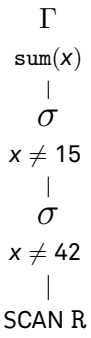
3. Übungsblatt

Ausgabe: 21. Oktober 2019 · Besprechung: 28. Oktober / 04. November 2019

Einleitung

Für dieses Übungsblatt stellen wir Ihnen das System `WeeDB` zur Verfügung. Es handelt sich um die Minimalvariante eines in-memory Datenbanksystems und eignet sich zur Untersuchung von Ausführungsmodellen. Der gegebene Code enthält die Operatoren Scan, Selektion und Aggregation für die Verarbeitungsmodelle *Tuple-at-a-time* und *Operator-at-a-time*.

Um Anfragen auszuführen werden Anfragepläne zunächst als Hierarchie von relationalen Operatoren spezifiziert. Ein vollwertiges System würde diese Pläne mit Hilfe eines SQL-Parsers und des Optimierers generieren. Der nachfolgende Plan liest die Relation `R`, filtert Elemente mit den Werten 15 und 42 und summiert die übrigen Elemente.




```
//build query plan
RelOperator* root;
root = new AggregationOp ( AggOp::SUM,
   new SelectionOp ( PredType::EQUALS_NOT, 15,
        new SelectionOp ( PredType::EQUALS_NOT, 42,
            new ScanOp ( R )
        )
    )
);
```

Die Operatoren unterstützen das `open()`, `next()`, `close()` Interface für Tuple-at-a-time-Verarbeitung und ein `getRelation()` Interface für Operator-at-a-time-Verarbeitung. Anfragepläne können damit folgendermaßen ausgeführt werden:

```
//execute tuple-at-a-time
root->open();
Tuple* t = root->next();
while ( t != nullptr) {
    t = root->next();
    print ( t );
}
root->close();
```

```
//execute operator-at-a-time
Relation resultRelation =
    root->get();
```

Aufgabe — Analyse (Besprechung: 28. Oktober)

Führen Sie unterschiedliche Anfragen mit Tuple-at-a-time und Operator-at-a-time aus. Nutzen Sie die gegebenen Anfragen und formulieren Sie auch selbst Anfragen. Analysieren Sie das Laufzeitverhalten und die Nutzung der Systemresourcen CPU, CPU-Caches und Hauptspeicher(-bus) und beantworten Sie die folgenden Fragen:

- Wie stark beanspruchen die Verarbeitungsmodellen welche Systemresourcen?
- Welche Änderungen des Resourcenbedarfs entstehen wenn Sie die Anzahl der Operatoren variieren? Entspricht die Änderung Ihrer Erwartung? Führen Sie Überschlagsrechnungen durch.

Die folgenden Metriken können bei der Untersuchung nützlich sein:

- Die Anzahl der *Speicherzugriffe*, z.B. Cache-Lines die zwischen Hautspeicher und L3 Cache bewegt werden,
- die Anzahl ausgeführter *Instruktionen*,
- und die Anzahl der *Cache-Hits* und *Cache-Misses*.

Um einzuschätzen, ob eine Metrik die untersuchten Effekte geeignet wiederspiegelt können wiederum Überschlagsrechnungen nützlich sein. Für eine Relation aus 200 M Tupeln mit je 8 Byte ergibt sich beispielsweise ein Datenvolumen von 1.6 GB.

Aufgabe — Vector-at-a-time (Besprechung: 04. November)

Das Verarbeitungsmodell *Vector-at-a-time* zielt darauf ab die Effizienz von Operator-at-a-time durch eine bessere Nutzung der CPU-Caches zu steigern. Ähnlich wie bei Tuple-at-a-time (Volcano) werden Daten der Relationen gelesen und ohne weiteren I/O von Operator zu Operator weiter gereicht. Anstelle von einzelnen Tupeln reichen Operatoren jeweils Blöcke (Vektoren) von Tupeln an den nächsten Operator weiter. Bei einer passenden Blockgröße bleiben die Daten dadurch im CPU-Cache. Vector-at-a-time kann somit als Kombination der beiden Verarbeitungsmodelle Operator-at-a-time und Tuple-at-a-time gesehen werden. Dies spiegelt sich auch in den Interfaces der Verarbeitungsmodelle wieder:

```
//volcano                  //vector-at-a-time                 //op.-at-a-time
void open();       ⟶      void openVec();          ⟵       Relation getRel();
Tuple* next();             Relation* nextVec();
void close();              void closeVec();
```

Erweitern Sie `WeeDB` um das Verarbeitungsmodell *Vector-at-a-time*. Implementieren Sie dazu in den Dateien `BaseOperator.h`, `Operators.h`, und `OperatorsVector.cpp` das vector-at-a-time Interface für die Operatoren des DBMS. Sie können sich dazu an den Implementationen der anderen Verarbeitungsmodelle orientieren.

Aufgabe – Evaluation (Besprechung 04. November)

Evaluieren Sie Ihre Implementierung des Verarbeitungsmodells Vector-at-a-time. Erweitern Sie dazu die vorangegangene Analyse um Vector-at-a-time und schätzen Sie auch die erwarteten Ergebnisse ab. Beantworten Sie durch Experimente die folgenden Fragen:

- Welchen Einfluss hat die Blockgröße auf die Ausführung von Vector-at-a-time?
- Kann Vector-at-a-time die Cache-Effizienz gegenüber Operator-at-a-time steigern?