Elmélet

A lekérdezés optimalizálás folyamata:

1. Információk begyűjtése a rendszerről, tudjuk milyen indexek vannak egy táblán és milyen típusúak 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.
4. Lekérdezés átalakítása, korrigálása. A lekérdezésünket átírjuk más alakra ugyan azt a lekérdezést. Ritkán van, de lehet, hogy ezzel is tudunk hatni a lekérdezés feldolgozóra.
5. 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
- Indexekről
- 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
```

```
EXPLAIN PLAN FOR
...
...
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

```
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);
```