

Projektaufgabe 2

Phase 1 – Legen der Basis (2.5 P)

Datenmanagement jenseits von Relationen

Gruppen Nummer 8

Weilert Alexander, 12119653

Jovanovic Dragana, 11850325

May 14, 2024

Dieses Reporting Template dient der Vorbereitung der Abgabe von Phase 1.

Datengenerator für Matrizen mit Sparsity (0.5 Punkte)

Zeigen Sie den Code der Funktion `generate()` als Listing oder Screenshot. Gehen Sie (kurz) auf die wesentlichen Aspekte ein.

Die Funktion `generate(int l, double sparsity)` erstellt Tabellen mit der Größe `l` und einem `sparsity`. Beim Start werden die Tabellen A und B gelöscht und neu erstellt. Die View C wird gelöscht, da sie in der Laufzeit generiert wird. Danach wird die Funktion `generateMatrix(int l, double sparsity)` aufgerufen, um Arrays für die Matrizen zu erstellen. Die Funktion vergleicht die `sparsity` mit `Math.random()` und fügt an den entsprechenden Stellen der Matrix entweder eine zufällige Ganzzahl zwischen 1 und 11 oder 0 hinzu. Die Matrix wird entweder mit `'int[l-1][l] matrix'` oder `'int[l][l-1] matrix'` definiert. Danach werden die generierten Matrizen mit `insertMatrix(String tableName, int[][] matrix)` in das DBMS eingefügt.

```
public int[][][] generate(int l, double sparsity) {
    try (Statement statement = this.connection.createStatement()) {
        statement.execute("DROP VIEW IF EXISTS C");
        statement.execute("DROP TABLE IF EXISTS A, B");
        // Create Table
        statement.execute("CREATE TABLE A " +
            "(i INT, j INT, val INT, PRIMARY KEY (i, j))");
        statement.execute("CREATE TABLE B " +
            "(i INT, j INT, val INT, PRIMARY KEY (i, j))");

        int[][][] matrix = new int[2][l][l];
        int[][] matrixA = new int[l-1][l];
        int[][] matrixB = new int[l][l-1];

        System.out.println("--- Matrix A ---");
        matrix[0] = generateMatrix(matrixA, sparsity);
        System.out.println("--- Matrix B ---");
        matrix[1] = generateMatrix(matrixB, sparsity);
    }
```

```

        insertMatrix("A", matrix[0]);
        insertMatrix("B", matrix[1]);

        return matrix;
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

public int[][] generateMatrix(int[][] matrix, double sparsity) {
    Random random = new Random();
    for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix[0].length; j++) {
            if (random.nextDouble() > sparsity) {
                matrix[i][j] = random.nextInt(1, 11);
                // Random value between 0 and 10
            } else {
                matrix[i][j] = 0;
            }
            System.out.print(matrix[i][j] + " ");
        }
        System.out.println();
    }
    return matrix;
}

public void insertMatrix(String tableName, int[][] matrix) {
    try (Statement statement = this.connection.createStatement())
    {
        StringBuilder insertQuery = new StringBuilder(
            "INSERT INTO " + tableName + " VALUES ");
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[i].length; j++) {
                if (matrix[i][j] != 0) {
                    insertQuery.append("(").append(i+1)
                        .append(",").append(j+1).append(",")
                        .append(matrix[i][j]).append("),");
                }
            }
        }
        insertQuery.deleteCharAt(insertQuery.length() - 1);
        statement.executeUpdate(insertQuery.toString());
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}
}

```

Import der Matrizen in das DBMS (0.5 Punkte)

Geben Sie das Create Table Statement für Matrix A an.

```
CREATE TABLE A (i INT, j INT, val INT, PRIMARY KEY (i, j))
```

Die Ansicht der Primary Keys ist bei uns nicht vorhanden, da in Version 1 die Tabellen keine Primary Keys hatten. Die Tabellen haben in der neueren Version Primary Keys wie im unteren Bild dargestellt. Die kommenden Screenshots wurden nicht mehr ersetzt, da wir keinen erneuten Aufwand betreiben wollten die Rechnungen per Hand nochmals

durchzuführen, weil wir nur beweisen mussten das unser System die Berechnung richtig durchführt.

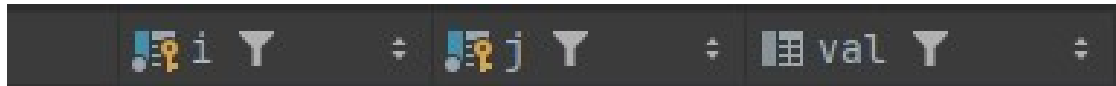


Figure 1: Attributnamen mit Keys

Für die Übung bereiten Sie eine Demo des Datenimports vor.

Da der Import der Daten für die Matrix automatisch und zufällig passiert, muss mit den Daten gerechnet werden, die generiert werden. Mit der folgenden Seite: *Matrix Rechner*, können Sie sich die Matrixmultiplikation berechnen lassen.

Wahl des Toy Beispiels (0.5 Punkte)

Geben Sie Matrix A und B als 2D Array an.

0	0	9	9	8
0	1	3	2	0
0	0	5	2	9
0	0	0	0	0

Figure 2: 2D Ansicht der Tabelle A

8	5	0	3
3	5	0	8
1	0	7	0
0	10	0	0
0	9	0	3

Figure 3: 2D Ansicht der Tabelle B

Zeigen Sie die äquivalente Darstellung der Matrix A und B in der Datenbank.

	i	j	val
1	1	3	9
2	1	4	5
3	1	5	8
4	2	2	1
5	2	3	3
6	2	4	2
7	3	3	5
8	3	4	2
9	3	5	9

Figure 4: Tabelle A im DBMS

	i	j	val
1	1	1	8
2	1	2	5
3	1	4	3
4	2	1	3
5	2	2	5
6	2	4	8
7	3	1	1
8	3	3	7
9	4	2	10
10	5	2	9
11	5	4	3

Figure 5: Tabelle B im DBMS

Implementierung von Ansatz 0 (0.5 Punkte)

Zeigen Sie den Code der Matrixmultiplikation als Listing oder Screenshot. Erläutern Sie (kurz), welche Laufzeit ihr Algorithmus hat und warum das Kriterium keinen Algorithmus mit sub-kubischer Laufzeit zu wählen erfüllt ist.

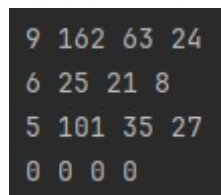
```
public void ansatz0(int [][] matrixA, int [][] matrixB) {
    try (Statement statement = this.connection.createStatement())
```

```

{
    statement.execute("DROP TABLE IF EXISTS matrix_algorithm");
    statement.execute("CREATE TABLE
        matrix_algorithm (i INT, j INT, val INT, PRIMARY KEY (i, j))");
    StringBuilder insertQuery = new StringBuilder(
        "INSERT INTO matrix_algorithm VALUES ");
    int[][] result = new int[matrixA.length][matrixB[0].length];
    System.out.println("--- Matrix Calculator ---");
    for (int i = 0; i < matrixA.length; i++) {
        for (int j = 0; j < matrixB[0].length; j++) {
            for (int k = 0; k < matrixA[0].length; k++) {
                result[i][j] += matrixA[i][k] * matrixB[k][j];
            }
            insertQuery.append("(").append(i+1).append(",")
                .append(j+1).append(",").append(result[i][j]).append("),");
            System.out.print(result[i][j] + " ");
        }
        System.out.println();
    }
    insertQuery.deleteCharAt(insertQuery.length() - 1);
    statement.executeUpdate(insertQuery.toString());
} catch (SQLException e) {
    throw new RuntimeException(e);
}
}

```

Ein sub-kubischer Algorithmus bewegt sich in der Laufzeit kleiner $O(n^3)$. Unser Algorithmus hat im worst case eine Laufzeit von $O(n^3)$ und ist somit kubisch. [1], Ein kubischer Algorithmus muss die Bedingung erfüllen in der Laufzeit $O(n^3)$ zu sein. [2], Dies ist bei uns der Fall, da dieser alle Spalten und Zeilen, der drei verschiedenen for-Schleifen durchläuft, die jeweils die länge n besitzen.



9	162	63	24
6	25	21	8
5	101	35	27
0	0	0	0

Figure 6: Ergebnis C der Matrixmultiplikation anhand des Algorithmus

Implementierung von Ansatz 1 (0.5 Punkte)

Berechnen Sie von Hand das Ergebnis $C = A \times B$ für ihr Toy Beispiel und geben Sie es nachfolgend an.

```

public void ansatz1() {
    try (Statement statement = this.connection.createStatement()) {
        statement.execute("CREATE VIEW C AS " +
            "SELECT a.i, b.j, SUM(A.val * B.val) " +
            "FROM a, b " +
            "WHERE a.j = b.i " +
            "GROUP BY a.i, b.j");

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
}

```


References

Important: Reference your information sources!

Remove this section if you use footnotes to reference your information sources.

References

- [1] *Computational complexity of matrix multiplication*. https://en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication. 12.05.2024. Wikipedia, 2024.
- [2] Prof. i. R. Dr. Thomas Letschert. *Algorithmen und Datenstrukturen CS1017*. https://homepages.thm.de/~hg51/Veranstaltungen/A_D/Folien/ad-04.pdf. 12.05.2024.