Elmélet

SQL tuning:

Lassan futó lekérdezések optimalizálása / 'hangolása', teljesítménybeli elvárásoknak megefelelése érdekében.

Oracle-ben nagyrészt automatikus, de rá lehet segíteni.

Szintaktika / Szemantika ellenőrzés:

- **Szintaktika** - jó lekérdezésű SQL kód, betartottuk e a szabályait pl FROM helyett nem e FORM-t írtunk.
- **Szemantikai** - A lekérdezésnek van e értelme pl SELECT * FROM non_existing_table; - Nem értelmezhető lekérdezés, mert nincs ilyen tábla.

Shared Pool Check:

Az Oracle a lekérdezéshez egy hash (#) értéket generál és ezt tárolja el a shared pool-ban. Ha a lekérdezés már volt, akkor nem kell újra lefordítani, csak a hash értéket kell megnézni és ha van, akkor a shared pool-ból lekérdezi a lekérdezést. Ha nincs, akkor újra lefordítja és tárolja el a shared pool-ban.

Ha van találat, akkor is meg kell bizonyosodnia arendszernek, hogy a felhasználónak van eg jogosultsága a táblához. Ha nincs, akkor nem fut le a lekérdezés, hanem hibát dob.

Optimalizálás

CBO (Cost Based Optimization):

- **Cost** - Az Oracle számolja ki
- A legkisebb cost-ot választja
- Lekérdezés teljes költségét számolja ki (megbecsüli, mert nem feltétlenül tudja a legfrissebb satisztikát (/ adatokat)).

Rule Based Optimization:

Lefektettek alap szabályokat pl, ha az oszlopon van index, akkor egy where esetén előbb azt vizsgálja meg.

Row Source Generation:

Miután kiválasztotta a legkisevv költségű tervet, utána lefordításra kerül elemi utasításokra. Row source - rekordok halmaza
A végrehajtó lépésekknek mindig van: - egy inputja, ami lehet egy vagy több row source - egy outputja, ami egy row source

ID 0 (Select, legelső sor)-ban látszódik a Cost-nál az össz költséget

Row Source Tree:

Megmutatja az általa legenerált fastruktűra:

- a tábla sorrendjét
- a hozzáférés módját az egyes táblákhoz
- a táblák összekapcsolásának módját
- az egyéb műveleteket

Postorder módszer szerint kell értelmezni.

Ahhoz, hogy a szülő műveletet el tudjuk végezni, ahhoz el kell végezni az alatta lévő műveleteket.

Access Path

A hozzáférés módját határozza meg, hogyan olvasunk ki egy row source-ból.

Típusai:

- **Full Table Scan** - A tábla teljes tartalmát olvassa ki
- **Table Access by Rowid** - A tábla egy adott sorát olvassa ki
- **B-Fa indexek:**
  - Index Unique Scan
  - Index Range Scan
  - Index Full Scan
  - ...

Full Table Scan:

Minden sort beolvassa a táblából pl, ha:

- nincs benne index
- hint van beleírva -> kényszerítjük vele (/*+ hint szövege */)
- az egyéb módszerek költségesebbek lennének (az oszlop túl kicsi szelektivitású, ...)

Table Access by Rowid:

Csak a rowid sorokat akarja kiolvasni.

A rowid megmutatja hol van fizikailag a rekord.

Index:

Opcionális adatstruktúra, amely bizonyos esetekben felgyorsíthatja a lekérdezésekhez való hozzáférést.

Index Unique Scan:

Legfeljebb 1 rowid-t ad vissza (lehet, hogy nem is ad vissza semmit, mert unique).

Index Range Scan:

Egy értéktartományhoz tartozó rowid-kat ad vissza.

Index Full Scan:

Kiolvassa a levél elemeket a b fából az elejétől a végéig, sorrendben.

Indexelt, nem null mezőre rendelkezést kérünk. Ha pl rendezni akarunk akkor mégtobb értelme van ennek.

Index Fast Full Scan:

Nem veszi figyelembe a sorrendet, csak megkeresi a levél elemeket és különbiző sorrendben kilistázza (nem kel az 'ugrálást megtartani').

Index Skip Scan:

Nem az első mezőre, hanem a második mezőre keresünk. (Át lehet ugrani részt.)

Összetett inddünk van ls a keresett feltétel a második oszlolpára vonatkozik.

Index Join Scan:

A két indexből kiszedi az oszlopértékeket a rowid-ket és utána a rowid-k alapján összekapcsolja a táblákat.

Join

3 különböző algoritmussal tudja végrehajtani.

Meg kell határozni, hogy a 3 közül melyiket választja + meg kell kapnia a 2 row source-t és kitalálja a módszerét, ami összefügg azzal, hogy milyen típúsú joint futtatunk.

Ha több táblát join-olunk össze, akkor eldönti, hogy azokat millenő sorrendben join-olja össze.

Cél: minél kevesebb rekorddal kelljen dolgozni.

3 módszer:

- Nested Loop Join
- Sort Merge Join
- Hash Join

- (+ Cartesian Join)

Nested Loop Join:

A külső row source minden sorához megkeresi a belső row source-ból a hozzá tartozót. A kicsi row source-t választja külsőnek, ha egy nagyot és egy kicsit szeretnénk össze join-olni.

Arra optimalizál, hogy az első néhány sort adja vissza nagyon gyorsan.

Sort Merge Join:

A két row source-t rendezzük, majd összehasonlítjuk a sorokat.

elindul először az egyikből, veszi az első sort és megnézni, hogy a másiknak a 2. sorában összepasszol e. Ha összepasszolnak, akkor a következőt veszi még mindig a 2. táblából. Ha nem passzolnak, akkor lép 1-et az első táblából.

Akkor használják, ha amúgy is szükséges rendezni, vagy nagy tábláink vannak.

(A rendezés dága művelet.)

Hash Join:

Az egyik row source-joz egy hash táblát épít fel.

A kisebb táblához az összekötő oszlop értékeit hash-eli és készít belőle egy hash táblát. A második táblán az összekötő részt hash-eli és megnézi, hogy egyezik e a másik hash értékével.

Akkor tudja használni, hogy ha =-jel van a kettő között (egyezést keresünk).

Cartesian Join:

Minden sort minden sorral párosít.

Ha ilyen van, akkor elírás vagy hiba van valószínűleg.

(Nagyon ritkán kell ténylegesen.)

Gyakorlat

EXPLAIN PLAN FOR SELECT * FROM employees ORDER BY first_name DESC; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT last_name, salary FROM employees WHERE salary > 4000; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT * FROM employees WHERE employee_id > 190; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT first_name, employee_id FROM employees WHERE employee_id>10 and first_name like 'A%'; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

/Unique scan because employee id is unique/ EXPLAIN PLAN FOR SELECT * FROM employees WHERE employee_id=106; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

/Index range scan, because it is not unique/ EXPLAIN PLAN FOR SELECT * FROM employees WHERE department_id=100; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT * FROM employees WHERE department_id = 20; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT * FROM employees WHERE department_id>=20; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

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

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

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

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

EXPLAIN PLAN FOR SELECT COUNT(department_id) FROM employees; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT MIN(department_id) FROM employees; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT employee_id, job_id FROM employees; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

/Index skip scan, az osszetett index az emp_id és a date_pk mentén keres/ EXPLAIN PLAN FOR SELECT start_date FROM job_history WHERE start_date < sysdate; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT first_name FROM employees WHERE first_name like 'A%'; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT first_name FROM employees WHERE first_name like '%a'; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT * FROM employees LEFT JOIN departments USING (department_id); SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT * FROM employees INNER JOIN departments USING (department_id); SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT * FROM employees INNER JOIN departments USING (department_id) WHERE employee_id=106; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT employee_id, salary,department_name FROM employees FULL JOIN departments USING (department_id); SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT last_name, job_title FROM employees FULL JOIN jobs USING (job_id); SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT last_name, job_title FROM employees NATURAL JOIN jobs; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT employee_id, salary, job_title FROM employees, jobs WHERE salary < max_salary; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT employee_id, salary, job_title FROM employees, jobs WHERE employees.job_id=jobs.job_id AND salary < max_salary; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT employee_id, salary, department_name FROM employees, departments; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

/+ORDERED emiatt kell cartesian-t csinalnia, mert eredeti sorrend alapján az 1. sor a masik tabla 1. soraval nem egyezik .../ EXPLAIN PLAN FOR SELECT */+ORDERED/* e.last_name, d.department_name, l.country_id, l.state_province FROM employees e, locations l, departments d WHERE e.department_id = d.department_id AND d.location_id = l.location_id; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT * FROM employees WHERE employee_id > 190; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT SUM(salary) osszeg, manager_id FROM employees GROUP BY manager_id; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR SELECT SUM(salary) osszeg, manager_id FROM employees GROUP BY manager_id ORDER BY manager_id; SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

EXPLAIN PLAN FOR

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);