INFS3200 Advanced Database Systems

Assignment (25%)

Semester 1, 2023

Student name: Cao Quoc Thang Hoang

Student id: s4759487

Table of Contents

Part 1.

Question 1:

(1)

```
SELECT  SalesRank, Title, Publisher, Pages
FROM Book3
ORDER BY SalesRank DESC
FETCH FIRST 15 ROWS ONLY
UNION
SELECT  SalesRank, Title, Publisher, Pages
FROM Book3
ORDER BY SalesRank DESC
LIMIT 10
-- FETCH FIRST 10 ROWS ONLY
```

(2)

Table 3 should be used to solve this type of question because this relation contains SalesRank attribute which will be used for ranking them and put them in specific order.

Question 2:

(1)

The code of Question 2 should be similar to the code below (if with book1):

```
# Find all books whose publisher name is "XXX" (or among multiple publishers), return
# their book titles and author info

SELECT tile, authors
FROM Book1
WHERE Publisher LIKE '%XXX%'


# Find all books that are published in a given year, return their book IDs,
# languages,number of pages, HardcoverPrice and EbookPrice (in 2018 for instance)
SELECT id, `language`, pages, HardcoverPrice, EbookPrice
FROM Book1
WHERE pubyear = 2018
```

Therefore, the strategy will be used is vertical fragmentation. Two fragmentations are:

BookFrag_1 (id, title, authors)

BookFrag_2 (id, language, pages, HardcoverPrice, EbookPrice, pubyear,pubyear, pubmonth, pubday, edition, ...)

(2)

Assume that we horizontally fragment the table into three fragments based on

the following predicate:

Fragment 1: pages ≤ 200
Fragment 2: 200 < pages ≤ 600
Fragment 3: pages > 800

Answer:

The set of minterm given above is not valid because is not satisfied Completeness (loss tuples in range (600 < pages ≤ 800)). Therefore, the valid predicate set should be:

From given predicates, we have:

P1: page <= 200
~P1: page > 200

P2: 200 < pages <= 600
~P2: page <= 200 && page > 600

P3: page > 800
~P3: page <= 800

Therefore, there are 2^3 = 8 set of predicates (possible minterm predicates):

M1: P1 ^ P2 ^ P3 = empty
M2: P1 ^ P2 ^ ~P3 = empty
M3: P1 ^ ~P2 ^ P3 = empty
M4: P1 ^ ~P2 ^ ~P3 = P1
M5: ~P1 ^ P2 ^ P3 = empty
M6: ~P1 ^ P2 ^ ~P3 = P2
M7: ~P1 ^ ~P2 ^ P3 = P3
M8: ~P1 ^ ~P2 ^ ~P3 = (600 - 800]

Therefore, M4, M6, M7 and M8 will be chosen as the valid set of predicates.

Part2:

Question 3:

Star schema:

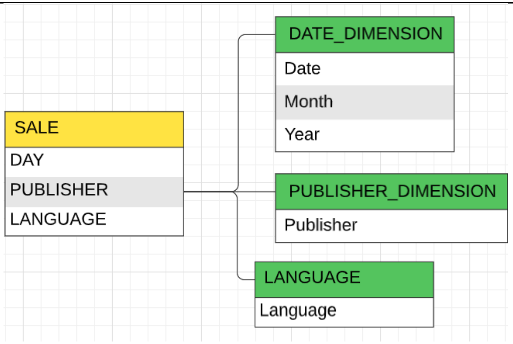


Figure 1: Star schema for Sale

Dimension = 3

DATE_DIMENSION

Date	Month	Year

PUBLISHER_DIMENSION

Publisher

LANGUAGE

Language

Fact table:

SALE
DAY
PUBLISHER
LANGUAGE

Among all dimension tables and fact table, the table that has the most record is SALE fact table. It is because Fact table can contain foreign keys with reference to all the primary keys of Dimension tables.

Question 4:

(1)

The advantages of building bitmap index are:

- It can reduce query time by converting normal comparison between numbers and chars (size depend on data type) to bit operation.
- Reduce storage space by reducing the size of table when storing in memory.
- Increase I/O processing by keeping the fixed part (dictionary part, can be negligible when table size is large) and only change the dynamic part (number of records)

“Day” column is not suitable for using bitmap indexing because “Day” column have too many distinct values which significantly affect the size of bitmap index.

(2)

There are 8 records in the example -> 8 column. Moreover, “Publisher” column only contains four distinct values and “Language” only contains two -> 6 rows (4+2). Therefore, the bitmap can be built as:

AAAI Press	1	0	0	0	1	0	0	0
Springer International Publishing	0	1	0	0	0	1	0	0
Springer London	0	0	1	0	0	0	1	0
IEEE Computer Society Press	0	0	0	1	0	0	0	1
English	1	1	1	1	0	0	0	0
Spanish	0	0	0	0	1	1	1	1

Green cells are for “Publisher” and blue cells are for “Language”

(3)

To find the total sale of “English” books published by “AAAI Press”, we need:

- Scan though “AAAI Press” vector and get records that AAAI Press = 1.
- Scan though “English” vector and get records that English = 1.
- Using AND operation to merge both results.
- Summarize their SALE.

RECORD	R1	R2	R3	R4	R5	R6	R7	R8
AAAI Press	1	0	0	0	1	0	0	0
Springer International Publishing	0	1	0	0	0	1	0	0
Springer London	0	0	1	0	0	0	1	0
IEEE Computer Society Press	0	0	0	1	0	0	0	1
English	1	1	1	1	0	0	0	0
Spanish	0	0	0	0	1	1	1	1
Result	1	0	0	0	0	0	0	0

Green cells are for “Publisher”, blue cells are for “Language”, orange cells are cell that satisfied the condition while scanning the vector

Therefore, the result will be R1 (the first record).

Part 3:

Question 5:

(1)

The global schema is:
Book(id, tile, authors, pub_year, pub_month, pub_day, Publisher, isbn13, pages)

(2)

Structural heterogeneity issue:

- *Type conflicts* such as authors maybe contain a list of all the book authors but in Book3 or Book4, multiple authors are stored in different attributes of type varchar.

- Solution:

One of the possible solutions is creating 3 attributes called “author1”, “author2”, “author3” at global schema. Local schema such as Book2 will need to merge (concatenate) 3 attributes into one attribute by a string format and save it into “authors”. Moreover, Book3 can directly connect their attributes with corresponding ones at global schema. Furthermore, global schema can have the “authors” attribute as same as local schema of Book2 and Book4, whereas other schema will need a data transformation to split that day format into smaller attributes.

However, considering real world problems, for instance:

+ One book can have less or more than 3 authors
+ The number of attribute (or the schema structure in general) should not be altered too frequently (there are 4 authors or more that write 1 book)
+ It will be a bad design to have many redundant NULL values (a book has solely one author so author2 and author3 will contain NULL value).

Therefore, the name “authors” is preferred rather than multiple attributes.

(3)

Semantic heterogeneity issue:

- *Different ontology* such as “publication_day” in Book2 only contain day but that of Book 4 may contain all day, month, and year together.

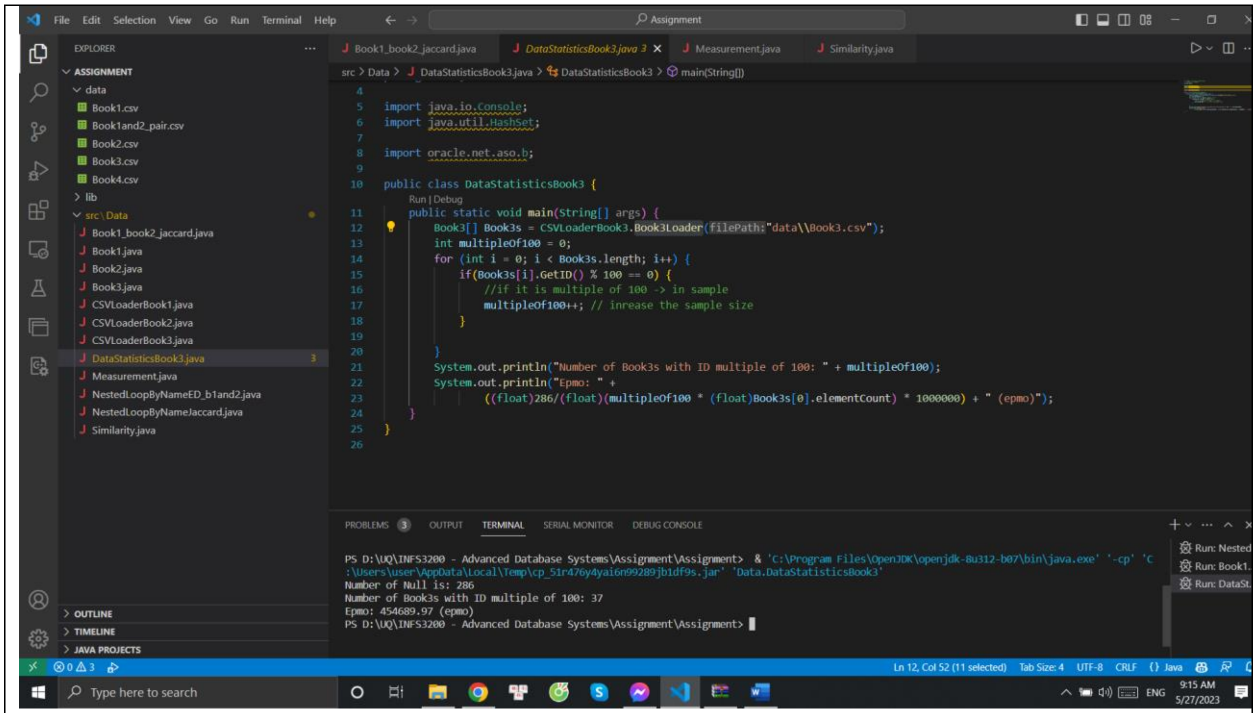
- Solution:

  Similar to the solution above, global schema can have one attribute to store a string of concatenation of day month and year by a specific format or multiple attributes to according to day, month and year separately.

  However, the real world only has days, months and years to specify an exact publication date, thus, the separation of these into multiple attributes seems to be a better option.

Part 4.

Question 6:



```java
import java.io.Console;
import java.util.HashSet;

import oracle.net.aso.b;

public class DataStatisticsBook3 {
    public static void main(String[] args) {
        Book3[] Book3s = CSVLoaderBook3.Book3Loader(filePath:"data\\Book3.csv");
        int multipleOf100 = 0;
        for (int i = 0; i < Book3s.length; i++) {
            if(Book3s[i].GetID() % 100 == 0) {
                //if it is multiple of 100 -> in sample
                multipleOf100++; // inrease the sample size
            }

        }
        System.out.println("Number of Book3s with ID multiple of 100: " + multipleOf100);
        System.out.println("Epmo: " +
                ((float)286/(float)(multipleOf100 * (float)Book3s[0].elementCount) * 1000000) + " (epmo)");
    }
}
```

```
PS D:\UQ\INFS3200 - Advanced Database Systems\Assignment\Assignment>  & 'C:\Program Files\OpenJDK\openjdk-8u312-b07\bin\java.exe' '-cp' 'C
:\Users\user\AppData\Local\Temp\cp_51r476y4yai6n99289jb1df9s.jar' 'Data.DataStatisticsBook3'
Number of Null is: 286
Number of Book3s with ID multiple of 100: 37
Epmo: 454689.97 (epmo)
PS D:\UQ\INFS3200 - Advanced Database Systems\Assignment\Assignment>
```

Code in file: “DataStatisticsBook3.java” and “CSVLoaderBook3.java”

Code in file: “DataStatisticsBook3.java” and “CSVLoaderBook3.java”

(1) There are 37 records that have ID are the multiple of 100.
(2) There are 286 records that have NULL value. ()
(3) Errors per million opportunities:
= ((number of NULL value) / (sample size * number of attributes) *100) * 1,000,000
= ((286 / (37*17)) * 1,000,000 ≈ 454689.97 epmo

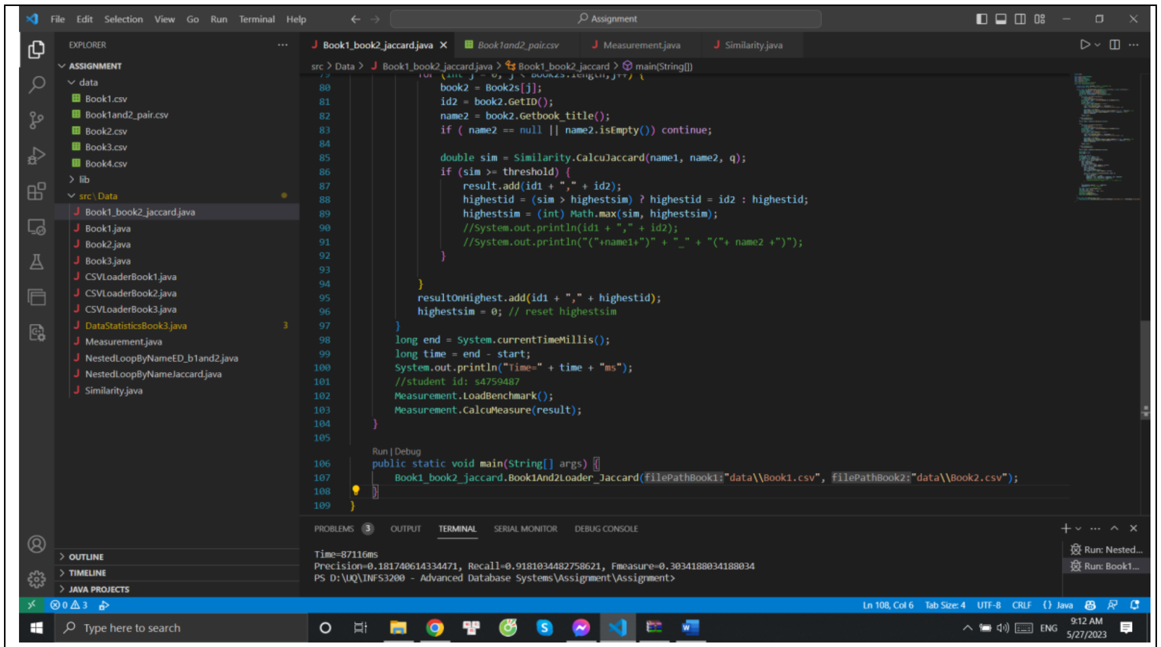
Question 7:

(1)

a. “Richmond Shee, Kirtikumar Deshpande and K. Gopalakrishnan;”
b. “K. Gopalakrishnan, Kirtikumar Deshpande, and Richmond Shee”

Comparing two strings a and b in general we can easily recognize that there are a lot of characters that follow a specific order in the each string such as “Gopalakrishnan”, “Kirtikumar” and “Richmond”. Therefore, *Jaccard distance* would be a better approach thanks to its property of keeping the order of character in each tuple whereas *edit* distance only consider individual character separately. Finally, *Jaccard distance* is more likely to regard them as similar than *edit distance*.

(2)



```java
                book2 = Book2s[j];
                id2 = book2.GetID();
                name2 = book2.Getbook_title();
                if ( name2 == null || name2.isEmpty()) continue;

                double sim = Similarity.CalcuJaccard(name1, name2, q);
                if (sim >= threshold) {
                    result.add(id1 + "," + id2);
                    highestid = (sim > highestsim) ? highestid = id2 : highestid;
                    highestsim = (int) Math.max(sim, highestsim);
                    //System.out.println(id1 + "," + id2);
                    //System.out.println("("+name1+")" + "_" + "("+ name2 +")");
                }

            }
            resultOnHighest.add(id1 + "," + highestid);
            highestsim = 0; // reset highestsim
        }
        long end = System.currentTimeMillis();
        long time = end - start;
        System.out.println("Time=" + time + "ms");
        //student id: s4759487
        Measurement.LoadBenchmark();
        Measurement.CalcuMeasure(result);
    }

    public static void main(String[] args) {
        Book1_book2_jaccard.Book1And2Loader_Jaccard(filePathBook1:"data\\Book1.csv", filePathBook2:"data\\Book2.csv");
    }
}
```

```
Time=87116ms
Precision=0.181740614334471, Recall=0.9181034482758621, Fmeasure=0.3034188034188034
PS D:\UQ\INFS3200 - Advanced Database Systems\Assignment\Assignment>
```

Code in file: “Book1_book2_jaccard.java”

The **precision**, **recall** and **F-measure** are 0.181740614334471 0.9181034482758621 and 0.3034188034188034 respectively.