

RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes, informācijas tehnoloģijas un enerģētikas fakultāte

1.praktiskais darbs

**Datu bāzes loģiskā arhitektūra un SQL priekšrakstu izpildes realizēšanas metodes (access paths)**

Izstrādāja: Natans Šalamberidze

171RMC203

Pārbaudīja: lektors Jānis Eiduks

2023./2024. māc. Gads

Darba mērķis

1. Padziļināti apgūt SQL priekšrakstu jeb vaicājumu izpildes tehnoloģiju. Vaicājumu izpildē tiek izmantotas dažādas datu izgūšanas un apstrādes metodes (access paths). Katra metode ir izstrādāta savu funkciju veikšanai un pie noteiktām datu īpašībām (struktūras, apjoma, sadalījuma).

Darba uzdevums

1. Praktiskajā darbā nepieciešamo datu tabulu izveidošana.

1.1. Divu savstarpēji saistītu pamattabulu ar 5 vai 6 kolonām definēšana (CREATE TABLE...).

1.2. PL/SQL programmu izveidošana tabulu aizpildīšanai ar datiem, izmantojot datu bāzes sistēmas gadījuma lielumu ģenerēšanas funkcijas. Vienai tabulai rindu skaits >= 100 000 un otrai tabulai rindu skaits >= 1000 rakstiem. Veikt tabulu aizpildīšanu ar datiem un uzzīmēt tabulu ER diagrammu. Ģenerētās tabulas varēs izmantot arī nākošajos darbos.

2. SQL priekšrakstu definēšana analizējamo SQL vaicājumu izpildes komandu (access paths) apskatei: pilna tabulas skenēšana, unikāli un neunikāli B-koku indeksi, indeksa diapazona skenēšana, tabulu ligzdotas cilpas savienojums, heš savienojums, šķirošanas un sapludināšanas variants.

3. SQL priekšrakstu izpilde. Ja nepieciešams konkrētas izpildes komandas iegūšanai, SQL priekšrakstos jāiekļauj norādes (hints) optimizatoram.

4. Jāiegūst vaicājumu izpildes plāni un jāapraksta ko katra plāna komanda dara konkrētam vaicājumam. Sarežģītākiem vaicājumiem arī jāuzzīmē komandu izpildes koks (tas dos arī labāku izpratni).

6. Secinājumos jāapkopo iegūtie rezultāti un priekšstati par metožu izmantošanas situācijām.

**1. Praktiskajā darbā nepieciešamo datu tabulu izveidošana.**

**1.1. Divu savstarpēji saistītu pamattabulu ar 5 vai 6 kolonām definēšana (CREATE TABLE...).**

CREATE TABLE PD1\_Clients (

ClientID NUMBER PRIMARY KEY,

ClientName VARCHAR2(100),

Industry VARCHAR2(100),

Discount INTEGER,

City VARCHAR2(100)

);

/

CREATE TABLE PD1\_Orders (

OrderID NUMBER PRIMARY KEY,

ClientID NUMBER,

OrderDate DATE,

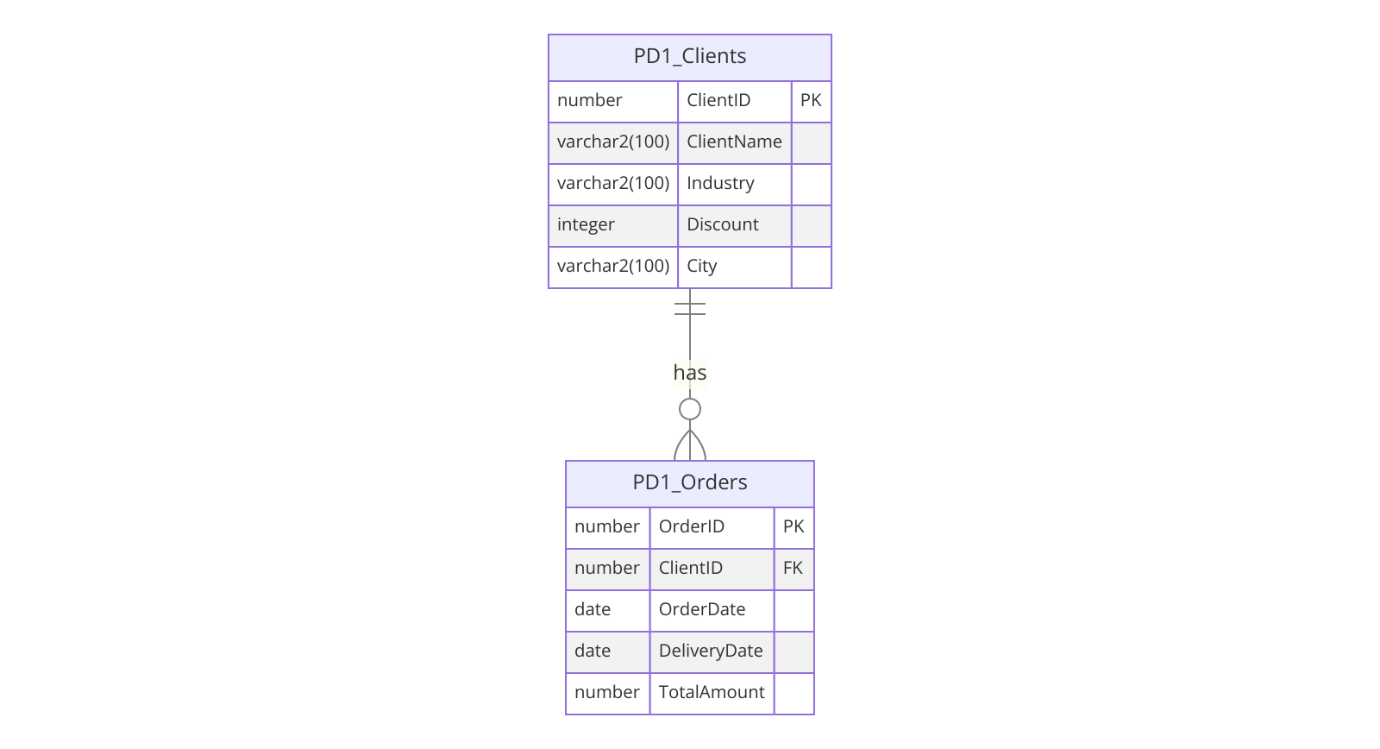
DeliveryDate DATE,

TotalAmount NUMBER,

FOREIGN KEY (ClientID) REFERENCES PD1\_Clients(ClientID)

);

/



**1.2. PL/SQL programmu izveidošana tabulu aizpildīšanai ar datiem, izmantojot datu bāzes sistēmas gadījuma lielumu ģenerēšanas funkcijas. Vienai tabulai rindu skaits >= 100 000 un otrai tabulai rindu skaits >= 1000 rakstiem. Veikt tabulu aizpildīšanu ar datiem un uzzīmēt tabulu ER diagrammu. Ģenerētās tabulas varēs izmantot arī nākošajos darbos.**

DECLARE

i NUMBER;

j NUMBER;

order\_date DATE;

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

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

TYPE industry\_array IS VARRAY(5) OF VARCHAR2(50);

TYPE city\_array IS VARRAY(5) 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');

industries industry\_array := industry\_array('Tehnoloģijas', 'Finanses', 'Tirdzniecība', 'Izglītība', 'Veselības aprūpe');

cities city\_array := city\_array('Rīga', 'Jūrmala', 'Liepāja', 'Ventspils', 'Daugavpils');

BEGIN

FOR i IN 1..1000 LOOP

INSERT INTO PD1\_Clients(ClientID, ClientName, Industry, Discount, City)

VALUES (

i,

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

industries(TRUNC(DBMS\_RANDOM.VALUE(1, 5))),

ROUND(DBMS\_RANDOM.VALUE(0, 30)),

cities(TRUNC(DBMS\_RANDOM.VALUE(1, 5)))

);

END LOOP;

FOR j IN 1..100000 LOOP

order\_date := SYSDATE - TRUNC(DBMS\_RANDOM.VALUE(1, 365));

INSERT INTO PD1\_Orders(OrderID, ClientID, OrderDate, DeliveryDate, TotalAmount)

VALUES (

j,

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

order\_date,

order\_date + TRUNC(DBMS\_RANDOM.VALUE(1, 30)),

ROUND(DBMS\_RANDOM.VALUE(100, 10000))

);

END LOOP;

COMMIT;

END;

/

**2. SQL priekšrakstu definēšana analizējamo SQL vaicājumu izpildes komandu (access paths) apskatei: pilna tabulas skenēšana, unikāli un neunikāli B-koku indeksi, indeksa diapazona skenēšana, tabulu ligzdotas cilpas savienojums, heš savienojums, šķirošanas un sapludināšanas variants.**

***Pilna Tabulas Skenēšana***

SELECT \* FROM PD1\_Clients;

SELECT \* FROM PD1\_Orders;

***Indeksu Izmantošana***

*Unikālie un Neunikālie B-koku Indeksi: Vispirms izveidojam indeksus.*

CREATE INDEX idx\_client\_city ON PD1\_Clients(City);

CREATE INDEX idx\_order\_total ON PD1\_Orders(TotalAmount);

*Indeksa Diapazona Skenēšana*

SELECT /\*+ INDEX(c idx\_client\_city) \*/ \* FROM PD1\_Clients c WHERE City BETWEEN 'A' AND 'M';

SELECT \* FROM PD1\_Orders WHERE TotalAmount BETWEEN 100 AND 10000;

***Savienojumu (Joins) Metodes***

*Ligzdotas Cilpas Savienojums (Nested Loops)*

SELECT /\*+ USE\_NL(c o) \*/ \* FROM PD1\_Clients c JOIN PD1\_Orders o ON c.ClientID = o.ClientID;

*Heš Savienojums (Hash Join)*

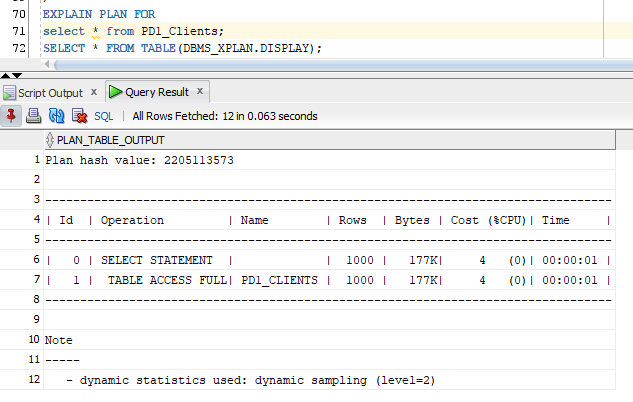
SELECT /\*+ USE\_HASH(c o) \*/ \* FROM PD1\_Clients c JOIN PD1\_Orders o ON c.ClientID = o.ClientID;

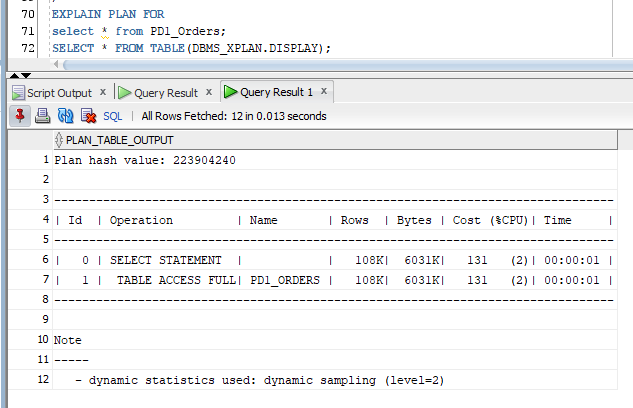
*Šķirošanas un Sapludināšanas Savienojums (Sort Merge Join)*

SELECT /\*+ USE\_MERGE(c o) \*/ \* FROM PD1\_Clients c JOIN PD1\_Orders o ON c.ClientID = o

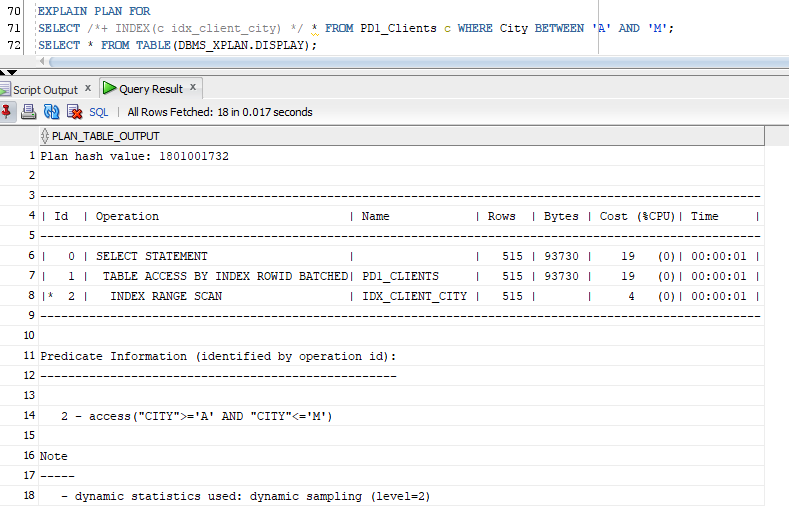
**3. SQL priekšrakstu izpilde. Ja nepieciešams konkrētas izpildes komandas iegūšanai, SQL priekšrakstos jāiekļauj norādes (hints) optimizatoram.**

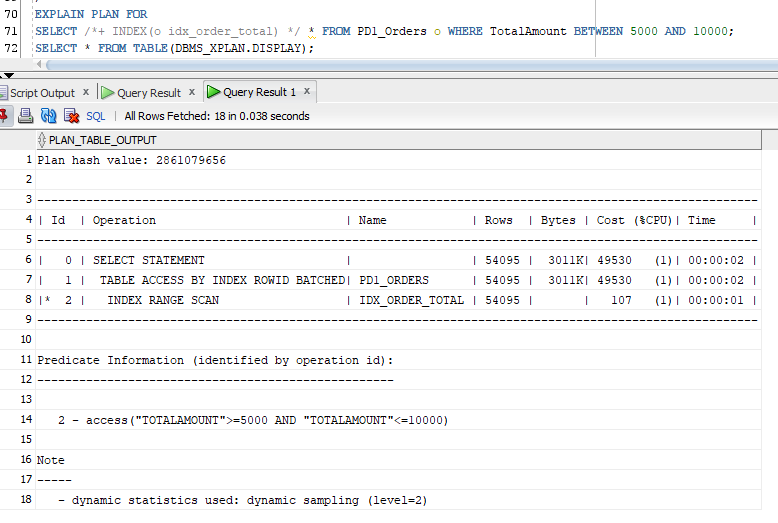
**4. Jāiegūst vaicājumu izpildes plāni un jāapraksta ko katra plāna komanda dara konkrētam vaicājumam. Sarežģītākiem vaicājumiem arī jāuzzīmē komandu izpildes koks (tas dos arī labāku izpratni).**



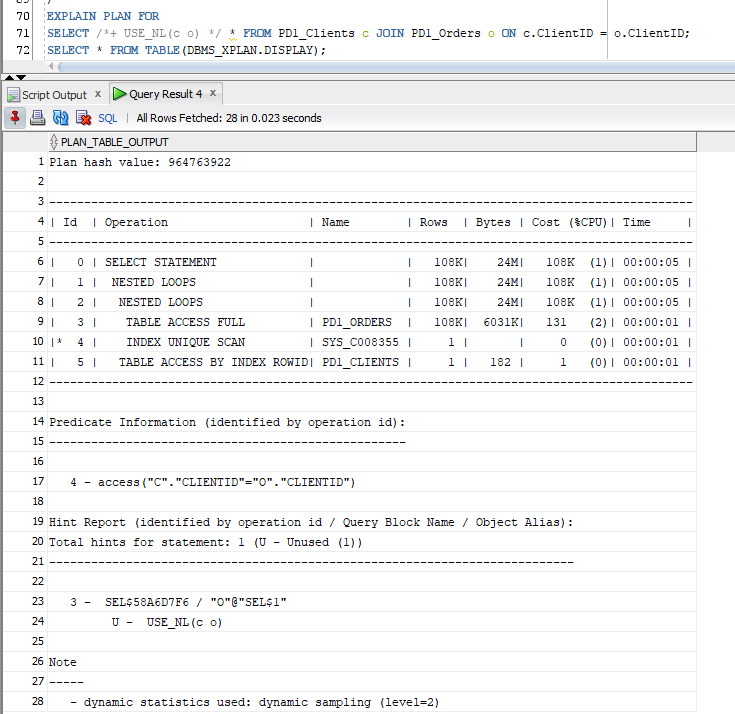


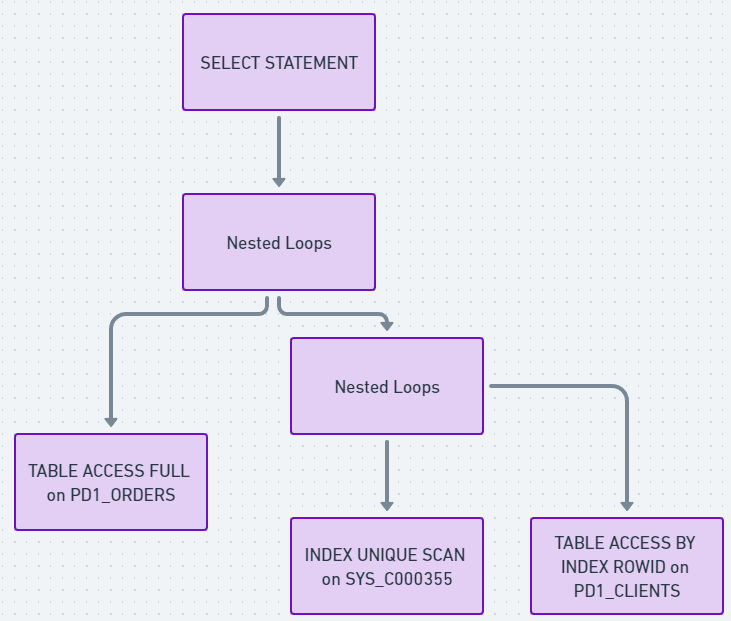
Pilna Tabulas Skenēšana (TABLE ACCESS FULL): Šī metode tiek izmantota, kad SQL vaicājums prasa pārbaudīt katrai rindai atbilstību nosacījumiem vai kad nav pieejami indeksi.





Index Range Scan un Table Access by Index ROWID Batched operācijas nodrošina, ka tikai nepieciešamās rindas tiek izlasītas no datu bāzes, kas samazina piekļuves laiku un resursu patēriņu salīdzinājumā ar pilnu tabulas skenēšanu.



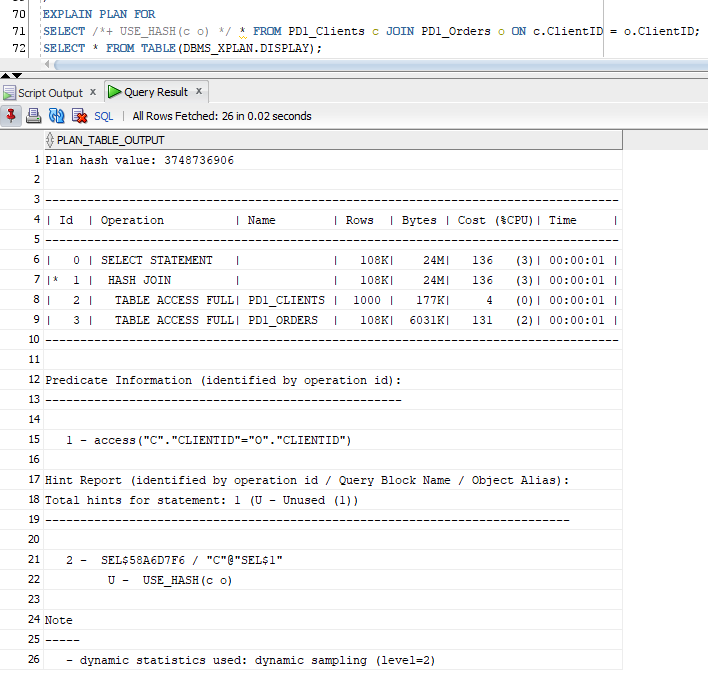


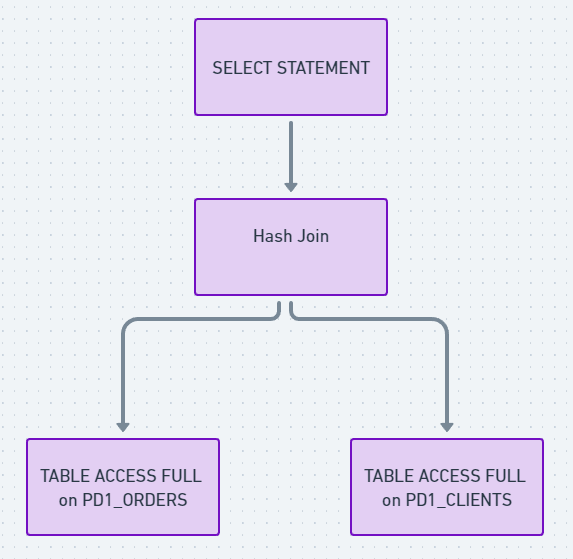
NESTED LOOPS: Norāda, ka izmantots ligzdotas cilpas savienojums, kas ir tipiska pieeja, kad viena tabula ir salīdzinoši maza un otrai tabulai ir indekss pēc savienojuma kolonnas.

TABLE ACCESS FULL (PD1\_ORDERS): Norāda, ka tabula PD1\_Orders tiek skenēta pilnībā, kas varētu būt optimāla izvēle, ja datu bāze uzskata, ka tabula nav pārāk liela vai nav efektīvāka indeksa izmantošanas iespēja.

INDEX UNIQUE SCAN (SYS\_C008355): Šī operācija norāda, ka izmantots unikāls indekss, kas, visticamāk, ir uz primārās atslēgas PD1\_Clients tabulā, lai efektīvi atrastu atbilstošo rindu, izmantojot ClientID no PD1\_Orders.

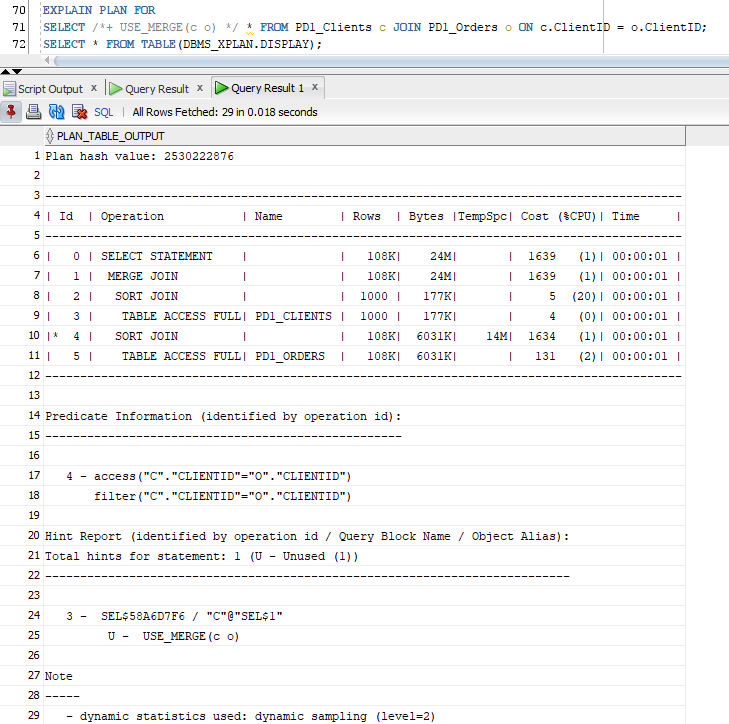
TABLE ACCESS BY INDEX ROWID (PD1\_CLIENTS): Norāda, ka, izmantojot iepriekš minēto indeksu, datu bāze iegūst konkrētās rindas no PD1\_Clients tabulas, izmantojot ROWID, kas ir ātrākais veids, kā piekļūt individuālām rindām.

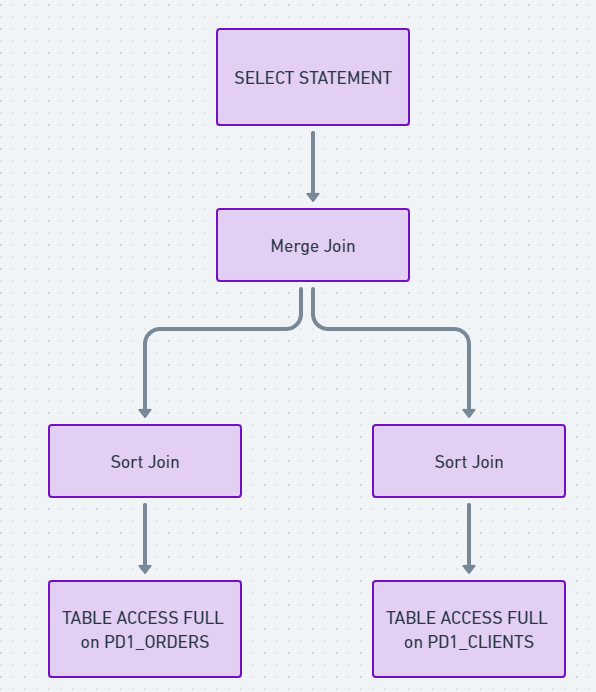




HASH JOIN: Norāda, ka izmantots heš savienojums starp divām tabulām. Šis savienojuma veids ietver heš tabulas izveidi no vienas tabulas datiem un pēc tam šīs heš tabulas izmantošanu, lai efektīvi meklētu atbilstošas rindas no otrās tabulas.

TABLE ACCESS FULL (PD1\_Clients un PD1\_Orders): Abas tabulas tiek skenētas pilnībā, lai izveidotu heš tabulu un veiktu savienojumu. Tas norāda, ka, lai gan heš savienojums ir izvēlēts kā optimāls, joprojām ir nepieciešams izlasīt visas rindas no abām tabulām.





MERGE JOIN: Norāda, ka izmantots šķirošanas un sapludināšanas savienojums. Šis savienojuma veids ir efektīvs, ja rindiņas abās tabulās ir vai tiks sakārtotas pirms savienošanas.

SORT JOIN: Katras no tabulām rindas tiek sakārtotas pēc savienojuma kolonnas pirms to sapludināšanas.

TABLE ACCESS FULL (PD1\_Clients un PD1\_Orders): Abas tabulas tiek skenētas pilnībā, lai iegūtu visas nepieciešamās rindas un pēc tam tās sakārtotu.

**6. Secinājumos jāapkopo iegūtie rezultāti un priekšstati par metožu izmantošanas situācijām.**

Šajā praktiskajā darbā tika izveidotas divas savstarpēji saistītas datu tabulas: PD1\_Clients un PD1\_Orders. Abas tabulas tika aizpildītas ar lielu datu apjomu, izmantojot PL/SQL programmas, kas ģenerēja datus izmantojot datu bāzes gadījuma lieluma ģenerēšanas funkcijas. Analizējot dažādas SQL vaicājumu izpildes metodes un izpildes plānus, ieguvām vērtīgas atziņas par optimālākajiem datu izgūšanas veidiem atkarībā no konkrētas situācijas.

Pilna Tabulas Skenēšana piemērota, kad nav pieejami indeksi vai kad jāapgūst liela datu daļa no tabulas.

Indeksa izmantošana ļauj būtiski paātrināt datu izgūšanu, jo īpaši, ja tiek lietots indeksa diapazona skenējums.

Nested Loops: Efektīvi izmantojams, ja viena no tabulām ir ievērojami mazāka, un otrajai tabulai ir piemērots indekss.

Hash Join: Īpaši noderīgs lielu datu apjomu apstrādē, kad nepieciešams efektīvi veikt savienojumus.

Sort Merge Join: Labi piemērots situācijām, kur abas tabulas jau ir kārtotas pēc savienojuma kolonnas, vai kad ir nepieciešams kārtot datus pirms to sapludināšanas.

Secinot, šis praktiskais darbs palīdzēja apgūt dažādas datu izgūšanas un apstrādes metodes, izprast to piemērotību konkrētām situācijām un izvēlēties optimālākos risinājumus lielā datu apjoma pārvaldībai. Tas ir svarīgs pamats efektīvai datu bāzes sistēmu izstrādei un pārvaldībai.