 프레임워크가 존재한다. CORBA, COM, RMI 등등 다양한 기술이 존재하지만 현 프로젝트에서는 google remote procedure call 이라고 불리는 gRPC 를 이용하여 클라이언트와 서버 프로그램을 구현하고자 한다.

위 개념을 기반으로 구현하고자 하는 프로그램은 수강신청 프로그램이다. 해당 프로그램에서 클라이언트가 서버에게 요청하는 서비스는 데이터 CRUD 작업이다. 로그인, 강좌 신청, 강좌 삭제 등 클라이언트는 서버에게 데이터를 변경하거나 읽는 CRUD 작업을 요청하고 서버는 해당 요청에 맞게 동작하여 결과를 리턴한다. 해당 작업의 수월한 동작을 위해서 FileSystem 대신 Database 를 이용할 것이며 오라클 DBMS 를 사용한다. Database 를 통하여 작업한다면 데이터 무결성을 지켜 데이터를 더욱 완전하고 정확하게 처리할 수 있다. 또한 더 수월하고 다양한 기능의 데이터 CRUD 작업 처리가 가능하다. 따라서 현 프로젝트에서는 Broker 기술로는 gRPC 를 데이터 작업을 위해서는 오라클로 Database 를 구현하여 수강신청 프로그램을 완성하고자 한다.

## 1-2 기능 구현 범위 및 사용 기술

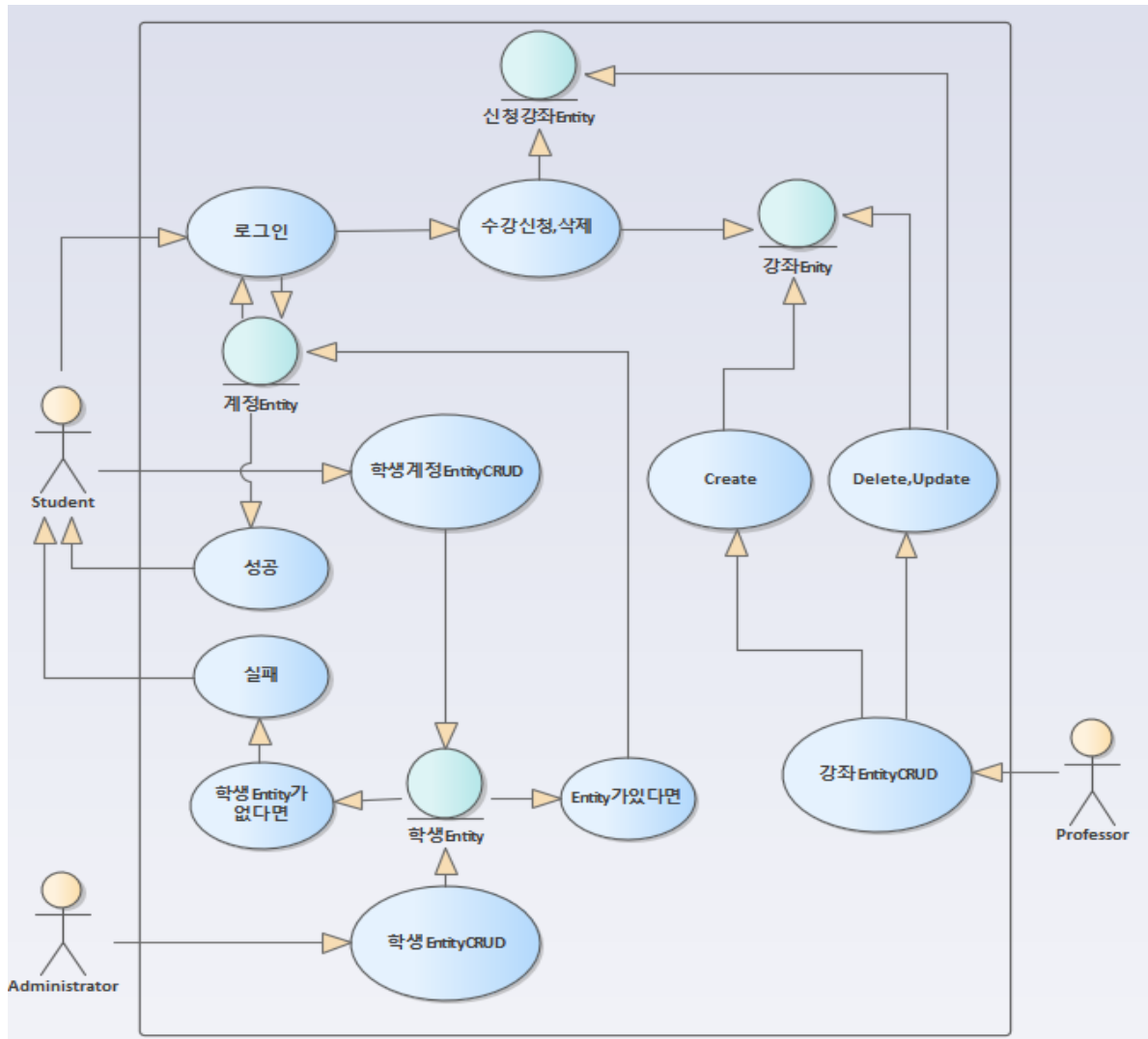
현재 목표로 하는 프로젝트의 구현 범위는 다음과 같다.

- 클라이언트 프로그램을 3 개로 구분하여 학생용, 교수용, 관리자용 클라이언트 프로그램을 구현하고 하나의 서버로 3 개의 클라이언트를 처리한다.
- 로그인 기능 구현
  - 3 개의 모든 프로그램은 (학생, 교수, 관리자) 로그인 기능을 이용하여 서버에게 토큰을 발행받고 다음 작업을 진행 할 수 있다.
- 관리자 프로그램 기능 구현
  - 관리자 클라이언트는 학생 Entity 를 CRUD 할 수 있고 교수 프로그램에서 생성된 강좌와 학생을 조건(학생의 성씨나 성별, 강좌의 학점이나 수강신청한 학생 인원수 등등)을 설정하여 데이터를 읽을 수 있다.
- 학생 프로그램 기능 구현
  - 학생 클라이언트는 학생 계정 Entity 를 CRUD 할 수 있다. 현 프로젝트에서는 학생과 학생 계정을 서로 다른 데이터(Table)로 구현하고자 하며 관리자가 추가한 학생만 학생 계정을 생성할 수 있도록 한다. 즉 Database 에 학생 Entity 가 존재해야 학생 계정 Entity 를 생성할 수 있는 Total Participation 관계이다.
  - 학생 계정을 통해서 로그인 했다면 학생은 교수 프로그램에서 생성한 강좌 Entity 를 선택하여 수강신청 내역에 추가하거나 수강신청 내역에서 특정 강좌를 삭제할 수 있다. 강좌를 선택하는 과정에서 (동일 강좌 선택, 2 개를 초과하는 재수강과목 신청 등등) 총 5 개의 제약조건을 설정하고 해당 조건에 맞게 강좌를 선택할 수 있도록 구현하고자 한다.
- 교수 프로그램 기능 구현
  - 교수 클라이언트는 로그인 이후 강좌 CRUD 작업이 가능하다. 교수가 삭제한 강좌는 학생의 수강신청 내역에서도 삭제될 수 있도록 하며 본인이 개설한 강좌를 신청한 학생 정보를 읽어올 수 있다.
- 서버 기능 구현

- 학생, 교수, 관리자 클라이언트는 서로 다른 동작을 하며 다른 목적을 가지지만 서버에게 요청하는 기능이 굉장히 유사하다. 모든 프로그램은 로그인을 통하여 서버에게 토큰을 발행받고, 토큰을 제공하며 Entity Crud 작업을 요청한다. 예를 들어 모든 프로그램은 서버에게 강좌내역을 요청한다. 서로 다른 3 개의 클라이언트이지만 서버에게 보내는 요청과 서버가 보내야하는 응답의 형태가 매우 유사하기 때문에 하나의 서버로 3 개의 클라이언트를 처리하는 효율적인 방향으로 시스템을 구현하고자 한다.
- 효율적으로 해당 작업을 수행하기 위해서 유사한 요청과 응답이 필요한 서비스를 3 대의 클라이언트에게 하나의 API 로 처리하고자 한다. 예를 들어 3 개의 클라이언트가 모두 필요로 하는 로그인 기능이나 강좌 내역을 요청하는 기능은 하나의 API 로 구현하여 내부에서 경우를 나눈다. 이를 통해 코드의 양을 줄이고 효율적인 처리가 가능하도록 설계하고자 한다.

## 2. 시스템 설계

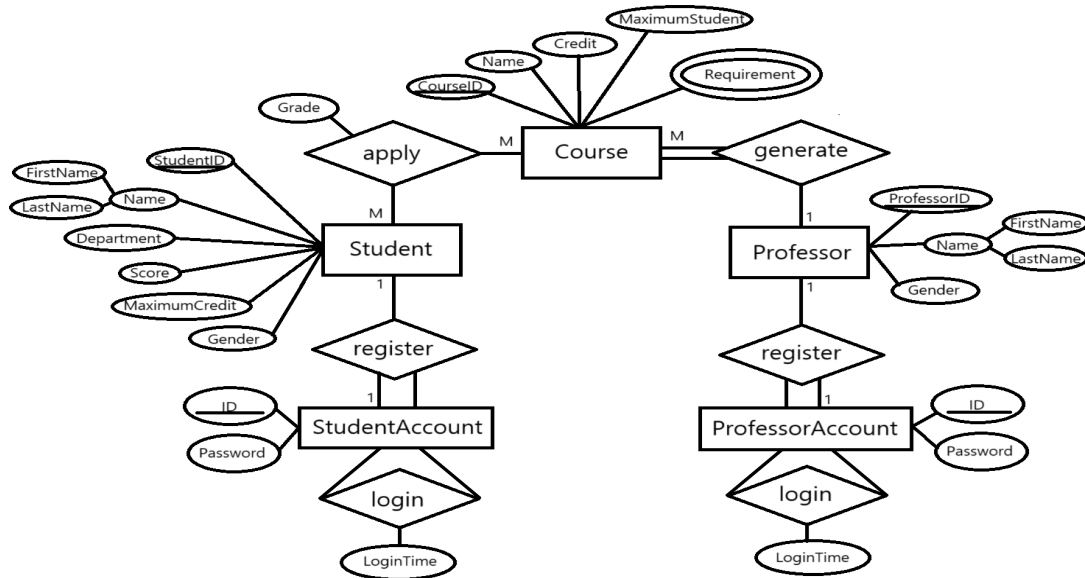
### 2-1 UseCaseDiagram 과 시스템의 동작방식



위 UseCaseDiagram 은 시스템이 동작하는 알고리즘을 나타내는 다이어그램이다. 각각의 Actor 는 서로 다른 3 개의 학생, 교수, 관리자 프로그램이다. 서버는 서로 다른 클라이언트를 하나의 서버로 모두 처리한다.(네모 박스는 서버) 각각의 프로그램은 사용자의 동작을 통해 Database 에 존재하는 Entity CRUD 작업을 서버에게 요청한다.

그 과정에서 서버는 데이터 무결성을 지키며 데이터를 처리하고 BusinessRule 을 기반으로 사용자의 동작을 제어하며 그에 따른 응답을 전송한다. (BusinessRule 은 예를들어 클라이언트가 2 개를 초과하는 재수강 과목을 신청할 경우 서버가 요청을 막아야 하는 상황)

## 2-2 Entity-Relationship Diagram 과 데이터 베이스의 개념 및 구조



위 Original-ER 다이어그램은 전반적인 프로그램의 동작과 더불어 전체적인 Database 의 개념을 ER 다이어그램으로 모델링한 것이다. 위 다이어그램은 전체적인 개념을 나타낸 것으로 실제 Database 가 저장된 물리적 구조인 Internal Schema 와 약간의 차이를 보인다.

실제 구현한 Database 에서는 Course 를 해당 학기의 강좌를 담는 Course 테이블과 이전 학기까지 강좌를 담는 PreviousCourse 테이블로 구분하였다. 두 테이블의 Attribute 에는 차이가 없지만 더 다양한 기능을 구현하기 위하여 테이블을 두 개로 나눈 것이다.

Student, Course 는 다대다 Cardinality Constraint 이기 때문에 두 테이블을 구현하는 단계에서는 ApplicationList 테이블을 추가하였다. ApplicationList 또한 동일한 Attribute 를 가지는 AttendedCourse\_List 테이블을 만들어 이전 수강내역을 담아두는 테이블을 추가했다. (수강신청의 선수과목 여부와 재수강 여부를 판단하기 위하여) 학생이 강좌를 신청하면 Course 에 존재하는 Entity 를 ApplicationList 에 추가한다. PreviousCourse 에 존재하는 Entity 는 AttendedCourse\_List 테이블에 추가된다.

Professor 의 강좌 CRUD 작업은 CourseEntity 를 대상으로 동작하며 PreviousCourse 와 AttendedCourseList 는 데이터 Retrieve 와 BusinessRule 을 (2 개를 초과하는 재수강 신청을 막는 BusinessRule, 선수과목에 대한 이수 여부를 따지는 BusinessRule 등등) 충족하기 위해 존재하는 테이블이다. 더 자세한 구현은 코드와 실행 결과를 통한 다음장에서 설명하고자 한다.



## 3. 코드와 실행결과

해당 목차에서는 기능에 따라 어떻게 API 를 구현하였으며 클라이언트가 특정 기능을 사용하기 위해서 요청을 보내는 코드와 과정, 서버가 요청을 받아 처리하고 응답을 보내는 일련의 과정을 기능별로 나눠서 설명하도록 하겠다.

### 3-1. 로그인, 토큰발행과 저장, 로깅

로그인은 학생, 교수, 관리자 3 개의 모든 프로그램에서 요청하는 기능이다. 각각의 프로그램은 모두 서로 다른 Database 테이블에 계정 데이터를 저장하고 있다. 따라서 서버는 해당 로그인 요청이 어떤 프로그램으로 부터 전송된 것인지 구분할 수 있어야 한다. 클라이언트가 전송한 Id, Password 를 해당 사용자에게 맞는 Database 계정 테이블로 이동하여 정확한 입력 여부를 판단하고 클라이언트에게 응답을 전송해야 한다.

로그인 API 구현

```
@Override
public void login(LoginRequest request, StreamObserver<LoginResponse> responseObserver) {
    LoginResponse response = null;
    try {
        response = makeLoginResponse(request.getId(), request.getPassword(), request.getProcess());
        logging(request.getId(), request.getProcess());
    } catch (NullPointerException nullDataError) {
        response = LoginResponse.newBuilder().setResult(nullDataError.getMessage()).build();
    } catch (ServerErrorException serverError) {
        response = LoginResponse.newBuilder().setResult(serverError.getMessage()).build();
    } catch (ExecuteQueryException queryError) {
        response = LoginResponse.newBuilder().setResult("ServerError").build();
    } finally {
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}

private LoginResponse makeLoginResponse(String id, String password, String process) throws NullPointerException, ServerErrorException,
Object object= accountDao.login(id, password, process);
String token = tokenGenerator.generateToken(object);

if(process.equals("Admin")) return LoginResponse.newBuilder().setToken(token).setResult("Success").build();
else if(process.equals("Student")){
    Student student = (Student) object;
    return LoginResponse.newBuilder().setToken(token).setResult("Success")
        .setName(student.getFirstName()+student.getLastName()).setObjectId(student.getStudentId()).build();
}
else if(process.equals("Professor")){
    Professor professor = (Professor) object;
    return LoginResponse.newBuilder().setToken(token).setResult("Success")
        .setName(professor.getFirstName()+professor.getLastName()).setObjectId(professor.getProfessorId()).build();
}
else throw new ServerErrorException("ServerError");
}

private void logging(String id, String process) {
    for (int i = 0; i < 10; i++) {
        if (logging.logging(id, process)) break;
        else System.out.println(i + 1 + "번 로깅 실패/재반복");
    }
}
```

클라이언트의 요청과 서버의 응답 (학생 프로그램)

```
*****LOGIN*****
ID: Jason
Password: Lee
LeeJason님 환영합니다.
*****MENU*****
```

```
ID: LoginFail
Password: LoginFail
입력한 ID와 PASSWORD에 맞는 계정이 없습니다.
재시도 하시겠습니까?
1: 재시도
2: 로그인 종료
입력:
```

위 사진은 서버에서 구현한 로그인 API 와 학생 클라이언트가 해당 API 를 사용하여 요청을 보내는 코드와 코드 실행 결과이다. 관리자와 교수 프로그램 모두 로그인 기능을 사용하기 위한 동일한

구조를 가지며 ID, Password 를 사용자에게 입력받고 입력값과 함께 프로그램 이름을 서버에게 전송한다. 서버는 클라이언트가 학생, 교수, 관리자인 경우를 나누고 Dao 를 통해서 데이터베이스에 접근하여 클라이언트가 전달한 ID, Password 가 올바른지 판단한다.

서버는 하나의 로그인 API 를 구현하고 내부에서 경우를 나눈다. 따라서 모든 프로세스가 동일한 API 를 사용해 로그인 기능을 요청할 수 있는 것이다. 서버에서는 전달받은 ID, Password 를 AccountDao 에 전달하고 클라이언트에 따라서 서로 다른 계정 테이블에 접근할 수 있도록 적절한 쿼리를 만들어 이를 실행한다.

AccountDao 코드(클라이언트가 학생인 경우)

```
if(process.equals("Student")) {
    query = constant.getStudentLoginQuery(id, password);
    loginobjectResultSet = super.retrieve(query);
    if(loginobjectResultSet == null) throw new NullDataException("NullData");
    query = constant.getAttendedCourseQuery(id, password);
    attendedCourseResultSet = super.retrieve(query);
    return getStudent(loginobjectResultSet, attendedCourseResultSet);
}
```

Dao 코드 (retrieve)

```
public ResultSet retrieve(String query) throws ExecuteQueryException, ServerError {
    try {
        System.out.println(query);
        statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(query);
        if(resultSet == null) throw new ServerErrorException("ServerError");
        if(resultSet.next()) return resultSet;
        else return null;
    } catch (SQLException sqlError) {
        throw new ExecuteQueryException(sqlError.getMessage());
    }
}
```

위 사진은 각각 AccountDao 와 Dao(부모 클래스)의 코드이다. id 와 password 로 쿼리를 만들어 실행하고 실행 결과집합을 가져온다. StudentEntity 의 Attribute 와 AttendedCourse\_List 테이블에서 학생의 데이터를 결과집합으로 가져오고 getStudent() 메서드를 통해 학생 ValueObject 를 생성한다. (이전 수강내역은 변하지 않는 정적인 값이기 때문에 로그인 시 데이터를 가져와 필드로 초기화 함)

로그인이 성공하여 클라이언트 정보의 ValueObject 를 생성한 이후에는 추가적으로 logging 작업과 토큰을 발행해야 한다. 로그인 이후 클라이언트는 요청을 보낼 때 토큰을 데이터와 같이 보내며 서버는 전달받은 토큰의 유효시간이 지나지 않았는지를 판단하고, 서버의 HashMap 에 저장해둔 토큰인지 여부를 판단하여 오류가 없다면 클라이언트에게 서비스를 제공하고 오류가 있다면 "TokenError"를 응답한다.

logging 코드(로그인한 ID 와 시간을 저장)

```
public boolean logging(String id, String process) {
    try {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        String now = simpleDateFormat.format(new Date());
        File file = null;
        if(process.equals("Student")) file = new File("FileData/StudentLogging");
        else if(process.equals("Professor")) file = new File("FileData/ProfessorLogging");
        else if(process.equals("Admin")) file = new File("FileData/AdminLogging");
        String loggingLine = id + " " + now;
        FileWriter fileWriter = new FileWriter(file, true);
        BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
        bufferedWriter.write(loggingLine);
        bufferedWriter.newLine();
        bufferedWriter.close();
        return true;
    }
}
```

logging 결과 (학생 로깅 파일)

```
1 Minsu 2023-10-16 21:04:20
2 Jason 2023-10-17 15:01:42
3 Kitea 2023-10-17 21:33:07
4 Minsu 2023-10-17 21:39:07
5 Minsu 2023-10-18 22:15:42
6 Kitea 2023-10-22 12:09:06
7 Minsu 2023-10-22 12:12:41
8 Kyungmin 2023-10-22 12:13:38
9 Kyungmin 2023-10-22 12:19:01
0 Jason 2023-10-22 14:52:30
1 Jason 2023-10-22 14:53:07
```

로그인 성공 시 logging 메서드를 사용하여 파일에 계정 ID 와 로그인 시간을 저장하는 코드 결과

자바 RMI 에서는 클라이언트와 서버가 객체를 주고받을 수 있지만 gRPC 에서는 프로토콜 버퍼라는 마크업 언어를 사용하여 데이터를 직렬화 역직렬화하여 주고받는다. 따라서 자바의 ValueObject 를 주고받을 수 없다. 때문에 밑 사진과 같이 프로토콜 버퍼에서 Message 로 ValueObject 를 생성한다.

```
message Token{
    string tokenValue = 1;
}
message Admin {
    string id = 1;
    string password = 2;
}
message Professor{
    int32 professorId = 1;
    string firstName = 2;
    string lastName = 3;
    string gender = 4;
    string id = 5;
    string password = 6;
}

message Student{
    int32 studentId = 1;
    string firstName = 2;
    string lastName = 3;
    string department = 4;
    int32 maximumCredit = 5;
    double score = 6;
    string gender = 7;
    repeated Course attendedCourse = 8;
    string id = 9;
    string password = 10;
}

message Course{
    int32 courseId = 1;
    string name = 2;
    int32 requirement1 = 3;
    int32 requirement2 = 4;
    int32 credit = 5;
    int32 professorId = 6;
    string professorName = 7;
    int32 numberOfStudent = 8;
    int32 maximumStudent = 9;
}
```

프로토콜 버퍼에서 Token, Admin, Professor, Student, Course Message 를 ValueObject 로 정의한 것이다. 이렇게 하면 gRPC 클라이언트와 서버도 ValueObject 를 주고받을 수 있기 때문에 더 다양한 형태로 데이터를 주고받을 수 있다. 로그인에 성공한 경우에 Message Token 을 ValueObject 로 생성하고 전달한다. 사용자 ID 를 기반으로 토큰을 생성하여 HashMap 에 Token 을 Key, 클라이언트의 ValueObject 를 Value 로 저장한다. 이후 사용자가 토큰과 함께 기능을 요청하면 HashMap 에 존재하는 Token 인지 유효시간이 지나지 않았는지 여부를 따지고 오류가 없다면 요청을 수행한다.

학생 토큰 생성 및 저장((K)토큰: (V)객체)

토큰값 생성 (발행 시간과 유저 Id 를 담는다)

```
private static HashMap<Token, Student> studentAndToken = new HashMap<>();
public Token generateToken(Object object) {
    if(object instanceof Student) {
        Student student = (Student) object;
        String tokenValue = generateTokenValue(student.getStudentId());
        Token token = Token.newBuilder().setTokenValue(tokenValue).build();
        studentAndToken.put(token, student);
        return token;
    }
}

private String generateTokenValue(Integer userId) {
    Date now = new Date();
    Date expiration = new Date(now.getTime() + 3600000);
    headerMap.put("typ", "JWT");
    headerMap.put("alg", "HS256");
    return Jwts.builder().setHeader(headerMap).setIssuer(userId.toString())
        .setIssuedAt(now).setExpiration(expiration)
        .signWith(SECRET_KEY).compact();
}
```

토큰 유효여부 판단 메서드(토큰 발행시간과 발행 여부 판단)

```
public boolean validateToken(Token token, String process) throws TokenException {
    try {
        Date now = new Date();
        Jws<Claims> claims = Jwts.parserBuilder().setSigningKey(SECRET_KEY)
            .build().parseClaimsJws(token.getTokenValue());
        if(process.equals("Student")) {
            if (studentAndToken.containsKey(token) && 3600000 > now.getTime() - claims.getBody().getIssuedAt().getTime()) return true;
            else throw new TokenException("TokenError");
        }
    }
}
```

## 3-2. 학생 Entity CRUD

### 2-1: 학생 Entity Retrieve

학생 Entity Retrieve 요청은 관리자와 교수 클라이언트가 학생 리스트를 출력하기 위하여 전달하는 요청이다. 교수 프로그램은 오직 본인이 개설한 강좌를 신청한 학생만 출력할 수 있고 관리자 프로그램은 학생 검색 조건을 추가하거나 모든 학생을 출력할 수 있다.

#### getStudentList API

```
@Override
public void getStudentList(StudentListRequest request, StreamObserver<StudentListResponse> responseObserver) {
    StudentListResponse response = null;
    try {
        tokenGenerator.validateToken(request.getToken(), request.getProcess());
        List<Student> studentList = studentDao.getStudentList(request.getCondition(), request.getConditionValue());
        response = StudentListResponse.newBuilder().setResult("Success").addAllStudent(studentList).build();
    } catch (TokenException tokenError) {
        response = StudentListResponse.newBuilder().setResult(tokenError.getMessage()).build();
    } catch (NullDataException nullDataError) {
        response = StudentListResponse.newBuilder().setResult(nullDataError.getMessage()).build();
    } catch (ExecuteQueryException sqlError) {
        response = StudentListResponse.newBuilder().setResult("ServerError").build();
    } catch (ServerErrorException serverError) {
        response = StudentListResponse.newBuilder().setResult(serverError.getMessage()).build();
    } finally {
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```

#### StudentDao

```
public List<Student> getStudentList(String condition, String conditionValue) throws NullDataException {
    String studentListQuery = getStudentListQuery(condition, conditionValue);
    ResultSet studentResultSet = super.retrieve(studentListQuery);
    if(studentResultSet == null) throw new NullDataException("NullData");
    else return addStudentInList(studentResultSet);
}
private List<Student> addStudentInList(ResultSet resultSet) throws ExecuteQueryException {
    try {
        List<Student> studentList = new ArrayList<Student>();
        while (true) {
            Student student = Student.newBuilder()
                .setFirstName(resultSet.getString("FIRSTNAME"))
                .setLastName(resultSet.getString("LASTNAME"))
                .setStudentId(resultSet.getInt("STUDENTID"))
                .setDepartment(resultSet.getString("DEPARTMENT"))
                .setMaximumCredit(resultSet.getInt("MAXIMUMCREDIT"))
                .setScore(resultSet.getDouble("SCORE"))
                .setGender(resultSet.getString("GENDER")).build();
            studentList.add(student);
        }
    }
}
```

위 사진은 학생 Entity Retrieve 요청에 응답하는 API 와 StudentDao 이다. 요청에 따라 Database 에 접근하는 쿼리를 만들고 실행하여 결과집합을 가져온다. 결과집합을 학생 ValueObject 로 만들어 리스트에 담고 클라이언트에게 응답하는 코드이다.

관리자 클라이언트는 모든 학생 Entity 를 요청할 수 있으며 성씨, 학과, 학점, 성별로 학생을 구분하고 검색 조건을 추가할 수 있다. 교수 클라이언트는 오직 본인이 개설한 강좌를 신청한 학생 Entity 만 요청할 수 있다. 따라서 서버는 클라이언트가 전달하는 요청과 조건에 따라서 올바른 쿼리를 만들고 Database 에 접근해야 한다.

#### 교수 프로그램의 개설 강좌를 신청한 학생리스트 출력

```
3: 신청학생 정보보기
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 3
모든 강좌입니다.
1번 강좌 --> 강좌명: C++_Programming, 강좌ID: 23456, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: ParkSoyoung, 교수ID: 1914, 최대수강자수: 20, 신청 학생수: 2
확인할 강좌의 번호를 입력해주세요
번호: 1
반상 리스트입니다.
1번 학생 --> 이름: KoKyungmin, 학번: 20070452, 학과: CS, 성별: male, 학점: 3.5, 신청가능학점: 18
2번 학생 --> 이름: KimMinsu, 학번: 20100128, 학과: CS, 성별: male, 학점: 3.0, 신청가능학점: 18
*****MENU*****
```

#### 관리자 프로그램의 학생리스트 출력

```
*****학생검색 조건*****
1: 성씨
2: 학과
3: 학점
4: 성별
5: 전체학생
X: 나가기
입력: 1
검색할 성씨를 영문으로 입력하세요. (Kim, Park ...)
입력: Lee
해당 조건의 학생입니다.
1번 학생 --> 이름: LeeJason, 학번: 20090421, 학과: ME, 성별: male, 학점: 4.0, 신청가능학점: 18
2번 학생 --> 이름: LeeMijung, 학번: 20110298, 학과: ME, 성별: female, 학점: 3.5, 신청가능학점: 18
```

## 2-2: 학생 Entity Create, Update, Delete

관리자 클라이언트는 학생 Entity 를 생성, 수정, 삭제할 수 있다.

### Dao 의 Create, Update, Delete

```
public boolean create(String query) throws NullDataException, ExecuteQueryException {
    try {
        System.out.println(query);
        statement = connection.createStatement();
        int insertValueNumber = statement.executeUpdate(query);
        if(insertValueNumber == 0) throw new NullDataException("NullData");
        else return true;
    } catch (SQLException sqlError) {
        throw new ExecuteQueryException(sqlError.getMessage());
    }
}
```

```
public boolean update(String query) throws ExecuteQueryException, NullDataException {
    try {
        System.out.println(query);
        statement = connection.createStatement();
        int insertValueNumber = statement.executeUpdate(query);
        if(insertValueNumber == 0) throw new NullDataException("NullData");
        else return true;
    } catch (SQLException sqlError) {
        throw new ExecuteQueryException(sqlError.getMessage());
    }
}
```

```
public int delete(String query) throws ExecuteQueryException {
    System.out.println(query);
    try {
        statement = connection.createStatement();
        return statement.executeUpdate(query);
    } catch (SQLException sqlError) {
        throw new ExecuteQueryException(sqlError.getMessage());
    }
}
```

### student(Create, Delete, Update) API

```
@Override
public void createStudent(CreateStudentRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        tokenGenerator.validateToken(request.getToken(), "Admin");
        if(studentDao.createStudent(request.getStudent()) result = "Success";
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullDataException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        if (sqlError.getMessage().contains(STUDENTID_PrimaryKey)) result = "AlreadyExistSameStudentId";
        else result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}

@Override
public void deleteStudent(DeleteStudentRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        tokenGenerator.validateToken(request.getToken(), "Admin");
        if(studentDao.deleteStudent(request.getStudent()) result = "Success";
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullDataException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
}
```

```
@Override
public void updateStudent(UpdateStudentRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        tokenGenerator.validateToken(request.getToken(), "Admin");
        if(studentDao.updateStudent(request.getOriginStudent(), request.getNewStudent()) result = "Success";
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullDataException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
}
```

## 학생 Entity 생성과 조회

```
학생정보를 입력하세요
성씨: 노
이름: 태영
(8자리) 학번: 60211660
검색할 학과를 영문으로 입력하세요.
(CS, EE, ME)
입력: ME
학생의 성별을 입력하세요.
(male, female)
입력: male
학생을 정상적으로 등록했습니다.
```

```
검색할 성씨를 영문으로 입력하세요. (Kim, Park ...)
입력: 노
해당 조건의 학생입니다.
1번 학생 -->> 이름: 노태영, 학번: 60211660, 학과: ME, 성별: male, 학점: 0.0, 신청가능학점: 18
```

## 학생 Entity 수정과 조회(수정할 학생 Entity 선택 후 수정)

```
1번 학생 -->> 이름: 노태영, 학번: 60211660
2번 학생 -->> 이름: LeeJason, 학번: 60211661
3번 학생 -->> 이름: ParkHongsun, 학번: 60211662
4번 학생 -->> 이름: KimYunmi, 학번: 60211663
5번 학생 -->> 이름: KimHokyung, 학번: 60211664
6번 학생 -->> 이름: JungPhilsoo, 학번: 60211665
7번 학생 -->> 이름: AhnJonghyuk, 학번: 60211666
8번 학생 -->> 이름: HwangMyunghan, 학번: 60211667
9번 학생 -->> 이름: LeeMijung, 학번: 60211668
10번 학생 -->> 이름: KimSungsuk, 학번: 60211669
11번 학생 -->> 이름: ParkKitea, 학번: 60211670
12번 학생 -->> 이름: KoKyungmin, 학번: 60211671
13번 학생 -->> 이름: KimChulmin, 학번: 60211672
14번 학생 -->> 이름: ParkKiyong, 학번: 60211673
15번 학생 -->> 이름: KimMinsu, 학번: 60211674
16번 학생 -->> 이름: KimJungMi, 학번: 60211675
17번 학생 -->> 이름: JangGoyoung, 학번: 60211676
18번 학생 -->> 이름: KimSoyoung, 학번: 60211677

수정할 학생의 번호를 입력해주세요
입력: 1
새로운 학생정보를 입력하세요
성씨: 홍
이름: 길동
검색할 학과를 영문으로 입력하세요.
(CS, EE, ME)
입력: CS
학생의 성별을 입력하세요.
(male, female)
입력: female
선택한 학생을 수정했습니다.
```

```
검색할 성씨를 입력하세요. (Kim, Park ...)
입력: 홍
해당 조건의 학생입니다.
1번 학생 -->> 이름: 홍길동, 학번: 60211660, 학과: CS, 성별: female, 학점: 0.0, 신청가능학점: 18
```

## 학생 Entity 삭제와 조회

```
1번 학생 -->> 이름: 홍길동, 학번: 60211660
2번 학생 -->> 이름: LeeJason, 학번: 60211661
3번 학생 -->> 이름: ParkHongsun, 학번: 60211662
4번 학생 -->> 이름: KimYunmi, 학번: 60211663
5번 학생 -->> 이름: KimHokyung, 학번: 60211664
6번 학생 -->> 이름: JungPhilsoo, 학번: 60211665
7번 학생 -->> 이름: AhnJonghyuk, 학번: 60211666
8번 학생 -->> 이름: HwangMyunghan, 학번: 60211667
9번 학생 -->> 이름: LeeMijung, 학번: 60211668
10번 학생 -->> 이름: KimSungsuk, 학번: 60211669
11번 학생 -->> 이름: ParkKitea, 학번: 60211670
12번 학생 -->> 이름: KoKyungmin, 학번: 60211671
13번 학생 -->> 이름: KimChulmin, 학번: 60211672
14번 학생 -->> 이름: ParkKiyong, 학번: 60211673
15번 학생 -->> 이름: KimMinsu, 학번: 60211674
16번 학생 -->> 이름: KimJungMi, 학번: 60211675
17번 학생 -->> 이름: JangGoyoung, 학번: 60211676
18번 학생 -->> 이름: KimSoyoung, 학번: 60211677

삭제할 학생의 번호를 입력해주세요
입력: 1
선택한 학생을 삭제했습니다.
```

```
입력: 1
검색할 성씨를 입력하세요. (Kim, Park ...)
입력: 홍
해당 조건의 학생이 없습니다
```

관리자 클라이언트가 학생 Entity 를 생성할 때는 학생에 대한 정보를 입력하고 입력한 정보를 학생 ValueObject 로 만들어 서버에게 요청을 보낸다. StudentValue Object 를 전달받은 서버는 학생 객체 내부의 필드를 이용하여 쿼리를 만들고 Database 에 접근하여 학생 Entity 를 생성한다.

학생 Entity 를 삭제할 때는 모든 학생리스트를 출력하고 클라이언트가 선택한 학생을 서버에게 전달하여 삭제한다. 마찬가지로 학생 Entity 를 수정할 때도 모든 학생리스트를 출력하여 수정할 학생을 선택하고 새로운 데이터를 입력하여 기존 Database 에 저장된 Entity 를 수정한다.

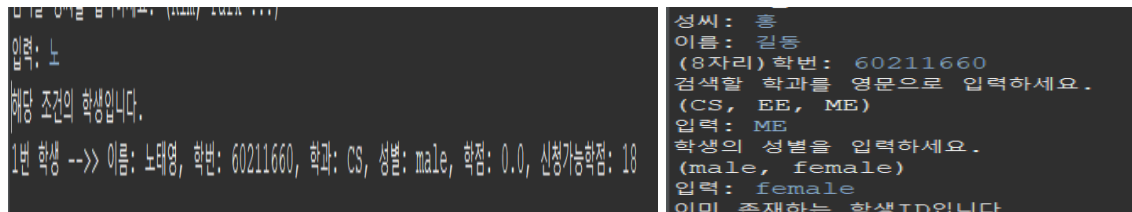
관리자가 학생 Entity 를 CRUD 할 때는 Entity-Integrity 와 Referential-Integrity 를 지키며 데이터를 수정해야 하며 무결성을 깨뜨리는 쿼리에 대해서는 예외처리 로직을 다음과 같이 구현한다.

Student 테이블에서 PrimaryKey 역할을 하는 StudentId 의 Entity-Integrity 를 지키기 위해 동일한 studentId 로 학생을 생성하는 것에 대하여 예외처리를 추가한다.

```
} catch (ExecuteQueryException sqlError) {  
    if (sqlError.getMessage().contains(STUDENTID_PrimaryKey)) result = "AlreadyExistSameStudentId";
```

이미 존재하는 studentId(60211660)

동일한 studentId(60211660)로 학생 Entity 생성



입력: 60211660  
이미 존재하는 학생입니다.  
1번 학생 -->> 이름: 노태영, 학번: 60211660, 학과: CS, 성별: male, 학점: 0.0, 신청가능학점: 18

성씨: 홍  
이름: 길동  
(8자리) 학번: 60211660  
검색할 학과를 영문으로 입력하세요.  
(CS, EE, ME)  
입력: ME  
학생의 성별을 입력하세요.  
(male, female)  
입력: female  
이미 존재하는 학생 ID입니다.

위 사진은 관리자 클라이언트가 동일한 studentId 를 사용하여 학생 Entity 를 생성하려할 때 이를 예외처리를 통해서 막는 결과이다.

학생 Entity 가 삭제됨에 따라 학생 Entity 의 PrimaryKey 를 기반으로 관계를 가지는 모든 행을 삭제해야 한다. 그렇지 않는다면 예외발생을 통해 Entity 삭제를 막아 오류가 발생한다. 혹은 Database 에 따라서 Referential-Integrity 가 깨져 데이터의 완전성과 정확성을 잃을 수 있기 때문에 먼저 관계를 가지는 모든 행을 삭제하는 작업이 선 수행되어야 한다.

StudentDao 에서 삭제할 StudentEntity 와 관계를 맺는 모든 테이블의 데이터를 Delete 한 후 StudentEntity 를 최종적으로 삭제한다. studentId 를 ForeignKey 로 하는 StudentAccount, ApplicationList, AttendedCourseList 테이블에 학생 Entity 와 관계를 맺는 모든 레코드를 삭제하는 코드이다.

```
public boolean deleteStudent(Student student) throws NullDataException, ExecuteQueryException {  
    String deleteStudentAccount = constant.getDeleteAccountQueryForIntegrity(student);  
    String deleteApplicationList = constant.getDeleteApplicationListQueryForIntegrity(student);  
    String deleteAttendedList = constant.getDeleteAttendedListQueryForIntegrity(student);  
    String deleteStudent = constant.getDeleteStudentQuery(student);  
    super.delete(deleteStudentAccount);  
    super.delete(deleteApplicationList);  
    super.delete(deleteAttendedList);  
    if(super.delete(deleteStudent) == 0) throw new NullDataException("NullData");  
    else return true;
```

학생 Entity 를 수정하는 경우에는 StudentId 수정을 막아놔기 때문에 발생하는 데이터 무결성 오류가 없으며 일반적으로 sql 이 동작하지 않거나, 서버에 오류가 발생한 상황에 대해서만 예외처리를 해두었다.



### 3-3. 강좌 EntityCrud

#### 3-1: 강좌 Entity Retrieve

강좌 리스트 출력은 모든 클라이언트에서 수행되는 작업이다. 학생 프로그램에선 학생 본인이 이전에 수강했던 강좌 리스트, 현재 신청한 수강신청 리스트, 모든 강좌 리스트를 요청한다. 교수 클라이언트에서는 본인이 개설한 강좌와 타 교수가 개설한 강좌를 구분하여 요청한다. 관리자 클라이언트는 검색 조건을 추가한 강좌 리스트나 전 강좌 리스트를 요청한다. 뿐만 아니라 Course 테이블과 ApplicationList 테이블을 Join 하여 해당 강좌를 신청한 학생 인원 수까지 출력한다.

getCoursList API

```
@Override
public void getCourseList(CourseListRequest request, StreamObserver<CourseListResponse> responseObserver) {
    CourseListResponse response = null;
    try {
        Token token = request.getToken();
        List<Course> courseList = new ArrayList<Course> ();
        tokenGenerator.validateToken(token, request.getProcess());

        if(request.getIsAttendedList()) {
            courseList = tokenGenerator.getStudent(token).getAttendedCourseList();
            if(courseList.size() == 0) throw new NullPointerException("NullData");
        } else if(request.getIsApplicationList()) {
            Student student = tokenGenerator.getStudent(token);
            courseList = courseDao.getApplicationList(student.getStudentId());
        } else courseList = courseDao.getCourseList(request.getCondition(), request.getConditionValue());

        response = CourseListResponse.newBuilder().setResult("Success").addAllCourse(courseList).build();
    } catch (TokenException tokenError) {
        response = CourseListResponse.newBuilder().setResult(tokenError.getMessage()).build();
    } catch (NullPointerException nullDataError) {
        response = CourseListResponse.newBuilder().setResult(nullDataError.getMessage()).build();
    } catch (ExecuteQueryException sqlError) {
        response = CourseListResponse.newBuilder().setResult("ServerError").build();
    } catch (ServerErrorException serverError) {
        response = CourseListResponse.newBuilder().setResult(serverError.getMessage()).build();
    } finally {
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```

위 사진은 강좌 Entity Retrieve 요청에 응답하는 API 이다. 이 API 또한 로그인처럼 API 내부에서 경우를 나누고 수강신청 내역 요청, 이전 수강내역 요청, 강좌리스트 요청을 하나의 API 로 해결한다. 요청에 따라 Database 에 접근하는 쿼리를 만들고 실행하여 결과집합을 가져온다. 결과집합을 강좌 ValueObject 로 만들어 리스트에 담고 클라이언트에게 응답한다.

먼저 학생 클라이언트가 요청하는 이전 수강내역은 학생 객체가 필드로 가지는 정적인 값이기 때문에 요청을 보낸 학생 ValueObject 내부에 존재하는 이전 수강내역을 클라이언트에게 전달한다. 이전 수강내역은 로그인 과정에서 학생 객체를 생성할 때 초기화한 정적인 필드값이다. 때문에 Dao 를 통해 Database 에서 데이터를 가져올 필요가 없다.

수강신청 내역 요청은 먼저 클라이언트가 전달한 토큰을 통해 HashMap 에 저장된 학생객체의 StudentId 를 가져온다. StudentId 를 기반으로 ApplicationList(수강신청 테이블)에 접근하는 쿼리를 만들어 결과 집합을 가져온 후 List 를 만들고 클라이언트에게 리턴한다.

강좌 리스트 요청은 전 강좌 또는 추가적인 조건의 일부 강좌를 요청하는 두 가지 경우로 나뉜다. 따라서 클라이언트의 요청에 맞는 쿼리를 만들고 결과집합을 가져와 리스트를 리턴해야 한다.



## 일반 강좌와 수강신청 강좌를 구분한 CourseDao 결과집합을 리스트에 담아 리턴하는 메서드

```
public List<Course> getApplicationList(int studentId) throws ServerException, ExecuteQueryException {
    String query = constant.getApplicationListQuery(studentId);
    ResultSet courseResultSet = super.retrieve(query);
    return addCourseInList(courseResultSet);
}

public List<Course> getCourseList(String condition, String conditionValue) throws ExecuteQueryException, ServerException {
    String query;
    if(condition.equals("CREDIT")) query = constant.getCourseListQuery("WHERE CREDIT = ", conditionValue, "");
    else if(condition.equals("REQUIREMENT1") && conditionValue.equals("NotNull")) query = constant.getCourseListQuery("WHERE REQUIREMENT1 = ", conditionValue, "");
    else if(condition.equals("REQUIREMENT1") && conditionValue.equals("Null")) query = constant.getCourseListQuery("WHERE REQUIREMENT1 = ", conditionValue, "");
    else if(condition.equals("MYCOURSE")) query = constant.getCourseListQuery("WHERE PROFESSOR.PROFESSORID = ", conditionValue, "");
    else if(condition.equals("MYCOURSE")) query = constant.getCourseListQuery("WHERE PROFESSOR.PROFESSORID = ", conditionValue, "");
    else if(condition.equals("MYCOURSE")) query = constant.getCourseListQuery("WHERE PROFESSOR.PROFESSORID = ", conditionValue, "");
    else if(condition.equals("NUMBEROFSTUDENT")) query = constant.getCourseListQuery("WHERE MAXIMUMSTUDENT = ", conditionValue, "");
    else query = constant.getCourseListQuery("", "", "");
    ResultSet courseResultSet = super.retrieve(query);
    return addCourseInList(courseResultSet);
}

private List<Course> addCourseInList(ResultSet courseResultSet) throws ExecuteQueryException {
    List<Course> courseList = new ArrayList<Course>();
    if(courseResultSet == null) return courseList;
    try {
        while (true) {
            Course course = Course.newBuilder()
                .setCourseId(courseResultSet.getInt("COURSEID"))
                .setName(courseResultSet.getString("NAME"))
                .setRequirement1(courseResultSet.getInt("REQUIREMENT1"))
                .setRequirement2(courseResultSet.getInt("REQUIREMENT2"))
                .setCredit(courseResultSet.getInt("CREDIT"))
                .setProfessorId(courseResultSet.getInt("PROFESSORID"))
                .setProfessorName(courseResultSet.getString("FIRSTNAME") + courseResultSet.getString("LASTNAME"))
                .setNumberOfStudent(courseResultSet.getInt("NUMBEROFSTUDENT"))
                .setMaximumStudent(courseResultSet.getInt("MAXIMUMSTUDENT"))
                .build();
            courseList.add(course);
            if(!courseResultSet.next()) return courseList;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return courseList;
}
```

위 코드는 CourseDao 에서 사용자의 요청에 맞는 쿼리를 만든 후 결과 집합을 가져와 Course 객체를 만들어 List 에 담고 리턴하는 코드이다. 사용자의 요청이 수강신청 내역인지 아닌지 여부를 판단하고 맞다면 studentId 를 통해 ApplicationList(수강신청 테이블)에서 결과 집합을 가져와 신청한 강좌를 객체로 만들고 List 에 담는다. 수강신청 내역을 요청하는 것이 아니라면 클라이언트의 조건을 판단하고 조건에 맞는 쿼리를 만들고 쿼리를 실행하여 결과를 리턴한다.

## 학생 클라이언트의 전 강좌, 이전 수강내역, 수강신청 내역 요청과 응답 결과

```
1번 강좌 --> 강좌명: C++ Programming, 강좌ID: 23456, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: ParkSoyoung, 교수ID: 1914, 최다수강자수: 20, 신청 학생수: 2
2번 강좌 --> 강좌명: Managing Software Development, 강좌ID: 17653, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: KimKitea, 교수ID: 2226, 최다수강자수: 20, 신청 학생수: 0
3번 강좌 --> 강좌명: Methods Of Software Development, 강좌ID: 17652, REQUIREMENT1: 23456, REQUIREMENT2: 0, 학점: 3, 교수이름: KoMinsu, 교수ID: 1554, 최다수강자수: 20, 신청 학생수: 0
4번 강좌 --> 강좌명: Analysis of Software Artifacts, 강좌ID: 17654, REQUIREMENT1: 17651, REQUIREMENT2: 0, 학점: 3, 교수이름: AhnJiyoung, 교수ID: 9872, 최다수강자수: 20, 신청 학생수: 0
5번 강좌 --> 강좌명: Architectures of Software Systems, 강좌ID: 17655, REQUIREMENT1: 12345, REQUIREMENT2: 17651, 학점: 3, 교수이름: LeeHyunsu, 교수ID: 4561, 최다수강자수: 20, 신청 학생수: 0

*****MENU*****
1: 모든강좌 출력
2: 이전 수강내역 출력
3: 수강신청 내역 출력
4: 수강신청 변경
5: ID/비밀번호 변경
6: 계정삭제
X: 종료
입력: 2
1번 강좌 --> 강좌명: Java Programming, 강좌ID: 12345, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: ParkSoyoung, 교수ID: 1914, 최다수강자수: 0, 신청 학생수: 0
2번 강좌 --> 강좌명: Managing Software Development, 강좌ID: 17653, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: KimKitea, 교수ID: 2226, 최다수강자수: 0, 신청 학생수: 0

*****MENU*****
1: 모든강좌 출력
2: 이전 수강내역 출력
3: 수강신청 내역 출력
4: 수강신청 변경
5: ID/비밀번호 변경
6: 계정삭제
X: 종료
입력: 3
1번 강좌 --> 강좌명: C++ Programming, 강좌ID: 23456, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: ParkSoyoung, 교수ID: 1914, 최다수강자수: 20, 신청 학생수: 2
```

## 교수 클라이언트의 본인 개설강좌와 타 교수 개설강좌

```
1: 본인 개설강좌 출력
2: 타강좌 출력
3: 신청학생 정보보기
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 1
모든 강좌입니다.
1번 강좌 --> 강좌명: C++ Programming, 강좌ID: 23456, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: ParkSoyoung, 교수ID: 1914, 최다수강자수: 20, 신청 학생수: 2

*****MENU*****
1: 본인 개설강좌 출력
2: 타강좌 출력
3: 신청학생 정보보기
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 2
모든 강좌입니다.
1번 강좌 --> 강좌명: Managing Software Development, 강좌ID: 17653, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: KimKitea, 교수ID: 2226, 최다수강자수: 20, 신청 학생수: 0
2번 강좌 --> 강좌명: Methods Of Software Development, 강좌ID: 17652, REQUIREMENT1: 23456, REQUIREMENT2: 0, 학점: 3, 교수이름: KoMinsu, 교수ID: 1554, 최다수강자수: 20, 신청 학생수: 0
3번 강좌 --> 강좌명: Analysis of Software Artifacts, 강좌ID: 17654, REQUIREMENT1: 17651, REQUIREMENT2: 0, 학점: 3, 교수이름: AhnJiyoung, 교수ID: 9872, 최다수강자수: 20, 신청 학생수: 0
4번 강좌 --> 강좌명: Architectures of Software Systems, 강좌ID: 17655, REQUIREMENT1: 12345, REQUIREMENT2: 17651, 학점: 3, 교수이름: LeeHyunsu, 교수ID: 4561, 최다수강자수: 20, 신청 학생수: 0
```

관리자 프로세스의 선수과목 유무 조건을 추가한 강좌내역 출력 (선수과목 조건)

```
1: 선수과목 O
2: 선수과목 X
X: 나가기
입력: 1
해당 조건의 강좌입니다.
1번 강좌 --> 강좌명: Methods_of_Software_Development, 강좌ID: 17652, REQUIREMENT1: 23456, REQUIREMENT2: 0
2번 강좌 --> 강좌명: Analysis_of_Software_Artifacts, 강좌ID: 17654, REQUIREMENT1: 17651, REQUIREMENT2: 0
3번 강좌 --> 강좌명: Architectures_of_Software_Systems, 강좌ID: 17655, REQUIREMENT1: 12345, REQUIREMENT2: 0
*****강좌검색 조건*****
1: 전공/교양
2: 학점
3: 선수과목 유무
4: 신청학생 수
5: 모든 강좌
X: 나가기
입력: 3
1: 선수과목 O
2: 선수과목 X
X: 나가기
입력: 2
해당 조건의 강좌입니다.
1번 강좌 --> 강좌명: C++ Programming, 강좌ID: 23456, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: 김민준
2번 강좌 --> 강좌명: Managing_Software_Development, 강좌ID: 17653, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: 이영희
```

### 3-2: 강좌 Entity Create, Update, Delete

교수 클라이언트는 강좌 Entity 를 생성, 수정, 삭제할 수 있다.

Course(Create, Delete, Update) API

```
@Override
public void creatCourse(CreateCourseRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        tokenGenerator.getProfessor(request.getToken());
        if(courseDao.createNewCourse(request.getCourse())) result = "Success";
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullPointerException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        if (sqlError.getMessage().contains(COURSEID_PrimaryKey)) result = "AlreadyExistSameCourseId";
        else if (sqlError.getMessage().contains(COURSENAME_Unique)) result = "AlreadyExistSameCourseName";
        else result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}

@Override
public void deleteCourse(DeleteCourseRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        tokenGenerator.getProfessor(request.getToken());
        if(courseDao.deleteCourse(request.getCourse().getCourseId())) result = "Success";
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullPointerException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```

```
@Override
public void updateCourse(UpdateCourseRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        tokenGenerator.getProfessor(request.getToken());
        if(courseDao.updateCourse(request.getOriginCourse(), request.getNewCourse())) result = "Success";
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullPointerException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        if (sqlError.getMessage().contains(COURSENAME_Unique)) result = "AlreadyExistSameCourseName";
        else result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```

## CourseDao

```
public boolean createNewCourse(Course course) throws NullDataException, ExecuteQueryException {
    String createCourseQuery = constant.getCreateCourseQuery(course);
    return super.create(createCourseQuery);
}

public boolean deleteCourse(int courseId) throws ExecuteQueryException, NullDataException {
    String deleteCourseInApplicaition = constant.getDeleteCourseInApplicationQuery(courseId);
    String deleteCourse = constant.getDeleteCourse(courseId);
    super.delete(deleteCourseInApplicaition);
    if(super.delete(deleteCourse) == 0) throw new NullDataException("NullData");
    else return true;
}

public boolean updateCourse(Course originCourse, Course newCourse) throws ExecuteQueryException, NullDataException {
    String updateCourseQuery = constant.getUpdateCourseQuery(originCourse, newCourse);
    return super.update(updateCourseQuery);
}
```

## 교수 클라이언트의 강좌 Entity 생성과 조회

```
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 4
강좌정보를 입력하세요
강좌명: 클라이언트_서버_프로그래밍
(5자리) 강좌번호: 98765
선수과목 유무
1: 선수과목 O
2: 선수과목 X
입력: 1

모든 강좌입니다.
1번 강좌 -->> 강좌명: Java_Programming,
2번 강좌 -->> 강좌명: C++_Programming,
3번 강좌 -->> 강좌명: Models_of_Software,
4번 강좌 -->> 강좌명: Managing_Software,
5번 강좌 -->> 강좌명: Methods_Of_Software,
6번 강좌 -->> 강좌명: Analysis_of_Software,
7번 강좌 -->> 강좌명: Architectures_of_Software,

1번 선수과목 입력: 숫자를 입력하세요.
1번 선수과목 입력: 1번 선수과목 입력: 1
2번 선수과목이 없다면 0을 입력하십시오.
2번 선수과목 입력: 2
학점: 3
최대학생수: 30
강좌가 정상적으로 생성됐습니다.

1: 본인 개설강좌 출력
2: 타강좌 출력
3: 신청학생 정보보기
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 1
모든 강좌입니다.
1번 강좌 -->> 강좌명: 클라이언트_서버_프로그래밍, 강좌ID: 98765, REQUIREMENT1: 12345, REQUIREMENT2: 23456, 학점: 3, 교수이름: ChoiSungwoon, 교수ID: 1354, 최대수강자수: 30, 수강자수: 0
```

## 강좌 Entity 수정과 조회

```
6: 개설강좌 수정하기
입력: 6
모든 강좌입니다.
1번 강좌 -->> 강좌명: 클라이언트_서버_프로그래밍, 강좌ID: 98765, REQUIREMENT1: 12345, REQUIREMENT2: 23456, 학점: 3, 교수이름: ChoiSungwoon, 교수ID: 1354, 최대수강자수: 30, 수강자수: 0

수정할 강좌의 번호를 입력해주세요.
번호: 1
수정할 정보를 입력하세요
강좌명: 분산_프로그래밍
선수과목 유무
1: 선수과목 O
2: 선수과목 X
입력: 2
학점: 1
최대학생수: 10
선택한 강좌를 수정했습니다.
*****MENU*****
1: 본인 개설강좌 출력
2: 타강좌 출력
3: 신청학생 정보보기
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 1
모든 강좌입니다.
1번 강좌 -->> 강좌명: 분산_프로그래밍, 강좌ID: 98765, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 1, 교수이름: ChoiSungwoon, 교수ID: 1354, 최대수강자수: 10, 수강자수: 0
```

## 강좌 Entity 삭제와 조회

```
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 5
모든 강좌입니다.
1번 강좌 -->> 강좌명: 분산_프로그래밍, 강좌ID: 98765, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 1, 교수이름: ChoiSungwoon, 교수ID: 1354, 최대수강자수: 10, 수강자수: 0

삭제할 강좌의 번호를 입력해주세요.
번호: 1
선택한 강좌를 삭제했습니다.
*****MENU*****
1: 본인 개설강좌 출력
2: 타강좌 출력
3: 신청학생 정보보기
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 1
모든 강좌입니다.
해당 조건의 강좌가 없습니다
```

교수 클라이언트가 강좌 Entity 를 생성할 때는 강좌에 대한 정보를 입력받고 입력받은 정보를 강좌 ValueObject 로 만들어 서버에게 요청을 보낸다. 강좌 ValueObject 를 전달받은 서버는 강좌 객체 내부의 필드를 이용하여 쿼리를 만들고 Database 에 접근하여 강좌 Entity 를 생성한다.

강좌 Entity 를 삭제할 때는 교수 본인이 개설한 강좌리스트를 출력하고 클라이언트가 선택한 강좌를 서버에게 전달하여 삭제한다. 마찬가지로 강좌를 수정할 때도 모든 강좌리스트를 출력하여 수정할 강좌를 선택하고 새로운 데이터를 입력하여 기존 Database 에 저장된 Entity 를 수정한다.

교수 클라이언트 강좌 Entity 를 CRUD 하는 것도 학생 Entity 와 마찬가지로 Entity-Integrity 와 Referential-Integrity 를 지키며 데이터를 수정해야 한다. 따라서 강좌 테이블에서 PrimaryKey 역할을 하는 CourseId 의 Entity-Integrity 를 지키기 위해 동일한 CourseId 로 강좌를 생성하는 것에 대하여 예외처리를 추가한다. 또한 Database 를 생성하며 강좌을 이름에도 Unique 조건을 추가하여 동일한 이름의 강좌가 생성되는 것을 막아놨기 때문에 이 둘을 예외처리 해야한다.

```
private final String COURSEID_PrimaryKey = "SYS_C008457";
private final String COURSENAME_Unique = "SYS_C008458";
```

```
catch (ExecuteQueryException sqlError) {
    if (sqlError.getMessage().contains(COURSEID_PrimaryKey)) result = "AlreadyExistSameCourseId";
    else if (sqlError.getMessage().contains(COURSENAME_Unique)) result = "AlreadyExistSameCourseName";
}
```

위 사진은 createCourse API 에서 동일한 courseId, courseName 으로 강좌 생성이 요청됐을 때 해당 예외를 처리하는 코드이다.

```
if (sqlError.getMessage().contains(COURSENAME_Unique)) result = "AlreadyExistSameCourseName";
```

위 사진은 updateCourse API 의 예외이며 강좌를 수정할 경우도 마찬가지로 변경하려는 강좌의 이름이 동일한 경우를 예외처리한다.

```
public boolean deleteCourse(int courseId) throws ExecuteQueryException, NullDataException {
    String deleteCourseInApplicaition = constant.getDeleteCourseInApplicationQuery(courseId);
    String deleteCourse = constant.getDeleteCourse(courseId);
    super.delete(deleteCourseInApplicaition);
    if(super.delete(deleteCourse) == 0) throw new NullDataException("NullData");
    else return true;
}
```

강좌를 삭제할 때 Referential-Integrity 를 지키기 위해 강좌의 courseId 를 ForeignKey 로 하는 Application\_List 에 존재하는 Entity 를 먼저 삭제한 이후 Course 테이블의 Entity 를 삭제한다.

클라이언트의 동일한 이름 강좌생성 요청과 동일한 courseId(17655)의 강좌생성 요청 실행결과

```
3번 강좌 -->> 강좌명: Methods_of_Software_Development, 강좌ID: 17652,
4번 강좌 -->> 강좌명: Analysis_of_Software_Artifacts, 강좌ID: 17654,
5번 강좌 -->> 강좌명: Architectures_of_Software_Systems, 강좌ID: 17655

*****MENU*****
1: 본인 개설강좌 출력
2: 타강좌 출력
3: 신청학생 정보보기
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 4
강좌정보를 입력하세요
강좌명: Architectures_of_Software_Systems
(5자리) 강좌번호: 45897
선수과목 유무
1: 선수과목 O
2: 선수과목 X
입력: 2
학점: 3
최대학생수: 10
같은 이름의 강좌가 이미 존재합니다.
*****MENU*****
1: 본인 개설강좌 출력
2: 타강좌 출력
3: 신청학생 정보보기
4: 새로운 강좌 등록하기
5: 개설강좌 삭제하기
6: 개설강좌 수정하기
입력: 4
강좌정보를 입력하세요
강좌명: 소프트웨어_아키텍처
(5자리) 강좌번호: 17655
선수과목 유무
1: 선수과목 O
2: 선수과목 X
입력: 2
학점: 1
최대학생수: 10
같은 강좌ID가 이미 존재합니다.
```

클라이언트의 동일한 이름으로 강좌 Entity 수정요청 실행결과

```
1번 강좌 -->> 강좌명: 소프트웨어_아키텍처, 강좌ID:

수정할 강좌의 번호를 입력해주세요
번호: 1
수정할 정보를 입력하세요
강좌명: Architectures_of_Software_Systems
선수과목 유무
1: 선수과목 O
2: 선수과목 X
입력: 2
학점: 3
최대학생수: 10
동일한 이름의 강좌가 이미 존재합니다.
```

## 3-4. 학생계정 EntityCrud

### 4-1: 학생계정 Entity Retrieve(ID 찾기, Password 찾기)

학생계정 Entity Retrieve 요청은 학생 클라이언트가 계정의 ID와 비밀번호를 Retrieve 하기 위하여 전달하는 요청이다. ID를 요청하기 위해서 이름, 학번을 입력하여 서버에게 전달한다. 서버는 해당 입력의 여부를 판단하고 Student 테이블과 StudentAccount 테이블을 조인하여 입력한 학생 Entity가 관계를 맺는 학생계정 Entity를 읽어와 ID를 리턴한다. Password 요청도 마찬가지로 Student 테이블과 StudentAccount 테이블을 조인하여 Password를 찾고 리턴한다.

findAccount, findPassword API

```
public void findAccount(FindAccountRequest request, StreamObserver<FindAccountResponse> responseObserver) {
    FindAccountResponse response = null;
    try {
        String id = studentAccountDao.findStudentAccount(request.getFirstName(), request.getLastName(), request.getStudentId());
        response = FindAccountResponse.newBuilder().setResult("Success").setId(id).build();
    } catch (NullPointerException nullDataError) {
        response = FindAccountResponse.newBuilder().setResult(nullDataError.getMessage()).build();
    } catch (ExecuteQueryException sqlError) {
        response = FindAccountResponse.newBuilder().setResult("ServerError").build();
    } catch (ServerException serverError) {
        response = FindAccountResponse.newBuilder().setResult(serverError.getMessage()).build();
    } finally {
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}

@Override
public void findPassword(FindPasswordRequest request, StreamObserver<FindPasswordResponse> responseObserver) {
    FindPasswordResponse response = null;
    try {
        String password = studentAccountDao.findPassword(request.getFirstName(), request.getLastName(),
            request.getStudentId(), request.getId());
        response = FindPasswordResponse.newBuilder().setResult("Success").setPassword(password).build();
    } catch (NullPointerException nullDataError) {
        response = FindPasswordResponse.newBuilder().setResult(nullDataError.getMessage()).build();
    } catch (ExecuteQueryException sqlError) {
        response = FindPasswordResponse.newBuilder().setResult("ServerError").build();
    } catch (ServerException serverError) {
        response = FindPasswordResponse.newBuilder().setResult(serverError.getMessage()).build();
    }
    responseObserver.onNext(response);
    responseObserver.onCompleted();
}
```

StudentAccountDao

```
public String findStudentAccount(String firstName, String lastName, int studentId) throws NullPointerException {
    String studentAccountQuery = constant.getStudentAccountQuery(firstName, lastName, studentId);
    ResultSet resultSet = super.retrieve(studentAccountQuery);
    return getResult(resultSet, "ID");
}

public String findPassword(String firstName, String lastName, int studentId, String id) throws NullPointerException {
    String studentPasswordQuery = constant.getStudentAccountQuery(firstName, lastName, studentId)
        + constant.getStudentPasswordQuery(id);
    ResultSet resultSet = super.retrieve(studentPasswordQuery);
    return getResult(resultSet, "PASSWORD");
}

public String getResult(ResultSet resultSet, String result) throws NullPointerException {
    try {
        return resultSet.getString(result);
    } catch (SQLException sqlError) {
        throw new NullPointerException("NullData");
    }
}
```

클라이언트의 아아디, 비밀번호 찾기 실행결과

```
*****계정찾기*****
영문 이름 입력
성: 노
이름: 태영
학번: 60211660
노태영님의 ID: TestId
*****MENU*****
1: 로그인
2: 회원가입
3: 계정찾기
4: 비밀번호 찾기
X: 종료
입력: 4
*****비밀번호 찾기*****
영문 이름 입력
성: 노
이름: 태영
학번: 60211660
ID: TestId
노태영님의 PASSWORD: TestPw
*****MENU*****
1: 로그인
2: 회원가입
3: 계정찾기
4: 비밀번호 찾기
X: 종료
입력: 1
*****LOGIN*****
학생계정 ID: TestId
학생계정 Password: TestPw
노태영님 환영합니다.
*****MENU*****
```

4-2: 학생계정 Entity Create(Register), Update, Delete

account(Register, Update, Delete) API

```
@Override
public void register(RegisterRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        if(studentAccountDao.register(request.getFirstName(), request.getLastName(), request.getStudentId(), request.getId(), request.getPassword()))
            result = "Success"; //DONE!!!!!!
    } catch (NullPointerException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        if(sqlError.getMessage().contains(STUDENTID_Unique)) result = "AlreadyExistAccount"; //DONE!!!!!!
        else if (sqlError.getMessage().contains(ACCOUNTID_PrimaryKey)) result = "AlreadyExistId"; //DONE!!!!!!
        else result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}

@Override
public void updateAccount(UpdateAccountRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        Student student = tokenGenerator.getStudent(request.getToken());
        checkIdPassword(request.getId(), request.getPassword(), student.getId(), student.getPassword()); //DONE!!!!!!
        if(studentAccountDao.updateAccount(request.getId(), request.getPassword(), request.getNewId(), request.getNewPassword())) {
            result = "Success";
        }
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullPointerException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        if(sqlError.getMessage().contains(ACCOUNTID_PrimaryKey)) result = "AlreadyExistId"; //DONE!!!!!!
        else result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```



```

@Override
public void deleteAccount(DeleteAccountRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        Student student = tokenGenerator.getStudent(request.getToken()); //DONE!!!!!!
        checkStudentInfo(student, request.getFirstName(), request.getLastName(), request.getStudentId(), request.getId(), request.getPassword());
        if(studentAccountDao.deleteAccount(request.getFirstName(), request.getLastName(), request.getStudentId(), request.getId(), request.getPassword()))
            result = "Success";
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullDataException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}

```

## AccountDao

```

public boolean register(String firstName, String lastName, int studentId, String id, String password) throws NullDataException {
    String registerQuery = constant.getRegisterQuery(firstName, lastName, studentId, id, password);
    return super.create(registerQuery);
}

public boolean updateAccount(String usedId, String usedPassword, String newId, String newPassword) throws NullDataException {
    String updateStudentAccountQuery = constant.getUpdateStudentAccountQuery(usedId, usedPassword, newId, newPassword);
    return super.update(updateStudentAccountQuery);
}

public boolean deleteAccount(String firstName, String lastName, int studentId, String id, String password) throws NullDataException {
    String deleteStudentAccount = constant.getDeleteAccountQuery(firstName, lastName, studentId, id, password);
    if(super.delete(deleteStudentAccount) == 0) throw new NullDataException("NullData");
    else return true;
}

```

학생계정은 학생 Entity 에 종속적인 Total Participation 관계이다. 따라서 학생 Entity 가 존재하는 사용자만 학생계정을 생성할 수 있다. 학생계정을 생성하기 위해선 이름, 학번, ID, Password 를 입력하고 서버에게 전달한다. 서버는 해당 입력에 맞게 쿼리를 만들고 학생계정 Entity 를 생성한다. 학생계정을 수정, 삭제하기 위해선 먼저 해당 계정으로 로그인 한 뒤 현재 ID,Password, 새로운 ID,Password 를 입력하여 계정을 수정할 수 있으며 이름, 학번 계정을 입력하여 계정을 삭제할 수 있다. 그러나 서버에서 바로 입력값을 쿼리로 만들고 실행시킨다면 A 사용자가 B 사용자의 계정을 삭제하거나 수정할 수 있기 때문에 토큰을 통해 학생 ValueObject 를 가져오고 본인의 계정이 맞는지를 먼저 판단하고 수정과 삭제를 수행한다.

## 서버의 판단 코드

```

private void checkIdPassword(String inputId, String inputPassword, String originId, String originPassword) throws NullDataException {
    if (originId.equals(inputId) && originPassword.equals(inputPassword)) {}
    else throw new NullDataException("NoneMatchInputIdPassword"); //DONE!!!!!!
}

private void checkStudentInfo(Student student, String firstName, String lastName, int studentId, String id, String password) {
    if(student.getFirstName().equals(firstName) && student.getLastName().equals(lastName)
        && student.getStudentId()==studentId && student.getId().equals(id) && student.getPassword().equals(password) ) {}
    else throw new NullDataException("NoneMatchStudentInfo");
}

```



## 계정 생성과 로그인

```
*****
영문 이름 입력
성: 노
이름: 태영
학번: 60211660
ID: TestId
Password: TestPw
회원가입 되었습니다.

학생계정 ID: TestId
학생계정 Password: TestPw
노태영님 환영합니다.
```

## 계정 수정과 로그인

```
현재 ID: TestId
현재 PASSWORD: TestPw
새로운 ID: NewTestId
새로운 PASSWORD: NewTestPw
새로운 ID: NewTestId
새로운 PASSWORD: NewTestPw

학생계정 ID: NewTestId
학생계정 Password: NewTestPw
노태영님 환영합니다.
```

## 계정 삭제와 로그인

```
입력: 6
성: 노
이름: 태영
학번: 60211660
ID: TestId
PASSWORD: TestPw
노태영님의 계정을 삭제했습니다.

학생계정 ID: TestId
학생계정 Password: TestPw
입력한 ID와 PASSWORD에 맞는 계정이 없습니다.
재시도 하시겠습니까?
1: 재시도
2: 로그인 종료
입력: 2
```

학생 객체를 생성, 수정하는 경우에도 데이터 무결성을 지키기 위한 예외처리를 해줘야 한다. 먼저 StudentAccount 테이블은 Student 테이블의 StudentId를 ForeignKey로 가지고 있기 때문에 만약 Student에 존재하는 StudentId가 아니라면 Referential-Integrity 오류가 발생한다. 또한 StudentAccount 테이블의 PrimaryKey인 ID가 중복되는 경우도 Entity-Integrity 오류가 발생한다. 따라서 이러한 예외처리를 통해 결과를 사용자에게 리턴해야 한다. 계정 수정도 마찬가지로 동일한 ID로 수정 요청이 들어오면 Entity-Integrity 오류가 발생하기 때문에 예외처리가 필요하다.

Database를 구현할 때 StudentAccount 테이블의 ForeignKey인 StudentId에 여러 개의 계정을 생성할 수 없도록 Unique를 부여하였다. 따라서 일대일의 Cardinality를 지키기 때문에 이미 계정이 존재하는 사용자가 계정을 생성하는 것 또한 예외처리하고 결과를 리턴한다.

따라서 이미 존재하는 ID로 계정을 생성, 수정하는 경우, 이미 학생 계정 Entity가 존재하는 학생의 계정을 생성하는 경우, 학생 Entity가 존재하지 않는 학생의 계정을 생성하는 경우 이 모든 요청에 대한 예외처리를 해줘야 한다.

이미 존재하는 ID 의 계정 생성

```
영문 이름 입력
성: 노
이름: 태영
학번: 60211660
ID: Minsu
Password: TestPw
이미 존재하는 아이디입니다.
다른 아이디로 회원가입하세요.
```

이미 학생계정 Entity 가 있는 학생의 계정 생성

```
영문 이름 입력
성: 노
이름: 태영
학번: 60211660
ID: TestId
Password: TestPw
이미 등록된 회원입니다.
```

예외처리 코드

```
catch (ExecuteQueryException sqlError) {
    if(sqlError.getMessage().contains(STUDENTID_Unique)) result = "AlreadyExistAccount";
    else if (sqlError.getMessage().contains(ACCOUNTID_PrimaryKey)) result = "AlreadyExistId";
}
```

이미 존재하는 ID 로 계정수정

```
현재 ID: NewTestId
현재 PASSWORD: NewTestPw
새로운 ID: Minsu
새로운 PASSWORD: Password
이미 존재하는 ID입니다. 다른 ID를 사용해주세요.
```

예외처리 코드

```
catch (ExecuteQueryException sqlError) {
    if(sqlError.getMessage().contains(ACCOUNTID_PrimaryKey)) result = "AlreadyExistId";
}
```

존재하지 않는 학생 Entity 의 계정생성 요청

```
영문 이름 입력
성: no
이름: taeyoung
학번: 20020914
ID: Minsu
Password: testpw
학생이 존재하지 않습니다. 학교에 문의하여 먼저 등록하세요.
```

예외처리 코드(DAO 에서 executeUpdate 가 0 일 때 NullDataException 을 throw)

```
public boolean create(String query) throws NullDataException, ExecuteQueryException {
    try {
        System.out.println(query);
        statement = connection.createStatement();
        int insertValueNumber = statement.executeUpdate(query);
        if(insertValueNumber == 0) throw new NullDataException("NullData");
        else return true;
    } catch (SQLException sqlError) {
        throw new ExecuteQueryException(sqlError.getMessage());
    }
}
```

### 3-5. 수강신청, 삭제

수강신청은 학생클라이언트에서 수행하는 기능이다. 먼저 개설된 모든 강좌를 요청하고 출력하여 사용자는 그 중 하나의 강좌를 선택하고 서버에게 전달한다. 서버에는 학생의 토큰과 선택한 강좌가 전달되고 토큰을 통해서 학생 ValueObject, 선택한 강좌 ValueObject 를 만든다. 이를 이용하여 총 5 가지의 조건을 따르고 모두 만족할 시 Database 의 ApplicationList 에 저장한다.

수강신청 API

```
@Override
public void choiceNewCourse(CrudApplicationRequest request, StreamObserver<ResultResponse> responseObserver) {
    ResultResponse response = null;
    String result = null;
    try {
        Student student = tokenGenerator.getStudent(request.getToken());
        Course choiceCourse = request.getCourse();
        List<Course> applicationList = courseDao.getApplicationList(student.getStudentId());
        result = insertCourseInApplicationList(choiceCourse, applicationList, student);
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullDataException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        result = "ServerError";
    } catch (ServerErrorException serverError) {
        result = serverError.getMessage();
    } finally {
        response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```

제약조건을 따지는 메서드

```
private String insertCourseInApplicationList(Course choiceCourse, List<Course> applicationList, Student student) throws NullDataException, ExecuteQueryException {
    boolean noneOverMaximumCredit = student.getMaximumCredit() >= getApplicationCredit(applicationList) + choiceCourse.getCredit();
    boolean noneSameCourse = checkNoneSameCourse(applicationList, choiceCourse);
    boolean noneOverMaximumStudent = choiceCourse.getMaximumStudent() > choiceCourse.getNumberOfStudent();
    boolean attendedRequirementCourse = checkRequirementCourse(choiceCourse, student.getAttendedCourseList());
    boolean noneOverTwoReApplication = checkOverTwoReApplication(applicationList, student.getAttendedCourseList());

    if(noneSameCourse && noneOverMaximumCredit && noneOverMaximumStudent && attendedRequirementCourse && noneOverTwoReApplication)
        return applicationListDao.insertCourseInApplicationList(student.getStudentId(), choiceCourse.getCourseId());
    else if(!noneOverMaximumCredit) throw new NullDataException("AlreadyFullMaximumCredit");
    else if(!noneSameCourse) throw new NullDataException("AlreadyExistInApplicationList");
    else if(!noneOverMaximumStudent) throw new NullDataException("AlreadyFullMaximumStudent");
    else if(!noneOverTwoReApplication) throw new NullDataException("OverTwoReApplication");
    else if(!attendedRequirementCourse) throw new NullDataException("RequirementExist");
    else throw new NullDataException("ServerError");
}
```

제약조건은 총 다섯가지가 존재한다.

1. 최대 신청학점을 초과했는지에 대한 여부 (noneOverMaximumCredit)
2. 이미 동일한 강좌를 선택했는지에 대한 여부(noneSameCourse)
3. 신청한 강좌의 최대 신청인원이 초과했는지에 대한 여부(noneOverMaximumStudent)
4. 선수과목 요구사항을 이수했는지에 대한 여부(attendedRequirementCourse)
5. 한 학기에 2 개로 제한하는 재수강을 2 개를 초과하여 선택했는지에 대한 여부 (noneOverTwoReApplicaiton)

이처럼 총 다섯가지의 제약조건이 존재하고 코드 실행을 통해서 각각의 예외가 발생하면 사용자에게 이를 리턴한다. 모든 제약조건을 충족했다면 강좌를 성공적으로 신청한 결과를 사용자에게 리턴한다.

(모든 제약조건을 데이터 변경없이 수행하는 부분에서는 무리가 있어 DBMS 를 이용하여 임의로 값을 수정하고 해당 제약조건을 충족하지 못하도록 하였다. 예를 들어 학생의 최대 수강학점을 2 학점으로 바꿔 최대 신청학점 초과 예외에 걸리도록 하거나, 강좌의 최대 수강 인원을 0 명으로 설정하여 결과를 출력해보고 정상작동 하는지를 판단했다)

1. 최대 신청학점을 초과했는지에 대한 여부 (해당 학생의 MaximumCredit 속성을 2 로 바꾸고 3 학점 강좌를 신청한 경우)

```
6번 강좌 -->> 강좌명: Architectures_of_Software_Systems, 강좌ID: 17655, REQUIREMENT1: 12345, REQUIREMENT2: 17651, 학점: 3,
신청할 강좌의 번호를 입력해주세요
입력: 6
신청 가능학점을 초과했습니다.
```

2. 이미 동일한 강좌를 선택했는지에 대한 여부 (1 번 강좌 연속 선택)

<pre>1번 강좌 --&gt;&gt; 강좌명: 소프트웨어_아키텍처, 2번 강좌 --&gt;&gt; 강좌명: C++_Programming 3번 강좌 --&gt;&gt; 강좌명: Managing_Softwa 4번 강좌 --&gt;&gt; 강좌명: Methods_of_Soft 5번 강좌 --&gt;&gt; 강좌명: Analysis_of_Sof 6번 강좌 --&gt;&gt; 강좌명: Architectures_o  신청할 강좌의 번호를 입력해주세요 입력: 1 성공적으로 신청했습니다.</pre>	<pre>1번 강좌 --&gt;&gt; 강좌명: 소프트웨어 2번 강좌 --&gt;&gt; 강좌명: C++_Pro 3번 강좌 --&gt;&gt; 강좌명: Managin 4번 강좌 --&gt;&gt; 강좌명: Methods 5번 강좌 --&gt;&gt; 강좌명: Analysi 6번 강좌 --&gt;&gt; 강좌명: Archite  신청할 강좌의 번호를 입력해주세요 입력: 1 동일한 강좌가 이미 신청됐습니다.</pre>
--	---

3. 신청한 강좌의 최대 신청인원이 초과했는지에 대한 여부 (강좌의 최대 신청인원을 0 명으로 설정)

```
1번 강좌 -->> 강좌명: 소프트웨어_아키텍처, 강좌ID: 78945, REQUIREMENT1: 0, REQUIREMENT2: 0, 학점: 3, 교수이름: ChoiSungwoon, 교수ID: 1354, 최대수강자수: 0, 신청 학생수: 0

1번 강좌 -->> 강좌명: 소프트웨어_아키텍처,
2번 강좌 -->> 강좌명: C++_Programmin
3번 강좌 -->> 강좌명: Managing_Softw
4번 강좌 -->> 강좌명: Methods_of_Sof
5번 강좌 -->> 강좌명: Analysis_of_So
6번 강좌 -->> 강좌명: Architectures_

신청할 강좌의 번호를 입력해주세요
입력: 1
최대수강생을 초과합니다.
```

(Database 에서 강좌를 읽어올 때 신청한 인원 수까지 계산하여 강좌의 필드로 저장하기 때문에 이러한 제약조건을 설정하는 것이 가능하다)

4. 선수과목 요구사항을 이수했는지에 대한 여부(AttendedList 테이블을 조회함)

```
6번 강좌 -->> 강좌명: Architectures_of_Software_Systems, 강좌ID: 17655, REQUIREMENT1: 12345,
신청할 강좌의 번호를 입력해주세요
입력: 6
선수과목이 수행되지 않았습니다.
```

5. 한 학기에 2 개로 제한하는 재수강을 2 개를 초과하여 선택했는지에 대한 여부

이전 수강내역

현재 수강신청 내역

이전 수강내역	현재 수강신청 내역
2: 이전 수강내역 출력	3: 수강신청 내역 출력
3: 수강신청 내역 출력	4: 수강신청 변경
4: 수강신청 변경	5: ID/비밀번호 변경
5: ID/비밀번호 변경	6: 계정삭제
6: 계정삭제	X: 종료
X: 종료	입력: 3
입력: 2	1번 강좌 -->> 강좌명: C++_Programming, 강좌ID: 23456, RE
1번 강좌 -->> 강좌명: C++_Programming, 강좌ID: 23456, RE	1번 강좌 -->> 강좌명: C++_Programming, 강좌ID: 23456, RE
2번 강좌 -->> 강좌명: Managing_Software_Development, 강	2번 강좌 -->> 강좌명: Methods_of_Software_Development,
2번 강좌 -->> 강좌명: Managing_Software_Development, 강	
3번 강좌 -->> 강좌명: Methods_of_Software_Development,	

2 개를 초과하는 재수강 과목 선택

```
1번 강좌 -->> 강좌명: 소프트웨어_아키텍처, 강
2번 강좌 -->> 강좌명: C++_Programming,
3번 강좌 -->> 강좌명: Managing_Software_Development, 강
4번 강좌 -->> 강좌명: Methods_of_Software_Development, 강
5번 강좌 -->> 강좌명: Analysis_of_Software_Development, 강
6번 강좌 -->> 강좌명: Architectures_of_Software_Systems, 강

신청할 강좌의 번호를 입력해주세요
입력: 3
한 학기에 2개를 초과하는 재수강을 금지합니다.
```

위 다섯가지의 제약조건을 모두 통과하여 최종적으로 강좌를 신청하는 결과와 조회

신청 과정	결과
1번 강좌 -->> 강좌명: 소프트웨어_아키텍처, 강	입력: 3
2번 강좌 -->> 강좌명: C++_Programming	1번 강좌 -->> 강좌명: Methods_of_Software_Development
3번 강좌 -->> 강좌명: Managing_Software_Development, 강	2번 강좌 -->> 강좌명: Managing_Software_Development
4번 강좌 -->> 강좌명: Methods_of_Software_Development, 강	3번 강좌 -->> 강좌명: 소프트웨어_아키텍처, 강좌ID: 78945
5번 강좌 -->> 강좌명: Analysis_of_Software_Development, 강	
6번 강좌 -->> 강좌명: Architectures_of_Software_Systems, 강	
신청할 강좌의 번호를 입력해주세요	
입력: 1	
성공적으로 신청했습니다.	

## 신청강좌 삭제 API

```
@Override
public void deleteApplication(CrudApplicationRequest request, StreamObserver<ResultResponse> responseObserver) {
    String result = null;
    try {
        Student student = tokenGenerator.getStudent(request.getToken());
        result = applicationListDao.deleteCourseInApplicationList(student.getStudentId(), request.getCourse().getCourseId());
    } catch (TokenException tokenError) {
        result = tokenError.getMessage();
    } catch (NullDataException nullDataError) {
        result = nullDataError.getMessage();
    } catch (ExecuteQueryException sqlError) {
        result = "ServerError";
    } finally {
        ResultResponse response = ResultResponse.newBuilder().setResult(result).build();
        responseObserver.onNext(response);
        responseObserver.onCompleted();
    }
}
```

## ApplicationListDao

```
public String deleteCourseInApplicationList(int studentId, int courseId) throws ExecuteQueryException, NullDataException {
    String deleteCourseInApplicationQuery = constant.getDeleteCourseInApplicationQuery(studentId, courseId);
    if(super.delete(deleteCourseInApplicationQuery) == 0) throw new NullDataException("NullData");
    else return "Success";
}
```

신청강좌 삭제는 클라이언트 본인이 신청한 수강신청 내역을 요청하고 출력하여 사용자는 그 중 삭제할 하나의 강좌를 선택하고 서버에게 전달한다. 서버에는 학생의 토큰과 삭제할 강좌가 전달되고 토큰을 통해서 학생의 StudentId를 가져오고, 선택한 강좌의 ValueObject를 가져와 ApplicationList에 존재하는 신청 Entity를 삭제하고 결과를 사용자에게 리턴한다.

## 삭제할 강좌 선택과 결과

```
1번 강좌 -->> 강좌명: Methods_of_Software_Development
2번 강좌 -->> 강좌명: Managing_Software_Development,
3번 강좌 -->> 강좌명: 소프트웨어_아키텍처, 강좌ID: 78945, RE

삭제할 강좌의 번호를 입력해주세요
입력: 3
선택한 강좌를 삭제했습니다.
```

## 수강신청 내역 조회

```
3: 수강신청 내역 출력
4: 수강신청 변경
5: ID/비밀번호 변경
6: 계정삭제
X: 종료
입력: 3
1번 강좌 -->> 강좌명: Methods_of_Software_Development,
2번 강좌 -->> 강좌명: Managing_Software_Development, 강
```

## 4. 프로젝트 결과와 소감

2 달동안 프로젝트를 진행하며 멘토링에서 다양한 지적과 개선사항을 요구받았다. 보다 더 적은 코드로 더 나은 기능과 완성성을 추구하는 방법을 지속적으로 생각했고 이를 코드에 반영했다. 3-1 에서 설명하는 로그인 기능은 원래 500 줄을 넘는 코드의 양을 가지고 있었다. 그러나 이를 지속적으로 개선하고 더 나은 코드로 만들고자 노력했다. 결과적으로 Login API 와 AccountDao 를 합쳐도 200 줄이 안되는 코드로 만들었으며 동일한 기능을 수행하도록 이를 개선하였다. 마지막 멘토링 이전에 위에서 설명한 기능들의 구현은 이미 끝난 상태였으나 기능을 유지하며 코드를 줄이는 과정을 반복했고 결과적으로 상당히 많은 부분을 개선하였다. 다만 아쉬웠던 부분도 존재한다. 더욱 다양한 상황에서 코드를 테스트해보지 못한 것이 상당히 아쉬운 부분으로 남는다. 다수의 클라이언트를 서버가 처리해도 속도에 대한 문제가 없으며 오류 없이 정상작동 하는지 테스트를 진행하지 못했다. 단순히 서버가 기능을 제공하는 것 이상의 안정성에 대한 검토가 부족했다. 예를 들어 데이터를 처리하는 작업에서 너무 많은 테이블 Join 이 발생하고 이것이 성능 저하를 불러오는 수준인지 Database 수업 교수님께 따로 질문하였다. 많은 쿼리가 테이블 Join 을 포함하고 이 정도면 충분히 성능 저하를 가져올 수 있다는 지적을 받았으나 한번 설계한 Database 의 구조를 바꾸는 작업은 쉽지 않았으며 성능 개선에 대한 검토가 매우 부족했다.

이 프로젝트를 진행하며 단순히 기능을 구현하는 것은 전혀 어렵지 않다는 생각이 들었으나 그것보다 더 중요한 적은 양의 코드로 어떻게 안정적이고 빠른 프로그램을 구현할지에 대한 개선 과정은 매우 어려웠으며 다양한 환경에서 테스트하는 것이 쉽지 않았다. 필자는 이전까지 여러 프로젝트에서 애자일 방식을 이용하여 체계적인 설계보다는 구현을 먼저 진행하며 구현과 수정을 반복하여 프로젝트를 진행했다. 그리고 이번 프로젝트에서도 유사한 방식으로 진행했으나 기본적인 틀을 갖추지 않은 상태에서 구현을 먼저 수행한 결과 생각보다 많은 부분이 목표했던 방향과는 달라져 있었다. 그리고 그것을 수정하는 부분에서 가장 많은 시간을 소요했으며 시간을 단축하여 더 짧고, 안정적인 코드를 만들지 못한 것이 아쉬운 부분으로 남는다. 애자일 방식으로 프로젝트를 진행하더라도 바로 구현부터 들어가는 것이 아닌 전체적인 틀(UseCase 다이어그램 혹은 ER 다이어그램)을 설계하고 그 이후에 애자일 방식을 적용하는 것이 목표와 맞는 더 좋은 결과물을 만들 수 있다는 것을 깨달았다.

따라서 중간시험 이후의 EventBus 프로젝트에서는 위에서 설명한 아쉬웠던 부분들을 개선하여 프로젝트의 방향성을 잘 세우고 안정적인 프로그램을 더욱 간결하게 구현해보고자 한다.