

RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes, informācijas tehnoloģijas un enerģētikas fakultāte

3.praktiskais darbs

**SQL priekšrakstu paralēlā izpilde (daudzserveru procesi)**

Izstrādāja: Natans Šalamberidze

171RMC203

Pārbaudīja: lektors Jānis Eiduks

2023./2024. māc. Gads

Praktiskajā darbā jāiepazīstas ar datu paralēlo apstrādi (prosessing) un vaicājumu paralēlo izpildi (execution), ražotāja/patērētāja (produser/consumer) modeli un vaicājumu izpildes paralēlisma pakāpi.

1. Jānodefinē 2 savstarpēji loģiski saistītas tabulas. Jāuzraksta tabulu aizpildīšanas programmas ar datiem. Vienai tabulai 1000 rindu otrai 100 000 rindas. Jāaizpilda tabulas ar datiem. Var izmantot tabulas no iepriekšējiem darbiem. Darba sākumā jāattēlo tabulu struktūru.

2. Jādefinē 3 vaicājumi vienai tabulai. Vaicājumos jālieto klauzulas ORDER BY, GROUP BY, HAVING un agregāti.

3. Jādefinē 3 vaicājumi saistītām tabulām. Vaicājumos jālieto klauzulas ORDER BY, GROUP BY, HAVING un agregāti.

4. Izmantojot izpildes paralēlos serverus (norādot paralēlisma pakāpi) jāpārliecinās, ka ar tiem tiek iegūts labāks rezultāts.

5. Jāizpilda vaicājumi ar paralēliem serveriem un:

a) jāiegūst vaicājuma izpildes plāns;

b) jāizanalizē un jāpaskaidro izpildes plāna kolonu un to vērtību semantika jeb jēga Jūsu vaicājumiem;

c) jāuzzīme izpildes plāna paralēlo serveru darbības detalizēta diagramma un tās darbība jāpaskaidro.

6. Jāizveido vaicājumi, kuru izpildes plānos var redzēt, ka tiek izmantotas 3 dažādas izplatīšanas metodes (distribution methods).

7. Jāpaskaidro šo vaicājumu izpildes plāna darbība un jāizvērtē iegūtie rezultāti.

8. Kopējie secinājumi par izmantotajām tehnoloģijām.

**2. Jānodefinē 2 savstarpēji loģiski saistītas tabulas. Jāuzraksta tabulu aizpildīšanas programmas ar datiem. Vienai tabulai 1000 rindu otrai 100 000 rindas. Jāaizpilda tabulas ar datiem. Var izmantot tabulas no iepriekšējiem darbiem. Darba sākumā jāattēlo tabulu struktūru.**

CREATE TABLE customer (

id NUMBER,

name VARCHAR2(100),

age NUMBER,

phone\_number VARCHAR2(20)

)tablespace tablespace\_1;

CREATE TABLE payments (

id NUMBER,

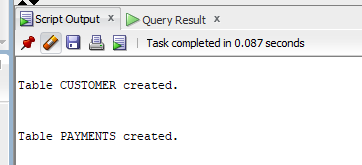
user\_id NUMBER,

invoice\_number VARCHAR2(200),

payment\_date DATE,

amount NUMBER

)tablespace tablespace\_2;



DECLARE

i NUMBER;

TYPE name\_array IS VARRAY(15) OF VARCHAR2(50);

TYPE surname\_array IS VARRAY(15) OF VARCHAR2(50);

first\_names name\_array := name\_array('Jānis', 'Pēteris', 'Māris', 'Andris', 'Āris', 'Artūrs', 'Miks', 'Lauris', 'Renārs', 'Raimonds', 'Kristaps', 'Edgars', 'Juris', 'Rihards', 'Andris');

surnames surname\_array := surname\_array('Bērziņš', 'Liepiņš', 'Ozoliņš', 'Saulītis', 'Kalniņš', 'Ķēniņš', 'Sējējs', 'Priedītis', 'Vītols', 'Liepa', 'Gulbis', 'Krūmiņš', 'Muižnieks', 'Lūsis', 'Šmits');

BEGIN

FOR i IN 1..1000 LOOP

INSERT INTO customer (id, name, age, phone\_number)

VALUES (i,

first\_names(TRUNC(DBMS\_RANDOM.VALUE(1, 15))) || ' ' || surnames(TRUNC(DBMS\_RANDOM.VALUE(1, 15))),

ROUND(DBMS\_RANDOM.VALUE(18, 80)),

TO\_CHAR('+3712' || ROUND(DBMS\_RANDOM.VALUE(1000000, 9999999)))

);

END LOOP;

FOR i IN 1..100000 LOOP

INSERT INTO payments (id, user\_id, invoice\_number, payment\_date, amount)

VALUES (i,

ROUND(DBMS\_RANDOM.VALUE(1, 1000)),

'Rekins - ' || TO\_CHAR(ROUND(DBMS\_RANDOM.VALUE(1000000, 9999999))),

SYSDATE - ROUND(DBMS\_RANDOM.VALUE(1, 365)),

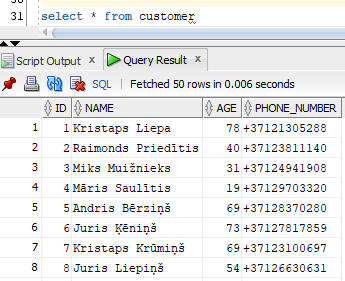
ROUND(DBMS\_RANDOM.VALUE(10, 1000), 2));

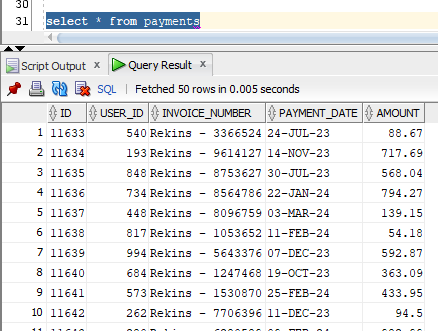
END LOOP;

COMMIT;

END;

/





**2. Jādefinē 3 vaicājumi vienai tabulai. Vaicājumos jālieto klauzulas ORDER BY, GROUP BY, HAVING un agregāti.**

***ORDER BY un GROUP BY ar agregātu:***

SELECT age, COUNT(\*) AS customer\_count

FROM customer

GROUP BY age

ORDER BY age;

***HAVING ar GROUP BY un agregātu:***

SELECT age, COUNT(\*) AS customer\_count

FROM customer

GROUP BY age

HAVING COUNT(\*) > 2;

***ORDER BY ar agregātu:***

SELECT id, name, age

FROM customer

ORDER BY age DESC;

**3. Jādefinē 3 vaicājumi saistītām tabulām. Vaicājumos jālieto klauzulas ORDER BY, GROUP BY, HAVING un agregāti.**

***ORDER BY un GROUP BY ar agregātu (izmantojot apvienotus datus no abām tabulām):***

SELECT c.name, SUM(p.amount) AS total\_payments

FROM payments p

JOIN customer c ON p.user\_id = c.id

GROUP BY c.name

ORDER BY total\_payments DESC;

***HAVING ar GROUP BY un agregātu (izmantojot apvienotus datus no abām tabulām):***

SELECT c.name, COUNT(\*) AS payment\_count

FROM payments p

JOIN customer c ON p.user\_id = c.id

GROUP BY c.name

HAVING COUNT(\*) > 5;

***ORDER BY ar agregātu (izmantojot apvienotus datus no abām tabulām):***

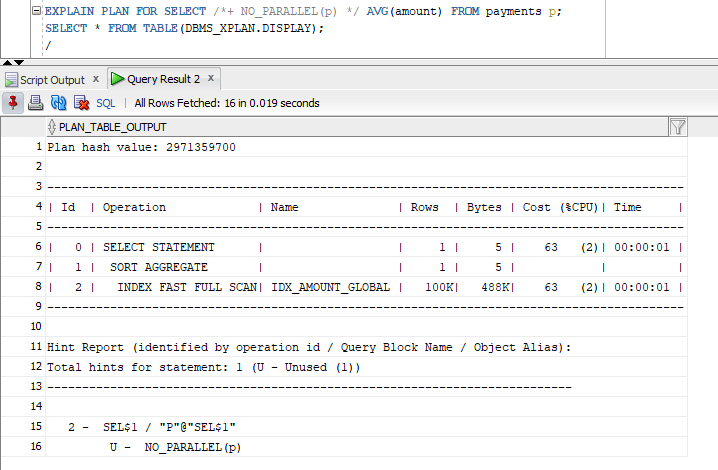
SELECT c.name, p.payment\_date, p.amount

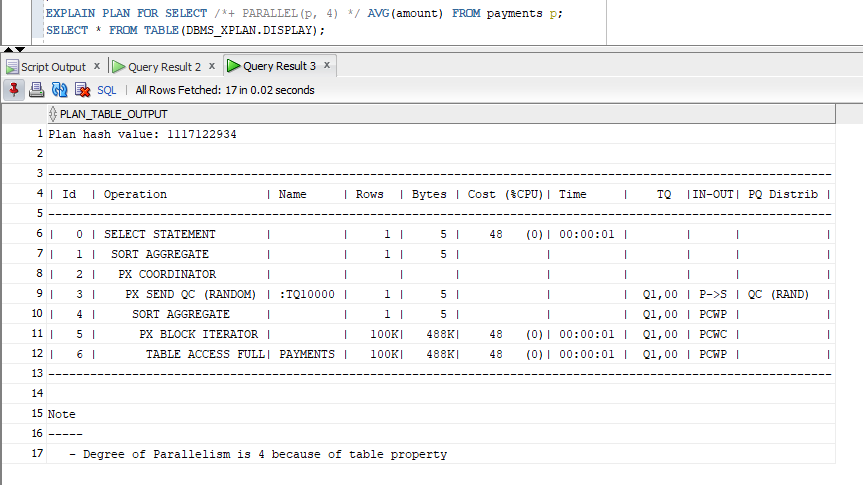
FROM payments p

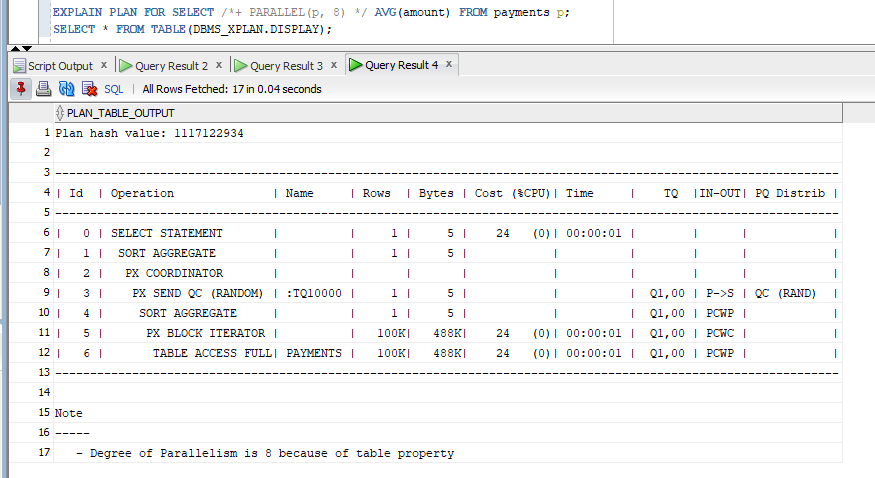
JOIN customer c ON p.user\_id = c.id

ORDER BY p.payment\_date DESC;

**4. Izmantojot izpildes paralēlos serverus (norādot paralēlisma pakāpi) jāpārliecinās, ka ar tiem tiek iegūts labāks rezultāts.**



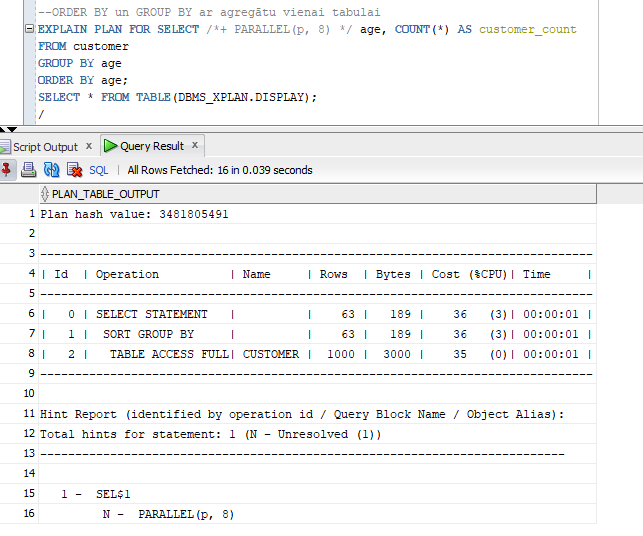




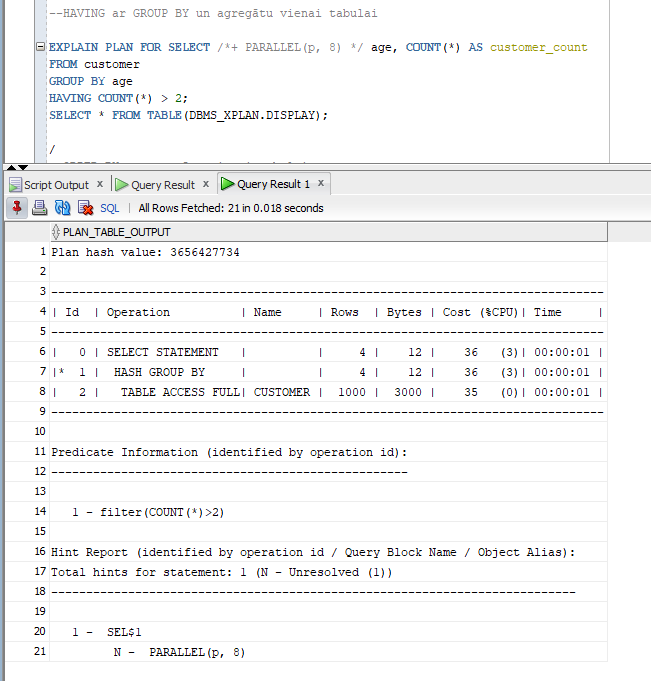
**5. Jāizpilda vaicājumi ar paralēliem serveriem un:**

**a) jāiegūst vaicājuma izpildes plāns;**

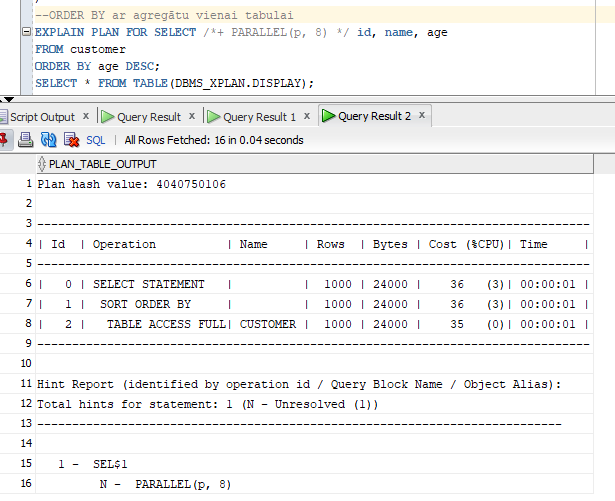
Vaicājums 2.1.:



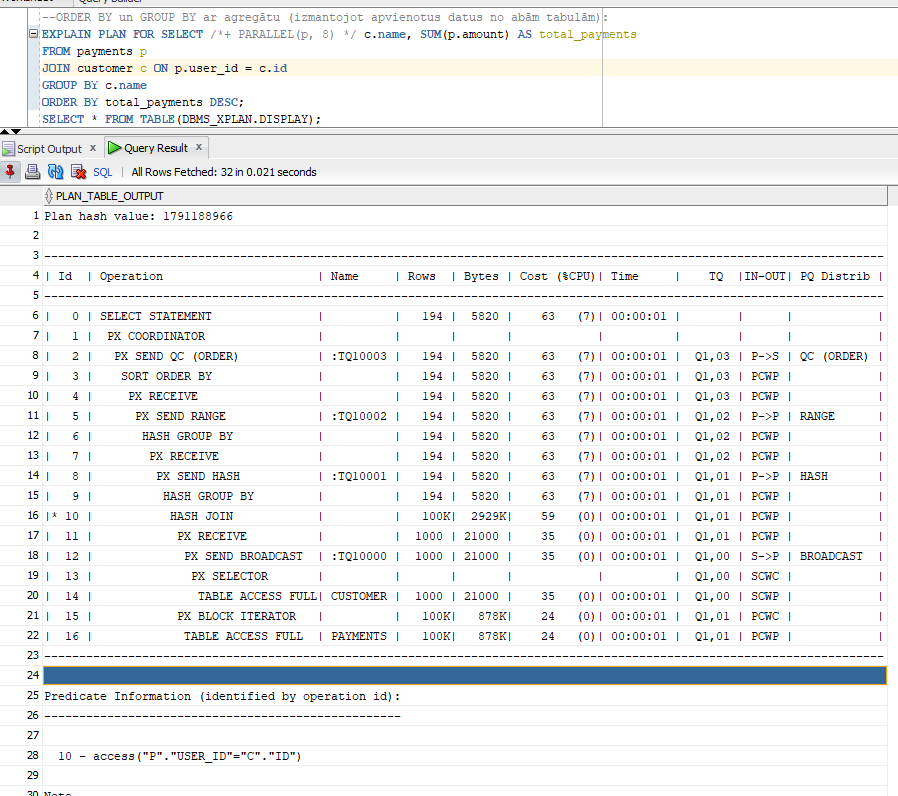
Vaicājums 2.2.:



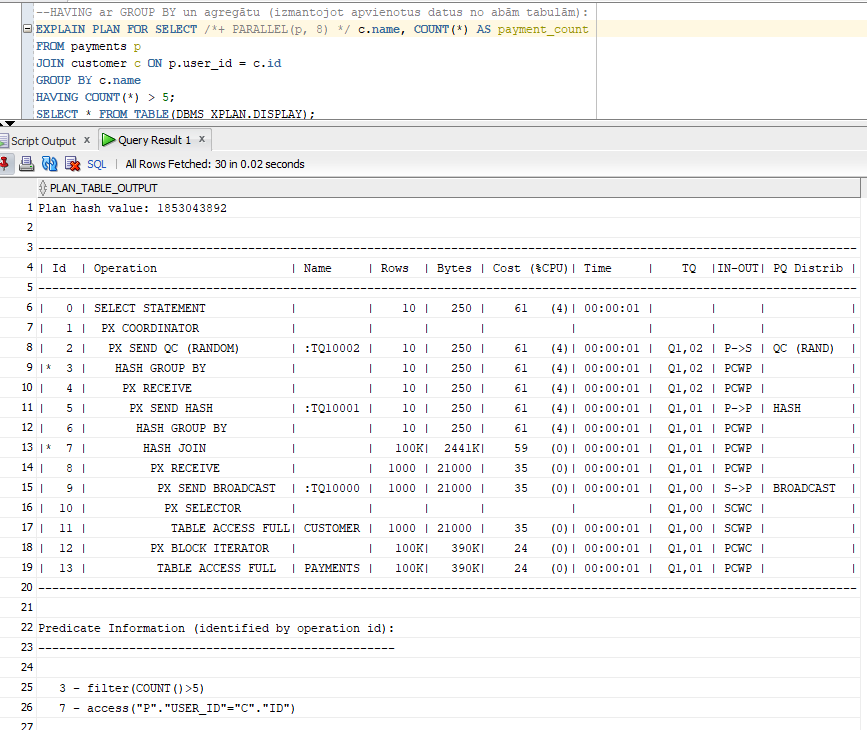
Vaicājums 2.3.:



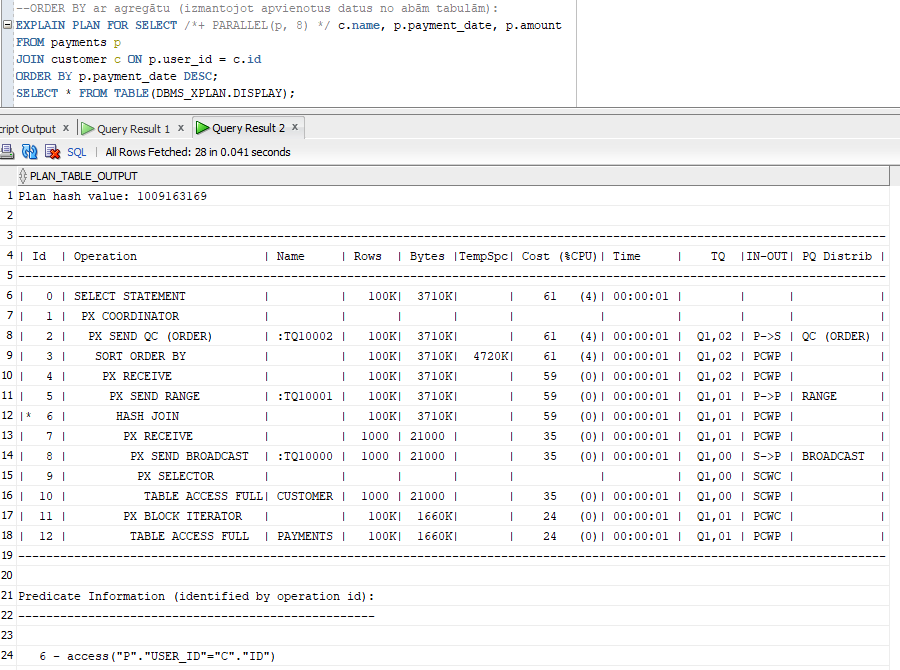
Vaicājums 3.1.:



Vaicājums 3.2.:



Vaicājums 3.3.:



**b) jāizanalizē un jāpaskaidro izpildes plāna kolonu un to vērtību semantika jeb jēga Jūsu vaicājumiem;**

Id: Rāda operācijas identifikatoru, kuru izmanto, lai atsauktos uz citām operācijām vai detalizētāku informāciju.

Operation: Apraksta veiktās operācijas tipu, piemēram, HASH JOIN, SORT ORDER BY. Šīs operācijas norāda, kā tiek apstrādāti dati.

Name: Parasti norāda tabulas vai indeksa nosaukumu, ar ko tiek veikta operācija. Šajā gadījumā dažām operācijām nav norādīta tabula vai indekss.

Rows: Apskaitītais rindu skaits, ko operācija apstrādā. Šis skaits ir prognoze, balstoties uz datubāzes optimizētāja statistiku.

Bytes: Datiem piešķirtais atmiņas apjoms baitos, ko operācija apstrādā.

TempSpc: Ja operācija izmanto pagaidu diska vietu, šeit parādīts šīs vietas apjoms.

Cost (%CPU): Operācijas izmaksas un CPU laika procents, kas tiek prognozēts. Šis rādītājs palīdz saprast, cik resursietilpīga ir katras operācijas izpilde.

Time: Prognozētais laiks, kas nepieciešams operācijas izpildei.

TQ: norāda uz "Table Queue", kas ir identifikators paralēlās apstrādes vidē. Šī kolonna raksturo konkrētās paralēlās izpildes vienības vai uzdevuma rindu (queue), kurā operācija tiek izpildīta.

IN-OUT: kā dati tiek pārsūtīti starp dažādām operācijām paralēlā izpildes vidē

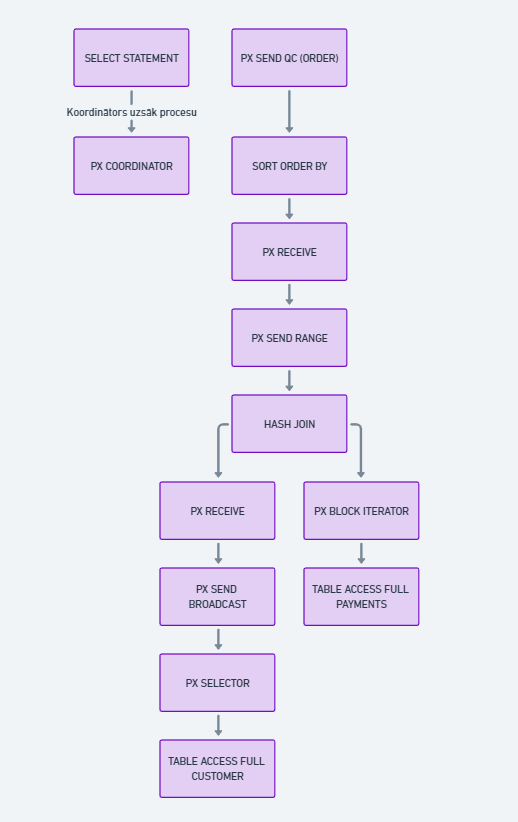
* P->S - Paralēls uz Seriālu. Šī vērtība norāda, ka dati no paralēlās operācijas tiek pārsūtīti uz seriālo operāciju, kas nozīmē, ka dati no vairākiem paralēlajiem pavedieniem tiek apvienoti vai koordinēti centrālajā procesā.
* P->P - Paralēls uz Paralēlu. Šis apzīmējums norāda, ka dati tiek pārsūtīti starp paralēlajiem pavedieniem, saglabājot datu apstrādes paralēlismu bez pārslēgšanās uz seriālu apstrādi.
* S->P - Seriāls uz Paralēlu. Šis rādītājs parāda, ka dati no seriālās operācijas tiek pārsūtīti uz paralēlo operāciju, kas nozīmē datu sadalīšanu vairākās paralēlajās plūsmās.

PQ Distrib: Paralēlo apstrādes distribūcijas metode.

* BROADCAST: Šī metode tiek izmantota, kad visiem paralēlajiem procesiem ir nepieciešama visa datu kopija. Datus no vienas operācijas nosūta visiem paralēlajiem procesiem, lai tie varētu veikt nepieciešamās operācijas ar pilnīgu datu kopu. Šis paņēmiens ir noderīgs operācijām, kurām nepieciešams piekļūt visiem datiem, piemēram, pievienojot datus vai veicot masīvas apstrādes uzdevumus.
* QC (ORDER): Dati tiek sakārtoti un nosūtīti koordinatora procesam (Query Coordinator). Koordinators pārvalda datu apstrādi un rezultātu apkopošanu. Šī metode tiek izmantota, kad ir nepieciešams veikt kārtējās datu apkopošanas vai kārtošanas operācijas pirms galīgā rezultāta nosūtīšanas lietotājam.
* RANGE: Dati tiek sadalīti diapazonos, kas ļauj paralēlajiem procesiem efektīvi apstrādāt tikai savus datu segmentus. Katrs paralēlais process saņem datu daļu, kas ir noteiktā diapazonā, un apstrādā to neatkarīgi no pārējiem. Tas samazina nepieciešamo datu pārvietošanas apjomu un var uzlabot kopējo sistēmas veiktspēju.
* HASH: Šī distribūcijas metode izmanto hash funkcijas, lai vienmērīgi sadalītu datus starp paralēlajiem procesiem. Katrs datu elements tiek piešķirts konkrētam paralēlam procesam, balstoties uz tā hash vērtību. Tas nodrošina, ka dati tiek sadalīti taisnīgi un efektīvi, samazinot iespējamību, ka viens process kļūst par "galveno punktu" un citi procesi paliek neizmantoti.

**c) jāuzzīme izpildes plāna paralēlo serveru darbības detalizēta diagramma un tās darbība jāpaskaidro.**

Vaicājums 3.2. plāna diagramma.



SELECT STATEMENT - SQL izpildes paziņojums; saknes solis visam plānam.

PX COORDINATOR - Paralēlo procesu koordinators; kontrolē paralēlo izpildi.

PX SEND QC (ORDER) - Sūta datus uz vaicājuma koordinatoru, sakārtojot tos pēc noteikta kārtības kritērija.

SORT ORDER BY - Dati tiek sakārtoti pēc norādītās secības.

PX RECEIVE - Koordinators saņem apstrādātos datus no paralēliem procesiem.

PX SEND RANGE - Dati tiek sadalīti starp paralēlajiem procesiem pēc diapazona.

HASH JOIN - Veic tabulu savienošanu izmantojot hašošanas metodi.

PX RECEIVE - Saņem datus no citiem paralēlajiem procesiem.

PX SEND BROADCAST - Izplatīšanas operācija, sūtot datus no viena paralēlā procesa uz visiem pārējiem.

PX SELECTOR - Izvēlas datus, kuri jāapstrādā.

TABLE ACCESS FULL CUSTOMER - Pilns piekļuves darbības veikšana klientu tabulai.

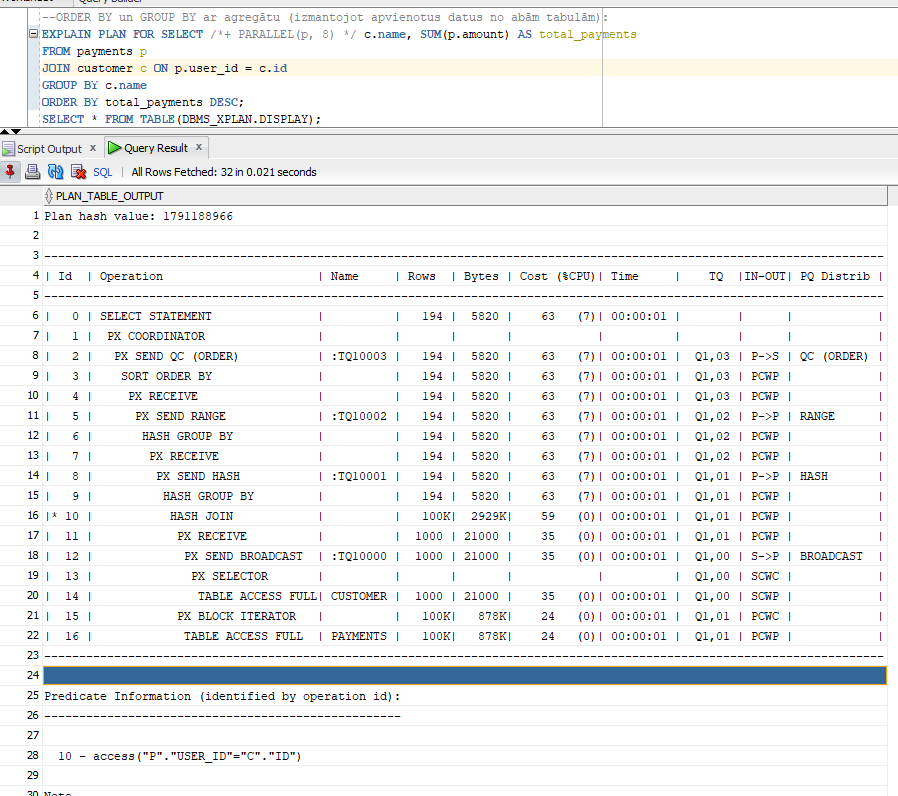
PX BLOCK ITERATOR - Iterācija caur datu blokiem, sagatavojot tos nākamajai apstrādei.

TABLE ACCESS FULL PAYMENTS - Pilns piekļuves darbības veikšana maksājumu tabulai.

**6. Jāizveido vaicājumi, kuru izpildes plānos var redzēt, ka tiek izmantotas 3 dažādas izplatīšanas metodes (distribution methods).**

**7. Jāpaskaidro šo vaicājumu izpildes plāna darbība un jāizvērtē iegūtie rezultāti.**

Darba gaitā izveidoti vaicājumi izmanto dažādu izplatīšanas metodes. Piemēram vaicājuma 3.2. tika izmantoti vairāki izplatīšanas metodes vienlaicīgi.



QC (ORDER):

Operācija ar ID 2 "PX SEND QC (ORDER)".

Datus sakārto un nosūta koordinatoram. Šī metode tiek izmantota, lai sakārtotu datus pēc noteiktām kritērijām (piemēram, pēc maksājuma summas), un pēc tam nosūtītu sakārtotos datus atpakaļ koordinatoram, lai apkopotu galīgos rezultātus.

RANGE:

Operācija ar ID 5 "PX SEND RANGE".

Dati tiek sadalīti diapazonos un nosūtīti dažādiem paralēlajiem procesoriem. Šī metode nodrošina, ka katra paralēlā vienība saņem apakškopu no kopējā datu apjoma, balstoties uz diapazonu, kas palīdz izlīdzināt slodzi un optimizēt datu apstrādi.

HASH:

Operācija ar ID 8 "PX SEND HASH".

Dati tiek sadalīti, izmantojot hash funkciju, kas nodrošina vienmērīgu datu izplatīšanu starp paralēlajiem procesiem. Šī metode ir efektīva, lai samazinātu datu pārsūtīšanas izmaksas un nodrošinātu efektīvu datu apstrādi lielos apjomos.

BROADCAST:

Operācija ar ID 12 "PX SEND BROADCAST".

Datus nosūta visiem paralēlajiem procesiem. Raidījuma metode tiek izmantota gadījumos, kad visiem paralēlajiem procesiem ir nepieciešams pilns datu kopums, lai veiktu apstrādi. Piemēram, tas noderēs, ja kāda nākamā operācija prasa piekļuvi visiem datiem.

Šīs metodes kopā veido efektīvu datu apstrādes sistēmu, kas ļauj optimizēt paralēlo apstrādi, samazinot laiku un resursu patēriņu, kas nepieciešams lielu datu apjomu apstrādei. Katra metode ir piemērota specifiskiem apstrādes scenārijiem un palīdz sasniegt maksimālu veiktspēju datubāzes vaicājumu izpildē.

**8. Kopējie secinājumi par izmantotajām tehnoloģijām.**

Izmantotā paralēlā apstrāde datubāzēs ļauj būtiski paaugstināt veiktspēju, jo ļauj izpildīt datubāzes vaicājumus, izmantojot vairākus procesorus vienlaicīgi. Šī pieeja ir īpaši efektīva liela datu apjoma apstrādei un sarežģītu vaicājumu izpildei, kas prasa intensīvu datu apstrādi.

Definējot paralēlisma pakāpi, var iegūt labāku kontrolēti un efektīvi izmantot sistēmas resursus. Piemēram, pārāk augsta paralēlisma pakāpe var palielināt sistēmas slodzi un samazināt kopējo veiktspēju, kamēr optimāli izvēlēta pakāpe var nodrošināt maksimālu datu apstrādes ātrumu.

Izmantojot dažādas datu izplatīšanas metodes, piemēram, BROADCAST, RANGE, HASH, un QC (ORDER), ir iespējams specifiski pielāgot datu apstrādes stratēģiju atkarībā no vaicājuma specifikācijām un datu struktūras. Katra metode piedāvā savas priekšrocības dažādos scenārijos:

BROADCAST ir noderīga, kad nepieciešama datu kopijas visiem procesoriem, piemēram, pie kompleksiem apvienošanas vaicājumiem.

RANGE nodrošina datu sadalīšanu pēc diapazona, kas ir efektīva pie lieliem datu apjomiem.

HASH ir efektīva pie savienojumiem, nodrošinot vienmērīgu datu sadalījumu.

QC (ORDER) ir svarīga datu kārtošanas un galīgo rezultātu apkopošanas procesā.

Praktiskajā darbā izmantotās tehnoloģijas un metodes ļāva iegūt būtisku veiktspējas uzlabojumu, ļaujot efektīvāk apstrādāt lielu datu apjomu un samazināt vaicājumu izpildes laiku. Paralēlo apstrādi izmantojot, tika demonstrēta datubāzes spēja efektīvi apstrādāt sarežģītus vaicājumus, kas izmanto sarežģītas datu savienošanas un agregācijas operācijas.

Eksperimenti un praktiskie uzdevumi apliecināja paralēlās apstrādes un moderno datubāzu pārvaldības sistēmu spēju efektīvi risināt sarežģītus uzdevumus, nodrošinot augstu datu apstrādes veiktspēju un elastību. Izmantojot šīs tehnoloģijas, ir iespējams būtiski palielināt darba produktivitāti un optimizēt resursu izmantojumu, kas ir īpaši svarīgi mūsdienu datu intensīvajā vidē.