TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe25

Alice Rey, Maximilian Reif, Tobias Goetz (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss25/impldb/

Blatt Nr. 10

Hinweise Die Aufgaben können auf `http://xquery.db.in.tum.de/` getestet werden. Die Daten für das Unischema können mit doc('uni2') geladen werden. Zur Lösung der Aufgaben können Sie die folgenden XQuery-Funktionen verwenden:

max(NUM), count(X), tokenize(STR,SEP), sum(NUM), contains(HAY,NEEDLE)

1. max(NUMBERS) - Returns largest number from list
2. count(LIST) - Return the number of elements in the list
3. tokenize(STR,SEP) - Splits up the string at the seperator
4. sum(NUMBERS) - Returns sum of all numbers in list
5. contains(HAY,NEEDLE) - Checks if the search string (NEEDLE) is contained in the string (HAY)
6. distinct-values(LIST) - Returns the distinct values from the list

Hausaufgabe 1

Sie sollen für die Alexander-Maximilians-Universität (AMU) ein Hauptspeicherdatenbanksystem optimieren. In dem System sind die Daten aller Studenten gespeichert. Schätzen Sie für jede der untenstehenden Anfragen einzeln, ob ein Row- oder Column-Store besser geeignet ist.

Relationen

Studenten: MatrNr (8 Byte), Name (48 Byte), Studiengang (4 Byte), Semester (4 Byte)

MatrNr ist der Primärschlüssel der indiziert ist.

Anfragen:

1. select * from Studenten;
2. select Semester, count(*) from Studenten group by Semester;
3. select Name, Studiengang, Semester from Studenten where MatrNr = 42;
4. select Studiengang from Studenten where MatrNr = 42;
5. select * from Studenten where Semester < 5;
6. select * from Studenten where Semester = 25;
7. insert into studenten values(4242, Max Meyer, Info, 7);

Lösung:

1. Beides optimal: Gesamte Tabelle wird gelesen
2. Column-Store: Nur Spalte Semester wird gelesen. Bei Row Store muss die gesamte Tabelle gelesen werden.
3. Row-Store: Zeile passt in eine Cache Line (CL). Bei Column Store müssen mindestens 3 CLs gelesen werden.
4. Theoretisch Column-Store: Nur Studiengang muss gelesen werden, da ein Index existiert. Row-Store aber in der Realität gleich schnell, da Daten immer CL granular gelesen werden.
5. Beides optimal: Es muss die gesamte Tabelle gelesen werden, da das Prädikat nicht sehr selektiv ist.
6. Column-Store: Prädikat ist hoch selektiv. Es wird wahrscheinlich kein Student gefunden. Demnach müssen die anderen Felder nicht nachgeladen werden.
7. Row-Store: Zeile passt in eine CL. Damit muss beim Einfügen nur eine CL in den Speicher geschrieben werden. Bei Column Store müsste jedes Element einzeln geschrieben werden.

Hausaufgabe 2

In (pseudo) C++ kann eine 'Row-Store-artige' Datenstruktur wie folgt angelegt werden:

```
struct Tuple {
  int MatrNr;
  RuntimeString Name;
  int Semester;
}
Tuple data[10000] = {};
```

Notieren Sie, wie die Daten in Form eines Column Stores gehalten werden können in (pseudo) C++.

Erklären Sie Ihrem Tutor, welche Vor- und Nachteile Row- und Column Stores jeweils haben. Was würden Sie für Amazons Webseite verwenden? Was verwenden Sie für die Controlling Datenbank?

```
int MatrNrs[10000] = {};
RuntimeString Names[10000] = {};
int Semesters[10000] = {};
```

Row Store: Besser, wenn tendenziell viele Attribute des Tupels benutzt werden. Schlecht, wenn nur auf einen Bruchteil des Tupel zugegriffen wird, da viel mehr Daten geladen werden müssen und die Lokalität in der gesamten Speicherhierarchie dann schlechter ist.
Column Store defakto umgekehrt.
Im Schnitt verwendet man heute Row Stores für transaktionale Daten, Column Stores für analytische Daten. Hiervon kann abgewichen werden. Zu bedenken ist, welche Probleme entstehen können, wenn die Anwendungslogik nicht sinnvoll mit dem Datenbanksystem umgeht, beispielsweise weil immer alle Daten (`SELECT * FROM`) und nicht nur die benötigten ausgelesen werden.

Hausaufgabe 3

Gegeben eine Tabelle *Produkte* mit folgendem Schema und 10000 Einträgen:

Id (8 Byte) | Name (32 Byte) | Preis (8 Byte) | Anzahl (8 Byte)

Wieviele Daten werden für folgende Queries in die CPU-Caches geladen? Unterscheiden sie jeweils zwischen Row und Column Store.

1. *select * from Produkte*
2. *select Anzahl from Produkte*

Daten können maximal mit Granularität (64 Byte) in den Cache geladen werden. Das heißt, selbst wenn nur auf einen 64 Bit Integer Wert zugegriffen wird, muss ein kompletter 64-Byte Block geladen werden. Mit diesem Hintegrund ergeben sich folgende Ergebnisse:

1. *select * from Produkte*
   a) Row: $10000 * 56 = 560000$ Byte
   b) Column: $10000 * 8 + 10000 * 32 + 10000 * 8 + 10000 * 8 = 560000$ Byte
2. *select Anzahl from Produkte*
   a) Row: $10000 * 56 = 560000$ Byte
   b) Column: $10000 * 8 = 80000$ Byte

Hausaufgabe 4

Formulieren Sie die zuvor in SQL bearbeiteten Anfragen zur Universitätsdatenbank in XQuery. Erstellen Sie insbesondere XQuery-Anfragen, um folgende Fragestellungen zu beantworten:

a) Suchen Sie die Professoren, die Vorlesungen halten.

```
doc('uni2')//ProfessorIn[.//Vorlesung]/Name
```

b) Finden Sie die Studenten, die alle Vorlesungen gehört haben.

```
doc('uni2')//Student[count(tokenize(hoert/@Vorlesungen," "))=
count(//Vorlesung)]/Name
```

c) Finden Sie die Studenten mit der größten Semesterzahl unter Verwendung von Aggregatfunktionen.

```
let $maxsws:=max(doc('uni2')//Student/Semester)
return doc('uni2')//Student[Semester=$maxsws]
(: alternativ als Einzeiler :)
return doc('uni2')//Student[Semester=max(//Student/Semester))]
```

d) Berechnen Sie die Gesamtzahl der Semesterwochenstunden, die die einzelnen Professoren erbringen. Dabei sollen auch die Professoren berücksichtigt werden, die keine Vorlesungen halten.

```
for $p in doc('uni2')//ProfessorIn
return <Prof>{$p/Name}<Summe>{sum($p//SWS)}</Summe></Prof>
```

e) Finden Sie die Studenten, die alle vierstündigen Vorlesungen gehört haben.

```
let $fourcount:=count(doc('uni2')//Vorlesung[SWS=4])
for $s in doc('uni2')//Student
where count(
  for $h in tokenize($s/hoert/@Vorlesungen," ")
  where doc('uni2')//Vorlesung[@VorlNr=$h and SWS=4]
  return $h
) = $fourcount
return $s/Name
```

f) Finden Sie die Namen der Studenten, die in keiner Prüfung eine bessere Note als 3.0 hatten.

```
for $s in doc('uni')//Student
where count(
  for $p in $s//Pruefung
  where $p/@Note < 3
  return $p
) = 0
return $s
```

g) Berechnen Sie den Umfang des Prüfungsstoffes jedes Studenten. Es sollen der Name des Studenten und die Summe der Semesterwochenstunden der Prüfungsvorlesungen ausgegeben werden.

```
for $s in doc('uni')//Student
return <Student>{$s/Name}<sum>{
  sum(for $p in $s//Pruefung
      return doc('uni')//Vorlesung[@VorlNr=$p/@Vorlesung]/SWS)}
</sum></Student>
```

h) Finden Sie Studenten, deren Namen den eines Professors enthalten.

```
for $s in doc('uni')//Student
where doc('uni')//ProfessorIn[contains($s/Name,Name)]
return $s/Name
```

i) Ermitteln Sie den Bekanntheitsgrad der Professoren unter den Studenten, wobei wir annehmen, dass Studenten die Professoren nur durch Vorlesungen oder Prüfungen kennen lernen.

```
for $p in doc('uni')//ProfessorIn
return
  <Professor>
    {$p/Name}
    <Bekanntheit>
    {
      count(
        doc('uni')//Student[.//Pruefung[@Pruefer=$p/@PersNr]]/Name
        union
        ( for $v in $p//Vorlesung/@VorlNr
          return doc('uni')//Student[contains(hoert/@Vorlesungen,$v)]/Name)
      )
    }
    </Bekanntheit>
  </Professor>
```

Hausaufgabe 5

Schreiben Sie eine Anfrage, die folgendes Ergebnis zurückgibt:

```
<Universitaet>
  <Fakultaet Name="Philosophie" AnzahlAssistenten="3">
    <Professor Name="Sokrates" AnzahlAssistenten="2"/>
    <Professor Name="Russel" AnzahlAssistenten="1"/>
  </Fakultaet>
  <Fakultaet Name="Physik" AnzahlAssistenten="2">
    <Professor Name="Kopernikus" AnzahlAssistenten="2"/>
  </Fakultaet>
  <Fakultaet Name="Theologie" AnzahlAssistenten="1">
    <Professor Name="Augustinus" AnzahlAssistenten="1"/>
  </Fakultaet>
</Universitaet>
```

```
<Universitaet>
{for $f in doc('uni')//Fakultaet
    let $fa := count($f//Assistent)
    order by $fa descending
       return <Fakultaet Name="{$f/FakName}" AnzahlAssistenten="{$fa}">{
          for $p in $f//ProfessorIn
          let $pa := count($p//Assistent)
          where $pa > 0
          order by $pa descending
       return <Professor Name="{$p/Name}" AnzahlAssistenten="{$pa}" />
    }</Fakultaet>
}
</Universitaet>
```

Hausaufgabe 6

Beantworten Sie folgende Anfragen mithilfe von JSONpath über das nachfolgende JSON-Dokument. Zum Auswerten der Anfragen können Sie folgende Schnittstelle verwenden: `https://jsonpath.com/`

```
{
  "shop": {
    "name": "Tech␣&␣More",
    "categories": [
      {
        "name": "Laptops",
        "products": [
          { "id": 101, "name": "Laptop␣A", "price": 899, "tags": ["ultrabook", "ssd"] },
          { "id": 102, "name": "Laptop␣B", "price": 1299, "tags": ["gaming", "16GB"] }
        ]
      },
      {
        "name": "Phones",
        "products": [
          { "id": 201, "name": "Phone␣X", "price": 699, "tags": ["5G", "oled"] },
          { "id": 202, "name": "Phone␣Y", "price": 499, "tags": ["budget", "4G"] },
          { "id": 203, "name": "Phone␣Z", "price": 999, "tags": ["flagship", "5G"] }
        ]
```

```
      }
    ]
  }
}
```

1. Geben Sie den Namen aller Produkte aus

   `$..products[*].name`

2. Geben Sie die ersten zwei Produkte aus der Kategorie „Phones“ aus

   `$.shop.categories[?(@.name=='Phones')].products[0:2]`

3. Geben Sie alle Preise aus

   `$..price`

4. Geben Sie alle Produkt-Tags aus von Produkten mit einem Preis < 1000€

   `$..products[?(@.price < 1000)].tags[*]`

5. Geben Sie alle Produkte aus mit einem 5G Tag

   `$..products[?(@.tags[?(@ == '5G')])].name`