Adatbázisok 2.

Készítette:
Gerstweiler Anikó Éva- GEAQAAI.ELTE
Molnár Dávid - MODQABI.ELTE

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

1. Az Oracle adatbázis-kezelő felépítése, működése, komponensei, példányok, rendszerállományok, memóriakezelése, rendszergazdai feladatok.

Felépítése:

Két fő részből áll, az adatbázisból, vagyis a fizikai struktúrák rendszeréből, és a példányból, vagyis a memória struktúrák és folyamatok rendszeréből. Az előbbi rendszer a vezérlő fájlból, a helyrehozó napló fájlokból, az adatfájlokból, a paraméterfájlból és a jelszófájlból áll, míg az utóbbi rendszert a memóriában lefoglalt System Global Area (SGA) és az adatbázis-műveletek végrehajtásáért felelős szerver- és háttérfolyamatok alkotják.

Működése:

Mielőtt egy felhasználó küld egy SQL utasítást az Oracle szervernek, 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 kerül létrehozásra egy folyamat, amit szerverfolyamatnak hívunk. Ez a folyamat kommunikál az Oracle példánnyal a kliensen futó felhasználó folyamat nevében. A szerver folyamat hajtja végre a felhasználó SQL utasításait. A kapcsolat egy kommunikációs útvonal a felhasználó folyamat és az Oracle szerver között. Háromféleképp lehet egy Oracle szerverhez kapcsolódni:

- 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 szoftvert 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 NT szerveren futó alkalmazást, amely egy távoli UNIX rendszeren futó Oracle adatbázisból gyűjti ki az adatokat.

Kapcsolódáskor egy munkaszakasz (session) kezdődik. A munkaszakasz a felhasználó érvényesítése (validálása) esetén kezdődik és a kilépéséig vagy egy abnormális megszakításig tart. Egy felhasználó több munkaszakaszt is nyithat. Ehhez szükséges, hogy az Oracle szerver elérhető, használható legyen.

Komponensek:

Amikor egy alkalmazás vagy Oracle eszköz (mint például az Enterprise Manager) elindul, akkor a szerver elindít egy szerverfolyamatot, amely lehetővé teszi az alkalmazás utasításainak végrehajtását. Az Oracle egy példány indításakor háttérfolyamatokat is elindít, amelyek kommunikálnak egymással és az operációs rendszerrel. A háttérfolyamatok kezelik a memóriát, puffereket, végrehajtják az írási, olvasási műveleteket a lemezen, karbantartásokat végeznek. A legfontosabb háttérfolyamatok a következők:

- System monitor (SMON), mely katasztrófa utáni indításkor elvégzi a helyreállítást.
- Process monitor (PMON), mely ha egy felhasználói folyamat megszakad, akkor elvégzi a szükséges takarítást, karbantartást.
- Database writer (DBWn), mely az adatpufferből kiírja lemezre (fájlba) a módosított blokkokat.
- Checkpoint (CKPT), mely ellenőrzési pontok esetén elindítja a DBWn folyamatokat és frissíti az adatbázis összes adatállományát és vezérlő állámányát.
- Log writer (LGWR), mely a helyreállítási napló bejegyzéseit írja ki a lemezre.
- Archiver (ARCn), mely a helyreállítási napló állomány másolatait a mentésre kijelölt lemezekre írja ki, mikor egy naplófájl betelik vagy a következő online redo naplóba folytatódik az írás (log switch).

Példányok:

Egy Oracle példány két részből áll: memória pufferekből, melyet System Global Area-nak (SGA) nevezünk, és háttérfolyamatokból. A példány tétlen (idle) állapotban van, amíg nem indítják el. Indításnál a példány a paraméter fájlból kiolvasott adatoknak megfelelően lesz konfigurálva. Miután a példány elindult, és az adatbázis nyitva van, a felhasználók hozzáférhetnek az adatbázisban tárolt adatokhoz.

Rendszerállományok:

A fontosabb rendszerállományok: a vezérlő fájl (Control file), a helyrehozó napló fájlok (Redo Log files), a paraméterfájl (Parameter file) és a jelszófájl (Password file).

A példány indításakor, amikor az adatbázist felcsatlakoztatjuk, be kell olvasni a vezérlő fájlt. A vezérlő fájl megléte kritikus, nélküle nem lehet megnyitni az adatfájlokat, ugyanis a benne lévő bejegyzések határozzák meg az adatbázist alkotó fizikai fájlokat. Ha további fájlokat adunk az adatbázishoz a vezérlő fájl automatikusan módosul. A vezérlő fájl helyét az inicializálási paraméterben adjuk meg. Adatvédelmi szempontból fontos, hogy legalább három különböző fizikai eszközön legyen másolata (multiplexálás). Ez is inicializálási paraméterrel adható meg, több fájlnév megadása esetén az Oracle szerver elvégzi a többszörös másolatok szinkronizációját.

A napló állományok az adatbázisbeli változásokat rögzítik, melyek tranzakciók és belső szerver akciók következményei. Védik a konzisztens állapotot az áramkimaradás, lemezhiba, rendszerhiba esetén. Multiplexálni kell őket adatvédelmi okokból, nehogy egy esetleges lemezhiba esetén elvesszen a napló tartalma. Maga a napló fájlok csoportjaiból áll, egy-egy csoportban (melyek azonosítószámmal vannak ellátva) egy fájl és a másolatai vannak. A naplóíró folyamat (Log Writer Process - LGWR) írja ki a helyrehozó rekordokat a pufferből egy csoportba, amíg vagy tele nem lesz a fájl, vagy nem érkezik egy direkt felszólítás, hogy a következő csoportba folytassa a kiírást. A helyrehozó csoportok feltöltése körkörösen történik.

A paraméter fájl határozza meg hogy a példány hogyan lesz konfigurálva indításnál, tartalmazza például az SGA memóriarészeinek méretét.

A jelszó fájl miatt lehetséges távolról csatlakoznia felhasználóknak, hogy adminisztratív feladatokat végezhessenek, meghatározza kik indíthatnak el és állíthatnak le egy példányt.

Memóriakezelés:

Egy Oracle példányhoz tartozó memóriaszerkezet két fő részből áll, a System Global Area-ból (SGA) és a Program Global Area-ból (PGA). Az SGA egy megosztott memóriaterület, minden szerver- és háttérfolyamat számára elérhető, továbbá a példányhoz tartozó adatokat és vezérlési információkat is tartalmazhat. A PGA ezzel szemben privát memóriaterület, azaz minden szerver- és háttérfolyamathoz külön, saját PGA tartozik. Az SGA a következő adatszerkezetekből áll:

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

Az SGA dinamikus, azaz a pufferek mérete szükség esetén a példány leállítása nélkül változtatható. A PGA mérete és tartalma attól függ, hogy a példányt osztott módra (Shared Server Mode) konfiguráltuk-e. A PGA általában a következőkből áll:

- Private SQL area, mely futási időben szükséges memóriaszerkezeteket, adatokat, hozzárendelési információkat tartalmazza. Minden olyan munkaszakasz (session), amely kiad egy SQL utasítást, lefoglal egy saját SQL területet.
- Session memory, mely a munkaszakaszhoz (session) tartozó információk, változók tárolásához szükséges terület.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

Rendszergazdai feladatok:

Az Oracle Adatbázis Rendszergazda (Oracle Database Administrator - DBA) feladata az adatbázis megtervezése, megvalósítása, és karbantartása. Elsőként fel kell mérnie a szerver hardverét, meg kell győződnie annak megfelelőségéről (kapacitás, hiánytalanság, stb.), majd telepítenie kell az Oracle szoftvert (ezek ismerete szintén elengedhetetlen). Ezek után az adatbázist létre kell hoznia, a felhasználók számára elérhetővé kell tennie, és meg kell valósítania az adatbázistervet. Mindezek mellett gondoskodnia kell a megfelelő biztonsági mentésekről, az esetleges hibák utáni helyreállításról, és felügyelnie kell az adatbázis teljesítményét.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

2. Lemezegységek, blokkok, fájlok felépítése, RAID megoldások.

Lemezegységek:

Az elsődleges tárak a gyorsítótár és a központi memória, ezek a leggyorsabbak ugyanakkor a legdrágábbak is. A másodlagos tárak a flash memória és a mágneslemez, melyek olcsóbbak, de lassabbak is. Külső tárak pedig az optikai lemez és a mágnesszalag, ezek a legolcsóbbak, de a leglassabbak is. A leggyakrabban a mágneslemezeket használják, a tetszőleges elérés, a viszonylagos olcsóság, és a nem felejtés miatt.

Blokkok:

A lemezek korongokból állnak, melyeket mágneses tároló réteg borít. A korongfelület logikai felosztását sávnak nevezzük. A sáv hardver általi felosztása a szektor, az operációs rendszer általi felosztása a blokk. Tipikus blokkméret az 512B, 2kB, 4kB.

A lemezek I/O-ja megegyezik a blokkok I/O-jával, a hardvercímet át kell alakítani cilinder, felület, és szektor számmá.

Fájlok felépítése:

Minden adatbázis logikailag egy vagy több táblatérre van felosztva. A táblaterek egy vagy több fájlból állnak. Az adatfájlok mindig csak egy táblatérhez tartoznak. A táblatér mérete szerint kétféle lehet:

- Nagy fájlból álló táblatér (bigfile tablespace), mely egyetlen fájlból áll, de 4Gb-nyi blokkot tartalmazhat.
- Kis fájlokból álló táblatér (small file tablespace), mely több kisebb fájlból áll.

Az adatbázis objektumai (mint például a táblák és indexek) szegmensként vannak eltárolva a táblatérben, így egy táblatér több szegmensből is állhat. A szegmensek egy vagy több extensből állnak, melyek továbbbonthatóak folyamatos adatblokkokra. Ez azt jelenti, hogy egy extens egy adatfájlban lehet eltárolva. Az adatblokkok az adatbázis legkisebb írható/olvasható egységei, és operációs rendszerbeli blokkokra képezhetők le. Az adatblokkok mérete az adatbázis létrehozásakor adható meg (alapbeállítás szerint 8Kb, mely a legtöbb adatbázishoz megfelelő). Statikus adatbázis (adattárház) esetén nagyobb méretet érdemes használni, dinamikus adatbázis (tranzakciós adatbázis) esetén kisebbet. A maximális méret operációs rendszer függő, a minimális méret 2Kb, de ezt ritkán ajánlott használni.

RAID megoldások:

Olcsó Lemezek Redundáns Tömbje (RAID - Redundant Array of Inexpensive Disks), mely általában azonban nem igaz, a fejlettebb RAID megoldások még ma sem olcsóak. A RAID-nek 7 szintje van (0.-6.-ig), és a fő jellemzőik a megbízhatóság, a redundancia, és a párhuzamosság.

A 0. szintű RAID gyakorlatilag csak lemezek összekapcsolása, a végső kapacitás a lemezek kapacitásának összege. Blokkszintű csíkozás jellemzi, maximális sávszélességgel, automatikus terheléselosztással. Annak ellenére, hogy a legjobb az írási teljesítménye, nem elterjedt, hiszen a nem redundáns tárolás miatt nem megbízható.

Az 1. szintű RAID tükrözést használ ("Két megegyező példány tárolása két különböző lemezen."), így a kapacitás gyakorlatilag a lemezek kapacitásának összegének fele. Szekvenciális írás, párhuzamos olvasás jellemzi, átviteli sebessége hasonló egyetlen lemezéhez. Szintén nem terjedt el a használata, mivel a megoldások közül a legdrágább.

A 2. és 3. szintű RAID bitcsíkozást használ, és itt már megjelenik a hibafelismerés és -javítás is. A RAID 2 hibajavító kódot (ECC) használ, és csak kicsivel olcsóbb az 1. szintűnél, így nem használják. A RAID 3 a 2. szintű RAID továbbfejlesztése blokkonként egy paritásbittel, továbbá a hibafelismerés a lemezvezérlőben van. Nem használják, mivel a RAID 4 magába foglalja, és ezzel kiváltja.

A 4. szintű RAID blokkszintű csíkozást használ, és paritásblokkot tárol az adatlemezek minden blokkjához. Problémát jelenthet azonban a paritáslemez meghibásodása, ekkor a hibafelismerés lehetősége elveszik ugyanis.

Ezt a 4. szintet magába foglaló 5. szintű RAID küszöböli ki, ahol a paritáslemez már nem torlódási pont, a paritásblokkok a lemezek között szétosztva kerülnek tárolásra.

A 6. szintű RAID két lemezhibát is eltűr, P+Q redundanciasémát használ, azaz minden 4 bit adathoz 2 bit redundáns adatot tárol. Így viszont sokkal drágábbak az írások.
Összegzésként elmondható, hogy a 2. és 3. szintet nem használják, mivel a RAID 4 magába foglalja őket, viszont jobb náluk. A leggyakrabban azonban az 5. szintű RAID-et használják, mivel a nála fejlettebb RAID 6 nagyon sokszor már felesleges. Az ideális szint azonban a kívánt alkalmazástól függ, a teljesítmény és a tárolás függvényében.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

3. Fizikai fájlszervezés, feladata, költségek, paraméterek, kupac, rendezett, hasító indexelt megoldások, módosítás, keresés, példákkal, előnyök, hátrányok.

Fizikai fájlszervezés, feladata:

A fő célok a gyors lekérdezés, gyors adatmódosítás, minél kisebb tárolási terület. Nincs azonban általánosan legjobb optimalizáció, az egyik cél a másik rovására javítható (például indexek használatával csökken a keresési idő, nő a tárméret és a módosítási idő).

Az adatbázis-alkalmazások alapján az adatbázis lehet:

- Statikus: ritkán módosul, a lekérdezések gyorsasága a fontosabb.
- Dinamikus: gyakran módosul, ritkán végzünk lekérdezést.

A memória műveletek nagyságrendekkel gyorsabbak, mint a háttértárolóról való beolvasás/kiírás. Az iró-olvasó fej nagyobb adategységeket, blokkokat olvas be, melyek mérete függhet az operációs rendszertől, hardvertől, adatbáziskezelőtől. A blokkméretet fixnek tekintjük (Oracle esetén 8kB az alapértelmezés). Feltételezzük, hogy a beolvasás, kiírás költsége arányos a háttértároló és a memória között mozgatott blokkok számával.

Célszerű a fájlokat blokkokba szervezni. A fájl rekordokból áll, melyek szerkezete eltérő is lehet. A rekord tartalmaz:

- Leíró fejlécet: rekordstruktúra leírása, belső/külső mutatók (hol kezdődik egy mező, melyek a kitöltetlen mezők, melyik a következő ill. előző rekord), törlési bit, statisztikák.
- Mezőket: melyek üresek, vagy adatot tartalmaznak.

A rekordhossz lehet állandó, vagy változó (változó hosszú mezők, ismétlődő mezők miatt). Az egyszerűség kedvéért feltesszük, hogy állandó hosszú rekordokból áll a fájl, melyek hossza az átlagos rekordméretnek felel meg.

A blokkok tartalmaznak:

- Leíró fejlécet: rekordok száma, struktúrája, fájlok leírása, belső/külső mutatók (hol kezdődik a rekord, hol vannak üres helyek, melyik a következő ill. az előző blokk), statisztikák (melyik fájlból hány rekord szerepel a blokkban).
- Rekordokat: egy vagy több fájlból.
- Üres helyeket.

Költségek, paraméterek:

A költségek méréséhez paramétereket vezetünk be:

- l: length, rekordméret (bájtokban).
- b: blokkméret (bájtokban).
- T: tuple, rekordok száma.
- bf: blokkolási faktor, azaz mennyire rekord fér el egy blokkban. Egyenlő b/l alsó egészrészével.
- B: a fájl mérete blokkokban. Egyenlő T/bf felső egészrészével.
- M: memória mérete blokkokban.

A relációs algebrai kiválasztás felbontható atomi kiválasztásokra, így elég ezek költségét vizsgálni. A legegyszerűbb kiválasztás az A=a, ahol "A" egy keresési mező, "a" pedig egy konstans. Kétféle bonyolultságot szokás vizsgálni, az átlagos és a legrosszabb esetet.

Az esetek vizsgálatánál az is számít, hogy az A=a feltételnek megfelelő rekordból lehet-e több, vagy biztos, hogy csak egy lehet. Fel szoktuk tenni, hogy az A=a feltételnek eleget tevő rekordokból nagyjából egyforma számú rekord szerepel, ez az egyenletességi feltétel.

Az A oszlopban szereplő különböző értékek számát képméretnek hívjuk, és I(A)-val jelöljük. A képméret egyenlő $|\prod_A(R)|$-el. Egyenletességi feltétel esetén $T(\sigma_{A=a}(R)) = T(R) / I(A)$ és $B(\sigma_{A=a}(R)) = B(R) / I(A)$.

Az az esetet vizsgáljuk, mikor az A=a feltételű rekordok közül elég az elsőt megkeresni. A módosítási műveletek a beszúrás (INSERT), frissítés (UPDATE), és a törlés (DELETE). Az egyszerűsített esetben nem foglalkozunk azzal, hogy a beolvasott rekordokat bent lehet tartani a memóriábann, későbbi keresések céljára.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

Kupac:

A kupac (heap) fájlok rendezetlen rekordokból állnak, ez a legegyszerűbb fájlszerkezet. A beszúrás hatékony, viszont a keresés és a törlés lassú. Egyenlőségi keresésnél átlagosan a lapok felét kell beolvasni, míg intervallum keresés esetén minden lapot be kell olvasni.

A rekordokat a blokk első üres helyére tesszük, a beérkezés sorrendjében. A tárméret kupac szervezés esetén B. Az A=a keresési idő legrosszabb esetben B, átlagos esetben egyenletességi feltétel esetén B/2. A beszúrás költsége 1 olvasás + 1 írás, a frissítés költsége 1 keresés + 1 írás, és a törlés költsége 1 keresés + 1 írás (üres hely marad, vagy a törlési bitet állítják át).

Rendezett:

A rendezett fájlokban a rekordok rendezetten helyezkednek el, a rendező mező szerint rendezve. Ha a rendező mező ugyanaz, mint a kulcs, rendező kulcsmezőnek nevezzük. Lassú beszúrás és törlés, viszont gyors logaritmikus keresés jellemzi.

A rendezettség miatt a blokkok láncolva vannak, és a következő blokkban nagyobb értékű rekordok szerepelnek, mint az előzőben. Ha a rendező mező és a kereső mező nem esik egybe, akkor kupac szervezést jelent. Ha azonban egybeesnek, akkor bináris (logaritmikus) keresést lehet alkalmazni:

- Beolvassuk a középső blokkot.
- Ha nincs benne az A=a értékű rekord, akkor eldöntjük, hogy a blokklánc második felében, vagy az első felében szerepelhet-e egyáltalán.
- Beolvassuk a felezett blokklánc középső blokkját.
- Addig folytatjuk, amíg megtaláljuk a rekordot, vagy a vizsgálandó maradék blokklánc már csak 1 blokkból áll.

Így a keresési idő $\log_2(b)$.

A beszúrás költsége a keresés költsége + az üres hely miatt a rekordok eltolásának költsége az összes blokkban, az adott találati blokktól kezdve (B/2 blokkot be kell olvasni, majd az eltolások után visszírni, így ez B művelet). Erre a szokásos megoldások a gyűjtő blokk használata, és az üres hely hagyása.

Gyűjtő (túlcsordulási) blokk használata:

- Az új rekordok számára nyitunk egy blokkot, ha betelik, hozzáláncolunk egy újabb blokkot.
- A keresést 2 helyen végezzük: $\log_2(B-G)$ költséggel keresünk a rendezett részben, és ha nem találjuk, akkor a gyűjtőben is megnézzük (G blokkművelet, ahol G a gyűjtő mérete), azaz az összköltség: $\log_2(B-G)+G$.
- Ha a G túl nagy a $\log_2(B)$-hez képest, akkor újrarendezzük a teljes fájlt (a rendezés költsége $B*\log_2(B)$).

Üres helyeket hagyunk a blokkokban:

- Például félig üresek a blokkok.
- A keresés után 1 blokkművelettel visszaírjuk a blokkot, amibe beírtuk az új rekordot.
- Tárméret 2*B lesz.
- Keresési idő: $\log_2(2*B) = 1+\log_2(B)$.
- Ha betelik egy blokk, vagy elér egy határt a telítettsége, akkor 2 blokkba osztjuk szét a rekordjait, a rendezettség fenntartásával.

A törlés költsége a keresés költsége + a törlés elvégzése/törlés bit beállítása után visszaírás (1 blokkírás). Túl sok törlés után újra kell szervezni a fájlt. A frissítés költsége a törlés költsége + beszúrás költsége.

Hasító indexelt:

A rekordok edényekbe (bucket) particionálva helyezkednek el, melyekre a h hasítómezőn értelmezett függvény osztja szét őket. Hatékony egyenlőségi keresés, beszúrás, és törlés jellemzi, viszont nem támogatja az intervallumos keresést.

A rekordokat blokkláncokba soroljuk, és a blokklánc utolsó blokkjának első üres helyére tesszük a rekordokat a beérkezés sorrendjében. A blokkláncok száma lehet előre adott (statikus hasítás, ekkor a számot K-val jelöljük), vagy a tárolt adatok alapján változhat (dinamikus hasítás). A besorolás az

indexmező értékei alapján történik. Egy $h(x) \in \{1,...,K\}$ hasító függvény értéke mondja meg, hogy melyik kosárba tartozik a rekord, ha x volt az indexmező értéke a rekordban. A hasító függvény általában maradékos osztáson alapul (például mod(K)). Akkor jó egy hasító függvény, ha nagyjából egyforma hosszú blokkláncok keletkeznek, azaz egyenletesen sorolja be a rekordokat. Ekkor a blokklánc B/K blokkból áll.

A keresés költsége:

- Ha az indexmező és keresési mező eltér, akkor kupac szervezést jelent.
- Ha az indexmező és keresési mező megegyezik, akkor csak elég a h(a) sorszámú kosarat végignézni, amely B/K blokkból álló kupacnak felel meg, azaz B/K legrosszabb esetben. A keresés így K-szorosára gyorsul.

A tárméret B, ha minden blokk nagyjából tele van. Nagy K esetén azonban sok olyan blokklánc lehet, amely egy blokkból fog állni, és a blokkban is csak 1 rekord lesz. Ekkor a keresési idő: 1 blokkbeolvasás, de B helyett T számú blokkban tároljuk az adatokat.

Módosításnál B/K blokkból álló kupac szervezésű kosarat kell módosítani.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

4. Fizikai fájlszervezés, feladata, költségek, paraméterek, elsődleges index, másodlagos index, bitmap index, módosítás, keresés, példákkal, előnyök, hátrányok.

Fizikai fájlszervezés, feladata:

A fő célok a gyors lekérdezés, gyors adatmódosítás, minél kisebb tárolási terület. Nincs azonban általánosan legjobb optimalizáció, az egyik cél a másik rovására javítható (például indexek használatával csökken a keresési idő, nő a tárméret és a módosítási idő).

Az adatbázis-alkalmazások alapján az adatbázis lehet:

- Statikus: ritkán módosul, a lekérdezések gyorsasága a fontosabb.
- Dinamikus: gyakran módosul, ritkán végzünk lekérdezést.

A memória műveletek nagyságrendekkel gyorsabbak, mint a háttértárolóról való beolvasás/kiírás. Az iró-olvasó fej nagyobb adategységeket, blokkokat olvas be, melyek mérete függhet az operációs rendszertől, hardvertől, adatbáziskezelőtől. A blokkméretet fixnek tekintjük (Oracle esetén 8kB az alapértelmezés). Feltételezzük, hogy a beolvasás, kiírás költsége arányos a háttértároló és a memória között mozgatott blokkok számával.

Célszerű a fájlokat blokkokba szervezni. A fájl rekordokból áll, melyek szerkezete eltérő is lehet. A rekord tartalmaz:

- Leíró fejlécet: rekordstruktúra leírása, belső/külső mutatók (hol kezdődik egy mező, melyek a kitöltetlen mezők, melyik a következő ill. előző rekord), törlési bit, statisztikák.
- Mezőket: melyek üresek, vagy adatot tartalmaznak.

A rekordhossz lehet állandó, vagy változó (változó hosszú mezők, ismétlődő mezők miatt). Az egyszerűség kedvéért feltesszük, hogy állandó hosszú rekordokból áll a fájl, melyek hossza az átlagos rekordméretnek felel meg.

A blokkok tartalmaznak:

- Leíró fejlécet: rekordok száma, struktúrája, fájlok leírása, belső/külső mutatók (hol kezdődik a rekord, hol vannak üres helyek, melyik a következő ill. az előző blokk), statisztikák (melyik fájlból hány rekord szerepel a blokkban).
- Rekordokat: egy vagy több fájlból.
- Üres helyeket.

Költségek, paraméterek:

A költségek méréséhez paramétereket vezetünk be:

- l: length, rekordméret (bájtokban).
- b: blokkméret (bájtokban).
- T: tuple, rekordok száma.
- bf: blokkolási faktor, azaz mennyire rekord fér el egy blokkban. Egyenlő b/l alsó egészrészével.
- B: a fájl mérete blokkokban. Egyenlő T/bf felső egészrészével.
- M: memória mérete blokkokban.

A relációs algebrai kiválasztás felbontható atomi kiválasztásokra, így elég ezek költségét vizsgálni. A legegyszerűbb kiválasztás az A=a, ahol "A" egy keresési mező, "a" pedig egy konstans. Kétféle bonyolultságot szokás vizsgálni, az átlagos és a legrosszabb esetet.

Az esetek vizsgálatánál az is számít, hogy az A=a feltételnek megfelelő rekordból lehet-e több, vagy biztos, hogy csak egy lehet. Fel szoktuk tenni, hogy az A=a feltételnek eleget tevő rekordokból nagyjából egyforma számú rekord szerepel, ez az egyenletességi feltétel.

Az A oszlopban szereplő különböző értékek számát képméretnek hívjuk, és I(A)-val jelöljük. A képméret egyenlő $|\prod_A(R)|$-el. Egyenletességi feltétel esetén $T(\sigma_{A=a}(R)) = T(R) / I(A)$ és $B(\sigma_{A=a}(R)) = B(R) / I(A)$.

Az az esetet vizsgáljuk, mikor az A=a feltételű rekordok közül elég az elsőt megkeresni. A módosítási műveletek a beszúrás (INSERT), frissítés (UPDATE), és a törlés (DELETE). Az egyszerűsített esetben nem foglalkozunk azzal, hogy a beolvasott rekordokat bent lehet tartani a memóriábann, későbbi keresések céljára.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

Az indexek keresést gyorsító segédstruktúrák. Több mezőre is lehet indexet készíteni. Nem csak a főfájlt, hanem az indexet is karban kell tartani, ami plusz költséget jelent. Ha a keresési mező egyik indexmezővel sem esik egybe, akkor kupac szervezést jelent. Az indexrekordok szerkezete (a,p), ahol "a" egy érték az indexelt oszlopban, "p" egy blokkmutató, arra a blokkra mutat, amelyben az A=a értékű rekordot tároljuk. Az index mindig rendezett az indexértékek szerint.

Elsődleges index:

Elsődleges index esetén a főfájl is rendezett (az indexmező szerint), így emiatt csak egy elsődleges indexet lehet megadni. Elég a főfájl minden blokkjának legkisebb rekordjához készíteni indexrekordot, így azok száma: T(I) = B (ritka index). Indexrekordból sokkal több fér egy blokkba, mint a főfájl rekordjaiból: bf(I) >> bf, azaz az indexfájl sokkal kisebb rendezett fájl, mint a főfájl: B(I) = B/bf(I) << B = T/bf.

Keresésnél, mivel az indexfájlban nem szerepel minden érték, ezért csak fedő értéket kereshetünk, a legnagyobb olyan indexértéket, amely a keresett értéknél kisebb vagy egyenlő. Fedő érték keresése az index rendezettsége miatt bináris kereséssel történik: $\log_2(B(I))$. A fedő indexrekordban szereplő blokkmutatónak megfelelő blokkot még be kell olvasni. Így a költség $1+\log_2(B(I)) << \log_2(B)$ (rendezett eset).

Módosításnál a rendezett fájlba kell beszúrni. Ha az első rekord változik a blokkban, akkor az indexfájlba is be kell szúrni, ami szintén rendezett. A megoldás az, hogy üres helyeket hagyunk a főfájl, és az indexfájl blokkjaiban is. Ezzel a tárméret duplázódhat, de a beszúrás legfeljebb egy főrekord, és egy indexrekord visszaírását jelenti.

Másodlagos index:

Másodlagos index esetén a főfájl rendezetlen (az indexfájl mindig rendezett). Másodlagos indexből többet is meg lehet adni. A főfájl minden rekordjához kell készíteni indexrekordot, így az indexrekordok száma: T(I) = T (sűrű index). Indexrekordból sokkal több fér egy blokkba, mint a főfájl rekordjaiból: bf(I) >> bf, azaz az indexfájl sokkal kisebb rendezett fájl, mint a főfájl: B(I) = T/bf(I) << B=T/bf.

Az indexben a keresés az index rendezettsége miatt bináris kereséssel történik: $\log_2(B(I))$. A talált indexrekordban szereplő blokkmutatónak megfelelő blokkot még be kell olvasni. Így a költség $1+\log_2(B(I)) << \log_2(B)$ (rendezett eset). Az elsődleges indexnél rosszabb a keresési idő, mert több az indexrekord.

A főfájl kupac szervezésű. Rendezett fájlba kell beszúrni. Ha az első rekord változik a blokkban, akkor az indexfájlba is be kell szúrni, ami szintén rendezett. Megoldás: üres helyeket hagyunk a főfájl, és az indexfájl blokkjaiban is. Ezzel a tárméret duplázódhat, de a beszúrás legfeljebb egy főrekord és egy indexrekord visszaírását jelenti.

Bitmap index:

A bitmap indexeket az oszlopok adott értékeihez szokták hozzárendelni, az alábbi módon:

- Ha az oszlopban az i.-ik sor értéke megegyezik az adott értékkel, akkor a bitmap index i.-ik tagja egy 1-es.
- Ha az oszlopban az i.-ik sor értéke viszont nem egyezik meg az adott értékkel, akkor a bitmap index i.-ik tagja egy 0.

Így egy lekérdezésnél csak megfelelően össze kell AND-elni, illetve OR-olni a bitmap indexeket, és az így kapott számsorozatban megkeresni, hol van egyes.

A bináris értékeket szokás szakaszhossz kódolással tömöríteni a hatékonyabb tárolás érdekében.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

5. Fizikai fájlszervezés, feladata, költségek, paraméterek, többszintű indexek, B-fa, B+-fa, B*-fa, módosítás, keresés, példákkal, előnyök, hátrányok.

Fizikai fájlszervezés, feladata:

A fő célok a gyors lekérdezés, gyors adatmódosítás, minél kisebb tárolási terület. Nincs azonban általánosan legjobb optimalizáció, az egyik cél a másik rovására javítható (például indexek használatával csökken a keresési idő, nő a tárméret és a módosítási idő).

Az adatbázis-alkalmazások alapján az adatbázis lehet:

- Statikus: ritkán módosul, a lekérdezések gyorsasága a fontosabb.
- Dinamikus: gyakran módosul, ritkán végzünk lekérdezést.

A memória műveletek nagyságrendekkel gyorsabbak, mint a háttértárolóról való beolvasás/kiírás. Az iró-olvasó fej nagyobb adategységeket, blokkokat olvas be, melyek mérete függhet az operációs rendszertől, hardvertől, adatbáziskezelőtől. A blokkméretet fixnek tekintjük (Oracle esetén 8kB az alapértelmezés). Feltételezzük, hogy a beolvasás, kiírás költsége arányos a háttértároló és a memória között mozgatott blokkok számával.

Célszerű a fájlokat blokkokba szervezni. A fájl rekordokból áll, melyek szerkezete eltérő is lehet. A rekord tartalmaz:

- Leíró fejlécet: rekordstruktúra leírása, belső/külső mutatók (hol kezdődik egy mező, melyek a kitöltetlen mezők, melyik a következő ill. előző rekord), törlési bit, statisztikák.
- Mezőket: melyek üresek, vagy adatot tartalmaznak.

A rekordhossz lehet állandó, vagy változó (változó hosszú mezők, ismétlődő mezők miatt). Az egyszerűség kedvéért feltesszük, hogy állandó hosszú rekordokból áll a fájl, melyek hossza az átlagos rekordméretnek felel meg.

A blokkok tartalmaznak:

- Leíró fejlécet: rekordok száma, struktúrája, fájlok leírása, belső/külső mutatók (hol kezdődik a rekord, hol vannak üres helyek, melyik a következő ill. az előző blokk), statisztikák (melyik fájlból hány rekord szerepel a blokkban).
- Rekordokat: egy vagy több fájlból.
- Üres helyeket.

Költségek, paraméterek:

A költségek méréséhez paramétereket vezetünk be:

- l: length, rekordméret (bájtokban).
- b: blokkméret (bájtokban).
- T: tuple, rekordok száma.
- bf: blokkolási faktor, azaz mennyire rekord fér el egy blokkban. Egyenlő b/l alsó egészrészével.
- B: a fájl mérete blokkokban. Egyenlő T/bf felső egészrészével.
- M: memória mérete blokkokban.

A relációs algebrai kiválasztás felbontható atomi kiválasztásokra, így elég ezek költségét vizsgálni. A legegyszerűbb kiválasztás az A=a, ahol "A" egy keresési mező, "a" pedig egy konstans. Kétféle bonyolultságot szokás vizsgálni, az átlagos és a legrosszabb esetet.

Az esetek vizsgálatánál az is számít, hogy az A=a feltételnek megfelelő rekordból lehet-e több, vagy biztos, hogy csak egy lehet. Fel szoktuk tenni, hogy az A=a feltételnek eleget tevő rekordokból nagyjából egyforma számú rekord szerepel, ez az egyenletességi feltétel.

Az A oszlopban szereplő különböző értékek számát képméretnek hívjuk, és I(A)-val jelöljük. A képméret egyenlő $|\Pi_A(R)|$-el. Egyenletességi feltétel esetén $T(\sigma_{A=a}(R)) = T(R) / I(A)$ és $B(\sigma_{A=a}(R)) = B(R) / I(A)$.

Az az esetet vizsgáljuk, mikor az A=a feltételű rekordok közül elég az elsőt megkeresni. A módosítási műveletek a beszúrás (INSERT), frissítés (UPDATE), és a törlés (DELETE). Az egyszerűsített esetben nem foglalkozunk azzal, hogy a beolvasott rekordokat bent lehet tartani a memóriábann, későbbi keresések céljára.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

Az indexek keresést gyorsító segédstruktúrák. Több mezőre is lehet indexet készíteni. Nem csak a főfájlt, hanem az indexet is karban kell tartani, ami plusz költséget jelent. Ha a keresési mező egyik indexmezővel sem esik egybe, akkor kupac szervezést jelent. Az indexrekordok szerkezete (a,p), ahol "a" egy érték az indexelt oszlopban, "p" egy blokkmutató, arra a blokkra mutat, amelyben az A=a értékű rekordot tároljuk. Az index mindig rendezett az indexértékek szerint.

Többszintű indexek:

Az indexfájl (1. indexszint) is fájl, ráadásul rendezett, így ezt is meg lehet indexelni, elsődleges indexszel. A főfájl lehet rendezett vagy rendezetlen (az indexfájl mindig rendezett). A t-szintű index: az indexszinteket is indexeljük, összesen t szintig.

A t-ik szinten (I(t)) bináris kereséssel keressük meg a fedő indexrekordot. Követjük a mutatót, minden szinten, és végül a főfájlban: $\log_2(B(I(t)))$ + t blokkolvasás. Ha a legfelső szint 1 blokkból áll, akkor t+1 blokkolvasást jelent. Minden szint blokkolási faktora megegyezik, mert egyforma hosszúak az indexrekordok.

	Főfájl	1. szint	2. szint	...	t. szint
Blokkok száma	B	B/bf(I)	$B/bf(I)^2$	...	$B/bf(I)^t$
Rekordok száma	T	B	B/bf(I)	...	$B/bf(I)^{(t-1)}$
Blokkolási faktor	bf	bf(I)	bf(I)	...	bf(I)

A t-ik szinten 1 blokk: $1=B/bf(I)^t$. Azaz $t=\log_{bf(I)}B < \log_2(B)$, tehát jobb a rendezett fájlszervezésnél. A $\log_{bf(I)}B < \log_2(B(I))$ is teljesül általában, így az egyszintű indexeknél is gyorsabb.

B-fa, B+-fa, B-fa:*

Logikailag az index egy rendezett lista. Fizikailag a rendezett sorrendet táblába rendezett mutatók biztosítják. A fa struktúrájú indexek B-fákkal ábrázolhatóak. A B-fák megoldják a bináris fák kiegyenlítetlenségi problémáját, mivel "alulról" töltjük fel őket. A B-fa egy csomópontjához több kulcsérték tartozhat. A mutatók más csomópontokra mutatnak, és így az összes kulcsértékre az adott csomóponton.

Mivel a B-fák kiegyenlítettek (minden ág egyenlő hosszú, vagyis ugyanazon a szinten fejeződik be), kiküszöbölik a változó elérési időket, amik a bináris fákban megfigyelhetőek. Bár a kulcsértékek és a hozzájuk kapcsolódó címek még mindig a fa minden szintjén megtalálhatók, és ennek eredménye: egyenlőtlen elérési utak, és egyenlőtlen elérési idő, valamint komplex fa-keresési algoritmus az adatfájl logikailag soros olvasására. Ez kiküszöbölhető, ha nem engedjük meg az adatfájl címek tárolását levélszint felett. Ebből következően: minden elérés ugyanolyan hosszú utat vesz igénybe, aminek egyenlő elérési idő az eredménye, és egy logikailag soros olvasása az adatfájlnak a levélszint elérésével megoldható. Nincs szükség komplex fa-keresési algoritmusra.

A B+-fa egy olyan B-fa, mely legalább 50%-ban telített. A szerkezeten kívül a telítettséget biztosító karbantartó algoritmusokat is beleértjük.

A B*-fa egy olyan B-fa, mely legalább 66%-ban telített.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

6. SQL lekérdezés átalakítása relációs algebrai kifejezéssé, lekérdezésfordító, algebrai optimalizálás, szabályok, heurisztikákon alapuló algoritmus, példákkal.

SQL lekérdezés átalakítása relációs algebrai kifejezéssé, lekérdezésfordító:

A lekérdezésfeldolgozó 3 fő lépésből áll:

- SQL lekérdezés elemzése, azaz átalakítása elemzőfává.
- Elemzőfa átalakítása, egy a relációs algebrával megfogalmazott kifejezésfává, amit logikai lekérdezéstervnek nevezünk.
- Logikai lekérdezésterv átalakítása fizikai lekérdezéstervvé, amely a végrehajtásra kerülő műveletek mellett mutatja, a végrehajtás sorrendjét, a használt algoritmusokat és az adatokat. Itt fel kell mérni az egyes választási lehetőségek várható költségét.

Az elemzőfa csomópontjai az alábbiak lehetnek:

- Atomok, melyek lexikai elemek. Pl. kulcsszavak, attribútumnevek, relációnevek, konstansok, zárójelek, operátorok.
- Szintaktikus kategóriák: ezek nevek, amelyek a lekérdezés olyan egységeit képezik, amelyek hasonló szerepet töltenek be a lekérdezésben. Pl <SWF> a megszokott SELECT-FROM-WHERE-nek felel meg.
- Ha a csomópont atom, akkor annak nincsenek gyerekei. Azonban ha a csomópont egy szintaktikus kategória, akkor annak gyerekeit a nyelvet megadó nyelvtan valamely szabálya írja le.

A nyelvtan szabályai:

- <Lekérdezés> ::= <SFW>
- <Lekérdezés> ::= (<Lekérdezés>)
- <SFW> ::= SELECT <SelLista> FROM <FromLista> WHERE <Feltétel>
- <SelLista> ::= <Attribútum> , <SelLista>
- <SelList> ::= <Attribútum>
- <FromLista> ::= <Reláció>, <FromLista>
- <FromLista> ::= <Reláció>
- <Feltétel> ::= <Feltétel> AND <Feltétel>
- <Feltétel> ::= <Sor> IN <Lekérdezés
- <Feltétel> ::= <Attribútum> = <Attribútum>
- <Feltétel> ::= <Attribútum> LIKE <Minta>
- <Sor> ::= <Attribútum>

Az <Attribútum>, a <Minta> és a <Reláció> speciális szintaktikus kategóriák. Az <Attribútum> egyetlen gyereke egy olyan karakterlánc lehet, amely egy attribútum neveként értelmezhető az AB sémában. A <Reláció> is olyan karakterlánc lehet, amely értelmes relációnevet jelent az adott sémában, a <Minta> egy „” közötti karaktersorozat, amely egy érvényes SQL-minta.

Az előfeldolgozó funkciói:

- szemantikus ellenőrzések,
- relációk használatának ellenőrzése,
- attribútumnevek ellenőrzése,
- típusellenőrzés.

A kommutatív szabály azt mondja ki, hogy nem számít az argumentumok sorrendje, az eredmény mindig ugyanaz lesz. Az asszociatív szabály azt mondja, hogy ha az operátort kétszer használjuk, akkor zárójelezhetjük balról és jobbról is. A relációs algebra kommutatív és asszociatív operátorai: descartes-szorzat, természetes összekapcsolás, unió, metszet.

Mivel kiválasztások lényegesen csökkenthetik a relációk méretét, ezért a legfontosabb szabály az optimalizáció során, hogy a kiválasztásokat addig vigyük lefelé a fában, amíg ez nem változtatja meg a kifejezés eredményét. Elsőként ha egy kiválasztás feltétele összetett, szétvágjuk az alkotóelemeire. Ezt szétvágási szabálynak nevezzük.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

Algebrai optimalizálás:

A relációs algebrai kifejezéseket minél gyorsabban akarjuk kiszámolni.
A kiszámítás költsége arányos a relációs algebrai kifejezés részkifejezéseinek megfelelő relációk tárolási méreteinek összegével.
A módszer az, hogy műveleti tulajdonságokon alapuló ekvivalens átalakításokat alkalmazunk, azért, hogy várhatóan kisebb méretű relációk keletkezzenek.
Az eljárás heurisztikus, tehát nem az argumentum relációk valódi méretével számol.
Az eredmény nem egyértelmű: Az átalakítások sorrendje nem determinisztikus, így más sorrendben végrehajtva az átalakításokat más végeredményt kaphatunk, de mindegyik általában jobb költségű, mint amiből kiindultunk.

Szabályok:

Algebrai optimalizáció szabályai:

- Kommutatív operátorok: természetes összekapcsolás, descartes-szorzat, unió, metszet.
- Asszociatív szabályok: természetes összekapcsolás, descartes-szorzat, unió, metszet.
- Legyen $\underline{A}$ és $\underline{B}$ két részhalmaza az E reláció oszlopainak úgy, hogy $\underline{A} \subseteq \underline{B}$. Ekkor $\Pi_A(\Pi_B(E)) \cong \Pi_A(E)$.
- Kiválasztások felcserélhetősége, felbontása: Legyen F1 és F2 az E reláció oszlopain értelmezett kiválasztási feltétel. Ekkor $\sigma_{F1 \wedge F2}(E) \cong \sigma_{F1}(\sigma_{F2}(E)) \cong \sigma_{F2}(\sigma_{F1}(E))$.
- Legyen F az E relációnak csak az A oszlopain értelmezett kiválasztási feltétel. Ekkor $\Pi_{\underline{A}}(\sigma_F(E)) \cong \sigma_F(\Pi_A(E))$.
- Kiválasztás és szorzás felcserélhetősége: Legyen F az E1 reláció oszlopainak egy részhalmazán értelmezett kiválasztási feltétel. Ekkor $\sigma_F(E1 \times E2) \cong \sigma_F(E1) \times E2$.
- Kiválasztás és egyesítés felcserélhetősége: Legyen E1, E2 relációk sémája megegyező, és F a közös sémán értelmezett kiválasztási feltétel. Ekkor $\sigma_F(E1 \cup E2) \cong \sigma_F(E1) \cup \sigma_F(E2)$.
- Kiválasztás és a kivonás felcserélhetősége: Legyen E1, E2 relációk sémája megegyező, és F a közös sémán értelmezett kiválasztási feltétel. Ekkor $\sigma_F(E1 - E2) \cong \sigma_F(E1) - \sigma_F(E2)$.
- Kiválasztás és a természetes összekapcsolás felcserélhetősége: Legyen F az E1 és E2 közös oszlopainak egy részhalmazán értelmezett kiválasztási feltétel. Ekkor $\sigma_F(E1 |\times| E2) \cong \sigma_F(E1) |\times| \sigma_F(E2)$.
- Vetítés és a szorzás felcserélhetősége: Legyen i=1,2 esetén $\underline{Ai}$ az Ei reláció oszlopainak egy halmaza, valamint legyen $\underline{A} = \underline{A1} \cup \underline{A2}$. Ekkor $\Pi_{\underline{A}}(E1 \times E2) \cong \Pi_{\underline{A1}}(E1) \times \Pi_{\underline{A2}}(E2)$.
- Vetítés és egyesítés felcserélhetősége: Legyen E1 és E2 relációk sémája megegyező, és legyen $\underline{A}$ a sémában szereplő oszlopok egy részhalmaza. Ekkor $\Pi_{\underline{A}}(E1 \cup E2) \cong \Pi_{\underline{A}}(E1) \cup \Pi_{\underline{A}}(E2)$.

Heurisztikákon alapuló algoritmus:

Az optimalizáló algoritmus a következő heurisztikus elveken alapul:

- Minél hamarabb szelektáljunk, hogy a részkifejezések várhatóan kisebb relációk legyenek.
- A szorzás utáni kiválasztásokból próbáljunk természetes összekapcsolásokat képezni, mert az összekapcsolás hatékonyabban kiszámolható, mint a szorzatból történő kiválasztás.
- Vonjuk össze az egymás utáni unáris műveleteket (kiválasztásokat és vetítéseket), és ezekből lehetőleg egy kiválasztást, vagy vetítést, vagy kiválasztás utáni vetítést képezzünk. Így csökken a műveletek száma, és általában a kiválasztás kisebb relációt eredményez, mint a vetítés.
- Keressünk közös részkifejezéseket, amiket így elég csak egyszer kiszámolni a kifejezés kiértékelése során.

Algebrai optimalizációs algoritmus:

- INPUT: relációs algebrai kifejezés kifejezésfája.
- OUTPUT: optimalizált kifejezésfa optimalizált kiértékelése.

Hajtsuk végre az alábbi lépéseket a megadott sorrendben:

- A kiválasztásokat bontsuk fel a 4. szabály segítségével: $\sigma_{F1 \wedge \ldots \wedge Fn}(E) \cong \sigma_{F1}(\ldots(\sigma_{Fn}(E)))$
- A kiválasztásokat a 4., 5., 6., 7., 8., 9. szabályok segítségével vigyük olyan mélyre a kifejezésfában, amilyen mélyre csak lehet.

- A vetítéseket a 3., 5., 10., 11. szabályok segítségével vigyük olyan mélyre a kifejezésfában, amilyen mélyre csak lehet. Hagyjuk el a triviális vetítéseket, azaz az olyanokat, amelyek az argumentum reláció összes attribútumára vetítenek.
- Ha egy relációs változóra vagy konstans relációra közvetlenül egymás után kiválasztásokat vagy vetítéseket alkalmazunk, akkor ezeket a 3., 4., 5. szabályok segítségével vonjuk össze egy kiválasztássá, vagy egy vetítéssé, vagy egy kiválasztás utáni vetítéssé, ha lehet (azaz egy Π.(σ.()) alakú kifejezéssé). Ezzel megkaptuk az optimalizált kifejezésfát.
- A gráfot a bináris műveletek alapján bontsuk részgráfokra. Minden részgráf egy bináris műveletnek feleljen meg. A részgráf csúcsai legyenek: a bináris műveletnek (∪, ⎯, ×) megfelelő csúcs és a csúcs felett a következő bináris műveletig szereplő kiválasztások (σ) és vetítések (Π). Ha a bináris művelet szorzás (×), és a részgráf equi-joinnak felel meg, és a szorzás valamelyik ága nem tartalmaz bináris műveletet, akkor ezt az ágat is vegyük hozzá a részgráfhoz.
- Az előző lépésben kapott részgráfok is fát képeznek. Az optimális kiértékeléshez ezt a fát értékeljük ki alulról felfelé haladva, tetszőleges sorrendben.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

7. A relációs algebrai műveletek megvalósítása, egy és többmenetes algoritmusok, műveleti költségek, outputméretek becslése.

Megvalósítások, és költségek:

A műveletek a kiválasztás (σ), a vetítés (π), az unió (U), a különbség (-), a szorzat (x), és az összekapcsolás (⋈).

Költségek:

- N_R: R rekordjainak száma.
- L_R: R egy rekordjának mérete.
- F_R: blokkolási tényező (egy lapon levő rekordok száma).
- B_R: az R reláció tárolásához szükséges lapok száma.
- V(A,R): az A mező különböző értékeinek száma R-ben (Képméret).
- SC(A,R): az A mező kiválasztási számossága R-ben (Szelektivitás).
  - A kulcs: S(A,R)=1.
  - A nem kulcs: $S(A,R)= N_R / V(A,R)$.
- HT_i: az i index szintjeinek száma.
- A törteket és logaritmusokat felfelé kerekítjük.

Kiválasztás esetén:

- Lineáris keresés: olvassunk be minden lapot és keressük az egyezéseket (egyenlőség vizsgálata esetén). Átlagos költség: nem kulcs B_R, kulcs $0.5*B_R$.
- Logaritmikus keresés: rendezett mező esetén $\lceil \log_2 B_R \rceil + m$. Az átlagos költség: m további oldalt kell beolvasni, és $m = \lceil SC(A,R)/F_R \rceil - 1$.
- Elsődleges/cluster index: átlagos költség egyetlen rekord esetén $HT_i + 1$, több rekord esetén $HT_i + \lceil SC(A,R)/F_R \rceil$.
- Másodlagos index: az átlagos költség kulcs mezőnél $HT_i + 1$, nem kulcs mezőnél legrosszabb eset $HT_i + SC(A,R)$, de a lineáris keresés kedvezőbb, ha sok a megfelelő rekord.

Összetett kiválasztás esetén:

- Konjunkciós kiválasztás: $\sigma\theta_1/\backslash\theta_2.../\backslash\theta_n$.
  - Végezzünk egyszerű kiválasztást a legkisebb költségű θi-re pl. a θi-hez tartozó index felhasználásával, a fennmaradó θ feltételek szerint szűrjük az eredményt. Költség: az egyszerű kiválasztás költsége a kiválasztott θ-ra.
  - Több index esetén válasszuk ki a θi-khez tartozó indexeket, és keressünk az indexekben és adjuk vissza a RID-ket. Válasz: RID-k metszete. Költség: a költségek összege + rekordok beolvasása.
- Diszjunkciós kiválasztás: σθ1\/θ2...\/θn.
  - Több index esetén a RID-k uniója.
  - Lineáris keresés.

Vetítés és halmazműveletek esetén:

- A halmazműveletekhez ki kell szűrni a duplikált értékeket.
- R ∩ S
- R U S
- Rendezés: sok művelet hatékony kiértékelése. A lekérdezés igényelheti. Megvalósítása lehet belső rendezés (ha a rekordok befférnek a memóriába), vagy külső rendezés.

Vetítés esetén:

- $\pi_{A1,A2...}(R)$
- Felesleges mezők törlése: átnézés és mezők eldobása.
- Duplikált rekordok törlése: az eredmény rekordok rendezése az összes mező szerint. A rendezett eredmény átnézése, duplikáltak (szomszédos) törlése.
- Költség: kezdeti átnézés + rendezés + végső átnézés.

Összekapcsolás esetén:

- Skatulyázott ciklusos (nested loop) összekapcsolás:

- Bármilyen összekapcsolási feltételnél működik.
- S belső reláció, és R külső reláció.
- Költség: legjobb eset, ha a kisebb reláció elfér a memóriában, ezt használjuk belső relációnak, ekkor: B_R+B_S. Legrosszabb eset, ha mindkét relációból csak 1-1 lap fér bele a memóriába, ekkor S-t minden R-beli rekordnál végig kell olvasni, így $N_R * B_S + B_R$.

- Blokk-skatulyázott ciklusos (block-nested loop) összekapcsolás:
  - Költség: legjobb eset, ha a kisebb reláció elfér a memóriában, ezt használjuk belső relációnak, ekkor: B_R+B_S. Legrosszabb eset, ha mindkét relációból csak 1-1 lap fér bele a memóriába, ekkor S-t minden R-beli lapnál végig kell olvasni, így $B_R * B_S + B_R$.
- Indexelt skatulyázott ciklusos összekapcsolás:
  - $R \bowtie S$
  - Index a belső relación (S).
  - A külső reláció (R) minden rekordjánál keresünk a belső reláció indexében.
  - Költség: $B_R + N_R * c$, ahol c a belső relációból index szerinti kiválasztás költsége. A kevesebb rekordot tartalmazó reláció legyen a külső.
- Összefésüléses rendező összekapcsolás:
  - $R \bowtie S$
  - A relációk rendezettek az összekapcsolási mezők szerint.
  - Egyesítjük a rendezett relációkat: mutatók az első rekordra mindkét relációban. Beolvasunk S-ből egy rekordcsoportot, ahol az összekapcsolási attribútum értéke megegyezik. Beolvasunk rekordokat R-ből és feldolgozzuk.
  - A rendezett relációkat csak egyszer kell végigolvasni.
  - Költség: rendezés költsége + $B_S + B_R$.
- Hasításos összekapcsolás:
  - $R \bowtie S$
  - Alkalmazzuk h1-et az összekapcsolási mezőre és felosztjuk a rekordokat a memóriában elférő részekre. R rekordjainak felosztása $R_0 \ldots R_{n-1}$. S rekordjainak felosztása $S_0 \ldots S_{n-1}$.
  - Az egymáshoz illő partíciók rekordjait összekapcsoljuk, hasítófüggvény alapján indexelt blokk-skatulyázott ciklusos összekapcsolással.
  - Költség: $2*(B_R + B_S) + (B_R + B_S)$.

Költségbecslés:

Fontos a művelet, a megvalósítás, a bemenet mérete, a kimenet mérete, és a rendezés.

A költségoptimalizáló átalakítja a kifejezéseket egyenértékű kifejezésekké szabályok segítségével. Korán kell elvégezni a kiválasztást, és a vetítést. A szorzatot követő kiválasztást helyettesítsük összekapcsolással. A legkisebb eredményt adó összekapcsolásokkal és kiválasztásokkal kezdjük, és készítsünk bal oldalon mély kiválasztási fákat.

Méretbecslés:

$\sigma_{A=v}(R)$ esetén SC(A,R).

$\sigma_{A<=v}(R)$ esetén $N_R * (v\text{-}min(A,R)) / (max(A,R)\text{-}min(A,R))$

$\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(R)$ esetén szorzódnak a valószínűségek, így $N_R * [(s_1/N_R)*(s_2/N_R)*\ldots*(s_n/N_R)]$

$\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(R)$ esetén annak valószínűsége, hogy egy rekordra egy θ se igaz: $[(1-s_1/N_R)*(1-s_2/N_R)*\ldots*(1-s_n/N_R)]$, így $N_R*(1-[(1-s_1/N_R)*(1-s_2/N_R)*\ldots*(1-s_n/N_R)])$.

R x S esetén $N_R * N_S$.

$R \bowtie S$ esetén, ha $R \cap S = \emptyset$: $N_R * N_S$. Ha $R \cap S$ kulcs R-en: a kimenet maximális mérete N_S. Ha $R \cap S$ idegen kulcs R-hez: N_S. Ha $R \cap S = \{A\}$, sem R-nek, sem S-nek nem kulcsa: $N_R*N_S / V(A,S)$, $N_S*N_R / V(A,R)$.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

8. Több tábla összekapcsolása, összekapcsolások sorrendje, futószalagosítás, materializáció, dinamikus programozási feladat, a félig-összekapcsolás (semi-join) és alkalmazása osztott lekérdezésekre.

Több tábla összekapcsolása:

Egy bal-mély (jobb-mély) fa egy olyan összekapcsolási fa, amelyben a jobb oldal (bal oldal) egy reláció, nem pedig egy köztes összekapcsolás eredménye.
A bokros fa se nem bal-, se nem jobb-mély.
Általában elegendő a bal-mély fákat vizsgálni:

- Kisebb keresési tér.
- Általában hatékony bizonyos összekapcsolási algoritmusokhoz (rendezzük a relációkat a kisebbtől a nagyobbak felé).
- Megengedi a csővezeték használatát.
- Elkerüli a köztes eredmények megjelenését (materializálás!) a lemezen.

Összekapcsolások sorrendje:

Az összekapcsolások kommutatívak és asszociatívak, azaz $(A \bowtie B) \bowtie C = A \bowtie (B \bowtie C)$.

Válasszuk azt a sorrendet, amely minimalizálja a köztes eredmények méreteinek összegét:

- Valószínűleg I/O és számítási hatékony.
- Ha az $A \bowtie B$ eredménye kisebb a $B \bowtie C$-énél, akkor válasszuk az $(A \bowtie B) \bowtie C$ -t.

Alternatív feltételek: lemez hozzáférések, CPU, válaszidő, hálózat.
Válasszuk ki az összekapcsolási fa alakját:

- Ekvivalens az n-utas összekapcsolások lehetséges zárójelezéseinek számával.
- Ismétlődés: $T(1) = 1$, $T(n) = \Sigma\, T(i)T(n-i)$, $T(6)=42$

A levelek permutációi (n!), azaz például n=6 esetén az összekapcsolási fák száma 42*6! (vagyis 42*720 = 30240).

Futószalagosítás, materializáció:

Materializáció az, amikor egy művelet eredményét átmeneti relációként tároljuk, amelyet a következő feldolgozhat.
Csővezeték vagy menet közbeni feldolgozás:

- Az egyik művelet eredményét átadjuk egy másiknak átmeneti reláció létrehozása nélkül.
- Megtakarítja az átmeneti reláció kiírásának és visszaolvasásának költségét.
- Általában a csővezetéket külön folyamat vagy szál valósítja meg.

A külső reláció minden rekordjához az egész belső relációt át kell nézni. A belső relációt nem lehet csővezetékbe tenni, hanem mindig materializálni kell. Ezért a bal-magas fák kívánatosak, mivel a belső relációk mindig alaprelációk. Csökkenti az optimális stratégia keresési terét, és lehetőséget ad a QO-nak a dinamikus feldolgozásra.

Dinamikus programozás:

Ésszerűtlen végignézni az összes lehetséges összekapcsolási sorrendet (n!). Használjuk a legjobb tervet a (k-1)-utas összekapcsolásból a legjobb k-utas összekapcsolás kiszámításához. Mohó heurisztikus algoritmus.
A legjobb összekapcsolási sorrend vagy algoritmus kiválasztása a relációk minden egyes részhalmazához nem feltétlen a legjobb döntés. A rendezéses összekapcsoló eljárások rendezhettek kimenetet ami hasznos lehet (pl. ORDER BY/GROUP BY, rendezés az összekapcsolás paraméterén).
Csővezeték használata skatulyázott ciklusos összekapcsolással (pipelining with nested-loop join).
1970-es évek, alapozó munka az összekapcsolási sorrend optimalizáláshoz System R-en (az IBM kutatási célú adatbázis rendszere).
Az érdekes rendezési sorrend felismerése. Minden egyes rendezésnél keressük meg a legjobb tervet a relációk minden egyes részhalmazához (érdekes tulajdonságonként egy terv). Kevés érdekes sorrend esetén a költség alacsony.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

A legjobb összekapcsolási fa megtalálása n reláció egy halmazához:

- Hogy megtaláljuk a legjobb összekapcsolási fát n reláció egy S halmazához, vegyük az összes lehetséges tervet mely így néz ki: S1⋈(S-S1), ahol S1 az S tetszőleges nem üres részhalmaza.
- Rekurzívan számítsuk ki S részhalmazainak összekapcsolásának költségeit, hogy meghatározzuk minden egyes terv költségét. Válasszuk a legolcsóbbat a 2_n-1 lehetséges közül.
- Mikor bármely részhalmaz terve kiszámításra került, az újbóli kiszámítás helyett tároljuk el és hasznosítsuk újra amikor ismét szükség lesz rá.

Semi-join:

Egy semijoin az A attribútumon R_i-ből R_j-be így írható le: R_j |x R_i. Arra használják, hogy csökkentse az adatátviteli költségeket.

Számítási lépések:

- Vetítsük R_i-t az A attribútumra és továbbítsuk ezt a projekciót (egy semi-join projekciót) az R_i feldolgozási helyéről az R_j feldolgozási helyére.
- Redukáljuk R_j-t R_j'-re az olyan rekordok törlésével, ahol az A attribútum nem egyezik meg egyetlen R_i[A] -beli értékkel sem.

Alkalmazása:

Az osztott lekérdezés-feldolgozás adatok begyűjtése a különböző hálózati helyekről.

Fázisai egy semi-join operátorral:

- Kezdeti helyi feldolgozás: minden kiválasztást, és projekciót helyben elvégzünk.
- Semi-join feldolgozás: a fennmaradó join műveletekből származtatunk egy semijoin programot és lefuttatjuk, hogy költséghatékony módon csökkentse a relációk méretét.
- Végső feldolgozás:minden érintett reláció a végső helyre továbbítódik és minden összekapcsolás ott kerül elvégzésre.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

9. A Q(A,B) JOIN R(B,C) JOIN S(C,D) háromféle kiszámítási módja és költsége (feltéve, hogy Q, R, S paraméterei megegyeznek, Q.B-re és S.C-re klaszterindexünk van).

Feltevések:

- $|Q| = |R| = |S| = T_0$ sor.
- Tömör elhelyezés esetén B_0 blokk.
- $I_0 <= B_0$ (I_0 minden oszlop képmérete egyenlő)
- Klaszterindexünk van Q.B-re és S.C-re**.**

a, Balról jobbra.

$Q \bowtie R$ költsége:

- $(B_0 + T_0B_0/I_0) + (B_0T_0+T_0B_0)/I_0 = B_0 + 3B_0T_0/I_0$

$(Q \bowtie R) \bowtie S$ költsége:

- $B_{QxR}(1+T_S/I_0)+(2T_{Q \bowtie R}*B_S)/I_0$
- $2B_0T_0/I_0 + 2B_0T_0^2/I_0^2 + (2* [T_0^2/I_0] * B_0)/I_0 = 2B_0T_0/I_0 + 4B_0T_0^2/I_0^2$

Összköltség:

- $(B_0 + 3B_0T_0/I_0) + 2B_0T_0/I_0 + 4B_0T_0^2/I_0^2 = B_0 + 5B_0T_0/I_0 + 4B_0T_0^2/I_0^2$

b, Balról jobbra és a memóriában összekapcsolva a harmadik táblával.

$Q \bowtie R$ -t közben feleslegesen kiírjuk, majd újra beolvassuk. Rögtön a memóriában összekapcsoljuk az abc -t $\sigma_{C=c}(S)$ -el, így a valódi költség:

- $B_0 + B_0T_0/I_0 + 4B_0T_0^2/I_0^2$

c, A középső ténytábla soraihoz kapcsolva a szélső dimenziótáblákat.

Ciklus: *for* az R minden bc sorára *do* $\sigma_{B=b}(Q) \bowtie \{bc\} \bowtie \sigma_{C=c}(S)$.

Feltesszük, hogy a fenti összekapcsolásokhoz szükséges sorok befférnek a memóriába: $2B_0/I_0 <= M$.

Költség:

- Ciklusmagot T_0-szor hajtjuk végre $\sigma_{B=b}(Q) \bowtie \{bc\} \bowtie \sigma_{C=c}(S)$ beolvasása: $T_0*[B_0/I_0] + B_0 + T_0B_0/I_0 = B_0 + 2B_0T_0/I_0$ az összes beolvasási költség

Output költség:

- $([2B_0T_0/I_0]*T_0 + [T_0^2/I_0]*B_0) / I_0 = 3B_0T_0^2/I_0^2$

Összköltség:

- $B_0 + 2B_0T_0/I_0 + 3B_0T_0^2/I_0^2$

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

10. Az Oracle költségalapú és szabályalapú optimalizálása, lekérdezésterveinek megjelenítése, értelmezése, Explain plan, tkprof, hintek, példák.

Költségalapú:

A költségalapú optimalizáló azt a stratégiát választja, amely minimális erőforrás felhasználásával előállítja a lekérdezés által elért sorokat.
A felhasználó beállíthatja, hogy a minimális erőforrás-használat az átvitel (minden sor előállítása) vagy a válaszidő (első sor előállítása) alapján legyen megállapítva.
A felhasználó tippeket adhat bizonyos döntésekhez, mint az elérési út vagy az összekapcsolási művelet.
Lekérdezhető a végrehajtási terv.

Szabályalapú:

15 szabály, hatékonyság szerint rendezve. Egy elérési út csak akkor kerül kiválasztásra, ha az állítás tartalmaz egy predikátumot vagy szerkezetet, amely elérhetővé teszi ezt az útvonalat.
Pontszámot rendel minden végrehajtási stratégiához a rangsor alapján, majd a legjobb (legkisebb) pontszámút választja.

Rang	Elérési út
1	Egy sor sorazonosító (ROWID) alapján
2	Egy sor cluster összekapcsolással
3	Egy sor hasított cluster kulccsal egyedi vagy elsődleges kulccsal
4	Egy sor egyedi vagy elsődleges kulccsal
5	Cluster összekapcsolás
6	Hasított cluster kulcs
7	Indexelt cluster kulcs
8	Összetett kulcs
9	Egy mezős indexek
10	Korlátos intervallumos keresés indexelt mezőn
11	Nem korlátos intervallumos keresés indexelt mezőn
12	Összefésüléses rendező összekapcsolás
13	MAX vagy MIN indexelt mezőn
14	ORDER BY indexelt mezőn
15	Teljes táblaolvasás

Lekérdezéstervek:

Egy végrehajtási terv olyan lépések listája, amelyeket az Oracle fog követni az SQL kifejezés végrehajtása érdekében. Minden lépés egyike az adatbázis szerver által ismert véges számú alap utasításoknak. Még a legösszetettebb SQL kifejezés is felbontható alap műveletek sorozatára.
A végrehajtási terveket lekérdezés után a tervtábla tartalmazza (az általános elnevezése plan_table, de akármilyen más elnevezés is használható), melynek fontos oszlopai:

- statement_id: Egyedi azonosító minden egyes végrehajtási tervhez.
- timestamp: Mikor jött létre a végrehajtási terv.
- operation: A végrehajtási terv egy lépésében végrehajtott műlevet, mint pl. ”table acces”.
- options: További információk a műveletről, mint pl. ”by index ROWID”.
- object_name: A hozzáfért tábla, index, nézet, stb. neve.
- optimizer: Az optimalizációs cél, amit a terv létrehozáskor használtunk.
- id: A végrehajtási terv lépésének azonosítója.
- parent_id: A szülő lépés azonosítója.

A tervek megtekintésénél nem feltétlenül jelenik meg mindegyik oszlop, megadható, hogy milyen részletesen szeretnénk látni a végrehajtási tervet.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

Explain plan:

Az Explain plan egy olyan kifejezés, amely lehetővé teszi számodra, hogy az Oracle bármely SQL kifejezéshez elkészítse a végrehajtási tervet annak tényleges végrehajtása nélkül. Így a végrehajtási terv megvizsgálható a tervtábla lekérdezésével.

Előfeltételei:

- INSERT jogosultság egy tervtáblára.
- Az elemzendő kifejezés végrehajtásához szükséges összes jogosultság.
- SELECT jogosultságok a használt nézettáblákra, ha a vizsgált kifejezés használt nézeteket.

Korlátai:

- Az igazi terv ami felhasználásra kerül, eltérhet attól amit az Explain plan mond, több okból:
  - Optimalizálói statisztikák, kurzor megosztás, kötött változó előreolvasás, dinamikus példány paraméterek csökkentik a terv stabilitását.
  - Az Explain plan nem nézi meg előre a kötött változókat.
  - Az Explain plan nem ellenőrzi a könyvtár cache-t, hogy a kifejezést feldolgozták-e már.
- Az Explain plan nem működik egyes lekérdezésekhez ("ORA-22905: cannot access rows for a non-nested table item").

Tkprof:

Egy adatbázis sessiont kezelő Oracle szerver folyamat részletes trace file-t készít, amikor az SQL nyomkövetés engedélyezve van a sessionhoz.

A Tkprof egy Oracle segédeszköz, ami nagyon hasznos és olvasható jelentésekké formázza az SQL trace fájlokat. A Tkprof-ot az operációs parancssorából lehet meghívni, nincs hozzá grafikus felület. Az Oracle 9i –től kezdve a Tkprof tudja olvasni a kiterjesztett SQL trace fájlokat és jelentést készít a várakozási esemény statisztikákról (a Tkprof egyik paramétere a kimenet generáláshoz a waits=yes).

Egy Tkprof jelentés elemei:

- Riport fejléc:
  - Tkprof verzió, futtatás dátuma, rendezési opció, trace fájl.
- Egy-egy bejegyzés minden különböző SQL kifejezéshez a trace fájlban:
  - SQL kifejezés listázása.
  - OCI hívási statisztikák: elemzések (parse) száma, execute és fetch hívások, feldolgozott sorok, felhasznált idő és I/O.
  - Elemzési információ: elemző felhasználó, rekurzív mélység, könyvtár cache tévesztések, optimalizáló mód.
  - Sor forrás műveletek listázása.
  - Végrehajtási terv listázása (opcionális).
  - Várakozási események listája (opcionális).
- Jelentés összefoglalása:
  - OCI hívási statisztikák összegzései.
  - A trace fájlban talált lekérdezések száma, hány volt ezekből különböző (egyedi), és hány volt megmagyarázva a jelentésben.

Az egyes lépéseknél sorok száma nem becslés, hanem a tényleges, pontos adat. Ez hasznos lehet amikor gyengén teljesítő lekérdezéseket akarunk javítani.

Hintek:

A hintek segítségével egy-egy lekérdezés során "tippeket" adhatunk a fordítónak, hogy mit használjon, vagy mit ne használjon a lekérdezés végrehajtásakor. Például:

- Tábla elérések (pl. FULL).
- Index-ek használata, fajtájuk (pl. INDEX).
- Összekapcsolások (pl. INDEX_JOIN).

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

11. Rendszerhibák kezelése, konzisztens adatbázis, tranzakciók, hibafajták, semmisségi (undo) naplózás és helyreállítás, ellenőrzőpont, ellenőrzőpont működés közben, példák.

Konzisztens adatbázis:

Konzisztens állapot: kielégíti az összes feltételt (megszorítást).
Konzisztens adatbázisnak nevezünk egy konzisztens állapotú adatbázist.
A konzisztencia sérülhet:

- Tranzakcióhiba: hibásan megírt, rosszul ütemezett, félbehagyott tranzakciók.
- Adatbázis-kezelési hiba: az adatbázis-kezelő valamelyik komponense nem, vagy rosszul hajtja végre a feladatát.
- Hardverhiba: elvész egy adat, vagy megváltozik az értéke.
- Adatmegosztásból származó hiba.

Tranzakciók:

Tranzakció a konzisztenciát megtartó adatkezelő műveletek sorozata. Ezek után mindig feltesszük:
Ha T tranzakció konzisztens állapotból indul + T tranzakció csak egyedül futna le → T konzisztens állapotban hagyja az adatbázis.
Helyesség feltétele:

- Ha leáll egy vagy több tranzakció (abort, vagy hiba miatt), akkor is konzisztens adatbázist kapunk.
- Mind egyes tranzakció induláskor konzisztens adatbázist lát.

A tranzakció az adatbázis-műveletek végrehajtási egysége, amely DML-beli utasításokból áll, és a következő tulajdonságokkal rendelkezik:

- Atomosság (A): a tranzakció „mindent vagy semmit" jellegű végrehajtása (vagy teljesen végrehajtjuk, vagy egyáltalán nem hajtjuk végre).
- Konzisztencia (C): az a feltétel, hogy a tranzakció megőrizze az adatbázis konzisztenciáját, azaz a tranzakció végrehajtása után is teljesüljenek az adatbázisban előírt konzisztenciamegszorítások.
- Elkülönítés (I): az a tény, hogy minden tranzakciónak látszólag úgy kell lefutnia, mintha ez alatt az idő alatt semmilyen másik tranzakciót sem hajtanánk végre.
- Tartósság (D): az a feltétel, hogy ha egyszer egy tranzakció befejeződött, akkor már soha többé nem veszhet el a tranzakciónak az adatbázison kifejtett hatása.

A konzisztenciát mindig adottnak tekintjük. A másik három tulajdonságot viszont az adatbázis-kezelő rendszernek kell biztosítania, de ettől időnként eltekintünk.
Feltesszük, hogy az adatbázis adategységekből, elemekből áll. Az adatbáziselem a fizikai adatbázisban tárolt adatok egyfajta funkcionális egysége, amelynek értékét tranzakciókkal lehet elérni (kiolvasni) vagy módosítani (kiírni). Az adatbáziselem lehet:

- reláció (vagy OO megfelelőjét, az osztálykiterjedés),
- relációsor (vagy OO megfelelője, az objektum),
- lemezblokk,
- lap.

A tranzakció és az adatbázis kölcsönhatásának három fontos helyszíne van:

- Az adatbázis elemeit tartalmazó lemezblokkok területe. (D)
- A pufferkezelő által használt virtuális vagy valós memóriaterület. (M)
- A tranzakció memóriaterülete. (M)

OLVASÁS: Ahhoz, hogy a tranzakció egy X adatbáziselemet beolvashasson, azt előbb memóriapuffer(ek)be (P) kell behozni, ha még nincs ott. Ezt követően tudja a puffer(ek) tartalmát a tranzakció a saját memóriaterületére (t) beolvasni.
ÍRÁS: Az adatbáziselem új értékének kiírása fordított sorrendben történik: az új értéket a tranzakció alakítja ki a saját memóriaterületén, majd ez az új érték másolódik át a megfelelő puffer(ek)be.
Fontos, hogy egy tranzakció sohasem módosíthatja egy adatbáziselem értékét közvetlenül a lemezen!
Az adatmozgatások alapműveletei:
INPUT(X): Az X adatbáziselemet tartalmazó lemezblokk másolása a memóriapufferbe.

READ(X,t): Az X adatbáziselem bemásolása a tranzakció t lokális változójába. Részletesebben: ha az X adatbáziselemet tartalmazó blokk nincs a memóriapufferben, akkor előbb végrehajtódik INPUT(X). Ezután kapja meg a t lokális változó X értékét.
WRITE(X,t): A t lokális változó tartalma az X adatbáziselem memóriapufferbeli tartalmába másolódik. Részletesebben: ha az X adatbáziselemet tartalmazó blokk nincs a memóriapufferben, akkor előbb végrehajtódik INPUT(X). Ezután másolódik át a t lokális változó értéke a pufferbeli X-be.
OUTPUT(X): Az X adatbáziselemet tartalmazó puffer kimásolása lemezre.
Az adatbázis elem mérete:
FELTEVÉS: az adatbáziselemek elférnek egy-egy lemezblokkban és így egy-egy pufferben is, azaz feltételezhetjük, hogy az adatbáziselemek pontosan a blokkok. Ha az adatbáziselem valójában több blokkot foglalna el, akkor úgy is tekinthetjük, hogy az adatbáziselem minden blokkméretű része önmagában egy adatbáziselem.
A naplózási mechanizmus atomos, azaz vagy lemezre írja X összes blokkját, vagy semmit sem ír ki.
A READ és a WRITE műveleteket a tranzakciók használják, az INPUT és OUTPUT műveleteket a pufferkezelő alkalmazza, illetve bizonyos feltételek mellett az OUTPUT műveletet a naplózási rendszer is használja.

Hibafajták:

- Hibás adatbevitel: pl. elütünk egy számot. Megoldás: triggerek, megszorítások.
- Készülékhibák: A lemezegységek jelentős sérülése, elsősorban a fejek katasztrófái, az egész lemez olvashatatlanná válását okozhatják.
- Katasztrofális hibák: Amikor az adatbázist tartalmazó eszköz teljesen tönkremegy. Pl. tűzkár. Védekezés: archiválás, redundáns osztott másolatok.
- Rendszerhibák: Minden tranzakciónak van állapota, mely azt képviseli, hogy mi történt eddig a tranzakcióban. Az állapot tartalmazza a tranzakció kódjában a végrehajtás pillanatnyi helyét és a tranzakció összes lokális változójának értékét. A rendszerhibák azok a problémák, melyek a tranzakció állapotának elvesztését okozzák.

Semmisségi naplózás:

A napló (log) naplóbejegyzések (log records) sorozata, melyek mindegyike arról tartalmaz valami információt, hogy mit tett egy tranzakció. Ha rendszerhiba fordul elő, akkor a napló segítségével rekonstruálható, hogy a tranzakció mit tett a hiba fellépéséig. A naplót (az archív mentéssel együtt) használhatjuk akkor is, amikor eszközhiba keletkezik a naplót nem tároló lemezen. Naplóbejegyzések:

- T_i kezdődik: (T_i, START)
- T_i írja A-t: (T_i, A, régi érték, új érték) - (néha elég csak a régi vagy csak az új érték, a naplózási protokolltól függően)
- T_i rendben befejeződött: (T_i, COMMIT)
- T_i a normálisnál korábban fejeződött be: (T_i, ABORT)

Az adatbázis visszaállítását olyan állapotba, mintha a tekintett tranzakció nem is működött volna semmisségi (undo) naplózásnak nevezzük.
Undo naplózás szabályai:

- U1. Ha a T tranzakció módosítja az X adatbáziselemet, akkor a (T, X, régi érték) naplóbejegyzést azelőtt kell a lemezre írni, mielőtt az X új értékét a lemezre írná a rendszer.
- U2. Ha a tranzakció hibamentesen befejeződött, akkor a COMMIT naplóbejegyzést csak azután szabad a lemezre írni, ha a tranzakció által módosított összes adatbáziselem már a lemezre íródott, de ezután rögtön.

Undo naplózás esetén a lemezre írás sorrendje

- Az adatbáziselemek módosítására vonatkozó naplóbejegyzések.
- Maguk a módosított adatbáziselemek.
- A COMMIT naplóbejegyzés.

Az első két lépés minden módosított adatbáziselemre vonatkozóan önmagában, külön-külön végrehajtandó (nem lehet a tranzakció több módosítására csoportosan megtenni)!

A naplóbejegyzések lemezre írásának kikényszerítésére a naplókezelőnek szüksége van a FLUSH LOG műveletre, mely felszólítja a pufferkezelőt az összes korábban még ki nem írt naplóblokk lemezre való kiírására.

Helyreállítás UNDO napló alapján:
A helyreállítás-kezelő feladata a napló használatával az adatbázist konzisztens állapotba visszaállítani. A helyreállítás-kezelő a naplót a végétől kezdi átvizsgálni (tehát az utoljára felírt bejegyzéstől a korábban felírtak irányában).

- Ha ugyanerre a T tranzakcióra vonatkozó COMMIT bejegyzéssel már találkozott, akkor nincs teendője, T rendesen befejeződött, hatásait nem kell tehát semmissé tenni.
- Minden más esetben T nem teljes vagy abortált tranzakció. A helyreállítás-kezelő az adatbáziselem értékét visszaállítjuk a régi értékre.
- Minden abortált vagy nem teljes tranzakcióra vonatkozóan <ABORT T> naplóbejegyzést ír a naplóba és kiváltja annak naplófájlba való kiírását is (FLUSH LOG).

Ellenőrzőpont:
CHECKPOINT képzése:

- Megtiltjuk az új tranzakciók indítását.
- Megvárjuk, amíg minden futó tranzakció COMMIT vagy ABORT módon véget ér.
- A naplót a pufferből a háttértárra írjuk (FLUSH LOG),
- Az adategységeket a pufferből a háttértárra írjuk.
- A naplóba beírjuk, hogy CHECKPOINT.
- A naplót újra a háttértárra írjuk: FLUSH LOG.
- Újra fogadjuk a tranzakciókat.

Ezután nyilván elég az első CHECKPOINT-ig visszamenni, hiszen előtte minden T_i már valahogy befejeződött.
Probléma: Hosszú ideig tarthat, amíg az aktív tranzakciók befejeződnek. (Új tranzakciókat sokáig nem lehet kiszolgálni.)
Megoldás: CHECKPOINT képzése működés közben.
A módszer lépései:

- <START CKPT(T_1,…,T_k)> naplóbejegyzés készítése, majd lemezre írása (FLUSH LOG), ahol T_1,…,T_k az éppen aktív tranzakciók nevei.
- Meg kell várni a T_1,…,T_k tranzakciók mindegyikének normális vagy abnormális befejeződését, nem tiltva közben újabb tranzakciók indítását.
- Ha a T_1,…,T_k tranzakciók mindegyike befejeződött, akkor <END CKPT> naplóbejegyzés elkészítése, majd lemezre írása (FLUSH LOG).

Készítette: **Utolsó módosítás:**
Gerstweiler Anikó Éva, Molnár Dávid 2011.01.08.

12.Helyrehozó (Redo) naplózás, semmiségi/helyrehozó (Undo/Redo) naplózás, archiválás, példák.

Helyrehozó naplózás:

A helyrehozó naplózás figyelmen kívül hagyja a be nem fejezett tranzakciókat és megismétli a normálisan befejezettek által végrehajtott választásokat.

A COMMIT naplóbejegyzés lemezre írását várja el, mielőtt bármit is változtatna a lemezen levő adatbázisban.

Az adatbázis elemek új értékeit tároljuk.

Szabályai: Mielőtt az adatbázis bármely X elemét a lemezen módosítanánk, szükséges, hogy az X ezen módosításaira vonatkozó összes naplóbejegyzése a lemezre kerüljenek.

A lemezre írás sorrendje:

- Az adatbázis elemek módosítását leíró naplóbejegyzések lemezre írása.
- A COMMIT naplóbejegyzés lemezre írása.
- Az adatbáziselemek értékének tényleges cseréje a lemezen.

Helyreállítás:

- Meghatározni a befejezett tranzakciókat
- Elemezni a naplót az elejétől kezdve. Minden <T,X,v> naplóbejegyzés megtalálásakor:
  - Ha T befejezetlen tranzakció, akkor nem kell tenni semmit.
  - Ha T befejezett tranzakció, akkor v értéket kell az X adatbáziselembe írni.
- Minden befejezetlen tranzakcióra vonatkozóan <ABORT T> naplóbejegyzést kell a naplóba írni és a naplót ki kell írni lemezre (FLUSH)

Ellenőrzőpont-képzés:

- Új probléma: a befejeződött tranzakciók módosításainak lemezre írása a befejeződés után sokkal később is történhet.
- Következmény: ugyanazon pillanatban aktív tranzakciók számát nincs értelme korlátozni, tehát nincs értelme az egyszerű ellenőrzőpont-képzésnek.
- A kulcsfeladat – amit meg kell tennünk az ellenőrzőpont-készítés kezdete és befejezése közötti időben – az összes olyan adatbáziselem lemezre való kiírása, melyeket befejezett tranzakciók módosítottak, és még nem voltak lemezre kiírva.
- Ennek megvalósításához a pufferkezelőnek nyilván kell tartania a piszkos puffereket, melyekben már végrehajtott, de lemezre még ki nem írt módosításokat tárol. Azt is tudnunk kell, hogy mely tranzakciók mely puffereket módosították.
- <START CKPT(T1,…,Tk)> naplóbejegyzés elkészítése és lemezre írása, ahol T1,…,Tk az összes éppen aktív tranzakció.
- Az összes olyan adatbáziselem kiírása lemezre, melyeket olyan tranzakciók írtak pufferekbe, melyek a START CKPT naplóba írásakor már befejeződtek, de puffereik lemezre még nem kerültek.
- <END CKPT> bejegyzés naplóba írása, és a napló lemezre írása.

Semmisségi/helyrehozó naplózás:

A naplóbejegyzés négykomponensű: a <T,X,v,w> naplóbejegyzés azt jelenti, hogy a T tranzakció az adatbázis X elemének korábbi v értékét w-re módosította. UR1: Mielőtt az adatbázis bármely X elemének értékét – valamely T tranzakció által végzett módosítás miatt – a lemezen módosítanánk, ezt megelőzően a <T,X,v,w> naplóbejegyzésnek lemezre kell kerülnie. Előbb naplózunk, utána módosítunk (WAL – Write After Log elv). A <T, COMMIT> bejegyzés megelőzheti, de követheti is az adatbáziselemek lemezen történő bármilyen megváltoztatását. A naplózás során a régi és az új értéket is tároljuk.

Helyreállítás:

- A legkorábbitól kezdve állítsunk helyre minden befejezett tranzakció hatását.
- A legutolsótól kezdve tegyük semmissé minden befejezetlen tranzakció cselekedeteit.

Ellenőrzőpont-képzés:

- Írjunk a naplóba <START CKPT(T1,…,Tk)> naplóbejegyzést, ahol T1,…,Tk az aktív tranzakciók, majd írjuk a naplót lemezre.

- Írjuk lemezre az összes piszkos puffert, tehát azokat, melyek egy vagy több módosított adatbáziselemet tartalmaznak. A helyrehozó naplózástól eltérően itt az összes piszkos puffert lemezre írjuk, nem csak a már befejezett tranzakciók által módosítottakat.
- Írjunk <END CKPT> naplóbejegyzést a naplóba, majd írjuk a naplót lemezre.

Védelmi módszerek lemezhibák ellen:

- Háromszoros redundancia: három másolat különböző lemezeken:
  - OUTPUT(X): 3 kiírás
  - INPUT(X): 3 beolvasás + szavazás
- Többszörös írás, egyszeres olvasás: N másolatot tartunk különböző lemezeken:
  - OUTPUT(X): N írás
  - INPUT(X): 1 másolat beolvasása → ha ok, kész, ha nem ok egy másik beolvasása
- Adatbázismentés + napló.

Ha az aktív adatbázis megsérül:

- Az adatbázis visszatöltése a mentésből.
- Redo napló alapján naprakész állapot visszaállítása.
- Az adatbázist a naplóból akkor tudjuk rekonstruálni, ha:
  - A naplót tároló lemez különbözik az adatbázist tartalmazó lemez(ek)től.
  - A naplót sosem dobjuk el az ellenőrzőpont-képzést követően.
  - A napló helyrehozó vagy semmisségi/helyrehozó típusú, így az új értékeket (is) tárolja.
  - Probléma: A napló esetleg az adatbázisnál is gyorsabban növekedhet, így nem praktikus a naplót örökre megőrizni.

A mentésnek két szintjét különböztetjük meg:

- Teljes mentés (full dump), amikor az egész adatbázisról másolat készül.
- Növekményes mentés (incremental dump), amikor az adatbázisnak csak azon elemeiről készítünk másolatot, melyek az utolsó teljes vagy növekményes mentés óta megváltoztak.

Mentés működés közben:

A működés közbeni archiválás az adatbázis elemeit valamely fix sorrendben másolja, mialatt megeshet, hogy ezen elemeket az éppen végrehajtott tranzakciók módosítják. Ennek eredményeként megtörténhet, hogy a biztonsági mentésbe másolt adatbáziselem értéke nem ugyanaz, mint a mentés megkezdésekor volt. Amíg a teljes mentés alatt keletkezett naplót megőrizzük, addig az eltérések a napló felhasználásával korrigálhatók.

- A <START DUMP> bejegyzés naplóba írása.
- A REDO vagy UNDO/REDO naplózási módnak megfelelő ellenőrzőpont kialakítása.
- Az adatlemez(ek) teljes vagy növekményes mentése.
- A napló mentése. A mentett naplórész tartalmazza legalább a 2. pontbeli ellenőrzőpont-képzés közben keletkezett naplóbejegyzéseket, melyeknek túl kell élniük az adatbázist hordozó eszköz meghibásodását.
- <END DUMP> bejegyzés naplóba írása.

Helyreállítás:

A teljes mentésből és a megfelelő növekményes mentésekből. Visszamásoljuk a teljes mentést, majd az ezt követő legkorábbi növekményes mentéstől kezdve végrehajtjuk a növekményes mentésekben tárolt változtatásokat.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

13.Az Oracle naplózási/helyreállítási megoldásai.

Az Oracle naplózási és archiválási rendszere:

Rendszerhiba esetén a helyreállítás-kezelő automatikusan aktivizálódik, amikor az Oracle újraindul.
A helyreállítás a napló (redo log) alapján történik. A napló olyan állományok halmaza, amelyek az adatbázis változásait tartalmazzák, akár lemezre kerültek, akár nem. Két részből áll: az online és az archivált naplóból.
Az online napló kettő vagy több online naplófájlból áll.
A naplóbejegyzések ideiglenesen az SGA (System Global Area) memóriapuffereiben tárolódnak, amelyeket a Log Writer (LGWR) háttérfolyamat folyamatosan ír ki lemezre. (Az SGA tartalmazza az adatbáziselemeket tároló puffereket is, amelyeket pedig a Database Writer háttérfolyamat ír lemezre.) Ha egy felhasználói folyamat befejezte egy tranzakció végrehajtását, akkor a LGWR egy COMMIT bejegyzést is kiír a naplóba.
Az online naplófájlok ciklikusan töltődnek föl. Például ha a naplót két fájl alkotja, akkor először az elsőt írja tele a LGWR, aztán a másodikat, majd újraírja az elsőt stb. Amikor egy naplófájl megtelt, kap egy sorszámot (log sequence number), ami azonosítja a fájlt.
A biztonság növelése érdekében az Oracle lehetővé teszi, hogy a naplófájlokat több példányban letároljuk. A multiplexelt online naplóban ugyanazon naplófájlok több különböző lemezen is tárolódnak, és ezek egyszerre módosulnak. Ha az egyik lemez megsérül, akkor a napló többi másolata még mindig rendelkezésre áll a helyreállításhoz.
Lehetőség van arra, hogy a megtelt online naplófájlokat archiváljuk, mielőtt újra felhasználnánk őket. Az archivált (offline) napló az ilyen archivált naplófájlokból tevődik össze.
A naplókezelő két módban működhet: ARCHIVELOG módban a rendszer minden megtelt naplófájlt archivál, mielőtt újra felhasználná, NOARCHIVELOG módban viszont a legrégebbi megtelt naplófájl mentés nélkül felülíródik, ha az utolsó szabad naplófájl is megtelt.

- ARCHIVELOG módban az adatbázis teljesen visszaállítható rendszerhiba és eszközhiba után is, valamint az adatbázist működés közben is lehet archiválni. Hátránya, hogy az archivált napló kezeléséhez külön adminisztrációs műveletek szükségesek.
- NOARCHIVELOG módban az adatbázis csak rendszerhiba után állítható vissza, eszközhiba esetén nem, és az adatbázist archiválni csak zárt állapotában lehet, működés közben nem. Előnye, hogy a DBA-nak nincs külön munkája, mivel nem jön létre archivált napló.

A naplót a LogMiner naplóelemző eszköz segítségével analizálhatjuk, amelyet SQL alapú utasításokkal vezérelhetünk.
A helyreállításhoz szükség van még egy vezérlőfájlra (control file) is, amely többek között az adatbázis fájlszerkezetéről és a LGWR által éppen írt naplófájl sorszámáról tartalmaz információkat. Az automatikus helyreállítási folyamatot a rendszer ezen vezérlőfájl alapján irányítja. Hasonlóan a naplófájlokhoz, a vezérlőfájlt is tárolhatjuk több példányban, amelyek egyszerre módosulnak. Ez a multiplexelt vezérlőfájl.

A rollback szegmensek és a helyreállítás folyamata:

Az Oracle az UNDO és a REDO naplózás egy speciális keverékét valósítja meg.
A tranzakciók hatásainak semmissé tételéhez szükséges információkat a rollback szegmensek tartalmazzák. Minden adatbázisban van egy vagy több rollback szegmens, amely a tranzakciók által módosított adatok régi értékeit tárolja attól függetlenül, hogy ezek a módosítások lemezre íródtak vagy sem. A rollback szegmenseket használjuk az olvasási konzisztencia biztosítására, a tranzakciók visszagörgetésére és az adatbázis helyreállítására is.
A rollback szegmens rollback bejegyzésekből áll. Egy rollback bejegyzés többek között a megváltozott blokk azonosítóját (fájlsorszám és a fájlon belüli blokkazonosító) és a blokk régi értékét tárolja. A rollback bejegyzés mindig előbb kerül a rollback szegmensbe, mint ahogy az adatbázisban megtörténik a módosítás. Az ugyanazon tranzakcióhoz tartozó bejegyzések össze vannak láncolva, így könnyen visszakereshetők, ha az adott tranzakciót vissza kell görgetni.

A rollback szegmenseket sem a felhasználók, sem az adatbázis-adminisztrátorok nem olvashatják. Mindig a SYS felhasználó a tulajdonosuk, attól függetlenül, ki hozta őket létre.
Minden rollback szegmenshez tartozik egy tranzakciós tábla, amely azon tranzakciók listáját tartalmazza, amelyek által végrehajtott módosításokhoz tartozó rollback bejegyzések az adott rollback szegmensben tárolódnak. Minden rollback szegmens fix számú tranzakciót tud kezelni. Ez a szám az adatblokk méretétől függ, amit viszont az operációs rendszer határoz meg. Ha explicit módon másképp nem rendelkezünk, az Oracle egyenletesen elosztja a tranzakciókat a rollback szegmensek között.
Ha egy tranzakció befejeződött, akkor a rá vonatkozó rollback bejegyzések még nem törölhetők, mert elképzelhető, hogy még a tranzakció befejeződése előtt elindult egy olyan lekérdezés, amelyhez szükség van a módosított adatok régi értékeire. Hogy a rollback adatok minél tovább elérhetők maradjanak, a rollback szegmensbe a bejegyzések sorban egymás után kerülnek be. Amikor megtelik a szegmens, akkor az Oracle az elejéről kezdi újra feltölteni. Előfordulhat, hogy egy sokáig futó tranzakció miatt nem írható felül a szegmens eleje, ilyenkor a szegmenst ki kell bővíteni.
Amikor létrehozunk egy adatbázist, automatikusan létrejön egy SYSTEM nevű rollback szegmens is a SYSTEM táblaterületen. Ez nem törölhető. Erre a szegmensre mindig szükség van, akár létrehozunk további rollback szegmenseket, akár nem. Ha több rollback szegmensünk van, akkor a SYSTEM nevűt az Oracle csak speciális rendszertranzakciókra próbálja használni, a felhasználói tranzakciókat pedig szétosztja a többi rollback szegmens között. Ha viszont túl sok felhasználói tranzakció fut egyszerre, akkor a SYSTEM szegmenst is használni fogja erre a célra.
Naplózás naplózása: Amikor egy rollback bejegyzés a rollback szegmensbe kerül, a naplóban erről is készül egy naplóbejegyzés, hiszen a rollback szegmensek (más szegmensekhez hasonlóan) az adatbázis részét képezik.
A helyreállítás szempontjából nagyon fontos a módosításoknak ez a kétszeres feljegyzése.
Ha rendszerhiba történik, először a napló alapján visszaállításra kerül az adatbázisnak a rendszerhiba bekövetkezése előtti állapota, amely inkonzisztens is lehet. Ez a folyamat a rolling forward.
A helyreállítás folyamán a rollback szegmens is visszaállítódik, amiben az aktív tranzakciók által végrehajtott tevékenységek semmissé tételéhez szükséges információk találhatók. Ezek alapján ezután minden aktív tranzakcióra végrehajtódik egy ROLLBACK utasítás. Ez a rolling back folyamat.

Az archiválás folyamata:
Az eszközhibák okozta problémák megoldására az Oracle is használja az archiválást.

- A teljes mentés az adatbázishoz tartozó adatfájlok, az online naplófájlok és az adatbázis vezérlőfájljának operációsrendszer-szintű mentését jelenti.
- Részleges mentés esetén az adatbázisnak csak egy részét mentjük, például az egy táblaterülethez tartozó adatfájlokat vagy csak a vezérlőfájlt. A részleges mentésnek csak akkor van értelme, ha a naplózás ARCHIVELOG módban történik. Ilyenkor a naplót felesleges újra archiválni.
- A mentés lehet teljes és növekményes is.
- Ha az adatbázist konzisztens állapotában archiváltuk, akkor konzisztens mentésről beszélünk. A konzisztens mentésből az adatbázist egyszerűen visszamásolhatjuk, nincs szükség a napló alapján történő helyreállításra.
- Lehetőség van az adatbázist egy régebbi mentésből visszaállítani, majd csak néhány naplóbejegyzést figyelembe véve az adatbázist egy meghatározott időpontbeli állapotába visszavinni. Ezt nem teljes helyreállításnak (incomplete recovery) nevezzük.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

14. Konkurenciavezérlés, ütemezés, sorbarendezhetőség, konfliktus-sorbarendezhetőség, megelőzési gráf, fogalmak, állítások (bizonyítás nélkül), példák.

Konkurenciavezérlés:

A tranzakciók közötti egymásra hatás az adatbázis-állapot inkonzisztenssé válását okozhatja, még akkor is, amikor a tranzakciók külön-külön megőrzik a konzisztenciát, és rendszerhiba sem történt.
Azt az általános folyamatot, amely biztosítja, hogy a tranzakciók egyidejű végrehajtása során megőrizzék a konzisztenciát, konkurenciavezérlésnek nevezzük.

Ütemezés:

Olyan ütemezéseket kell tekintenünk, amelyek biztosítják, hogy ugyanazt az eredményt állítják elő, mintha a tranzakciókat egyesével hajtottuk volna végre. Az ütemezés egy vagy több tranzakció által végrehajtott lényeges műveletek időben vett sorozata. Ütemezések során csak a READ és WRITE műveletekkel foglalkozunk.
Egy ütemezés soros, ha bármely két T és T' tranzakcióra, ha T-nek van olyan művelete, amely megelőzi a T' valamelyik műveletét, akkor a T összes művelete megelőzi T' valamennyi műveletét. Általában nem várjuk el, hogy az adatbázis végső állapota független legyen a tranzakciók sorrendjétől. A soros ütemezést a tranzakciók felsorolásával adjuk meg. Pl. (T_1, T_2).

Sorbarendezhetőség:

Egy ütemezés sorba rendezhető, ha ugyanolyan hatással van az adatbázis állapotára, mint valamelyik soros ütemezés, függetlenül az adatbázis kezdeti állapotától.
Jelölések:

- Egy tranzakció műveletét $r_i(X)$ vagy $w_i(X)$ formában fejezünk ki, amely azt jelenti, hogy a T_i tranzakció olvassa ill. írja az X adatbáziselemet.
- Egy T_i tranzakció az i indexű műveletekből álló sorozat.
- A T tranzakciók halmazának egy S ütemezése olyan műveletek sorozata, amelyben minden T halmazbeli T_i tranzakcióra teljesül, hogy T_i műveletei ugyanabban a sorrendben fordulnak elő S-ben, mint ahogy magában a T_i definíciójában szerepeltek. Azt mondjuk, hogy az S az őt alkotó tranzakciók műveleteinek átlapolása.

Konfliktus-sorbarendezhetőség:

Konfliktus akkor van, ha van olyan egymást követő műveletpár az ütemezésben, amelynek ha a sorrendjét felcseréljük, akkor legalább az egyik tranzakció viselkedése megváltozik. Tegyük fel, hogy T_i és T_j különböző tranzakciók. Ekkor nincs konfliktus, ha a pár:

- $r_i(X)$ és $r_j(Y)$, még akkor sem, ha $X = Y$.
- $r_i(X)$ és $w_j(Y)$, és ha $X \neq Y$.
- $w_i(X)$ és $r_j(Y)$, és ha $X \neq Y$.
- $w_i(X)$ és $w_j(Y)$, és ha $X \neq Y$.

Három esetben nem cserélhetjük fel a műveletek sorrendjét:

- $r_i(X)$ és $w_i(Y)$ konfliktusos pár, mivel egyetlen tranzakción belül a műveletek sorrendje rögzített, és az adatbázis-kezelő ezt a sorrendet nem rendezheti át.
- $w_i(X)$ és $w_j(X)$ konfliktusos pár, mivel X értéke az marad, amit T_j számolt ki. Ha felcseréljük a sorrendjüket, akkor pedig X-nek a T_i által kiszámolt értéke marad meg. A pesszimista feltevés miatt a T_i és a T_j által kiírt értékek lehetnek különbözőek, és ezért az adatbázis valamelyik kezdeti állapotára különbözni fognak.
- $r_i(X)$ és $w_j(X)$ továbbá $w_i(X)$ és $r_j(X)$ is konfliktusos pár. Ha átvisszük $w_j(X)$-et $r_i(X)$ elé, akkor a T_i által olvasott X-beli érték az lesz, amit a T_j kiírt, amiről pedig feltételeztük, hogy nem szükségképpen egyezik meg X korábbi értékével. Tehát $r_i(X)$ és $w_j(X)$ sorrendjének cseréje befolyásolja, hogy T_i milyen értéket olvas X-ből, ez pedig befolyásolja T_i működését.

Azt mondjuk, hogy két ütemezés konfliktus-ekvivalens, ha szomszédos műveleteinek konfliktus mentes cseréinek sorozatával az egyiket átalakíthatjuk a másikká. Azt mondjuk, hogy egy ütemezés

konfliktus-sorbarendezhető, ha konfliktus-ekvivalens valamely soros ütemezéssel. A konfliktus-sorbarendezhetőség elégséges feltétele a sorbarendezhetőségnek.

Megelőzési gráf:

Adott T_1 és T_2 tranzakciónak egy S ütemezése. Azt mondjuk, hogy T_1 megelőzi T_2-t, (jele: $T_1 <_S T_2$), ha van a T_1-ben olyan A_1 művelet és a T_2-ben olyan A_2 művelet, hogy:

- A_1 megelőzi A_2-t az S-ben,
- A_1 és A_2 ugyanarra az adatbáziselemre vonatkoznak és,
- A_1 és A_2 közül legalább az egyik írási művelet.

Egy körmentes gráf csúcsainak topologikus sorrendje a csúcsok bármely olyan rendezése, amelyben minden a → b élre az a csúcs megelőzi a b csúcsot a topologikus rendezésben.

Lemma: S_1, S_2 konfliktusekvivalens → gráf(S_1)=gráf(S_2)

Megjegyzés: gráf(S_1)=gráf(S_2)-ból nem következik, hogy S_1, S_2 konfliktusekvivalens.

Állítás: ha létezik $T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_n \rightarrow T_1$ n darab tranzakcióból álló kör, akkor a feltételezett soros sorrendben T_1 műveleteinek meg kell előzniük a T_2-ben szereplő műveleteket, amelyeknek meg kell előzniük a T_3-belieket és így tovább egészen T_n-ig. De T_n műveletei emiatt a T_1-beliek mögött vannak, ugyanakkor meg is kellene előzniük a T_1-belieket a $T_n \rightarrow T_1$ él miatt. Ebből következik, hogy ha a megelőzési gráf tartalmaz kört, akkor az ütemezés nem konfliktus-sorbarendezhető.

Készítette: **Utolsó módosítás:**
Gerstweiler Anikó Éva, Molnár Dávid 2011.01.08.

15.Zárolási ütemező, kétfázisú zárolás, holtpont, várakozási gráf, fogalmak, állítások (bizonyítás nélkül), példák.

Zárolási ütemező:

Az ütemező feladata az, hogy megakadályozza az olyan műveleti sorrendeket, amelyek nem sorba rendezhető ütemezésekhez vezetnek. Egyszerű megoldás: zárak használata. Csak egyféle zár van, amelyet a tranzakcióknak meg kell kapniuk az adatbáziselemre, ha bármilyen műveletet akarnak végrehajtani az elemen. Az lenne az ideális, ha az ütemező akkor és csak akkor továbbítana egy kérést, ha annak végrehajtása nem vezetne inkonzisztens adatbázis-állapotba. Minden ütemező egy tesztet hajt végre a sorbarendezhetőség biztosítására, azonban letilthat olyan műveleteket is, amelyek önmagukban nem vezetnének inkonzisztenciához. A zárolási ütemező a konfliktus-sorbarendezhetőséget követeli meg. Két új műveletet vezetünk be:

- $l_i(X)$: a T_i tranzakció az X adatbáziselemre zárolást kér (lock).
- $u_i(X)$: a T_i tranzakció az X adatbáziselem zárolását feloldja (unlock)

Tranzakciók konzisztenciája:

- A tranzakció akkor és csak akkor írhat egy elemet, ha már korábban zárolta az elemet, és még nem oldotta fel a zárat.
- Ha a tranzakció zárol egy elemet, akkor azt később fel kell szabadítania.
- Ha egy T_i tranzakcióban van egy $r_i(X)$ vagy egy $w_i(X)$ művelet, akkor van korábban egy $l_i(X)$ művelet, és van később egy $u_i(X)$ művelet, de a zárolás és az írás/olvasás között nincs $u_i(X)$.

Az ütemezés jogszerűsége: nem zárolhatja két tranzakció ugyanazt az elemet, csak úgy, ha az egyik előbb már feloldotta a zárat. Azaz ha egy ütemezésben van olyan $l_i(X)$ művelet, amelyet $l_j(X)$ követ, akkor e két művelet között lennie kell egy $u_i(X)$ műveletnek.

A záráson alapuló ütemező feladata, hogy akkor és csak akkor engedélyezze a kérések végrehajtását, ha azok jogszerű ütemezéseket eredményeznek.

Ezt a döntést segíti a zártábla, amely minden adatbáziselemhez megadja azt a tranzakciót, ha van ilyen, amelyik pillanatnyilag zárolja az adott elemet.

A zártábla szerkezete (egyféle zárolás esetén): Zárolások(elem, tranzakció) relációt, ahol a T tranzakció zárolja az X adatbáziselemet. Az ütemezőnek csak le kell kérdeznie ezt a relációt illetve egyszerű INSERT és DELETE utasításokkal kell módosítania.

Kétfázisú zárolás:

Van egy feltétel, amellyel biztosítani tudjuk, hogy konzisztens tranzakciók jogszerű ütemezése konfliktus-sorbarendezhető legyen. Ezt a feltételt kétfázisú zárolásnak nevezzük (2FZ), miszerint: minden tranzakcióban minden zárolási művelet megelőzi az összes zárfeloldási műveletet. Az első fázisban csak zárolásokat adunk ki, míg a második fázisban csak feloldjuk a zárokat.

Tétel: bármely konzisztens, 2FZ tranzakciók bármely S jogszerű ütemezése átalakítható konfliktus-ekvivalens soros ütemezéssé.

Probléma: nem lehet 2FZ segítségével a holtpont kialakulását megelőzni. Holtpont: Az ütemező arra kényszeríti a tranzakciót, hogy egy olyan zárra várjon, amelyet egy másik tranzakció tart zárva.

Holtpontkezelés: megoldása során legalább egy tranzakciót vissza kell görgetni, abortálni kell, majd újraindítani. A holtpont érzékelésére és feloldására egyszerű módszer az időtúllépés módszere, megadunk egy időkorlátot, amely arra vonatkozik, hogy egy tranzakció mennyi ideig lehet aktív, és ha ezt a határt túllépi, visszagörgetjük.

Várakozási gráf:

Azok a holtpontok, amelyek azért alakultak ki, mert az egyik tranzakció a másik birtokában levő zárakra vár, jól kezelhetők a várakozási gráfokkal. A gráfban azt tartjuk nyilván, hogy melyik tranzakció melyik tranzakcióra vár. Segítségével észlelhetőek lesznek a már kialakult holtpontok és meg is előzhetőek a kialakulásuk. Az utóbbinál a várakozási gráfot egész idő alatt nyilván kell tartani, és az olyan műveleteket, amelyek következtében a gráfban kör alakul ki, nem szabad megengednünk.

A zártáblában minden X elemhez tartozik egy lista, amelyben azon tranzakciók vannak, amelyek

várnak X zárolására illetve már zárolták az X adatbáziselemet. A várakozási gráf csúcsai a listában található tranzakcióknak felelnek meg. A gráfban irányított él fut T-ből U-ba, ha létezik olyan A adatbázis elem, hogy:

- U zárolja A-t.
- T arra vár, hogy zárolhassa A-t.
- T csak akkor kapja meg a számára megfelelő módban A zárját, ha először U lemond róla.

Ha nincs irányított kör a gráfban, akkor végül minden tranzakció be tudj a fejezni a működését, mert mindig lesz egy olyan tranzakció, amely nem vár semelyik másikra és továbbléphet.

Tétel: Az ütemezés során egy adott pillanatban pontosan akkor nincs holtpont, ha az adott pillanathoz tartozó várakozási gráfban nincs irányított kör.

Megoldások holtpont ellen:

- Rajzoljuk folyamatosan a várakozási gráfot, és ha holtpont alakul ki, akkor ABORT-áljuk az egyik olyan tranzakciót, aki benne van a kialakult irányított körben.
- Pesszimista hozzáállás: ha hagyjuk, hogy mindenki össze-vissza kérjen zárat, abból baj lehet. Előzzük inkább meg a holtpont kialakulását valahogyan. Lehetőségek:
  - Minden egyes tranzakció előre elkéri az összes zárat, ami neki kelleni fog. Ha nem kapja meg az összeset, akkor egyet se kér el, el se indul. Ilyenkor biztos nem lesz holtpont, mert ha valaki megkap egy zárat, akkor le is tud futni, nem akad el. Az csak a baj ezzel, hogy előre kell mindent tudni.
  - Feltesszük, hogy van egy sorrend az adategységeken és minden egyes tranzakció csak e szerint a sorrend szerint növekvően kérhet újabb zárakat. Itt lehet, hogy lesz várakozás, de holtpont biztos nem lesz.
- Időkorlát segítségével.

Készítette: **Utolsó módosítás:**
Gerstweiler Anikó Éva, Molnár Dávid 2011.01.08.

16. Különböző zármódú zárolási rendszerek, kompatibilitási mátrix, felminősítés, módosítási zárak, növelési zárak, fogalmak, állítások (bizonyítás nélkül), példák.

A zárolások legfőbb problémája, hogy a T tranzakciónak akkor is zárolnia kell egy X adatbáziselemet, ha csak olvasni akarja X-et. → Két különböző zárat használjunk. Egyiket csak olvasásra (osztott zár - s) a másikat csak íráshoz (kizárólagos zár – e). Tetszőleges X adatbáziselemet csak egyszer lehet zárolni kizárólagosan vagy akárhányszor lehet zárolni osztottan, ha még nem volt zárolva kizárólagosan.
Amikor írni akarjuk X-et, akkor X-en kizárólagos zárral kell rendelkeznünk, de ha csak olvasni akarjuk, akkor X-en akár osztott, akár kizárólagos zár megfelel.
Feltételezzük, hogy ha olvasni akarjuk X-et, de írni nem, akkor előnyben részesítjük az osztott zárolást.
Az $sl_i(X)$ jelölést használjuk arra, hogy a T_i tranzakció osztott zárat kér az X adatbáziselemre, az $xl_i(X)$ jelölést pedig arra, hogy a T_i kizárólagos zárat kér X-re. Továbbra is $u_i(X)$-szel jelöljük, hogy T_i feloldja X zárását, vagyis felszabadítja X-et minden zár alól.

Tranzakciók konzisztenciája:
Nem írhatunk kizárólagos zár fenntartása nélkül, és nem olvashatunk valamilyen zár fenntartása nélkül. Pontosabban fogalmazva: bármely T_i tranzakcióban:
- az $r_i(X)$ olvasási műveletet meg kell, hogy előzze egy $sl_i(X)$ vagy egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$; pl. $sl_i(X)...r_i(X)$vagy $xl_i(X)...r_i(X)$
- a $w_i(X)$ írási műveletet meg kell, hogy előzze egy $xl_i(X)$ úgy, hogy közben nincs $u_i(X)$; pl. $xl_i(X)...w_i(X)$

Minden zárolást követnie kell egy ugyanannak az elemnek a zárolását feloldó műveletnek.

Tranzakciók kétfázisú zárolása:
A zárolásoknak meg kell előzniük a zárak feloldását. Pontosabban fogalmazva: bármely T_i kétfázisú zárolású tranzakcióban egyetlen $sl_i(X)$ vagy $xl_i(X)$ műveletet sem előzhet meg egyetlen $u_i(Y)$ művelet sem semmilyen Y-ra. Tehát a $sl_i(X)...u_i(Y)$, vagy $xl_i(X)...u_i(Y)$ a helyes.

Az ütemezések jogszerűsége:
Egy elemet vagy egyetlen tranzakció zárol kizárólagosan, vagy több is zárolhatja osztottan, de a kettő egyszerre nem lehet. Pontosabban fogalmazva:
- Ha $xl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ vagy $sl_j(X)$ valamely i-től különböző j-re anélkül, hogy közben ne szerepelne $u_i(X)$. pl. $xl_i(X)...u_i(X)...xl_j(X)$ vagy $xl_i(X)...u_i(X)...sl_j(X)$
- Ha $sl_i(X)$ szerepel egy ütemezésben, akkor ezután nem következhet $xl_j(X)$ valamely i-től különböző j-re anélkül, hogy közben ne szerepelne $u_i(X)$. pl. $sl_i(X)...u_i(X)...xl_j(X)$

Tétel: Konzisztens 2FZ tranzakciók jogszerű ütemezése konfliktus-sorbarendezhető.

Kompatibilitási mátrix:
A kompatibilitási mátrix minden egyes zármódhoz rendelkezik egy-egy sorral és egy-egy oszloppal. A sorok egy másik tranzakció által az X elemre elhelyezett záraknak, az oszlopok pedig az X-re kért zármódoknak felelnek meg. A kompatibilitási mátrix használatának szabálya:
Egy A adatbáziselemre C módú zárat akkor és csak akkor engedélyezhetünk, ha a táblázat minden olyan R sorára, amelyre más tranzakció már zárolta A-t R módban, a C oszlopban "Igen" szerepel.

	S	X
S	Igen	Nem
X	Nem	Nem

Tétel: Egy csak zárkéréseket és zárelengedéseket tartalmazó jogszerű ütemezés sorbarendezhető akkor és csak akkor, ha a kompatibilitási mátrix alapján felrajzolt megelőzési gráf nem tartalmaz irányított kört.
Sorbarendezhetőség biztosítása tetszőleges zármodellben:

- Az ütemező egyik lehetősége a sorbarendezhetőség elérésére, hogy folyamatosan figyeli a megelőzési gráfot, és ha irányított kör keletkezne, akkor ABORT-ot rendel el.
- Másik lehetőség a protokollal való megelőzés. Tetszőleges zármodellben értelmes a 2PL és igaz az alábbi tétel:

Tétel: Ha valamilyen zármodellben egy jogszerű ütemezésben minden tranzakció követi a 2PL-t, akkor az ütemezéshez tartozó megelőzési gráf nem tartalmaz irányított kört, azaz az ütemezés sorbarendezhető.
Megjegyzés: Minél gazdagabb a zármodell, minél több az "Igen" a kompatibilitási mátrixban, annál valószínűbb, hogy a megelőzési gráfban nincs irányított kör, minden külön protokoll nélkül.
L_2 erősebb L_1-nél, ha a kompatibilitási mátrixban L_2 sorában/oszlopában minden olyan pozícióban "Nem" áll, amelyben L_1 sorában/oszlopában "Nem" áll.
Például az SX zárolási séma esetén X erősebb S-nél (X minden zármódnál erősebb, hiszen X sorában és oszlopában is minden pozíción "Nem" szerepel).

Felminősítés:
Azt mondjuk, hogy a T tranzakció felminősíti az L_1 zárját az L_1-nél erősebb L_2 zárra az A adatbáziselemen, ha L_2 zárat kér (és kap) A-ra, amelyen már birtokol egy L_1 zárat (azaz még nem oldotta fel L_1-et). Például: $sl_i(A).....Xl_i(A)$. A felminősítés válogatás nélküli alkalmazása a holtpontok új forrását jelenti.
Pl. T_1 és T_2 is kaphat osztott zárat A-ra. Ezután mindkettő megpróbálja ezt felminősíteni kizárólagossá, de az ütemező mindkettőt várakozásra kényszeríti, hiszen a másik már osztottan zárolta A-t. Emiatt egyik végrehajtása sem folytatódhat, vagy mindkettőnek örökösen kell várakoznia, vagy addig amíg a rendszer fel nem fedezi hogy holtpont van, abortálja a két tranzakció valamelyikét és a másiknak engedélyezi az A -ra kizárólagos zárat.
Az $ul_i(X)$ módosítási zár a T_i tranzakciónak csak X olvasására ad jogot, X írására nem. Később azonban csak a módosítási zárat lehet felminősíteni írásira, az olvasási zárat nem (azt csak módosítására). A módosítási zár tehát nem csak a holtpontproblémát oldja meg, hanem a kiéheztetés problémáját is.

	S	X	U
S	Igen	Nem	Igen
X	Nem	Nem	Nem
U	Nem	Nem	Nem

Az U módosítási zár úgy néz ki, mintha osztott zár lenne, amikor kérjük, és úgy néz ki, mintha kizárólagos zár lenne, amikor már megvan.

Számos tranzakciónak csak az a hatása, hogy növeli vagy csökkenti a tárolt értékeket. A növelési műveletek érdekes tulajdonsága, hogy tetszőleges sorrendben kiszámíthatók. A növelési művelet jelöljük inc(A,c)-vel, megnöveli A értékét egy c konstanssal. Ha c negatív → csökkentés. Formálisan: inc(A,c) a következő lépések atomi végrehajtására szolgál: READ(A,t); t:= t+c; WRITE(A,t);
Szükségünk van a növelési műveletnek megfelelő növelési zárra, amelyet $il_i(X)$-szel fogunk jelölni, mely a T_i tranzakciónak az X-re vonatkozó növelési zárolás kérése. Az $inc_i(X)$ jelentése: a T_i tranzakció megnöveli X adatbáziselemet egy konstanssal.
Egy konzisztens tranzakció csak akkor végezheti el X-en a növelési műveletet, ha egyidejűleg növelési zárat tart fenn rajta. A növelési zár viszont nem teszi lehetővé sem az olvasási, sem az írási műveleteket. $il_i(X)...inc_i(X)$
Az $inc_i(X)$ művelet konfliktusban áll $r_j(X)$-szel és $w_j(X)$-szel is $j \neq i$-re, de nem áll konfliktusban $inc_j(X)$-szel.
Egy jogszerű ütemezésben bármennyi tranzakció bármikor fenntarthat X-en növelési zárat. Ha viszont egy tranzakció növelési zárat tart fenn X-en, akkor egyidejűleg semelyik más tranzakció sem tarthat fenn sem osztott, sem kizárólagos zárat X-en.

	S	X	I
S	Igen	Nem	Nem
X	Nem	Nem	Nem
I	Nem	Nem	Igen

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

17.Zárolási ütemező felépítése, zártáblák.

Zárolási ütemező felépítése:

- Maguk a tranzakciók nem kérnek zárakat, vagy figyelmen kívül hagyjuk, hogy ezt teszik. Az ütemező szúrja be a zárolási műveleteket az adatokhoz hozzáférő olvasási, írási, illetve egyéb műveletek sorába.
- Nem a tranzakciók, hanem az ütemező oldja fel a zárakat, mégpedig akkor, amikor a tranzakciókezelő a tranzakció véglegesítésére vagy abortálására készül.

Az I. rész fogadja a tranzakciók által generált kérések sorát, és minden adatbázis-hozzáférési művelet elé beszúrja a megfelelő zárolási műveletet. Az adatbázis-hozzáférési és zárolási műveleteket ezután átküldi a II. részhez (a COMMIT és ABORT műveleteket nem).
A II. rész fogadja az I. részen keresztül érkező zárolási és adatbázis-hozzáférési műveletek sorozatát. Eldönti, hogy a T tranzakció késleltetett-e (mivel zárolásra vár). Ha igen, akkor magát a műveletet késlelteti, azaz hozzáadja azoknak a műveleteknek a listájához, amelyeket a T tranzakciónak még végre kell hajtania. Ha T nem késleltetett, vagyis az összes előzőleg kért zár már engedélyezve van, akkor megnézi, hogy milyen műveletet kell végrehajtania. Ha a művelet adatbázis-hozzáférés, akkor továbbítja az adatbázishoz, és végrehajtja. Ha zárolási művelet érkezik, akkor megvizsgálja a zártáblát, hogy a zár engedélyezhető-e. Ha igen, akkor úgy módosítja a zártáblát, hogy az az éppen engedélyezett zárat is tartalmazza. Ha nem, akkor egy olyan bejegyzést készít a zártáblában, amely jelzi a zárolási kérést. Az ütemező II. része ezután késlelteti a T tranzakció további műveleteit mindaddig, amíg nem tudja engedélyezni a zárat.
Amikor a T tranzakciót véglegesítjük vagy abortáljuk, akkor a tranzakciókezelő COMMIT, illetve ABORT műveletek küldésével értesíti az I. részt, hogy oldja fel az összes T által fenntartott zárat. Ha bármelyik tranzakció várakozik ezen zárfeloldások valamelyikére, akkor az I. rész értesíti a II. részt.
Amikor a II. rész értesül, hogy egy X adatbáziselemen felszabadult egy zár, akkor eldönti, hogy melyik az a tranzakció, vagy melyek azok a tranzakciók, amelyek megkapják a zárat X-re. A tranzakciók, amelyek megkapták a zárat, a késleltetett műveleteik közül annyit végrehajtanak, amennyit csak végre tudnak hajtani mindaddig, amíg vagy befejeződnek, vagy egy másik olyan zárolási kéréshez érkeznek el, amely nem engedélyezhető.
Zárolási műveleteket beszúró ütemező: Ha csak egymódú zárak vannak, akkor az ütemező I. részének a feladata egyszerű. Ha bármilyen műveletet lát az A adatbáziselemen, és még nem szúrt be zárolási kérést A-ra az adott tranzakcióhoz, akkor beszúrja a kérést. Amikor véglegesítjük vagy abortáljuk a tranzakciót, az I. rész törölheti ezt a tranzakciót, miután feloldotta a zárakat, így az I. részhez igényelt memória nem nő korlátlanul.
Amikor többmódú zárak vannak, az ütemezőnek szüksége lehet arra (például felminősítésnél), hogy azonnal értesüljön, milyen későbbi műveletek fognak előfordulni ugyanazon az adatbáziselemen.

Zártábla:

Absztrakt szinten a zártábla egy olyan reláció, amely összekapcsolja az adatbáziselemeket az elemre vonatkozó zárolási információval. A táblát pl. egy hasítótáblával lehet megvalósítani, amely az adatbáziselemek címeit használja tördelésre. Azok az elemek, amelyek nincsenek zárolva, nincsenek a táblában, így a méret csak a zárolt elemek számával arányos. Egy tipikus adatbáziselemhez a bejegyzés a következő komponensekből áll:

- Csoportos mód: a legszigorúbb feltételek összefoglalása, amivel egy tranzakció szembesül, amikor egy új zárolást kér az adatbáziselemen. Az osztott-kizárólagos-módosítási szabály (SXU) sémához:
  - S: csak osztott zárak vannak
  - U: egy módosítási zár van, de lehet még 1 vagy több módosítási zár
  - X: csak egy kizárólagos zár van és semmilyen más zár nincs
- Várakozási bit: azt adja meg, hogy van-e legalább egy tranzakció, amely az A zárolására várakozik.
- Az összes olyan tranzakciót leíró lista, amelyek zárolják A-t vagy A zárolására várakoznak. Tartalmazza:
  - A zárolást fenntartó vagy a zárolásra várakozó tranzakció nevét,

- ennek a zárnak a módja,
- a tranzakció fenntartja-e vagy várakozik a zárra,
- az adott tranzakció következő bejegyzése $T_{köv}$.

Zárolási kérések kezelése:

Ha a T tranzakció zárat kér A-ra. Ha nincs A-ra bejegyzés a zártáblában, akkor biztos, hogy zárak sincsenek A-n, így létrehozhatjuk a bejegyzést, és engedélyezhetjük a kérést. Ha a zártáblában létezik bejegyzés A-ra, akkor megkeressük a csoportos módot, és ez alapján várakoztatunk, beírjuk a várakozási listába, vagy megengedjük a zárat (például ha T_2 X-et kér A-ra).

Zárfeloldások kezelése:

Több különböző megközelítés lehetséges, mindegyiknek megvan a saját előnye:

- Első beérkezett első kiszolgálása: Azt a zárolási kérést engedélyezzük, amelyik a legrégebb óta várakozik. Ez biztosítja, hogy ne legyen kiéheztetés.
- Elsőbbségadás az osztott záraknak: Először az összes várakozó osztott zárat engedélyezzük. Ezután egy módosítási zárolást engedélyezünk, ha várakozik ilyen. A kizárólagos zárolást csak akkor engedélyezzük, ha semmilyen más igény nem várakozik. Ez a stratégia csak akkor engedi a kiéheztetést, ha a tranzakció U vagy X zárolásra vár.
- Elsőbbségadás a felminősítésnek: Ha van olyan U zárral rendelkező tranzakció, amely X zárrá való felminősítésre vár, akkor ezt engedélyezzük előbb. Máskülönben a fent említett stratégiák valamelyikét követjük.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

18.Figyelmeztető zárak, fantomok, nem megismételhető olvasás.

Adatbáziselemekből álló hierarchiák kezelése:

Kétféle fastruktúrával fogunk foglalkozni:

- Az első fajta fastruktúra, amelyet figyelembe veszünk, a zárolható elemek (zárolási egységek) hierarchiája. Megvizsgáljuk, hogyan engedélyezünk zárolást mind a nagy elemekre, mint például a relációkra, mind a kisebb elemekre, mint például a reláció néhány sorát tartalmazó blokkokra vagy egyedi sorokra.
- A másik lényeges hierarchiafajtát képezik a konkurenciavezérlési rendszerekben azok az adatok, amelyek önmagukban faszervezésűek. Ilyenek például a B-fa-indexek. A B-fák csomópontjait adatbáziselemeknek tekinthetjük, így viszont az eddig tanult zárolási sémákat szegényesen használhatjuk, emiatt egy új megközelítésre van szükségünk.

Az alapkérdés, hogy kicsi vagy nagy objektumokat zároljunk?

- Ha nagy objektumokat (például dokumentumokat, táblákat) zárolunk, akkor kevesebb zárra lesz szükség, de csökken a konkurens működés lehetősége.
- Ha kicsi objektumokat (például számlákat, sorokat vagy mezőket) zárolunk, akkor több zárra lesz szükség, de nő a konkurens működés lehetősége.
- Megoldás: Kicsi és nagy objektumokat is zárolhassunk!

Az adatbáziselemek több (például: három) szintjét különböztetjük meg:

- A relációk a legnagyobb zárolható elemek.
- Minden reláció egy vagy több blokkból vagy lapból épül fel, amelyekben a soraik vannak.
- Minden blokk egy vagy több sort tartalmaz.

Figyelmeztető zárak:

Az adatbáziselemek hierarchiáján a zárak kezelésére szolgáló szabályok alkotják a figyelmeztető protokollt, amely tartalmazza mind a „közönséges” zárakat, mind a „figyelmeztető” zárakat. A közönséges zárakat S és X, a figyelmeztető zárakat a közönséges zárak elő helyezett I előtaggal: IS, IX jelöljük. A figyelmeztető protokoll szabályai:

- Ahhoz hogy elhelyezzünk egy közönséges zárat valamely elemen, a hierarchia gyökerénél kell kezdenünk.
- Ha már annál az elemnél vagyunk, amelyet zárolni szeretnénk, akkor nem kell tovább folytatnunk, hanem kérjünk a közönséges zárat.
- Ha az elem, amelyet zárolni szeretnénk, lejjebb van a hierarchiában, akkor elhelyezünk egy figyelmeztető zárat ezen a csomóponton. Amikor a figyelmeztető zárat megkaptuk, akkor az ehhez a csomóponthoz tartozó utódcsomóponttal folytatjuk. A 2.-ik és a 3-ik lépéssel folytatjuk a keresett csomópontig.

	IS	IX	S	X
IS	Igen	Igen	Igen	Nem
IX	Igen	Igen	Nem	Nem
S	Igen	Nem	Igen	Nem
X	Nem	Nem	Nem	Nem

Ha IS zárat kérünk egy N csomópontban, az N egy leszármazottját szándékozzuk olvasni. Ez csak abban az esetben okozhat problémát, ha egy másik tranzakció már jogosulttá vált arra, hogy az N által reprezentált teljes adatbáziselemet felülírja (X). Ha más tranzakció azt tervezi, hogy N-nek csak egy részelemét írja (ezért az N csomóponton egy IX zárat helyezett el), akkor lehetőségünk van arra, hogy engedélyezzük az IS zárat N-en, és a konfliktust alsóbb szinten oldhatjuk meg, ha az írási és olvasási szándék valóban egy közös elemre vonatkozik.

Ha az N csomópont egy részelemét szándékozzuk írni (IX), akkor meg kell akadályoznunk az N által képviselt teljes elem olvasását vagy írását (S vagy X). Azonban más tranzakció, amely egy részelemet olvas vagy ír, a potenciális konfliktusokat az adott szinten kezeli le, így az IX nincs konfliktusban egy másik IX-szel vagy IS-sel N-en.

Az N csomópontnak megfeleltetett elem olvasása (S) nincs konfliktusban sem egy másik olvasási zárral N-en, sem egy olvasási zárral N egy részelemén, amelyet N-en egy IS reprezentál. Azonban egy X vagy egy IX azt jelenti, hogy más tranzakció írni fogja legalább egy részét az N által reprezentált elemnek. Ezért nem tudjuk engedélyezni N teljes olvasását.
Nem tudjuk megengedni az N csomópont írását sem (X), ha más tranzakciónak már joga van arra, hogy olvassa vagy írja N-et (S,X), vagy arra, hogy megszerezze ezt a jogot N egy részelemére (IS,IX).

Nem megismételhető olvasás:
Tegyük fel, hogy van egy T_1 tranzakció, amely egy adott feltételnek eleget tevő sorokat válogat ki egy relációból. Ezután hosszas számításba kezd, majd később újra végrehajtja a fenti lekérdezést.
Tegyük fel továbbá, hogy a lekérdezés két végrehajtása között egy T_2 tranzakció módosít vagy töröl a táblából néhány olyan sort, amely eleget tesz a lekérdezés feltételének.
A T_1 tranzakció lekérdezését ilyenkor nem ismételhető olvasásnak nevezzük.
A nem ismételhető olvasással az a probléma, hogy mást eredményez a lekérdezés másodszori végrehajtása, mint az első.
A tranzakció viszont elvárhatja (ha akarja), hogy ha többször végrehajtja ugyanazt a lekérdezést, akkor mindig ugyanazt az eredményt kapja.

Fantom:
Ugyanez a helyzet akkor is, ha a T_2 tranzakció beszúr olyan sorokat, amelyek eleget tesznek a lekérdezés feltételének. A lekérdezés másodszori futtatásakor most is más eredményt kapunk, mint az első alkalommal. Ennek az az oka, hogy most olyan sorokat is figyelembe kellett venni, amelyek az első futtatáskor még nem is léteztek. Az ilyen sorokat nevezzük fantomoknak.

A fenti jelenségek általában nem okoznak problémát, ezért a legtöbb adatbázis-kezelő rendszer alapértelmezésben nem is figyel rájuk. A fejlettebb rendszerekben azonban a felhasználó kérheti, hogy a nem ismételhető olvasások és a fantomolvasások ne hajtódjanak végre. Ilyen esetekben rendszerint egy hibaüzenetet kapunk, amely szerint a T_1 tranzakció nem sorbarendezhető ütemezést eredményezett, és az ütemező abortálja T_1-et.
Figyelmeztető protokoll használata esetén viszont könnyen megelőzhetjük az ilyen szituációkat, mégpedig úgy, hogy a T_1 tranzakciónak S módban kell zárolnia a teljes relációt, annak ellenére, hogy csak néhány sorát szeretné olvasni. A módosító/törlő/beszúró tranzakció ezek után IX módban szeretné zárolni a relációt. Ezt a kérést az ütemező először elutasítja, és csak akkor engedélyezi, amikor a T_1 tranzakció már befejeződött, elkerülve ezáltal a nem sorbarendezhető ütemezést.

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

~~19.Időbélyegzés, érvényesítés.~~

Ez a tétel nem fog szerepelni vizsgán!

Készítette:
Gerstweiler Anikó Éva, Molnár Dávid

Utolsó módosítás:
2011.01.08.

20.Az Oracle tranzakció-kezelési megoldásai, elkülönítési szintek, zárolások.

Tranzakció-kezelés:

Az Oracle alapvetően a zárolás módszerét használja a konkurenciavezérléshez. Felhasználói szinten a zárolási egység lehet a tábla vagy annak egy sora. A zárakat az ütemező helyezi el és oldja fel, de lehetőség van arra is, hogy a felhasználó (alkalmazás) kérjen zárat. Az Oracle alkalmazza a kétfázisú zárolást, a figyelmeztető protokollt és a többváltozatú időbélyegzőket is némi módosítással.

Az Oracle minden lekérdezés számára biztosítja az olvasási konzisztenciát, azaz a lekérdezés által olvasott adatok egy időpillanatból (a lekérdezés kezdetének pillanatából) származnak. Emiatt a lekérdezés sohasem olvas piszkos adatot, és nem látja azokat a változtatásokat sem, amelyeket a lekérdezés végrehajtása alatt véglegesített tranzakciók eszközöltek. Ezt utasítás szintű olvasási konzisztenciának nevezzük.

Kérhetjük egy tranzakció összes lekérdezése számára is a konzisztencia biztosítását, ez a tranzakció szintű olvasási konzisztencia. Ezt úgy érhetjük el, hogy a tranzakciót sorba rendezhető vagy csak olvasás módban futtatjuk. Ekkor a tranzakció által tartalmazott összes lekérdezés a tranzakció indításakor fennálló adatbázis-állapotot látja, kivéve a tranzakció által korábban végrehajtott módosításokat.

A kétféle olvasási konzisztencia eléréséhez az Oracle a rollback szegmensekben található információkat használja fel. A rollback szegmensek tárolják azon adatok régi értékeit, amelyeket még nem véglegesített vagy nemrég véglegesített tranzakciók változtattak meg.

Amint egy lekérdezés vagy tranzakció megkezdi működését, meghatározódik a system change number (SCN) aktuális értéke. Az SCN a blokkokhoz mint adatbáziselemekhez tartozó időbélyegzőnek tekinthető.

Ahogy a lekérdezés olvassa az adatblokkokat, összehasonlítja azok SCN-jét az aktuális SCN értékkel, és csak az aktuálisnál kisebb SCN-nel rendelkező blokkokat olvassa be a tábla területéről.

A nagyobb SCN-nel rendelkező blokkok esetén a rollback szegmensből megkeresi az adott blokk azon verzióját, amelyhez a legnagyobb olyan SCN érték tartozik, amely kisebb, mint az aktuális, és már véglegesített tranzakció hozta létre.

Elkülönítési szintek:

Az SQL92 ANSI/ISO szabvány a tranzakció-elkülönítés négy szintjét definiálja, amelyek abban különböznek egymástól, hogy az alábbi három jelenség közül melyeket engedélyezik:

- Piszkos olvasás: a tranzakció olyan adatot olvas, amelyet egy másik, még nem véglegesített tranzakció írt.
- Nem ismételhető olvasás: a tranzakció újraolvas olyan adatokat, amelyeket már korábban beolvasott, és azt találja, hogy egy másik, már véglegesített tranzakció módosította vagy törölte őket.
- Fantomok olvasása: a tranzakció újra végrehajt egy lekérdezést, amely egy adott keresési feltételnek eleget tevő sorokkal tér vissza, és azt találja, hogy egy másik, már véglegesített tranzakció további sorokat szúrt be, amelyek szintén eleget tesznek a feltételnek.

	Piszkos olvasás	Nem ismételhető olvasás	Fantomok olvasása
Nem olvasásbiztos	Lehetséges	Lehetséges	Lehetséges
Olvasásbiztos	Nem lehetséges	Lehetséges	Lehetséges
Megismételhető olvasás	Nem lehetséges	Nem lehetséges	Lehetséges
Sorbarendezhető	Nem lehetséges	Nem lehetséges	Nem lehetséges

Az Oracle ezek közül használja:

- Olvasásbiztos: Ez az alapértelmezett tranzakció-elkülönítési szint. Egy tranzakció minden lekérdezése csak a lekérdezés (és nem a tranzakció) elindítása előtt véglegesített adatokat látja. Piszkos olvasás sohasem történik. A lekérdezés két végrehajtása között a lekérdezés által olvasott adatokat más tranzakciók megváltoztathatják, ezért előfordulhat nem ismételhető olvasás és

fantomok olvasása is. Olyan környezetekben célszerű ezt a szintet választani, amelyekben várhatóan kevés tranzakció kerül egymással konfliktusba.
- Sorbarendezhető: A sorbarendezhető tranzakciók csak a tranzakció elindítása előtt véglegesített változásokat látják, valamint azokat, amelyeket maga a tranzakció hajtott végre INSERT, UPDATE és DELETE utasítások segítségével. Nem hajtanak végre nem ismételhető olvasásokat, és nem olvasnak fantomokat. Ezt a szintet olyan környezetekben célszerű használni, amelyekben nagy adatbázisok vannak, és rövidek a tranzakciók, amelyek csak kevés sort módosítanak, valamint ha kicsi az esélye annak, hogy két konkurens tranzakció ugyanazokat a sorokat módosítja, illetve ahol a hosszú tranzakciók elsősorban csak olvasási tranzakciók. Az Oracle csak akkor engedi egy sor módosítását egy sorbarendezhető tranzakciónak, ha el tudja dönteni, hogy az adott sor korábbi változásait olyan tranzakciók hajtották végre, amelyek még a sorbarendezhető tranzakció elindítása előtt véglegesítődtek. Amennyiben egy sorbarendezhető tranzakció megpróbál módosítani vagy törölni egy sort, amelyet egy olyan tranzakció változtatott meg, amely a sorba rendezhető tranzakció indításakor még nem véglegesítődött, az Oracle hibaüzenetet ad („Cannot serialize access for this transaction”).
- Csak olvasás mód: Nem része a szabványnak! A csak olvasást végző tranzakciók csak a tranzakció elindítása előtt véglegesített változásokat látják, és nem engednek meg INSERT, UPDATE és DELETE utasításokat.

Zárolások:

Bármelyik elkülönítési szintű tranzakció használja a sor szintű zárolást, ezáltal egy T tranzakciónak várnia kell, ha olyan sort próbál írni, amelyet egy még nem véglegesített konkurens tranzakció módosított. T megvárja, míg a másik tranzakció véglegesítődik vagy abortál, és felszabadítja a zárat. Ekkor:
- Ha abortál, akkor T végrehajthatja a sor módosítását, függetlenül az elkülönítési szintjétől.
- Ha a másik tranzakció véglegesítődik, akkor T csak akkor hajthatja végre a módosítást, ha az elkülönítési szintje az olvasásbiztos. Egy sorbarendezhető tranzakció ilyenkor abortál, és „Cannot serialize access” hibaüzenetet ad.

A zárakat az Oracle automatikusan kezeli, amikor SQL-utasításokat hajt végre. Mindig a legkevésbé szigorú zármódot alkalmazza, így biztosítja a legmagasabb fokú konkurenciát. Lehetőség van arra is, hogy a felhasználó kérjen zárat.

Egy tranzakcióban szereplő SQL-utasításnak adott zár a tranzakció befejeződéséig fennmarad (kétfázisú zárolás). Ezáltal a tranzakció egy utasítása által végrehajtott változtatások csak azon tranzakciók számára láthatók, amelyek azután indultak el, miután az első tranzakció véglegesítődött.

Az Oracle akkor szabadítja fel a zárakat,
- amikor a tranzakció véglegesítődik
- vagy abortál,
- illetve ha visszagörgetjük a tranzakciót egy mentési pontig (ekkor a mentési pont után kapott zárak szabadulnak fel).

Az Oracle a zárakat a következő általános kategóriákba sorolja:
- DML-zárak (adatzárak): Az adatok védelmére szolgálnak. Két szinten kaphatnak ilyen zárat a tranzakciók: sorok szintjén, vagy teljes táblák szintjén
- DDL-zárak (szótárzárak): A sémaobjektumok (pl. táblák) szerkezetének a védelmére valók.
- Belső zárak: A belső adatszerkezetek, adatfájlok védelmére szolgálnak, kezelésük teljesen automatikus.

Egy tranzakció tetszőleges számú sor szintű zárat fenntarthat. Sorok szintjén csak egyféle zármód létezik, a kizárólagos (írási – X).

A többváltozatú időbélyegzés és a sor szintű zárolás kombinációja azt eredményezi, hogy a tranzakciók csak akkor versengenek az adatokért, ha ugyanazokat a sorokat próbálják meg írni. Részletesebben:
- Adott sorok olvasója nem vár ugyanazon sorok írójára.

- Adott sorok írója nem vár ugyanazon sorok olvasójára, hacsak az olvasó nem a SELECT … FOR UPDATE utasítást használja, amely zárolja is a beolvasott sorokat.
- Adott sorok írója csak akkor vár egy másik tranzakcióra, ha az is ugyanazon sorokat próbálja meg írni ugyanabban az időben.

Egy tranzakció kizárólagos DML-zárat kap minden egyes sorra, amelyet az alábbi utasítások módosítanak: INSERT, UPDATE, DELETE és SELECT … FOR UPDATE.
Ha egy tranzakció egy tábla egy sorára zárat kap, akkor a teljes táblára is zárat kap, hogy elkerüljük az olyan DDL-utasításokat, amelyek felülírnák a tranzakció változtatásait, illetve hogy fenntartsuk a tranzakciónak a táblához való hozzáférés lehetőségét.
Egy tranzakció tábla szintű zárat kap, ha a táblát az alábbi utasítások módosítják: INSERT, UPDATE, DELETE, SELECT … FOR UPDATE és LOCK TABLE.
Táblák szintjén ötféle zármódot különböztetünk meg (a felsorolás sorrendjében egyre erősebbek):

- row share (RS) vagy subshare (SS),
- row exclusive (RX) vagy subexclusive (SX),
- share (S),
- share row exclusive (SRX) vagy share-subexclusive (SSX)
- exclusive (X).

SQL-utasítás	Zármód	RS	RX	S	SRX	X
SELECT...FROM	-	I	I	I	I	I
INSERT INTO nev	RX	I	I	N	N	N
UPDATE nev	RX	I*	I*	N	N	N
DELETE FROM nev	RX	I*	I*	N	N	N
SELECT .. FROM..FOR UPDATE	RS	I*	I*	I*	I*	N
LOCK TABLE nev IN ROW SHARE MODE	RS	I	I	I	I	N
LOCK TABLE nev IN ROW EXCLUSIVE MODE	RX	I	I	N	N	N
LOCK TABLE nev IN SHARE MODE	S	I	N	I	N	N
LOCK TABLE nev IN SHARE ROW EXCLUSIVE MODE	SRX	I	N	N	N	N
LOCK TABLE nev IN EXCLUSIVE MODE	X	N	N	N	N	N

Az RS zár azt jelzi, hogy a zárat fenntartó tranzakció sorokat zárolt a táblában, és később módosítani kívánja őket.
Az RX zár általában azt jelzi, hogy a zárat fenntartó tranzakció egy vagy több módosítást hajtott végre a táblában.
Az S zárat csak a LOCK TABLE utasítással lehet kérni. Más tranzakció nem módosíthatja a táblát. Ha több tranzakció egyidejűleg S zárat tart fenn ugyanazon a táblán, akkor egyikük sem módosíthatja a táblát (még akkor sem, ha az egyik a SELECT…FOR UPDATE utasítás hatására sor szintű zárakat tart fenn). Más szóval egy S zárat fenntartó tranzakció csak akkor módosíthatja a táblát, ha nincs másik olyan tranzakció, amely szintén S zárral rendelkezik ugyanezen a táblán.
Az SRX zárat csak a LOCK TABLE utasítással lehet kérni. Egy adott táblán egy időpillanatban csak egy tranzakció tarthat fenn SRX zárat. Más tekintetben megegyezik az S zárral.
Az X zárat csak a LOCK TABLE utasítással lehet kérni. Egy adott táblán egy időpillanatban csak egy tranzakció tarthat fenn X zárat, és ennek joga van a táblát kizárólagosan írni. Más tranzakciók ilyenkor csak olvashatják a táblát, de nem módosíthatják, és nem helyezhetnek el rajta zárakat.
A módosító DML-utasítások és a SELECT…FOR UPDATE utasítás az érintett sorokra kizárólagos sor szintű zárakat helyeznek, így más tranzakciók nem módosíthatják vagy törölhetik a zárolt sorokat, amíg a zárakat elhelyező tranzakció nem véglegesítődik vagy abortál.
Ha az utasítás alkérdést tartalmaz, az nem jár sor szintű zárolással.
Az alkérdések garantáltan konzisztensek a lekérdezés kezdetekor fennálló adatbázis-állapottal, és nem látják a tartalmazó módosító utasítás által véghezvitt változtatásokat.
Egy tranzakcióban lévő lekérdezés látja a tranzakció egy korábbi módosító utasítása által végrehajtott változtatásokat, de nem látja a tartalmazó tranzakciónál később elindult tranzakciók módosításait.

A módosító utasítás a sor szintű zárakon kívül a módosított sorokat tartalmazó táblákra is elhelyez egy-egy RX zárat. Ha a tartalmazó tranzakció már fenntart egy S, SRX vagy X zárat a kérdéses táblán, akkor nem kap külön RX zárat is, ha pedig RS zárat tartott fenn, akkor az felminősül RX zárrá.
Mivel sorok szintjén csak egyfajta zármód létezik (kizárólagos), nincs szükség felminősítésre.
Táblák szintjén az Oracle automatikusan felminősít egy zárat erősebb módúvá, amikor szükséges. Például egy SELECT … FOR UPDATE utasítás RS módban zárolja a táblát. Ha a tranzakció később módosít a zárolt sorok közül néhányat, az RS mód automatikusan felminősül RX módra.
Zárak kiterjesztésének nevezzük azt a folyamatot, amikor a szemcsézettség egy szintjén (pl. sorok szintjén) lévő zárakat az adatbázis-kezelő rendszer a szemcsézettség egy magasabb szintjére (pl. a tábla szintjére) emeli. Például ha a felhasználó sok sort zárol egy táblában, egyes rendszerek ezeket automatikusan kiterjesztik a teljes táblára. Ezáltal csökken a zárak száma, viszont nő a zárolt elemek zármódjának erőssége.
Az Oracle nem alkalmazza a zárkiterjesztést, mivel az megnöveli a holtpontok kialakulásának kockázatát.