1. Óra

A relációs adatmodell egy olyan adatmodell, amelynek legfontosabb eleme a matematikai (halmaz)reláció fogalma.

- Egyed: az az entitás, amihez az attribútumok tartoznak (amiről az adatokat gyűjtjük).
- Kapcsolat: két reláció közötti viszony.
- Függés: egy reláción belüli adatok közötti viszony.
- Kulcs: az az attribútum, vagy az attribútumok olyan kombinációja, amely egyedi, ezáltal lehetővé teszi a rekordok azonosítását.

Kulcsok fajtái
• Egyszerű: egy attribútumból áll.
• Összetett: több attribútumból áll.
• Kulcsok típusai
• Elsődleges kulcs: a kulcsok közül kiválasztjuk azt amelyik az időben állandó (vagyis nem változhat meg). Ezt használjuk majd a rekordok azonosítására, a különböző táblákban lévő kapcsolódó adatok jelölésére.
• Idegen kulcs: az az attribútum, amelyik a kapcsolódó táblában elsődleges kulcsként funkcionál.

Kapcsolatok típusai
• Egy-az-egyhez: A tábla minden rekordjához egy és csakis egy rekord kapcsolódik B táblában, és B tábla minden rekordjához egy és csakis egy rekord tartozik A táblában.
• Egy-a-többhöz: A tábla minden rekordjához nulla vagy több rekord kapcsolódik B táblában, de B tábla minden rekordjához egy és csakis egy rekord tartozik A táblában.
• Több-a-többhöz: A tábla minden rekordjához nulla vagy több rekord kapcsolódik B táblában, és B tábla minden rekordjához nulla vagy több rekord tartozik A táblában.

Adatmodell szintjei
Konceptuális adatmodellezés
A koncepcionális adatmodell meghatározza a vállalkozás és az adatok általános struktúráját. Üzleti koncepciók rendszerezéséhez használatos, a vállalkozás résztvevői és az adattervezők által meghatározottak szerint. Az entitások és az entitáskapcsolatok egyaránt az elméleti modellben kerülnek meghatározásra.
Logikai adatmodellezés
Egy logikus adatmodell az elméleti modellre épül, az egyes entitásokon belüli adatok meghatározott attribútumaival és az attribútumok közötti meghatározott kapcsolatokkal. Ez az adattervezők és üzleti elemzők által meghatározott szabályok és adatszerkezetek technikai modellje, ami segít döntéseket hozni arról, milyen az adatok és az üzleti igények által megkövetelt tényleges modell.
Fizikai adatmodellezés
A fizikai adatmodell a logikai adatmodell konkrét megvalósítása, amelyet az adatbázisgazdák és a fejlesztők hoznak létre. Egy meghatározott adatbázis-eszközhöz és adattárolási technológiához fejlesztik, és adatösszekötőkkel használható, amelyek az adatokat az üzleti rendszereken keresztül a felhasználók részére szolgáltatják, szükség szerint. Ez az a „valami", amihez a többi modell vezetett – adatállományának tényleges megvalósulása.

Kapcsolatok

• A relációs adatmodell csak az egy-a-többhöz kapcsolattípust támogatja, így minden egyéb relációs kapcsolatot a normalizálás során ilyen formára kell alakítani.
• Technikai okokból előfordul, hogy egy-az-egyhez kapcsolatokat is szerepeltetünk egy relációs adatbázisban, de az elmélet szerint ez egy relációnak számít.
• A kapcsolatok leírására az ERD (entity relationship diagram – egyed kapcsolati diagram) szolgál.

Normalizálás

Az adatbázisok kialakításakor egyik legfőbb feladatunk, a redundancia-mentes adatszerkezet kialakítása. Ennek egyik módja a normalizálás.

A normalizálás során egy kezdeti állapotból több fázison keresztül átalakítjuk az adatbázist. Az átalakítás fázisait normálformáknak nevezzük. Megkülönböztetjük a nulladik, az első, második, harmadik stb. normálformát. A normálformák jelölésére az 1NF, 2NF, 3NF, BCNF jelöléseket használjuk. Az adatbázis kialakítása mindig az alacsonyabbtól a magasabb normálformák felé halad. 1NF->2NF->3NF->BCNF. Sok esetben elég ha az adatokat 3NF formára hozzuk.

A normalizálás alapja a funkcionális függőség. Adatok között akkor áll fenn funkcionális függőség, ha egy vagy több adat konkrét értékéből más adatok egyértelműen következnek.

Az első normálforma azt mondja ki, hogy az attribútumok tartománya kizárólag atomi (egyszerű, oszthatatlan) értékeket tartalmazhat, és hogy a rekordokban bármely attribútum értéke csak egyetlen érték lehet az adott attribútum tartományából.

Az 1NF ezáltal megtiltja, hogy egy rekordon belül az attribútumok értéke egy értékhalmaz, egy érték n-es vagy ezek kombinációja legyen.

A második normálforma pontos definíciója két kritériumnak tesz eleget: a reláció első normál formában van, és a reláció minden nem elsődleges attribútuma teljes funkcionális függőségben van az összes reláció kulccsal.

A reláció nem tartalmaz részleges függést. Részleges függésnek hívjuk, mikor egy másodlagos (nem kulcs) attribútum az összetett kulcs csak egy részétől függ.

A harmadik normálforma azt jelenti, hogy a mennyiben több érték is tartozik hozzá a második normálformában, külön relációba kell kiszervezni az adatokat.

Az SQL nyelv felépítését tekintve az alábbi részekre bontható:

Data Definition Language (DDL)

A CREATE parancs:

Segítségével tudunk létrehozni adatbázisokat, táblákat, mindent ami a séma felépítéséhez szükséges.

Az ALTER parancs segítségével tudunk a meglévő sémáink szerkezetén módosítani.

A DROP parancs segítségével tudunk sémát törölni.

A TRUNCATE parancs segítségével kiüríthető, alapállapotba helyezhető az adott sémaelem.

A COMMENT parancs segítségével megjegyzések fűzhetőek az adatokhoz.

A RENAME parancs segítségével átnevezhetőek a sémák.

Az adatbázis sémájának létrehozásához használatos parancsok.

Data Manipulation Language (DML)

Az adatbázis sémájában lévő adatok manipulálásához szükséges parancsok.

A SELECT parancs segítségével tudunk bizonyos adatsémákból (tábla | nézettábla) adatokat úgy lekérdezni, hogy abban csak azokat az attribútum értékeket látjuk, ami nekünk szükséges (projekció) és csak azokat a rekordokat, amik az adott feltételeknek megfelelnek (szelekció).

Az INSERT segítségével hozzá tudunk adni új rekordot az adott táblához.

Az UPDATE parancs segítségével módosítani tudunk a meglévő rekordok attribútumainak értékein.

A DELETE parancs segítségével törölni tudunk, bizonyos feltételnek eleget tevő rekordokat.

A Call parancs segítségével meghívunk egy PL/SQL vagy Java alprogramot.

Az EXPLAIN PLAN parancs segítségével láthatjuk az adat elérési útját.

A LOCK TABLE egy adatbázison végzett zárolási művelet, amely egy teljes táblát érint, azaz annak egészét védi a párhuzamos módosítások ellen.
Data Control Language (DCL)
Az adatbázis megfelelő elemeihez való hozzáférés megadásában használatos parancsok.
A Data Control Language a jogok kiosztását teszi lehetővé, mellyel kontrollálni tudjuk, hogy ki, mihez fér hozzá.
A GRANT parancs segítségével rendszerjogosultságot adunk.
A REVOKE parancs segítségével visszavonjuk a jogosultságot.
Transaction Control Language (TCL)
Az egyes tranzakciók kezelésére használatos parancsok.
A COMMIT parancs véglegesíti az elkészült munkát.
A SAVEPOINT biztonsági mentési pont
A ROLLBACK parancs segítségével vissza tudjuk pörgetni az utolsó savepoint-ig az összes utasítást, így lényegében az előzőekben kiadott parancsok semmisek.

SQL Standard-ek története

SEQUEL (70's)
SQL-86, első ANSII
SQL-89 / SQL 1
SQL-92 / SQL 2
SQL: 1999 / SQL 3
SQL: 2003
SQL: 2006, XML Query Language (XQuery) support
SQL: 2008

SQL adatbázisok:
MySQL
MariaDB
Oracle
PostgreSQL
MSSQL

NOSQL adatbázisok:
MongoDB
Redis
Cassandra
Elasticsearch
Firebase

2. Óra

Az Oracle architektúrája

1. Indexek

- Keresést gyorsító segédstruktúra
- Egy táblában több mezőre is lehet indexet készíteni
- Egy indexet több mezőre is lehet építeni (összetett index)
- Unique és non-unique index
- Fő fajtái:
  - B-fa index
  - Bitmap index

B-fa index

Ha egy érték többször szerepel a táblában, akkor az indexben is többször fog szerepelni, minden sorazonosítóval külön-külön. A levélblokkok mindkét irányban láncolva vannak, így növekvő és csökkenő keresésre is használható az index. (pl. WHERE o > x vagy WHERE o < y)

Bitmap index

Hasonló a B-fa indexhez, de a levelekben a kulcsérték mellett nem az egyes ROWID-k tárolódnak, hanem egy bittérkép és 2 ROWID.
Az első és utolsó érintett ROWID, valamint a köztük lévő sorokra vonatkozó bittérkép.
Minden sornak egy bit felel meg, ami azokra a sorokra lesz 1-es, amelyek az adott értéket tartalmazzák.
A BITMAP mechanizmus bittáblát hoz létre a tábla rekordjai felett.
Minden rekordhoz pontosan egy sor tartozik a bittáblában, míg a bittáblának annyi oszlopa van, ahány lehetséges értéket felvehetnek a kulcsmezők.
Módosításkor az egész bittérképet zárolni kell, így a bittérkép által érintett sorok sem módosíthatók a tranzakció végéig.
Tipikusan **kicsi szelektivitású mezők** esetén használjuk ezt az indextípust, ahol egy adott értékhez sok mező tartozik.
Olyan mezők felett érdemes bitmap jellegű indexelést fenntartani, melyekre gyakran futtatunk aggregátum jellegű - statisztikai - lekérdezést (pl: OLAP rendszerek esetén).

2. Adatszótár

- Oracle adatbázis-kezelő esetén a metaadatok az adatszótárban (data dictionary) tárolódnak
- Data dictionary : központi, csak olvasható referencia táblák és nézetek

Tartalma:
AB objektumok definíciója;
AB objektumok számára allokált és felhasznált területek;
oszlopok alapértelmezett értékei;
integritás megszorításokról információk;
az adatbázis felhasználóinak nevei;
az egyes felhasználókhoz tartozó jogok és szerepek;
naplózási információk;
egyéb általános adatbázis információk
Data dictionary szerkezete:

Base tables: adatbázisról infók; csak az adatbázis olvashatja és írhatja
User-Accessible Views: felhasználók számára megjelenített információk
Minden base table és user-accessible view a SYS sémában van

3. Adatbázis architektúrák

SEQUEL: Structured English Query Language, SQL elődje, ezért van a mai kiejtés
Szerver op. rsz.: **Linux**, Windows, Mac OS X, ...
Adatbázis instance kapcsolat
Mindegyik adatbázis példány pontosan egy adatbázishoz kapcsolódik
Ha ugyanazon a szerveren több adatbázis is található, akkor mindegyikhez külön-külön példány tartozik. Egy példány nem lehet megosztott.
Real Applications Cluster (RAC) konfiguráció esetén külön szervereken futó példányok kapcsolódnak ugyanahhoz az adatbázishoz
Egy SQL utasítás végrehajtásának folyamata
Mielőtt egy felhasználó küld egy SQL utasítást az ORACLE szervernek, előtte kapcsolódnia kell egy példányhoz.
A felhasználói alkalmazások, vagy például az iSQL*Plus felhasználó folyamatokat (user process) indítanak el.
Amikor a felhasználó az Oracle szerverre kapcsolódik, akkor készül egy folyamat, amit szerver folyamatnak (server process) hívunk.
Ez a folyamat kommunikál az Oracle példánnyal a kliensen futó felhasználó folyamat nevében. A szerver folyamat hajtja végre a felhasználó SQL utasításait.
Egy példányhoz kapcsolódáshoz szükséges tehát felhasználói folyamat és szerver folyamat
Az SQL utasítás típusától függ, hogy az Oracle szerver milyen komponenseire lesz szükség:
Az Oracle szerver nem minden komponense vesz részt egy SQL utasítás végrehajtásában.
Kapcsolódás Oracle szerverhez
OPERÁCIÓS RENDSZEREN KERESZTÜL:
A felhasználó belép abba az operációs rendszerbe, ahol az Oracle példány fut és elindít egy alkalmazást, amely eléri az adatbázist ezen a rendszeren. Ekkor a kommunikáció útvonalat az operációs rendszer belső kommunikációs folyamatai hozzák létre.
KLIENS-SZERVER KAPCSOLATON KERESZTÜL:
A felhasználó a helyi gépén elindít egy alkalmazást, amely a hálózaton keresztül kapcsolódik ahhoz a géphez, amelyen az Oracle példány fut. Ekkor a hálózati szoftver kommunikál a felhasználó és az Oracle szerver között.
HÁROMRÉTEGŰ (three-tiered) KAPCSOLATON KERESZTÜL:
A felhasználó gépe a hálózaton keresztül kommunikál egy alkalmazással vagy egy hálózati szerverrel, amely szintén a hálózaton keresztül össze van kötve egy olyan géppel, amelyen az Oracle példány fut. Például a felhasználó egy böngésző segítségével elér egy szerveren futó alkalmazást, amely egy távoli UNIX rendszeren futó Oracle adatbázisból gyűjti ki az adatokat.
Az Oracle struktúrái három fő csoportra oszthatók:

- Tároló struktúrák (az adatbázis)
  - Fizikailag: különféle típusú fájlok
  - Logikailag
- Memória struktúrák
  - SGA (System Global Area)
  - PGA (Program Global Area)
- Szerverfolyamatok
  - az adatbázis-műveletek végrehajtásáért felelősek

SGA adatszerkezet

- **Database buffer cache**: A beolvasott adatblokkok pufferterülete

- **Redo log buffer**: A helyreállításhoz szükséges redo napló pufferterülete, innen íródik ki a napló a lemezen tárolt redo naplófájlokba
- **Shared pool**: A felhasználók által használható közös puffer
- Large pool: A nagyon nagy Input/Output igények kielégítéséhez használható puffer
- Java pool: A Java Virtuális Gép (JVM) Java kódjaihoz és adataihoz használható puffer
- Streams pool: Az Oracle Stream-ek pufferterülete

Shared pool fő komponensei

- Az elemzés (parse) alatt a szerver folyamat az SGA-nak ezen a részén fordítja le (compile) az SQL utasítást
- A méretét a SHARED_POOL_SIZE inicializáló paraméter állítja be.

Könyvtár library cache

- Az utoljára kiadott SQL utasításokról tartalmaz információt a közös SQL terület (shared SQL area) nevű memóriaszerkezetben:
- Ha egy SQL utasítást újra végrehajtunk és a közös SQL terület már tartalmazza a végrehajtási tervet, akkor nem kell újra lefordítania a szerverfolyamatnak az utasítást. Ezzel időt és memóriát lehet spórolni.
- Ha sokáig nem használják fel az SQL utasítást, akkor kikerül a Cache-ből.

Adatszótár cache

- Az adatszótár utoljára használt definícióit tartalmazza.
- Tartalmazhat információkat adatbázisfájlokról, táblákról, indexekről, oszlopokról, felhasználókról, jogosultságokról és más objektumokról is.
- Elemzés (parse) alatt a szerverfolyamat a nevek feloldásához, jogosultságok megállapításához először itt keresi az információt. Ha itt nem találja meg, akkor kezdeményezi a szükséges információk beolvasását az adatfájlokból.

Database Buffer Cache

- Az utoljára beolvasott blokkokat tárolja.
- A mérete a DB_BLOCK_SIZE inicializáló paramétertől függ.
- A pufferek számát a DB_BLOCK_BUFFERS adja meg.
- Működése:
  - Amikor egy lekérdezést kell végrehajtani, a szerverfolyamat először megnézi, hogy a keresett blokk nincs-e itt. Ha nem találja a pufferben, csak akkor olvassa be a blokk másolatát az adatfájlból.
  - Ha már nincs hely a pufferben, akkor a legrégebben használt adatblokk helyére olvassa be az újat.

Redo log buffer

- Körpuffer az SGA területen
- Az adatbázisban végrehajtott módosításokról tárol információt
- Redo bejegyzéseket tartalmaz, amelyek lehetővé teszik a DML és DDL műveletek újbóli végrehajtását
  - hiba utáni helyreállítás
- Tartalmát a log writer háttérfolyamat az aktív redo log fájlba írja

PGA

Nincs megosztva a folyamatok között, csak a hozzá tartozó szerverfolyamat írhatja

A szerver folyamat indulásakor foglalja le ezt a területet, befejezéskor pedig felszabadítja.

A PGA általában a következőkből áll:

Private SQL area: Futási időben szükséges memóriaszerkezeteket, adatokat, hozzárendelési információkat tartalmaz.

Memória-intenzív műveletekhez (rendezés, csoportosítás, join) szükséges terület

Session memory: A munkamenethez (session) tartozó információk, változók tárolásához szükséges terület.

Folyamatok

Felhasználói folyamat (**User process**)

Adatbázis folyamatok (Database processes)

szerver folyamatok (a felhasználói munkamenet indulásakor jön létre)

háttérfolyamatok (a példánnyal együtt indulnak el)

Daemonok, alkalmazás folyamatok

nem adatbázis-specifikus folyamatok

pl. hálózati kommunikáció kezelése

Az Oracle egy példány indításakor háttérfolyamatokat is elindít, amelyek kommunikálnak egymással és az operációs rendszerrel.

Amikor egy alkalmazás vagy Oracle eszköz (mint például az Enterprise Manager) kapcsolódik az adatbázishoz, akkor az Oracle szerver elindít egy szerverfolyamatot, amely lehetővé teszi az alkalmazás utasításainak végrehajtását.

A háttérfolyamatok kezelik a memóriát, puffereket, végrehajtják az írási, olvasási műveleteket a lemezen, karbantartásokat végeznek.

Legfontosabb háttérfolyamatok

Database writer (DBWn): Az adatpufferből (database buffer cache) kiírja lemezre, egy adatfájlba a módosított blokkokat

Log writer (LGWR): A Redo log buffer bejegyzéseit írja ki a lemezre a Redo napló fájl(ok)ba

System monitor (SMON): Adatbázis karbantartási feladatok (pl. katasztrófa utáni helyreállítás, undo és temp táblaterek karbantartása, inkonzisztens állapotú adatszótár tisztítása)

Process monitor (PMON): Monitorozza a többi háttérfolyamatot, szükség esetén elindítja és leállítja őket; ha egy folyamat vagy munkamenet megszakad, akkor elvégzi a szükséges takarítást

4. Adatbázis tranzakciók

Az adatbázis-tranzakciók biztonságos és kiszámítható programozási modellt biztosítanak az adatok egyidejű változásainak kezeléséhez.

A tranzakció egy egység, mely során különböző adatokat olvasunk az adatbázisból illetve módosítunk

A tranzakciónak konzisztens adatbázis képet kell látnia

A tranzakció végrehajtása során az adatbázis időlegesen inkonzisztens lehet

A tranzakció befejezésével (Commit) az adatbázisnak konzisztens állapotba kell kerülnie.

Atomicitás

Az atomicitás megköveteli, hogy több műveletet atomi (oszthatatlan) műveletként lehessen végrehajtani, azaz vagy az összes művelet sikeresen végrehajtódik, vagy egyik sem.

Konzisztencia

A konzisztencia biztosítja, hogy az adatok a tranzakció előtti érvényes állapotból ismét egy érvényes állapotba kerüljenek. Minden erre vonatkozó szabálynak (hivatkozási integritás, adatbázis triggerek stb.) érvényesülnie kell.

Izoláció

A tranzakciók izolációja azt biztosítja, hogy az egy időben zajló tranzakciók olyan állapothoz vezetnek, mint amilyet sorban végrehajtott tranzakciók érnének el. Egy végrehajtás alatt álló tranzakció hatásai nem láthatóak a többi tranzakcióból.

Tartósság

A végrehajtott tranzakciók változtatásait egy tartós adattárolón kell tárolni, hogy a szoftver vagy a hardver meghibásodása, áramszünet, vagy egyéb hiba esetén is megmaradjon.

3. Óra

SQL tuning

Az SQL tuning a lassan futó lekérdezések "hangolását" jelenti annak érdekében, hogy megfeleljenek a teljesítménybeli elvárásoknak, Oracle-ben nagyrészt automatikus, de meg lehet változtatni az alapbeállításokat. Az SQL utasítás végrehajtása a következő részekből áll: Parsing (Hard Parse, Soft Parse), Optimalizálás, Row Source generálás, végrehajtás. A parszolás része a szintaktikai ellenőrzés (Syntax Check), szemantikai ellenőrzés (Semantic Check), Shared Pool Check. Szintaktikai hibának számít pédául a hiányzó vessző, elírt kulcsszó, elfelejtett idézőjel. A szemantikai ellenőrzés alatt megnézzük, hogy a lekérdezés "értelmes-e". Szemantikai hiba pl. nem létező táblára vagy változóra hivatkozás, adattípusok nem stimmelnek. Shared Pool check esetén megnézzük, hogy végrehajtottuk-e már ezt a lekérdezést korábban vagy nem, hash értéket generálunk a lekérdezéshez (SQL ID). Ezt a hash értéket összehasonlítjuk a korábban futtatott utasítások hash értékeivel, ha a korábban futtatott lekérdezések hash értékei között van találat, akkor szemantikai ellenőrzést és környezet ellenőrzés hajtunk végre.

Az optimalizálás alatt az SQL esetében az optimális végrehajtás megtervezését értjük. Az Oracle végrehajtási tervet (Execution plan-t) hoz létre, ami megmutatja, hogy az adott lekérdezést hogyan hajtjuk végre.

A CBO (Cost-Based Optimization) esetén létrehozunk több végrehajtási tervet, mindhez kiszámítjuk a költséget, ami adott környezetben a művelet becsült erőforrásigényét reprezentáló számérték, ezután a legkisebb költségű végrehajtási tervet választjuk.

A Row Source generálás az optimális végrehajtási terv "lefordítása" az SQL motor által végrehajtható lépések sorozatára.

A Row Source Three a Row Source Generator által előállított fastruktúra, amely megmutatja:a táblák sorrendjét, a hozzáférés módját az egyes táblákhoz,a táblák összekapcsolásának módját (ha van a lekérdezésben join), egyéb műveleteket (szűrés, rendezés, csoportosítás). Ennek a feldolgozása post-order bejárással: "először a gyereke(ke)t, aztán a szülőt", mivel a szülő feldolgozásához szükség van a gyerekek által meghatározott row source-okra.

Az access path megmutatja, hogyan olvasunk ki egy row source-t. Unáris művelet, bemenete és kimenete is egyetlen row source, különféle adatstruktúrákra különböző hozzáférési módokkal rendelkeznek. Access path típusok: Heap organized tables (Full Table Scan, Table Access by Rowid, Smple Table Scan), B-Tree indexes (Index Unique Scan, Index Range Scan, Index Full Scan, Index Fast Full Scan, Index Skip Scan, Index Join Scan), Bitmap indexes (Bitmap Index Single Value, Bitmap Index Range), Table clusters (Cluster Scans, Hash scans).

A Full Table Scan minden sort beolvas a táblából, a CBO akkor választja ezt, ha nem áll rendelkezésre más mód (pl. nincs index), ha"Full table scan" hint van megadva, ha az egyéb módszerek költségesebbek lennének, túl kicsi a tábla, túl kicsi szelektivitású oszlopra van WHERE feltétel megadva.

Table Access by Rowid esetén egyetlen rekordot olvasunk ki Rowid alapján, nagyon gyors, általában valamilyen index scan után választja ezt a CBO.

Az index bizonyos esetekben gyorsítja a rekordokhoz való hozzáférést.

Az index unique scan legfeljebb egy rowid-ot ad vissza, akkor választja ezt a CBO ha egyenlőség feltételünk van olyan mező(k)re, amin unique index van. Ahhoz, hogy a unique

index scan működjön, az adott oszlopon kell, hogy legyen index, amit létre kell hozni vagy létrejöhet magától (pl: Primary key).

Az Index Range Scan esetén egy értéktartományhoz tartozó rowid-kat olvasunk ki az indexből növekvő sorrendben. Az index range scan descending: ugyanez, csak visszafelé. Ezt választja a CBO ha egyenlőtlenség jellegű feltételünk van indexelt mező(k)re (between, <, >, ...), vagy egyenlőség, de nem unique index van rajta, vagy összetett index esetén az első mezőre.

Az Index full scan esetén a teljes indexet kiolvassuk sorrendben, ezáltal megspórolhatjuk a rendezést. Ezt választja a CBO ha indexelt nem-null mezőre rendezést kérünk, ha feltételünk van indexelt mezőre (nem kell, hogy első legyen), ha nincs ugyan feltétel, de minden listázandó mező benne van az indexben és legalább egy indexmező nem null.

Az index fast full scan a teljes index kiolvasása, de nem sorrendben, hanem ahogy épp a lemezen található, ezért nem spórolja meg a rendezést. Ezt akkor használja a CBO, ha elegendő csak az indexet végigolvasni az eredmény meghatározásához, nem szükséges a táblához hozzáférni vagy ha tipikusan index join vagy indexen végrehajtott aggregálás esetén.

Az Index skip scan esetén egy összetett index első mezőjét figyelmen kívül hagyjuk / nem szükséges. Ezt akkor használja a CBO, ha összetett index többedik mezőjére van feltétel vagy ha első indexmezőnek kevés lehetséges értéke van, míg a többinek sok.

Az Index join scan esetén több indexet kötünk össze hash joinnal, az összekapcsolás Rowid alapján történik. Akkor választja ezt a CBO, ha az összekapcsolt indexek együtt tartalmaznak minden hivatkozott mezőt.

A joinok végrehajtásához meg kell határozni minden row source-ra külön- külön az access path-t, row source páronként a join módszerét, a join típusát, kettőnél több táblás join esetén a join sorrendjét is. A cél az, hogy minél kevesebb rekorddal kelljen dolgozni. A CBO arra törekszik a módszer és a sorrend megválasztásánál, hogy minél korábbi lépésben lecsökkenjen a sorok száma (pl. ha vannak a táblák között olyanok, amelyeket összekapcsolva egyetlen sort kapunk, akkor ezt végzi el először).

Join módszerek: nested loop join, sort-merge join, hash join, Cartesian join.

Nested loop join esetén a külső (driving) row source minden sorához megkeresi a belső row source-ból a hozzá tartozót. Ha rendelkezésre áll megfelelő index a belső row source-ra, akkor használhat rowid szerinti hozzáférést. Általában a kisebb row source-t választja külsőnek. Akkor használja ezt a CBO, ha a row source-ok kevés rekordból állnak vagy ha FIRST_ROWS módban vagyunk, vagy ha hatékonyan tudjuk olvasni a belső row source-t (index).

A Sort merge join esetén csak egy ciklust van, rendezi a row source-okat (ha szükséges), így hatékonyabb megtalálni a párokat, az első row source minden sorához megkeresi a második row source-ban az első egyező kulcsot, majd addig olvassa a második row source rekordjait, amíg nem talál eltérő kulcsot. Ezt nagyobb adathalmazok esetén választja a CBO, ha van megfelelő index a külső row source-ra, ezért nem kell rendezni, egyéb okból amúgy is szükséges rendezni, nem equijoinról van szó, hanem egyenlőtlenségi feltétellel kapcsolunk össze.

A Hash join esetén a kisebbik row source-ra (a join mezőre) egy determinisztikus hash függvény segítségével hash táblát generálunk, amely tárolja az adott mező értékhez tartozó sort. Ezután a nagyobb row source sorait bejárva csak kiolvassa a hash táblából a hozzájuk tartozó sort. Ezt akkor választja a CBO, nagyobb adathalmazok esetén, ha equijoinról van szó, ha kisebb adathalmaz befér a memóriába (ebben az esetben különösen gazdaságos).

A Cartesian join vagyis cross join (keresztszorzat, direkt szorzat, Descartes-szorzat) olyan join, ahol nincs összekapcsolási feltétel, minden sort minden sorral párosítunk. Nem önálló join módszer, az előzőek közül valamelyikkel állítja elő a rendszer. Akkor használja ezt a CBO, ha nincs join feltétel, ha olyan join sorrendet kényszerítünk ki több tábla között, hogy a szomszédos tábláknak nincs közvetlen kapcsolata, vagy ha ez a hatékony megoldás (Pl. két nagyon pici táblánál, ha join-olni kell mindkettőt egy nagy táblára). Execution plan megjelenítése:

EXPLAIN PLAN FOR

SELECT e.employee_id, e.last_name

FROM employees e

WHERE e.employee_id = 19;

Lekérdezések optimalizálása

A hintek - tippek az optimalizálónak - segítségével kényszeríthetünk ki access path, join módszer, join sorrend választást. Nem ajánlott meggondolatlanul használni, a hatékonysága nagyon függ a környezettől, sok esetben csak csak tesztelésre használjuk.

Szintaktikailag: speciális komment -> SELECT /*+ hint_text */ ...

A hintet a SELECT, UPDATE, INSERT, MERGE, vagy DELETE után írjuk. A + közvetlenül a komment jel után következzen, nem szabad szóközt tenni közé. Csak egyetlen hintet tartalmazó komment lehet egy kifejezésben, de az tartalmazhat több, szóközzel elválasztott hintet. A szintaktikai hibás hintet figyelmen kívül hagyja a rendszer.

Gyakran használt hintek:

INDEX(tábla indexnév) - Index scan a megadott tábla megadott indexén INDEX(tábla indexnév) ;

INDEX_DESC - csökkenő sorrendű index scan

INDEX_FFS - index fast full scan

INDEX_SS - index skip scan

INDEX_JOIN(tábla indexnév1 indexnév2) - index join scan a megadott indexek összekapcsolásával

FULL - full table scan a megadott táblára

USE_NL - nested loop join, ahol a megadott tábla lesz a belső

USE_MERGE - sort-merge join

USE_HASH - hash join

ORDERED - Olyan sorrendben végezze az összekapcsolást, amilyen sorrendben a FROM-ban szerepel (nincs paramétere)

LEADING - Mely táblákkal kezdje az összekapcsolást (paraméterként megadható táblasorrend). Ha mindkettőt megadjuk, az ORDERED hint felülbírálja a LEADING hintet.

Az Oracle inkább a LEADING-et javasolja, mivel rugalmasabb.

FIRST_ROWS - válaszidőre optimalizál: az első n db sort a lehető leggyorsabban állítsa elő

ALL_ROWS - erőforrás felhasználásra optimalizál: összességében legyen kis költségű

UNNEST - az allekérdezést „olvassza be” az őt tartalmazó lekérdezésbe, és a CBO ennek megfelelően válasszon access path-okat és join módszereket

NO_UNNEST

NO_QUERY_TRANSFORMATION - semmilyen transzformációt ne végezzen

MATERIALIZE - az allekérdezésből készítsen ideiglenes táblát

INLINE - az előző ellentéte

Statisztikák

- Táblákról (pl. sorok száma, blokkok száma, stb.)
- Oszlopokról (pl. különböző értékek száma (NDV) egy oszlopban, NULL-ok száma, adatok eloszlása, stb.)
- Indexekről (levél blokkok száma, famagasság, stb.)
- Rendszerről (I/O, CPU teljesítmény és kihasználtság)

Hisztogramok

A CBO alapértelmezetten egyenletes eloszlásúnak feltételezi az adatokat, azaz az előforduló különféle értékekből kb ugyanannyi van egy oszlopban. Ha ez nem így van, akkor a join és szűrő kifejezések becslése nagyon pontatlan lehet. A hisztogram megmutatja, hogy melyik érték milyen gyakori egy oszlopban. Ennek ábrázolása "bucket"-ekkel Az endpoint value: a bucketben lévő legnagyobb érték. Az endpoint number: a bucket egyedi azonosítója (változó a szerepe). Segít a pontos kardinalitás-becslésben: kardinalitás=hány sorból fog állni az adott művelet eredményeként kapott row source. Frequency - minden érték külön bucketbe. Top frequency: ugyanez, csak a ritka értékeknek nincs bucket. Height-balanced, régebbi típus, úgy alakítja a bucketeket, hogy mindegyikbe közel ugyanannyi rekord essen (de aztán az azonos endpointtal rendelkezőket összevonja). Hybrid: az előző kettő előnyeit igyekszik egyesíteni, egyenlő szétosztásra törekszik, de olyan módon, hogy az azonos értékek ne legyenek megosztva több bucket között.

Index clustering factor

Megmutatja, hogy a B-fa indexen végrehajtott index-scan mennyire költséges I/O szempontból. Az egymás utáni index-bejegyzésekhez tartozó tábla rekordok mennyire tárolódnak közel egymáshoz az adatfájlban. Az index sorrendben történő kiolvasása során mennyit kell "ugrálni" az adatfájl blokkjai között.

Alacsony clustering factor esetében: az érték közelebb van a tábla adatblokkjainak számához, az indexben egymás után következő rekordok a táblában is közel vannak egymáshoz, az index scan így hatékony I/O szempontból. Magas clustering factor: az érték közelebb van a tábla rekordjainak számához, az indexben egymás után következő rekordok a táblában elszórva helyezkednek el, az index scan így kevésbé hatékony I/O szempontból.

A lekérdezés optimalizálás folyamata

0. lépés: információ gyűjtés, táblák szerkezetéről, adattípusokról. Megnézzük, hogy milyen megszorítások, indexek, materializált nézetek vannak. Megnézzük, hogy statisztikák frissek-e az adatok, van-e hisztogram.

1. lépés: explain plan segítségével azonosítjuk a magas költségű műveleteket.

2. lépés: Másik join módszer tesztelése, join algoritmus hintek alkalmazása (use_nl, use_hash, use_merge), allekérdezések hintjei alkalmazása: inline, materialize, no_query_transformation. Cardinality becslés pontosságának a kiértékelése, statisztika frissítés, hisztogram készítése.

3. lépés: Lekérdezés átalakítása, korrigálása. Helyesség ellenőrzése (összekapcsolási feltételek). Másik join típusok kipróbálása (pl. inner helyett left, ha amúgy mindegy). Allekérdezések használata pl. ha elkerülnénk a sort group by-t. Subquery factoring.

4. lépés: Index készítés idegen kulcsokra, ha join-olunk, feltétel mezőkre és function-based index használata.

Csoportosítás végrehajtása

A SORT GROUP BY a 10.2 verzió előtt is létező megoldás, a csoportosító mező szerint rendez, így az azonos csoportba tartozó rekordok egymás után következnek majd.

A HASH GROUP BY a 10.2 verziótól elérhető, általában gyorsabb a rendezés alapú megoldásnál, hash táblába végzi az aggregálást.

Az újabb verziókban is SORT GROUP BY-t választ a CBO, ha a row source rendezett, vagy amúgy is rendezni kell. Ez azonban teljesítmény-csökkenéshez vezethet. Nem mindegy, hogy csoportosítás előtt vagy után rendezünk, hiszen utána tipikusan sokkal kisebb a rekordok száma. A lekérdezés átalakításával megoldható. A csoportosítást tegyük allekérdezésbe, külön az ORDER BY-tól.

Query rewrite

Szükséges a HR-nek jogosultságot adni a SYS felhasználóval:

grant create materialized view to hr;

CREATE MATERIALIZED VIEW szaztiz

ENABLE QUERY REWRITE AS

SELECT * FROM employees WHERE department_id=110;