VerteilteWebInf Hausaufgabe 3

Gruppe 6

October 30, 2014

Aufgabe 1

- $R \bowtie S = R \bowtie (\Pi_C(R) \rtimes S)$

  " $\Rightarrow$ "
  $\forall t \in R \bowtie S : \exists t_1 \in R \wedge \exists t_2 \in S(t_1.C = t_2.C)$
  $\Rightarrow t_2 \in (\Pi_C(R) \rtimes S) \Rightarrow t_1 \bowtie t_2 \Rightarrow t \in (R \bowtie (\Pi_C(R) \rtimes S))$, da die Joinbedingung nur das Attribut C enthält.
  Man erhält damit alle möglichen Joinpartner der Relation S zur Relation R.
  " $\Leftarrow$ "
  $\forall t \in R \bowtie (\Pi_C(R) \rtimes S) : \exists t_1 \in R \wedge \exists t_2 \in \Pi_C(R) \rtimes S(t_1.C = t_2.C)$
  $\Rightarrow t_2 \in S \Rightarrow t_1 \bowtie t_2 \Rightarrow t \in R \bowtie S$

- $R \bowtie S = (\Pi_C(S) \rtimes R) \bowtie (\Pi_C(R) \rtimes S)$

  " $\Rightarrow$ "
  $\forall t \in R \bowtie S : \exists t_1 \in R \wedge \exists t_2 \in S(t_1.C = t_2.C)$
  $\Rightarrow t_1 \in (\Pi_C(S) \rtimes R) \wedge t_2 \in (\Pi_C(R) \rtimes S)$
  $\Rightarrow t_1 \bowtie t_2 \Rightarrow t \in ((\Pi_C(S) \rtimes R) \bowtie (\Pi_C(R) \rtimes S))$, da die Joinbedingung nur das Attribut C enthält.
  Auf der linken Seite des Joins erhält man somit alle Joinpartner von R für S, auf der rechten Seite alle Joinpartner von S für R.
  " $\Leftarrow$ "
  $\forall t \in (\Pi_C(S) \rtimes R) \bowtie (\Pi_C(R) \rtimes S) : \exists t_1 \in (\Pi_C(S) \rtimes R) \wedge \exists t_2 \in \Pi_C(R) \rtimes S(t_1.C = t_2.C)$
  $\Rightarrow t_1 \in R \wedge t_2 \in S \Rightarrow t_1 \bowtie t_2 \Rightarrow t \in R \bowtie S$

Aufgabe 2

a) $R_1 = \{[\underline{id : integer}, TokenID : bigint]\}$
$R_2 = \{[\underline{id : integer}, CustomerID : integer]\}$
$R_3 = \{[\underline{id : integer}, StoreID : integer]\}$
$R_4 = \{[\underline{id : integer}, amount : numeric(7, 2)]\}$
$R_5 = \{[\underline{id : integer}, time : timestamp]\}$

f) 3 Mio Datensätze:
Query1:
Column Store: 13 Millisekunden
Row Store: 35 Millisekunden
Query2:
Column Store: 412 Millisekunden
Row Store: 579 Millisekunden

Sowohl für Query1 als auch für Query2 ist die Column Store Implementierung schneller. Dies könnte vor allem daran liegen, dass Aggregationsfunktionen (OLAP) effizienter auf Spalten-basierten Systemen arbeiten können. Da nur die entsprechenden Spalten und nicht die gesamte Zeile eine Tabelle in den Speicher geladen werden muss.