

RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

3.praktiskais darbs

**Datu glabāšana operatīvajā atmiņā (in memory database)**

**kolonu formātā**

Izstrādāja: Natans Šalamberidze

171RMC203

Pārbaudīja: lektors Jānis Eiduks

2023./2024. māc. Gads

**Darba uzdevums**

1. Datu bāzes sistēmas globālajā atmiņā (operatīvajā atmiņā) jāizveido atmiņas apgabals (buferis), kurā varēs ierakstīt datu tabulas un materializētos skatus kolonu formātā. Varēs izveidot kolonveida glabātuvi (In Memory Database).  Tāpēc šo datu glabātuvi sauc par kolonu glabātuvi (column datastore). Lai to izdarītu jānomaina sistēmas parametrs INMEMORY\_SIZE uz vēlamo kolonu glabātuves atmiņas apjomu (parasi 2G, bet jāapskata sava datora iespējas). Pēc tam obligāti jāaizver datu bāzes vadības instance (SHUTDOWN) un no jauna tā jāstartē. Lai pārliecinātos, ka kolonu glabātuve izveidota, jāapskatās sistēmas parametra INMEMORY\_SIZE vērtība, vai tā ir > 0.
2. Ja otrajā darbā izveidota apjomīga faktu tabula (>= 100000 ierakstu) un loģiski sakārtos un izprotamas dimensiju tabulas, tās var izmantot turpmākiem uzdevumiem. Šajā situācijā var arī izmantot 2. darba vicājumus un materializētos skatus. Pretējā gadījumā jāizveido divas liela apjoma saistītas tabulas.
3. Darba pamatuzdevums ir: noskaidrot un salīdzināt vienu un to pašu vaicājumu **ātrdarbības atšķirības, ja tiek izmantotas parastas tabulas un tabulas kolonveida glabātuvē (In Memory Database) kā arī materializēto skatu izmantošanas  ātrdarbības atšķirības, ja tie izvietoti parastā atmiņā un kolonveida glabātuvē. Iegūtie  rezultāti jāsagrupē teksts redaktora WORD tabulās, lai var izdarīt analīzi un secinājumus.**
4. Tātad jāiveido divas liela apjoma parastas tabulas un šādas pašas tabulas kolonu glabātuvē.
5. Jāveic 3 vaicājumi gan parastām tabulām, gan tabulām kolonu glabātuvē. Jāiegūst attiecīgie izpildes laiki priekš izvērtēšanas.
6. Jāizveido 2 materializētie parastie materializētie skati un tādi paši kolonu glabātuvē. Jānosaka un jāfiksē to ātrdarbības rādītāji.
7. Jāsalīdzina iegūtie rezultāti kolonu glabātuvē un parastajā datu bāzē (rezultāti darba atskaitē jāsagrupē tabulā).
8. Jāveic izvērtējums un jāizdara secinājumi (ko var secināt pēc padarītā darbā iegūtajiem rezultātiem).
9. Un lūdzu domājiet, kad veidojiet vaicājumus un analizējiet rezultātus.
10. Ceru, ka Jums veiksies labāk, kā ar otro darbu.

2. Ja otrajā darbā izveidota apjomīga faktu tabula (>= 100000 ierakstu) un loģiski sakārtos un izprotamas dimensiju tabulas, tās var izmantot turpmākiem uzdevumiem. Šajā situācijā var arī izmantot 2. darba vicājumus un materializētos skatus. Pretējā gadījumā jāizveido divas liela apjoma saistītas tabulas.

Tabulas, materializētie skati, dimensijas un vaicājumi tiek izmantotas no iepriekšēja 2 PD. Tabulām būs izveidotas kopijas bez indeksiem. Veicot 3 vaicājumus no 2PD, mēs parbaudīsim kadas būs atšķirības starp parastam tabulam ar indeksiem un tabulam kuri būs ievietoti atmiņā (INMEMORY).

CREATE table DZIMUMS\_IM as (SELECT \* from DZIMUMS);

CREATE table KLIENTI\_IM as (SELECT \* from KLIENTI);

CREATE table GADS\_IM as (SELECT \* from GADS);

CREATE table LAIKS\_IM as (SELECT \* from LAIKS);

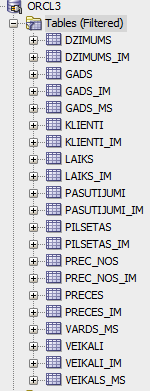
CREATE table PILSETAS\_IM as (SELECT \* from PILSETAS);

CREATE table PREC\_NOS\_IM as (SELECT \* from PREC\_NOS);

CREATE table PRECES\_IM as (SELECT \* from PRECES);

CREATE table VEIKALI\_IM as (SELECT \* from VEIKALI);

CREATE table PASUTIJUMI\_IM as (SELECT \* from PASUTIJUMI);



Pēc tabulas izveidošanas vajag arī izveidot dimensijas.

CREATE dimension DIM\_VEIKALI\_IM

level L\_PIL\_IM is PILSETAS\_IM.PILS\_NOS

level L\_VEIK\_IM is VEIKALI\_IM.VEIKALA\_NOS

hierarchy HIERARCHY\_veikali\_IM(

L\_VEIK\_IM child of

L\_PIL\_IM

JOIN KEY(VEIKALI\_IM.pilsetas\_f ) REFERENCES L\_PIL\_IM)

ATTRIBUTE L\_PIL\_IM DETERMINES PILSETAS\_IM.PILS\_ID

ATTRIBUTE L\_VEIK\_IM DETERMINES VEIKALI\_IM.VEIKALA\_ID;

CREATE dimension DIM\_PRECES\_IM

level L\_PRNOS\_IM is PREC\_NOS\_IM.prec\_nos

level L\_PRECES\_IM is PRECES\_IM.CENA

hierarchy HIERARCHY\_preces\_IM(

L\_PRECES\_IM child of

L\_PRNOS\_IM

JOIN KEY(PRECES\_IM.PREC\_NOS\_F ) REFERENCES L\_PRNOS\_IM)

ATTRIBUTE L\_PRECES\_IM DETERMINES PRECES\_IM.PREC\_ID

ATTRIBUTE L\_PRNOS\_IM DETERMINES PREC\_NOS\_IM.PREC\_NOS\_ID;

CREATE dimension DIM\_LAIKS\_IM

level L\_GADS\_IM is GADS\_IM.GADS\_ID

level L\_DATE\_IM is LAIKS\_IM.DATUMS

hierarchy HIERARCHY\_laiks\_IM(

L\_DATE\_IM child of

L\_GADS\_IM

JOIN KEY(LAIKS\_IM.GADS\_F) REFERENCES L\_GADS\_IM)

ATTRIBUTE L\_GADS\_IM DETERMINES GADS\_IM.GADS\_ID

ATTRIBUTE L\_DATE\_IM DETERMINES LAIKS\_IM.DATE\_ID;

CREATE dimension DIM\_Klienti\_IM

level L\_DZIM\_IM is DZIMUMS\_IM.DZIMUMS

level L\_KLT\_IM is KLIENTI\_IM.KLIENTA\_ID

hierarchy HIERARCHY\_klienti\_IM(

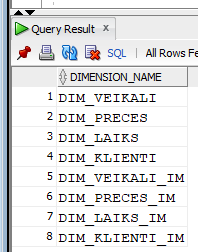
L\_KLT\_IM child of

L\_DZIM\_IM

JOIN KEY(KLIENTI\_IM.DZIMUMS\_F) REFERENCES L\_DZIM\_IM)

ATTRIBUTE L\_DZIM\_IM DETERMINES DZIMUMS\_IM.DZIM\_ID

ATTRIBUTE L\_KLT\_IM DETERMINES KLIENTI\_IM.KLIENTA\_ID;



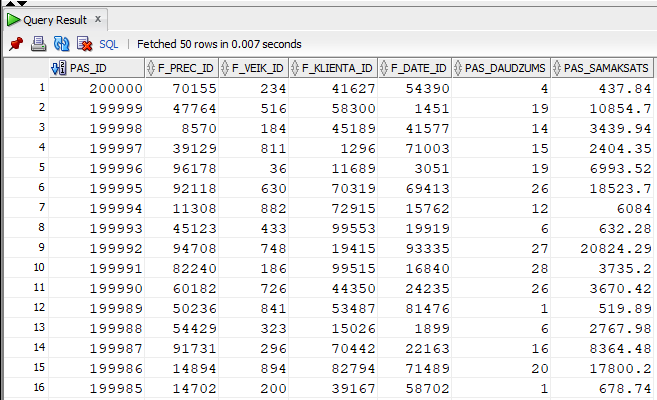
Izmantojot vaicājumu zemāk, ir redzams ka 4 jaunas dimensijas veiksmīgi izveidotas.

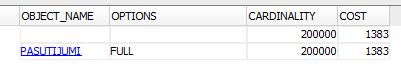
select DIMENSION\_NAME

from ALL\_DIMENSIONS;

Mēs esam pārliecinājušies, ka izmēri ir izveidoti, tagad mēs pārbaudīsim jaunu tabulu klātbūtni, izmantojot vienkāršu vaicājumu

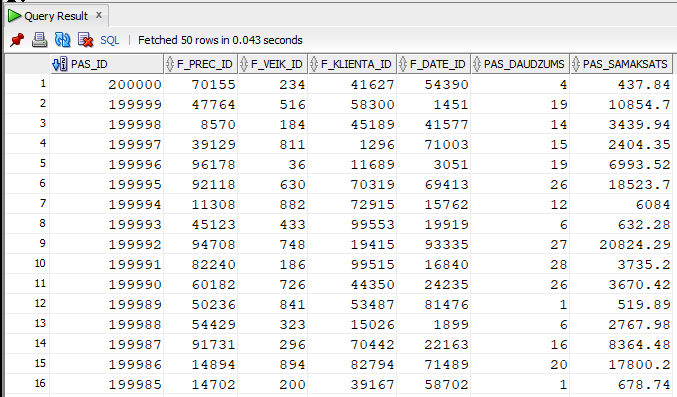
SELECT \* FROM pasutijumi order by F\_veik\_ID;

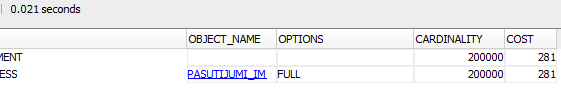




Un izpildām to pašu vaicājumu kopijas tabulai.

SELECT \* FROM pasutijumi\_im order by F\_veik\_ID;



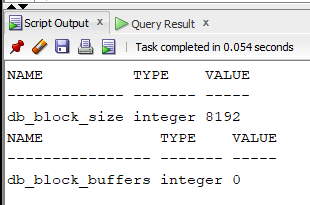


Redzam, ka tabulas ir identiskas, bet tomēr izmaksa starp tiem ir ļoti atšķirīga. Manuprāt, tas ir tāpēc, ka ORACLE ir ļoti gudra RDBPS. Tā atceras iepriekšējo vaicājumu un saprot, kā tā ir identiska kopija oriģinālai tabulai, tāpēc rezultātu atrod ātrāk. Bija arī domas par to, ka tās tabulas nebija izveidotas no pirmās reizes pareizi. Iepriekšējiem tabulām bija darītas daudz izmaiņas, visādas koriģēšanas, rediģēšanas, līdz ar to atmiņas šūnas, diskā vietā ir izkaisīti, bet tabulas kopijas ir jaunas, un pēdējās izveidotas tabulas, tāpēc piekļuve tiem ir ātrāka. Bet es varu kļūdīties.

Lai sāktu pildīt darbu, pārbaudīsim DB buferi. Mēs varam to izdarīt ar komandu:

show parameter DB\_BLOCK\_SIZE;

show parameter DB\_BLOCK\_BUFFERS;

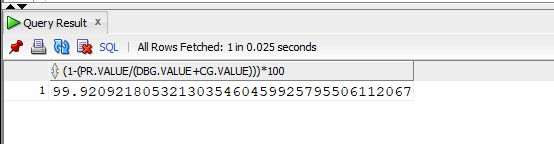


Un uzreiz pārbaudīsim datu bāzes bloku bufera izmantošanas efiktivitāti ar komandu:

select (1-(pr.value/(dbg.value+cg.value)))\*100

from v$sysstat pr, v$sysstat dbg, v$sysstat cg

where pr.name = 'physical reads' and dbg.name = 'db block gets' and cg.name = 'consistent gets';

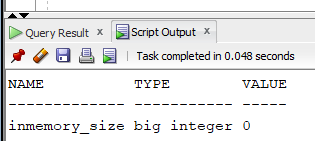


Ir skaidri redzam, ka DB bloka izmantošana ir efektīva, jo radītās nepārsniedz 80%.

**Bufera izveidošana**

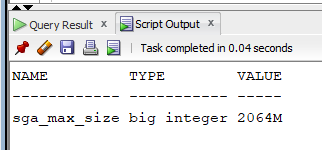
Vispirms pārbaudām atmiņas resursu, vai tas ir vai nav izdalīts.

show PARAMETER INMEMORY\_SIZE;



Tas nozīmē, ka nevar izmantot IMDB. Lai iespējotu tās izmantošanu, pirms datu bāzes **instances restartēšanas** jānorāda inicializācijas parametra INMEMORY\_SIZE vērtība, kas nav nulle. Var arī dinamiski palielināt INMEMORY\_SIZE lieluma iestatījumu, izmantojot priekšrakstu ALTER SYSTEM.

Apskatīsim SGA maksimālo apgabala izmēru kuru ir iespējams iedod sistēmai.



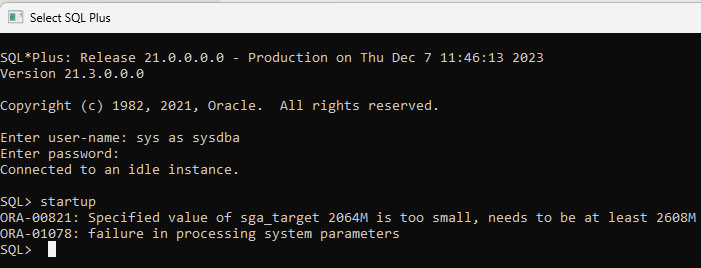
Tad palielinām atmiņu ar komandu

ALTER SYSTEM SET INMEMORY\_SIZE=2G SCOPE=SPFILE;

Tālāk ejam uz SQL Plus.

Izslēdzam datu bāzi ar komandu

shutdown immediate;



Mēs redzam kļūdu, jo no SGA\_TARGET inicializācijas parametra iestatījuma tiek atņemts In-Memory apgabals, bet SGA\_TARGET ir mazs. Vajag to palielināt.

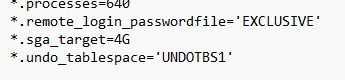
Ar komandu izveidojam PFILE kur mēs varēsim nomainīt sga\_target vērtību.

CREATE PFILE = 'ORACLE\_HOME\_PATH' FROM SPFILE;

Manā gadījumā

CREATE PFILE = 'C:\BD\_2023\dbs\custom\_init.ora' FROM SPFILE

Tajā failā mainām sga\_target vērtību uz 4G un saglābām.



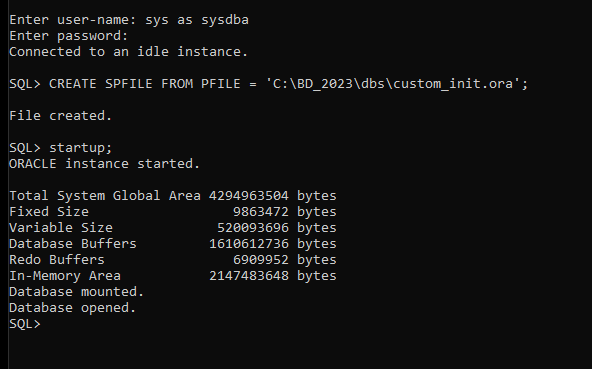
CREATE SPFILE FROM PFILE = 'ORACLE\_HOME\_PATH\_TO\_CREATED\_PFILE';

Manā gadījumā

CREATE SPFILE FROM PFILE = 'C:\BD\_2023\dbs\custom\_init.ora';

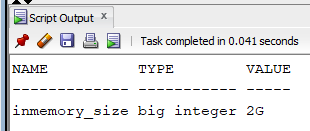
Pēc tam palaidām datu bāzi ar komandu

Startup;



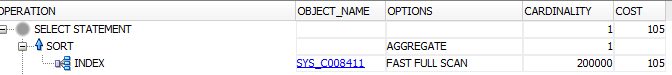
Datubāze tika veiksmīgi startēta un ir redzams, ka In-Memory area izmērs ir 2GB. To var arī pārbaudīt ar vaicājumu

SHOW PARAMETER INMEMORY\_SIZE;

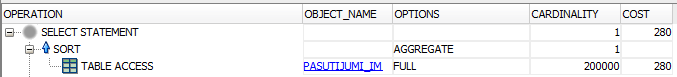


Viss ir izveidots, bet pirms tam ka ievietot tabulas operatīvajā atmiņās apgabalā, es gribu veikt vel divas vaicājumus ar agregācijas funkciju (count).

select /\* +FULL(PASUTIJUMI\_IM) NO\_PARALLEL(PASUTIJUMI\_IM)\*/COUNT(\*) from pasutijumi;



select /\* +FULL(PASUTIJUMI\_IM) NO\_PARALLEL(PASUTIJUMI\_IM)\*/COUNT(\*) from pasutijumi\_im;



Pēc rezultāta ir redzams, ka tagad COST vērtība oriģinālai tabulai ir mazāks, jo tiek izmantoti indeksi, kuri bija izveidotas 2. praktiskā darbā, bet kopijas tabulām indeksi nav pievienoti.

Ir pienācis laiks pārvietot tabulas operatīvajā atmiņā. To var izdarīt ar komandam:

ALTER TABLE pasutijumi\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

ALTER TABLE dzimums\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

ALTER TABLE gads\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

ALTER TABLE klienti\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

ALTER TABLE laiks\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

ALTER TABLE pilsetas\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

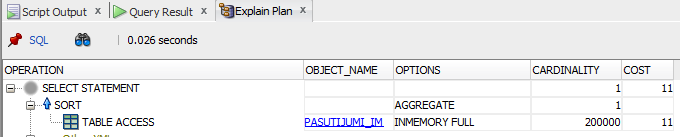
ALTER TABLE prec\_nos\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

ALTER TABLE preces\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

ALTER TABLE veikali\_IM INMEMORY MEMCOMPRESS FOR CAPACITY LOW PRIORITY HIGH;

Izpildām to pašu vaicājumu ar agregācijas funkciju:

select /\* +FULL(PASUTIJUMI\_IM) NO\_PARALLEL(PASUTIJUMI\_IM)\*/COUNT(\*) from pasutijumi\_im;



Pēc rezultāta ir redzams ka tabula nolasa datus no atmiņas un tā ir visātrākā. COST vērtība samazinājās no 280 uz 11. Tas ir ļoti labs rezultāts. Nolasīšana no atmiņas ir efektīvāk nekā tabulas indeksēšana, arī tur COST vērtība bija 105, kura ir > 11.

Jāveic 3 vaicājumi gan parastām tabulām, gan tabulām kolonu glabātuvē. Jāiegūst attiecīgie izpildes laiki priekš izvērtēšanas. Vaicājumi tiek paņemti no iepriekšēja 2. praktiska darba, jo iepriekšējā darbā bija skaidri redzamas atšķirības, neskatoties uz to ka vaicājumi nav sarežģīti. Cerams ka tagad ari varēsim ieraudzīt labus rezultātus.

1. Vaicājums – oriģinālai tabulai.

Ir sasummēti visi ieraksti: Cena, Daudzums, Samaksa visiem veikaliem, visas pilsētas, katrā gadā no 2000-2023.

select l.gads\_f, sum(pr.cena) as Preces\_cenas\_summa ,

sum(p.pas\_daudzums) as Summa, sum(p.pas\_samaksats) as Daudzums

from pasutijumi p, laiks l, preces pr

where

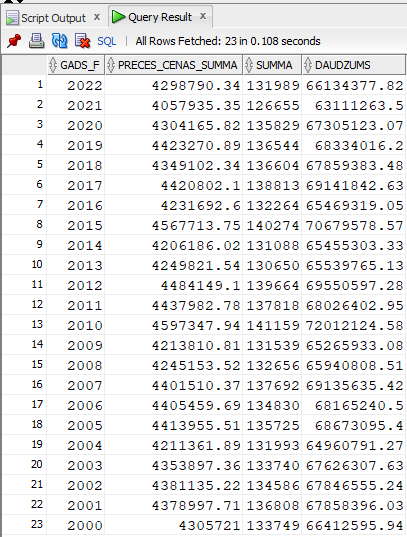
p.f\_date\_ID = l.date\_id

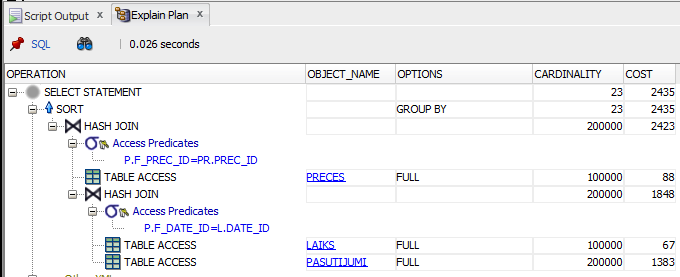
AND

p.f\_prec\_id = pr.prec\_id

group by l.gads\_f

order by l.gads\_f desc;





1. Vaicājums – atmiņas apgabala tabulai.

select l.gads\_f, sum(pr.cena) as Preces\_cenas\_summa ,

sum(p.pas\_daudzums) as Summa, sum(p.pas\_samaksats) as Daudzums

from pasutijumi\_im p, laiks\_im l, preces\_im pr

where

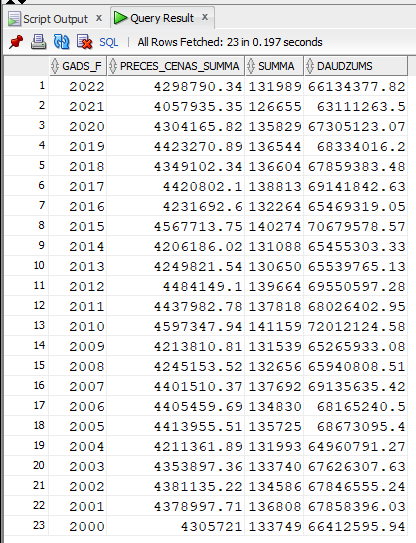
p.f\_date\_ID = l.date\_id

AND

p.f\_prec\_id = pr.prec\_id

group by l.gads\_f

order by l.gads\_f desc;



A screenshot of a computer

Description automatically generated

Var secināt ka tabulas datu rezultāti ir identiskas, bet COST vērtība ir samazinājās, kas ir ļoti labi un nozīme ka dati no atmiņas nolasās daudz ātrāk nekā parastai tabulai.

1. Vaicājums – oriģinālai tabulai.

Ir iegūta informācija par samaksātu summu pasūtījumiem un pasūtījuma daudzumu, pa veikaliem kur, veikala nosaukums sākas ar -**Vu**

select veikala\_nos, sum(pas\_samaksats) as Summa

from pasutijumi p, veikali v, pilsetas pils

where

p.f\_veik\_ID = v.veikala\_id AND v.pilsetas\_f = pils.pils\_id

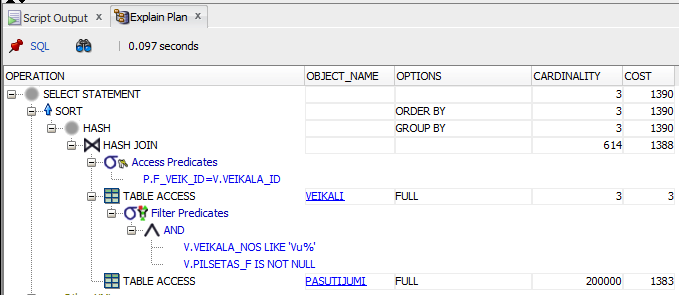
AND v.veikala\_nos LIKE 'Vu%'

group by veikala\_nos

order by Summa desc;

A screenshot of a computer

Description automatically generated



1. Vaicājums – atmiņas apgabala tabulai.

select veikala\_nos, sum(pas\_samaksats) as Summa

from pasutijumi\_im p, veikali\_im v, pilsetas\_im pils

where

p.f\_veik\_ID = v.veikala\_id AND v.pilsetas\_f = pils.pils\_id

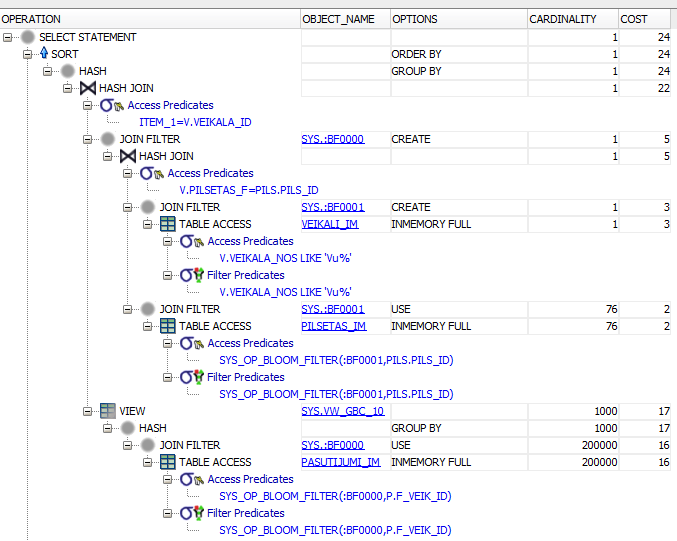
AND v.veikala\_nos LIKE 'Vu%'

group by veikala\_nos

order by Summa desc;

A screenshot of a computer

Description automatically generated



Otrajā gadījumā COST samazinājās vel vairāk, saņemto datu ierakstu ir daudz mazāks nekā pirmajā vaicājumā.

1. Vaicājums – oriģinālai tabulai.

Ir iegūta informācija par samaksātu summu pasūtījumiem, pa klientiem kur, klienta vārds sākas ar -**Ma**

select vards, sum(pas\_samaksats) as Summa, sum(pas\_samaksats) as Daudzums

from pasutijumi p, klienti k

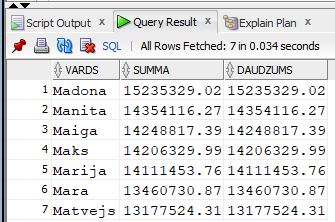
where

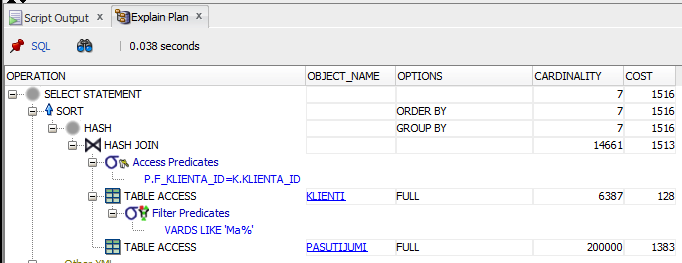
p.f\_klienta\_ID = k.klienta\_id

AND vards LIKE 'Ma%'

group by vards

order by Summa desc;





1. Vaicājums – atmiņas apgabala tabulai.

select vards, sum(pas\_samaksats) as Summa, sum(pas\_samaksats) as Daudzums

from pasutijumi\_im p, klienti\_im k

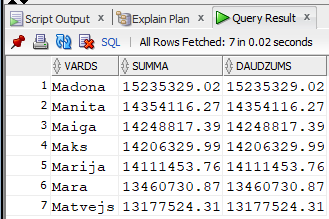
where

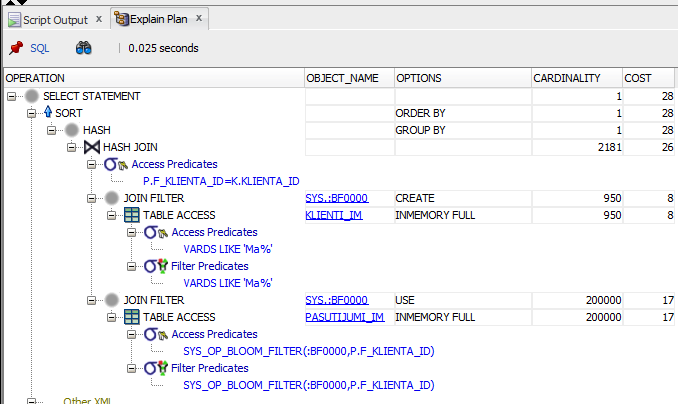
p.f\_klienta\_ID = k.klienta\_id

AND vards LIKE 'Ma%'

group by vards

order by Summa desc;





Trešajā gadījumā izmaksas COST arī samazinājās. Saņemto datu ierakstu ir arī nedaudz.

**Materializētie skati**

Jāizveido 2 materializētie parastie materializētie skati un tādi paši kolonu glabātuvē. Jānosaka un jāfiksē to ātrdarbības rādītāji. Materializētie skati arī ir paņemtas no iepriekšēja darba.

1. Materializētais skats

create materialized view GADS\_MS

BUILD IMMEDIATE

ENABLE QUERY REWRITE as

select l.gads\_f, sum(pr.cena) as Preces\_cenas\_summa ,

sum(p.pas\_daudzums) as Summa, sum(p.pas\_samaksats) as Daudzums

from pasutijumi p, laiks l, preces pr

where

p.f\_date\_ID = l.date\_id

AND

p.f\_prec\_id = pr.prec\_id

group by l.gads\_f

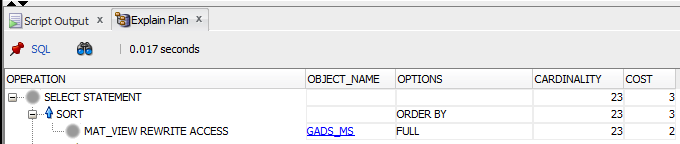
order by l.gads\_f desc;

analyze table pasutijumi COMPUTE STATISTICS;

analyze table laiks COMPUTE STATISTICS;

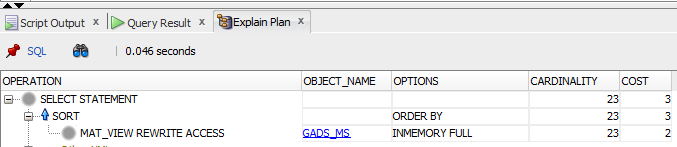
analyze table preces COMPUTE STATISTICS;

alter session set OPTIMIZER\_MODE = ALL\_ROWS;



Šo skatu ieliksim atmiņā ar komandu:

ALTER MATERIALIZED VIEW GADS\_MS INMEMORY PRIORITY HIGH;



Rezultātā nav atšķirības starp COST vērtībām, bet ir redzams, ka materializētais skats tiek veiksmīgi ielikts operatīvajā atmiņā.

1. Materializētais skats

create materialized view VEIKALS\_MS

BUILD IMMEDIATE

ENABLE QUERY REWRITE as

select veikala\_nos, sum(pas\_samaksats) as Summa

from pasutijumi p, veikali v, pilsetas pils

where

p.f\_veik\_ID = v.veikala\_id AND v.pilsetas\_f = pils.pils\_id

AND v.veikala\_nos LIKE 'Vu%'

group by veikala\_nos

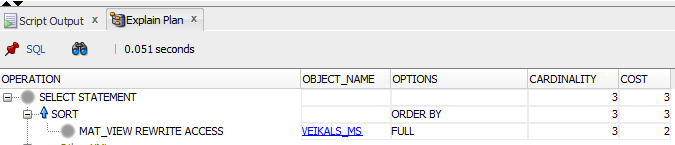
order by Summa desc;

analyze table pasutijumi COMPUTE STATISTICS;

analyze table veikali COMPUTE STATISTICS;

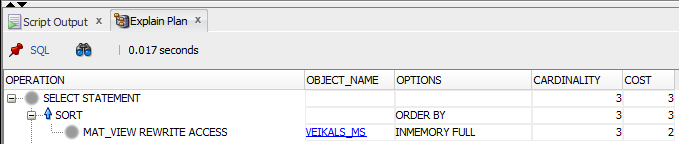
analyze table pilsetas COMPUTE STATISTICS;

alter session set OPTIMIZER\_MODE = ALL\_ROWS;



Šo skatu ieliksim atmiņā ar komandu:

ALTER MATERIALIZED VIEW VEIKALS\_MS INMEMORY PRIORITY HIGH;



1. Materializētais skats

create materialized view VARDS\_MS

BUILD IMMEDIATE

ENABLE QUERY REWRITE as

select vards, sum(pas\_samaksats) as Summa, sum(pas\_samaksats) as Daudzums

from pasutijumi p, klienti k

where

p.f\_klienta\_ID = k.klienta\_id

AND vards LIKE 'Ma%'

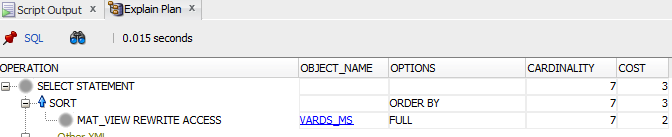
group by vards

order by Summa desc;

analyze table pasutijumi COMPUTE STATISTICS;

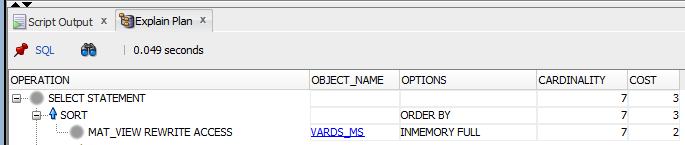
analyze table klienti COMPUTE STATISTICS;

alter session set OPTIMIZER\_MODE = ALL\_ROWS;



Šo skatu ieliksim atmiņā ar komandu:

ALTER MATERIALIZED VIEW VEIKALS\_MS INMEMORY PRIORITY HIGH;



Rezultātā mēs redzam, ka nevienā no variantiem COST Vērtība nav mainījusies. Es tam zinu tikai vienu izskaidrojumu. Materializētiem skatiem nav pietiekošs atmiņas apjoms, tam jābūt > 64K, līdz ar to dati nebija iekļauti kolonu krātuvē. Ja izveidot materializēto skatu ar vairākiem blokiem, rezultāts būs redzams un COST vērtība būs mazāk.

Rezultātu tabula.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | | 2 | | 3 | |
| Tabula | MatSkats | Tabula | MatSkats | Tabula | MatSkats |
| Parasta DB | 2435 | 3 | 1390 | 3 | 1516 | 3 |
| IN MEMORY | 928 | 3 | 24 | 3 | 28 | 3 |

**Secinājumi**

Ka bija teikts agrāk, atšķirības ir redzamas. Un priekšrocības ir atdotas operatīvai atmiņai, to var skaidri redzēt rezultātu tabulā. Vaicājumu COST vērtības samazinājās visām trim tabulām. Bija noskaidrots ka vaicājumi tabulām operatīvajā atmiņā COST izmaksas ir mazākas nekā tabulām ar izveidotiem indeksiem. No teorijas bija zināms, ka tā ari vajadzētu būt, mēs tikko par to pārliecinājāmies. Runājot par materializētiem skatiem, atšķirības starp rezultātiem es neesmu sasniedzis, jo tabulām kuram segments ir <64K uzlabojuma nebūs. Tabulām, kuras bija analizētas bija tieši šā problēma. Darba gaitā bija daudz problēmas un neskaidrības, ar kuriem man vajadzēja cīnīties. Bija kļūdas pieļautas arī neuzmanības dēļ. Piemēram ar to atmiņas palielināšanu. Es aizmirsu paskatīties sava datora spējas un kad aizveru datu bāzi es nevarēju to startēt, jo sga\_target vērtība nebija pietiekama. Vajadzēja veidot PFILE, lai to atrastu. Galu galā man izdevās palaist datu bāzi un strādāt tālāk. Darba gaitā es uzzināju daudz jaunas iespējas, kurus var veikt ar datu bāzi. Piemēram kā apskatīt DB segmentus, tabultelpas, kur apskatīt bloka izmērus un to efektivitāti. Beigās gribu atvainoties ka darbs nav nodots paredzētajā laika, bet neskatoties uz to, uzskatu, ka darbs nebija veltīgs.