

Projektaufgabe 2

Phase 1 – Legen der Basis (2.5 P)

Datenmanagement jenseits von Relationen

Gruppen Nummer 8

Weilert Alexander, 12119653

Jovanovic Dragana, 11850325

May 12, 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 werden die Funktionen `generateMatrixA(int l, double sparsity)` und `generateMatrixB(int l, double sparsity)` aufgerufen, um Arrays für die Matrizen zu erstellen. Beide Funktionen vergleichen die `sparsity` mit `Math.random()` und fügen an den entsprechenden Stellen der Matrix entweder eine zufällige Ganzzahl zwischen 1 und 11 oder 0 hinzu. Die Unterschiede zwischen den Funktionen liegen nur im Array-Aufbau. Die Matrix wird entweder mit `'int[l-1][l] matrix'` oder `'int[l][l-1] matrix'` definiert, ebenfalls wird dies in der Schleifenbedingung jeweils verkehrt aufgerufen. Danach werden die generierten Matrizen mit `insertMatrix(String tableName, int[][] matrix)` in das DBMS eingefügt. Die Funktionen `ansatz0(int[][] matrixA, int[][] matrixB)` und `ansatz1()` berechnen die Matrixmultiplikation für C auf unterschiedliche Weise, entweder per Algorithmus oder durch eine Selektion auf A und B.

```
public void 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[][] matrixA = generateMatrixA(l, sparsity);
        int[][] matrixB = generateMatrixB(l, sparsity);
        insertMatrix("A", matrixA);
        insertMatrix("B", matrixB);
    }
```

```

        ansatz0(matrixA, matrixB); // Matrix Calculator per Algorithm
        ansatz1();                // Matrix Calculator per Select

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

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

public int[][] generateMatrixB(int l, double sparsity) {
    Random random = new Random();
    int[][] matrixB = new int[l][l-1];
    System.out.println("--- Matrix B ---");
    for (int i = 0; i < l; i++) {
        for (int j = 0; j < (l - 1); j++) {
            if (random.nextDouble() > sparsity) {
                matrixB[i][j] = random.nextInt(1, 11);
            } else {
                matrixB[i][j] = 0;
            }
            System.out.print(matrixB[i][j] + " ");
        }
        System.out.println();
    }
    return matrixB;
}

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.

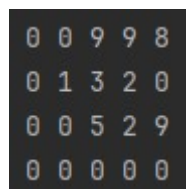
```
CREATE TABLE A (i INT, j INT, val INT, PRIMARY KEY (i, j))
```

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.

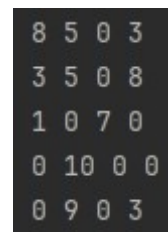
Wahl des Toy Beispiels (0.5 Punkte)

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

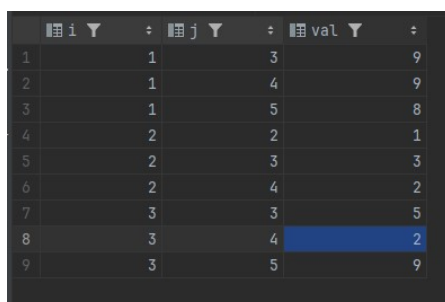
Figure 1: 2D Ansicht der Tabelle A



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

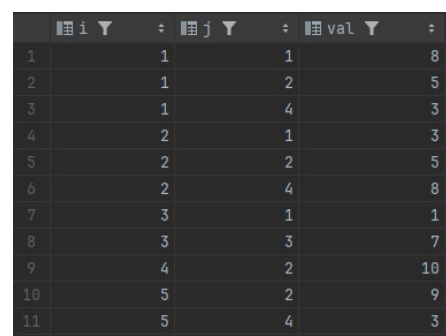
Figure 2: 2D Ansicht der Tabelle B

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



	i	j	val
1	1	3	9
2	1	4	9
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 3: 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

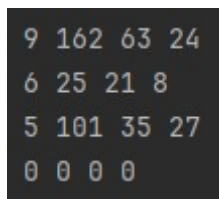
Figure 4: Tabelle B im DBMS

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 5: 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()) {
```

$$\begin{matrix} A & B \\ \begin{pmatrix} 0 & 0 & 9 & 9 & 8 \\ 0 & 1 & 3 & 2 & 0 \\ 0 & 0 & 5 & 2 & 9 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} & \begin{pmatrix} 8 & 5 & 0 & 3 \\ 3 & 5 & 0 & 8 \\ 1 & 0 & 7 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 9 & 0 & 3 \end{pmatrix} \end{matrix}$$

$$= \begin{pmatrix} 0 \cdot 8 + 0 \cdot 3 + 9 \cdot 1 + 9 \cdot 0 + 8 \cdot 0 & 0 \cdot 5 + 0 \cdot 5 + 9 \cdot 0 + 9 \cdot 10 + 8 \cdot 9 & 0 \cdot 0 + 0 \cdot 0 + 9 \cdot 7 + 9 \cdot 0 + 8 \cdot 0 & 0 \cdot 3 + 0 \cdot 8 + 9 \cdot 0 + 9 \cdot 0 + 8 \cdot 3 \\ 0 \cdot 8 + 1 \cdot 3 + 3 \cdot 1 + 2 \cdot 0 + 0 \cdot 0 & 0 \cdot 5 + 1 \cdot 5 + 3 \cdot 0 + 2 \cdot 10 + 0 \cdot 9 & 0 \cdot 0 + 1 \cdot 0 + 3 \cdot 7 + 2 \cdot 0 + 0 \cdot 0 & 0 \cdot 3 + 1 \cdot 8 + 3 \cdot 0 + 2 \cdot 0 + 0 \cdot 3 \\ 0 \cdot 8 + 0 \cdot 3 + 5 \cdot 1 + 2 \cdot 0 + 9 \cdot 0 & 0 \cdot 5 + 0 \cdot 5 + 5 \cdot 0 + 2 \cdot 10 + 9 \cdot 9 & 0 \cdot 0 + 0 \cdot 0 + 5 \cdot 7 + 2 \cdot 0 + 9 \cdot 0 & 0 \cdot 3 + 0 \cdot 8 + 5 \cdot 0 + 2 \cdot 0 + 9 \cdot 3 \\ 0 \cdot 8 + 0 \cdot 3 + 0 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 & 0 \cdot 5 + 0 \cdot 5 + 0 \cdot 0 + 0 \cdot 10 + 0 \cdot 9 & 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 7 + 0 \cdot 0 + 0 \cdot 0 & 0 \cdot 3 + 0 \cdot 8 + 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 3 \end{pmatrix}$$

$$= \begin{pmatrix} 9 & 162 & 63 & 24 \\ 6 & 25 & 21 & 8 \\ 5 & 101 & 35 & 27 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 6: Die von Hand berechnete Matrixmultiplikation

```

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);
}
}

```

Zeigen Sie, dass Ihr System *C* korrekt berechnet (z.B. als Screenshot)

Wir nehmen an, dass das von Hand berechnete Ergebnis richtig bestimmt worden ist. Vergleichen wir nun das von Hand berechnete, den Ansatz0 und den Ansatz1 so sehen wir eine Übereinstimmung aller Views, weswegen wir nun davon ausgehen können, dass der Ansatz1 die Summe richtig berechnet.

i	j	sum
1	1	9
2	1	162
3	1	63
4	1	24
5	2	6
6	2	25
7	2	21
8	2	8
9	3	5
10	3	101
11	3	35
12	3	27

Figure 7: Vom System Berechnete Matrixmultiplikation über die Selektion aus Ansatz1

Zeitmanagement

Benötigte Zeit pro Person (nur Phase 1):

Alexander Weilert: 6h

Dragana Jovanovic: 6h

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.