Ex6_LuisZüttel_GionRubitschung_D1P

April 16, 2024

[ ]:
```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

1 Aufgabe 1

Realisieren Sie folgende Aufgaben in university.db. Geben Sie je genau ein SQL statement an.

1. Erhöhe das Gehalt jedes Computer-Science-Instruktors um 10%.

```
update instructor
set salary = salary * 1.1
where dept_name = 'Comp. Sci.';
```

2. Lösche alle Kurse die nie angeboten wurden.

```
delete from course
where course_id not in (
  select course_id from section
);
```

3. Stelle alle Studierenden miat mehr als 100 credits als Instruktor im selben Department mit einem Gehalt von 30'000 ein.

```
insert into instructor
select ID, name, dept_name, 30000
from student
where tot_cred > 100;
```

2 Aufgabe 2 (Verständnis von Abfragen)

Gegeben ist wieder das Schema von **publications.db** vom letzten Übungsblatt.

[ ]:
```
%sql sqlite:///../data/publications.db
%config SqlMagic.displaylimit = None
```

```
displaylimit: Value None will be treated as 0 (no limit)
```

1. Wieso ergibt folgende Abfrage nicht wie erwartet alle Titel die weniger als 20 Dollar kosten? Was liefert die Abfrage stattdessen?

```
select title from titles
where price < 20;
```

[ ]:
```
%%sql

select title, price from titles
where price < 20;
```

```
Running query in 'sqlite:///../data/publications.db'
```

[ ]:
```
+-----------------------------------------------------+-------+
|                        title                        | price |
+-----------------------------------------------------+-------+
|         The Busy Executive's Database Guide         | 19.99 |
|         Emotional Security: A New Algorithm         |  7.99 |
|    Prolonged Data Deprivation: Four Case Studies    | 19.99 |
| Cooking with Computers: Surreptitious Balance Sheets | 11.95 |
|          Silicon Valley Gastronomic Treats          | 19.99 |
|                    Sushi, Anyone?                   | 14.99 |
|      Fifty Years in Buckingham Palace Kitchens      | 11.95 |
|           You Can Combat Computer Stress!           |  2.99 |
|                 Is Anger the Enemy?                 | 10.95 |
|                  Life Without Fear                  |   7   |
|               The Gourmet Microwave                 |  2.99 |
|            Straight Talk About Computers            | 19.99 |
+-----------------------------------------------------+-------+
```

Die Abfrage ignoriert alle Title die keinen Preis angegeben haben, sprich wo `price` auf `null` gesetzt ist. Die Abfrage liefert also alle Titel bei denen der Preis bekannt ist und unter 20 Dollar kostet.

2. Wieso ergibt folgende Abfrage nicht wie erwartet die Autoren zusammen mit den Verlegern, bei denen sie publiziert haben? Was liefert die Abfrage stattdessen? Korrigieren Sie die Abfrage.

```
select au_lname, pub_name
from authors natural join titleauthor natural join titles
natural join publishers;
```

[ ]:
```
%%sql

select au_lname, pub_name, city, state
from authors natural join titleauthor natural join titles
natural join publishers;
```

```
Running query in 'sqlite:///../data/publications.db'
```

[ ]:
```
+----------+----------------------+----------+-------+
| au_lname |       pub_name       |   city   | state |
+----------+----------------------+----------+-------+
|  Carson  | Algodata Infosystems | Berkeley |   CA  |
```

```
|  Bennet  | Algodata Infosystems | Berkeley |   CA  |
+----------+----------------------+----------+-------+
```

Durch den `natural join` werden alle gemeinsamen Attribute berücksichtigt. `authors` und `publishers` haben beide die Attribute `city` und `state`. Die Abfrage liefert also alle Autoren zusammen mit den Verlegern, bei denen sie publiziert haben, in der selben Stadt und im selben Staat sind.

[ ]:
```
%%sql

select au_lname, pub_name from authors
  join titleauthor on authors.au_id = titleauthor.au_id
  join titles on titleauthor.title_id = titles.title_id
  join publishers on titles.pub_id = publishers.pub_id;
```

```
Running query in 'sqlite:///../data/publications.db'
```

[ ]:
```
+----------------+----------------------+
|    au_lname    |       pub_name       |
+----------------+----------------------+
|     White      |    New Age Books     |
|     Green      | Algodata Infosystems |
|     Green      |    New Age Books     |
|     Carson     | Algodata Infosystems |
|    O'Leary     | Algodata Infosystems |
|    O'Leary     |   Binnet & Hardley   |
|    Straight    | Algodata Infosystems |
|     Bennet     | Algodata Infosystems |
|      Dull      | Algodata Infosystems |
|   Gringlesby   |   Binnet & Hardley   |
|    Locksley    | Algodata Infosystems |
|    Locksley    |    New Age Books     |
| Blotchet-Halls |   Binnet & Hardley   |
|    Yokomoto    |   Binnet & Hardley   |
|  del Castillo  |   Binnet & Hardley   |
|    DeFrance    |   Binnet & Hardley   |
|   MacFeather   | Algodata Infosystems |
|   MacFeather   |   Binnet & Hardley   |
|     Karsen     |   Binnet & Hardley   |
|    Panteley    |   Binnet & Hardley   |
|     Hunter     | Algodata Infosystems |
|     Ringer     |   Binnet & Hardley   |
|     Ringer     |    New Age Books     |
|     Ringer     |    New Age Books     |
|     Ringer     |    New Age Books     |
+----------------+----------------------+
```

3. Wieso ergibt folgende Abfrage nicht wie erwartet alle Verleger, die höchstens zwei Psychologiebücher verlegt haben? Was liefert die Abfrage stattdessen? Korrigieren Sie die Abfrage.

```
select pub_id, count(title_id) as numtitles
from titles
where type like 'psychology%'
group by pub_id
having numtitles <= 2;
```

[ ]:
```
%%sql

select pub_id, count(title_id) as numtitles
from titles
where type like 'psychology%'
group by pub_id
having numtitles <= 2;
```

```
Running query in 'sqlite:///../data/publications.db'
```

[ ]:
```
+--------+-----------+
| pub_id | numtitles |
+--------+-----------+
|  0877  |     1     |
+--------+-----------+
```

[ ]:
```
%%sql

with psychology_books as (
  select *
  from titles
  where type like 'psychology%'
)
select pub_id, count(pub_id) as numtitles
from psychology_books
group by pub_id
having numtitles <= 2
union
select pub_id, 0 from (
  select pub_id from publishers
  except
  select pub_id from psychology_books
);
```

```
Running query in 'sqlite:///../data/publications.db'
```

[ ]:
```
+--------+-----------+
| pub_id | numtitles |
+--------+-----------+
|  0877  |     1     |
|  1389  |     0     |
+--------+-----------+
```

3 Aufgabe 3 (Verständnis von Abfragen)

Gegeben sei folgendes Schema:

$person(\underline{name}, street, city)$

$purchase(\underline{name}, \underline{id}, number\_of\_items)$ $name \rightarrow person$ $id \rightarrow product$

$product(\underline{id}, supplier\_name, description, price)$ $supplier\_name \rightarrow supplier$

$supplier(\underline{name}, street, city)$

Beschreiben Sie umgangssprachlich das Resultat folgender Abfragen.

1. `sql select name from person natural join purchase;`

   Das Resultat zeigt alle Personen von `person`, die jeweils ein oder mehrere Käufe in `purchase` für eine beliebige Anzahl Produkte gemacht haben.

2. `sql select name from person natural join purchase natural join product natural join supplier;`

   Das Resultat zeigt bis und mit dem `natural join` auf `product`, dasselbe wie das obere Resultat mit dem Unterschied, dass nun die Produkthersteller, Beschrieb und Preis. Erst dann bei dem `natural join` auf `supplier` wird das Resultat so erweitert, bei denen der Name des Herstellers und der Person übereinstimmen und als gleiches Attribut angesehen wird.

3. `sql select supplier_name, avg(price) from product group by supplier_name;`

   Das Resultat zeigt die Namen aller Hersteller und deren Durchschnittspreis deren Produkte. (Produkte mit unbekannten Preisen werden nicht gezählt)

4. `sql select supplier_name, avg(price) from product where id in (select id from purchase) group by supplier_name;`

   Das Resultat zeigt dasselbe wie das obere Resultat, nur das Produkte beachtet werden, welche mindestens einmal von einen Kunden gekauft worden.

5. `sql select sum(price * number_of_items) from person natural join purchase natural join product where name = "Hans Muster";`

   Das Resultat zeigt den Preis insgesamt aller Produkte, welche Hans Muster gekauft hat.

6. `sql select id form product except select id from purchase where name = "Hans Muster";`

   Das Resultat gibt alle Ids der Produkte an, welche Hans Muster nicht gekauft hat.

4 Aufgabe 4 (Effiziente Join-Algorithmen)

Rufen Sie sich folgende Konzepte wieder in Erinnerung: - den Algorithmus für binäre Suche - die Zeitkomplexität eines Algorithmus (die Funktion, die aus der Grösse der Eingabe die Laufzeit im worst-case ermittelt)

Gegeben seien Relationen $r(A, B)$ und $s(B, C)$, als Listen von Tupeln. Attribut B ist vom Typ `Integer`. Gehen Sie von dem im Übungsblatt 4 entwickelten Algorithmus für den natürlichen Join aus, und verfeinern Sie ihn für die folgenden Fälle in einen *möglichst effizienten* Algorithmus, um den `natural join` der beiden Relationen zu berechnen. Geben Sie jeweils die Zeitkomplexität des Algorithmus an.

1. $\{B\}$ ist ein Superschlüssel für r

[ ]:
```
r = [('D', 4), ('A', 2), ('B', 1), ('C', 3) ]
s = [(3, 'E'), (1, 'F'), (2, 'G'), (2, 'H')]

def head(lst: list):
    return lst[:int(len(lst)/2)]

def tail(lst: list):
    return lst[int(len(lst)/2):]

def cons(element, lst: list) -> list:
    return [element] + lst

def natural_join(r: list, s : list) -> list:
    if not r or not s:
        return [] # return nil
    matching_tuples = []
    new_s = []
    tuple_r = r[0]
    for index, tuple_s in enumerate(s):
        if tuple_r[1] != tuple_s[0]:
            new_s.append(tuple_s) # Alle Elemente, die nicht passen, werden in␣
 ↪new_s gespeichert
            continue
        matching_tuples = cons((tuple_r[0],) + tuple_s, matching_tuples)
    return matching_tuples + natural_join(r[1:], new_s)

natural_join(r, s)
```

[ ]:
```
[('A', 2, 'H'), ('A', 2, 'G'), ('B', 1, 'F'), ('C', 3, 'E')]
```

Die Zeitkomplexität dieses Algorithmus wäre die Anzahl Tupel beider Relationen miteinander multipliziert, also $m \times n$

2. wie 1. und zusätzlich ist die Liste für r nach Attribut B sortiert

[ ]:
```
r = [('B', 1), ('A', 2), ('C', 3), ('D', 4)]
s = [(3, 'E'), (1, 'F'), (2, 'G'), (2, 'H')]

natural_join(r, s)
```

[ ]:
```
[('B', 1, 'F'), ('A', 2, 'H'), ('A', 2, 'G'), ('C', 3, 'E')]
```

Die Zeitkomplexität dieses Algorithmus ist dieselbe wie zuvor, da es keine Rolle für den Algorithmus spielt in welcher Reihenfolge die erste Menge ist.

[ ]:
```
r = [('B', 1), ('A', 2), ('C', 3), ('D', 4)]
s = [(4, 'G'), (4, 'H')]

def natural_join(r: list, s : list) -> list:
    if not r or not s:
        return [] # return nil
    matching_tuples = []
    tuple_r = r[0]
    non_matching_index = 0
    for index, tuple_s in enumerate(s):
        if tuple_r[1] != tuple_s[0]:
            non_matching_index = index
            break
        matching_tuples = cons((tuple_r[0],) + tuple_s, matching_tuples)
    return matching_tuples + natural_join(r[1:], s[non_matching_index:])

natural_join(r, s)
```

[ ]:
```
[('D', 4, 'H'), ('D', 4, 'G')]
```

Die Zeitkomplexität ist $ m + n $, da beide Relationen sortiert sind und beim ersten Tupel, welches nicht matched, zum nächsten Tupel von s geht.