Database

Assignment

#01

Name: 조민혁

Department: Mobile Systems Engineering

Student number: 32204292

Index

1. Introduction

그림 1. 데이터베이스 종류

현대 사회는 디지털화와 함께 데이터 중심으로 빠르게 전환되고 있으며, 이에 따라 생성되는 데이터의 양과 종류 또한 기하급수적으로 증가하고 있다. 데이터는 Primary Data, Secondary Data 뿐만 아니라 Qualitative Data, Quantitative Data, Internal Data, External Data 등 다양한 속성과 출처를 바탕으로 분류된다. 이처럼 복잡하고 방대한 데이터를 효과적으로 저장하고 관리하기 위해 파일 시스템(File System)이 도입되었지만, 파일 시스템은 데이터 중복, 데이터 종속성, 데이터 무결성 제어 부족 등의 한계를 지니고 있었다.

이러한 문제를 해결하고자 등장한 것이 바로 데이터베이스(Database) 시스템이다. 데이터베이스는 데이터를 구조화하여 저장하고, 사용자와 응용 프로그램이 이를 효율적으로 접근, 검색, 분석할 수 있도록 지원한다. 특히 DBMS (Database Management System)는 데이터의 무결성, 보안성, 일관성을 보장하면서도 데이터에 대한 다양한 연산을 지원하며, 기존 파일 시스템의 단점을 보완한다. 그림 1 은 현대 데이터 환경에서 사용되는 데이터베이스 시스템의 유형을 보여준다. 정형 데이터(Structured Data)를 다루는 관계형 데이터베이스(RDBMS), 반정형 데이터(Semi-Structured Data)를 처리하는 XML, JSON 기반 저장소, 비정형 데이터(Unstructured Data)를 위한 NoSQL 및 객체 저장소 등이 포함된다. 이러한 배경 속에서 데이터베이스는 산업, 과학, 공공기관 등 전 분야에서 핵심적인 기술로 자리매김하고 있다. 본 레포트에서는 대표적인 공개 센서 데이터셋인 HAR-UCI 를 기반으로 관계형 데이터베이스를 설계하고, MySQL 을 활용하여 데이터베이스를 구축한 후, SQL 쿼리를 통해 다양한 분석 작업을 수행한다. 나아가 새로운 테스트 데이터를 기반으로 활동을 예측하는 모델을 구현함으로써 데이터베이스의 실제 응용 가능성을 탐색한다.

본 레포트의 구성은 다음과 같다. 2 장은 본 과제를 수행함에 있어서 필요한 요구사항들을 살펴본다. 3 장에서는 과제를 수행하기전 필요한 기본 개념들을 살펴본다. 4 장에서는 2 장과 3 장의 내용을 바탕으로 과제를 구현한 코드 및 해당 코드 실행 결과를 본다. 마지막으로 5 장에서는 결론을 서술하며 본 레포트를 마무리한다.

2. Requirements

표 1. Requirements Table

Tasks	Index	Requirements
Task 1	1	Create Tables: Users, Activities, SensorReadings
	2	Users Table: user_id is Primary Key Activities Table: activity_id is Primary Key SensorReadings Table - reading_id is Primary Key with AUTO_INCREMENT - user_id is Foreign Key referencing Users(user_id) - activity_id is Foreign Key referencing Activities(activity_id)
	3	Users Table - Load from '/content/drive/MyDrive/Colab Notebooks/subject_records.txt' - Insert values as user_id Activities Table - Load from '/content/drive/MyDrive/Colab Notebooks/activity_labels.txt' - Insert activity_id and activity_name SensorReadings Table - Load from '/content/drive/MyDrive/Colab Notebooks/sensor_records.txt' - tBodyAcc-mean()-X, tBodyAcc-mean()-Y, tBodyAcc-mean()-Z, tBodyGyro-mean()-X, tBodyGyro-mean()-Y, tBodyGyro-mean()-Z -> AccelerometerX, AccelerometerY, AccelerometerZ, GyroscopeX, GyroscopeY, GyroscopeZ
	4	An ERD (Entity-Relationship Diagram Image) - Show the structure and relationships of Three tables Results of the SQL Queries from 'query_to_df' and Screenshots
Task 2	1	Execute a set of SQL Queries based on the database you created in Task 1
	2	Write and run the corresponding SQL query and include the screenshot and brief explanation of your output in the final report.
Task 3	1	Predict the most likely activity for each row by comparing it to statistical information derived from your database (created in Task 1).
	2	Explan your prediction method in the report.
	3	Include any formulas or reasoning for choosing your approach.
	4	Present a final result table (task3_result_pd).

본 과제를 수행하는데 있어 필요한 요구사항들이 Table 1 에 나타나 있다.

3. Concepts

본 장에서는 과제를 수행함에 있어 필요한 개념을 살펴본다.

3-1. Database (DB)

데이터베이스는 데이터를 효율적으로 저장, 관리, 검색, 분석하기 위해 고안된 구조화된 데이터의 집합이다. 초기에는 파일 시스템을 통해 데이터를 저장하였지만 이는 데이터 중복, 일관성 결여, 확장성 부족 등의 문제를 야기했다. 이에 대한 대안으로 등장한 것이 데이터베이스이며 이는 데이터베이스 관리 시스템을 추가하여 데이터의 중복 최소화, 무결성 보장, 독립성 확보를 가능하게 한다. 데이터베이스는 단순히 데이터를 담는 저장소가 아니라, 다음과 같은 구성 요소를 포함하는 데이터베이스 관리 시스템을 통해 운영된다.

먼저 데이터는 저장하여 관리하고자 하는 대상을 말한다. 이는 데이터 자체만으로는 의미가 없지만 데이터 집합, 가공된 데이터를 통해 정보로써의 의미를 가진다. 그 다음으로는 스키마이다. 스키마는 데이터 구조를 정의한 설계도로 테이블 구조를 정의하며 개념적 스키마, 논리적 스키마 등의 개념을 가진다. 질의어는 데이터를 조작하거나 조회하는데 있어 사용하는 언어이다. 이를 통해 데이터베이스에 저장된 데이터를 실험, 분석, 조작이 가능해진다. 마지막으로 제약 조건을 통해 데이터의 무결성과 정확성을 보장할 수 있다. 또한 데이터베이스의 조건을 저장하여 기준이 되는 데이터만 저장하고 조작할 수 있다.

또한 데이터베이스는 크게 다음과 같은 유형으로 분류될 수 있다. 먼저 관계형 데이터베이스는 데이터를 행과 열 기반의 테이블로 표현하며 대표적으로 MySQL, PostgreSQL 이 존재한다. 그 다음으로는 비관계형 데이터베이스가 존재하며 문서형, 키-값형, 그래프형 등 다양한 형태로 데이터를 저장할 수 있다. 대표적으로 MongoDB, Cassandra 가 존재한다.

본 과제에서 사용되는 MySQL 은 관계형 데이터베이스 시스템으로 테이블 간의 관계를 명시적으로 설정할 수 있으며 SQL 을 이용해 데이터 삽입, 검색, 수정, 삭제 등 다양한 연산을 수행할 수 있다.

따라서 데이터베이스는 단순한 저장을 넘어서 데이터 분석과 지능적 예측의 핵심 기반 역할을 수행하며 실시간 처리, 통계적 질의, 예측 분석 등의 응용 기술과 결합되어 현대 사회의 데이터 중심 의사결정 구조를 가능하게 만든다.

3-2. Database Management System (DBMS)

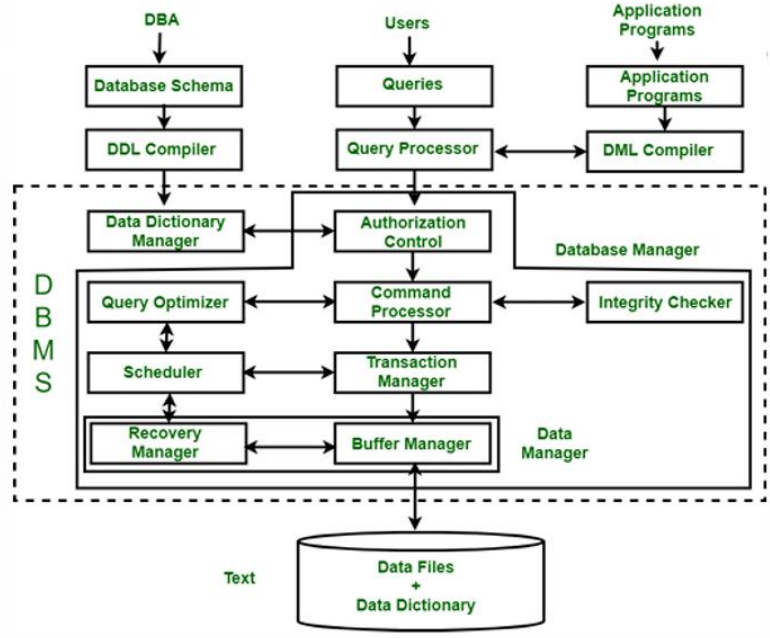


그림 2. DBMS Architecture

DBMS 는 데이터베이스를 효율적으로 관리하기 위한 핵심 소프트웨어 시스템으로 사용자 및 응용 프로그램이 데이터에 접근하고 조작할 수 있도록 인터페이스를 제공한다. 단순한 데이터 저장 기능을 넘어서 데이터 무결성, 일관성, 독립성, 보안성을 보장하며 질의와 데이터 조작을 위한 체계적인 환경을 구성한다. 그림 2 는 전형적인 DBMS 의 아키텍처를 시각적으로 나타낸 것이다. 이 구조는 사용자(User) 데이터베이스 관리자(DBA), 응용 프로그램(Application Program)과 DBMS 사이의 계층적 상호작용을 보여준다.

먼저, DBA(Database Administrator)는 시스템의 스키마(Schema)를 정의하고, 이를 DDL(데이터 정의 언어) Compiler 에 전달하여 내부적으로 테이블 구조, 제약조건, 인덱스 등을 설정한다. 사용자는 SQL 기반 질의어를 통해 데이터를 검색하거나 수정하며, 이는 Query Processor 를 거쳐 해석되고 실행 계획으로 변환된다. 한편, 응용 프로그램은 데이터 조작 명령어를 통해 DML Compiler 에 전달되며, 이를 통해 DBMS 는 명령을 실행할 수 있도록 준비한다. 이후 모든 명령어는 DBMS 의 Authorization Manager 를 통과하여 적절한 사용자만이 데이터에 접근할 수 있도록 제어된다. 다음 단계에서는 Data Dictionary Manager 가 작동하여 메타데이터 및 스키마 정보를 기반으로 데이터 구조의 유효성 및 중복성을 검사한다. 명령어가 승인되면, DBMS 는 실행을 최적화하고 효율적인 자원 사용을 위해 Query Optimizer, 스케줄러(Scheduler), Transaction Manager 를 통해 일관된 처리를 수행한다. 이 과정에서 동시성 제어, 트랜잭션 격리 수준, 롤백 및 커밋 등이 함께 관리된다. 마지막으로 Buffer Manager 와 Recovery Manager 는 메모리와 디스크 간의 데이터 입출력을 효율화하고, 시스템 오류 발생 시 데이터 복구 기능을 제공한다.

이처럼 DBMS 는 단순한 데이터 저장 장치가 아니라 데이터베이스 환경에서 데이터 제어, 최적화, 보안, 트랜잭션 처리를 종합적으로 담당하는 고급 소프트웨어 시스템이다. 이를 통해 사용자는

복잡한 내부 구조를 알지 못해도 고수준의 질의어만으로 안정적이고 효율적으로 데이터를 처리할 수 있다.

3-3. Entity-Relation Diagram (ERD)

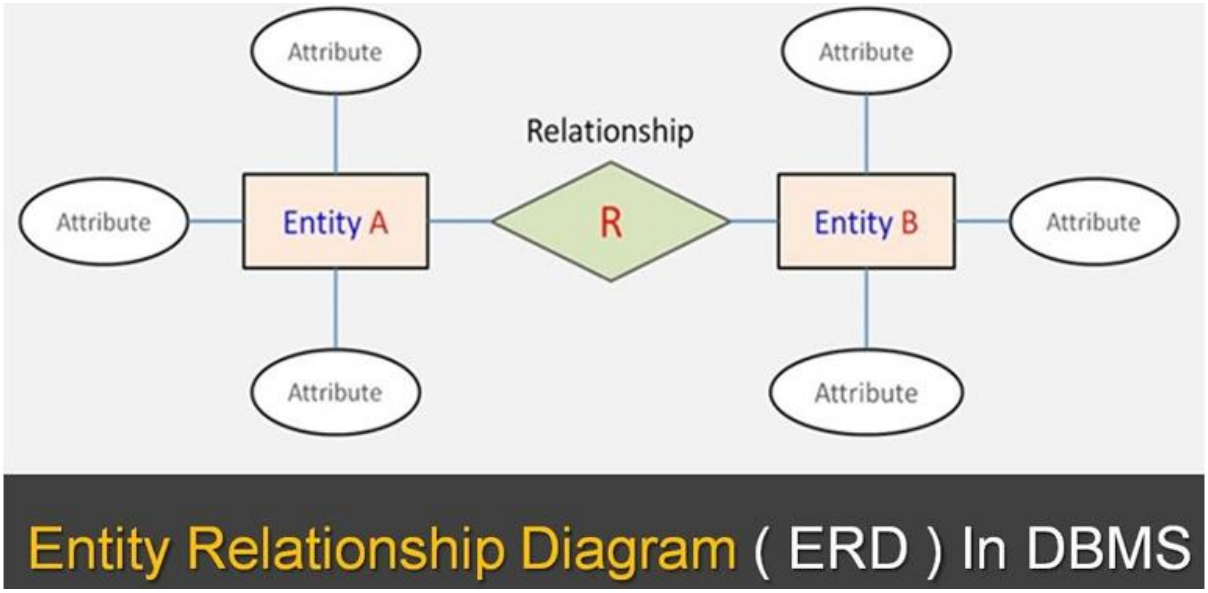


그림 3. Entity-Relation Diagram in DBMS

ERD(Entity-Relation Diagram)는 데이터베이스의 구조를 개념적으로 모델링하기 위한 다이어그램으로 현실 세계의 데이터를 추상화하여 Entity, 속성, 관계로 표현한다. 그림 3 은 이러한 개념을 시각적으로 도식화한 그림이다.

Entity 는 독립적으로 존재할 수 있는 객체 또는 개념으로 각각의 Entitiy 는 고유한 Attribute 를 가지며 이는 Entitiy 가 표현하는 대상의 특징을 나타낸다. Relationship 는 두 개 이상의 Entitiy 간의 논리적인 연관성을 의미하며 관계는 Entitiy 사이의 상호작용을 모델링한다. ERD 에서는 Entitiy 를 직사각형, 속성을 타원, 관계를 마름모로 나타내며 속성 중에서도 엔터티를 고유하게 식별할 수 있는 기본 키는 밑줄로 표시된다.

관계에는 참여도(Cardinality)와 선택성(Participation)이 정의되어 Entitiy 간의 연결 방식이 명확하게 표현되며 이는 일대일(1:1), 일대다(1:N), 다대다(M:N) 관계로 구분된다. 약한 Entitiy (Weak Entity)는 자체적으로 기본 키를 가지지 못하고 다른 Entitiy 에 의존하며 이때 식별 관계(Identifying Relationship)를 통해 연결된다. ER 모델은 개념적 설계 단계에서 사용되어 요구사항 분석을 구조적으로 도출하고, 이를 기반으로 관계형 모델로의 변환을 통해 논리적·물리적 데이터베이스 설계로 이어지는 중간 단계의 역할을 수행한다.

따라서 ERD 는 데이터베이스 시스템에서 데이터 구조를 명확히 정의하고, 무결성과 일관성을 유지하며 체계적인 스키마 설계를 위한 필수적인 도구로 간주된다.

3-4. Standard Query Language (SQL)

SQL(Structured Query Language)은 관계형 데이터베이스에서 데이터를 정의하고, 조작하며, 제어하기 위한 표준화된 질의 언어이다. ISO(국제표준화기구) 및 ANSI(미국표준협회)에 의해 표준으로 정의된 SQL 은 대부분의 관계형 데이터베이스 관리 시스템(DBMS)에서 공통적으로 사용되며 사용자가 데이터베이스와 직접 소통할 수 있는 주요 수단이다.

SQL 의 주요 목적은 데이터의 생성, 검색, 수정, 삭제를 가능하게 하는 것이며, 명령어 유형에 따라 크게 DDL(Data Definition Language), DML(Data Manipulation Language), DCL(Data Control Language)의 세 가지로 분류된다.

먼저, DDL 은 데이터베이스의 구조를 정의하는 데 사용된다. 대표적인 명령어로는 CREATE, DROP, ALTER 등이 있으며 이를 통해 테이블이나 뷰, 인덱스와 같은 데이터베이스 객체를 생성하거나 삭제하고 구조를 변경할 수 있다. 예를 들어, 테이블을 생성하는 CREATE TABLE 문은 테이블의 이름과 컬럼 구조, 제약 조건 등을 명시하여 데이터 저장 구조를 정의한다.

다음으로, DML 은 데이터 자체를 조작하는 데 사용되며 SELECT, INSERT, UPDATE, DELETE 등이 이에 해당한다. SELECT 는 데이터를 조회하는 데 사용되며 WHERE, GROUP BY, ORDER BY 등의 절과 함께 조합하여 복잡한 조건과 정렬, 집계를 수행할 수 있다. 본 과제에서는 이 SELECT 문을 활용하여 사용자별 평균 센서값을 계산하고, 활동별 통계를 출력하였다.

마지막으로 DCL 은 데이터 접근 권한을 제어하는 명령어로 GRANT 와 REVOKE 가 대표적이다. 이는 다중 사용자 환경에서 데이터 보안 및 접근 제어를 구현하는데 사용된다.

4. Implements and Results

4-1. Build Environment

구현된 코드를 설명하기에 앞서 실행 환경 및 프로그램을 실행하기 위한 환경 설정은 다음과 같다.

- 개발 환경: Colab
- 프로그래밍 언어: Python
- 프로그램 실행 순서: Colab 접속 -> Note 열기 -> CTRL+F9 (모든 Shell 실행)

4-2. Implements of the Entity-Relation Diagram

그림 4. ER Diagram version1

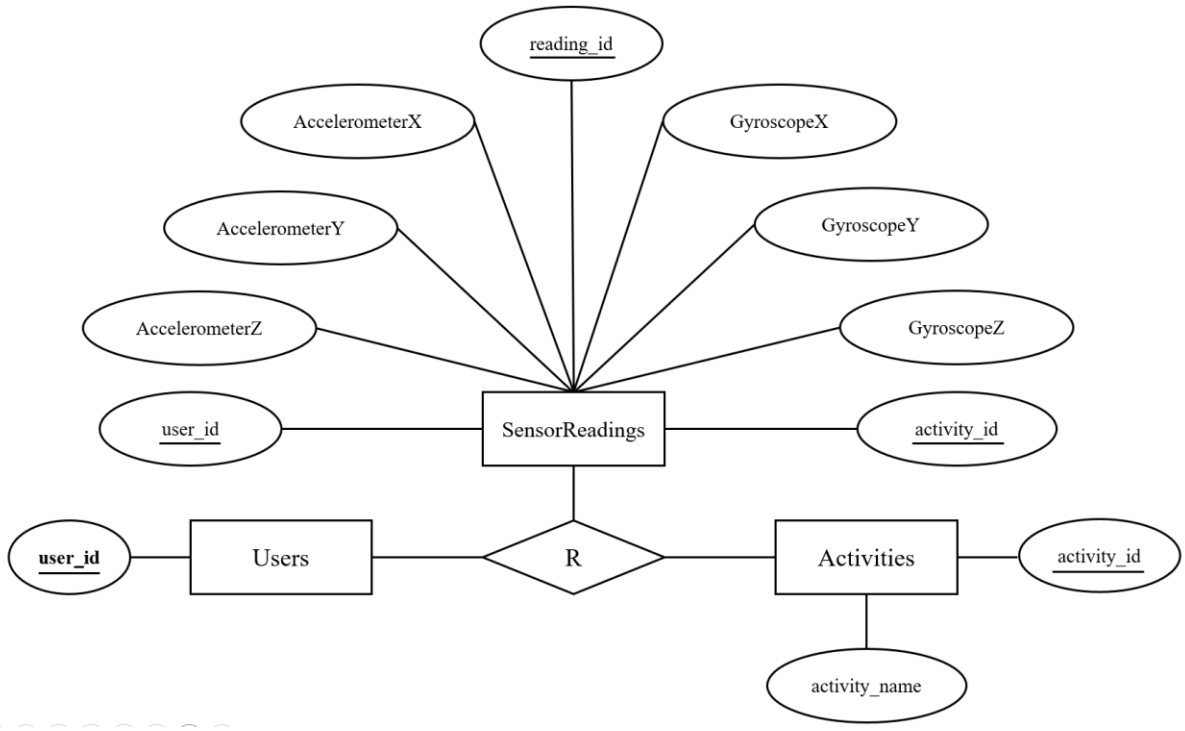


그림 5. ER Diagram version2

그림 4, 5 는 Users, Activities, SensorReadings 테이블의 관계를 보여주는 ER Diagram 이다. 이 ERD 는 관계형 데이터베이스 설계를 수행하는 데 있어 핵심적인 단계로 테이블 간의 논리적 연결 구조를 명확히 정의하고, 데이터 흐름의 방향성과 관계성을 직관적으로 파악할 수 있도록 한다.

먼저 Users 테이블은 사용자 정보를 저장하는 Entity 로 user_id 컬럼을 기본 키로 갖는다. 이는 각 사용자를 고유하게 식별하기 위한 값이며 이 값은 SensorReadings 테이블과의 Foreign Key 를 통해 연결된다. 즉, 하나의 User 는 여러 개의 센서 측정 기록을 가질 수 있으므로, 이 관계는 1:N 관계로 설계되었다.

Activities 테이블은 Activity 정보를 저장하며 activity_id 가 Primary Key 역할을 한다. 이 테이블은 각 활동의 고유한 식별 번호와 함께 해당 활동의 이름을 activity_name 컬럼에 저장한다. 이 테이블 역시 SensorReadings 테이블과 외래 키로 연결되며 하나의 활동은 여러 개의 센서 측정값들과 연관될 수 있으므로 역시 1:N 관계를 가진다.

SensorReadings 테이블은 본 데이터베이스에서 가장 핵심적인 Entity 로 실제 센서 데이터를 저장하는 역할을 수행한다. 이 테이블은 reading_id 를 Primary Key 로 하며 자동 증가되는 정수값을 통해 각 센서 측정 레코드를 고유하게 식별한다. user_id 와 activity_id 는 각각 Users 및 Activities 테이블과의 Foreign Key 로 설정되어 있어 센서 데이터가 어떤 사용자에 의해, 어떤 활동 중에 측정되었는지를 명확히 추적할 수 있게 한다. 이 외에도 AccelerometerX, AccelerometerY, AccelerometerZ, GyroscopeX, GyroscopeY, GyroscopeZ 와 같은 실수형 컬럼들은 실제 스마트폰 센서에서 수집된 6 가지 물리량을 저장하며, 각각 X, Y, Z 축 방향의 가속도와 각속도를 나타낸다.

이러한 ERD 설계는 정규화 원칙에 따라 각 데이터의 중복을 방지하고, 무결성을 유지할 수 있도록 구성되어 있다.

4-3. Task 1: Database Table Construction and ERD

```
cursor.execute("DROP TABLE IF EXISTS Users")

create_users_query = """
 CREATE TABLE Users (
  user_id INT,
  PRIMARY KEY (user_id)
 );
"""

cursor.execute(create_users_query)
```

그림 6. Users Table Schema Definition

그림 6 은 Users Table Schema 를 정의한 코드를 보여준다. 만약 기존에 Users 테이블이 존재하면 삭제하고 이후 create_users_query 를 통해 Table 을 생성하도록 쿼리를 두었다. 2 장에서 언급한 Requirements 에 따라 user_id 를 column 으로 두고 Primary Key 로 user_id 로 설정한 후 cursor 를 실행하여 테이블을 생성하였다.

```
users_file_path = base_path + 'subject_records.txt'

with open(users_file_path, 'r') as f:
  user_ids = f.readlines()

user_ids = [int(uid.strip()) for uid in user_ids]

unique_user_ids = sorted(set(user_ids))

insert_users_query = """
 INSERT
 INTO Users (user_id)
 VALUES (%s);
"""

for uid in unique_user_ids:
  cursor.execute(insert_users_query, (uid,))
```

그림 7. Insert Into Users Table Values

그림 7 은 생성한 Users Table 에 대해 값을 넣는 과정을 보여준다. 값을 넣기 위해 subject_records 파일을 불러온 후 라인을 읽어 가공한 후 Insert Query 를 통해 값을 넣도록 하였다.

그림 8. Table Results and query_to_df Results

그림 8 은 생성한 Users 테이블에 대해 값을 삽입한 후 결과 테이블과 query_to_df 함수에 describe_users_query 문과 select_users_query 문을 실행한 결과를 보여준다. 해당 실행 결과를 통해 user_id column 에 적절하게 값이 삽입된 것을 확인할 수 있고, query_to_df 함수를 실행함으로써 COUNT 개수가 21 개임과 user_id 에 대한 정보를 확인할 수 있다.

```
cursor.execute("DROP TABLE IF EXISTS Activities")

create_activities_query = """
 CREATE TABLE Activities (
  activity_id INT,
  activity_name VARCHAR(50) NOT NULL,
  PRIMARY KEY (activity_id)
 );
"""

cursor.execute(create_activities_query)
```

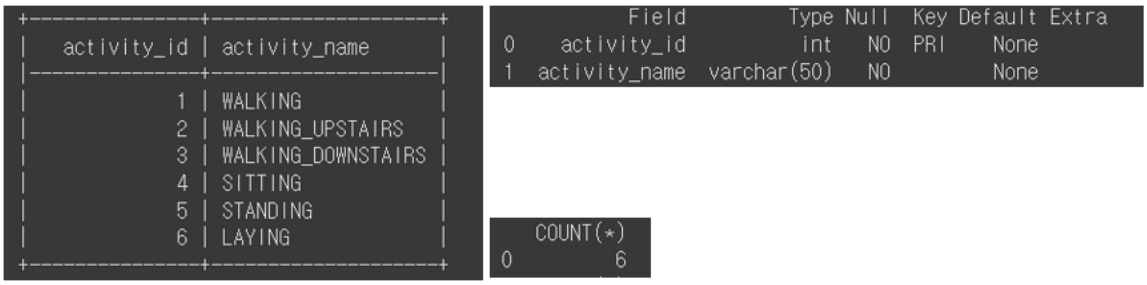
그림 9. Activities Table Schema Definition

그림 9 는 Activities Table Schema 에 대한 정의를 보여준다. Users 테이블과 같은 동작 흐름을 가지고 있고, 차이가 있다면 Users Table 은 1 column 으로 정의했다면 해당 테이블은 activity_id, activity_name 의 2 column 을 가지고 있었다.

```
activities_file_path = base_path + 'activity_records.txt'

with open(activities_file_path, 'r') as f:
  lines = f.readlines()

activity_data = []
for line in lines:
  parts = line.strip().split()
  activity_id = int(parts[0])
  activity_name = ' '.join(parts[1:])
  activity_data.append((activity_id, activity_name))

insert_activity_query = """
 INSERT
 INTO Activities (activity_id, activity_name)
 VALUES (%s, %s);
"""
```

그림 10. Insert Into Activities Table Values

그림 10 은 Activities 테이블에 대한 값을 삽입하는 과정을 보여준다. 해당 테이블은 2 column 이기에 activity_records 파일에서 값을 가져온 후 공백을 기준으로 값을 분리하는 과정이 필요했다. 이후 Users 테이블에 값을 넣는 과정과 동일하게 쿼리를 구성한 후 값을 삽입하였다.



activity_id	activity_name
1	WALKING
2	WALKING_UPSTAIRS
3	WALKING_DOWNSTAIRS
4	SITTING
5	STANDING
6	LAYING

	Field	Type	Null	Key	Default	Extra
0	activity_id	int	NO	PRI	None	
1	activity_name	varchar(50)	NO		None	

	COUNT(*)
0	6

그림 11. Activities Table Results and query_to_df Results

그림 11 은 Activities 테이블에 대한 결과와 query_to_df 에 대한 결과를 보여준다. Users 테이블과 동일한 쿼리를 사용했다. 왼쪽 그림을 통해 테이블에 column 이름과 올바르게 값이 삽입된 것을 확인할 수 있고, 각 column 에 대한 정보를 확인할 수 있다. 또한 COUNT 가 6 개로 올바르게 삽입된 것을 알 수 있다.

```
cursor.execute("DROP TABLE IF EXISTS SensorReadings")
```

```
create_sensorreadings_query = """
 CREATE TABLE SensorReadings (
  reading_id INT AUTO_INCREMENT,
  user_id INT,
  activity_id INT,
  AccelerometerX FLOAT,
  AccelerometerY FLOAT,
  AccelerometerZ FLOAT,
  GyroscopeX FLOAT,
  GyroscopeY FLOAT,
  GyroscopeZ FLOAT,
  PRIMARY KEY (reading_id),
  FOREIGN KEY (user_id) REFERENCES Users(user_id),
  FOREIGN KEY (activity_id) REFERENCES Activities(activity_id)
 );
"""
```

```
cursor.execute(create_sensorreadings_query)
```

그림 12. SensorReadings Table Schema Definition

그림 12 는 SensorReadings 테이블에 대한 스키마 정의를 보여준다. 이전 테이블과 동일하게 쿼리를 구성하여 실행하였다. 해당 테이블은 column 이 많기 때문에 올바르게 Primary Key 와 Foreign Key 를 사용해 적절하게 참조하도록 하였다.

```
features_path = base_path + 'features.txt'

target_features = [
    'tBodyAcc-mean()-X',
    'tBodyAcc-mean()-Y',
    'tBodyAcc-mean()-Z',
    'tBodyGyro-mean()-X',
    'tBodyGyro-mean()-Y',
    'tBodyGyro-mean()-Z'
]

with open(features_path, 'r') as f:
  lines = f.readlines()

feature_indices = []
for idx, line in enumerate(lines):
  feature_name = line.strip().split()[1]
  if feature_name in target_features:
    feature_indices.append(idx)

sensor_path = base_path + 'sensor_records.txt'
subject_path = base_path + 'subject_records.txt'
activity_path = base_path + 'sensor_activity_records.txt'
```

```
with open(sensor_path, 'r') as f:
  sensor_lines = f.readlines()

with open(subject_path, 'r') as f:
  user_ids = [int(line.strip()) for line in f]

with open(activity_path, 'r') as f:
  activity_ids = [int(line.strip()) for line in f]

assert len(sensor_lines) == len(user_ids) == len(activity_ids), "Data Length 일치 x"

sensor_insert_query = """
 INSERT
 INTO SensorReadings (
  user_id, activity_id,
  AccelerometerX, AccelerometerY, AccelerometerZ,
  GyroscopeX, GyroscopeY, GyroscopeZ
 )
 VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
"""
```

그림 13. Insert Into SensorReadings Values

그림 13 은 SensorReadings 테이블에 대해 적절하게 값을 넣는 과정을 보여준다. 해당 테이블에 값을 넣기 위해 3 개의 파일에서 값을 가져와서 적절하게 분리하여 값을 넣어주었다. 또한 column 명을 과제에서 요구한 대로 삽입하였으며 reading_id 는 AUTO_INCREMENT 로 설정했기에 따로 값을 삽입하지는 않았다. 또한 테이블에 삽입될 값은 각 행이 모두 같은 길이의 열을 가져야하기 때문에 그에 따른 assert 문을 사용해 동일 길이를 보장해주었다.

```
['reading_id', 'user_id', 'activity_id', 'AccelerometerX', 'AccelerometerY', 'AccelerometerZ', 'GyroscopeX', 'GyroscopeY', 'GyroscopeZ']
```

	Field	Type	Null	Key	Default	Extra
0	reading_id	int	NO	PRI	None	auto_increment
1	user_id	int	YES	MUL	None	
2	activity_id	int	YES	MUL	None	
3	AccelerometerX	float	YES		None	
4	AccelerometerY	float	YES		None	
5	AccelerometerZ	float	YES		None	
6	GyroscopeX	float	YES		None	
7	GyroscopeY	float	YES		None	
8	GyroscopeZ	float	YES		None	

	COUNT(*)
0	7352

그림 14. SensorReadings Table Results and query_to_df Results

그림 14 는 SensorReadings 테이블에 대한 결과와 query_to_df 를 실행한 결과를 보여준다. SensorReadings 테이블은 COUNT 가 7352 개 이기에 캡쳐하기에 어려움이 있어 columns 만 캡쳐하였다. 또한 query_to_df 를 통해 8 개의 column 에 대하 정보를 확인할 수 있었다.

4-4. Task 2: SQL Query Practice

```
# Q1. How many sensor records exist per user? (Please use 'GROUP BY')
q1_query = """
 SELECT user_id, COUNT(*) AS record_count
 FROM SensorReadings
 GROUP BY user_id;
"""
```

	user_id	record_count
0	1	347
1	3	341
2	5	302
3	6	325
4	7	308
5	8	281
6	11	316
7	14	323
8	15	328
9	16	366
10	17	368
11	19	360
12	21	408
13	22	321
14	23	372
15	25	409
16	26	392
17	27	376
18	28	382
19	29	344
20	30	383

그림 15. Question 1 and Answer

그림 15 는 질문 1 에 대한 대답과 그에 따른 결과를 보여준다. 질문 1 은 각 user 당 보유한 record 개수를 출력하라는 질문이었다. 이에 따라 COUNT 를 통해 개수를 출력하고자 하였고, user_id 로 그룹화 하여 각 user_id 에 해당하는 record_count 를 출력하였다.

```
# Q2. How many sensor records per activity (Please use GROUP BY)
q2_query = """
 SELECT activity_id, COUNT(*) AS activity_record_count
 FROM SensorReadings
 GROUP BY activity_id;
"""
```

	activity_id	activity_record_count
0	1	1226
1	2	1073
2	3	986
3	4	1286
4	5	1374
5	6	1407

그림 16. Question 2 and Answer

그림 16 은 질문 2 에 대한 대답과 그에 따른 결과를 보여준다. 질문 2 는 각 activity_id 당 보유한 record 개수를 출력하라는 질문이었다. 이에 따라 COUNT 를 통해 개수를 출력하고자 하였고, activity_id 에 해당하는 record_count 를 출력하였다.

```
# Q3. How many times did user 1 perform each activity?
q3_query = """
 SELECT activity_id, COUNT(*) AS activity_count
 FROM SensorReadings
 WHERE user_id = 1
 GROUP BY activity_id
 ORDER BY activity_id ASC;
"""
```

	activity_id	activity_count
0	1	95
1	2	53
2	3	49
3	4	47
4	5	53
5	6	50

그림 17. Question 3 and Answer

그림 17 은 질문 3 에 대한 대답과 그에 따른 결과를 보여준다. 질문 3 은 user 1 이 수행한 activity 에 대한 count 를 출력하라는 질문이었다. 이에 따라 activity_id 를 오름차순으로 정렬하고 user_id 에 대 해당하는 activity 개수를 출력하도록 하였고, 그에 따른 결과를 확인할 수 있었다.

```
# Q4. How many times did user 3 perform each activity?
q4_query = """
 SELECT activity_id, COUNT(*) AS activity_count
 FROM SensorReadings
 WHERE user_id = 3
 GROUP BY activity_id
 ORDER BY activity_id ASC;
"""
```

	activity_id	activity_count
0	1	58
1	2	59
2	3	49
3	4	52
4	5	61
5	6	62

그림 18. Question 4 and Answer

그림 18 은 질문 4 에 대한 대답과 그에 따른 결과를 보여준다. 질문 4 는 user 3 이 수행한 activity 에 대한 count 를 출력하라는 질문이었다. 이에 따라 activity_id 를 오름차순으로 정렬하고 user_id 에 대 해당하는 activity 개수를 출력하도록 하였고, 그에 따른 결과를 확인할 수 있었다.

```
# Q5. What is the average AccelerometerX value for each activity?
q5_query = """
 SELECT activity_id, AVG(AccelerometerX) AS avg_accelerometer_x
 FROM SensorReadings
 GROUP BY activity_id
"""
```

	activity_id	avg_accelerometer_x
0	1	0.276260
1	2	0.261930
2	3	0.288169
3	4	0.273449
4	5	0.279294
5	6	0.269191

그림 19. Question 5 and Answer

그림 19 는 질문 5 에 대한 대답과 그에 따른 결과를 보여준다. 질문 5 는 각 activity 당 AccelerometerX 의 평균값을 출력하라는 질문이었다. 이에 따라 activity_id 를 그룹화하여 각 activity_id 당 AccelerometerX 의 평균값을 AVG 연산을 통해 계산한 후 출력하였다.

그림 20. Question 6 and Answer

그림 20 은 질문 6 에 대한 대답과 그에 따른 결과를 보여준다. 질문 6 은 각 activity 당 AccelerometerY 의 평균값을 출력하라는 질문이었다. 이에 따라 activity_id 를 그룹화하여 각 activity_id 당 AccelerometerY 의 평균값을 AVG 연산을 통해 계산한 후 출력하였다.

4-5. Task 3: Activity Prediction

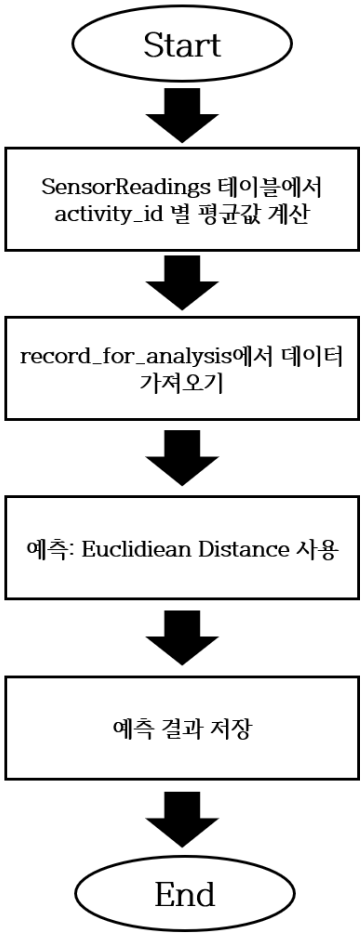


그림 21. Process of Task 3

그림 21 은 Task 3 를 수행하는데 있어 진행되는 프로세스를 보여준다. Task 3 는 새롭게 제공되는 데이터로부터 어떤 Activity 를 수행했는 지 예측하는 것이다. 이에 따라 먼저 SensorReadings

테이블에서 activity_id 별 평균값을 계산하였다. 이후 record_for_analysis 에서 데이터를 가져왔다. 이후 Euclidean Distance 를 사용해 데이터를 예측하였고 이에 대한 예측 결과를 저장하였다.

$$distance = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$$

그림 22. Formular of Euclidean Distance

그림 22 는 Euclidean Distance 에 대한 공식을 보여준다. 본 레포트에서는 해당 방법을 통해 예측을 진행하였다. 왜냐하면 Euclidean Distance 는 벡터 간의 절대적인 거리 차이를 기반으로 유사성을 계산하는 가장 대표적이고 직관적인 방법이기 때문이다. 특히 HAR-UCI 데이터셋은 동일한 단위로 측정된 센서 데이터(tBodyAcc, tBodyGyro)로 구성되어 있어 각 Feature 간의 스케일이 유사하고 정규화가 추가로 필요하지 않으며 차이의 크기 자체가 의미를 가지므로 거리 기반 비교가 효과적이다. 또한 Euclidean Distance 는 계산이 간단하고 연산 속도가 빠르며 벡터 간의 차이가 클수록 명확히 구분되므로 본 레포트처럼 활동 간 평균 센서 벡터를 기반으로 최근접 활동을 판단하는 데 적합하다. 위 공식을 수학적으로 보면 각 차원별로 두 벡터 x 와 y 의 차이를 제곱하여 더한 후 루트를 씌우는 방식으로 거리 값을 도출한다. 이때 차이가 클수록 거리가 커지고, 유사한 벡터일수록 거리가 작아지기 때문에 예측 문제에서 가장 가까운 평균 벡터를 가진 활동을 선택하는 기준으로 사용하기에 적합하다. 다른 방법으로는 Cosine Similarity 나 Manhattan Distance 등을 고려할 수도 있으나 Cosine Similarity 는 방향성에만 집중하여 센서 세기 차이를 반영하지 못하고, Manhattan Distance 는 축 단위 거리합으로 계산되어 상대적으로 예측 경계가 불분명할 수 있다. 따라서 본 레포트에서는 센서 평균 벡터 간의 전체적인 차이를 가장 명확하게 반영할 수 있는 Euclidean Distance 를 예측 기준으로 선택하였다.

```
# STEP 1) SensorReadings 테이블에서 activity_id 별 평균값 구하기
avg_vector_query = """
 SELECT activity_id,
 AVG(AccelerometerX) AS acc_x,
 AVG(AccelerometerY) AS acc_y,
 AVG(AccelerometerZ) AS acc_z,
 AVG(GyroscopeX) AS gyr_x,
 AVG(GyroscopeY) AS gyr_y,
 AVG(GyroscopeZ) AS gyr_z
 FROM SensorReadings
 GROUP BY activity_id;
"""
print(query_to_df(avg_vector_query))
activity_means_df = query_to_df(avg_vector_query)
```

그림 23. Step 1 of Task 3

그림 23 은 Task 3 의 첫 번째 단계인 SensorReadings 테이블에서 평균값을 계산하는 쿼리를 보여준다. 각 Feature 에 대해 평균을 구하기 위해 activity_id 별로 AVG 연산을 수행한 후 activity_means_df 변수에 저장하였다.

```
# STEP 2) record_for_analysis.txt 데이터 가져오기
analysis_file_path = base_path + 'records_for_analysis.txt'

column_names = [
    'tBodyAcc-mean()-X',
    'tBodyAcc-mean()-Y',
    'tBodyAcc-mean()-Z',
    'tBodyGyro-mean()-X',
    'tBodyGyro-mean()-Y',
    'tBodyGyro-mean()-Z'
]

analysis_df = pd.read_csv(analysis_file_path, sep = '₩s+', header=0, names=column_names)

print(analysis_df.head())
```

그림 24. Step 2 of Task 3

그림 24 는 Task 3 의 두 번째 단계인 records_for_analysis 파일로부터 데이터를 가져오는 과정을 보여준다. Column 이름은 과제에서 요구한 대로 정의했으며 이에 따라 pandas 를 활용해 analysis_df 에 저장하였다.

```
# STEP 3) 예측: 각 행에 대해 DB로부터 얻은 활동별 평균 벡터들과 비교
# - Euclidean Distance를 사용해 가장 가까운 활동을 찾음

def euclidean_distance(vec1, vec2):
  return np.sqrt(np.sum((vec1 - vec2) ** 2))

predicted_activities = []

for i in range(len(analysis_df)):
  test_vector = analysis_df.iloc[i].values.astype(float)
  min_dist = float('inf')
  predicted_label = None

  for j in range(len(activity_means_df)):
    activity_vector = activity_means_df.iloc[j, 1:].values.astype(float)
    dist = euclidean_distance(test_vector, activity_vector)

    if dist < min_dist:
      min_dist = dist
      predicted_label = activity_means_df.iloc[j]['activity_id']

  predicted_activities.append(predicted_label)
```

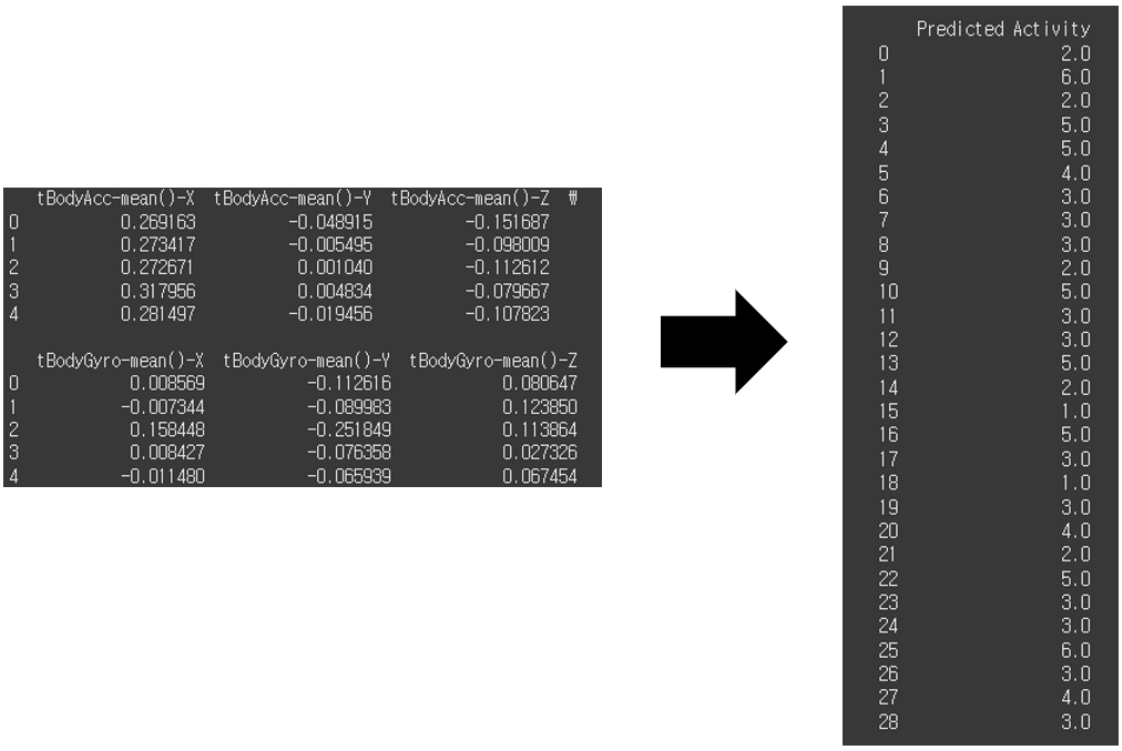
그림 25. Step 3 of Task 3

그림 25 는 Task 3 의 세 번째 단계인 예측 단계이다. 앞서 언급한 Euclidean Distance 방법을 이용하기 위해 별도로 함수를 정의했다. 또한 사전에 읽어온 데이터셋에 대해 기존에 계산한 데이터셋과 Euclidean Distance 함수를 통해 계산된 예측 결과를 predicted_activities 에 append 하였다.

```
task3_result_pd = analysis_df.copy()
task3_result_pd['Predicted Activity'] = predicted_activities
```

그림 26. Step 4 of Task 3

그림 26 은 Task 3 의 네 번째 단계인 예측 결과 저장 단계를 보여준다. 과제에서 언급한대로 task3_result_pd 라는 이름으로 데이터를 저장하였다. 또한 'Predicted Activity' Column 을 두어 예측된 결과를 해당 열에 저장하도록 하였다.



	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	#
0	0.269163	-0.048915	-0.151687	
1	0.273417	-0.005495	-0.098009	
2	0.272671	0.001040	-0.112612	
3	0.317956	0.004834	-0.079667	
4	0.281497	-0.019456	-0.107823	

	tBodyGyro-mean()-X	tBodyGyro-mean()-Y	tBodyGyro-mean()-Z
0	0.008569	-0.112616	0.080647
1	-0.007344	-0.089983	0.123850
2	0.158448	-0.251849	0.113864
3	0.008427	-0.076358	0.027326
4	-0.011480	-0.065939	0.067454

	Predicted Activity
0	2.0
1	6.0
2	2.0
3	5.0
4	5.0
5	4.0
6	3.0
7	3.0
8	3.0
9	2.0
10	5.0
11	3.0
12	3.0
13	5.0
14	2.0
15	1.0
16	5.0
17	3.0
18	1.0
19	3.0
20	4.0
21	2.0
22	5.0
23	3.0
24	3.0
25	6.0
26	3.0
27	4.0
28	3.0

그림 27. Results of task3_result_pd

그림 27 은 task3_result_pd 의 출력 결과를 보여준다. 왼쪽 그림은 각 Feature 들의 평균값을 보여주며 많은 양의 데이터가 출력되었기에 일부만 출력하여 가져왔다. 오른쪽 그림은 예측한 결과의 결과값을 저장한 Feature 를 보여준다. Euclidean Distance 수식을 통해 각 행동에 대해 가장 유사한 행동을 예측한 것을 확인할 수 있으며 각 데이터에 대해 골고루 분포를 보인 것을 확인할 수 있다.

5. Conclusion

본 레포트에서는 HAR-UCI 데이터셋을 기반으로 데이터베이스를 구축하고 이에 대해 SQL Query 를 이용해 데이터를 분석하였다. 그리고 새롭게 들어온 데이터셋에 대해 기존 데이터베이스의 데이터셋을 바탕으로 행동을 예측하는 과제를 수행하였다. 이를 성공적으로 수행하기 위해 2 장에서 본 과제가 요구하는 요구사항들을 살펴보았고, 3 장에서 이를 구현하기 위한 관련 개념들을 살펴보았다. 이러한 요구사항과 개념들을 바탕으로 4장에서 구현한 코드와 그에 따른 결과를 확인할 수 있었고, 최종적으로 새로운 데이터셋에 대한 행동을 예측할 수 있었다. 결론적으로 본 과제를 통해 단순한 데이터 저장소로서의 데이터베이스를 넘어서 복잡한 데이터셋의 구성과 예측 문제 해결의 기반으로 데이터베이스를 활용할 수 있는 가능성을 확인할 수 있었다. 또한 SQL 을 단순한 질의 언어로 활용하는 것에 그치지 않고, 데이터 분석과 분석된 결과를 기존 예측 수식에 활용할 수 있는 핵심 도구로서 접목시킬 수 있는 실무적 역량을 배양할 수 있었다.