Big Data technologies

Database and Big Data technologies - Óbuda University 2023

Midterm requirements

- [ ] Two tests, one in the middle of the semester, one at the end of the semester:
  - [ ] First test: 5th week. Topic: Relational databases, tuning.
    - Theory and Practice (40 points)
  - [ ] Second test: 13th week. Topic: Big data and Spark.
    - Theory (10 points)
- [ ] Must pass both tests and all homework with at least 51% each.
- [ ] Weekly summary of each lecture (20 points)
- [ ] Student has to solve a homework project in the topics of Cassandra, MongoDB and Spark (30 points)
  - Student's participation during lectures is considered as part of the Homework

Oracle Virtual Machine

- Download from here
- VMWare Workstation 17 license key: 4A4RR-813DK-M81A9-4U35H-06KND
- Client tools to Oracle Database:
  - SQL Developer
  - SQL Plus
- Schema: hr / hr
- Online possibility to reach the database: apex.oracle.com
  - HR schema has to be created by scripts

---

author: Albert Dávid neptun code: H1B5EF date: 2023.03.05.

Elmélet

- Entity - egyedi valóságban létező dolog
- Entity type - entity-ket közös tulajdonságok alapján leíró csoport
  - Logikai kategória
  - Ugyanolyan attribútumokkal leírt objektumok
  - Például: Tej, kenyér, csoki → Termék

- Entity Instance - egy Entyt type egyik specifikus példánya
- ER (Entity Relationship) diagram - egy adatmodell, amelyben az entitások és azok közötti kapcsolatok láthatók

Levels Of Data Modeling

1. Conceptual data model - egyedek és a legmagasabb szintű kapcsolatok beazonosítja
2. Logical data model - az egyedek attribútumai is jelen vannak
3. Physical data model - DBRMS szinen (függ attól, hogy milyen rendszerben lesz megvalósítva)

Erőse gyedhalmazok: van primary key(), ami egyedi az egyedeknek)
Gyenge egyedhalmazok: nincs primary key() (jele: négyzet a négyzetben)
Foreign key: egy másik tábla egyedi kulcsa

- a másiik táblában is kulcsnak kell lennie (vagy elég csak egyedinek lennie)
- a másik táblában is léteznie kell az értéknek (esetleg null is lehet)

ISA kapcsolat - öröklődés

Normalizálás

Functional dependency: egy attribútum egy másik attribútum függvénye (egy relációban az attribútommok hogy függnek egymástól pl.: ha tudjuk a termék id-t, akkor tudjuk a termék nevét/márkáját is)

1NF - atomi adatok tárolása (attr. - nem lehet összetett / nem tartalmazhat több értéket)

Megoldási lehetőségek:

- horizontális - több attribútum egy táblában (több attr.-ok bevezeése)
- vertikális - az öszetett attr. szétszedése (több táblába szedése, új sorok/egyedek jönnek létre)

2NF - 1NF + a nem kuécs attr.-ok funkcionális függvénye a prim kulcs-tól

Ha 1NF és egyszerű a prim kulcs, akkor a 2NF biztosan teljesül (csak összetett prim kulcs esetén kell vizsgálni)
Megoldás:

- Megkeresni a függést, ami sérti a 2NF-t, és kiszervezni egy új táblába

3NF - 2NF + nem lehet funkc. függés a nem kulcs attr.-ok között

Megoldás:

- Hasonló módon mint a 2NF-nél kiszervezni külön táblába

BCNF - 3NF-re hasonlít - "Ha van függőség a táblázatban, akkor az kulcs-függőség" (kicsit szigorúbb, mint a 3NF)

SQL

1. DDL - Data Definition Language

    - Adatbázis objektumok manipulálása (kreálása, szerkezetének módosítása, törölése)
2. DML - Data Manipulation Language
    - Adatok manipulálása (insert, update, delete)
3. DCL - Data Control Language
    - Jogosultságok kezelése (grant, revoke)
4. TCL - Transaction Control Language
    - Tranzakciók kezelése (commit, rollback)

---

author: Albert Dávid neptun code: H1B5EF date: 2023.03.06. lecture: 2

Elmélet

Indexek

A lekérdezések optimalizálására / gyorsítására szolgáló adatstruktúrák.
(Ha nem kell az egész tábla, akkor csak az indexeket kell lekérdezni, így gyorsabb a lekérdezés).

Az index lehet:

- **Unique** - Ha a levél elemekben lévő kulcsok egyediek (unique / primary key az oszlop a táblában).
- **Nem unique** - Egy érték többször is előfordul az oszlopban.

- 2 fajtája van:
    - **B-trees**
        - A lekérdezés gyorsabb, de a tárolás lassabb.
        - Fa struktújájú adat halmaz, gyökér elem fent van, levél elemben sorrendbe rendezve vannak az értékek (by default növekvőben))
        - összekötés van a levélelemek között □ a lekérdezés tud ugrálni a levélelemek között
    - **Bitmap index**:
        - Kulcs érték található a 2 ROWID-ban (Start ROWID, End ROWID) és egy bitmap sorozattal (melyik sor kell az adatokból)
        - Amikor sok az ismétlődő elem, akkor érdemes ezt használni.
        - Jól tudja támogatni, ahol aggregációk történnek (pl. összesítés, stb.)

Index előnye: Ha egy oszlopra felépül akkor tudja gyorsítani a keresést, mert nem kell a teljes táblát lekérdezni, csak az indexet.

Ha a tábla mellé van index struktúránk, akkor a táblát karban kell tartani + tárolási többletköltsége van.

Adatszótár

Metaadatok az adatszótárban (data dictionary: központi, csak olvasható táblák és nézetek) tárolódnak.

Data dictionary szerkezete:

- **Base tables**
- **User-Accessible Views**

Data dictionary típusai:

- **DBA** - Data Base Administrator (az adminisztrátor tudja olvasni és írni)
- **ALL** - Amihez jogosultság van
- **USER** - azokról az objektumokról, aminek én vagyok a tulajdonosa

Adatbázis architektúra

3 nyelv, amit megért:

- **SQL** - Dekleratív nyelv, nem mondja meg, hogy hogyan kell megoldani, csak azt, hogy mit kell megoldani.
- **PL/SQL és Java** - Imperatív nyelv, megmondja, hogy hogyan kell megoldani.

Architektúra:

- Amikor a kliens kiadja az első utasítást akkor létrejön egy *user process*, ami kommunikál a *server process*-sel.
  Van egy listener a kettő között, ami figyeli mindig, hogy van-e bejövő kérés.
  Ez egy munkafolyamat (session), aminek vége, ha a felhasználó kilép vagy a kapcsolat megszakad vagy ...
- Adatbázis szerver több Instance (szolgáltatásból) épül fel.
  - Memory Structure (System Global Area)
  - Process Structure (Process Control Block)
- Adatbázis alkotó fájlok struktúrája (fizikai rész) Storage Structures

Egy adatbázis példány mindig egy adatbázishoz kötődik.

Lehet, hogy több adatbázis példány egy adatbázishoz kötődik.

Egy példány (Instance) nem kötődhet több adatbázishoz (Database-hez).

Kliens kapcsolódása az adatbázishoz:

- Van az adatbázis szerver, oda belép a felhasználó a szerver operációs rendszerén keresztül.
- Külön gépen van a kliens és külön gépen a szerver és a hálózaton keresztül kapcsolódik a kettő.
- Háromrétegű kapcsolaton keresztül- a programon keresztül lérjük le, ami kapcsolódik az alkalmazás szerveréhez, ami lekérdezi az adatbázist.

Az Oracle struktúra három fő csoportra osztható:

- **Tároló struktúrák**
- **Memória struktúrák**
- **Szerverfolyamatok**

SGA

Database buffer cache - A beolvasott adadtokat tárolja, hogy ne kelljen mindig a lemezről olvasni.(Memória)

Redo log buffer - Memóriából a módosított adat bekerül ide naplózásra, utána a háttértárra kiírásra kerül.

Shared pool:

- **Library** cache - Shared SQL Area - Követi a folyamatot.
  (Beérkezett SQL utasításokat tárolja, átalakítja).
  A végrehajtási terv itt tárolódik.
- **Data dictionary cache** - Adatszótár, leggyakrabban kiolvasott adatokat tárolja.
- **Server result cache** - A lekérdezés eredményeit tárolja.

PGA

- private SQL area - A folyamatoknak saját területük van, ahol a végrehajtási terv tárolódik.
- Session memory - A folyamatoknak saját területük van, ahol a végrehajtási terv tárolódik.
  Az adott session-höz tartozó ideiglenes adatok találhatóak meg.

Adatbázis transzakciók

Összetartozó DML (módosító) utasítások sorozata

Van egy eleje és egy vége. Feltételezzók, hogy a tranzakció kezdetekor az adatok megfelelnek az elvárásainknak és a végén is van egy konzisztens állapot.
Közte lehet egy nem konzisztens állapot.

Egy egységként értelmezzük, konzisztens állapot között mozgunk.

ACID:

- Atomicity - Atomosság - A tranzakció egységesen hajtódik végre, vagy teljesen, vagy semmiképpen.
- Consistency - Konzisztencia - A tranzakció végén a konzisztens állapotban kell lennie.
- Izolation - Izoláció - Azt várja el, hogy amikor pl 2 tranzakció lefutott, akkor utána úgy nézzen ki az adatbázis állapota, mintha külön-külön futott volna.
  (Tud várakoztatni, eldönti mely tranzakciókat fésülhet össze.)
- Durability - Tartóság - Lefutott egy tranzakció, sikeres volt, ...

Gyakorlat

OLTP

Szervezet mindennapos működéséhez tartozó folyamatok (pl boltban való fizetés = módosítási kérések), ezekből sok van, gyakran használt műveletek, általában kevés adatot érintenek.

DSS/DWW

- Egy napban csak egyszer töltünk be adatokat, de akkor nagyon sokat. Több sort, táblát, nagyon sok adatokat érintünk ilyenkor.
- Lekérdezések: Nem teszünk fel sok lekérdezést.
- Olvasás és adatok kombinálása jelenti a fő feladatot.

Aggregáció(s művelet): Több sort érint, több művelet, több adatból állítjuk elő a választ.

author: Albert Dávid neptun code: H1B5EF date: 2023.03.26. lecture: 3_4

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); P

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);