Aufgabe 1

Evaluation der Relation r:

A	**B**	**C**	**D**	**Evaluation**
”A”	1000	3	””	true
”A”	700	NULL	”agh”	unknown
”A”	NULL	0	”abcdf”	unknown
”A”	1000	4	NULL	true
”B”	NULL	NULL	”bdf”	unknown
”B”	1500	NULL	”c”	unknown
NULL	1000	8	””	false
NULL	700	12	NULL	true

- $\sigma_{B \cdot C < 5000 \text{ or } D \text{ is unknown}}(r)$

A	**B**	**C**	**D**
”A”	1000	3	””
”A”	1000	4	Null
Null	700	12	Null

- $_A g_{avg(B),sum(C)}(r)$

A	**avg(B)**	**sum(C)**
”A”	675	7
”B”	750	0

- $_A g_{avg(B)}(\pi_{A,B}(r))$

A	**avg(B)**
”A”	566.66
”B”	750

- natural join

A	**B**	**C**	**D**	**E**
”B”	NULL	NULL	”bdf”	1
”B”	1500	NULL	”c”	1

- right outer join

A	**B**	**C**	**D**	**E**
”B”	NULL	NULL	”bdf”	1
”B”	1500	NULL	”c”	1
”C”	NULL	NULL	NULL	2
”C”	NULL	NULL	NULL	3

- full outer join

A	B	C	D	E
"A"	1000	3	""	NULL
"A"	700	NULL	"agh"	NULL
"A"	NULL	0	"abcdf"	NULL
"A"	1000	4	NULL	NULL
"B"	NULL	NULL	"bdf"	1
"B"	1500	NULL	"c"	1
NULL	1000	8	""	NULL
NULL	700	12	NULL	NULL
"C"	NULL	NULL	NULL	2
"C"	NULL	NULL	NULL	3

Aufgabe 2

Wir nehmen an, dass $r.C$ n und $s.C$ m Elemente hat.
Alle Algorithmen machen Gebrauch folgender Hilfsmethode:

Listing 1: Unsortierte Listen

```
1  combine(tubel r, tupel s)
2  t = new tupel
3  n = size(r)
4  m = size(s)
5
6  for i=1 to n
7    add r[i] to t
8
9  for j=2 to m
10   add s[j] to t
11
12 return t
```

- Bei beidseits unsortierten Listen muss für jedes Element in $s.C$ das passende Schlüsselelement in $r.C$ gefunden werden, und dies kann nur beim Durchgehen durch alle Elemente erreicht werden. Dies ergibt eine Zeitkomplexität von $T(n) = \Theta(n \cdot m)$. Nehmen wir an, dass n und m gleichschnell wachsen, so könnte man die Zeitkomplexität auf $\Theta(n^2)$ vereinfachen.

Listing 2: Unsortierte Listen

```
13 naturalJoinUnsorted(list r, list s)
14 l = new list
15 n = size(r)
16 m = size(s)
17
18 for i=1 to n
19   for j=1 to m
20     if r[i] = s[j]
21       add combine(r[i],s[j]) to l
22       next i
23
24 return l
```

- Bei sortiertem $r.C$ muss für jedes Element in $s.C$ das passende Schlüsselelement in $r.C$ gefunden werden, und dies kann nur erreicht werden, indem man mit einer Suche in $r.C$ den (falls existierenden) richtigen Schlüssel sucht, welches eine Zeit von $\Theta(log(n))$ braucht[1]. Dies ergibt eine Zeitkomplexität von $T(n) = \Theta(n \cdot log(m))$. Nehmen wir an, dass n und m gleichschnell wachsen, so könnte man die Zeitkomplexität auf $\Theta(n \cdot log(n))$ vereinfachen.

Listing 3: r.C ist sortiert

```
naturalJoinHalfSorted(list r, list s)
l = new list
n = size(r)
m = size(s)

for i=1 to n
  logSearch(r[i] in s.C)
    add combine(r[i],s[j]) to l

return l
```

- Bei sortiertem $r.C$ und sortiertem $s.C$ muss das passende Schlüsselelement in $r.C$ gefunden werden, und dies kann sehr einfach erreicht werden, indem man für jedes Element von $s.C$ ($\Theta(n)$ wie in den vorigen) das nächsthöhere Element in $r.C$ untersucht (welches in konstanter Zeit ($\Theta(1)$) möglich ist[2]). Dies ergibt insgesamt eine Zeitkomplexität von $T(n) = \Theta(n)$.

Listing 4: Beidseits sortiert

```
naturalJoinSorted(list r, list s)
l = new list
n = size(r)
m = size(s)
int key=0

for i=1 to n
  do
    if r[i] = s[key]
      add combine(r[i],s[j]) to l
    key++
    if r[i] < s[key]
      key-1
      next i
  while (r[i] < s[key])

return l
```

Aufgabe 3

- $\sigma_{A>10}(r-t) = \sigma_{A>10}(r) - \sigma_{A>10}(t)$. Es bleiben in beiden Fällen nur die x, y, z übrig, welche nur in r enthalten sind und für welche $x > 10$ gilt. Folgend ein Beispiel.

[1] Z.B. durch halbieren, prüfen des mittleren Elements ob grösser oder kleiner des gesuchten, und das ganze zu wiederholen.
[2] Um genau zu sein, beträgt die Laufzeit der inneren Schleife auch $\Theta(m)$, weil `key` durch jedes Element läuft (von 0 bis m). Daraus folgt eine gesamte Laufzeit von $\Theta(n \cdot m)$. Bei gleichschnellem Wachsen von n und m ergibt dies $\Theta(2n)$, welches eine asymptotische Laufzeit von $\Theta(n)$ ergibt.

r

A	B	C
1	2	3
7	9	2
12	7	8
13	1	2

t

A	B	C
1	2	3
7	8	5
12	7	8
11	1	2

A	B	C
13	1	2

- $r \bowtie (\sigma_{B=''X'' \vee C=''Z''}(s))$: Von (s) werden nur die Zeilen ausgewählt für die gilt $B ='' X'' \vee C ='' Z''$ Mit $\bowtie$ werden nur die Zeilen beachtet für die das gleiche gilt.

  $\sigma_{B=''X'' \vee C=''Z''}(r \bowtie s)$: Mit $r \bowtie s$ werden zwar alle Zeilen beachtet for die gilt $B(r) = B(s) \vee C(r) = C(s)$, aber nur die ausgewählt wo $B ='' X''$ und $C ='' Z''$. $\Rightarrow$ Damit sind die Ausdrücke äquivalent.

- Angenommen $z \in B(r) \vee z \notin B(s)$. $r \div \pi_B(s)$ enthält nur die Spalten A und C von r.
  $r \div \pi_B(s) \times \pi_B(s)$ enthält nur die Spalten A(r), C(r) und B(s). Dass heisst, alle Elemente von B(r) die nicht Elemente von B(s) sind können nicht im Kreuzprodukt enthalten sein. D.h, auf der linken Seite wäre ein z vorhanden, aber auf der rechten nicht.