

중앙대학교 소프트웨어학부

# 혈액 보관소 DB

데이터베이스 시스템

20181807 김찬경

2023-6-2

# 프로젝트 최종 보고서

## 프로젝트 요약

보고 날짜	프로젝트 이름	작성자
2023 년 6 월 2 일	혈액 보관소 DB 시스템	김찬경(20181807)

## 개발 환경

운영체제: WINDOWS 11

개발 언어: JAVA

IDE: INTELLIJ IDEA

DB: 관계형 데이터베이스 사용(그 중 MySQL)

## 프로젝트 설명

이 프로젝트의 목표는 혈액 보관소들의 데이터를 모아놓은 혈액 보관소 DB 시스템을 구축하는 것이다. 이를 위해 4 가지의 기능(테이블 생성, 레코드 삽입, bitmap index 생성, bitmap index 를 이용한 질의 처리)을 구현하였고 번호를 입력받아 해당 기능들을 수행하도록 구현하였다.

1 번은 혈액 보관소 테이블을 만드는 기능 / 2 번은 해당 테이블에 1000 개의 레코드를 한번에 삽입하는 기능  
3 번은 비트맵 인덱스를 만드는 기능 / 4 번은 비트맵 인덱스를 이용하여 multiple key query 를 하는 기능  
5 번은 비트맵 인덱스를 이용하여 count query 를 하는 기능 / 6 번은 끝내는 기능으로 구성하였다.  
해당 기능들의 상세한 설명은 뒤에서 이어서 하겠다.

## 설계 설명(설계, 구현 상에서 고려한 내용)

### 1. 설정 및 가정한 내용

테이블은 하나로 설정하고 테이블 하나에 대량의 레코드를 삽입한다고 결정하였다.

테이블 스키마는 다음과 같이 설정했다.

<u>serial_num</u>	<u>center_id</u>	<u>donor_id</u>	<u>donor_name</u>	<u>donor_gender</u>	<u>blood_type</u>	<u>donation_date</u>

serial\_num 은 일련의 번호로 B+ 트리 인덱싱을 위한 컬럼으로 일련번호 부여와 인덱싱을 위해 SERIAL 설정을 하였다. 또한 MySQL 에서는 primary key 에 대하여 기본적으로 B+ 트리 인덱싱을 적용하므로 PRIMARY KEY 설정을 하였다.

`serial_num SERIAL PRIMARY KEY`

serial\_num 을 제외한 모든 컬럼은 랜덤으로 설정하였다. 대량의 레코드 삽입 시 편의성을 위해 center\_id 는 1 부터 100 까지로 가정하였고, donor\_id 는 1 부터 1000000 까지로 범위를 크게 잡았고, donor\_name 은 알파벳에서 랜덤으로 골라 길이가 3 부터 7 까지의 랜덤 스트링으로 구성하였고 donor\_gender 는 M,F 중 랜덤으로 고르도록 설정하였고 BITMAP INDEX 로 설정하였다. blood\_type 은 혈액형으로 +, -도 구분해야하므로 총 8 개의 타입으로 { "A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-" }로 설정하였다. 이 안에서 랜덤으로 설정하였으며 또한 앞서 말한 기능에서 비트맵을 이용한 multiple key query 기능이 필요하여 성별과 마찬가지로 이것도 비트맵 인덱스로 설정하였다. 마지막으로 donation\_date 는 2013-05-07 일부터 2023-05-07 안에서 랜덤으로 골라 설정하도록 하였다.

그리하여 다음과 같이 설정하였다.

```
int centerId=rand.nextInt(100)+1;
int donorId=rand.nextInt(1000000)+1;
String donorName="";
for(int j=0;j<rand.nextInt(5)+3;j++){
    donorName+=(char)('a'+rand.nextInt(26));
}
String donorGender=genders[rand.nextInt(2)];
String bloodType=bloodTypes[rand.nextInt(8)];

long startDate = Date.valueOf("2013-05-07").getTime();
long endDate = Date.valueOf("2023-05-07").getTime();
long randomTime = startDate + (long) (rand.nextDouble() * (endDate - startDate));
Date donationDate= new Date(randomTime);
```

---

## 2. 구현 상 고려한 이슈들

---

### I. 현실 세계 값과의 차이

현실 세계의 값은 굉장히 다양하기에 구현할 시 괴리가 클 수 밖에 없었고 최대한 현실 세계를 반영하려 값의 범위를 위처럼 설정하였고 다양성을 위해 랜덤하게 레코드를 삽입하도록 하였다.

### II. 기부자 이름의 다양성 부족

랜덤하게 값을 삽입 시 정해진 리스트에서 뽑아 기부자 이름을 정하려 했으나 대량의 데이터를 삽입한다는 가정하에 다양성이 떨어질 것을 예상하여 위의 코드와 같이 여러 문자를 랜덤으로 정해 연결하는 방식으로 수정하여 다양성을 늘렸다.

### III. 기부자가 같은데 기부자 이름, 성별, 혈액형이 다르게 삽입되는 경우

위에서 언급한 것처럼 모든 레코드의 내용이 랜덤으로 설정하기에 같은 기부자 ID 를 가지고 있음에도 불구하고 이름과 성별 그리고 혈액형이 다르게 설정되는 문제가 발생하였다. 이 문제를 해결하기 위해 추가적으로 함수를 만들어 기부자 ID 가 같으면 기부자 이름, 성별, 혈액형을 같게 만들도록 설정하였다.

함수:

```
public static void updateDuplicateDonors(Connection conn) throws
SQLException {
    String sql="SELECT donor_id, donor_name, donor_gender,
blood_type FROM blood_donations GROUP BY donor_id HAVING COUNT(*)
> 1";
    PreparedStatement pstmt=conn.prepareStatement(sql);
    ResultSet rs=pstmt.executeQuery();

    while(rs.next()){
        int donorId=rs.getInt("donor_id");
        String donorName=rs.getString("donor_name");
        String donorGender=rs.getString("donor_gender");
        String bloodType=rs.getString("blood_type");
        sql="UPDATE blood_donations SET donor_name = ?,
donor_gender = ?, blood_type = ? WHERE donor_id = ?";
        pstmt=conn.prepareStatement(sql);
        pstmt.setString(1, donorName);
        pstmt.setString(2, donorGender);
        pstmt.setString(3, bloodType);
        pstmt.setInt(4, donorId);
    }
}
```

```

pstmt.executeUpdate();

}
System.out.println("Duplicate donors updated successfully!");
}

```

사용자가 UI 에서 레코드 삽입 기능을 실행하면 이 함수도 같이 호출되어 이러한 문제가 발생하지 않도록 바꾸었다.

이전:

```

mysql> select donor_id, donor_name, donor_gender, blood_type from blood_donations group by donor_id having count(*)>1;
+-----+-----+-----+-----+
| donor_id | donor_name | donor_gender | blood_type |
+-----+-----+-----+-----+
| 43087 | qrdv | F | A- |
| 212610 | cnk | F | B+ |
| 643653 | drg | M | B- |
| 597651 | gzs | M | A- |
| 668181 | trqae | F | A- |
| 890079 | vtzt | M | B- |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> select * from blood_donations where donor_id=43087;
+-----+-----+-----+-----+-----+-----+-----+
| serial_num | center_id | donor_id | donor_name | donor_gender | blood_type | donation_date |
+-----+-----+-----+-----+-----+-----+-----+
| 118 | 66 | 43087 | qrdv | F | A- | 2022-09-21 |
| 481 | 12 | 43087 | sbgole | M | AB+ | 2016-08-29 |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

함수 추가하기 전 다음과 같이 같은 기부자 id 를 가지고 있음에도 불구하고 이름과 성별, 혈액형이 다른 것을 볼 수 있다.

이후:

```

mysql> select * from blood_donations where donor_id=43087;
+-----+-----+-----+-----+-----+-----+-----+
| serial_num | center_id | donor_id | donor_name | donor_gender | blood_type | donation_date |
+-----+-----+-----+-----+-----+-----+-----+
| 118 | 66 | 43087 | qrdv | F | A- | 2022-09-21 |
| 481 | 12 | 43087 | qrdv | F | A- | 2016-08-29 |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

```

함수 시행 후 통일된 것을 볼 수 있다.

#### IV. BITMAP INDEX 설정 시 어떤 컬럼에 대해 BITMAP INDEX 를 만들 것인지에 대한 고민

donor\_gender 를 bitmap index 로 설정하기로 정하였다. 그 이유는 성별이 적은 수의 고유한 값을 갖기 때문에 비트맵 인덱스의 크기가 작아지고 검색 속도가 매우 빨라질 것이라 판단했기 때문이다. 또한 추가적인 비트맵 인덱스를 사용한 multiple key query 를 위해 한정적인 고유한 값을 가진 혈액형도 비트맵 인덱스로 만들었다.

- V. BITMAP INDEX 기능을 지원하지 않는데 어떻게 BITMAP INDEX 를 표현할지에 대한 고민
- MySQL 에서 공식적으로 B+ 트리 인덱스에 대해 지원하지만 BITMAP INDEX 기능을 지원하지 않기에 어떤 식으로 구현을 하고 표현을 해야 할 지에 대한 고민을 하였다. 수업시간에 배운 것을 토대로 나타내기엔 테이블이 좋다고 판단하여 성별에 관한 비트맵 인덱스 테이블, 혈액형에 관한 비트맵 인덱스 테이블을 만들어 질의처리에 사용하였다.

## 구현 설명

### 기능 선택 부분

```
Scanner sc=new Scanner(System.in);
int choice;
do{
    System.out.println("1. Create table");
    System.out.println("2. Insert record");
    System.out.println("3. Making Bitmap Index");
    System.out.println("4. Multiple Key Query");
    System.out.println("5. Count Query");
    System.out.println("6. Exit");
    System.out.print("Enter your choice: ");
    choice = sc.nextInt();
}
```

번호를 입력해 기능을 선택하는 부분으로 번호 입력에 앞서 해당 번호에 어떤 기능이 있는지 텍스트 기반 UI 로 한눈에 보여준다. 기능에는 테이블 생성 기능, 레코드 삽입 기능, 비트맵 인덱스 생성 기능, multiple key 질의 처리 기능, count 질의처리 기능, 마지막으로 프로그램을 종료하는 기능이 있다.

```

switch(choice){
    case 1:
        createTable(conn);
        break;
    case 2:
        insertRecord(conn);
        updateDuplicateDonors(conn);
        break;
    case 3:
        makeBitmapgenderIndex(conn);
        makeBitmapbloodtypeIndex(conn);
        break;
    case 4:
        multiplekeyQuery(conn);
        break;
    case 5:
        countQuery(conn);
        break;
    case 6:
        System.out.println("Exiting...");
        break;
    default:
        System.out.println("Invalid choice, please try again.");
}
}while(choice!=6);
conn.close();

```

Switch 문으로 구현하였고 잘못된 번호를 입력할 경우 다시 입력하도록 만들었으며 프로그램 종료인 6 번을 입력하기 전까지 계속해서 원하는 기능을 입력할 수 있도록 하였다. 또한 앞서 말한 기능 중에서 레코드 삽입 기능인 2 번에서 대량의 레코드 삽입 시 랜덤으로 모든 값을 결정해 삽입하므로, donor\_id 가 같은, 즉 같은 사람임에도 불구하고 이름, 성별, 혈액형이 다르게 삽입되는 문제가 발생하였다. 이를 해결하기 위해 레코드를 삽입하는 기능인 insertRecord(conn) 이후, updateDuplicateDonors(conn) 기능을 통해 기부자 id 가 같은 경우, 이름, 성별, 혈액형을 통일시키는 함수를 추가하였다.

비트맵 인덱스 생성 기능인 3 번의 경우, 비트맵 인덱스 생성 시 성별에 관한 비트맵 인덱스, 혈액형에 관한 비트맵 인덱스를 둘 다 만들도록 구현하였다.

---

## 테이블 생성 기능

---

```
public static void createTable(Connection conn) throws SQLException {
    String sql="CREATE TABLE blood_donations (serial_num SERIAL PRIMARY KEY, center_id INT, donor_id INT, donor_name VARCHAR(255), donor_gender CHAR(1), blood_type VARCHAR(5), donation_date DATE)";
    Statement stmt=conn.createStatement();
    stmt.executeUpdate(sql);
    System.out.println("Table created successfully!");
}
```

serial_num	center_id	donor_id	donor_name	donor_gender	blood_type	donation_date

이러한 테이블을 만들고 실행창에 테이블이 성공적으로 생성되었다는 영어 문장을 출력하는 기능이다.

---

## 레코드 삽입 기능

---

```
public static void insertRecord(Connection conn) throws SQLException {
    String[] bloodTypes = {"A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"};
    String[] genders = {"M", "F"};
    Random rand = new Random();

    for(int i=0;i<1000;i++) {
        int centerId=rand.nextInt( bound: 100)+1;
        int donorId=rand.nextInt( bound: 1000000)+1;
        String donorName="";
        for(int j=0;j<rand.nextInt( bound: 5)+3;j++){
            donorName+=(char)('a'+rand.nextInt( bound: 26));
        }
        String donorGender=genders[rand.nextInt( bound: 2)];
        String bloodType=bloodTypes[rand.nextInt( bound: 8)];
        long startDate = Date.valueOf( s: "2013-05-07").getTime();
        long endDate = Date.valueOf( s: "2023-05-07").getTime();
        long randomTime = startDate + (long) (rand.nextDouble() * (endDate - startDate));
        Date donationDate= new Date(randomTime);

        String sql = "INSERT INTO blood_donations (center_id, donor_id, donor_name, donor_gender, blood_type, donation_date) VALUES (?, ?, ?, ?, ?, ?)";
        PreparedStatement pstmt=conn.prepareStatement(sql);
        pstmt.setInt( parameterIndex: 1, centerId);
        pstmt.setInt( parameterIndex: 2,donorId);
        pstmt.setString( parameterIndex: 3, donorName);
        pstmt.setString( parameterIndex: 4, donorGender);
        pstmt.setString( parameterIndex: 5,bloodType);
        pstmt.setDate( parameterIndex: 6, donationDate);
        pstmt.executeUpdate();
    }
    System.out.println("Record Inserted successfully!");
}
```

자동으로 설정해주는 serial num 을 제외하고 나머지 attribute 에 대해 랜덤으로 삽입하도록 설정했으며 기부 센터에 대해서는 1 부터 100 까지 범위 중 랜덤으로, 기부자에 대해선 1 부터 1000000 까지에서 랜덤으로, 기부자 이름은 3 부터 7 까지의 길이를 가지도록 설정하였으며 성별에 대해선 M,F 중 랜덤으로, 혈액형에 대해선 RH-, RH+까지



고려하여 총 8 개의 혈액형 중 랜덤으로, 기부 날짜는 20130507 부터 20230507 까지 10 년까지의 날짜 중 랜덤으로 삽입하도록 설정하였다. 한번 기능 시행 시 1000 개의 레코드를 삽입하도록 설정하였다.

```
public static void updateDuplicateDonors(Connection conn) throws SQLException {
    String sql="SELECT donor_id, donor_name, donor_gender, blood_type FROM blood_donations GROUP BY donor_id HAVING COUNT(*) > 1";
    PreparedStatement pstmt=conn.prepareStatement(sql);
    ResultSet rs=pstmt.executeQuery();

    while(rs.next()){
        int donorId=rs.getInt( columnLabel: "donor_id");
        String donorName=rs.getString( columnLabel: "donor_name");
        String donorGender=rs.getString( columnLabel: "donor_gender");
        String bloodType=rs.getString( columnLabel: "blood_type");
        sql="UPDATE blood_donations SET donor_name = ?, donor_gender = ?, blood_type = ? WHERE donor_id = ?";
        pstmt=conn.prepareStatement(sql);
        pstmt.setString( parameterIndex: 1, donorName);
        pstmt.setString( parameterIndex: 2, donorGender);
        pstmt.setString( parameterIndex: 3, bloodType);
        pstmt.setInt( parameterIndex: 4, donorId);
        pstmt.executeUpdate();
    }
    System.out.println("Duplicate donors updated successfully!");
}
```

앞선 레코드 대량 삽입 시 아무리 랜덤으로 설정해도 donor\_id, 즉 기부자가 같은 경우가 발생하였으며 기부자가 같음에도 불구하고 랜덤으로 삽입하다 보니 기부자 id 가 같은데 이름, 성별, 혈액형이 다르게 설정되는 문제가 발생하였다. 이를 해결하기 위해 추가적인 updateDuplicateDonors(Connection conn) 함수를 통해 기부자 id 가 같은 경우 이름, 성별, 혈액형을 통일시키도록 설정하였다. 추가적인 기능 선택 필요없이 레코드 삽입 기능인 2 번 입력 시 레코드 삽입을 한 후 이어서 이 함수를 실행하도록 설정하였다.

1. 성별에 관한 비트맵 인덱스 생성 함수

```
public static void makeBitmapgenderIndex(Connection conn) throws SQLException {
    String indexTable = "donor_gender_index";
    String createTableSQL = "CREATE TABLE " + indexTable + " (gender CHAR(1), bitmap LONGTEXT)";
    Statement stmt = conn.createStatement();
    stmt.executeUpdate(createTableSQL);
    String[] genders = {"M", "F"};

    for (String gender : genders) {
        String selectRecordsSQL = "SELECT serial_num FROM blood_donations WHERE donor_gender = ?";
        PreparedStatement recordsPstmt = conn.prepareStatement(selectRecordsSQL);
        recordsPstmt.setString(1, gender);
        ResultSet recordsRs = recordsPstmt.executeQuery();

        int number = getRecordCount(conn);
        StringBuilder bitmap = new StringBuilder();
        for (int i = 0; i < number; i++) {
            bitmap.append("0");
        }

        while (recordsRs.next()) {
            int serialNum = recordsRs.getInt("serial_num") - 1;
            bitmap.setCharAt(serialNum, '1');
        }

        String insertIndexSQL = "INSERT INTO " + indexTable + " (gender, bitmap) VALUES (?, ?)";
        PreparedStatement insertPstmt = conn.prepareStatement(insertIndexSQL);
        insertPstmt.setString(1, gender);
        insertPstmt.setString(2, bitmap.toString());
        insertPstmt.executeUpdate();
    }
    System.out.println("Gender Bitmap index created successfully!");
}
```

성별에 관한 비트맵 인덱스를 만드는 함수로 3 번 기능이다. MySQL에서는 BitmapIndex를 직접적으로 지원하지 않기에 테이블 형태로 만들기로 하였고, 젠더에 따라 비트 스트링으로 구성된 테이블을 만드는 함수이다. 여기에 비트 스트링 길이는 레코드 수에 따라 달라지므로 뒤에서 설명할 getRecordCount(conn) 함수를 통해 record 수를 받아와 그 길이만큼 길게 만든다.

## 2. 혈액형에 관한 비트맵 인덱스 생성 함수

```
public static void makeBitmapbloodtypeIndex(Connection conn) throws SQLException{
    String indexTable = "blood_type_index";
    String createTableSQL = "CREATE TABLE " + indexTable + " (blood_type VARCHAR(3), bitmap LONGTEXT)";
    Statement stmt = conn.createStatement();
    stmt.executeUpdate(createTableSQL);
    String[] blood_types = {"A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"};

    for (String blood_type : blood_types) {
        String selectRecordsSQL = "SELECT serial_num FROM blood_donations WHERE blood_type = ?";
        PreparedStatement recordsPstmt = conn.prepareStatement(selectRecordsSQL);
        recordsPstmt.setString(1, blood_type);
        ResultSet recordsRs = recordsPstmt.executeQuery();

        int number=getRecordCount(conn);
        StringBuilder bitmap = new StringBuilder();
        for (int i = 0; i < number; i++) {
            bitmap.append("0");
        }
        while (recordsRs.next()) {
            int serialNum = recordsRs.getInt("serial_num") - 1;
            bitmap.setCharAt(serialNum, '1');
        }

        String insertIndexSQL = "INSERT INTO " + indexTable + " (blood_type, bitmap) VALUES (?, ?)";
        PreparedStatement insertPstmt = conn.prepareStatement(insertIndexSQL);
        insertPstmt.setString(1, blood_type);
        insertPstmt.setString(2, bitmap.toString());
        insertPstmt.executeUpdate();
    }

    System.out.println("Blood type Bitmap index created successfully!");
}
```

위의 성별에 관한 비트맵 인덱스 생성과 마찬가지로 이 함수는 혈액형에 관한 비트맵 인덱스 생성 함수이다. 마찬가지로 테이블 형태로 표현하였고 각 혈액형, 총 8 개에 관한 비트스트링을 각각 만들어 테이블에 저장한다.

### 3. 그 외 비트맵 인덱스 생성 시 필요한 추가적인 함수

```
public static int getRecordCount(Connection conn) throws SQLException{
    String countSQL = "SELECT COUNT(*) AS count FROM blood_donations";
    Statement countStmt = conn.createStatement();
    ResultSet countRs = countStmt.executeQuery(countSQL);
    countRs.next();
    return countRs.getInt( columnLabel: "count");
}
```

앞서 말한 getRecordCount(conn) 함수로 각 비트 스트링의 길이는 레코드 수만큼 길어야하므로 레코드 수를 알아내기 위해 기존에 만든 blood\_donations 테이블에서 record 수를 가져와 반환하는 함수이다.

---

### Multiple key query 기능

---

```
public static void multiplekeyQuery(Connection conn) throws SQLException {
    Scanner sc = new Scanner(System.in);
    String attribute = null;
    System.out.printf("Enter gender (M/F) : ");
    String gender = sc.nextLine();
    System.out.printf("Enter blood type(A+, A-, B+, B-, O+, O-, AB+, AB-) : ");
    String bloodType = sc.nextLine();
    System.out.printf("Enter bit operation(AND, OR, NOT) : ");
    String operation = sc.nextLine();
    if (operation.equals("NOT")) {
        System.out.printf("Enter attribute(Gender, BloodType) : ");
        attribute = sc.nextLine();
    }
}
```

처음에 성별, 혈액형, 그리고 어떤 연산을 할 건지에 대한 입력을 받는다. 연산을 입력받는 이유는 일반 질의처리가 아닌, 비트맵 인덱스를 사용하기에, 즉 비트 스트링을 사용하여 더 효과적이고 빠르게 원하는 질의처리를 하는 기능이기때문에 연산을 추가적으로 입력받기로 해서 AND, OR, NOT 중에서 3 개를 고르면 된다. 그리고 만약 NOT 을 입력받으면, 어떤 attribute 에 대해 NOT 연산을 할 건지 추가적으로 입력해야한다. 예를 들어 NOT 까지 입력하고 Gender 라고 하면 앞서 입력받은 gender 에 대해 반대되는 레코드들을 출력하는 형태이다.

```

String genderindexQuery = "SELECT bitmap FROM donor_gender_index WHERE gender=?";
PreparedStatement genderpstmt = conn.prepareStatement(genderindexQuery);
genderpstmt.setString( parameterIndex: 1, gender);
ResultSet genderRs = genderpstmt.executeQuery();

String bloodtypeindexQuery = "SELECT bitmap FROM blood_type_index WHERE blood_type=?";
PreparedStatement bloodtypepstmt = conn.prepareStatement(bloodtypeindexQuery);
bloodtypepstmt.setString( parameterIndex: 1, bloodType);
ResultSet bloodtypeRs = bloodtypepstmt.executeQuery();

if (genderRs.next() && bloodtypeRs.next()) {
    String genderBitmap = genderRs.getString( columnLabel: "bitmap");
    String bloodTypeBitmap = bloodtypeRs.getString( columnLabel: "bitmap");
    StringBuilder resultBitmap = new StringBuilder();

```

Multiple query 를 위해 두 가지 비트맵 인덱스를 다 가져온다. 앞서 입력한 젠더에 해당되는 비트 스트링, 입력한 혈액형에 해당되는 비트 스트링을 가져오는 부분이다.

```

if (operation.equals("AND")) {
    for (int i = 0; i < genderBitmap.length(); i++) {
        if (genderBitmap.charAt(i) == '1' && bloodTypeBitmap.charAt(i) == '1') {
            resultBitmap.append("1");
        } else {
            resultBitmap.append("0");
        }
    }
} else if (operation.equals("OR")) {
    for (int i = 0; i < genderBitmap.length(); i++) {
        if (genderBitmap.charAt(i) == '1' || bloodTypeBitmap.charAt(i) == '1') {
            resultBitmap.append("1");
        } else {
            resultBitmap.append("0");
        }
    }
} else if (operation.equals("NOT")) {
    if (attribute.equals("Gender")) {
        for (int i = 0; i < genderBitmap.length(); i++) {
            if (genderBitmap.charAt(i) == '0') {
                resultBitmap.append("1");
            } else {
                resultBitmap.append("0");
            }
        }
    } else if (attribute.equals("BloodType")) {
        for (int i = 0; i < bloodTypeBitmap.length(); i++) {
            if (bloodTypeBitmap.charAt(i) == '0') {
                resultBitmap.append("1");
            } else {
                resultBitmap.append("0");
            }
        }
    }
}

```

여기부터는 연산을 수행하는 부분인데, AND, OR, NOT 연산에 따라 각각 비트 스트링의 각 비트 연산을 수행해가며 resultBitmap 을 만든다.

```

    } else {
        System.out.println("다시 제대로 입력해주세요");
        return;
    }
    System.out.println("Result Bitmap : " + resultBitmap.toString());

    for (int i = 0; i < resultBitmap.length(); i++) {
        if (resultBitmap.charAt(i) == '1') {
            printRecord(conn, serialNum: i + 1);
        }
    }
}
}

```

만약 연산의 종류가 제대로 입력되지 않으면 다시 제대로 입력해달라는 문구를 출력한다. 제대로 결과물이 나와 resultBitmap 이 구성된다면 해당 resultBitmap 을 실행창에서 출력하고 해당되는 record, 즉 비트스트링에서 각 비트가 1 인 경우를 printRecord() 함수를 통해 다 출력하는 형태로 구현하였다.

```

public static void printRecord(Connection conn, int serialNum) throws SQLException{
    String selectRecord="SELECT donor_name, donation_date FROM blood_donations WHERE serial_num=?";
    PreparedStatement pstmt=conn.prepareStatement(selectRecord);
    pstmt.setInt( parameterIndex: 1, serialNum);
    ResultSet rs=pstmt.executeQuery();

    if(rs.next()){
        String donorName=rs.getString( columnLabel: "donor_name");
        String donationDate=rs.getString( columnLabel: "donation_date");
        System.out.println("기부자 : "+donorName+ ", 기부 일자 : " + donationDate);
    }
}
}

```

앞서 말한 printRecord 로 해당되는 record 의 serial number 를 이용해 기부자 이름과 기부 일자를 출력하는 함수이다.

```
public static void countQuery(Connection conn) throws SQLException{
    Scanner sc=new Scanner(System.in);
    String gender=null;
    String bloodtype=null;
    System.out.print("Enter attribute (Gender/BloodType) : ");
    String attribute=sc.nextLine();
    if(attribute.equals("Gender")){
        System.out.print("Enter gender (M/F) : ");
        gender=sc.nextLine();
        String temp_query="SELECT bitmap FROM donor_gender_index WHERE gender = ?";
        PreparedStatement temp_pstmt=conn.prepareStatement(temp_query);
        temp_pstmt.setString(1, gender);
        ResultSet temp_rs=temp_pstmt.executeQuery();

        if(temp_rs.next()){
            String bitmap=temp_rs.getString("bitmap");
            int count=0;
            for(int i=0;i<bitmap.length();i++){
                if(bitmap.charAt(i)=='1'){
                    count++;
                }
            }
            System.out.println("성별이 " +gender+" 인 사람의 수 : "+count);
        }else{
            System.out.println("해당 성별에 관한 사람이 없습니다");
        }
    }
}
```

처음에 어떤 attribute 에 대해 countQuery 를 할 건지 입력한다. 이 부분은 Gender, 성별을 입력했을 때 실행하는 부분으로 M, F 중 고른 후 해당되는 비트맵 인덱스(비트 스트링)을 가져온 후 거기서 1 인 비트의 수를 세서 해당 사람의 수를 출력한다. 만약 해당되는 사람이 없는 경우 해당 사람이 없다고 출력한다.



```

}else if(attribute.equals("BloodType")){
    System.out.print("Enter blood type(A+, A-, B+, B-, O+, O-, AB+, AB-) : ");
    bloodtype=sc.nextLine();
    String temp_query="SELECT bitmap FROM blood_type_index WHERE blood_type= ?";
    PreparedStatement temp_pstmt=conn.prepareStatement(temp_query);
    temp_pstmt.setString( parameterIndex: 1, bloodtype);
    ResultSet temp_rs=temp_pstmt.executeQuery();

    if(temp_rs.next()){
        String bitmap=temp_rs.getString( columnLabel: "bitmap");
        int count=0;
        for(int i=0;i<bitmap.length();i++){
            if(bitmap.charAt(i)=='1'){
                count++;
            }
        }
        System.out.println("혈액형이 " +bloodtype+" 인 사람의 수 : "+count);
    }else{
        System.out.println("해당 혈액형인 사람이 없습니다");
    }
}
}else{
    System.out.print("다시 제대로 입력해주세요!");
}
}

```

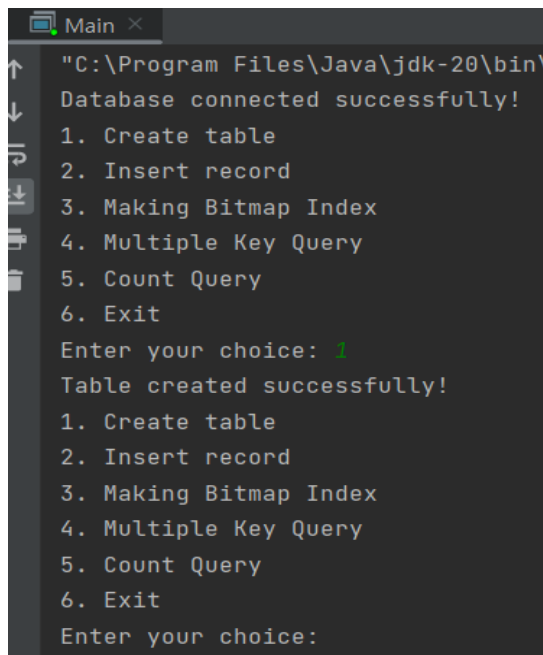
이 부분은 혈액형을 선택했을 때 해당되는 부분으로 위와 마찬가지로 선택한 혈액형에 해당되는 비트맵 인덱스(비트 스트링)을 가져와 1 인 비트의 수를 세서 출력한다. 만약 해당되는 사람이 없으면 해당 사람이 없다고 출력한다. 만약 위에서 attribute 를 잘못 입력하면 다시 제대로 입력해달라고 출력한다.

## 테이블 생성 기능

테이블 생성하기 이전

```
mysql> show tables;  
Empty set (0.01 sec)
```

1 번 기능으로 테이블 생성하면 테이블이 성공적으로 만들어졌다고 출력한다.



```
Main x  
"C:\Program Files\Java\jdk-20\bin\  
Database connected successfully!  
1. Create table  
2. Insert record  
3. Making Bitmap Index  
4. Multiple Key Query  
5. Count Query  
6. Exit  
Enter your choice: 1  
Table created successfully!  
1. Create table  
2. Insert record  
3. Making Bitmap Index  
4. Multiple Key Query  
5. Count Query  
6. Exit  
Enter your choice:
```

결과를 좀 더 정확하게 보기 위해 mysql 을 통해 결과를 확인해보면

```
mysql> show tables;  
Empty set (0.01 sec)  
  
mysql> show tables;  
+-----+  
| Tables_in_blood_storage |  
+-----+  
| blood_donations         |  
+-----+  
1 row in set (0.01 sec)  
  
mysql>
```

이렇게 새롭게 blood\_donations 이라는 테이블이 생긴 것을 확인할 수 있다.

---

## 테이블 레코드 삽입 기능

---

2 번 기능을 통해 레코드를 삽입하기 전 레코드가 없음을 확인할 수 있다.

```
mysql> select * from blood_donations;  
Empty set (0.01 sec)
```

2 를 입력하여 2 번 기능으로 레코드를 삽입하면 레코드가 성공적으로 삽입되었고 중복되는 기부자들에 대한 정보를 업데이트했다고 출력한다.

```
1. Create table  
2. Insert record  
3. Making Bitmap Index  
4. Multiple Key Query  
5. Count Query  
6. Exit  
Enter your choice: 2  
Record Inserted successfully!  
Duplicate donors updated successfully!
```

Mysql 로 제대로 삽입되었는지 확인해보면 다음과 같이 성공적으로 삽입된다.

975	73	494368	ekfn	F	O+	2014-02-11
976	21	76729	mmbka	M	B+	2020-05-13
977	17	856655	kwbax	M	A-	2017-09-20
978	35	885306	dkkany	F	O+	2021-08-12
979	21	263052	tymrh	F	A+	2013-09-12
980	63	369374	omcjs	F	B+	2015-09-25
981	92	490098	hpwf	F	B+	2022-11-03
982	19	820599	vgifpn	F	O-	2018-02-17
983	32	564675	nwqz	F	B-	2016-08-30
984	33	583228	robfu	M	AB+	2020-11-05
985	8	387368	zcwdp	F	B+	2015-06-29
986	39	225243	muwae	F	B-	2017-10-19
987	49	923564	jfuq	M	A-	2022-10-23
988	29	782496	ahhj	M	B+	2019-11-24
989	7	807464	qrb	M	A+	2022-08-10
990	17	725979	xzbj	M	A+	2014-06-16
991	79	929833	twjjtf	M	B-	2017-09-19
992	69	142710	yuz	M	O+	2016-11-30
993	56	811672	lohedx	M	O+	2015-09-12
994	78	590708	rorq	F	O-	2019-04-13
995	90	641219	chsfwx	M	O-	2017-10-21
996	37	708269	owsqy	M	AB+	2018-08-09
997	25	702607	mindvm	F	B-	2014-12-24
998	15	118207	oifim	M	AB-	2022-06-10
999	7	965783	hfuz	M	A-	2015-11-18
1000	77	293688	mdfq	F	A-	2014-02-17

1000 rows in set (0.01 sec)

## 비트맵 인덱스 생성 기능

	3975	65	850910	yrbpo	M	AB+	2014-11-03
	3976	52	327979	pxqbdsv	F	O+	2017-07-01
	3977	15	701632	xcg	F	B+	2017-11-04
	3978	87	330764	hyzc	M	A+	2020-03-01
	3979	26	51076	pilpb	M	A-	2018-05-01
	3980	20	834035	intn	M	O+	2014-01-28
	3981	89	208501	sansx	F	A+	2019-03-04
	3982	50	147265	xqf	M	B-	2022-03-30
	3983	5	841563	ppmsv	M	AB+	2018-04-22
	3984	85	581169	xbi	M	A+	2015-12-05
	3985	72	813266	rwwvo	M	O+	2017-03-03
	3986	42	12325	dmtol	F	B-	2019-09-24
	3987	60	671937	xgp	M	AB+	2018-08-23
	3988	74	242763	cfct	F	AB-	2022-06-16
	3989	85	800050	iihv	F	AB+	2016-05-05
	3990	4	185673	lqmd	F	AB-	2019-09-23
	3991	48	223969	jcey	F	AB+	2019-05-29
	3992	37	529382	ouwd	F	A-	2018-05-19
	3993	49	176760	xxyzi	M	O-	2021-03-02
	3994	20	339836	nkhem	F	B+	2013-10-31
	3995	41	870521	dsawh	F	AB+	2022-03-02
	3996	61	619426	bcpmg	M	AB-	2018-12-13
	3997	28	685374	iwdp	F	A+	2019-10-12
	3998	65	6626	nhlviwz	F	A-	2022-10-21
	3999	84	471967	zrjv	F	O-	2015-03-08
	4000	95	95833	fij	F	A-	2015-11-12

+-----+-----+-----+-----+-----+-----+-----+

4000 rows in set (0.01 sec)

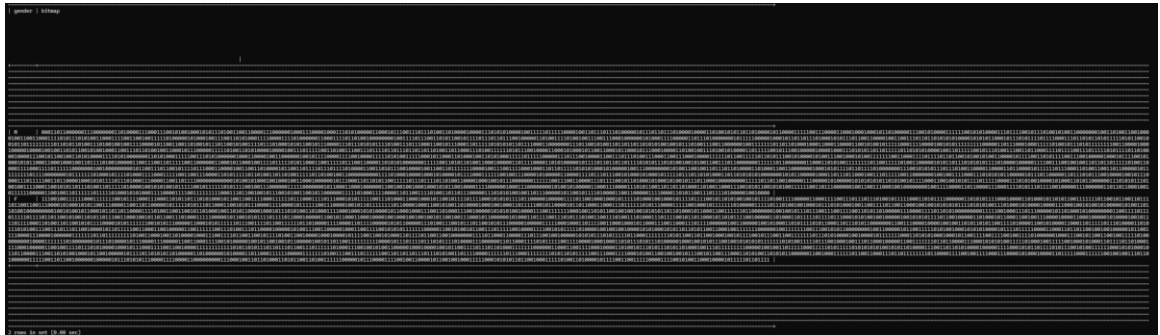
mysql>

```
1. Create table
2. Insert record
3. Making Bitmap Index
4. Multiple Key Query
5. Count Query
6. Exit
Enter your choice: 3
Gender Bitmap index created successfully!
Blood type Bitmap index created successfully!
```

Mysql 을 통해 제대로 만들어졌는지 확인해보면 두 테이블이 추가적으로 잘 생성된 것을 볼 수 있다.

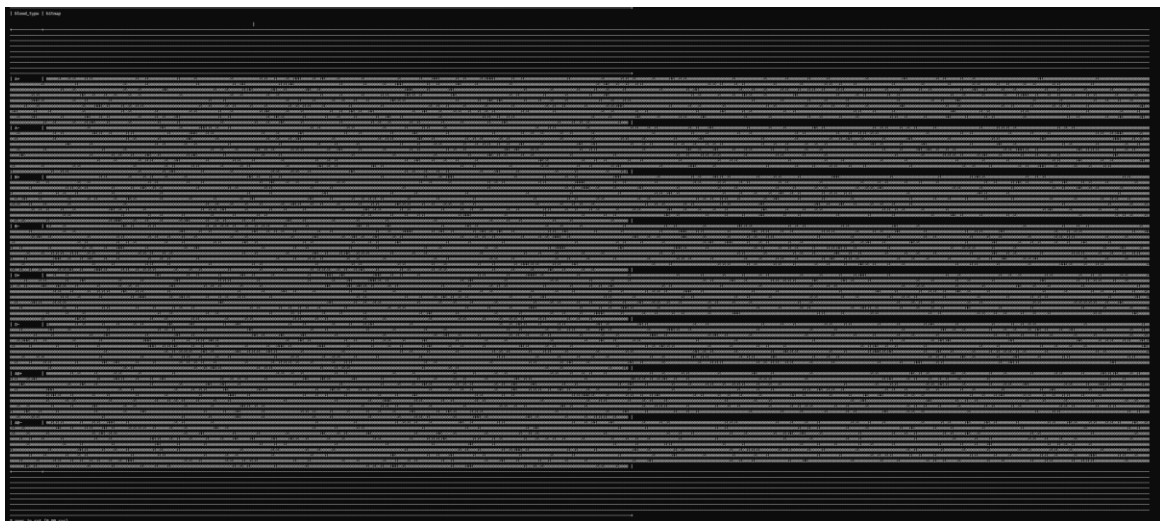
```
mysql> show tables;
+-----+
| Tables_in_blood_storage |
+-----+
| blood_donations          |
| blood_type_index         |
| donor_gender_index       |
+-----+
3 rows in set (0.01 sec)
```

#### 1. donor\_gender\_index



다음과 같이 각 성별에 대해 1 과 0 으로 구성된 비트 스트링으로 저장되어 있는 것을 볼 수 있다.

#### 2. blood\_type\_index



---

*Multiple key query 기능*

---

[illegible]

이어서 결과 비트맵에서 비트가 1 인 레코드들을 쭉 출력한다.

페이지 21

---

### Count query 기능

---

위의 multiple key query 기능처럼 다양한 경우가 있으므로, 혈액형이 A-(RH- A 형)인 사람 수를 구하고 싶다고 가정해보겠다. 기능을 실행해보면 다음과 같이 잘 출력됨을 알 수 있다.

```
1. Create table
2. Insert record
3. Making Bitmap Index
4. Multiple Key Query
5. Count Query
6. Exit
Enter your choice: 5
Enter attribute (Gender/BloodType) : BloodType
Enter blood type(A+, A-, B+, B-, O+, O-, AB+, AB-) : A-
혈액형이 A- 인 사람의 수 : 485
```

MySQL 문으로 잘 동작했는지 확인해보면 잘 출력됨을 알 수 있다.

```
mysql> SELECT blood_type, count(*) from blood_donations group by blood_type;
+-----+-----+
| blood_type | count(*) |
+-----+-----+
| O-        | 505      |
| B-        | 540      |
| AB-       | 482      |
| O+        | 491      |
| A+        | 511      |
| AB+       | 520      |
| A-        | 485      |
| B+        | 466      |
+-----+-----+
8 rows in set (0.00 sec)
```

---

## 종료 기능

---

```
1. Create table
2. Insert record
3. Making Bitmap Index
4. Multiple Key Query
5. Count Query
6. Exit
Enter your choice: 6
Exiting...

종료 코드 0(으)로 완료된 프로세스
```

6 번을 입력하면 프로그램이 종료된다.

## 실행파일 만드는 법

개발환경 : WINDOWS 11 환경에서 프로젝트를 진행했으며 JAVA(SDK: ORACLE OPEN JDK VERSION 20)을 사용해서 구현을 진행하였다. IDE 는 INTELLIJ IDEA 를 사용했으며 MySQL 을 사용했다.

실행파일 만드는 법은 다음과 같다.

IntelliJ IDEA 환경에서 파일->프로젝트 구조->아티팩트->+를 누른다->JAR->종속 요소 포함 모듈에서 선택->메인 클래스 선택 후 확인

이어서 빌드->아티팩트 필드->빌드

out/artifacts/DBS\_Project\_jar/DBS\_Project.jar 파일이 생성된 것을 확인할 수 있다.

DBS\_Project.jar 에서 터미널을 실행하여 java -jar ./DBS\_Project.jar 이라고 입력하면 기능이 잘 동작함을 볼 수 있다.

여기서 주의할 점은 mysql-connector-j-8.0.33.jar 파일이 종속된 채로 실행파일이 만들어져야 에러가 나지 않는다는 점이다.



