2023.03.27.

Előadás

# Ismétlés:

Execution plan (végrehajtás folyamat)

A CBO-t befolyásolja a hintek, a statisztikák (amibe ugyan úgy be tudunk avatkozni)

# A lekérdezés optimalizálás folyamata

1. Információk begyűjtése a rendszerről, tudjuk milyen indexek vannak egy táblán és milyen típúsúak pl unique vagy nem. Materializált nézetek vannak e
2. Lekérjük az explain plan-t. Azt értelmezzük, utána azonosítjuk a magas költségű műveletet. Megnézzük, hohy a magas költségű műveletet le tudjuk e csökkenteni.
3. Másik join módszer tesztelése.

Kipróbálunk más join módszert és megnézzük, hogy gyorsabban fut e le tőle

Allekérdezések hintjei. Az allekérdezések feldolgozását és kielemzését tudja elvégezni az optimalizáló.

Cardinality (Kardinalitás) becslés pontossága, hogy mennyire frissek.

1. Lekérdezés átalakítása, korrigálása.

A lekérdezésünket átírjuk más alakra ugyan azt a lekérdezést.

Ritkán van, de lehet, hogy ezzel is tudunk hatni a lekérdezés feldolgozóra.

1. Index készítés.

Átgondoljuk, hogy milyen indexeink vannak és átgondoljuk, hogy hova kéne bevezetni egy új indexet, mert pl sok lekérdezésünk van rá. (Viszont adott műveleteket lassítanak az indexek, ezért meg kell fontolni.)

Szelektivitás (adott oszlopban hány különböző érték található).

Clustering factor

# Információ gyűjtés

## Adatszótár fontosabb nézetei:

user\_indexes

user\_ind\_columns

user\_histograms

user\_tab\_col\_

# Statisztikák

Táblákról

Oszlopokról

Indexekról

Rendszerről

# Hisztogrammok

A CBO alapértelmezettként egyenletes eloszlásúnak feltételezi az adatokat. Ez félrevezetheti

Ha ez nem így van, akkor a join és a szűrés (elég rossz becslést fog adni).

Kb. melyik érték milyen gyakori egy oszlopban

Bucket-ek:

endpoint value – a bucket-ekben az oszlopban megtalálható értékek vannak

endpoint number – a bucket egyedi azonosítója (ez egy egyedi érték)

Típusai:

* Frequency – minden lehetséges értéket külön bucket-be rakja. A különböző értékeket rakja be egy olszopba és megszámolja, hogy hányszor fordulnak elő.
* Height-balanced tudta, hogy mennyi bucket-je van és megpróbálta a benne lévő értékeket egyenletesen elosztani.
* Hybrid – a frequency és a height balanced ötvözése. A lehetséges értékeket egyenlően igyekszik elosztani a bucketek között. (Kép 1-es és 5-ösöket tárolja az első bucket-ben, 10 és 25 a 2.-ban és 50 és 100 a 3.-ban. Az endpoint value a legnagyobb érték (5,25 és 100). A repeat count a legnagyobb értéknek a gyakorisága. (Nem stimmel a dia / kép.) 🡪 A jobb oldali kép a hybrid (, nem engedi, hogy átcsússzanak, mint a baloldalinál).

# Hintek

Az optimalizáló működését tudjuk befolyásolni. Tudjuk kényszeríteni, access path, joint, használjon e materailized view-t vagy nem ... .

Ha bennehagyunk egy hint-et akkor lehet, hogy nem a leg optimalizáltabb működést kényszerítünk ki belőle.

Tesztelésre jó ezt használni.

1 select után 1 hint-et lehet csak beírni, de több hint-et beleírhatunk a /\*+ hintek \*/ részébe.

Ha elírtunk 1 hint-et, akkor nem fog lefutni (csak maga a hint nem, nem veszi figyelembe).

Leggyakoribb hint-ek:

* INDEX (ezen a táblán használd ezt a bizonyos indexet, amikor a lekérdezést megvalósítod)
* INDEX \_DESC
* INDEX \_FFS (Fast Full Scan)
* INDEX\_JOIN (az indexek join-olását akarjuk kikényszeríteni)
* FULL (Full Table Scan)
* USE\_NL
* USE\_MERGE
* USE\_HASH
* ORDERED (a join-ba felsorolt tábla sorrendet használja)
* LEADING (nem írjuk elő a teljes sorrendet csak azt, hogy mivel kezdje (a joint))
* FIRST\_ROWS (megadhatunk egy számot paraméterbe, hogy csak a végeredmény első x sorára optimalizáljon)
* ALL\_ROWS (teljes adathalmaz megjelenítésére optimalizált)
* UNNEST (az allekérdezéseket megpróbálja a fő lekérdezésbe ’beolvasni’ és úgy lekérni)
* NO\_UNNEST
* NO\_QUERRY\_TRANSFORMATION
* MATERIALIZE (készítsen e materializált táblát vagy nem)
* INLINE

# Csoportosítás végrehajtása

## SORT GROUP BY

A csoportosító mező szerint rendez, így az azonos csoportba tartozó rekordok egymás után következnek majd.

Ha már a tábla alapból rendezett akkor ezt használja.

## HASH GROUP BY

Veszi a groupby oszlopát, azt hasheli és az alapján csoportosítja.

# Index clustering factor

Ha van index egy olszopra akkor jobb egy oszlopra nem a full table scan-t használni, hanem először keresni az indexre, megtalálni a row id-kat és utána kinyúlni a disk-re.

Nem biztos, hogy ez ilyen egyszerű, mert lehet, hogy az index struktúrában ott vannak egymáshoz közel azok amik kellenek, de lehet, hogy a háttértáron össze vissza vannak elhelyezve.

Ez azt mutatja meg, hogy mennyire vannak szétszorva a táblához tartozó sorok a memóriában.

Az index clustering factor értéke kicsi és közelíti az adatblokkok számát, ha a memóriában rendezett módon van index alapján (b-fa).

Ha az index clustering factor értéke nagy, akkor köszelíti a tábla sorainak számát

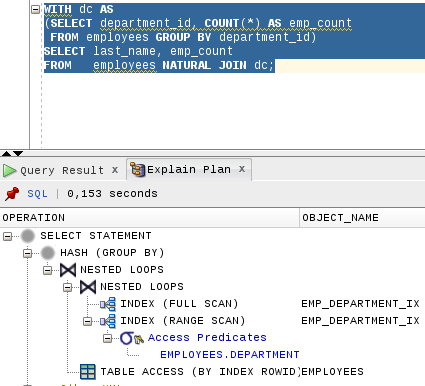
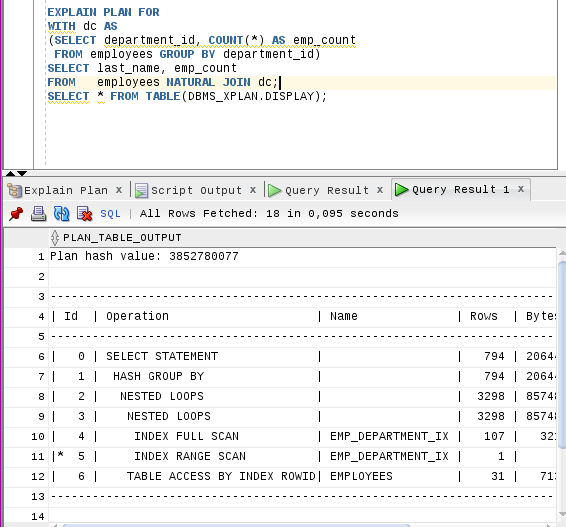
Az oracle tárolja az index clustering factor értékét.

Gyakorlat

Subquery factoring – Megjeleníti minden ember nevét és hogy az ő department-ben hányan dolgoznak.

WITH dc AS (SELECT department\_id ...) a lekérdezés módja.

select, alatta hash, alatta 2 nested loops, alatta 2 index és alatta accass predicates

 az index full scan végig scan-eli az emp\_department\_ix-et

ZH-ban lesz Explain Plan. El kell tudni ’magyarázni’, hogy hogyan fut le, milyen logika alapján hajtódik végre.

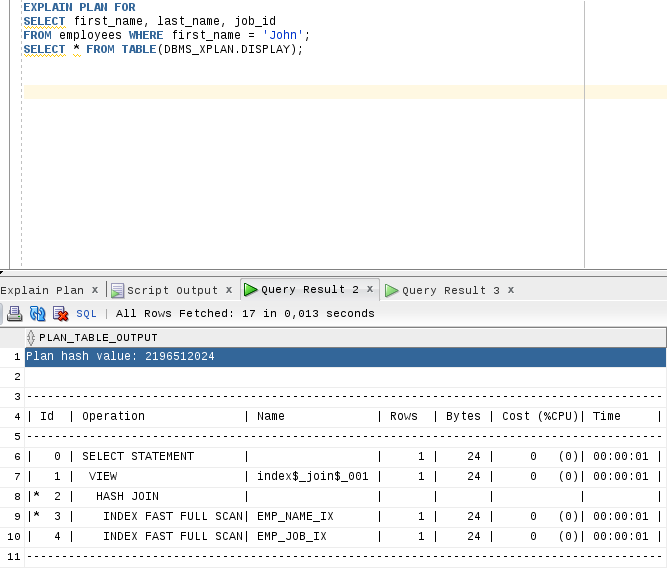
ZH-ban elég ha EXPLAIN PLAN-el megadjuk az ID-kat, hogy milyen sorrendben hajtódtak végre.

EXPLAIN PLAN FOR

...

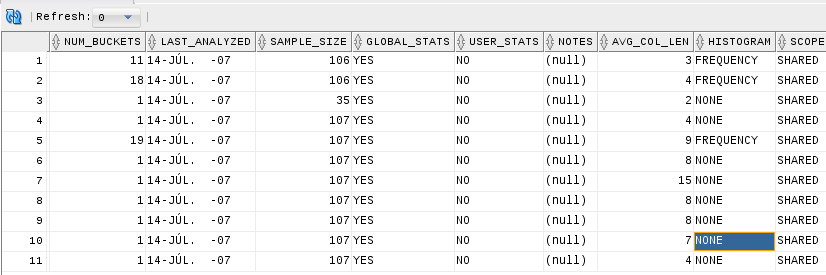
...

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);



Nincsen hisztogram a first name-re, ezért csak 1-es kardinalitást becsül.

Ezt meg lehet vizsgálni a table rész statisztika megnézésével vagy utasítással.



SELECT COLUMN\_NAME, HISTOGRAM

FROM USER\_TAB\_COL\_STATISTICS

WHERE TABLE\_NAME = 'EMPLOYEES';

CREATE TABLE logs AS

SELECT level log\_id, dbms\_random.string('a',100) cookie,

round(dbms\_random.value() \* (5000 - 1000) + 1000) nums,

round(dbms\_random.value() \* (206 - 100) + 100) employee\_id

FROM dual CONNECT BY LEVEL <= 50000;

ALTER TABLE logs ADD CONSTRAINT log\_pk PRIMARY KEY (log\_id);

ALTER TABLE logs ADD CONSTRAINT log\_fk

FOREIGN KEY (employee\_id) REFERENCES employees (employee\_id);

EXPLAIN PLAN FOR

SELECT department\_id, nums

FROM logs l INNER JOIN employees e ON l.employee\_id = e.employee\_id

WHERE nums > 3000;

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

/\* Gyorsitashoz, mert a nums lassitja -> erre alaklmazni valamit

(ha idegen kulcs van definialva akkor arra használni indexet).\*/

CREATE INDEX logs\_nums\_ix ON logs(nums);

CREATE INDEX logs\_emp\_id\_FK ON logs(employee\_id);

Ehhez nagyon hasonló lesz a ZH-n.

/\* 0-at ad vissza \*/

EXPLAIN PLAN FOR

SELECT d.department\_id, SUM(salary)

FROM departments d INNER JOIN employees e

ON e.department\_id = d.department\_id

GROUP BY d.department\_id HAVING SUM(salary) > 10000;

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

EXPLAIN PLAN FOR

SELECT department\_name, összfizu

FROM departments a INNER JOIN

( SELECT de.department\_id, SUM(salary) összfizu

FROM departments de INNER JOIN employees em

ON de.department\_id = em.department\_id

GROUP BY de.department\_id

HAVING SUM(salary) > 10000 ) b

ON a.department\_id = b.department\_id;

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

SELECT nums FROM logs WHERE UPPER(cookie) = 'FGH';

/\* hogy milyen substring van benne az fuggetlen az indextol, felesleges index-elni akkor.

Jo viszont, ha teljes egyenloseg van és mindenképp fel akarjuk gyorsitani + van ertelme is.\*/

CREATE INDEX logs\_UPPER\_ix ON logs(UPPER(cookie));

ZH példák

Gyakorlat

# 1

Compose a data dictionary query to show which indexes exists in the LOCATIONS table (1)

How many indexes are defined? What are the names of the indexes? On which column is the given index defined? (1)

What type of indexes are this?

# 2

Írni egy lekérdezést, ami cartesian join-t használ (=nem adunk meg neki feltételt)

# 3

Adjunk meg olyan lekérdezést, ami összetett, használ pl groupby-t vagy where-t, joint, ...

# 4

LESZ

SELECT \* FROM employees WHERE employee\_id = 106 🡪 Milyen access path-ot használt és miért

(full table scan-t használ, mert nincs rajta index)

# 5

Futtassuk a TestGenerate script és futtassuk a select-et. Analizáljuk az execution plan-jét.

Mi a cost-ja? Melyik az a lépés, ami a cost nagyját kiteszi? Milyen oszlopára épül a hisztogramm?

Tegyünk egy oszlopára hisztogrammot. Gyorsítsuk pl rakjunk index-et egy oszlopára (ami az órán is volt.)

Elmélet

Mi a funkcionális függőség

Egy b-fa index struktúrában mi a belső csúcs szerepe (navigáció segítés, tartalma:pointerek) és mi a levelek szerepe?

Mikor használ Nested group-ot

Leír egy Join típust és megkérdezzük melyik (előbb összekapcsolja a .... vagy nested táblát épít)

Mond egy táblát és melyik NF (normál formát) teljesíti (1NF, 2NF, 3NF, BNF)?

Igaz vagy hamis

1. Egy tábla o1 oszlopára b-fa indexet építünk. Az o1 oszlop nem unique típusú. Ekkor az index leveleiben biztos szerepel legalább 1 érték többször is. HAMIS

2. Egy tábla o1 oszlopára b-fa indexet építünk. Az o1 oszlop nem unique típusú. Ezen az index struktúrán Index Full Scan művelet biztos nem fog végrehajtódni, Index Range Scan viszont lehetséges. HAMIS

3. Ha az employee tába last\_name oszlopához létezik B-fa típusú index, akkor azt az optimalizáló WHER UPPER(...) IGAZ

4. A CBO működését a felhasználó is tudják befolyásolni. IGAZ

5. A táblatér logikai tárolási egysége az adatbázisnak, melyhez egymással logikai

6. Full table scan nagyon költséges műveletm ezért optimalizáló kizárólag abban az esetben választja ki, ha nem létezik index a feltételben metalálható oszlopra.

7. Ha az indexelt oszlopra egyenlőség feltétel van megadva akkor Index range scan Access Path nem használható. HAMIS

8.

9. A külső (driving) row source.hoz megkeresi a belső row source-ból a hozzá tartozót

Nested loop join

10. Sprt merge-t használ az optimalizáló, ha Alapból kicsi a tábla vagy ha alapból is sorted

Órai munka önálló

Logs táblához mutassunk be Join-okat, access path típúsokat, hinteket.

Ilyen várható a zh-ban.

Amit vár: Felírjuk az access path és utána leírjuk a lekérdezést.