데이터베이스 설계 Part2

1. 기능 테스트 데이터 02

2. 설계 03

3. 구현 04

4. 실행 결과 07

5. 정확성 검증 12

6. 소스코드 16

1. 기능 테스트 데이터

   기능을 테스트하기 위해 아래와 같이 6개의 인덱스 파일과 여기에 삽입될 6개의 테이블을 준비한다.

   1) 테이블 1: 정상적인 테이블로, 다른 테이블들과 Join될 기준점이다.

f1;3;A0;B0;C0;4;4;4
10,0001;X001;Y001|0002;null;Y002|0003;X003;null|0004;null;Y004|0005;X005;null|0006;X006;Y006|0007;null;null|0010;X010;Y010|0008;null;Y008|0009;X009;null

   2) 테이블 2: 테이블 1과 같이 정상적인 테이블이다.

f2;3;A0;D0;E0;4;4;4
10,0001;Z001;W001|0002;null;W002|0003;Z003;null|0004;null;W004|0005;Z005;null|0006;Z006;W006|0007;null;null|0010;Z010;W010|0008;null;W008|0009;Z009;null

   3) 테이블 3: 이상이 존재하는 테이블로, 레코드 간 간격이 존재하고 일부 레코드는 중복되어 있다.

f3;3;A0;F0;G0;4;4;4
10,0001;Z001;W001|0002;Z002;W002|0006;Z006;W006|0006;Z007;W007|0006;Z006;W006|0008;Z008;W008|0010;Z010;W010|0010;Z013;W013|0011;Z011;W011|0012;Z012;W012

   4) 테이블 4: 2개의 column만 존재하는 테이블로 구성은 정상적이다.

f4;2;A0;H0;4;5
10,0001;A0001|0002;null|0003;A0003|0004;null|0005;A0005|0006;A0006|0007;null|0010;A0010|0008;null|0009;A0009

   5) 테이블 5: Primary key가 아닌 필드를 조인 키로 테스트하기 위한 테이블이다.

f5;3;I0;B0;J0;4;4;8
10,0001;B001;C0000001|0002;null;C0000002|0003;B003;null|0004;null;C0000004|0005;B005;null|0006;B006;C0000006|0007;null;null|0010;B010;C0000010|0008;null;C0000008|0009;B009;null

   6) 테이블 6: 두 테이블을 조인하기 위한 조인 키가 존재하되 그 필드 길이가 다른 테이블이다.

f6;4;A0;J0;K0;L0;5;4;4;4
10,0001;X001;C0000001|0002;null;C0000002|0003;X003;null|0004;null;C0000004|0005;X005;null|0006;X006;C0000006|0007;null;null|0010;X010;C0000010|0008;null;C0000008|0009;X009;null

2. 설계
   A. 요구사항 분석
      i. 입력: 2개의 관계형 테이블, 조인 키
      ii. 출력: 조인 결과를 파일로 저장, SQL 조인 결과와 검증
   B. 알고리즘
      i. 머지 조인
         1) 두 테이블이 정렬되어 있어야 연속 스캔으로 조인이 가능하다.
         2) 이미 정렬된 인덱스가 없으면 외부 정렬이 필요하다.
         3) 각 블록들은 오직 한 번만 읽어야 한다.
         4) 조인 속성 위의 같은 값을 가진 모든 쌍은 match되어야 한다.
      ii. 버퍼 활용
         1) 같은 조인 키 그룹의 레코드를 메모리 내에 버퍼링한다.
         2) 각 테이블에서 키가 바뀔 때까지 누적하여 두 그룹 간의 Cartesian product를 수행한다.
   C. 데이터 구조
      i. 메타데이터: 기존의 컬럼 이름-타입-고정 길이를 파일 헤더에 저장한다.
      ii. 버퍼: 리스트를 사용해 그룹 단위로 모아서 조인한 다음 해제한다.
   D. 설계
      i. 헤더 조회: 두 테이블에서 LIMIT 1 쿼리로 컬럼 메타데이터를 수집한다.
      ii. 머지 조인 실행
         1) 양쪽 커서를 첫 레코드로 이동한다.
         2) 양쪽이 모두 유효한 동안:
            i. 키를 비교하면서 작은 쪽으로 커서를 이동시킨다.
            ii. 동일할 시 동일한 키가 끝날 때까지 버퍼를 담는다.
            iii. 두 버퍼 사이에 중첩 반복문으로 레코드를 결합한다.
      iii. 결과 저장: 중복을 체크한 뒤 결과를 획득한다.
      iv. 검증: 사용자가 작성한 SQL 결과를 읽고 이를 내부 조인 결과와의 문자열 집합으로 비교한다.
   E. 관리
      i. 그룹 크기가 다를 시: 필드 길이를 32로 통일하여 처리한다.
      ii. 조인 그룹을 처리한 후 즉시 버퍼를 비운다.
      iii. 키가 NULL인 레코드는 무시한다.
      iv. 조인 키로 인해 동일한 컬럼이 존재할 경우, 둘 다 표시한다.

3. 구현

A. JoinProcessor

i. performMergeJoinRows(): 두 테이블을 머지 조인 방식으로 읽어 들여 키가 같은 모든 레코드 쌍을 메모리 내에서 결합한 결과 리스트를 반환한다.

```
16  private List<List<String>> performMergeJoinRows(Connection conn, String tableR, String tableS, String joinKey) throws SQLException {
17      List<List<String>> result = new ArrayList<>();
18
19      try (
20          Statement stR = conn.createStatement(
21              ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
22          Statement stS = conn.createStatement(
23              ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
24          ResultSet rsR = stR.executeQuery(
25              "SELECT * FROM " + tableR + " ORDER BY " + joinKey);
26          ResultSet rsS = stS.executeQuery(
27              "SELECT * FROM " + tableS + " ORDER BY " + joinKey)
28      ) {
29          ResultSetMetaData mR = rsR.getMetaData(), mS = rsS.getMetaData();
30          int cR = mR.getColumnCount(), cS = mS.getColumnCount();
31          int idxR = -1, idxS = -1;
32          for (int i = 1; i <= cR; i++) {
33              if (mR.getColumnName(i).equalsIgnoreCase(joinKey)) idxR = i;
34          }
35          for (int i = 1; i <= cS; i++) {
36              if (mS.getColumnName(i).equalsIgnoreCase(joinKey)) idxS = i;
37          }
38          if (idxR < 0 || idxS < 0) {
39              throw new SQLException("Join key not found in both tables");
40          }
41
42          boolean hasR = rsR.next(), hasS = rsS.next();
43          while (hasR && hasS) {
44              String keyR = rsR.getString(idxR), keyS = rsS.getString(idxS);
45
46              // NULL은 매칭 대상에서 제외
47              if (keyR == null) { hasR = rsR.next(); continue; }
48              if (keyS == null) { hasS = rsS.next(); continue; }
49
50              int cmp = keyR.compareTo(keyS);
51              if (cmp < 0) {
52                  hasR = rsR.next();
53                } else if (cmp > 0) {
54                    hasS = rsS.next();
55                } else {
56                    // 같은 키 그룹 버퍼링
57                    List<List<String>> bufR = new ArrayList<>();
58                    List<List<String>> bufS = new ArrayList<>();
59                    String cur = keyR;
60                    do {
61                        bufR.add(readRow(rsR, cR));
62                        hasR = rsR.next();
63                    } while (hasR && cur.equals(rsR.getString(idxR)));
64                    do {
65                        bufS.add(readRow(rsS, cS));
66                        hasS = rsS.next();
67                    } while (hasS && cur.equals(rsS.getString(idxS)));
68
69                    // 조인
70                    for (List<String> r : bufR) {
71                        for (List<String> s : bufS) {
72                            List<String> merged = new ArrayList<>(r);
73                            for (int j = 0; j < s.size(); j++) {
74                                merged.add(s.get(j));
75                            }
76                            result.add(merged);
77                        }
78                    }
79                }
80            }
81        }
82
83      return result;
84  }
```

1) 두 테이블 R, S을 조인 키로 오름차순 정렬하여 읽어 들인다. 이때 양쪽에서 키를 찾지 못하면 SQLException이 발생한다.
2) 첫 번째 레코드를 지정하고 둘 다 데이터가 남아있는 동안의 루프에 진입한다.
3) 두 테이블을 지정된 키로 오름차순 정렬하여 ResultSet을 생성한다.
4) 두 커서를 동시에 전진시켜 키를 비교한다. keyR가 keyS보다 작으면 R쪽 커서를 전진시키고, keyR이 keyS보다 크면 S쪽 커서를 전진시킨다.
5) 동일할 시 같은 키 그룹을 버퍼링 후 교차 곱을 수행한다.
    1) 버퍼링: 같은 키를 갖는 연속 레코드를 bufR와 bufS에 각각 모아둔다.
    2) 조인: 두 버퍼의 모든 조합을 합쳐 Cartesian product를 수행해 새로운 레코드인 필드 값 리스트를 만들어 결과 리스트에 추가한다.
6) 이때 중복되는 키나 NULL 값은 건너뛴다.
7) 결과를 반환한다.

ii. executeMergeJoin(): 머지 조인을 수행하여 파일에 저장하고 출력한다.

```java
public void executeMergeJoin(String tableR, String tableS, String joinKey, String outTable) throws Exception {
    String datFile = outTable + ".dat";
    Set<String> seenMerged = new HashSet<>();

    try (Connection conn = DriverManager.getConnection(URL, USER, PASS)) {
        // 테이블 별 헤더 메타데이터 구성
        List<FileStructure.FieldInfo> fields = new ArrayList<>();
        Set<String> seen = new HashSet<>();

        // 테이블 1 메타데이터 조회
        try (
            Statement hdrStR = conn.createStatement();
            ResultSet rs0R   = hdrStR.executeQuery(
                "SELECT * FROM " + tableR + " ORDER BY " + joinKey + " LIMIT 1")
        ) {
            ResultSetMetaData mR = rs0R.getMetaData();
            int cR = mR.getColumnCount();
            for (int i = 1; i <= cR; i++) {
                String name = mR.getColumnName(i);
                fields.add(new FileStructure.FieldInfo(name, 32));
                seen.add(name);
            }
        }

        // 테이블 2 메타데이터 조회
        try (
            Statement hdrStS = conn.createStatement();
            ResultSet rs0S   = hdrStS.executeQuery(
                "SELECT * FROM " + tableS + " ORDER BY " + joinKey + " LIMIT 1")
        ) {
            ResultSetMetaData mS = rs0S.getMetaData();
            int cS = mS.getColumnCount();
            for (int i = 1; i <= cS; i++) {
                String name = mS.getColumnName(i);
                if (!seen.contains(name)) {
                    fields.add(new FileStructure.FieldInfo(name, 32));
                    seen.add(name);
                }
            }
        }

        FileStructure.Metadata fsMeta = new FileStructure.Metadata(fields);

        // 파일 헤더 기록
        try (RandomAccessFile raf = new RandomAccessFile(datFile, "rw")) {
            fileStructure.writeHeader(raf, fsMeta);
        }
        // 실제 머지 조인 결과 받아오기
        List<List<String>> rows = performMergeJoinRows(conn, tableR, tableS, joinKey);

        // 중복 없이 저장 및 출력
        for (List<String> merged : rows) {
            String keyStr = String.join("|", merged);
            if (seenMerged.add(keyStr)) {
                System.out.println("Insert record: " + merged);
                fileStructure.insertRecord(
                    datFile,
                    fsMeta,
                    new FileStructure.Record(merged, (byte)0)
                );
            }
        }

        System.out.println("Merge join completed. Result saved to '" + datFile + "'.");
    }
}
```

1) 출력 파일명과 중복 체크를 준비한다.
2) JDBC로 메타데이터를 조회해 파일 헤더를 정의한다.
    1) 테이블 R의 메타데이터를 읽으며 각 컬럼 별 이름을 수집한다.
    2) 테이블 S의 메타데이터를 읽으며 동일하게 수집한다.
    3) 이렇게 모은 리스트로 메타데이터를 생성한다.
3) 파일 헤더를 기록한다.
4) performMergeJoinRows를 호출하여 머지 조인의 결과를 획득한다.
5) 결과 레코드를 파일에 기록하고 메시지를 출력한다.

iii. validateWithSqlJoin(): 내부 머지 조인 결과가 SQL 질의를 통한 조인 결과를 비교한다.

```java
public void validateWithSqlJoin(String sql, String tableR, String tableS, String joinKey) throws Exception {
    List<List<String>> sqlRows = new ArrayList<>();
    try (
        Connection conn = DriverManager.getConnection(URL, USER, PASS);
        Statement st    = conn.createStatement();
        ResultSet rs    = st.executeQuery(sql)
    ) {
        ResultSetMetaData md = rs.getMetaData();
        int sqlCols = md.getColumnCount();

        List<Integer> useIdx = new ArrayList<>();
        for (int i = 1; i <= sqlCols; i++) {
            useIdx.add(i);
        }

        while (rs.next()) {
            List<String> row = new ArrayList<>(useIdx.size());
            for (int idx : useIdx) {
                String v = rs.getString(idx);
                row.add(v == null ? "" : v);
            }
            sqlRows.add(row);
        }

        // 내부 머지 결과 획득
        List<List<String>> mergeRows = performMergeJoinRows(conn, tableR, tableS, joinKey);

        // 결과 비교
        Set<String> sqlSet   = sqlRows.stream()
                                      .map(r -> String.join("|", r))
                                      .collect(Collectors.toSet());
        Set<String> mergeSet = mergeRows.stream()
                                        .map(r -> String.join("|", r))
                                        .collect(Collectors.toSet());

          Set<String> onlyInSql   = new HashSet<>(sqlSet);
          onlyInSql.removeAll(mergeSet);
          Set<String> onlyInMerge = new HashSet<>(mergeSet);
          onlyInMerge.removeAll(sqlSet);

          // 결과 출력
          System.out.println("SQL JOIN row count:   " + sqlSet.size());
          System.out.println("Merge-Join row count: " + mergeSet.size());
          System.out.println("Only in SQL (" + onlyInSql.size() + "):");
          onlyInSql.stream().limit(10).forEach(r -> System.out.println("   " + r));
          System.out.println("Only in Merge (" + onlyInMerge.size() + "):");
          onlyInMerge.stream().limit(10).forEach(r -> System.out.println("   " + r));
      }
  }
```

1) SQL 질의 실행 후 리스트 결과를 얻는다.
2) performMergeJoinRows를 호출하여 내부 조인 결과를 획득한다.
3) 두 결과를 문자열 set으로 변환해 차집합을 계산한다.
4) 각 집합의 크기와 서로 다른 샘플 행(최대 10개)을 출력한다.

iv. readRow(): 현재 ResultSet 커서가 위치한 행의 첫 번째 컬럼부터 cols개까지 String으로 읽어 리스트로 반환한다.

```java
private List<String> readRow(ResultSet rs, int cols) throws SQLException {
    List<String> row = new ArrayList<>(cols);
    for (int i = 1; i <= cols; i++) {
        String v = rs.getString(i);
        row.add(v == null ? "" : v);
    }
    return row;
}
```

B. 기존 코드 수정

i. FileStructure.java의 readRecord()

```java
public Record readRecord(RandomAccessFile raf, Metadata meta, int block, int offset) throws IOException {
    raf.seek((long)block * BLOCK_SIZE + offset);
    int nextBlock = raf.readInt();
    int nextOffset= raf.readInt();
    byte nullBitmap = raf.readByte();
    List<String> vals = new ArrayList<>(meta.fieldCount);
    for (FieldInfo f : meta.fields) {
        byte[] buf = new byte[f.length];
        raf.readFully(buf);
        vals.add(new String(buf, StandardCharsets.UTF_8).trim());
    }
    Record r = new Record(vals, nullBitmap);
    r.nextRecordBlock  = nextBlock;
    r.nextRecordOffset = nextOffset;
    return r;
}
```

1) raf.readFully로 바이트 길이를 정확하게 읽도록 한다.
2) nullBitmap을 별도로 읽게 하여 null 여부를 판별할 수 있도록 한다.
3) UTF-8 인코딩을 명시하여 오류를 잡을 수 있도록 한다.

ii. MainApp.java의 case 7: SQL Merge Process

```
85                case "7":
86                    System.out.print("Enter first table file name: ");
87                    String tableR = scanner.nextLine().trim();
88                    System.out.print("Enter second table file name: ");
89                    String tableS = scanner.nextLine().trim();
90                    System.out.print("Enter join key column name: ");
91                    String joinKey = scanner.nextLine().trim();
92                    System.out.print("Enter output file name (without .dat): ");
93                    String outName = scanner.nextLine().trim();
94
95                    JoinProcessor jp = new JoinProcessor();
96                    try {
97                        jp.executeMergeJoin(tableR, tableS, joinKey, outName);
98                    } catch (Exception e) {
99                        System.out.println("Failed to run merge join: " + e.getMessage());
100                   }
101                   break;
```

1) 테이블 1, 테이블2, 조인 키, 저장할 파일 이름을 입력 받는다.
2) 입력 받은 정보로 Merge Join을 수행한다.

iii. MainApp.java의 case 8: Validate SQL JOIN vs Merge-Join

```
104               case "8":
105                   System.out.print("Enter full SQL JOIN query: ");
106                   String userSql = scanner.nextLine().trim();
107
108                   System.out.print("Enter first table name (for merge algorithm): ");
109                   String tblR = scanner.nextLine().trim();
110                   System.out.print("Enter second table name: ");
111                   String tblS = scanner.nextLine().trim();
112                   System.out.print("Enter join key column name: ");
113                   String jk   = scanner.nextLine().trim();
114
115                   JoinProcessor validator = new JoinProcessor();
116                   try {
117                       validator.validateWithSqlJoin(userSql, tblR, tblS, jk);
118                   } catch (Exception e) {
119                       System.out.println("Validation failed: " + e.getMessage());
120                   }
121                   break;
```

1) MySQL에 보낼 SQL 조인 질의를 입력 받는다.
2) 내부 Merge Join을 수행할 테이블 1, 테이블2, 조인 키를 입력 받는다.
3) 입력 받은 질의와 정보로 두 조인 결과 테이블을 비교하여 출력한다.

4. 실행 결과

A. 조인 기능

i. 테이블 1을 생성한다.

ii. 테이블 1 - 테이블 2 조인

1) 테이블 2를 생성한다.

```
All records inserted into MySQL table 'f2'.
Auto search: range 0001 ~ 0010
Block:1 Offset:0 Search-key:0001
D0: Z001
E0: W001
Block:1 Offset:21 Search-key:0002
D0: null
E0: W002
Block:1 Offset:42 Search-key:0003
D0: Z003
E0: null
Block:1 Offset:63 Search-key:0004
D0: null
E0: W004
Block:1 Offset:84 Search-key:0005
D0: Z005
E0: null
Block:1 Offset:105 Search-key:0006
D0: Z006
E0: W006
Block:1 Offset:126 Search-key:0007
D0: null
E0: null
Block:1 Offset:147 Search-key:0008
D0: null
E0: W008
Block:1 Offset:168 Search-key:0009
D0: Z009
E0: null
Block:1 Offset:189 Search-key:0010
D0: Z010
E0: W010
```

2) 테이블 1과 테이블 2를 조인한다.

```
Enter first table file name: f1
Enter second table file name: f2
Enter join key column name: A0
Enter output file name (without .dat): join12
Insert record: [0001, X001, Y001, 0001, Z001, W001]
Insert record: [0002, , Y002, 0002, , W002]
Insert record: [0003, X003, , 0003, Z003, ]
Insert record: [0004, , Y004, 0004, , W004]
Insert record: [0005, X005, , 0005, Z005, ]
Insert record: [0006, X006, Y006, 0006, Z006, W006]
Insert record: [0007, , , 0007, , ]
Insert record: [0008, , Y008, 0008, , W008]
Insert record: [0009, X009, , 0009, Z009, ]
Insert record: [0010, X010, Y010, 0010, Z010, W010]
Merge join completed. Result saved to 'join12.dat'.
```

iii. 테이블 1 - 테이블 3 조인

1) 테이블 3을 생성한다.

```
All records inserted into MySQL table 'f3'.
Auto search: range 0001 ~ 0012
Block:1 Offset:0 Search-key:0001
F0: Z001
G0: W001
Block:1 Offset:21 Search-key:0002
F0: Z002
G0: W002
Block:1 Offset:42 Search-key:0006
F0: Z006
G0: W006
Block:1 Offset:63 Search-key:0006
F0: Z007
G0: W007
Block:1 Offset:84 Search-key:0006
F0: Z006
G0: W006
Block:1 Offset:105 Search-key:0008
F0: Z008
G0: W008
Block:1 Offset:126 Search-key:0010
F0: Z010
G0: W010
Block:1 Offset:147 Search-key:0010
F0: Z013
G0: W013
Block:1 Offset:168 Search-key:0011
F0: Z011
G0: W011
Block:1 Offset:189 Search-key:0012
F0: Z012
G0: W012
```

2) 테이블 1과 테이블 3을 조인한다.

```
Enter first table file name: f1
Enter second table file name: f3
Enter join key column name: A0
Enter output file name (without .dat): join13
Insert record: [0001, X001, Y001, 0001, Z001, W001]
Insert record: [0002, , Y002, 0002, Z002, W002]
Insert record: [0006, X006, Y006, 0006, Z006, W006]
Insert record: [0006, X006, Y006, 0006, Z007, W007]
Insert record: [0008, , Y008, 0008, Z008, W008]
Insert record: [0010, X010, Y010, 0010, Z010, W010]
Insert record: [0010, X010, Y010, 0010, Z013, W013]
Merge join completed. Result saved to 'join13.dat'.
```

iv. 테이블 1 - 테이블 4 조인

1) 테이블 4를 생성한다.

```
All records inserted into MySQL table 'f4'.
Auto search: range 0001 ~ 0010
Block:1 Offset:0 Search-key:0001
H0: A0001
Block:1 Offset:18 Search-key:0002
H0: null
Block:1 Offset:36 Search-key:0003
H0: A0003
Block:1 Offset:54 Search-key:0004
H0: null
Block:1 Offset:72 Search-key:0005
H0: A0005
Block:1 Offset:90 Search-key:0006
H0: A0006
Block:1 Offset:108 Search-key:0007
H0: null
Block:1 Offset:126 Search-key:0008
H0: null
Block:1 Offset:144 Search-key:0009
H0: A0009
Block:1 Offset:162 Search-key:0010
H0: A0010
```

2) 테이블 1과 테이블 4를 조인한다.

```
Enter first table file name: f1
Enter second table file name: f4
Enter join key column name: A0
Enter output file name (without .dat): join14
Insert record: [0001, X001, Y001, 0001, A0001]
Insert record: [0002, , Y002, 0002, ]
Insert record: [0003, X003, , 0003, A0003]
Insert record: [0004, , Y004, 0004, ]
Insert record: [0005, X005, , 0005, A0005]
Insert record: [0006, X006, Y006, 0006, A0006]
Insert record: [0007, , , 0007, ]
Insert record: [0008, , Y008, 0008, ]
Insert record: [0009, X009, , 0009, A0009]
Insert record: [0010, X010, Y010, 0010, A0010]
Merge join completed. Result saved to 'join14.dat'.
```

v. 테이블 1 - 테이블 5 조인

1) 테이블 5를 생성한다.

```
All records inserted into MySQL table 'f5'.
Auto search: range 0001 ~ 0010
Block:1 Offset:0 Search-key:0001
B0: X001
J0: C0000001
Block:1 Offset:25 Search-key:0002
B0: null
J0: C0000002
Block:1 Offset:50 Search-key:0003
B0: X003
J0: null
Block:1 Offset:75 Search-key:0004
B0: null
J0: C0000004
Block:1 Offset:100 Search-key:0005
B0: X005
J0: null
Block:1 Offset:125 Search-key:0006
B0: X006
J0: C0000006
Block:1 Offset:150 Search-key:0007
B0: null
J0: null
Block:1 Offset:175 Search-key:0008
B0: null
J0: C0000008
Block:1 Offset:200 Search-key:0009
B0: X009
J0: null
Block:1 Offset:225 Search-key:0010
B0: X010
J0: C0000010
```

2) 테이블 1과 테이블 5를 조인한다.

```
Enter first table file name: f1
Enter second table file name: f5
Enter join key column name: B0
Enter output file name (without .dat): join15
Insert record: [0001, X001, Y001, 0001, X001, C0000001]
Insert record: [0003, X003, , 0003, X003, ]
Insert record: [0005, X005, , 0005, X005, ]
Insert record: [0006, X006, Y006, 0006, X006, C0000006]
Insert record: [0009, X009, , 0009, X009, ]
Insert record: [0010, X010, Y010, 0010, X010, C0000010]
Merge join completed. Result saved to 'join15.dat'.
```

vi. 테이블 1 - 테이블 6 조인

1) 테이블 6을 생성한다.

```
All records inserted into MySQL table 'f6'
Auto search: range 0001 ~ 0010
Block:1 Offset:0 Search-key:0001
J0: X001
K0: Y001
L0: Z001
Block:1 Offset:26 Search-key:0002
J0: null
K0: Y002
L0: Z002
Block:1 Offset:52 Search-key:0003
J0: X003
K0: null
L0: Z003
Block:1 Offset:78 Search-key:0004
J0: null
K0: Y004
L0: Z004
Block:1 Offset:104 Search-key:0005
J0: X005
K0: null
L0: Z005
Block:1 Offset:130 Search-key:0006
J0: X006
K0: Y006
L0: Z006
Block:1 Offset:156 Search-key:0007
J0: null
K0: null
L0: Z007
Block:1 Offset:182 Search-key:0008
J0: null
K0: Y008
L0: Z008
Block:1 Offset:208 Search-key:0009
J0: X009
K0: null
L0: Z009
Block:1 Offset:234 Search-key:0010
J0: X010
K0: Y010
L0: Z010
```

2) 테이블 1과 테이블 6을 조인한다.

```
Enter first table file name: f1
Enter second table file name: f6
Enter join key column name: A0
Enter output file name (without .dat): join16
Insert record: [0001, X001, Y001, 0001, X001, Y001, Z001]
Insert record: [0002, , Y002, 0002, , Y002, Z002]
Insert record: [0003, X003, , 0003, X003, , Z003]
Insert record: [0004, , Y004, 0004, , Y004, Z004]
Insert record: [0005, X005, , 0005, X005, , Z005]
Insert record: [0006, X006, Y006, 0006, X006, Y006, Z006]
Insert record: [0007, , , 0007, , , Z007]
Insert record: [0008, , Y008, 0008, , Y008, Z008]
Insert record: [0009, X009, , 0009, X009, , Z009]
Insert record: [0010, X010, Y010, 0010, X010, Y010, Z010]
Merge join completed. Result saved to 'join16.dat'.
```

B. 기존 기능 테스트

i. 필드 탐색

1-2 테이블 조인	1-3 테이블 조인
File name: join12 Search field name: A0 Block:1 Offset:0 Field A0 Value: 0001 Block:1 Offset:169 Field A0 Value: 0002 Block:1 Offset:338 Field A0 Value: 0003 Block:1 Offset:507 Field A0 Value: 0004 Block:1 Offset:676 Field A0 Value: 0005 Block:1 Offset:845 Field A0 Value: 0006 Block:2 Offset:0 Field A0 Value: 0007 Block:2 Offset:169 Field A0 Value: 0008 Block:2 Offset:338 Field A0 Value: 0009 Block:2 Offset:507 Field A0 Value: 0010	Selection: 3 File name: join13 Search field name: A0 Block:1 Offset:0 Field A0 Value: 0001 Block:1 Offset:169 Field A0 Value: 0002 Block:1 Offset:338 Field A0 Value: 0006 Block:1 Offset:507 Field A0 Value: 0006 Block:1 Offset:676 Field A0 Value: 0008 Block:1 Offset:845 Field A0 Value: 0010 Block:2 Offset:0 Field A0 Value: 0010
1-4 테이블 조인	1-5 테이블 조인
File name: join14 Search field name: A0 Block:1 Offset:0 Field A0 Value: 0001 Block:1 Offset:137 Field A0 Value: 0002 Block:1 Offset:274 Field A0 Value: 0003 Block:1 Offset:411 Field A0 Value: 0004 Block:1 Offset:548 Field A0 Value: 0005 Block:1 Offset:685 Field A0 Value: 0006 Block:1 Offset:822 Field A0 Value: 0007 Block:2 Offset:0 Field A0 Value: 0008 Block:2 Offset:137 Field A0 Value: 0009 Block:2 Offset:274 Field A0 Value: 0010	File name: join15 Search field name: A0 Block:1 Offset:0 Field A0 Value: 0001 Block:1 Offset:169 Field A0 Value: 0003 Block:1 Offset:338 Field A0 Value: 0005 Block:1 Offset:507 Field A0 Value: 0006 Block:1 Offset:676 Field A0 Value: 0009 Block:1 Offset:845 Field A0 Value: 0010

ii. 레코드 범위 탐색

1-2 테이블 조인	

```
Block:1 Offset:169 Search-key:0002
B0:
C0: Y002
D0: 0002
E0:
Block:1 Offset:338 Search-key:0003
B0: X003
C0:
D0: 0003
E0: Z003
Block:1 Offset:507 Search-key:0004
B0:
C0: Y004
D0: 0004
E0:
Block:1 Offset:676 Search-key:0005
B0: X005
C0:
D0: 0005
E0: Z005
```

```
Block:1 Offset:845 Search-key:0006
B0: X006
C0: Y006
D0: 0006
E0: Z006
Block:2 Offset:0 Search-key:0007
B0:
C0:
D0: 0007
E0:
Block:2 Offset:169 Search-key:0008
B0:
C0: Y008
D0: 0008
E0:
Block:2 Offset:338 Search-key:0009
B0: X009
C0:
D0: 0009
E0: Z009
```

1-5 테이블 조인

```
Block:1 Offset:0 Search-key:0001
B0: X001
C0: Y001
I0: 0001
J0: X001
Block:1 Offset:169 Search-key:0003
B0: X003
C0:
I0: 0003
J0: X003
Block:1 Offset:338 Search-key:0005
B0: X005
C0:
I0: 0005
J0: X005
```

```
Block:1 Offset:507 Search-key:0006
B0: X006
C0: Y006
I0: 0006
J0: X006
Block:1 Offset:676 Search-key:0009
B0: X009
C0:
I0: 0009
J0: X009
Block:1 Offset:845 Search-key:0010
B0: X010
C0: Y010
I0: 0010
J0: X010
```

iii. 메타데이터 확인

1) 테이블 1 - 테이블 2 조인

```
Enter file name (without extension) to check metadata: join12
Metadata is loaded to memory from File join12.dat
Field numbers: 5
List of fields in memory for file 'join12.dat':
  0: A0 (length=32)
  1: B0 (length=32)
  2: C0 (length=32)
  3: D0 (length=32)
  4: E0 (length=32)
```

2) 테이블 1 - 테이블 3 조인

```
Enter file name (without extension) to check metadata: join13
Metadata is loaded to memory from File join13.dat
Field numbers: 5
List of fields in memory for file 'join13.dat':
  0: A0 (length=32)
  1: B0 (length=32)
  2: C0 (length=32)
  3: F0 (length=32)
  4: G0 (length=32)
```

3) 테이블 1 - 테이블 4 조인

```
Enter file name (without extension) to check metadata: join14
Metadata is loaded to memory from File join14.dat
Field numbers: 4
List of fields in memory for file 'join14.dat':
  0: A0 (length=32)
  1: B0 (length=32)
  2: C0 (length=32)
  3: H0 (length=32)
```

4) 테이블 1 - 테이블 5 조인

```
Enter file name (without extension) to check metadata: join15
Metadata is loaded to memory from File join15.dat
Field numbers: 5
List of fields in memory for file 'join15.dat':
  0: A0 (length=32)
  1: B0 (length=32)
  2: C0 (length=32)
  3: I0 (length=32)
  4: J0 (length=32)
```

5) 테이블 1 - 테이블 6 조인

```
Enter file name (without extension) to check metadata: join16
Metadata is loaded to memory from File join16.dat
Field numbers: 6
List of fields in memory for file 'join16.dat':
  0: A0 (length=32)
  1: B0 (length=32)
  2: C0 (length=32)
  3: J0 (length=32)
  4: K0 (length=32)
  5: L0 (length=32)
```

5. 정확성 검증

위에서 테스트하지 않았던 Validate SQL JOIN vs Merge-Join 기능으로 정확성을 검증한다.

이때 두 테이블을 직접 비교하기 위해서 validateWithSqlJoin()의 끝에 다음과 같은 코드를 추가하고, 이는 각각의 작업에 대한 결과 테이블을 출력한다.

```
System.out.println("=== SQL JOIN 결과 ===");
Set<String> seen = new HashSet<>();
for (List<String> row : sqlRows) {
    String key = String.join("|", row);
    if (seen.add(key)) {
        System.out.println(String.join(" | ", row));
    }
}
System.out.println("=== 내부 머지-조인 결과 ===");
seen.clear();
for (List<String> row : mergeRows) {
    String key = String.join("|", row);
    if (seen.add(key)) {
        System.out.println(String.join(" | ", row));
    }
}
```

A. 테이블 1 - 테이블 2 조인

```
SELECT f1.*, f2.* FROM f1 JOIN f2 ON f1.A0 = f2.A0;
```

위의 SQL 질의를 실행해 비교한 결과는 다음과 같다.

```
Enter full SQL JOIN query: SELECT f1.*, f2.* FROM f1 JOIN f2 ON f1.A0 = f2.A0;
Enter first table name (for merge algorithm): f1
Enter second table name: f2
Enter join key column name: A0
SQL JOIN row count:   10
Merge-Join row count: 10
Only in SQL (0):
Only in Merge (0):
```

SQL 질의를 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== SQL JOIN 결과 ===
0001 | X001 | Y001 | 0001 | Z001 | W001
0002 |  | Y002 | 0002 |  | W002
0003 | X003 |  | 0003 | Z003 |
0004 |  | Y004 | 0004 |  | W004
0005 | X005 |  | 0005 | Z005 |
0006 | X006 | Y006 | 0006 | Z006 | W006
0007 |  |  | 0007 |  |
0008 |  | Y008 | 0008 |  | W008
0009 | X009 |  | 0009 | Z009 |
0010 | X010 | Y010 | 0010 | Z010 | W010
```

내부 기능을 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== 내부 머지-조인 결과 ===
0001 | X001 | Y001 | 0001 | Z001 | W001
0002 |  | Y002 | 0002 |  | W002
0003 | X003 |  | 0003 | Z003 |
0004 |  | Y004 | 0004 |  | W004
0005 | X005 |  | 0005 | Z005 |
0006 | X006 | Y006 | 0006 | Z006 | W006
0007 |  |  | 0007 |  |
0008 |  | Y008 | 0008 |  | W008
0009 | X009 |  | 0009 | Z009 |
0010 | X010 | Y010 | 0010 | Z010 | W010
```

두 테이블이 동일하므로 따라서 해당 테이블들에 대해 정확하게 조인되었음을 확인할 수 있다.

B. 테이블 1 - 테이블 3 조인

```
SELECT f1.*, f3.* FROM f1 JOIN f3 ON f1.A0 = f3.A0;
```

위의 SQL 질의를 실행해 비교한 결과는 다음과 같다.

```
Enter full SQL JOIN query: SELECT f1.*, f3.* FROM f1 JOIN f3 ON f1.A0 = f3.A0;
Enter first table name (for merge algorithm): f1
Enter second table name: f3
Enter join key column name: A0
SQL JOIN row count:   7
Merge-Join row count: 7
Only in SQL (0):
Only in Merge (0):
```

SQL 질의를 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== SQL JOIN 결과 ===
0001 | X001 | Y001 | 0001 | Z001 | W001
0002 |  | Y002 | 0002 | Z002 | W002
0006 | X006 | Y006 | 0006 | Z006 | W006
0006 | X006 | Y006 | 0006 | Z007 | W007
0008 |  | Y008 | 0008 | Z008 | W008
0010 | X010 | Y010 | 0010 | Z010 | W010
0010 | X010 | Y010 | 0010 | Z013 | W013
```

내부 기능을 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== 내부 머지-조인 결과 ===
0001 | X001 | Y001 | 0001 | Z001 | W001
0002 |  | Y002 | 0002 | Z002 | W002
0006 | X006 | Y006 | 0006 | Z006 | W006
0006 | X006 | Y006 | 0006 | Z007 | W007
0008 |  | Y008 | 0008 | Z008 | W008
0010 | X010 | Y010 | 0010 | Z010 | W010
0010 | X010 | Y010 | 0010 | Z013 | W013
```

두 테이블이 동일하므로 따라서 해당 테이블들에 대해 정확하게 조인되었음을 확인할 수 있다.

C. 테이블 1 - 테이블 4 조인

```
SELECT f1.*, f4.* FROM f1 JOIN f4 ON f1.A0 = f4.A0;
```

위의 SQL 질의를 실행해 비교한 결과는 다음과 같다.

```
Enter full SQL JOIN query: SELECT f1.*, f4.* FROM f1 JOIN f4 ON f1.A0 = f4.A0;
Enter first table name (for merge algorithm): f1
Enter second table name: f4
Enter join key column name: A0
SQL JOIN row count:   10
Merge-Join row count: 10
Only in SQL (0):
Only in Merge (0):
```

SQL 질의를 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== SQL JOIN 결과 ===
0001 | X001 | Y001 | 0001 | A0001
0002 |  | Y002 | 0002 |
0003 | X003 |  | 0003 | A0003
0004 |  | Y004 | 0004 |
0005 | X005 |  | 0005 | A0005
0006 | X006 | Y006 | 0006 | A0006
0007 |  |  | 0007 |
0008 |  | Y008 | 0008 |
0009 | X009 |  | 0009 | A0009
0010 | X010 | Y010 | 0010 | A0010
```

내부 기능을 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== 내부 머지-조인 결과 ===
0001 | X001 | Y001 | 0001 | A0001
0002 |  | Y002 | 0002 |
0003 | X003 |  | 0003 | A0003
0004 |  | Y004 | 0004 |
0005 | X005 |  | 0005 | A0005
0006 | X006 | Y006 | 0006 | A0006
0007 |  |  | 0007 |
0008 |  | Y008 | 0008 |
0009 | X009 |  | 0009 | A0009
0010 | X010 | Y010 | 0010 | A0010
```

두 테이블이 동일하므로 따라서 해당 테이블들에 대해 정확하게 조인되었음을 확인할 수 있다.

D. 테이블 1 - 테이블 5 조인

```
SELECT f1.*, f5.* FROM f1 JOIN f5 ON f1.B0 = f5.B0;
```

위의 SQL 질의를 실행해 비교한 결과는 다음과 같다.

```
Enter full SQL JOIN query: SELECT f1.*, f5.* FROM f1 JOIN f5 ON f1.B0 = f5.B0;
Enter first table name (for merge algorithm): f1
Enter second table name: f5
Enter join key column name: B0
SQL JOIN row count:   6
Merge-Join row count: 6
Only in SQL (0):
Only in Merge (0):
```

SQL 질의를 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== SQL JOIN 결과 ===
0001 | X001 | Y001 | 0001 | X001 | C0000001
0003 | X003 |  | 0003 | X003 |
0005 | X005 |  | 0005 | X005 |
0006 | X006 | Y006 | 0006 | X006 | C0000006
0009 | X009 |  | 0009 | X009 |
0010 | X010 | Y010 | 0010 | X010 | C0000010
```

내부 기능을 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== 내부 머지-조인 결과 ===
0001 | X001 | Y001 | 0001 | X001 | C0000001
0003 | X003 |  | 0003 | X003 |
0005 | X005 |  | 0005 | X005 |
0006 | X006 | Y006 | 0006 | X006 | C0000006
0009 | X009 |  | 0009 | X009 |
0010 | X010 | Y010 | 0010 | X010 | C0000010
```

두 테이블이 동일하므로 따라서 해당 테이블들에 대해 정확하게 조인되었음을 확인할 수 있다.

E. 테이블 1 - 테이블 6 조인

```
SELECT f1.*, f6.* FROM f1 JOIN f6 ON f1.A0 = f6.A0;
```

위의 SQL 질의를 실행해 비교한 결과는 다음과 같다.

```
Enter full SQL JOIN query: SELECT f1.*, f6.* FROM f1 JOIN f6 ON f1.A0 = f6.A0;
Enter first table name (for merge algorithm): f1
Enter second table name: f6
Enter join key column name: A0
SQL JOIN row count:   10
Merge-Join row count: 10
Only in SQL (0):
Only in Merge (0):
```

SQL 질의를 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== SQL JOIN 결과 ===
0001 | X001 | Y001 | 0001 | X001 | Y001 | Z001
0002 |  | Y002 | 0002 |  | Y002 | Z002
0003 | X003 |  | 0003 | X003 |  | Z003
0004 |  | Y004 | 0004 |  | Y004 | Z004
0005 | X005 |  | 0005 | X005 |  | Z005
0006 | X006 | Y006 | 0006 | X006 | Y006 | Z006
0007 |  |  | 0007 |  |  | Z007
0008 |  | Y008 | 0008 |  | Y008 | Z008
0009 | X009 |  | 0009 | X009 |  | Z009
0010 | X010 | Y010 | 0010 | X010 | Y010 | Z010
```

내부 기능을 통해 생성한 조인 결과 테이블은 다음과 같다.

```
=== 내부 머지-조인 결과 ===
0001 | X001 | Y001 | 0001 | X001 | Y001 | Z001
0002 |  | Y002 | 0002 |  | Y002 | Z002
0003 | X003 |  | 0003 | X003 |  | Z003
0004 |  | Y004 | 0004 |  | Y004 | Z004
0005 | X005 |  | 0005 | X005 |  | Z005
0006 | X006 | Y006 | 0006 | X006 | Y006 | Z006
0007 |  |  | 0007 |  |  | Z007
0008 |  | Y008 | 0008 |  | Y008 | Z008
0009 | X009 |  | 0009 | X009 |  | Z009
0010 | X010 | Y010 | 0010 | X010 | Y010 | Z010
```

두 테이블이 동일하므로 따라서 해당 테이블들에 대해 정확하게 조인되었음을 확인할 수 있다.

만약 두 테이블이 동일하지 않을 시 다음과 같은 결과를 표시한다.

```
Enter full SQL JOIN query: SELECT f1.*, f6.* FROM f1 JOIN f6 ON f1.A0 = f6.A0;
Enter first table name (for merge algorithm): f1
Enter second table name: f2
Enter join key column name: A0
SQL JOIN row count:   10
Merge-Join row count: 10
Only in SQL (10):
   0006|X006|Y006|0006|X006|Y006|Z006
   0005|X005||0005|X005||Z005
   0007|||0007|||Z007
   0008||Y008|0008||Y008|Z008
   0002||Y002|0002||Y002|Z002
   0009|X009||0009|X009||Z009
   0004||Y004|0004||Y004|Z004
   0001|X001|Y001|0001|X001|Y001|Z001
   0010|X010|Y010|0010|X010|Y010|Z010
   0003|X003||0003|X003||Z003
Only in Merge (10):
   0003|X003||0003|Z003|
   0004||Y004|0004||W004
   0002||Y002|0002||W002
   0006|X006|Y006|0006|Z006|W006
   0007|||0007||
   0008||Y008|0008||W008
   0010|X010|Y010|0010|Z010|W010
   0005|X005||0005|Z005|
   0001|X001|Y001|0001|Z001|W001
   0009|X009||0009|Z009|
```

따라서 조인을 정확하게 수행하고 있음을 확인할 수 있다.

6. 소스코드

FileStructure.java

```
package main_package;

import java.io.RandomAccessFile;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.*;

public class FileStructure {
	private static final int BLOCK_SIZE = 1024;
	private static final int FIELD_NAME_SIZE = 16;

	// 필드 정보
	public static class FieldInfo {
	    String name;
	    int length;
	    public FieldInfo(String name, int length) {
	        this.name = name;
	        this.length = length;
	    }
	}

	// 메타데이터
	public static class Metadata {
	    int firstRecordBlock;
	    int firstRecordOffset;
	    int fieldCount;
	    List<FieldInfo> fields;

	    public Metadata(List<FieldInfo> fields) {
	        this.firstRecordBlock = -1;
	        this.firstRecordOffset = -1;
	        this.fieldCount = fields.size();
	        this.fields = fields;
	    }
	}

	// 레코드
	public static class Record {
	    int nextRecordBlock;
	    int nextRecordOffset;
	    List<String> fieldValues;
	    byte nullBitmap;

	    public Record(List<String> fieldValues, byte nullBitmap) {
	        this.nextRecordBlock = -1;
	        this.nextRecordOffset = -1;
	        this.fieldValues = fieldValues;
	        this.nullBitmap = nullBitmap;
	    }
	}

	// 메타데이터 헤더에 기록
	public void writeHeader(RandomAccessFile raf, Metadata meta) throws
```

```
        IOException {
            ByteBuffer buffer = ByteBuffer.allocate(BLOCK_SIZE);
            buffer.putInt(meta.firstRecordBlock);
            buffer.putInt(meta.firstRecordOffset);
            buffer.putInt(meta.fieldCount);

            for (FieldInfo field : meta.fields) {
                byte[] nameBytes =
        field.name.getBytes(StandardCharsets.UTF_8);
                byte[] fixedName = new byte[FIELD_NAME_SIZE];
                System.arraycopy(nameBytes, 0, fixedName, 0,
        Math.min(nameBytes.length, FIELD_NAME_SIZE));
                buffer.put(fixedName);
                buffer.putInt(field.length);
            }

            raf.seek(0);
            raf.write(buffer.array());
        }

        // 헤더에서 메타데이터 읽기
        public Metadata readHeader(RandomAccessFile raf) throws IOException
        {
            ByteBuffer buffer = ByteBuffer.allocate(BLOCK_SIZE);
            raf.seek(0);
            raf.readFully(buffer.array());
            buffer.rewind();
            int firstRecordBlock = buffer.getInt();
            int firstRecordOffset = buffer.getInt();
            int fieldCount = buffer.getInt();
            List<FieldInfo> fields = new ArrayList<>();

            for (int i = 0; i < fieldCount; i++) {
                byte[] fixedName = new byte[FIELD_NAME_SIZE];
                buffer.get(fixedName);
                String name = new String(fixedName,
        StandardCharsets.UTF_8).trim();
                int length = buffer.getInt();
                fields.add(new FieldInfo(name, length));
            }

            Metadata meta = new Metadata(fields);
            meta.firstRecordBlock = firstRecordBlock;
            meta.firstRecordOffset = firstRecordOffset;
            return meta;
        }

        // 레코드 크기 계산
        private int getRecordSize(Metadata meta) {
            int size = 4 + 4 + 1; // nextBlock + nextOffset + nullBitmap
            for (FieldInfo field : meta.fields) {
                size += field.length;
            }
            return size;
        }

        // 레코드 삽입
        public void insertRecord(String filename, Metadata meta, Record
```

```
        newRecord) throws IOException {
            try (RandomAccessFile raf = new RandomAccessFile(filename,
        "rw")) {
                int recordSize = getRecordSize(meta);
                int fileLength = (int) raf.length();
                int lastBlockStart = (fileLength / BLOCK_SIZE) * BLOCK_SIZE;
                int offsetInLastBlock = fileLength % BLOCK_SIZE;

                int newBlock = fileLength / BLOCK_SIZE;
                int newOffset = offsetInLastBlock;

                // 새 블록으로 이동
                if (offsetInLastBlock + recordSize > BLOCK_SIZE) {
                    newBlock++;
                    newOffset = 0;
                }

                // 헤더에 위치 기록
                if (meta.firstRecordBlock == -1 && meta.firstRecordOffset ==
        -1) {
                    meta.firstRecordBlock = newBlock;
                    meta.firstRecordOffset = newOffset;
                    raf.seek(0);
                    raf.writeInt(meta.firstRecordBlock);
                    raf.writeInt(meta.firstRecordOffset);
                } else {
                      // 새로운 레코드로 연결
                    int currBlock = meta.firstRecordBlock;
                    int currOffset = meta.firstRecordOffset;

                    while (true) {
                        raf.seek((long) currBlock * BLOCK_SIZE + currOffset);
                        int nextBlock = raf.readInt();
                        int nextOffset = raf.readInt();

                        if (nextBlock == -1 && nextOffset == -1) {
                            raf.seek((long) currBlock * BLOCK_SIZE +
        currOffset);
                            raf.writeInt(newBlock);
                            raf.writeInt(newOffset);
                            break;
                        } else {
                            currBlock = nextBlock;
                            currOffset = nextOffset;
                        }
                    }
                }

                writeRecord(raf, newBlock, newOffset, meta, newRecord);
            }
        }

        // 레코드 위치 기록
        public void writeRecord(RandomAccessFile raf, int block, int offset,
        Metadata meta, Record record) throws IOException {
            int recordSize = getRecordSize(meta);
            raf.seek((long) block * BLOCK_SIZE + offset);
```

```
            ByteBuffer buffer = ByteBuffer.allocate(recordSize);

            buffer.putInt(record.nextRecordBlock);
            buffer.putInt(record.nextRecordOffset);
            buffer.put(record.nullBitmap);

            for (int i = 0; i < meta.fieldCount; i++) {
                  byte[] fixedField = new byte[meta.fields.get(i).length];
                  Arrays.fill(fixedField, (byte) ' ');

                  // null 비트가 0이면 실제 값 복사, 1이면 공백 그대로 남김
                  if (((record.nullBitmap >> i) & 1) == 0) {
                      String val = record.fieldValues.get(i);
                      byte[] src = val.getBytes(StandardCharsets.UTF_8);
                      System.arraycopy(src, 0, fixedField, 0,
                                       Math.min(src.length,
      fixedField.length));
                  }
                  buffer.put(fixedField);
            }

            raf.write(buffer.array());
       }

       public Record readRecord(RandomAccessFile raf, Metadata meta, int
      block, int offset) throws IOException {
            raf.seek((long)block * BLOCK_SIZE + offset);
            int nextBlock = raf.readInt();
            int nextOffset= raf.readInt();
            byte nullBitmap = raf.readByte();
            List<String> vals = new ArrayList<>(meta.fieldCount);
            for (FieldInfo f : meta.fields) {
                byte[] buf = new byte[f.length];
                raf.readFully(buf);
                vals.add(new String(buf, StandardCharsets.UTF_8).trim());
            }
            Record r = new Record(vals, nullBitmap);
            r.nextRecordBlock  = nextBlock;
            r.nextRecordOffset = nextOffset;
            return r;
       }
}
```

RecordBulkManager.java

```
package main_package;

import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Types;
import java.util.*;

public class RecordBulkInserter {
       // 조인 테스트용 인자
       private static final boolean ALLOW_DUPLICATE_KEYS = false;

       private FileStructure fileStructure = new FileStructure();
```

```
    private RecordSearcher recordSearcher = new RecordSearcher();
    private MetadataManager metadataManager = new MetadataManager();

    public void bulkInsertFromDataFile(String fileName, String
dataFilePath) {
        try (BufferedReader br = new BufferedReader(new
FileReader(dataFilePath))) {
            String dataFilename = fileName + ".dat";

            FileStructure.Metadata meta;
            try (RandomAccessFile raf = new
RandomAccessFile(dataFilename, "rw")) {
                meta = fileStructure.readHeader(raf);
            } catch (IOException e) {
                System.out.println("Failed to read header from file: " +
fileName);
                return;
            }

            List<FileStructure.Record> recordList = new ArrayList<>();
            List<String> searchKeys = new ArrayList<>();

            String line;
            while ((line = br.readLine()) != null) {
                line = line.trim();
                if (line.isEmpty()) continue;

                String[] parts = line.split(",", 2);
                if (parts.length < 2) {
                    System.out.println("Invalid record line: " + line);
                    continue;
                }

                String recordsData = parts[1].trim();
                String[] recordStrArr = recordsData.split("\\|");
                for (String recStr : recordStrArr) {
                    String[] fieldValuesArr = recStr.split(";");
                    List<String> fieldValues = new ArrayList<>();
                    byte nullBitmap = 0;

                    for (int i = 0; i < fieldValuesArr.length; i++) {
                        String val = fieldValuesArr[i].trim();
                        if (val.isEmpty() || val.equalsIgnoreCase("null"))
{
                            nullBitmap |= (1 << i);
                            fieldValues.add("");
                        } else {
                            fieldValues.add(val);
                        }
                    }

                    String searchKey = fieldValues.get(0);
                    if (!searchKey.isEmpty()) {
                        searchKeys.add(searchKey);
                    }

                    FileStructure.Record record = new
FileStructure.Record(fieldValues, nullBitmap);
                    recordList.add(record);
```

```
                }
            }

            recordList.sort((r1, r2) ->
    r1.fieldValues.get(0).compareTo(r2.fieldValues.get(0)));

            for (int i = 0; i < recordList.size() - 1; i++) {
                String currKey = recordList.get(i).fieldValues.get(0);
                String nextKey = recordList.get(i +
    1).fieldValues.get(0);
                if (currKey.equals(nextKey) && !ALLOW_DUPLICATE_KEYS) {
                    System.out.println("Insertion failed due to
    duplicated keys in record file.");
                    return;
                }
            }

            for (FileStructure.Record rec : recordList) {
                fileStructure.insertRecord(dataFilename, meta, rec);
            }

            System.out.println("Records inserted into file '" +
    dataFilename + "' successfully (no duplicate keys).");

            metadataManager.createTableInMySQL(fileName, meta);
            insertAllRecordsIntoMySQL(fileName, meta, recordList);

            if (!searchKeys.isEmpty()) {
                Collections.sort(searchKeys);
                String minKey = searchKeys.get(0);
                String maxKey = searchKeys.get(searchKeys.size() - 1);
                System.out.println("Auto search: range " + minKey + " ~ "
    + maxKey);
                recordSearcher.searchRecordsByKeyRange(fileName, minKey,
    maxKey);
            }

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

     private void insertAllRecordsIntoMySQL(String tableName,
    FileStructure.Metadata meta, List<FileStructure.Record> records) {
        String url = "{url}";
        String user = "{id}";
        String pass = "{pw}";

        try (Connection conn = DriverManager.getConnection(url, user,
    pass)) {
            StringBuilder sb = new StringBuilder();
            sb.append("INSERT INTO ").append(tableName).append(" (");
            for (int i = 0; i < meta.fieldCount; i++) {
                sb.append(meta.fields.get(i).name);
                if (i < meta.fieldCount - 1) sb.append(",");
            }
            sb.append(") VALUES(");
            for (int i = 0; i < meta.fieldCount; i++) {
                sb.append("?");
```

```
                if (i < meta.fieldCount - 1) sb.append(",");
            }
            sb.append(")");

            String insertSQL = sb.toString();
            System.out.println("Insert SQL: " + insertSQL);

            try (PreparedStatement pstmt =
    conn.prepareStatement(insertSQL)) {
                for (FileStructure.Record rec : records) {
                    for (int i = 0; i < meta.fieldCount; i++) {
                        String val = rec.fieldValues.get(i);
                        if (((rec.nullBitmap >> i) & 1) == 1) {
                            pstmt.setNull(i+1, Types.CHAR);
                        } else {
                            pstmt.setString(i+1, val);
                        }
                    }
                    pstmt.addBatch();
                }
                pstmt.executeBatch();
            }
            System.out.println("All records inserted into MySQL table '"
    + tableName + "'.");

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

    /*
    public static void main(String[] args) {
        String dataFilePath = "D:\SchoolHomework\4-
    1\Database\testdata.txt";
        RecordBulkInserter inserter = new RecordBulkInserter();
        inserter.bulkInsertFromDataFile(dataFilePath);
    }
    */
}
```

MetadataManager.java

```
package main_package;

import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

public class MetadataManager {
      private static final int BLOCK_SIZE = 1024;
      private static final int FIELD_NAME_SIZE = 16;

      // 필드 리스트
```

```
    private List<FileStructure.FieldInfo> fieldsInMemory = new
ArrayList<>();

    public void loadMetadataFromFile(String fileName) {
        String dataFile = fileName + ".dat";
        try (RandomAccessFile raf = new RandomAccessFile(dataFile,
"r")) {
            ByteBuffer buffer = ByteBuffer.allocate(BLOCK_SIZE);
            raf.seek(0);
            raf.readFully(buffer.array());
            buffer.rewind();

            int firstRecordBlock = buffer.getInt();          // 첫 레
코드 블록 번호
            int firstRecordOffset = buffer.getInt();   // 첫 레코드 오
프셋
            int fieldCount = buffer.getInt();                // 필드
개수

            List<FileStructure.FieldInfo> tempFields = new
ArrayList<>();
            for (int i = 0; i < fieldCount; i++) {
                byte[] fixedName = new byte[FIELD_NAME_SIZE];
                buffer.get(fixedName);
                String name = new String(fixedName,
StandardCharsets.UTF_8).trim();
                int length = buffer.getInt();
                tempFields.add(new FileStructure.FieldInfo(name,
length));
            }

            fieldsInMemory = tempFields; // 메모리 구조에 저장
            System.out.println("Metadata is loaded to memory from File
" + fileName + ".dat");
            System.out.println("Field numbers: " + fieldCount);

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

    // 필드 정보 반환
    public List<FileStructure.FieldInfo> getFieldsInMemory() {
        return fieldsInMemory;
    }

    // 파일 헤더에 저장
    public void saveMetadataToFile(String fileName, int
firstRecordBlock, int firstRecordOffset) {
        String dataFile = fileName + ".dat";
        try (RandomAccessFile raf = new RandomAccessFile(dataFile,
"rw")) {
            ByteBuffer buffer = ByteBuffer.allocate(BLOCK_SIZE);

            buffer.putInt(firstRecordBlock);
```

```
                buffer.putInt(firstRecordOffset);
                buffer.putInt(fieldsInMemory.size());

                for (FileStructure.FieldInfo field : fieldsInMemory) {
                    byte[] nameBytes =
    field.name.getBytes(StandardCharsets.UTF_8);
                    byte[] fixedName = new byte[FIELD_NAME_SIZE];
                    System.arraycopy(nameBytes, 0, fixedName, 0,
    Math.min(nameBytes.length, FIELD_NAME_SIZE));
                    buffer.put(fixedName);
                    buffer.putInt(field.length);
                }

                raf.seek(0);
                raf.write(buffer.array());
                System.out.println("File " + fileName + ".dat's header
    information has been updated.");

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

        public void createTableInMySQL(String tableName,
    FileStructure.Metadata meta) {
            String url = "{url}";
            String user = "{id}";
            String pass = "{pw}";

            try (Connection conn = DriverManager.getConnection(url, user,
    pass);
                 Statement stmt = conn.createStatement()) {
                StringBuilder sb = new StringBuilder();
                sb.append("CREATE TABLE IF NOT EXISTS
    ").append(tableName).append(" (");
                for (int i = 0; i < meta.fieldCount; i++) {
                    String colName = meta.fields.get(i).name;
                    int colLen = meta.fields.get(i).length;
                    sb.append(colName).append("
    CHAR(").append(colLen).append(")");
                    if (i < meta.fieldCount - 1) sb.append(", ");
                }
                sb.append(")");

                stmt.executeUpdate(sb.toString());
                System.out.println("MySQL table '" + tableName + "'
    created/verified.");
            } catch (SQLException e) {
                   System.out.println("[ERROR] Failed to create MySQL
    table '" + tableName + "'.");
                System.out.println("Reason: " + e.getMessage());
                e.printStackTrace();
            }
        }
}
```

RecordSearcher.java

```
package main_package;

import java.io.*;
```

```
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
// import java.util.*;

public class RecordSearcher {
       private static final int BLOCK_SIZE = 1024;
       private static final int FIELD_NAME_SIZE = 16;

       // 헤더에서 메타데이터 읽기
       public FileStructure.Metadata readHeader(RandomAccessFile raf)
      throws IOException {
           ByteBuffer buffer = ByteBuffer.allocate(BLOCK_SIZE);
           raf.seek(0);
           raf.readFully(buffer.array());
           buffer.rewind();
           int firstRecordBlock = buffer.getInt();
           int firstRecordOffset = buffer.getInt();
           int fieldCount = buffer.getInt();

           FileStructure.Metadata meta = new FileStructure.Metadata(new
      java.util.ArrayList<>());
           meta.firstRecordBlock = firstRecordBlock;
           meta.firstRecordOffset = firstRecordOffset;
           meta.fieldCount = fieldCount;

           for (int i = 0; i < fieldCount; i++) {
               byte[] fixedName = new byte[FIELD_NAME_SIZE];
               buffer.get(fixedName);
               String name = new String(fixedName,
      StandardCharsets.UTF_8).trim();
               int length = buffer.getInt();
               meta.fields.add(new FileStructure.FieldInfo(name, length));
           }
           return meta;
       }

       // 탐색키 범위 레코드 검색
       public void searchRecordsByKeyRange(String fileName, String minKey,
      String maxKey) {
           String dataFilename = fileName + ".dat";
           try (RandomAccessFile raf = new RandomAccessFile(dataFilename,
      "r")) {
               FileStructure.Metadata meta = readHeader(raf);
               int keyLength = meta.fields.get(0).length;

               int block = meta.firstRecordBlock;
               int offset = meta.firstRecordOffset;

               while (block != -1 && offset != -1) {
                   raf.seek((long) block * BLOCK_SIZE + offset);
                   int nextRecordBlock = raf.readInt();
                   int nextRecordOffset = raf.readInt();
                   byte nullBitmap = raf.readByte();

                   byte[] keyBytes = new byte[keyLength];
                   raf.read(keyBytes);
                   String key = new String(keyBytes,
      StandardCharsets.UTF_8).trim();
```

```
                // System.out.println("keylength = " + key.length());

                // 키가 범위 내에 있으면 출력
                if (key.compareTo(minKey) >= 0 && key.compareTo(maxKey)
      <= 0) {
                    System.out.println("Block:" + block + " Offset:" +
      offset + " Search-key:" + key);

                    int dataOffset = 9 + keyLength; // 포인터+비트맵+길이
                    for (int i = 1; i < meta.fieldCount; i++) { // 0번 키
      제외
                        if (((nullBitmap >> i) & 1) == 0) {
                            raf.seek((long) block * BLOCK_SIZE + offset +
      dataOffset);
                            byte[] fieldBytes = new
      byte[meta.fields.get(i).length];
                            raf.read(fieldBytes);
                            String fieldValue = new String(fieldBytes,
      StandardCharsets.UTF_8).trim();
                            System.out.println(meta.fields.get(i).name +
      ": " + fieldValue);
                            dataOffset += meta.fields.get(i).length;
                        } else {
                            System.out.println(meta.fields.get(i).name +
      ": null");
                        }
                    }
                }

                block = nextRecordBlock;
                offset = nextRecordOffset;
            }

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

    /*
    public static void main(String[] args) {
        RecordSearcher searcher = new RecordSearcher();
        searcher.searchRecordsByKeyRange("f1", "0001", "9999");
    }
    */
}
```

FieldSearcher.java

```
package main_package;

import java.io.*;

public class FieldSearcher {
    private final FileStructure fs = new FileStructure();

    // 모든 레코드 값 검색
    public void searchField(String fileName, String searchField) {
        String dataFilename = fileName + ".dat";
```

```
            try (RandomAccessFile raf = new RandomAccessFile(dataFilename,
        "r")) {
                FileStructure.Metadata meta = fs.readHeader(raf);

                // 인덱스 검색
                int fieldIndex = -1;
                for (int i = 0; i < meta.fieldCount; i++) {
                    if
        (meta.fields.get(i).name.equalsIgnoreCase(searchField)) {
                        fieldIndex = i;
                        break;
                    }
                }
                if (fieldIndex == -1) {
                    System.out.println("Cannot find field '" + searchField +
        "'.");
                    return;
                }

                int block  = meta.firstRecordBlock;
                   int offset = meta.firstRecordOffset;
                   while (block != -1 && offset != -1) {
                    FileStructure.Record rec = fs.readRecord(raf, meta,
        block, offset);
                    String val = rec.fieldValues.get(fieldIndex).trim();
                    System.out.println(
                        "Block:" + block +
                        " Offset:" + offset +
                        " Field " + searchField +
                        " Value: " + (val.isEmpty() ? "null" : val)
                    );

                    block  = rec.nextRecordBlock;
                    offset = rec.nextRecordOffset;
                }

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

        /*
        public static void main(String[] args) {
            FieldSearcher searcher = new FieldSearcher();
            searcher.searchField("f1", "C");
        }
        */
}
```

MainApp.java

```
package main_package;

import java.util.*;
import java.io.*;

public class MainApp {
          public static void main(String[] args) {
              Scanner scanner = new Scanner(System.in);
              FileStructure fileStructure = new FileStructure();
```

```
        RecordBulkInserter inserter = new RecordBulkInserter();
        FieldSearcher fieldSearcher = new FieldSearcher();
        RecordSearcher recordSearcher = new RecordSearcher();
        MetadataManager metadataManager = new MetadataManager();

        while (true) {
            System.out.println("\n==== Database System ====");
            System.out.println("1. Create new data file (from
config .txt)");
            System.out.println("2. Record Insertion (With data
file)");
            System.out.println("3. Search Field");
            System.out.println("4. Search Record Range");
            System.out.println("5. Check Metadata(Memory)");
            System.out.println("6. Exit");
            System.out.println("7. SQL Merge Process");
            System.out.println("8. Validate SQL JOIN vs Merge-Join");
            System.out.print("Selection: ");
            String choice = scanner.nextLine();

            switch (choice) {
                case "1":
                    System.out.print("Enter config text file path to
create .dat: ");
                    String configPath = scanner.nextLine().trim();
                    boolean createSuccess =
createNewDataFileFromConfig(configPath, fileStructure,
metadataManager);
                    if (!createSuccess) {
                        System.out.println("Failed to create data file
due to invalid config.");
                    }
                    break;

                case "2":
                    System.out.print("Enter file name to insert into
(without extension): ");
                    String insertFileName = scanner.nextLine().trim();
                    System.out.print("Record insertion data file path:
");
                    String dataFilePath = scanner.nextLine();
                    inserter.bulkInsertFromDataFile(insertFileName,
dataFilePath);
                    break;

                case "3":
                    System.out.print("File name: ");
                    String fNameField = scanner.nextLine();
                    System.out.print("Search field name: ");
                    String field = scanner.nextLine();
                    fieldSearcher.searchField(fNameField, field);
                    break;

                case "4":
                    System.out.print("File name: ");
                    String fNameRange = scanner.nextLine();
                    System.out.print("Search-key minimum value: ");
                    String minKey = scanner.nextLine();
                    System.out.print("Search-key maximum value: ");
```

```
                    String maxKey = scanner.nextLine();
                    recordSearcher.searchRecordsByKeyRange(fNameRange,
minKey, maxKey);
                    break;

                case "5":
                    System.out.print("Enter file name (without
extension) to check metadata: ");
                    String metaFileName = scanner.nextLine().trim();

metadataManager.loadMetadataFromFile(metaFileName);

                    List<FileStructure.FieldInfo> loadedFields =
metadataManager.getFieldsInMemory();
                    if (loadedFields.isEmpty()) {
                        System.out.println("No metadata loaded or file
not found.");
                    } else {
                        System.out.println("List of fields in memory
for file '" + metaFileName + ".dat':");
                        for (int i = 0; i < loadedFields.size(); i++) {
                            System.out.println("  " + i + ": "
                                + loadedFields.get(i).name + "
(length=" + loadedFields.get(i).length + ")");
                        }
                    }
                    break;

                case "6":
                    System.out.println("Program exited.");
                    scanner.close();
                    return;

                case "7":
                    System.out.print("Enter first table file name: ");
                    String tableR = scanner.nextLine().trim();
                    System.out.print("Enter second table file name:
");
                    String tableS = scanner.nextLine().trim();
                    System.out.print("Enter join key column name: ");
                    String joinKey = scanner.nextLine().trim();
                    System.out.print("Enter output file name
(without .dat): ");
                    String outName = scanner.nextLine().trim();

                    JoinProcessor jp = new JoinProcessor();
                    try {
                        jp.executeMergeJoin(tableR, tableS, joinKey,
outName);
                    } catch (Exception e) {
                        System.out.println("Failed to run merge join: "
+ e.getMessage());
                    }
                    break;

                case "8":
                      System.out.print("Enter full SQL JOIN query: ");
                      String userSql = scanner.nextLine().trim();
```

```
                        System.out.print("Enter first table name (for
    merge algorithm): ");
                        String tblR = scanner.nextLine().trim();
                        System.out.print("Enter second table name: ");
                        String tblS = scanner.nextLine().trim();
                        System.out.print("Enter join key column name:
    ");
                        String jk   = scanner.nextLine().trim();

                        JoinProcessor validator = new JoinProcessor();
                        try {
                            validator.validateWithSqlJoin(userSql, tblR,
    tblS, jk);
                        } catch (Exception e) {
                            System.out.println("Validation failed: " +
    e.getMessage());
                        }
                        break;

                default:
                    System.out.println("Wrong input.");
                    break;
            }
        }
    }

    // 필드 정보 입력
    private static boolean createNewDataFileFromConfig(String
    configPath, FileStructure fileStructure, MetadataManager
    metadataManager) {
        try (BufferedReader br = new BufferedReader(new
    FileReader(configPath))) {
            String line = br.readLine();
            if (line == null || line.trim().isEmpty()) {
                System.out.println("Config file empty or invalid.");
                return false;
            }
            line = line.trim();
            String[] parts = line.split(";");
            if (parts.length < 3) {
                System.out.println("Not enough fields in config.
    Format must be: fileName;fieldCount;fieldName...;fieldLength...");
                return false;
            }

            String fileName = parts[0].trim();
            int fieldCount = Integer.parseInt(parts[1].trim());

            int totalNeeded = 2 + fieldCount + fieldCount;  // "파일
    명,필드개수" + "필드Count" + "길이Count"
            if (parts.length != totalNeeded) {
                System.out.println("Mismatched field count vs actual
    input. Needed " + totalNeeded + " parts, got " + parts.length);
                return false;
            }

            List<String> fieldNames = new ArrayList<>();
            int idx = 2;
```

```
                for (int i = 0; i < fieldCount; i++) {
                    fieldNames.add(parts[idx++].trim());
                }

                List<Integer> fieldLengths = new ArrayList<>();
                for (int i = 0; i < fieldCount; i++) {

      fieldLengths.add(Integer.parseInt(parts[idx++].trim()));
                }

                List<FileStructure.FieldInfo> fields = new ArrayList<>();
                for (int i = 0; i < fieldCount; i++) {
                    fields.add(new
      FileStructure.FieldInfo(fieldNames.get(i), fieldLengths.get(i)));
                }

                String dataFile = fileName + ".dat";
                try (RandomAccessFile raf = new RandomAccessFile(dataFile,
      "rw")) {
                    fileStructure.writeHeader(raf, new
      FileStructure.Metadata(fields));
                    System.out.println("File '" + dataFile + "' created
      with user-defined fields from " + configPath);
                    metadataManager.loadMetadataFromFile(fileName);
                } catch (Exception e) {
                    System.out.println("Error creating file: " +
      e.getMessage());
                    return false;
                }

                return true;

            } catch (IOException e) {
                System.out.println("Could not read config file: " +
      e.getMessage());
                return false;
            } catch (NumberFormatException e) {
                System.out.println("Invalid integer value in config: " +
      e.getMessage());
                return false;
            }
        }
}
```

JoinProcessor.java

```
package main_package;

import java.io.RandomAccessFile;
import java.sql.*;
import java.util.*;
import java.util.stream.Collectors;

public class JoinProcessor {
          private static final String URL  = "{url}";
          private static final String USER = "{id}";
          private static final String PASS = "{pw}";

          private final FileStructure fileStructure = new FileStructure();
```

```
    // 머지 조인 알고리즘
    private List<List<String>> performMergeJoinRows(Connection conn,
String tableR, String tableS, String joinKey) throws SQLException {
        List<List<String>> result = new ArrayList<>();

        try (
            Statement stR = conn.createStatement(
                ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_READ_ONLY);
            Statement stS = conn.createStatement(
                ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_READ_ONLY);
            ResultSet rsR = stR.executeQuery(
                "SELECT * FROM " + tableR + " ORDER BY " + joinKey);
            ResultSet rsS = stS.executeQuery(
                "SELECT * FROM " + tableS + " ORDER BY " + joinKey)
        ) {
            ResultSetMetaData mR = rsR.getMetaData(), mS =
rsS.getMetaData();
            int cR = mR.getColumnCount(), cS = mS.getColumnCount();
            int idxR = -1, idxS = -1;
            for (int i = 1; i <= cR; i++) {
                if (mR.getColumnName(i).equalsIgnoreCase(joinKey))
idxR = i;
            }
            for (int i = 1; i <= cS; i++) {
                if (mS.getColumnName(i).equalsIgnoreCase(joinKey))
idxS = i;
            }
            if (idxR < 0 || idxS < 0) {
                throw new SQLException("Join key not found in both
tables");
            }

            boolean hasR = rsR.next(), hasS = rsS.next();
            while (hasR && hasS) {
                String keyR = rsR.getString(idxR), keyS =
rsS.getString(idxS);

                // NULL은 매칭 대상에서 제외
                if (keyR == null) { hasR = rsR.next(); continue; }
                if (keyS == null) { hasS = rsS.next(); continue; }

                int cmp = keyR.compareTo(keyS);
                if (cmp < 0) {
                    hasR = rsR.next();
                } else if (cmp > 0) {
                    hasS = rsS.next();
                } else {
                    // 같은 키 그룹 버퍼링
                    List<List<String>> bufR = new ArrayList<>();
                    List<List<String>> bufS = new ArrayList<>();
                    String cur = keyR;
                    do {
                        bufR.add(readRow(rsR, cR));
                        hasR = rsR.next();
                    } while (hasR && cur.equals(rsR.getString(idxR)));
                    do {
```

```
                        bufS.add(readRow(rsS, cS));
                        hasS = rsS.next();
                    } while (hasS && cur.equals(rsS.getString(idxS)));

                    // 조인
                    for (List<String> r : bufR) {
                        for (List<String> s : bufS) {
                            List<String> merged = new ArrayList<>(r);
                            for (int j = 0; j < s.size(); j++) {
                              merged.add(s.get(j));
                            }
                            result.add(merged);
                        }
                    }
                }
            }
        }

        return result;
    }

    // 파일에 저장
    public void executeMergeJoin(String tableR, String tableS, String
joinKey, String outTable) throws Exception {
        String datFile = outTable + ".dat";
        Set<String> seenMerged = new HashSet<>();

        try (Connection conn = DriverManager.getConnection(URL, USER,
PASS)) {
            // 테이블 별 헤더 메타데이터 구성
            List<FileStructure.FieldInfo> fields = new ArrayList<>();
            Set<String> seen = new HashSet<>();

            // 테이블 1 메타데이터 조회
            try (
                Statement hdrStR = conn.createStatement();
                ResultSet rs0R   = hdrStR.executeQuery(
                    "SELECT * FROM " + tableR + " ORDER BY " + joinKey
+ " LIMIT 1")
            ) {
                ResultSetMetaData mR = rs0R.getMetaData();
                int cR = mR.getColumnCount();
                for (int i = 1; i <= cR; i++) {
                    String name = mR.getColumnName(i);
                    fields.add(new FileStructure.FieldInfo(name, 32));
                    seen.add(name);
                }
            }

            // 테이블 2 메타데이터 조회
            try (
                Statement hdrStS = conn.createStatement();
                ResultSet rs0S   = hdrStS.executeQuery(
                    "SELECT * FROM " + tableS + " ORDER BY " + joinKey
+ " LIMIT 1")
            ) {
                ResultSetMetaData mS = rs0S.getMetaData();
```

```
                int cS = mS.getColumnCount();
                for (int i = 1; i <= cS; i++) {
                    String name = mS.getColumnName(i);
                    if (!seen.contains(name)) {
                        fields.add(new FileStructure.FieldInfo(name,
32));
                        seen.add(name);
                    }
                }
            }

            FileStructure.Metadata fsMeta = new
FileStructure.Metadata(fields);

            // 파일 헤더 기록
            try (RandomAccessFile raf = new RandomAccessFile(datFile,
"rw")) {
                fileStructure.writeHeader(raf, fsMeta);
            }
            // 실제 머지 조인 결과 받아오기
            List<List<String>> rows = performMergeJoinRows(conn,
tableR, tableS, joinKey);

            // 중복 없이 저장 및 출력
            for (List<String> merged : rows) {
                String keyStr = String.join("|", merged);
                if (seenMerged.add(keyStr)) {
                    System.out.println("Insert record: " + merged);
                    fileStructure.insertRecord(
                        datFile,
                        fsMeta,
                        new FileStructure.Record(merged, (byte)0)
                    );
                }
            }

            System.out.println("Merge join completed. Result saved to
'" + datFile + "'.");
        }
    }

    // 내부 머지와 MySQL 머지 비교
    public void validateWithSqlJoin(String sql, String tableR, String
tableS, String joinKey) throws Exception {
        List<List<String>> sqlRows = new ArrayList<>();
        try (
            Connection conn = DriverManager.getConnection(URL, USER,
PASS);
            Statement st    = conn.createStatement();
            ResultSet rs    = st.executeQuery(sql)
        ) {
            ResultSetMetaData md = rs.getMetaData();
            int sqlCols = md.getColumnCount();

            List<Integer> useIdx = new ArrayList<>();
            for (int i = 1; i <= sqlCols; i++) {
               useIdx.add(i);
```

```
                }

                while (rs.next()) {
                    List<String> row = new ArrayList<>(useIdx.size());
                    for (int idx : useIdx) {
                        String v = rs.getString(idx);
                        row.add(v == null ? "" : v);
                    }
                    sqlRows.add(row);
                }

                // 내부 머지 결과 획득
                List<List<String>> mergeRows = performMergeJoinRows(conn,
    tableR, tableS, joinKey);

                // 결과 비교
                Set<String> sqlSet   = sqlRows.stream()
                                              .map(r -> String.join("|", r))
                                              .collect(Collectors.toSet());
                Set<String> mergeSet = mergeRows.stream()
                                                .map(r -> String.join("|",
    r))
                                                .collect(Collectors.toSet());

                Set<String> onlyInSql   = new HashSet<>(sqlSet);
                onlyInSql.removeAll(mergeSet);
                Set<String> onlyInMerge = new HashSet<>(mergeSet);
                onlyInMerge.removeAll(sqlSet);

                // 결과 출력
                System.out.println("SQL JOIN row count:   " +
    sqlSet.size());
                System.out.println("Merge-Join row count: " +
    mergeSet.size());
                System.out.println("Only in SQL (" + onlyInSql.size() +
    "):");
                onlyInSql.stream().limit(10).forEach(r ->
    System.out.println("   " + r));
                System.out.println("Only in Merge (" + onlyInMerge.size()
    + "):");
                onlyInMerge.stream().limit(10).forEach(r ->
    System.out.println("   " + r));
            }
        }

        // ResultSet 현재 행에서 모든 컬럼 값을 읽어 List<String> 반환
        private List<String> readRow(ResultSet rs, int cols) throws
    SQLException {
            List<String> row = new ArrayList<>(cols);
            for (int i = 1; i <= cols; i++) {
                String v = rs.getString(i);
                row.add(v == null ? "" : v);
            }
            return row;
        }
}
```