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

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe24

Alice Rey, Maximilian Bandle, Michael Jungmair (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss24/impldb/

Blatt Nr. 11

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

Führen Sie die folgenden Abfragen in der Spark-Shell aus. Als Grundlage für die Abfragen dient das TPC-H Schema. Laden Sie dazu die TPC-H Daten wie in der Vorlesung gezeigt in die Spark-Shell.

(a) Ermitteln Sie pro Marktsegment die Anzahl der Bestellungen in 1997.

(b) Ermitteln Sie die Zahl der Kunden und Lieferanten pro Land.

(c) Ermitteln Sie die Stückzahlen der verschiedenen Bauteile in Deutschland.

(d) Ermitteln Sie, welche Kunden kein *goldenrod lavender spring chocolate lace* bestellt haben.

Lösung:

(a) Ermitteln Sie pro Marktsegment die Anzahl der Bestellungen in 1997.

```
val ordersOf1997 = orders.where(year($"o_orderdate") === 1997)
val customerOrders = ordersOf1997.join(customer, $"c_custkey" === $"o_custkey")
val mktSegmentOrders = customerOrders
   .groupBy($"c_mktsegment")
   .agg(count($"o_orderkey").as("orderCount"))

mktSegmentOrders.show
```

(b) Ermitteln Sie die Zahl der Kunden und Lieferanten pro Land.

```
val countryCustomer = customer.groupBy($"c_nationkey")
   .agg(count($"c_custkey").as("customerCount"))
val countrySupplier = supplier.groupBy($"s_nationkey")
   .agg(count($"s_suppkey").as("supplierCount"))
val countryCustSupp = nation
   .join(countryCustomer, $"c_nationkey" === $"n_nationkey")
   .join(countrySupplier, $"s_nationkey" === $"n_nationkey")

countryCustSupp
   .select($"n_nationkey", $"n_name", $"customerCount", $"supplierCount")
   .show
```

(c) Ermitteln Sie die Stückzahlen der verschiedenen Bauteile in Deutschland.

```
val germany = nation.filter($"n_name" === "GERMANY")
val germanSupplier = supplier
   .join(germany, $"n_nationkey" === $"s_nationkey", "leftsemi")
val germanParts = partsupp
   .join(germanSupplier, $"ps_suppkey" === $"s_suppkey","leftsemi")
   .groupBy($"ps_partkey")
   .agg(sum($"ps_availqty").as("stueckzahl"))

germanParts.show
```

(d) Ermitteln Sie, welche Kunden kein *goldenrod lavender spring chocolate lace* bestellt haben.

```
val goldenRodLavenderSpringChoclateLaceOrders = part
   .filter($"p_name" === "goldenrod␣lavender␣spring␣chocolate␣lace")
   .join(partsupp, $"ps_partkey" === $"p_partkey")
   .join(lineitem, $"l_partkey" === $"ps_partkey")
   .join(orders, $"o_orderkey" === $"l_orderkey")
   .select($"o_custkey")
   .distinct

val noChocolateCustomer = customer.join(
   goldenRodLavenderSpringChoclateLaceOrders,
   $"o_custkey" === $"c_custkey",
   "leftanti")

noChocolateCustomer.show
```

Hausaufgabe 2

Führen Sie die folgenden Abfragen in der Spark-Shell aus. Als Grundlage für die Abfragen dient das TPC-H Schema. Laden Sie dazu die TPC-H Daten wie in der Vorlesung gezeigt in die Spark-Shell.

(a) Laden Sie die `region.tbl` Datei als DataFrame Objekt in die Spark-Shell.

Um die Datei als DataFrame zu laden, braucht man das Format der Datei, in diesem Fall CSV, das Schema der Daten, das Zeichen, mit dem die Spalten in der CSV-Datei getrennt werden, sowie den Pfad. Für das Schema gibt man für jede Spalte den Namen und den Typ an, sowie ob das Feld auch einen null-Wert enthalten darf:

```
val region = spark.read.format("csv").schema(StructType(
   List(
     StructField("r_regionkey", IntegerType, false),
     StructField("r_name", StringType, false),
     StructField("r_comment", StringType, false)
   )
 )).option("delimiter", "|").load("region.tbl")
```

(b) Ermitteln Sie die Namen aller Regionen.

Mithilfe der `select` Funktion können bestimmte Spalten ausgewählt werden:

```
region.select($"r_name").show
```

(c) Ermitteln Sie die Zahl der Länder die nicht in Europa liegen.

Zuerst wird aus dem DataFrame `region` Europa entfernt. Danach wird das gefilterte DataFrame mit `nation` gejoined um alle nicht-europäischen Länder zu finden. Statt der Aktion `show` wird die Aktion `count` verwendet, die keinen Text ausgibt, sondern einen Integer zurückgibt:

```
val notEurope = region.filter($"r_name" =!= "EUROPE")

val nonEuropeanCountries = nation
   .join(notEurope, $"r_regionkey" === $"n_regionkey", "leftsemi")

nonEuropeanCountries.count
```

(d) Ermitteln Sie die größte Bestellung aus dem Jahr 1996.

Zuerst werden alle Bestellungen herausgesucht, die im Jahr 1996 getätigt wurden. Danach sucht man alle Einträge aus dem `lineitem` DataFrame heraus, die zu einer der Bestellungen aus dem Jahr 1996 gehören. Um den Gesamtumfang der Bestellung zu ermitteln summiert man die Menge der `lineitems` pro Bestellung auf.

```
val ordersOf1996 = orders
    .filter(year($"o_orderdate") === 1996)

val orderSizes = ordersOf1996
    .join(lineitem, $"o_orderkey" === $"l_orderkey")
    .groupBy($"o_orderkey")
    .agg(sum($"l_quantity").as("size"))

val maxOrderSize = orderSizes
    .agg(max($"size").as("max_size"))

val biggestOrder = orderSizes
    .join(maxOrderSize, $"size" === $"max_size")
    .select($"o_orderkey", $"size")

biggestOrder.show
```

(e) Ermitteln Sie welcher europäische Kunde im Jahr 1996 am meisten Geld ausgegeben hat.

Zuerst werden alle Kunden herausgesucht aus europäischen Ländern mithilfe der `nation` und `region` DataFrames. Die Bestellungen werden dann wie bereits in Aufgabe 4 nach dem Jahr 1996 gefiltert. Danach werden die Gesamtpreise der Bestellungen pro Kunde aufsummiert. Der Kunde mit den höchsten Ausgaben kann dann wieder mit einer Sortierung ermittelt werden:

```
val europeanCountries = nation
    .join(region, $"r_regionkey" === $"n_regionkey")
    .filter($"r_name" === "EUROPE")
    .select($"n_nationkey")

val customerOfEurope = customer
    .join(europeanCountries, $"n_nationkey" === $"c_nationkey", "leftsemi")
    .select($"c_custkey")

val ordersOf1996 = orders
    .filter(year($"o_orderdate") === 1996)

val customerWithCosts = ordersOf1996
    .join(customerOfEurope, $"o_custkey" === $"c_custkey")
    .groupBy($"c_custkey")
    .agg(sum($"o_totalprice").as("costs"))

val highestCosts = customerWithCosts
    .agg(max($"costs").as("highest_costs"))

val customerWithHighestCosts = customerWithCosts
    .join(highestCosts, $"costs" === $"highest_costs")
    .select($"c_custkey", $"costs")

customerWithHighestCosts.show
```

(f) Ermitteln Sie welche Unternehmen keine Kunden in Europa haben.

Um herauszufinden welche Unternehmen keine europäische Kunden haben, müssen alle Einträge in `lineitem` ermittelt werden, die von europäischen Kunden stammen. Dafür werden die DataFrames `region`, `nation`, `customer`, `orders` und `lineitem` miteinander verbunden. Mit einem anti-Join von `supplier` mit der Ergebnis-Relation bleiben nur die Unternehmen übrig, die keine Kunden in Europa haben:

```
val lineitemsOrderedByEuropeans = region
    .filter($"r_name" === "EUROPE")
    .join(nation, $"r_regionkey" === $"n_regionkey")
    .join(customer, $"n_nationkey" === $"c_nationkey")
    .join(orders, $"o_custkey" === $"c_custkey")
    .join(lineitem, $"o_orderkey" === $"l_orderkey")

val companiesWithoutEuropeanCustomers = supplier
    .join(lineitemsOrderedByEuropeans,
         $"l_suppkey" === $"s_suppkey", "leftanti")

companiesWithoutEuropeanCustomers.show
```

Hausaufgabe 3

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 4

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 5

Datenbanksysteme erlauben JSON-Objekte eingebettet als Attribute in Tabellen. Der zugehörige Syntax ist seit 2017 standardisiert[1] und zum Beispiel in PostgreSQL integriert[2]. Das nachfolgende Statement erstellt eine Hilfstabelle, die einen Ausschnitt des Uni-Schemas als JSON-Objekt enthält (und lässt sich in `hyper-db.de` eingeben).

```
with uni_json (name, doc) as (values ('VirtU', '{
  "Name": "Virtuelle Universitaet der Grossen Denker",
  "UniLeitung": {"Rektor": "Sokrates", "Kanzler": "Erhard"},
  "Fakultaeten":  [
    { "Name": "Philosophie", "Professoren": [
        { "PersNr": 2125, "Name": "Sokrates", "Rang": "C4",
          "Vorlesungen": [ {"VorlNr": 5041, "Titel": "Ethik", "SWS": 4},
            {"VorlNr": 5049, "Titel": "Maeeutik", "SWS": 2},
            {"VorlNr": 4052, "Titel": "Logik", "SWS": 4}]
        }]}]}'::json))
```

1. Geben Sie in SQL den Namen der jeweils ersten Fakultät in `uni_json` aus.

   ```
   select doc->'Fakultaeten'->0 from uni_json
   ```

2. Geben Sie in SQL die Personalnummer (`PersNr`) des ersten Professors der jeweils ersten Fakultät aus.

   ```
   select doc->'Fakultaeten'->0->'Professoren'->0->'PersNr' from uni_json
   ```

3. Joinen Sie diese mit der SQL-Relation `pruefen` und `Studenten`, um die Namen aller von ihm geprüften Studenten auszugeben.

   ```
   select s.Name from uni_json, pruefen p, Studenten s
   where cast(doc->'Fakultaeten'->0->'Professoren'->0->>'PersNr' as int) =
   p.PersNr and p.MatrNr=s.MatrNr;
   ```

[1] `https://standards.iso.org/ittf/PubliclyAvailableStandards/c067367_ISO_IEC_TR_19075-6_2017.zip`
[2] `https://www.postgresql.org/docs/current/functions-json.html`