Robert Bielas
laboratorium z Netezzy - raport

UWAGA: proszę zwracać uwagę na komentarze pod screenami/tekstami planera, gdyż pokazują one moją interpretację wyników.

Część pierwsza, zad. 11
Najlepsza dystrybucja tabel dla tabel w LABDB: w tabelach joinowanych (orders i lineitem, part i lineitem, supplier i part sup itp.) zaproponowałbym dystrybucję wg kolumny, która łączy odpowiednie pary tabel. Jeśli nie zależy nam na informacjach z kilku tabel, ale z jednej, zwróciłbym uwagę na to, którą operację wyszukiwania będziemy najczęściej wykonywać: np. wg. daty. Wtedy zdystrybuowałbym tabelę, posortowaną wg interesującej nas kolumny, i zdystrybuował po kluczu którego zbiór unikalnych wartości ma moc >= 22 (tyle najwyżej data slice'ów jest w serwerze Netezzy wykorzystywanej na laboratorium).

Część druga, zad. 1

```
select count(*) from paymentreport, rezultat:
```
100000000

```
select count(*) from paymentreportdetail, rezultat:
```
600000000

```
select count(*) from company, rezultat: 22

select count(*) from COMPANYHEADQUARTER, rezultat: 43
select count(*) from Paymentreport_age
```
100000000

```
select count(*) from Paymentreport_gender, rezultat:
```
100000000
```
Wniosek: tabele są pokaźnych rozmiarów

zad. 2 rezultat:
```

DATASLICEID	rows
1	4546633
2	4546634
3	4546634
4	4546634
5	4546634
6	4545108
7	4545108
8	4545108
9	4545108
10	4545108
11	4545108
12	4545108
13	4545108
14	4545108
15	4545108
16	4545108
17	4545108
18	4545107
19	4545107
20	4545107
21	4545107
22	4545107

zad. 3
Dystrybucja tabeli paymentreport_gender (po kluczu gender):

```
select datasliceid, count(datasliceid) as "rows"
from PAYMENTREPORT_GENDER
group by datasliceid
order by 1
```

DATASLICEID	rows
10	49999997
15	50000003

Dystrybuja tabeli paymentreport_age (po kluczu age)

```
select datasliceid, count(datasliceid) as "rows"
from PAYMENTREPORT_AGE
group by datasliceid
order by 1
```

DATASLICEID	rows
1	5000542
2	5997064
3	4999551
4	5003441
5	4995738
6	5000843
7	5000580
8	5003587
9	5000312
10	5002074
11	4004265
12	3997798
13	3999722
14	4001385
15	3998297
16	3995568
17	4003395
18	4002063
19	4999312
20	3998698
21	3998174
22	3997591

Po utworzeniu tabel w mojej bazie na tych dystrybucjach rozmieszczenie krotek było identyczne.

zad. 4

```
137  select gender, avg(salary)
138  from PAYMENTREPORT
139  group by gender
140
```

Output | Result 1

Drag a column header here to group by that column.

GENDER	AVG
M	26003,028533
F	25997,030155

U_bielas 1 query, 1,300 sec., 2 rows laby.sql

```
141  select age, avg(salary)
142  from PAYMENTREPORT
143  group by age
144  order by 1
```

Output | Result 1

Drag a column header here to group by that column.

AGE	AVG
0	26003,247861
1	26001,350730

U_bielas 1 query, 1,356 sec., 68 rows * laby.sql

Sprawdzenie czasu wykonania obu agregacji dla dystrybucji po age:

```
147  select age, avg(salary)
148  from PAYMENTREPORT_age
149  group by age
150  order by 1
151
```

Output | Result 1

Drag a column header here to group by that column.

AGE	AVG
0	26003,247861
1	26001,350730
2	26005,294429

U_bielas 1 query, 1,350 sec., 68 rows laby.sql

```
157  select gender, avg(salary)
158  from PAYMENTREPORT_age
159  group by gender
160
```

Output | Result 1

Drag a column header here to group by that column.

GENDER	AVG
F	25997,030155
M	26003,028533

U_bielas 1 query, 1,425 sec., 2 rows * laby.sql

Czas wykonania jest rozsądny, gdyż nie występuje data skew - dane wydystrybuowane po age są równomiernie rozłożone po większej ilości SPU - jak w przypadku dystrybucji po id. Sprawdzenie czasu wykonania obu agregacji dla dystrybucji po gender:

```
152  select age, avg(salary)
153  from PAYMENTREPORT_gender
154  group by age
155  order by 1
```

Output | Result 1

Drag a column header here to group by that column.

AGE	AVG
0	26003,247861
1	26001,350730
2	26005,294429

U_bielas 1 query, 9,016 sec., 68 rows * laby.sql

```
161  select gender, avg(salary)
162  from PAYMENTREPORT_gender
163  group by gender
164
165
```

Output | Result 1

Drag a column header here to group by that column.

GENDER	AVG
M	26003,028533
F	25997,030155

U_bielas 1 query, 9,043 sec., 2 rows * laby.sql

Tutaj nie dziwi nas, że czasy są podobne - w przetwarzanie są zaangażowane tylko dwa S-Blade'y (bo do dwóch wydystrybuowano dane) - reszta śpi.

zad. 5

```
select count(*) from PAYMENTREPORTDETAIL_ALL_BONUSCODE
```

1600000000

```
select count(*) from PAYMENTREPORTDETAIL_ALL2
```

3200000000

```
select count(*) from PAYMENTREPORTDETAIL_ALL2_BONUSCODE
```

3200000000

```
select count(*) from PAYMENTREPORTDETAIL_ALL
```

1600000000

```
dystrybucja w paymentreportdetail_all
```

DATASLICEID	rows
1	72746141
2	72746142
3	72746142
4	72746142
5	72746142
6	72721723
7	72721723
8	72721723
9	72721723
10	72721723
11	72721723
12	72721723
13	72721723
14	72721723
15	72721723
16	72721723
17	72721723
18	72721723
19	72721723
20	72721723
21	72721723
22	72721723

dystrybucja w paymentreportdetail_all_bonuscode

```
151  select datasliceid, count(datasliceid) as "rows"
152  from PAYMENTREPORTDETAIL_ALL_BONUSCODE
153  group by DATASLICEID
154  order by 1
155
156
```

Output | Result 1

Drag a column header here to group by that column.

DATASLICEID	rows
1	400006187
2	800027467
3	399966346

dystrybucja w paymentreportdetail_all2

```
155
156  select datasliceid, count(datasliceid) as "rows"
157  from PAYMENTREPORTDETAIL_ALL2
158  group by DATASLICEID
159  order by 1
```

Output | Result 1

Drag a column header here to group by that column.

DATASLICEID	rows
1	145492284
2	145492285
3	145492285
4	145492285
5	145492285
6	145443446
7	145443446
8	145443446
9	145443446
10	145443446
11	145443446
12	145443446
13	145443446
14	145443446
15	145443446
16	145443446
17	145443445
18	145443445
19	145443445
20	145443445
21	145443445
22	145443445

dystrybucja w paymentreportdetail_all2_bonuscode:

```
161
162  select datasliceid, count(datasliceid) as "rows"
163  from PAYMENTREPORTDETAIL_ALL2_BONUSCODE
164  group by DATASLICEID
165  order by 1
166
```

Output | Result 1

Drag a column header here to group by that column.

DATASLICEID	rows
1	800037916
2	1599976875
3	799985209

tabela paymentreportdetail_all jest zdystrybuowana na id
tabela paymentreportdetail_all2 jest rozdystrybowana na id
tabela paymentreportdetail_all2_bonuscode jest rozdystrybowana na bonuscode
tabela paymentreportdetail_all_bonuscode jest rozdystrybowana na bonuscode

Polecenia agregujące:

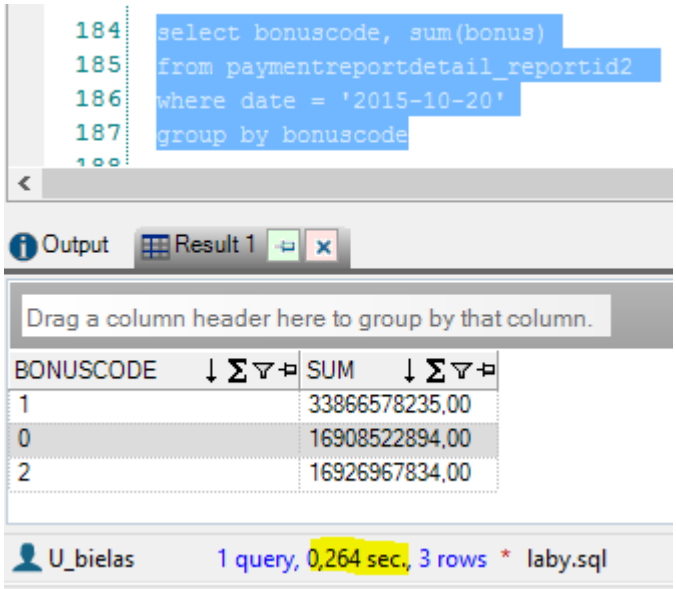
```
select date, sum(bonus)
from PAYMENTREPORTDETAIL_ALL
group by date

czas: 13.916 sec

select bonuscode, sum(bonus)
from PAYMENTREPORTDETAIL_REPORTID
group by bonuscode

czas: 4.793 sec

select bonuscode, sum(bonus)
from PAYMENTREPORTDETAIL_ALL_BONUSCODE
group by bonuscode
czas: 85.993 sec (!!!)
```

Ostatnia agregacja wykonywała się najwolniej, ponieważ rozdystrybuowano ją na bonuscode(są 3). Mamy tu do czynienia ze zjawiskiem data_skew: tylko 3 data slice'y były zaangażowane w przetwarzanie (do trzech slice'ów zostały zdystrybuowane dane).

Druga agregacja jest najszybsza, gdyż dane zostały przechowane równomiernie w największej ilości slice'ów. Więcej S-Bladów przetwarzało mniej danych -> wynik marzenie.

6.

Czas wykonania query dla paymentreportdetail_reportid2 (dystrybucja po payment_report, posortowane po dacie):



DATASLICEID	rep2
1	27279798
2	27279804
3	27279804
4	27279804
5	27279804
6	27270648
7	27270648
8	27270648
9	27270648
10	27270648
11	27270648
12	27270648
13	27270648
14	27270648
15	27270648
16	27270648
17	27270648
18	27270642
19	27270642
20	27270642
21	27270642
22	27270642

Czas wykonania query dla paymentreportdetail_reportid

```
190  select bonuscode, sum(bonus)
191  from paymentreportdetail_reportid
192  where date = '2015-10-20'
193  group by bonuscode
```

Output | Result 1

Drag a column header here to group by that column.

BONUSCODE	SUM
1	33866578235,00
2	16926967834,00
0	16908522894,00

U_bielas 1 query, 5,313 sec., 3 rows laby.sql

DATASLICEID	rep
1	27279798
2	27279804
3	27279804
4	27279804
5	27279804
6	27270648
7	27270648
8	27270648
9	27270648
10	27270648
11	27270648
12	27270648
13	27270648
14	27270648
15	27270648
16	27270648
17	27270648
18	27270642
19	27270642
20	27270642
21	27270642
22	27270642

Czas wykonania query dla paymentreportdetail_date:

```
195  select bonuscode, sum(bonus)
196  from paymentreportdetail_date
197  where date = '2015-10-20'
198  group by bonuscode
```

Output | Result 1

Drag a column header here to group by that column.

BONUSCODE	SUM
2	16926967834,00
1	33866578235,00
0	16908522894,00

U_bielas 1 query, 2,619 sec., 3 rows * laby.sql

DATASLICEID	date
1	30000824
2	29999864
3	30011297
4	29998409
5	29993986
6	29999312
7	24997732
8	20003703
9	20002352
10	20000642
11	20005831
12	20001057
13	24997217
14	30001279
15	30000847
16	29997767
17	29999905
18	30004882
19	29990426
20	30003865
21	29996971
22	29991832

Najbardziej efektywnie zostało wykonane query dla dystrybucji po payment_report, gdzie dane szostały posortowane po dacie(czyli w tabeli paymentreportdetail_reportid2).

Czsy wykonania dla większych tabel:
Czas wykonania dla paymentreportdetail_all:

```
200  select bonuscode, sum(bonus)
201  from paymentreportdetail_all
202  where date = '2015-10-20'
203  group by bonuscode
204
```

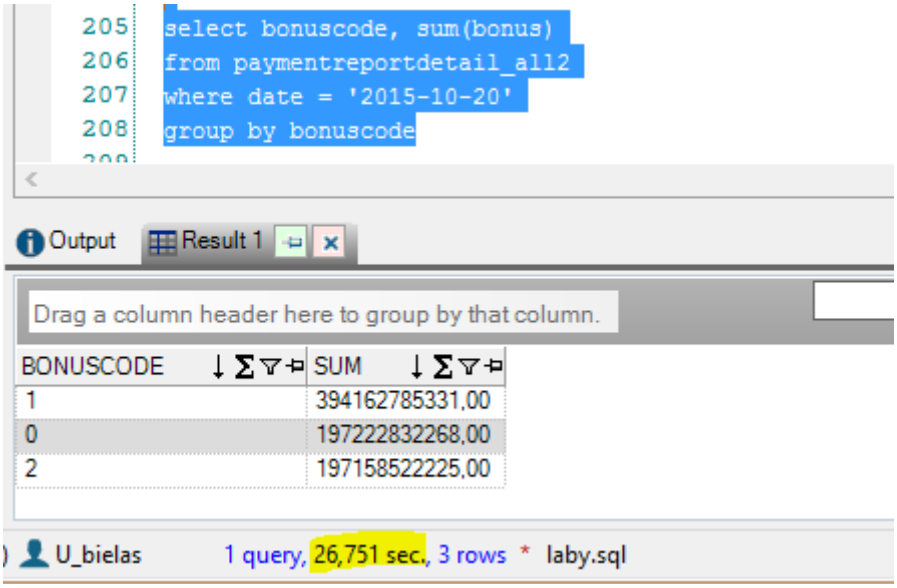
Output | Result 1

Drag a column header here to group by that column.

BONUSCODE	SUM
1	130523816882,00
2	65283759767,00
0	65220461642,00

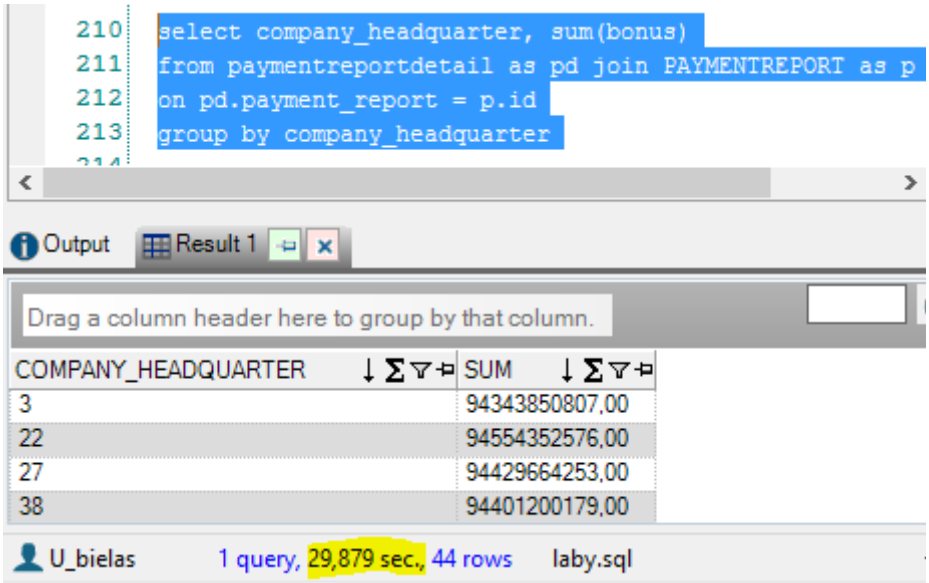
U_bielas 1 query, 13,501 sec., 3 rows * laby.sql

Czas wykonania dla paymentreportdetail_all2:



Czas wykonania jest proporcjonalny do ilości krotek w tabeli (w _all jest ich dwa razy mniej niż w _all2 -> w _all czas wykonania był 2 razy mniejszy). Pamiętajmy, że dane w tych tabelach zostały rozdystrybuowane na tym samym kluczu.

7.

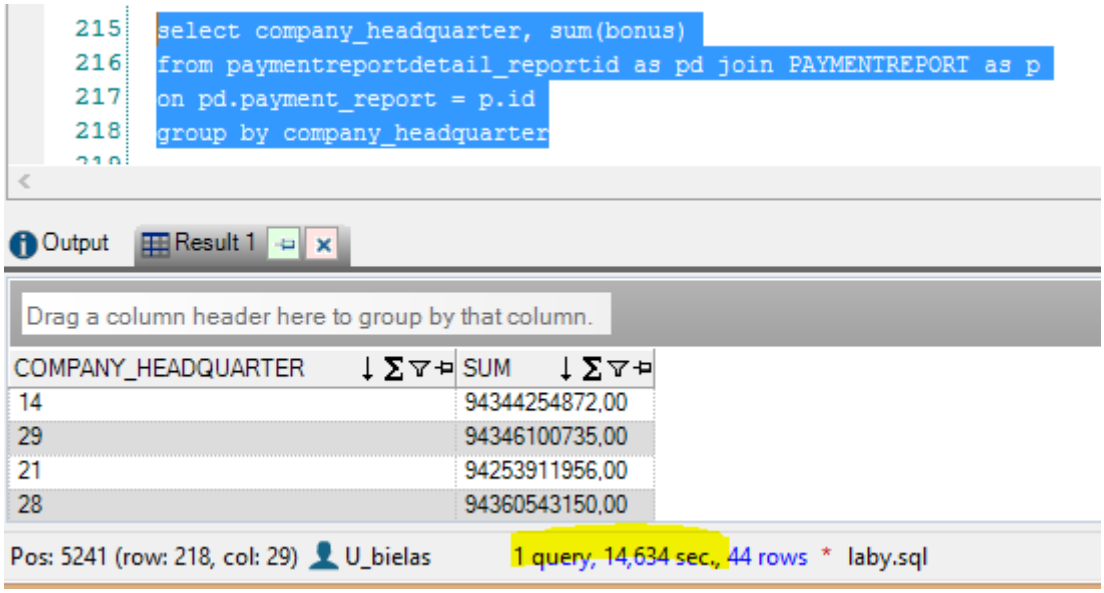


QUERY VERBOSE PLAN:

Node 1.
[SPU Sequential Scan table "PAYMENTREPORTDETAIL" as "PD" {(PD."ID")}]
-- Estimated Rows = 599005119, Width = 8, Cost = 0.0 .. 11718.8, Conf = 90.0 (FACT)
Restrictions:
(PD.PAYMENT_REPORT NOTNULL)
Projections:
1:PD.BONUS 2:PD.PAYMENT_REPORT
Cardinality:

PD.PAYMENT_REPORT 100.0M (JIT)
[SPU Distribute on {(PD.PAYMENT_REPORT)}]
[HashIt for Join]
Node 2.
[SPU Sequential Scan table "PAYMENTREPORT" as "P" {(P."ID")}]
-- Estimated Rows = 100000000, Width = 8, Cost = 0.0 .. 3595.5, Conf = 80.0 (FACT)
Restrictions:
(P."ID" NOTNULL)
Projections:
1:P.COMPANY_HEADQUARTER 2:P."ID"
Node 3.
[SPU Hash Join Stream "Node 2" with Temp "Node 1"
{(PD.PAYMENT_REPORT,P."ID")}]
-- Estimated Rows = 599005119, Width = 8, Cost = 39316.9 .. 84826.1, Conf = 64.0
Restrictions:
(P."ID" = PD.PAYMENT_REPORT)
Projections:
1:P.COMPANY_HEADQUARTER 2:PD.BONUS
Cardinality:
PD.PAYMENT_REPORT 85.2M (Adjusted)
Node 4.
[SPU Group]
-- Estimated Rows = 44, Width = 8, Cost = 39316.9 .. 98439.9, Conf = 0.0
Projections:
1:P.COMPANY_HEADQUARTER 2:PD.BONUS
[SPU Return]
[HOST Merge Group]
Node 5.
[Host Aggregate]
-- Estimated Rows = 44, Width = 20, Cost = 39316.9 .. 98439.9, Conf = 0.0
Projections:
1:P.COMPANY_HEADQUARTER 2:SUM(PD.BONUS)
[Host Return]

Z planu wynika, że jest wykonywana redystrybucja po kluczu paymentreport. Mamy do czynienia z single redistribution. Ponieważ taka sytuacja zachodzi, konieczność owej redystrybucji (wyciąganie odpowiednich danych z data slice'ów i przesłanie ich do odpowiedniego S-Blade'a) wpływa negatywnie na czas wykonania.



QUERY VERBOSE PLAN:

Node 1.
[SPU Sequential Scan table "PAYMENTREPORT" as "P" {(P."ID")}]
-- Estimated Rows = 100000000, Width = 8, Cost = 0.0 .. 3595.5, Conf = 80.0 (FACT)
Restrictions:
(P."ID" NOTNULL)
Projections:
1:P.COMPANY_HEADQUARTER 2:P."ID"
[HashIt for Join]
Node 2.
[SPU Sequential Scan table "PAYMENTREPORTDETAIL_REPORTID" as "PD" {(PD.PAYMENT_REPORT)}]
-- Estimated Rows = 599426939, Width = 8, Cost = 0.0 .. 11718.8, Conf = 90.0 (FACT)
Restrictions:
(PD.PAYMENT_REPORT NOTNULL)
Projections:
1:PD.BONUS 2:PD.PAYMENT_REPORT
Cardinality:
PD.PAYMENT_REPORT 100.0M (JIT)
Node 3.
[SPU Hash Join Stream "Node 2" with Temp "Node 1" {(P."ID",PD.PAYMENT_REPORT)}]
-- Estimated Rows = 599426939, Width = 8, Cost = 3595.5 .. 72817.7, Conf = 64.0
Restrictions:
(PD.PAYMENT_REPORT = P."ID")
Projections:
1:P.COMPANY_HEADQUARTER 2:PD.BONUS
Cardinality:
PD.PAYMENT_REPORT 85.2M (Adjusted)
Node 4.

[SPU Group]
-- Estimated Rows = 44, Width = 8, Cost = 3595.5 .. 86441.0, Conf = 0.0
Projections:
1:P.COMPANY_HEADQUARTER 2:PD.BONUS
[SPU Return]
[HOST Merge Group]
Node 5.
[Host Aggregate]
-- Estimated Rows = 44, Width = 20, Cost = 3595.5 .. 86441.0, Conf = 0.0
Projections:
1:P.COMPANY_HEADQUARTER 2:SUM(PD.BONUS)
[Host Return]

Czas wykonywania jest krótszy, gdyż nie trzeba dokonywać redystrybucji (jak wynika z planera - nie ma wzmianki distribute on(...)).
8.

```
220  select c.company_name, sum(bonus)
221  from paymentreportdetail_reportid as pd join paymentreport as p
222  on pd.payment_report = p.id join companyheadquarter as ch
223  on ch.id = p.company_headquarter  join company as c
224  on c.id = ch.company
225  group by c.company_name
```

Output | Result 1

Drag a column header here to group by that column.

COMPANY_NAME	SUM
Yozio	283449163204,00
Feedbug	283120922089,00
Livetube	188673549830,00
Dynabox	236186082003,00
Eayo	188850285736,00
Thoughtstorm	188918712105,00
Wikizz	94554352576,00
Browseblab	283041012034,00

Pos: 5530 (row: 225, col: 24) U_bielas 1 query, 19,497 sec., 21 rows laby.sql

QUERY VERBOSE PLAN:

Node 1.
[SPU Sequential Scan table "PAYMENTREPORTDETAIL_REPORTID" as "PD" {(PD.PAYMENT_REPORT)}]
-- Estimated Rows = 599426939, Width = 8, Cost = 0.0 .. 11718.8, Conf = 90.0 (FACT)
Restrictions:
(PD.PAYMENT_REPORT NOTNULL)
Projections:
1:PD.BONUS 2:PD.PAYMENT_REPORT
Cardinality:
PD.PAYMENT_REPORT 100.0M (JIT)
[HashIt for Join]
Node 2.
[SPU Sequential Scan table "COMPANYHEADQUARTER" as "CH" {(CH."ID")}]
-- Estimated Rows = 43, Width = 8, Cost = 0.0 .. 0.0, Conf = 100.0
Restrictions:
((CH."ID" NOTNULL) AND (CH.COMPANY NOTNULL))
Projections:
1:CH."ID" 2:CH.COMPANY
[HashIt for Join]
Node 3.
[SPU Sequential Scan table "COMPANY" as "C" {(C."ID")}]
-- Estimated Rows = 22, Width = 54, Cost = 0.0 .. 0.0, Conf = 80.0
Restrictions:
(C."ID" NOTNULL)
Projections:
1:C.COMPANY_NAME 2:C."ID"
Node 4.
[SPU Hash Join Stream "Node 3" with Temp "Node 2" {(C."ID")}]
-- Estimated Rows = 22, Width = 54, Cost = 0.0 .. 0.0, Conf = 64.0
Restrictions:
(C."ID" = CH.COMPANY)
Projections:
1:CH."ID" 2:C.COMPANY_NAME
Cardinality:
CH."ID" 22 (Adjusted)
CH.COMPANY 22 (Adjusted)
[SPU Broadcast]
[HashIt for Join]
Node 5.
[SPU Sequential Scan table "PAYMENTREPORT" as "P" {(P."ID")}]
-- Estimated Rows = 99935252, Width = 8, Cost = 0.0 .. 1139.2, Conf = 90.0 [BT: MaxPages=890 TotalPages=19580] (JIT-Stats)

Restrictions:
((P."ID" NOTNULL) AND (P.COMPANY_HEADQUARTER NOTNULL))
Projections:
1:P."ID" 2:P.COMPANY_HEADQUARTER
Cardinality:
P.COMPANY_HEADQUARTER 43 (Adjusted)
Node 6.
[SPU Hash Join Stream "Node 5" with Temp "Node 4" {(P."ID")}]
-- Estimated Rows = 51129664, Width = 54, Cost = 0.0 .. 10451.4, Conf = 51.2
Restrictions:
(P.COMPANY_HEADQUARTER = CH."ID")
Projections:
1:C.COMPANY_NAME 2:P."ID"
Cardinality:
CH."ID" 22 (Adjusted)
CH.COMPANY 22 (Adjusted)
P.COMPANY_HEADQUARTER 22 (Adjusted)
Node 7.
[SPU Hash Join Stream "Node 6" with Temp "Node 1" {(PD.PAYMENT_REPORT,P."ID")}]
-- Estimated Rows = 302930461, Width = 54, Cost = 36343.7 .. 76782.2, Conf = 46.1
Restrictions:
(P."ID" = PD.PAYMENT_REPORT)
Projections:
1:C.COMPANY_NAME 2:PD.BONUS
Cardinality:
CH.COMPANY 22 (Adjusted)
Node 8.
[SPU Group]
-- Estimated Rows = 22, Width = 54, Cost = 36343.7 .. 83667.0, Conf = 0.0
Projections:
1:C.COMPANY_NAME 2:PD.BONUS
[SPU Return]
[HOST Merge Group]
Node 9.
[Host Aggregate]
-- Estimated Rows = 22, Width = 66, Cost = 36343.7 .. 83667.0, Conf = 0.0
Projections:
1:C.COMPANY_NAME 2:SUM(PD.BONUS)
[Host Return]

```
227  select c.company_name, sum(bonus)
228  from paymentreportdetail as pd join paymentreport as p
229  on pd.payment_report = p.id join companyheadquarter as ch
230  on ch.id = p.company_headquarter  join company as c
231  on c.id = ch.company
232  group by c.company_name
233
```

Output | Result 1

Drag a column header here to group by that column.

COMPANY_NAME	SUM
Yozio	283449163204,00
Browseblab	283041012034,00
Dynabox	236186082003,00
Shuffletag	283224968548,00
Eayo	188850285736,00
Youspan	283162689779,00
Feedbug	283120922089,00
Realpoint	94420412810,00

Pos: 5794 (row: 232, col: 24) U_bielas 1 query, 31,465 sec., 21 rows * laby.sql

QUERY VERBOSE PLAN:

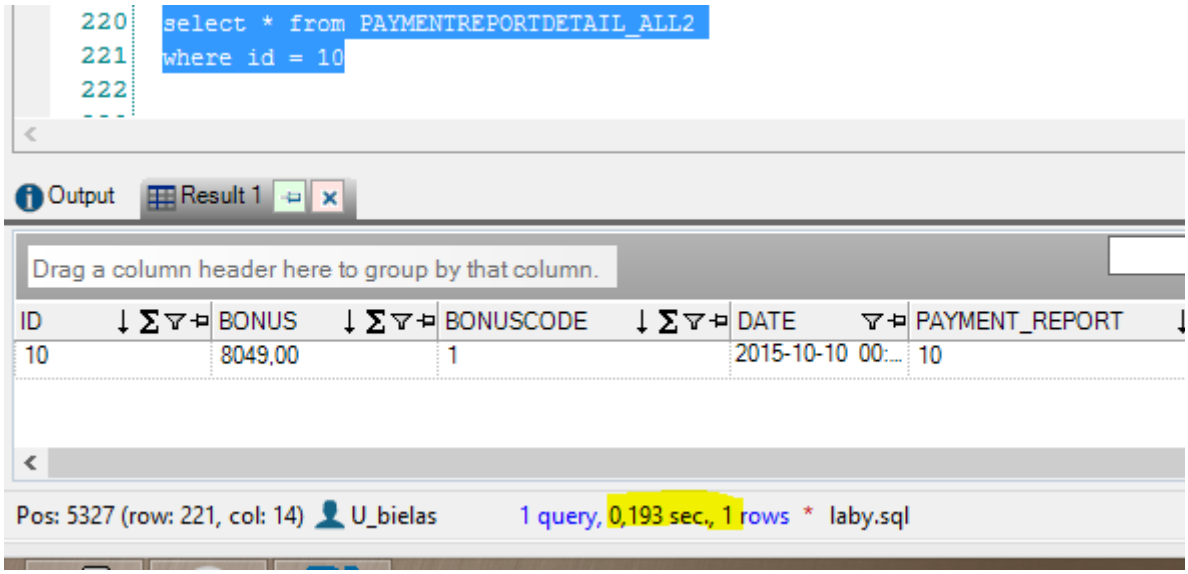
Node 1.
[SPU Sequential Scan table "PAYMENTREPORTDETAIL" as "PD" {(PD."ID")}]
-- Estimated Rows = 599005119, Width = 8, Cost = 0.0 .. 11718.8, Conf = 90.0 (FACT)
Restrictions:
(PD.PAYMENT_REPORT NOTNULL)
Projections:
1:PD.BONUS 2:PD.PAYMENT_REPORT
Cardinality:
PD.PAYMENT_REPORT 100.0M (JIT)
[SPU Distribute on {(PD.PAYMENT_REPORT)}]
[HashIt for Join]
Node 2.
[SPU Sequential Scan table "COMPANYHEADQUARTER" as "CH" {(CH."ID")}]
-- Estimated Rows = 43, Width = 8, Cost = 0.0 .. 0.0, Conf = 100.0
Restrictions:
((CH."ID" NOTNULL) AND (CH.COMPANY NOTNULL))
Projections:
1:CH."ID" 2:CH.COMPANY
[HashIt for Join]
Node 3.
[SPU Sequential Scan table "COMPANY" as "C" {(C."ID")}]
-- Estimated Rows = 22, Width = 54, Cost = 0.0 .. 0.0, Conf = 80.0

Restrictions:
(C."ID" NOTNULL)
Projections:
1:C.COMPANY_NAME 2:C."ID"
Node 4.
[SPU Hash Join Stream "Node 3" with Temp "Node 2" {(C."ID")}]
-- Estimated Rows = 22, Width = 54, Cost = 0.0 .. 0.0, Conf = 64.0
Restrictions:
(C."ID" = CH.COMPANY)
Projections:
1:CH."ID" 2:C.COMPANY_NAME
Cardinality:
CH."ID" 22 (Adjusted)
CH.COMPANY 22 (Adjusted)
[SPU Broadcast]
[HashIt for Join]
Node 5.
[SPU Sequential Scan table "PAYMENTREPORT" as "P" {(P."ID")}]
-- Estimated Rows = 99935252, Width = 8, Cost = 0.0 .. 1139.2, Conf = 90.0 [BT: MaxPages=890 TotalPages=19580] (JIT-Stats)
Restrictions:
((P."ID" NOTNULL) AND (P.COMPANY_HEADQUARTER NOTNULL))
Projections:
1:P."ID" 2:P.COMPANY_HEADQUARTER
Cardinality:
P.COMPANY_HEADQUARTER 43 (Adjusted)
Node 6.
[SPU Hash Join Stream "Node 5" with Temp "Node 4" {(P."ID")}]
-- Estimated Rows = 51129664, Width = 54, Cost = 0.0 .. 10451.4, Conf = 51.2
Restrictions:
(P.COMPANY_HEADQUARTER = CH."ID")
Projections:
1:C.COMPANY_NAME 2:P."ID"
Cardinality:
CH."ID" 22 (Adjusted)
CH.COMPANY 22 (Adjusted)
P.COMPANY_HEADQUARTER 22 (Adjusted)
Node 7.
[SPU Hash Join Stream "Node 6" with Temp "Node 1" {(PD.PAYMENT_REPORT,P."ID")}]
-- Estimated Rows = 302717287, Width = 54, Cost = 36327.2 .. 86000.2, Conf = 46.1
Restrictions:
(P."ID" = PD.PAYMENT_REPORT)
Projections:
1:C.COMPANY_NAME 2:PD.BONUS

Cardinality:
CH.COMPANY 22 (Adjusted)
Node 8.
[SPU Group]
-- Estimated Rows = 22, Width = 54, Cost = 36327.2 .. 92880.1, Conf = 0.0
Projections:
1:C.COMPANY_NAME 2:PD.BONUS
[SPU Return]
[HOST Merge Group]
Node 9.
[Host Aggregate]
-- Estimated Rows = 22, Width = 66, Cost = 36327.2 .. 92880.1, Conf = 0.0
Projections:
1:C.COMPANY_NAME 2:SUM(PD.BONUS)
[Host Return]

W obu powyższych sytuacjach pojawia się wzorzec, o którym wspomniano w zad.7:
tam gdzie jest konieczność zastosowania redystrybucji, negatywnie (czasowo) odbija się to na wydajności.

9. ciekawostka:



QUERY VERBOSE PLAN:

Node 1.

[SPU Sequential Scan table "PAYMENTREPORTDETAIL_ALL2" {(PAYMENTREPORTDETAIL_ALL2."ID")}]

-- Estimated Rows = 32000000, Width = 24, Cost = 0.0 .. 68181.8, Conf = 80.0

Restrictions:

(PAYMENTREPORTDETAIL_ALL2."ID" = 10)

Projections:

1:PAYMENTREPORTDETAIL_ALL2."ID" 2:PAYMENTREPORTDETAIL_ALL2.BONUS

3:PAYMENTREPORTDETAIL_ALL2.BONUSCODE

4:PAYMENTREPORTDETAIL_ALL2.DATE

5:PAYMENTREPORTDETAIL_ALL2.PAYMENT_REPORT

[SPU Return]

[Host Return]

```
223  select * from PAYMENTREPORTDETAIL_ALL2
224  where id between  1000000 and 1000050
225
```

Output | Result 1

Drag a column header here to group by that column.

ID	BONUS	BONUSCODE	DATE	PAYMENT_R
1000015	6525,00	1	2015-11-06 00:...	1000015
1000037	4309,00	1	2015-11-06 00:...	1000037
1000021	7596,00	0	2015-10-12 00:...	1000021
1000043	2527,00	2	2015-10-10 00:...	1000043
1000047	7286,00	2	2015-10-06 00:...	1000047
1000003	8644,00	2	2015-11-20 00:...	1000003
1000025	5035,00	2	2015-10-13 00:	1000025

Pos: 5422 (row: 225, col: 1) U_bielas 1 query, 0,339 sec., 51 rows * laby.sql

QUERY VERBOSE PLAN:

Node 1.

[SPU Sequential Scan table "PAYMENTREPORTDETAIL_ALL2" {(PAYMENTREPORTDETAIL_ALL2."ID")}]

-- Estimated Rows = 51, Width = 24, Cost = 0.0 .. 68181.8, Conf = 64.0

Restrictions:

((PAYMENTREPORTDETAIL_ALL2."ID" <= 1000050) AND (PAYMENTREPORTDETAIL_ALL2."ID" >= 1000000))

Projections:

1:PAYMENTREPORTDETAIL_ALL2."ID" 2:PAYMENTREPORTDETAIL_ALL2.BONUS

3:PAYMENTREPORTDETAIL_ALL2.BONUSCODE

4:PAYMENTREPORTDETAIL_ALL2.DATE

5:PAYMENTREPORTDETAIL_ALL2.PAYMENT_REPORT

[SPU Return]

[Host Return]

Czasy wyszukiwania jednego rekordu są porównywalne z czasem wyszukiwania 50 rekordów. Jest to prawdopodobnie zasługa zastosowanych zone map.