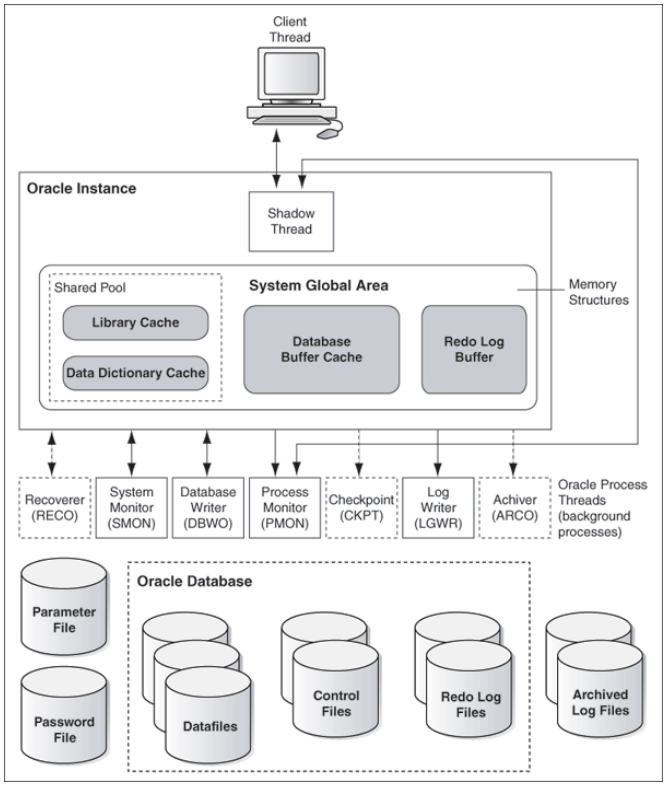
Önálló laboratórium 7. félév

Oracle adatbázisok optimalizálása
(Irodalom feldolgozás: Architektúra)



[1]

Nyárády Péter *QJA31E*

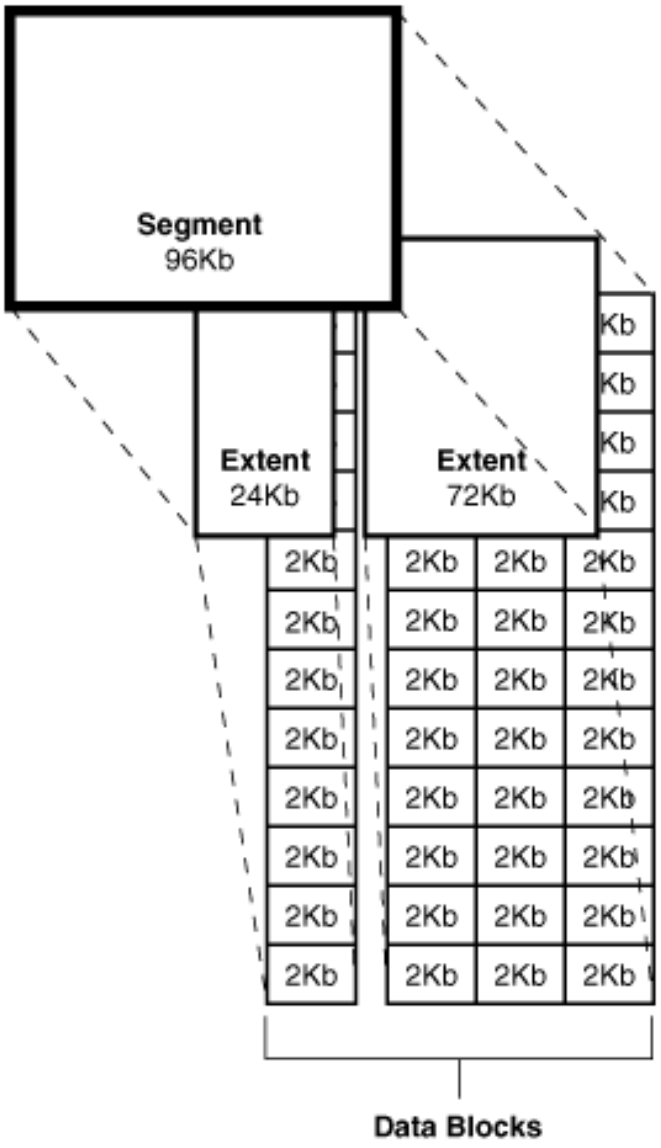
2008. május 6.

[1] `http://download.oracle.com/docs/cd/B28359_01/win.111/b32010/img/ntqrf003.gif`

Tartalomjegyzék

1. Logikai struktúra

Az Oracle Database logikai felépítése alapvetően három szintből áll. A legkisebb egységet az adatblokkok (**data blocks**) képezik, amik egy előre meghatározott, fix pár bytenyi részt jelentenek. A fizikailag folytonosan elhelyezkedő, valamilyen tárolási célból előre lefoglalt adatblokkok képezik a következő szintet, az **extent**eket. Ha egy extent betelik, de szükség van további szabad helyre, akkor új extentet kell foglalnunk. Az így keletkező, azonos célra foglalt, s azonos táblatéren (tablespace[2]) elhelyezkedő extentek alkotják a harmadik hierarchia szintet, a szegmenseket (**segment**):



[3]

[2] http://en.wikipedia.org/wiki/Tablespace

[3] http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/logical.htm#i10251

1.1. Adatblokkok

Az **adatblokk**ok méretét a `DB_BLOCK_SIZE` kezdeti paraméter beállításával adhatjuk meg, egy bizonyos felső korláttal, hogy elkerüljük a nagy adatblokkokból származó felesleges I/O-műveleteket.

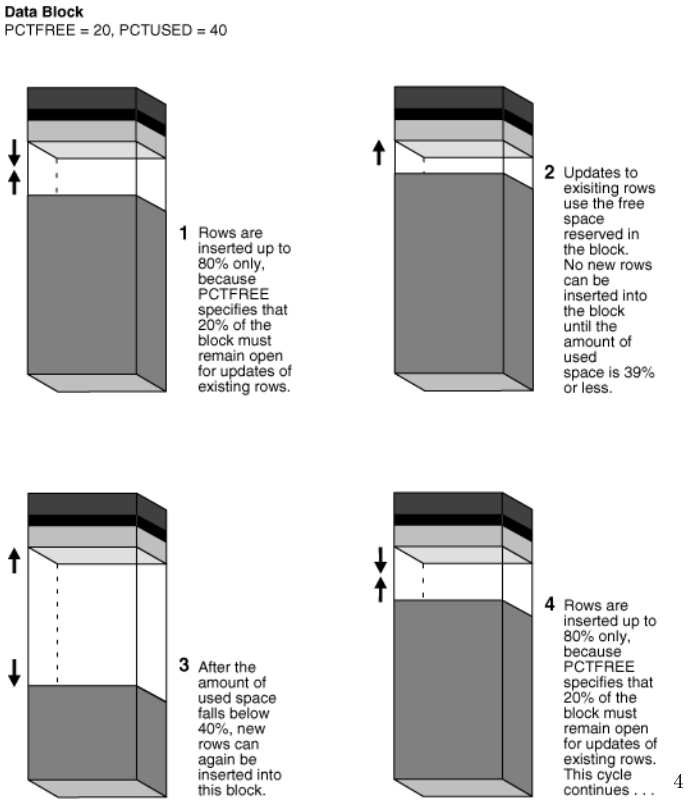
Az adatblokkra vonatkozó információkat tartalmazó overheadet (header, table directory, row directory) leszámítva használt és szabad területekre oszthatjuk a felhasználás szempontjából lényeges helyet. Használatban lévő területet felszabadítani nyilvánvalóan két utasítás fog: a `DELETE` és az olyan `UPDATE`, ami az addigi értéket egy kevesebb helyet foglalóra módosítja. Az így keletkező területek nem feltétlen fognak folytonosan elhelyezkedni, azonban mivel ezek az utasítások viszonylag gyakran előfordulhatnak, a töredezettség csökkentését csak akkor végzi el az Oracle Database, ha egy `INSERT` vagy `UPDATE` művelet olyan blokkot akar használni, ahol van elég hely számára, azonban nincs a blokkban olyan összefüggő hely, ami elég lenne számára.

Fontos még megemlíteni, hogyha egy tábla sorának az adatai nem férnek bele egy adatblokkba, akkor azt adatblokkok láncolásával illetve pointerekkel ugyan megoldja az adatbázis, de az ebből fakadó többszörös I/O műveletek csökkentik a teljesítményt.

Manuálisan felügyelt tablespaceknél két paramétert használhatunk az adatblokkok szabad területéhez történő hozzáférés vezérlésére:

- A `PCTFREE`-vel beállíthatjuk, hogy az adatblokk hány százalékát akarjuk update-ekre fenntartani. Azaz ha az adatblokkban felhasznált terület eléri a (100-`PCTFREE`)%-t, akkor kiszedjük az adatblokkot „free list”-ből (az a lista, amely tartalmazza, hogy mely adatblokkoknál alkalmazhatjuk az insert műveletet).
- A `PCTUSED` paraméter egy minimum értéket ad a használt területre, amíg nem kezdeményezhetünk új instertet. Azaz ha a felhasznált terület az itt megadott érték alá esik, akkor az adatblokk visszakerül a „free list”-be, s ismét lehet új sorokat is ide beilleszteni.

A két paraméter együttes működése:



[4]

1.2. Extentek

Az `extent`ek néhány folytonosan elhelyezkedő adatblokkot jelentenek, melyek lefoglalása egyidejűleg és előre történik. Például egy tábla létrehozásakor automatikusan lefoglal neki az Oracle Database egy kezdeti extentet, majd ha ez a terület kevésnek bizonyul, akkor további, legalább a kezdeti extent méretével egyező, vagy annál nagyobb extenteket. Ehhez a datafile-ok bitmapjeit használja, amik alapján megállapítható, hogy hol van megfelelő mennyiségű szomszédos szabad blokk.

Az extentek felszabadítása alapesetben csak akkor történik meg, ha töröljük az adott táblát. Kivételt képeznek ez alól a manuális műveletek, illetve a rollback szegmens, aminek méretét az Oracle Database periódikusan optimalizálja.

1.3. Szegmensek

Azok az extentek, melyek egy tablespacen belül logikailag összetartozó struktúrát képeznek, alkotnak egy **szegmens**et. Például egy táblához vagy indexhez tartozó extentek jelentik az adott táblához illetve indexhez tartozó szegmenset.

A hatékonyság szempontjából fontos tárolási paramétereket mind a táblák, mind az indexek létrehozásakor illetve módosításukkor beállíthatjuk. Egy indexnek azonban nem

[4] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/logical.htm#i19214`

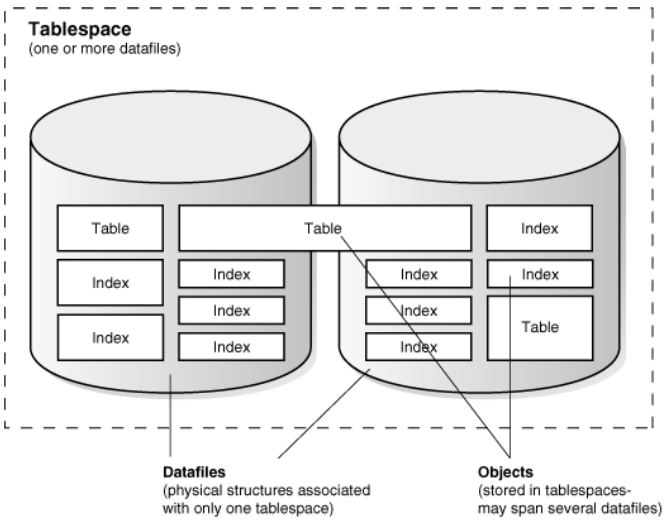
feltétlen kell az általa hivatkozott táblával egy tablespaceben lennie... így a tárolási paraméterek beállításánál akár egy másik tablespacet is választhatunk.

Léteznek még az ún. ideiglenes szegmensek is, melyeket az Oracle Database automatikusan lefoglal, ha valamely művelet végrehajtása során szüksége van a memóriát meghaladó területekre, vagy ha egy tranzakció során ideiglenes táblá(ka)t használunk. Az ideiglenes szegmensek tárolására használt tablespaceket a felhasználók létrehozásánál illetve módosításánál a `TEMPORARY TABLESPACE` paraméterrel állíthatjuk be. Ez alapesetben a `SYSTEM` tablespacet jelenti, azonban ajánlott legalább egy ilyen tablespacet létrehozni, hogy elkerüljük a `SYSTEM` nagymértékű tördelődését.

Fontos még megjegyezni, hogy a 9i verziótól ugyan már automatikusan működik a tranzakciók visszagörgetése - amivel egy inkonzisztens állapotból térhetünk vissza egy konzisztensbe -, azonban egy rosszul működő tranzakció túlságosan nagy részét foglalhatja el az visszagörgetéshez használt undo tablespacenek. Ezért lehetőség van a felhasználók egy bizonyos csoportjának, vagy akár egy adott felhasználónak is közvetlen korlátozni az undo területét. Erre szolgál az `UNDO_POOL` paraméter, melynek default értéke természetesen `UNLIMITED`. Így ha egy csoport egy felhasználója megtölti a csoport számára kijelölt területet, akkor nem hajthat végre további update-eket mindaddig, amíg egy másik csoportbéli felhasználó tranzakciója be nem fejeződik, s ezáltal területek szabadulnak fel.

2. Táblaterek és Adatfájlok

A logikai adattárolás legnagyobb egységei a táblaterek (**tablespace**s), míg a fizikai tárolás adatfájlok (**datafile**s) formájában történik:



[5]

Az Oracle adatbázis tartalmaz legalább két tablespacet, a `SYSTEM`-t és a `SYSAUX`-t, illetőleg egy harmadik, `TEMP` nevű tablespacet opcionálisan. A tablespacek adatait fizikailag egy

[5] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/physical.htm#i3562`

vagy több datafile tárolja. A fájlrendszer további részeit képezik még a visszagörgetéshez használt **Redo log** fájlok, illetve az adatbázis indításához és működéséhez elengedhetetlen **Control** fájlok. Az adatbázisunk méretét háromféleképpen növelhetjük:

- új tablespacet hozunk létre
- egy már létező tablespacehez új datafilet adunk hozzá
- egy már létező datafile méretet növeljük (engedélyezhetjük, hogy az adatbázis ezt dinamikus hajtsa végre, amint szüksége van több területre)

2.1. Táblaterek

A tablespacek, ahogyan azt már tárgyaltuk, szegmensekből épülnek fel, melyeket extentek alkotnak, amiket pedig összefüggő adatblokkok képeznek.

2.1.1. Terület-menedzselésük

A tablespacek területének menedzselésére kétféle mód kínálkozik:

- A **lokálisan vezérelt** tablespacek egy bitmapben tartják számon a datafileban használt és szabad területeket. Előnyük, hogy ezáltal nem kell foglalkozni a szomszédos szabad területek összeolvasztásával, illetve nincs szükség rekurzív területvezérlési műveletekre (pl. ha felszabadítunk helyet egy extentben, akkor az nem idéz elő semmilyen más műveletet a data dictionaryben vagy rollback szegmensben). A `SEGMENT SPACE MANAGENT` paraméterrel beállíthatjuk, hogy `AUTO` (alapértelmezett) vagy `MANUAL` módban szeretnénk-e kezelni a szegmensen belüli területeket. Előbbi esetben az adatbázis a bitmap alapján automatikusan kezeli a szabad területeket, míg utóbbiban egy (az előző részben már említett) free list-ben tartjuk nyilván az olyan adatblokkokat, ahova van lehetőség új sor beillesztésére.
- A **könyvtár vezérelt** mód az Oracle 9i verziójától elérhető. Ebben az esetben bitmapek helyett az adatbázis a `SYSTEM` tablespacen belül kezel egy data dictionaryt, melyet minden extentfoglalás és -felszabadítás esetén frissít, s ezekről a frissítésekről a rollback szegmensekben tárol információt.

Lehetőség van a `SYSTEM` tablespace kivételével (mivel annak data dictionaryjére mindig szüksége van egy futó adatbázisnak) bármely tablespacet online módból **offline mód**ba átállítani. Erre karbantartáskor vagy biztonsági mentés létrehozásakor lehet szükség, de bizonyos hibák esetén az adatbázis automatikusan is offline módba állíthat egy tablespacet. Ilyenkor természetesen semmilyen olyan SQL utasítás nem hajtható végre, ami az adott tablespacen belüli objektumokra hivatkozik.

Biztonsági okokból adott tablespaceket lehet csak olvasásra (**read-only**) is engedélyezni.

2.1.2. Ismertetésük

Bigfile táblaterek. A 64-bites rendszerek képességeinek kiaknázása érdekében lehetőség van ún. „bigfile tablespace"-k létrehozására. Ez lényegében annyit jelent, hogy az adott tablespaceünket mindössze egyetlen datafile alkotja, így a szokványos „smallfile tablespace"-kkel szemben jóval nagyobb datafilet használthatunk – egészen pontosan mivel a smallfile tablespacek maximum 1024 datafilet tartalmazhatnak, ezért a bigfile tablespaceünk egyetlen datafileja akár 1024szer lehet nagyobb. A datafilek számára is van egy felső korlát (64K), ezért mivel a bigfile tablespacek tablespacenként csak egy datafilet tartalmaznak, lehetőségünk van hatalmas adatbázisok tárolására egészen 8 exabyte (10^{18}) méretig, illetve a kevesebb datafile miatt könnyebben kezelhető lesz az adatbázisunk. Cserébe viszont a bigfile tablespacek (néhány kivételtől eltekintve) csak lokálisan vezérelt (locally managed), automatikus szegmens-foglalási tablespaceként működhetnek.

SYSTEM táblatér. A **SYSTEM tablespace** automatikusan létrejön, amikor az adatbázist létrehozzuk. Alapvetően könyvtár vezérelt (dictionary managed), de természetesen ezt átállíthatjuk lokális vezérlésre is, azonban ezáltal a későbbiekben nem tudunk könyvtár vezérelt tablespaceket létrehozni, s a létezőket is csak olvasni tudjuk. A `SYSTEM` tablespaceben tároljuk a data dictionary táblákat, illetve a tárolt PL/SQL eljárásokat.

Ha a táblatér terület-menedzselését lokális vezérlésre állítottuk, mindenképpen definiálnunk kell az ideiglenes adatok tárolására egy **ideiglenes tablespace**t, de ajánlott könyvtár vezérelt módban sem a `SYSTEM`-be szemetelni. A könnyebb kezelhetőség és a versenyhelyzet elkerülése érdekében természetesen ajánlott a felhasználók adatainak számára is külön tablespace(ke)t létrehozni.

SYSAUX táblatér. A **SYSAUX tablespace** a `SYSTEM`-t kiegészítő tablespace. Sok adatbázis komponens használja alapértelmezett tárolási helyként, illetve az összes, nem a `SYSTEM` tablespaceben lévő metaadatot itt tároljuk.

UNDO táblatér. Az **UNDO tablespace**k csak és kizárólag undo infomárciókat tartalmaznak, s csak automatikus undo menedzselt módban (alapértelmezett) alkalmazhatóak. Lehet ugyan több ilyen tablespaceünk, azonban használatban egyszerre csak egy lehet. Az undo tablespacek automatikusan létrejönnek, mindig lokálisan vezéreltek, s néhány ritkán előforduló körülménytől eltekintve a tranzakciókat első DML eljáráshívásuk után hozzárendeljük egy undo szegmenshez.

TEMPORARY táblatér. A `CREATE TEMPORARY TABLESPACE` utasítással ideiglenes tárolási célokra használt tablespaceket hozhatunk létre. Ezek nagyban növelhetik a rendezést használó utasítások végrehajtásának hatékonyságát. Természetesen egy temporary tablespace nem használható állandó sémaobjektumok tárolására.

2.1.3. Tablespace Repository

Lehetőség van tablespaceink adatbázisok közti szállítására is. Ehhez nyújt segítséget a **tablespace repository**, amelyben a tablespacek egy halmazának szállításához szükséges fájlokat tároljuk. Nem árt ügyelni arra, hogyha egy könyvtár vezérelt tablespacet költöztetünk át egy lokálisan vezérelt `SYSTEM` tablespaceszel rendelkező adatbázisba, akkor azt ott csak olvasni tudjuk majd, írni nem.

2.2. Adatfájlok

Egy tablespacet ugyebár fizikailag **datafile**(ok) alkotnak. Egy datafile csak egy tablespacehez és adatbázishoz tartozhat. Létrehozásakor az általa kijelölt terület formatálódik, s az adatbázis lefoglalja a későbbiekben szükséges extentek létrehozására. Az ideiglenes táblaterek ideiglenes datafilejai kevesebb tulajdonsággal bírnak szokványos társaiknál: például nem logolnak és nem lehet őket read-only módra állítani.

2.3. Control fájlok

Az adatbázis indításához és működéséhez az ún. **control file**okat használja. Ezeknek mindig írásra alkalmasnak kell lenniük, hisz az adatbázis működés közben folyamatosan frissíti bennük a fizikai architektúra és a visszagörgetéshez szükséges redo logok leírását. Ebből kifolyólag ha a control fileok nem elérhetőek vagy sérülnek, az adatbázis nem fog helyesen működni, ezért erősen ajánlott több azonos control filet egyszerre, különböző fizikai lemezeken tárolni és frissíteni.

3. Tranzakciókezelés

A **tranzakció** legalább egy SQL utasítást magába foglaló műveletsorozatot jelent. Elemi műveletként értelmezzük, ami annyit tesz, hogy a tranzakciót alkotó SQL utasítások mindegyike végrehajtódik (**commit**álódik), vagy egyik sem (**roll back** - visszagörgetés). Egy tranzakció jóváhagyása vagy visszagörgetése explicit módon a `COMMIT` vagy `ROLLBACK` utasításokkal, implicit módon az alkalmazásból való kilépéssel vagy egy DDL utasítás végrehajtásával történhet.

Utasítás szintű visszagörgetésről akkor beszélünk, ha egy SQL utasítás valamilyen hiba folytán nem tud lefutni. Ilyen hiba lehet például, ha már létező elsődleges kulcsot akarunk létrehozni, ha két SQL utasítás között versenyhelyzet áll elő, vagy egyszerűen csak szintaktikailag nem helyes az utasítás. A hibás utasítás nem okozza az egész tranzakció visszagörgetését, pusztán csak a saját feladatát nem tudja végrehajtani.

3.1. Resumable space allocation

Nagy adatbázis műveletek számára előnyös lehet, hogyha egy tranzakció helyfoglalási problémák következtében nem tud tovább futni, akkor nem visszagörgetés következik, hanem

lehetősége van az adatbázis adminisztrátornak a hibát elhárítania, majd a tranzakciók zökkenőmentesen futhatnak tovább.

3.2. Tranzakciók működése

Egy tranzakció futásának kezdetén a visszagörgetéshez szükséges információk tárolására hozzárendelünk a tranzakcióhoz egy undo tablespacet. A **tranzakció végén** a következők állhatnak elő:

- a felhasználó `COMMIT` vagy `ROLLBACK` utasítást ad ki.
- a felhasználó valamilyen DDL utasítást (`CREATE`, `DROP`, `RENAME` vagy `ALTER`) ad ki. Ebben az esetben az adatbázis először commitálja a tranzakciót, majd csak azután hajtja végre a DDL utasítást, mint egy egy utasításból álló külön tranzakciót.
- a felhasználó kapcsolatot bont az adatbázissal. Ekkor automatikusan commit hajtódik végre.
- a felhasználói processz abnormálisan áll le. Ekkor automatikusan rollback hajtódik végre.

3.2.1. Commit

Commitálni egy tranzakciót annyit tesz, mint véglegesíteni az adatbázisban az SQL utasítások által végzett műveleteket. A commit előtt az adatbázis a következő műveleteket hajtja végre: undo információkat generál, melyek a az SQL utasítások által módosított adatok módosítás előtti változatát tárolják. Illetve az SGA redo log bufferjébe redo log bejegyzéseket tesz, melyek az adatblokkokon illetve roll back blokkokon történt változásokat rögzítik. Előrefordulhat, hogy ezek az adatok a tranzakció commitálását megelőzően már a lemezre kerülnek. A **commit után** a következők mennek végbe:

1. az undo tablespacehez tartozó belső tranzakciós táblában feljegyezzük, hogy a tranzakciót commitálták, s beírjuk a tranzakció egyedi `SCN` (system change number) számát.
2. az SGA redo log bufferjének redo log fájljába bejegyzéseket tesz az `LGWR` (log writer process), s szintén beírja a tranzakció `SCN` számát, ami egyben a tranzakció commitálását is jelenti.
3. az adatbázis elengedi a tranzakció által használt zárakat.
4. a tranzakciót késznek jelöli.

3.2.2. Roll back

Egy tranzakció visszagörgetése (**roll back**) annyit jelent, hogy egy lezáratlan tranzakció minden addigi adatmódosítását visszaállítjuk az adatok eredeti értékére. A régi adatokat az undo tablespaceben, a változtatásokat a redo logban tároljuk. Lehetőség van savepointok definiálására, amikkel egy nagyobb tranzakciót kisebb részegységekre bonthatunk, így ha valahol hiba történik, akkor a visszagörgetés nem teljesen a tranzakció elejéig történik, hanem a legutolsó savepointig (már ha volt addig). A **visszagörgetés** a következő lépésekből áll:

1. az adatbázis az undo tablespace alapján minden addigi, a tranzakció SQL utasításai által végrehajtott módosítást visszacsinál. Savepoint esetén természetesen csak az adott savepointig kell visszacsinálni mindent, vagyis csak a savepoint után szereplő SQL utasítások által módosított adatokat kell a régi értékükre visszaállítani.

2. Ha volt savepointunk, akkor azt az adott savepointot természetesen megőrizzük, de minden utána megállapított savepointot törlünk.

3. Elengedjük a tranzakció zárjait. Természetesen ha savepointot alkalmaztunk, akkor itt is csak a savepoint után igényelt zárakat engedjük el. Azok az adatok, melyek a savepoint előtt már zárolva voltak, természetesen továbbra is zárolva maradnak.

3.2.3. Kétfázisú commit

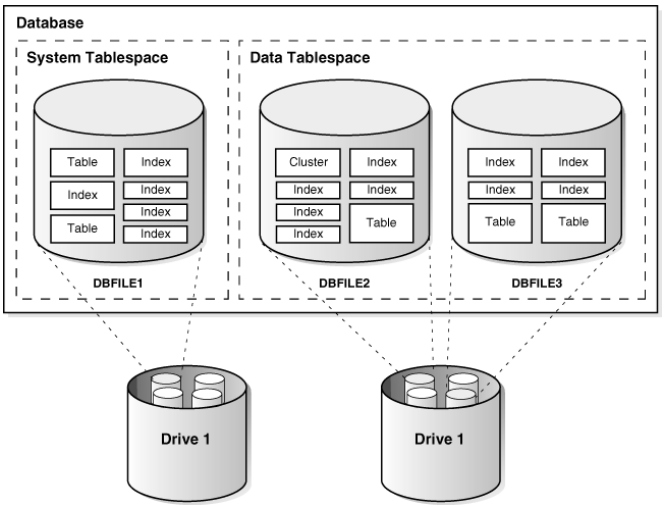
Egy elosztott adatbázisban hálózati vagy rendszer hibák esetén is gondoskodni kell a hálózat feletti tranzakcióvezérlésről és az adatok konzisztenciájáról. Elosztott tranzakciónak nevezünk egy tranzakciót, ha tartalmaz legalább egy olyan utasítást, ami módosít legalább két különböző hálózati végpontokon lévő adatbázisokat. A kétfázisú elv biztosítja, hogy elosztott tranzakció minden adatbázisa konzisztens marad, vagyis az összes adatbázis vagy commitálja a tranzakciót, vagy visszagörgeti annak addigi hatásait.

3.3. Autonóm tranzakciók

Léteznek ún. **autonóm tranzakciók**, melyeket más tranzakciók hívnak meg, de azoktól teljesen függetlenül futnak le. A hívást követően a meghívó tranzakció felfüggesztődik, a meghívott tranzakció teljesen független zárakkal és utasításokkal lefut, majd függetlenül attól, hogy committáltunk vagy visszagörgettük a tranzakciót, folytatódik a meghívó tranzakció futása. Ezáltal természetesen létre lehet hozni holtpontot, amit az adatbázis hibaüzenettel ugyan jelez, de ezekért a holtpontokért teljes mértékben az alkalmazás fejlesztője felel. Egy autonóm tranzakció is meghívhat más autonóm tranzakciókat, nincs semmilyen korlát a hívások mélységére. Lehetőség van azonban nem autonóm tranzakciók meghívására is. Ekkor a meghívott tranzació örökli a hívó tranzakció környezetét.

4. Séma objektumok

Sémának (**Schema**) nevezzük az egy felhasználóhoz tartozó logikai adatstruktúrák (séma objektumok) összességét. Ezek az objektumok nem feleltethetőek meg egy az egyben fizikai diszken tárolt fájloknak. Logikailag egy objektum egy tablespacen belül helyezkedik el, fizikailag azonban tárolódhat akár több datafileban is. A sémák és tablespacek között nincs semmilyen összefüggés: egy tablespace tartalmazhat objektumokat több különböző sémából, illetve egy séma objektumai is tárolódhatnak különböző tablespacekben.



[6]

Az alábbi objektumok tartoznak a séma objektumok közé:

- klaszterek
- kényszerek
- adatbázis
- hivatkozások
- adatbázis triggerek
- dimenziók
- külső eljáráskönyvtárak
- indexek és indextípusok
- Java osztályok
- materializált nézetek és a hozzájuk tartozó logok

[6] http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/schema.htm#i5716

- objektum táblák, objektum típusok és objektum nézetek
- operátorok
- szekvenciák
- tárolt függvények, eljárások és csomagok
- szinonimák
- táblák és indexelt táblák
- nézetek

Nem tartoznak séma alá a következő objektumok:

- kontextusok
- könyvtárak
- paraméter fájlok (`PFILE`s)
- szerver paraméter fájlok (`SPFILE`s)
- profilok
- szerepek
- rollback szegmensek
- tablespacek
- felhasználók

4.1. Táblák

A táblák jelentik az Oracle adatbázisban az alapvető adattárolási egységet. Alapvető tulajdonságainak részletes ismertetése alább[7].

4.1.1. Tárolásuk

A tábla létrehozásakor automatikusan lefoglalunk neki egy adatszegmenst a tablespacen belül. Ennek vezérlésére használhatjuk a már ismertetett `PCTFREE` és `PCTUSED` paramétereket, vagy beállíthatjuk az adatszegmens tárolási paramétereit. Klaszter használata esetén nincs lehetőség külön táblánként tárolási paramétereket állítani, hanem csak egységesen, az klaszter összes táblájára vonatkozóan állíthatjuk be őket.

[7] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/schema.htm#i5663`

4.1.2. Sorok formátuma és mérete

Ha egy sor 256 oszlopnál kevesebbet tartalmaz, s adatai elférnek egy adatblokkban, akkor az adatbázis a sort egy adatblokkon belül, egy darabban tárolja. Azonban ha a sor adatai méretüknél fogva nem tehetőek be egy adatblokkba, vagy a sor legalább 256 oszlopból áll, akkor több adatblokkra van szükség a tároláshoz. Ezt láncolásnak nevezzük – a további adatblokkok `ROWID`-jét a sor headerjében tároljuk (ábra[8]).

4.1.3. ROWID

A ROWID azonosítja az egyes sordarabokat helyük vagy címük alapján. Érdemes hivatkozni rájuk SQL utasításokban, hiszen értékük sosem változik meg (ez alól természetesen kivételt képez, hogyha az adatot valamilyen oknál fogva másik adatblokkba kell átvinnünk).

4.1.4. Oszlopsorrend

Tárolási szempontból általában megegyezik a tábla létrehozásakor megadott oszlopsorrenddel (kivétel pl. ha `LONG` típust használunk, ugyanis azt az adatbázis mindig utolsó oszlopként tárolja – illetve újonnan felvett oszlopok is mindig hátra kerülnek), ezért célszerű a gyakran `NULL` értéket felvevő oszlopokat a sorrendben hátulra tenni, mivel így (ha nincs LONG típusunk deklarálva) jelentős mennyiségű helyet takaríthatunk meg.

4.1.5. Táblák tömörítése

Az egy blokkon belül többször előforduló adatokat nem tároljuk el külön-külön többször, hanem csak egyszer a blokk elején (egy ún. szimbólum táblában), s a későbbi előfordulások alkalmával csak egy hivatkozást illesztünk be a szimbólum tábla megfelelő elemére. A tömörítés nem jelent semmilyen funkcionalitásbeli hátrányt, s a `LOB` (Long OBject) típusokon kívül minden más típussal működik. A törlés (`DELETE`) és beillesztés (`INSERT`) sem kerül több időbe, mint a tömörítés nélkül tárolt tábláknál. Egyedül az adatok frissítése (`UPDATE`) esetén fordulhat elő, hogy a végrehajtás lassabb lesz. Célszerű tehát a tömörítést minden csak olvasható, illetve ritkán változtatandó táblánál használni (mivel csökkenti a használt merevlemezt, memóriát -buffer cache- és gyorsítja a lekérdezés végrehajtást - cserébe azonban csak egy csekély CPU-val fizetünk).

4.1.6. Partícionált táblák

Partícionált táblák által lehetőség van az adatok kisebb részekre bontására, s ezáltal könnyebb managelésére.

[8] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/schema.htm#i20134`

4.1.7. Nested táblák

Egymásba ágyazott táblák: egy tábla oszloptípusának megadhatunk egy másik táblát is, ezáltal egymásba ágyazott táblákat is létrehozhatunk. A beágyazott táblát az Oracle adatbázis egy külön táblában tárolja.

4.1.8. Ideiglenes táblák

Az ideiglenes táblák (Temporary tables) nem permanens táblák. Tranzakcióhoz vagy sessionhöz tartozhatnak, így csak azok élettartama alatt léteznek. Mivel az adott tranzakció, ill. session kizárólagos joggal rendelkezik felettük, így zárkezelésre sincs szükség. Ellentétben a permanens táblákkal, számukra csak az első `INSERT` utasítás kiadását követően foglalunk szegmenst.

4.1.9. Külső táblák

Külső táblák (External tables) által lehetőség van külső adatbázisokban található adatokhoz történő hozzáférésre úgy, mintha azok a saját adatbázisunk egy táblájában lennének tárolva. A külső táblák nem tartalmazzák, hogy a külső forrásnál hogyan vannak az adatok tárolva (az adatok transzformálását az Access Driver végzi), csupán azok megjelenítéséért felelősek. Természetesen csak olvashatóak, nem rendelhetünk hozzájuk indexeket, s a virtuális oszlopok sem támogatottak.

ETL: `http://en.wikipedia.org/wiki/Extract,_transform,_load`

4.2. Nézetek

A nézetek (**view**s) olyan virtuális táblák, melyek adataikat egy vagy több fizikai táblából, vagy más nézetekből veszik. Fő feladatuk tehát az adatoknak egy előre megszabott formában történő megjelenítése, s ezáltal bizonyos adatok elrejtése. Működésük és a rajtuk végrehajtható műveletek többé-kevésbé megegyeznek a tábláknál megszokottakkal.

4.2.1. Tárolás

Mivel a nézetek csak egy lekérdezés által definiáltak, s nem tartalmaznak ténylegesen adatokat, ezért tárolási helyet sem kell biztosítani a számukra. Csupán magát a lekérdezést tároljuk el a data dictionaryben.

4.2.2. Felhasználás

- tábla bizonyos sorainak vagy oszlopainak rejtése
- adat komplexitás elrejtése
- egyszerűbb lekérdezések megfogalmazása

- többféle adat megjelenítés
- alkalmazások függetlenítése az alap táblákon végrehajtott változtatásoktól

4.2.3. Működés

1. A nézetre történő hivatkozás helyére behelyettesítődik a nézet által definiált lekérdezés.
2. Szintaktikai elemzést hajt végre az így kialakult utasításon.
3. Végrehajtja az utasítást.

4.2.4. Updatable Join Views

Két vagy több táblából vagy nézetből származtatott nézet, melyen engedélyezettek az `UPDATE`, `INSERT` és `DELETE` műveletek. Az adatszótár (data dictionary) `ALL_UPDATABLE_COLUMNS`, `DBA_UPDATABLE_COLUMNS` és `USER_UPDATABLE_COLUMNS` nézetei tartalmazzák, hogy a nézet mely oszlopai frissíthetőek (`UPDATE`elhetőek). Ennek feltétele, hogy a nézet ne tartalmazza a következő struktúrák egyikét sem:

- halmaz operátorok
- `DISTINCT` operátor
- aggregátumok és analitikus funkciók
- `GROUP BY`, `ORDER BY`, `CONNECT BY` és `START WITH` klauzulák
- kollekciós kifejezés vagy allekérdezés a `SELECT` után
- illesztések (néhány kivételtől eltekintve)

A nem frissíthető nézeteket `INSTEAD OF` triggerek használatával módosíthatjuk.

4.2.5. Object Views

Objektum nézetekből kinyerhetjük, frissíthetjük, beilleszthetjük és törölhetjük az adatokat pontosan úgy, mintha objektum típusként lennének tárolva.

4.2.6. Inline Views

Nem séma objektum, csak egy allekérdezés egy aliasszal.

4.3. Materializált Nézetek

A materializált nézeteket (**materialized view**s) adatok összegzésére, számítására, replikázására és szétosztására használhatjuk. Ebből kifolyólag főként adattárházaknál (data warehouse), döntéstámogató rendszereknél és elosztott vagy mobil számításoknál használjuk őket. Az optimalizáló (optimizer) automatikusan felismeri, hogy mikor lehet egy kérést materializált nézet segítségével kielégíteni, s automatikusan behelyettesíti azt a lekérdezésbe. Így nem szükséges közvetlen a táblákból vagy nézetekből kinyerni a kívánt adatokat, amivel növelhetjük a teljesítményt. Néhány szempontból tehát a materializált nézetek hasonlítanak az indexekre:

- tárhelyet fogyasztanak
- ha változnak az adatok a master táblájában, akkor frissíteni kell őket
- lekérdezések behelyettesítése által növelik az SQL végrehajtások teljesítményét
- felhasználók és alkalmazások szempontjából átlátszóak

Ellentétben azonban az indexekkel, a materializált nézetekhez közvetlenül is hozzáférhetünk egy `SELECT` utasítással, illetve a frissítés típusának függvényében akár közvetlenül is alkalmazhatunk rajtuk `INSERT`, `UPDATE` vagy `DELETE` műveleteket.

4.3.1. Nézeteken alkalmazott kényszerek

Multidimenziós adatok felismerhetősége érdekében lehetőség nézeteken is bizonyos kényszerek megfogalmazására: primary key, unique és referential integrity constrainteket adhatunk meg.

4.3.2. Frissítés

Kétféle frissítési eljárás használtható: inkrementális (fast refesh) és teljes. Előbbi esetén a materialized view log vagy a direct loader log tartalmazza a változtatásokat. A frissítés maga történhet azonnal vagy előre meghatározott időközönként.

4.3.3. Materialized View Logs

Séma objektumok, melyek az egyes master táblákban történt változásokat jegyzik a materializált nézet számára, ha az inkrementális frissítést használ. Helye a master tábla sémájában van.

Inkrementális frissítés használata esetén explicite létre kell hoznunk őket, mielőtt a hozzá tartozó materializált nézetet létrehoznánk. Példa: `http://download.oracle.com/docs/cd/B28359_01/server.111/b28313/basicmv.htm#i1006519`

4.4. Dimenziók

A dimenziók (**dimension**s) hierarchikus (gyermek/szülő) viszonyt határoznak meg oszloppárok vagy oszlophalmazok között. Minden gyermek pontosan egy szülőhöz van társítva. Mivel a dimenzió csak logikai kapcsolatok gyűjteménye, nincs tárolási terület hozzárendelve. A dimenziót létrehozó `CREATE DIMENSION` parancs meghatározza a hierarchia szinteket (LEVEL), a köztük lévő viszonyt (`HIERARCHY`), illetve opcionálisan az `ATTRIBUTE` ágban kiegészítő oszlopot vagy oszlop halmazt az adott szinthez. Az oszlopok származhatnak egy táblából (denormalized) vagy több táblából (normalized). Utóbbi esetén illesztést kell alkalmazni a `HIERARCHY` ágban.

A dimenzióknak főként BI-os alkalmazásoknál lehet haszna. Elsősorban az optimizer munkáját hivatottak segíteni. Vegyük a következő példát[9]:

```
CREATE DIMENSION time_dim
LEVEL time_key IS time.time_key
LEVEL month IS time.month
LEVEL quarter IS time.quarter
LEVEL year IS time.year
HIERARCHY calendar_rollup (
          time_key CHILD OF
          month    CHILD OF

          quarter CHILD OF
          year
)
ATTRIBUTE time_key determines (day_of_week, holiday)
ATTRIBUTE month    determines (month_name);
```

Ha van egy hónapos bontásban szereplő materializált nézetünk, azonban a lekérdezésben éves bontás szerepel, akkor az optimizier ezen dimenzió segítségével az éves bontást tartalmazó lekérdezésbe is be tudja helyettesíteni a materializált nézetet, s ezzel gyorsítja azt. Ehhez azonban először a `QUERY_REWRITE_INTEGRITY` paraméter segítségével engedélyeznünk kell a lekérdezések felülírását:

```
ALTER SESSION SET query_rewrite_integrity = TRUSTED;
```

4.5. Szekvencia Generátor

A szekvencia generátor (**sequence generator**) segítségével számok szekvenciális sorozatát állíthatjuk elő (pl. egyedi elsődleges kulcsoknak). Segítségével többfelhasználós rendszerek esetén elkerülhetünk felesleges zárolásokat (pl. egyszerre két felhasználó is tud adatokat bevinni egy táblába). Méretük maximum 38 számjegy lehet, definíciójukat az adatszótárban (data dictionary) tároljuk. A számok generálása független az egyes tábláktól, így egy szekvencia generátort akár több táblánál is használhatunk.

[9] `http://www.oracle.com/technology/oramag/oracle/03-sep/o53business.html`

4.6. Szinonimák

A szinonima (**synonym**) egy alternatív név bármely táblára, nézetre, materializált nézetre, szekvenciára, eljárásra, függvényre, csomagra, típusra, Java osztályra, felhasználó által definiált objektum-típusra vagy egy másik szinonimára. Mivel adatot a szinonima sem tárol, így számára sem szükséges területeket lefoglalni...a definícióját az adatszótárban (data dictionary) tároljuk. Használatának két fő célja a biztonság és a kényelem, valamint az, hogyha a master táblájának megváltozik a neve, akkor csak magát a szinonimát kell átírni, s utána a szinonimát használó alkalmazások probléma nélkül futnak majd. A szinonima lehet public vagy private. Előbbi minden felhasználó számára elérhető, míg utóbbi egy bizonyos felhasználó sémájában található, s mások számára a hozzáférhetőséget ő határozhatja meg.

4.7. Indexek

Az indexek (**index**es) opcionális, táblákhoz és klaszterekhez rendelhető struktúrák. Indexeket lehet egy vagy több oszlophoz rendelni, melyek segítségével egy SQL utasítás esetén gyorsabban megtalálja a keresett információt az adatbázis - ezáltal a helyesen használt indexek jelentik az I/O műveletek csökkentésének fő forrását. A következő index-sémák használhatóak: B-fa index, B-fa klaszter index, Hash klaszter index, Reverse key (inverz kulcs) index, Bitmap index, Bitmap join index. Az indexek használata automatikus, csak a létrehozásukkal és az esetleges törlésükkel kell foglalkozni. Továbbá az optimalizáló (optimizer) akár felhasználhat egy már létező indexet egy új index létrehozására, ami jóval gyorsabb index-építést eredményezhet.

4.7.1. Unique/Nonunique

Az egyedi (unique) indexek garantálják, hogy a hivatkozott oszlopban nem szerepelhet duplikált érték. Ajánlott ezt explicit megadni mindig a `CREATE UNIQUE INDEX` használatával, ugyanis nem garantált, hogy egy primary key vagy unique constraint automatikusan létrehoz egy új indexet, s ha létre is hozott, arra sincs garancia, hogy az az index unique lesz.

4.7.2. Visible/Invisible

A láthatatlan (invisible) indexeket csak a DML műveletek kezelik, s nem használhatja őket az optimalizáló (optimizer).

4.7.3. Összetett indexek (Composite/Concatenated Indexes)

Több oszlopra definiált indexek. Az oszlopok megadásának sorrendjét nem befolyásolja az, hogy az adott oszlopok milyen sorrendben szerepelnek a táblában, azonban maga a sorrend nagyon is fontos. Célszerű a gyakran hivatkozott oszlopokat a sorrendben előre helyezni. Összetett indexekkel lényeges javulást akkor tudunk elérni, ha sikerül olyan oszlopokat

találni, melyek (mind, vagy legalább nagyobb részük) gyakran szerepel(nek) egyszerre a `WHERE` feltételében. Maximum 32 (bitmap indexek esetén 30) oszlopot adhatunk meg.

4.7.4. Indexek és kulcsok (Keys)

Két különböző dologról van szó. Az indexek az adatbázisban tárolt, felhasználók által létrehozott struktúrák, melyek célja az adatok elérésének meggyorsítása. A kulcsok azonban szigorúan csak logikailag léteznek, s az integritás kényszerekért felelnek.

4.7.5. NULL értékek

Különböző értékként vannak számon tartva, kivéve, hogyha az index legalább két sorában lévő nem `NULL` érték azonos. Tehát az `UNIQUE` indexekkel - mivel ott minden nem `NULL` érték különböző - lehet biztosítani, hogy a `NULL` értéket tartalmazó sorok is különbözőként legyenek kezelve (kivétel, ha minden érték `NULL`).

4.7.6. Függvény-alapú indexek (Function-Based Indexes)

Létrehozhatunk indexeket olyan függvényekre és kifejezésekre is, melyek tartalmaznak legalább egy oszlopot egy indexelt táblából. Ezek a függvény-alapú indexek kiszámolják a függvény vagy kifejezés értékét, s azt tárolják az indexben. Típusát tekintve B-fa vagy bitmap indexek lehetnek. A függvény maga lehet aritmetikai kifejezés, vagy olyan kifejezés, ami tartalmaz PL/SQL függvényt, csomag függvényt, C hívást vagy SQL függvényt. Nem tartalmazhat azonban aggregátum függvényeket, `DETERMINISTIC`-usnak kell lennie, s nem vonatkozhat `LOB` típusú, `REF` vagy beágyazott tábla oszlopára sem.

Használat. Példák[10]:

- lekérdezések WHERE ágában szereplő feltételek kiszámításának gyorsítására:

```
CREATE INDEX idx ON table_1 (a + b * (c - 1), a, b);
SELECT a FROM table_1 WHERE a + b * (c - 1) < 100;
```

- case-insensitive keresések gyorsítása:

```
CREATE INDEX uppercase_idx ON employees (UPPER(first_name));
SELECT * FROM employees WHERE UPPER(first_name) = 'RICHARD';
```

Optimalizálás. Az optimalizálónak (optimizer) szüksége van statisztikákra ahhoz, hogy használni tudja a függvény-alapú indexeket. Az optimalizáló a kifejezésfák alapján választ egy SQL utasításhoz hozza illeszkedő függvény-alapú indexet. Nyilván minél kevesebb variáció lehetséges a `WHERE` klóz alapján, annál hatékonyabb lesz ez a keresés.

[10] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/schema.htm#CHDECAAD`

4.7.7. Tárolás

Az index létrehozását követően az adatbázis automatikusan lefoglal neki egy index szegmenst. A lefoglalt szegmens mérete a tábláknál már ismertetett módon történik. Azaz vagy tárolási paraméterek (storage parameteres) megadásával, vagy a `PCTFREE` és a `PCTUSED` paraméterekkel állíthatjuk be. Az index számára a létrehozásakor megadhatunk egy - az ownerjétől különböző - tablespacet is a tárolásra. Ezen túl, ha az indexet még egy külön, a tábláétól különböző diszkre is tesszük, akkor mivel az adatbázis a két diszket párhuzamosan tudja használni, teljesítménynövekedést érhetünk el.

4.7.8. Blokk formátuma

Az index létrehozását követően az adatbázis fetcheli és rendezi az indexelt oszlopokat, s eltárolja a rowid-ket minden egyes sor index értékéhez.

4.7.9. Belső struktúra

Az Oracle adatbázis B-fákat (ábra[11]) használ az indexek tárolására. Ezáltal a szekvenciális keresés átlagos n/2 idejét lecsökkenti $O(log(n))$-re.

4.7.10. Unique Scan

Az egyik leghatékonyabb adathozzáférési módszer. Az optimalizáló (optimizer) ezt a módszert használja unique index esetén.

4.7.11. Range Scan

Kevésbé hatékony adathozzáférési módszer. Az adatot az index oszlopok szerinti növekvő sorrendben adja vissza, duplikált sorok esetén `ROWID` szerinti növekvő sorrendben.

4.7.12. Kulcstömörítés

Lehetővé teszi, hogy az elsődleges kulcs (primary key) értékek csoportjait egy indexbe vagy egy index-szervezett táblába tömörítsük, csökkentve ezáltal az ismételt értékekből származó tárolási overheadet. Általában egy index egy csoportazonosító és egy egyedi részre bontható. A tömörítést a csoport részre végezzük: megadjuk, hogy az index milyen hosszú részét képezi, majd az ezáltal lehetséges csoportazonosítókat csak egyszer tárolunk el.

Teljesítmény. A kulcstömörítés jelentős helymegtakarítást jelenthet, ezáltal csökkentve az I/O műveleteket és növelve a teljesítményt, azonban a kulcsok újbóli felépítése némi CPU overheadet jelenthet az indexek scanelése közben.

[11] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/schema.htm#i5765`

Használat. Az alábbi esetekben célszerű kulcstömörítést használni:

- nonunique reguláris indexeknél, ugyanis az Oracle adatbázis a duplikált sorokhoz duplikált keyeket használ.
- olyan unique indexek esetén, ahol a kulcs egyediségét nem az első attribútum generálja. Pl.: egy elemre és az elemhez történt hozzáférések időbélyegére definiáltunk indexet. Ekkor a csoport rész (prefix) lehet az elem, az egyediséget pedig (suffix rész) az időbélyeg fogja garantálni.
- `VARRAY` vagy `NESTED TABLE` -t tartalmazó index-szervezett tábláknál, mivel az objektumazonosító ismételt a kollekciók minden elemére.

4.7.13. Reverse key (inverz kulcs) indexek

A hagyományos indexekkel ellentétben megfordítják minden indexelt oszlop bájtjait (a ROWID kivételével), míg az oszlopsorrendet változatlanul hagyják. Ezáltal a nagyjából egy időben létrehozott rekordok nem kerülnek a B*-fa azonos ágaira, hanem szépen szétszórjuk őket. Ugyan ezzel a módszerrel kizárjuk az indexen az index range scanninget használó lekérdezéseket, de cserébe RAC-os alkalmazások futását meggyorsíthatjuk, mivel a beillesztés elosztottan mehet végbe az index levelei között.

4.7.14. Bitmap indexek

Az Oracle adatbázis által használt B-fa indexeknél a kulcsértéket minden egyes `ROWID`hez eltároljuk, így ismétlődés fordulhat elő. Bitmap indexek esetén minden kulcsértékhez egy bitmapet használunk. A bitmap minden egyes bitje egy lehetséges `ROWID`-t reprezentál, s akkor van beállítva, hogyha a hozzá tartozó `ROWID` tartalmazza a kulcsértéket. Kevés különböző kulcsérték esetén a bitmap indexek használata (B-fa indexek helyett) jelentős helymegtakarítást jelent - alkalmazásának adattárházak (Data Warehouses) esetén van számos előnye. Lekérdezések hatékonyságának gyorsítására főként abban az esetben alkalmas, ha a `WHERE` feltételében ekvivalencia vizsgálatok (illetve azok `AND`, `OR` és `NOT` operátoros kombinációi) szerepelnek.

A legtöbb indexszel ellentétben `NULL` értéket tartalmazó sorokat is képes felhasználni, ami hasznos lehet a `COUNT` aggregátum használata esetén.

4.7.15. Bitmap Join indexek

Több illesztett táblára is létrehozhatunk bitmap (join) indexet, ami tárolási szempontból jóval előnyösebb a materializált join nézeteknél (mivel ők nem tömörítik a hivatkozott táblák `ROWID`-jét).

4.8. Index-szervezett táblák

A hagyományos táblákkal ellentétben – ahol az adatok rendezetlenül, kupacban (heap) tárolódnak –, az index-szervezett (index-organized) táblák adatait egy elsődleges kulcs szerint rendezett B-fában tároljuk. Ezáltal nem kell külön indexet létrehozni és fenntartani az elsődleges kulcsra (Összehasonlítás[12]).

4.8.1. Előnyei

- gyorsabb hozzáférés a tábla soraihoz
- mivel a többi, nem kulcs értékű információ is megtalálható a fában, nincs szükség további blokkhozzáférésre
- az elsődleges kulcs szerinti sorrendhelyes tárolás miatt minimális blokkhozzáférésre van szükség
- a ritkán használt nem kulcs típusú oszlopokat ki lehet pakolni egy külön kupacba, ezáltal csökkentve a B-fa méretét
- nem kell az elsődleges kulcsra külön indexet fenntartani (kevesebb tárhely igény)
- a sorrendhelyes kulcstárolás miatt kulcstömörítés alkalmazható

4.8.2. Overflow

Mivel az index-szervezett táblákban esetlegesen nagyon sok nem kulcs típusú oszlopot is tárolhatunk, az egyes levelek mérete igencsak megnőhet, s ezáltal elveszítheti a B-fa sűrűn fürtözött tulajdonságát. Ennek elkerülése végett az `OVERFLOW` klózzal két részre oszthatjuk az oszlopokat: az első részbe tartoznak az index bejegyzések és fizikai `ROWID`-k, valamint megadhatunk néhány nem kulcs típusú oszlopot, amit szeretnénk a B-fában tartani. A második részbe (overflow rész) kerül a többi nem kulcs típusú oszlop, melyeket külön tárolunk.

4.8.3. Másodlagos indexek

Index-szervezett tábláknál is van lehetőség nem kulcs típusú oszlopokra indexeket létrehozni (ún. másodlagos/secondary indexek). Ezek működése azonban általában lassabb a hagyományos táblákra létrehozott indexekénél (azonos működés érhető el, ha helyes sejtésünk van az adat fizikai elhelyezkedéséről).

4.8.4. UROWID

Az index-szervezett táblák sorainak azonosítására használható, logikai elsődleges kulcs alapú `ROWID`-k.

[12] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/schema.htm#g37535`

4.8.5. Alkalmazások

Az elsődleges kulcs alapú lekérdezéseket használó, valamint a tárhellyel takarékoskodni akaró alkalmazások számára ideális. Pl.: OLTP (Online Transaction Processing), Internet (kereső motorok és portálok), Elektronikus kereskedelem, Adattárházak (Data Warehouses), Analitikus függvények.

4.9. Alkalmazás Domain Indexek

Az Oracle adatbázis indexelése kiterjeszthető komplex adattípusokra – mint pl. dokumentumok, spatial adatok, képek, videók, stb. – is. Ezeket az indexeket hívjuk domain indexeknek, melyeket a Cartridge szoftverrel lehet szabályozni.

4.10. Klaszterek

Klaszternek (**cluster**) nevezzük táblák egy csoportját, melyek – mivel közös oszlopokat használnak – ugyanazon az adatblokkokon osztoznak (példa[13]).

4.10.1. Előnyei

- kevesebb I/O művelet az egy klaszteren belül lévő illesztett táblákra
- gyorsabb hozzáférés az egy klaszteren belül lévő illesztett táblákra
- klaszter kulcsértékek (és indexek) csak egyszer tárolódnak, ezáltal kevesebb tárhelyet fogyasztanak

4.11. Hash klaszterek

A hash klaszterek (**hash cluster**s) bizonyos értelemben megfelelnek a klaszterekre definiált hagyományos indexekkel, azzal a különbséggel, hogy a kulcs hash értékét tárolják. Jobb teljesítményt lehet velük elérni olyan tábláknál, melyen gyakran alkalmaznak ekvivalencia vizsgálatos feltételt tartalmazó lekérdezéseket, mivel a hash kulcs közvetlenül a diszken lévő adatra mutat.

5. Séma objektum függőségek

Vegyünk két objektumok, és nevezzük őket A-nek és B-nek. Ha A objektum a definíciójában hivatkozik B objektumra, akkor azt mondjuk, hogy A függ B-től. Ez a fejezet a séma objektumok között létrehozható függőségeket, illetve azt taglalja, hogy ezeket a függőségeket az Oracle adatbázis miként követi nyomon és menedzseli automatikusan.

[13] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/schema.htm#i30164`

5.1. Általánosságban

Egy objektum a definíciójában hivatkozhat egy másik objektumra. Például egy nézet hivatkozik a master tábláira és/vagy nézeteire, illetve egy esetleges alprogram törzsében más objektumokra is. Ha módosítunk egy hivatkozott objektumot, akkor a tőle függőségben lévő objektumok esetlegesen nem fognak működni – ez nyilván csak akkor fordul elő, ha a hivatkozott objektum olyan attribútumát változtatjuk meg, amire a tőle függőségben lévő objektum hivatkozik.

A `DBA_DEPENDENCIES`, `USER_DEPENDENCIES` és `ALL_DEPENDENCIES` táblák írják le az adatbázis objektumai között fennálló függőségeket.

5.2. Állapotok

A séma objektumok az alábbi állapotok valamelyikében lehetnek:

- `VALID`: sikeresen lefordult az adatszótárban (data dictionary) található aktuális definíció alapján.
- Compiled with Errors: legutóbbi fordítási kísérlet során hiba lépett fel.
- `INVALID`: egy hivatkozott objektum megváltozott (csak a függőségben lévő objektum tud `INVALID` állapotba kerülni, a hivatkozott nem).
- `UNAUTHORIZED`: egy hivatkozott objektumról visszavonták (`REVOKE`) a hozzáférési jogosultságát a hivatkozó objektumnak (csak a függőségben lévő objektum tud `UNAUTHORIZED` állapotba kerülni, a hivatkozott nem).

5.3. INVALID állapotba kerülés okai

Megkülönböztethetünk közvetlen (direct) és közvetett (indirect) függéseket az alapján, hogy A objektum a közvetlenül hivatkozott B objektumtól is függhet, illetve közvetetten a B objektum által hivatkozott C objektumtól is. A objektum mind a B, mind a C objektum megváltoztatása esetén INVALID állapotba kerülhet. A műveletekről, melyek hatással lehetnek egy objektum állapotára itt[14] található egy összefoglaló táblázat.

5.4. INVALID állapotba jutás elkerülése

- az új elemeket a csomag végére illesszük be, hogy az többi objektumra történő hivatkozások ne invalidálódjanak. Pl.[15]: `assert_var` beillesztésével a `set_var`-ra hivatkozó objektumok invalidálódnak.

[14] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/dependencies.htm#g1008856`

[15] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/dependencies.htm#CHDCEHIF`

```
CREATE OR REPLACE PACKAGE pkg1 IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE assert_var (v VARCHAR2);
  PROCEDURE set_var (v VARCHAR2);
END;
```

- a táblákra indirekt, nézeteken keresztül hivatkozzunk. Ezzel azt érjük el, hogy a táblába új sor felvételekor, illetve nem hivatkozott oszlopok módosításakor vagy törlésekor a függőségben lévő objektumok nem válnak `INVALID`-dá.

5.5. Objektumok „ReVALID”-álása

Egy objektum nem `VALID` (azaz nem érvényes/hivatkozható), ha a három másik állapot (Compiled with Errors, `UNAUTHORIZED`, `INVALID`) valamelyikében tartózkodik. Ha egy nem VALID objektumra hivatkozás történik, akkor a használat előtt ReVALID-álni kell, különben nem lehet majd használni. Ez a ReVALID-álási kísérlet automatikusan végbemegy.

- *Compiled with Errors objektum revalidálása:* a fordító automatikusan nem tudja revalidálni az objektumot. Megpróbálja újrafordítani azt. Ha sikerül `VALID` állapotba kerül, egyébként marad Compiled with Errors.
- *`UNAUTHORIZED` objektum revalidálása:* a fordító ellenőrzi a hozzáférési jogokat. Ha időközben megkaptuk őket, akkor `VALID` állapotra vált, egyébként hibaüzenetet ír ki.
- *`INVALID` SQL objektum revalidálása:* lásd Compiled with Errors revalidálás.
- *`INVALID` PL/SQL objektum revalidálása:* a PL/SQL fordító megnézi, hogy a hivatkozott objektumokban történt-e az `INVALID` objektumot érintő változás. Ha igen, akkor a fordító újrafordítja az `INVALID` objektumot, s siker esetén `VALID` állapotba helyezi. Ha nem történt az `INVALID` objektumot érintő változás, akkor újrafordítás nélkül próbálja meg a fordító revalidálni az objektumot (lásd fast revalidálás).
- *`INVALID` PL/SQL objektum „fast” revalidálása:* a fordító újrafordítás nélkül revalidálja az objektumot. Főként indirect függőségből származó invalidálódás esetén szokott sikerrel járni.

5.6. Remote Procedure Call (RPC) Függőség Menedzsment

Elosztott adatbázis esetén fordul elő, amikor egy helyi eljárás távoli eljáráshívást hajt végre.

- *Időbélyeg ellenőrzés:* az eljárások létrehozásakor, módosításkor és felülírásukkor mindig feljegyezzük a műveletek időbélyegét az adatszótárba (data dictionary). Ha egy olyan eljárást hívunk meg, amit tartalmaz távoli eljáráshívást is, akkor az adatbázis összehasonítja a fordítás időbélyegét a távoli eljárás éppen aktuális időbélyegével. A következő két eset fordulhat elő:

1. a helyi és a távoli eljárás időbélyegei egyeznek, az eljárás gond nélkül lefut.

2. ha a távoli eljárás bármely időbélyege nem egyezik, a helyi eljárás invalidálódik, s nem hibaüzenettel tér vissza az eljáráshívás. Továbbá invalidálódik az összes többi, arra a távoli eljárásra hivatkozó helyi eljárásunk is. Célszerű tehát a hálózaton keresztül hivatkozott objektumokat ritkán újrafordítani, hogy jelentős teljesítménycsökkenést tudnak okozni!

- *Aláírás ellenőrzés:* a távoli függőségekre egy alternatív lehetőséget jelentenek az RPC aláírások. Egy eljárás aláírása tartalmazza a nevét (csomag, eljárás vagy függvény neve), a paraméterek típusát, üzemmódját (`IN`/`OUT`/`IN OUT`) és számát, illetve függvény esetén a visszatérési érték típusát. Az aláírások használata enyhít néhány, az időbélyeg alapú modell esetén jelentkező problémát, melyek kritikusak lehetnek a teljesítményre nézve (pl. újrafordított távoli eljárás esetén, ha az aláírás nem változott, akkor gond nélkül lefut a helyi eljárásunk). Az aláírás adattípusok (link[16]) közötti váltáskor következik be (az adattípus osztályán belüli váltáskor nem).

- *Vezérlés:* a fenti két módot a REMOTE_DEPENDENCIES_MODE inicializáló paraméter segítségével állíthatjuk be ($= SIGNATURE|TIMESTAMP$)

6. Adatszótár

Az adatszótár (**data dictionary**) az Oracle adatbázis egyik legfontosabb részét képezi. A központi, csak olvasható referencia táblák és nézetek tartoznak hozzá, melyekben az adatbázisról tároljuk a következő lényeges információkat:

- séma objektumok definíciója
- séma objektumok számára allokált és felhasznált területek
- oszlopok alapértelmezett értékei
- integritás kényszerekről információk
- az adatbázis felhasználóinak nevei
- az egyes felhasználókhoz tartozó jogok és szerepek
- naplózási információk
- egyéb általános adatbázis információk

A SYSTEM tablespaceben tároljuk, ami mindig online, ezért ezek az információk mindig elérhetőek. Szerkezetileg kétféle objektumot tartalmaz:

[16] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/dependencies.htm#g4951307`

1. *Alap táblák (**Base tables**):* az adatbázisról tartalmaznak információkat. Csak az Oracle adatbázis írhatja és olvashatja őket. Az adatokat titkosított formában tárolják.

2. *Felhasználói nézetek (**User-Accessible Views**):* nézetek, melyek összegzik és a felhasználók számára emészthető formában megjelenítik az Alap táblákban tárolt információkat. A legtöbb felhasználó ezekhez a nézetekhez férhet hozzá.

Minden *Base table* és *User-Accessible View* a `SYS` felhasználó sémájában van, így egyetlen felhasználó sem módosíthatja őket, mivel az súlyos veszélyeket jelentene az adatintegritásra. Éppen ezért célszerű nagy biztonságban tartani ezt a központi accountot!

6.1. Felhasználás

Az adatszótár használatának három fő területe van:

- az adatbázis lekérdez információkat felhasználókról, séma objektumokról és tárolási struktúrákról
- az adatbázis módosítja az adatszótárt minden DDL utasítás végrehajtása esetén
- bármely felhasználó lekérdezhet információkat az adatbázisról

Az adatbázis a legtöbb nézethez létrehoz **Publikus Szinonimák**at, hogy azok kényelmesen elérhetőek legyenek, de a `SYS` felhasználóval további publikus szinonimákat is létre lehet hozni manuálisan. Fontos, hogy a séma objektumok elnevezései lehetőleg ne keveredjenek a publikus szinonimák nevével.

Az adatszótár lehetőleg minél nagyobb részét az SGA (System Global Area) **dictionary cache** részében tároljuk a gyors hozzáférés érdekében. Nyilván az egész adatszótárt nehéz lenne cacheelni, ezért a least recently used (`LRU`) algoritmus alapján a legrégebben használt adatokat dobjuk ki először.

Más Oracle adatbázis **termékek** is hivatkozhatnak, illetve létrehozhatnak objektumokat maguknak az adatszótárban. Célszerű az alkalmazásfejlesztőknek a hivatkozásoknál a publikus szinonimákat használni, mivel azok a legtöbb verzióban azonosak.

Az adatszótár nézetei általában három különböző prefixszel vannak ellátva, melyek jelentése:

- *`USER:`* felhasználói nézet – a felhasználó sémájának objektumairól: főként az adott felhasználóhoz tartozó információkat tartalmazzák. Így például a `SELECT object_name, object_type FROM USER_OBJECTS;` lekérdezés a sémánkban lévő objektumokat adja vissza.

- *`ALL:`* kiterjesztett felhasználói nézet – a felhasználó által hozzáférhető nézet: az adott felhasználó által elérhető „információkat” tartalmazzák. Így például a `SELECT owner, object_name, object_type FROM ALL_OBJECTS;` lekérdezés az általunk hozzáférhető objektumát adja vissza.

- *DBA:* adatbázis adminisztrátori nézet – minden felhasználó sémájában megtalálható nézet: globális információkat tartalmaznak az adatbázisról. Így például a `SELECT owner, object_name, object_type FROM SYS.DBA_OBJECTS;` az adatbázis összes objektumat adja vissza. Az „`O7_DICTIONARY_ACCESSIBILITY is false`" beállítással a felhasználóknak megtilthatjuk a `SYS` sémán belüli objektumokhoz való hozzáférésüket.

6.2. Dinamikus Teljesítmény táblák

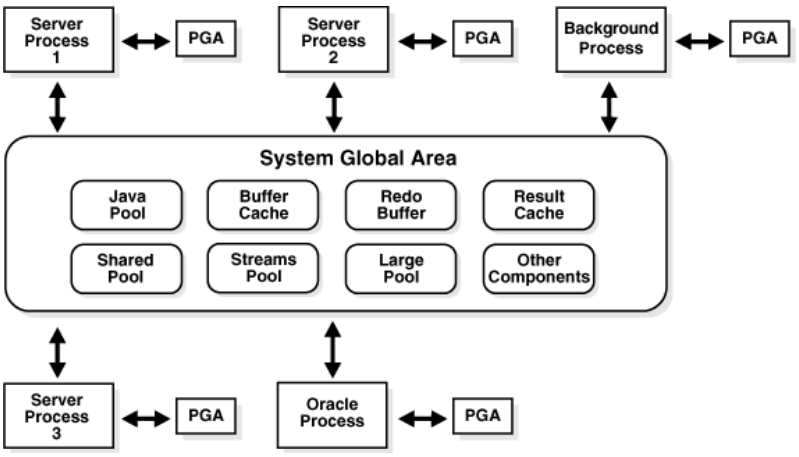
Virtuális táblák halmaza, melyek az adatbázis tevékenységeket rögzítik. Nem valós táblák, de az adminisztrátor létrehozhat rájuk nézeteket a felhasználók számára. Ezeket a nézeteket fix nézeteknek hívjak, mivel az adminisztrátor nem tudja módosítani vagy törölni őket.

A `SYS` felhasználóhoz tartoznak, s `V_$` karakterekkel kezdődik a nevük. Ezekre a táblákra hozhatunk létre nézetek, majd a nézetekre publikus szinonimákat, melyek neve `V$` karakterekkel kezdődik. Pl. `V$DATAFILE` tartalmaz információkat az adatbázis adatfájljairól, míg `V$FIXED_TABLE` magukról a dinamikus teljesítmény táblákról.

7. Memória architektúra

Az Oracle adatbázis memóriájában a következő komponenseket tároljuk:

- programkód
- információ a csatlakozott active és inactive sessionökről
- programvégrehajtás közben szükséges információk, állapotok
- az adatbázis processzei között megosztott információk (pl. zárak)
- cachelt adatok, mint pl. adatblokkok és redo log bejegyzések, amik ugyanakkor természetesen tárolva vannak a merevlemezeken is



[17]

[17] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/memory.htm#CHDHAHIJ`

A memória struktúrája alapvetően három része osztható:

- *Szoftver kód területek:* az éppen futó vagy futtatható kódokat tartalmazza. Általában a felhasználói programoktól eltérő helyen van.
- *System Global Area (SGA):* az összes szerver- és háttérfolyamat (processz) között megosztott memória struktúrák (SGA komponensek – lásd fenti ábra) csoportja. Adat és vezérlési információkat tartalmaz.
- *Program Global Area (PGA):* egy bizonyos szerver processzhez tartozó adat és vezérlési információkat tartalmazó memóriaterület, mely a szerverprocessz indításakor jön létre, s csak ő fér hozzá. Minden egyes szerverprocesszhez (és háttérfolyamathoz) tartozik egy-egy PGA. Az adatbázis inicializálási paramétereinél megadott PGA méret az összes PGA együttes méretére, s nem az egyes példányokra vonatkozik.

7.1. System Global Area (SGA)

Az adatbázis processzeinek halmaza az SGA-val együtt alkot egy adatbázis példányt (instance). Minden egyes adatbázis példány létrehozásakor az Oracle adatbázis automatikusan lefoglal memóriát az SGA számára, s visszaadja az operációs rendszernek a példány a leállításakor.

Az SGA egy írható és olvasható memóriaterület. Minden szerverprocessz és háttérfolyamat olvashatja, s néhány processz adatbázis műveletek során írhatja is.

Az SGA egy része a háttérfolyamatok számára tartalmaz általános információkat az adatbázis és a példány állapotáról. Ezt a területet hívjuk fixed SGA-nak. A változó rész elsősorban a processzek között kommunikált információt (pl. zárolási információk), illetve osztott szerver-architektúra esetén a kérés/válasz sorokat és a PGA tartalmának egy részét tárolja.

A legfontosabb SGA komponensek:

7.1.1. Database Buffer Cache

Az adatfájlokból (datafiles) beolvasott adatblokkok (data blocks) másolatait tartalmazza. Minden felhasználó konkurensen hozzáfér.

A cache bufferjei két listát tartalmaznak:

1. A *write list* tartalmazza a piszkos (dirty) buffereket, melyekben a módosított, de a diszkre még nem kiírt adatok vannak.
2. Az *LRU (Least Recently Used) list* tartalmazza az üres buffereket, a pinned (éppen hozzáférés alatt álló) buffereket és az olyan piszkos buffereket, amelyeket még nem tettünk át a write listbe.

Egy bufferhez történő hozzáférést követően a buffert az LRU lista MRU végére mozgatjuk. Minél több buffert helyezünk folyamatosan az LRU lista MRU végére, annál gyorsabban öregednek a dirty bufferek, s kerülnek az LRU lista LRU végére.

Ha az adatbázisnak szüksége van egy konkrét adatra, akkor először a buffer cacheben kezdi el keresni. Ha megtalálja az adatot a cacheben (cache hit), akkor közvetlenül a memóriából be tudja olvasni. Ha nincs bent az adat a cacheben (cache miss), akkor először be kell olvasni a merevlemezről az adatblokkot a memóriába, s csak utána lehet hozzáférni. Ebből következik az LRU lista előnye, hogy a gyakran használt adatokat bent tartja a memóriában, s ezáltal hozzájuk sokkal gyorsabb hozzáférést biztosít.

Az adatblokk cachebe történő beolvasásához először keresni kell egy üres (free) buffert az LRU listában. A keresést a lista LRU végéről kezdi. Ha keresés közben piszkos buffert találunk, akkor azt a buffert áthelyezzük a write listába, majd folytatjuk a keresést. Ha találtunk egy üres (free) buffert, akkor beolvassuk oda az adatblokkot, s a buffert az LRU lista MRU végére mozgatjuk. Ha elértük a keresési limitet, s nem sikerült üres (free) buffert találni, akkor leállítjuk a keresést, és jelzünk a DBW0 háttérfolyamatnak, hogy írjon ki néhány piszkos buffert a merevlemezre.

Full table scan esetén – mivel a beolvasott adatok általában csak rövid ideig kellenek – az adatokkal feltöltött buffereket az LRU lista LRU végére tesszük (a hagyományos adatbeolvasásnál ugyebár a lista MRU végére pakoljuk a buffereket, hogy ne egyből őket dobjuk majd ki). Ha a buffereket mégis a lista MRU végére szeretnénk pakolni, akkor a `CREATE`/`ALTER` parancs `CACHE` klózával tudjuk ezt megadni.

7.1.2. Redo Log Buffer

Itt tároljuk az adatbázison végrehajtott változtatásokat. Egy körkörös, fix méretű buffer, melynek merevlemezre írását az `LGWR` processz végzi. A Redo bejegyzések az adatbázis esetleges helyreállításához szükségesek. Mivel minden `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `ALTER` és `DROP` művelet által végrehajtott módosítást tartalmaznak, így remélhetőleg könnyedén vissza tudunk állítani egy korábbi, konzisztens állapotot. A DDL utasítások használata esetén természetesen nem tudunk egy objektum tetszőleges állapotához visszatérni, csak a pl. checkpointok által mentettekhez (ha megvannak a checkpoint óta történt változtatásokat tartalmazó logok).

7.1.3. Shared Pool

A Shared Pool három cache területből áll:

1. Library Cache:. tartalmazza a megosztott (shared) és privát (private – csak shared server[18] konfiguráció esetén) SQL területeket (areas), PL/SQL eljárásokat és csomagokat, valamint vezérlési struktúrákat.

[18] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28310/manproc001.htm#i1006130`

A shared(/private) SQL areak tartalmazzák a futtatott és éppen futás alatt álló SQL utasítások elemzési fáját és végrehajtási tervét. Az Oracle adatbázis automatikusan felismeri, hogyha két felhasználó ugyanazt az SQL utasítást futtatja, ezáltal sokfelhasználós rendszerek esetén, amikor egy utasítást gyakran és sokszor futtatnak le a felhasználók, jelentős memóriát takaríthatunk meg, mivel elegendő az utasítás elemzési fáját és végrehajtási tervét csak egy példányban tárolnunk, s ahhoz minden felhasználó egyaránt hozzáfér. A shared pool memóriájának menedzselésére szintén az LRU algoritmust használjuk (részletesebben lásd „Buffer Cache" vagy „Kapcsolódó Linkek").

PL/SQL programok kezelése majdnem teljesen hasonlóan történik a sima SQL utasításokéhoz. A lefordított programnak ugyanúgy foglalunk helyet a shared SQL areaban, de a session specifikus változókat a private SQL areaban tároljuk.

Néhány, az alap LRU algoritmustól eltérő esetben is kiteszünk egy shared SQL areat a Shared Poolból. Ezek:

- Az `ANALYZE` statisztikagyűjtő utasítás használata esetén minden olyan shared SQL areat kidobunk a shared poolból, ami tartalmaz hivatkozást az analizált séma objektumra.
- Ha egy SQL utasítás által hivatkozott objektum az utasítás shared poolba kerülését követően módosul, akkor az utasítás shared SQL areajá invaliddá válik, s az utasítást újra le kell fordítani a következő futtatása előtt.
- Ha az adatbázis globális nevét megváltoztatjuk, minden információt kiürít az adatbázis a shared poolból.
- Az `ALTER SYSTEM FLUSH SHARED_POOL` utasítással az adminisztrátor manuálisan kiürítheti az egész shared poolt. Ezzel az adatbázis újraindítása nélkül tud a shared pool adatbázispéldány indítása utáni teljesítményéről információkat gyűjteni.

2. Dictionary Cache:. az adatszótár (data dictionary) cacheelésére szolgál. Két részből áll: a data dictionary cache (másik nevén row cache) az adatokat sorok (rows) formájában, s nem blokkokban (bufferekben) tárolja. A másik rész, a library cache buffereket használ.

3. Result Cache:. a SQL query result cacheből és a PL/SQL query cacheből áll, melyekben a lekérdezések/függvények eredményét cacheeljük. A `DBMS_RESULT_CACHE` csomag, illetve a `V$RESULT_CACHE_*` nézetek segíthetnek az adminisztrátornak a karbantartásában. A `RESULT_CACHE_MODE`-ban állíthatjuk be, hogy minden SQL lekérdezés eredményét szeretnénk-e cacheelni, vagy csak bizonyos megjelölt lekérdezéseket. Természetesen, ha egy tranzakció egy olyan objektum bármely adatát vagy metaadatát módosítja, amit a lekérdezés eredményének kiszámításához használtunk, akkor a cacheelt eredmény azonnal invaliddá válik.

7.1.4. Large Pool

Az adatbázis adminisztrátor által konfigurálható opcionális memóriaterület, mely a következők számára biztosít extra, nagyméretű memóriát:

- session memória az osztott szervernek és az Oracle XA interfésznek (több adatbázissal kommunikáló tranzakciók használják), hogy ne a shared SQL cachet pakolják tele
- I/O szerver processzek
- Oracle adatbázis biztonsági mentési és visszaállítási műveletei

Ellentétben a Shared Pool-lal, a Large Pool nem használ LRU listákat.

7.1.5. Java Pool

A session-specifikus Java kódok és adatok használják. A Java Pool Advisor segítségével statisztikát gyűjthetünk, amely segítségével információt kapunk arról, hogy hogyan változtassuk a Java Pool méretét a hatékonyabb működés érdekében. Az Advisor a `statistics_level` legalább `TYPICAL`-ra állításával van bekapcsolva.

7.1.6. Streams Pool

Az Oracle Folyamok (Streams) használják. Mérete nulláról indul, és dinamikusan növekszik, amikor a Folyamoknak szüksége van memóriára.

7.2. Program Global Area (PGA)

Az Oracle adatbázis minden szerverfolyamat számára automatikusan lefoglal egy PGA-t. SQL utasítások végrehajtására, illetve session-specifikus (pl. bejelentkezési) információk tárolására használják. Két részből áll:

7.2.1. Session Memória

Session specifikus információkat tárol. Shared szerver konfiguráció esetén megosztottan működik, amúgy privát.

7.2.2. Privát SQL terület (Private SQL Area)

Bind variable értékeket, lekérdezések végrehajtásának állapotinformációit és munkaterületeit tartalmazza. Minden sessionhöz tartozik egy külön privát SQL terület, így ha két felhasználó ugyanazt a lekérdezést futtatja, mindegyiknek külön privát SQL területük van, de egy osztott SQL területet használnak. A privát SQL területek elhelyezkedése kapcsolatfüggő. Ha dedikált szerveren keresztül csatlakozunk, akkor a szerverfolyamat PGA-jában kap helyet. Azonban ha osztott szerveren keresztül kapcsolódunk, akkor a privát SQL terület egy része az SGA-ban tárolódik.

Implicit kurzorok számára osztott SQL területeket használ az adatbázis, de az explicit kurzoroknak külön privát SQL területet foglal le. Az `OPEN_CURSORS` inicializálási paraméterben adhatjuk meg, hogy egy felhasználói processz maximum hány privát SQL területet foglalhat le magának. A paraméter alapértéke 50.

Memóriafoglalás alapján a különbségek dedikált és osztott szerver esetén: táblázat[19]

7.3. Memóriakezelési eljárások

Memory Management Methods – táblázat[20]: inicializálási paraméterben adható meg. Az Oracle az automatic management method-t ajánlja. Instance PGA = az adatbázispéldányhoz tartozó összes egyedi PGA-k halmaza.

- *Automatic Memory Management* – SGA & Instance PGA: 11g újdonság. Csak a teljes memóriaterületet kell megadnunk, s onnantól az Oracle adatbázis mindent automatikusan elvégez. Dinamikusan elosztja a memóriát az SGA és az Instance PGA között, valamint külön-külön az egyes SGA komponensek és egyedi PGA-k méretét is meghatározza.
- *Automatic Shared Memory Management* – SGA: az Automatic Memory Managementtel szemben ebben az üzemmódban meg tudjuk adni az SGA célzott és maximális méretét. Ebben az esetben a rendszer megpróbálja az SGA méretét a célméreten tartani, és dinamikusan állítja az egyes SGA komponensek méretét.
- *Manual Shared Memory Managemen*t – SGA: ebben az üzemmódban nem csak az SGA méretét állíthatjuk be, hanem szabályozhatjuk az egyes SGA komponensek rendelkezésére álló memóriát is.
- *Automatic PGA Memory Management* – Instance PGA: Az Automatic Memory Management kikapcsolásával, s az Automatic Shared Memory Management vagy a Manual Shared Memory Management bekapcsolásával implicit bekapcsoljuk ezt az üzemmódot is. Beállíthatjuk az Instance PGA célzott méretét, és az egyedi PGA-k méretet a rendszer dinamikusan szabályozza majd. Ha nem adunk meg célzott méretet, akkor az adatbázis automatikus kiszámol és beállít egy értelmes alapértéket.
- *Manual PGA Memory Management* – Instance PGA: az Oracle adatbázis korábbi verzióiban a DBA-nak manuálisan kellett definiálnia a maximális munkaterületet (work area) minden SQL operátor számára, ez azonban nagyon körülményes és nehéz feladat, hiszen a munkaterület mérete állandóan változik. Az Oracle erősen ajánlja, hogy ne használjuk ezt az üzemmódot!

Ha az adatbázist a Database Configuration Assistant (DBCA) segítségével hozzuk létre, akkor „basic” installálási opciót választva automatic memory management lesz beállítva. „Advanced” installálást választva az alábbi három konfiguráció közül választhatunk:

[19] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/memory.htm#g2607`
[20] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/memory.htm#CHDFFHGH`

1. automatic memory management
2. automatic shared memory management + automatic PGA memory management
3. manual shared memory management + automatic PGA memory management

Ha kézzel, a CREATE DATABASE SQL paranccsal akarunk adatbázist létrehozni, akkor a 3. konfiguráció az alapértelmezett.

7.4. Szoftver kód területek

Olyan memóriaterületek, amiket az Oracle adatbázis futó vagy futtatható kódjainak tárolására használunk. Általában statikus méretű, csak szoftverfrissítéskor vagy újratelepítéskor változik. Az igényelt terület mérete operációs rendszertől függ. Csak olvasható, megosztottan (shared) és nem megosztottan (nonshared) installálhatjuk. Ajánlott az előbbit használni, mivel így a felhasználóknak nem kell többszörös másolatokat a memóriában tartaniuk, ami amúgy az általános teljesítményt rontaná.

8. Processz architektúra

Ez a fejezet az Oracle adatbázis processzeivel (folyamataival), illetve az adatbázis különböző konfigurációival foglalkozik.

Minden Oracle adatbázishoz csatlakozott felhasználónak két kód-modult kell futtatnia ahhoz, hogy hozzáférhessen egy adatbázis példányhoz. Ezek:

- Valamilyen adatbázis **alkalmazás** (pl. előfordító program) vagy **Oracle eszköz** (pl. SQL*Plus), amely SQL utasításokat továbbít egy Oracle adatbázishoz.
- Adatbázis **szerver kód**, amely értelmezi és végrehajtja az SQL utasításokat.

Ezeket a kód-modulokat futtatják a processzek. Ennek megfelelően a processzeket is két fő csoportra oszthatjuk: *felhasználói processzek*, melyek az alkalmazás kódját futtatják, illetve *Oracle adatbázis processzek*, ahová a szerver- és háttérfolyamatok tartoznak.

A processzek struktúrája függ az operációs rendszertől és az adatbázis beállításaitól is. Dedikált vagy osztott szerveres megoldások közül választhatunk. Dedikált szerveres kapcsolat esetén minden felhasználóhoz tartozik egy felhasználói (user) és egy dedikált szerver processz, míg osztott szerveres kapcsolatnál egy szerver processz akár több felhasználót is kiszolgálhat egyszerre.

8.1. Felhasználói processzek

Ha egy felhasználó valamilyen adatbázishoz csatlakozó alkalmazást vagy Oracle eszközt (pl. Enterprise Manager vagy SQL*Plus) futtat, akkor az adatbázis automatikusan létrehoz számára egy felhasználói processzt, mely a felhasználó alkalmazását fogja kezelni. Néhány fogalom:

- *Kapcsolat (Connection):* kommunikációs csatorna a felhasználói processz és az adatbázis példány között. Használhat interprocessz kommunikációt, ha a felhasználói processz és az adatbázis egy gépen helyezkedik el, illetve csatlakozhat hálózaton keresztül, ha különbözőn.
- *Session:* egy adott felhasználóhoz tartozó kapcsolat (a felhasználói processzen keresztül az adatbázishoz).

8.2. Szerverfolyamatok

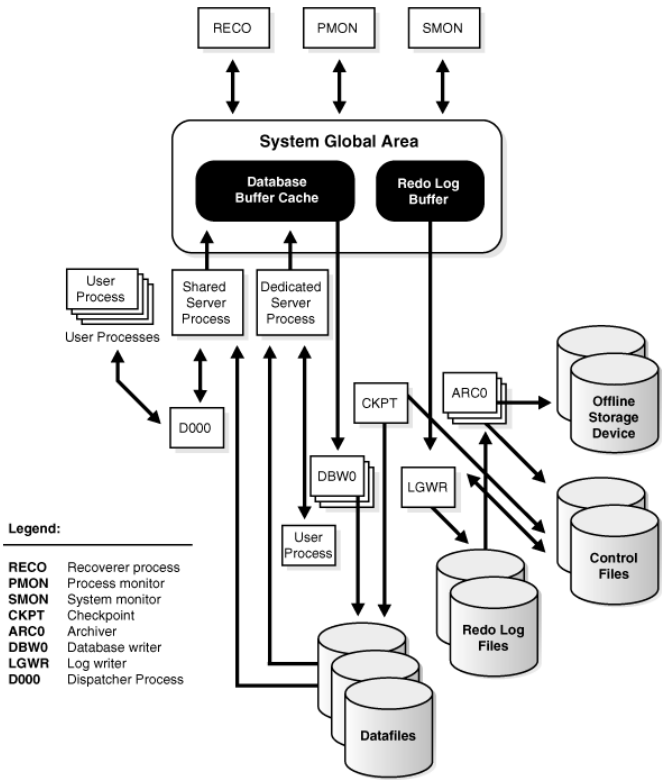
Az Oracle adatbázis szerver processzek segítségével kezeli le a felhasználói processzek kéréseit. Bizonyos esetekben, ha a felhasználói- és a szerver processz egy gépen fut, akkor lehetőség van a két processzt összevonni, hogy csökkentsük az interprocessz kommunikációból származó overheadet. Hálózaton keresztül történő kommunikáció esetén azonban mindenképpen szükség van két külön processzre.

Az egyes felhasználói alkalmazásokhoz rendelt szerver processzek a következő feladatokat láthatják el:

- elemzi és futtatja az alkalmazás által kiadott SQL utasításokat
- beolvassa a szükséges adatblokkokat az SGA osztott adatbázis bufferjeibe, ha azok még nincsenek bent
- olyan formában adja vissza az eredményeket, amiket az alkalmazás képes feldolgozni

8.3. Háttérfolyamatok

A teljesítmény maximalizálása és a felhasználók kezelése érdekében az Oracle adatbázis számos háttérfolyamatot (background process) futtat. Ezekről információkat a `V$BGPROCESS` nézetből nyerhetünk ki.



[21]

8.3.1. Archiver Processes (ARCn)

A rendszerben két redo log fájl található, egyszerre csak az egyiket írjuk. A két fájl cseréje után az éppen nem írás alatt álló fájlt ez a folyamat írja ki egy merevlemezre. Továbbá tranzakciókhoz kapcsolódó redo információk gyűjtésére is alkalmas, melyeket egy készenléti helyen tárol. A `LOG_ARCHIVE_MAX_PROCESSES` inicializálási paraméter segítségével korlátozhatjuk az archiváló folyamatok számát (aminek segítségével például nagy mennyiségű adat feltöltésekor keletkező jelentős archiválási munkát tudjuk szabályozni).

8.3.2. Checkpoint Process (CKPT)

Checkpointok előfordulásánál frissíti az adatfájlok (datafiles) fejlécét.

8.3.3. Database Writer Process (DBWn)

A bufferek tartalmának merevlemezekre írását végzi. Pontosabban a bufferekben található régen használt (*cold*) és módosított (piszkos - *dirty*) adatokat írja ki a merevlemezekre, illetve biztosítja, hogy a felhasználói processzek számára mindig legyen üres buffer, ahova adatblokkokat olvashatnak be. Alapvetően egy ilyen processz (**DBW0**) is elegendő, azonban többprocesszoros rendszerek esetén létrehozhatunk továbbiakat (maximum 20-t)

[21] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/process.htm#BEIFHCCJ`

`DB_WRITER_PROCESSES` inicializálási paraméter segítségével – de ezt egyébként is elvégzi az adatbázis a processzorok száma és processzorcsoportok alapján.

A piszkos buffereket a következő esetekben írjuk ki a merevlemezre:

- ha a szerver processz egy bizonyos számú buffer átvizsgálása után sem talált üreset, akkor jelez a DBWn-nek, aki aszinkron módon kiír néhány piszkos buffert a lemezre
- periodikusan ír ki buffereket, hogy újabb checkpointokat (az a pont a redo logban, ahonnan hiba esetén a visszaállítást kezdeni kell) érjen el. A checkpoint helyét a redo logban a buffer cache legöregebb piszkos bufferje határozza meg.

A jobb teljesítmény érdekében a DBWn kötegelve (*multiblock*) írja ki a blokkokat a merevlemezre.

8.3.4. Job Queue Processes

Kötegelt feldolgozás esetén használatosak. Gyakorlatilag egy ütemezőnek tekinthető, ami a megadott kezdeti idő és intervallum alapján megpróbálja végrehajtani a feladatot a megadott időintervallum előfordulásaikor. Egyszerre számos felhasználói feladatot képesek ütemezni. Működésük:

1. A koordinátor processz (CJQ0) periodikusan kiválasztja a rendszer JOB$ táblájából a futtatandó feladatokat. Az újonnan kiválasztott feladatokat idő szerinti sorrendbe helyezi.
2. A CJQ0 dinamikusan létrehoz Job Queue szolga processzeket (J000...J999) a feladatok futtatására.
3. A Job Queue processz lefuttatja az egyik CJQ0 által kiválasztott feladatot.
4. A végrehajtott feladat után a processz újabb feladatért jelentkezik (polling). Ha nincs más végrehajtandó feladat, akkor átmegy alvó üzemmódba, ahonnan periodikus időközönként felébred és kér újabb feladatokat. Ha bizonyos ideig nem talál új feladatot, akkor a processz leáll.

A Job Queue processzek maximális számát a `JOB_QUEUE_PROCESSES` inicializálási paraméter segítségével állíthatjuk be.

8.3.5. Log Writer Process (LGWR)

Ez a folyamat végzi a redo log bufferek tartalmának kiírását a redo log fájlokba. A redo log buffer egy körkörös buffer, azaz amint az LGWR kiírta a redo bejegyzéseket a redo log fájlokra, a szerver processzek felülírhatják a már kiírt bejegyzéseket az újakkal. Az LGWR általában elég gyors ahhoz, hogy még nagy terhelés esetén is mindig biztosítson szabad buffereket az új bejegyzések számára.

Ír a redo log fájlokba, ha:

- egy felhasználói processz jóváhagy (commit) egy tranzakciót, egy commit bejegyzést.
- redo log buffereket: 3 másodpercenként; ha 1/3ig megteltek; mielőtt a DBWn kiírja a piszkos buffereket.

Az LGWR egyidejűleg ír több aktív redo log fájlba. Ha az egyik fájl sérül, vagy elérhetetlenné válik, akkor folytatja a többi fájlba történő írást, s logolja a hibát az LGWR trace fájljában és a system alert logban. Ha minden fájl sérült, vagy elérhetetlen, mert még nem archiválták, akkor az LGWR nem tud tovább működni.

Ha egy felhasználó kiad egy `COMMIT` utasítást, akkor az LGWR egy commit bejegyzést tesz a redo log bufferbe, s azt a tranzakció redo bejegyzéseivel együtt azonnal a merevlemezre írja. Az adatblokkok kiírása azonban nem feltétlen azonnal - jellemzően egy másik, hatékonyabb időpontban történik. Ezt az eljárást hívjuk *fast commit*-nak, mivel az adatbázis ugyan visszajelzi a tranzakció jóváhagyásának sikerességét, de az új adatokat még nem rögzítettük a merevlemezen. A jóváhagyást követően a tranzakcióhoz rendelünk egy *system change number (SCN)*-t is, amely RAC-ok és elosztott adatbázisok esetén a szinkronizált visszaállításnál elengedhetetlen.

Nagy terhelés esetén az I/O műveletek csökkentése és a teljesítmény növelése érdekében az LGWR több commitot egyszerre, úgynevezett *group commit*-ként is ki tud írni.

8.3.6. Process Monitor Process (PMON)

Amikor egy felhasználói processz sikertelenül ér véget, ő végzi a processz helyreállítását: kitakarítja az adatbázis buffer cacheét, s felszabadítja a felhasználói processz által használt erőforrásokat. Periodikusan ellenőrzi az ütemező és a szerver processzek állapotát is, s újraindítja őket, ha szükséges. Ezen kívül információkat az adatbázis példányról, valamint a hálózati listenerek ütemező processzeiről. Működése hasonló az SMON-éhoz: periodikusan ellenőrzi, hogy szükség van-e rá, s működésbe lép, ha egy másik processz igényli.

8.3.7. Queue Monitor Processes (QMNn)

Opcionális háttérfolyamat az Oracle Streams Advanced Queuing számára, mely az üzenetsorokat monitorozza. Maximálisan 10 ilyen sorfelügyelő processzt állíthatunk be. Hasonlóan a job queue processzekhez, ő is különbözik abban a többi háttérfolyamattól, hogy meghibásodása nem okozza az adatbázis példány meghibásodását.

8.3.8. Recoverer Process (RECO)

Elosztott adatbázisok esetén használatos háttérfolyamat, mely az elosztott tranzakciók hibáit kezeli. Bizonytalan elosztott tranzakció esetén a RECO automatikusan csatlakozik a másik adatbázishoz, s miután helyreállította a kapcsolatot az adatbázis szerverek között, eltávolítja az adott tranzakcióknak megfelelő sorokat mindegyik adatbázis várakozó (pending) tranzakciós táblájából. Ha nem sikerül a RECO-nak helyreállítania a kapcsolatot a másik adatbázissal, exponenciálisan növő időközönként újra próbálkozik.

8.3.9. System Monitor Process (SMON)

Az adatbázis példány indítását követő helyreállításokat végzi, ha erre szükség van. Feladatai köze tartozik még az ideiglenes szegmensek használat utáni kitakarítása, valamint a könyvtár vezérelt táblaterek (dictionary managed tablespaces) összefüggő üres extentjeinek összeolvasztása. Ha a példány helyreállítása során fájlolvasási vagy offline hiba miatt bármely lefutott tranzakciót ki kellett hagyni, akkor az SMON ezt később azonnal újra próbálja, ha az adott táblatér (tablespace) vagy fájl ismét elérhető (online) lesz. Hasonlóan a PMON-hoz, periodikusan ellenőrzi, hogy szükség van-e rá, meghívta-e egy másik processz.

8.3.10. Egyéb Oracle adatbázis háttérfolyamatok (röviden)

- *ACMS (automatic controlfile to memory service):* Oracle RAC környezetben használatos. Biztosítja az elosztott SGA frissítését globális commit/abort esetén.
- *ASMB:* a kommunikációt látja el az Automatic Storage Management példánnyal.
- *DBRM (database resource manager):* erőforrás tervezést vagy más erőforrás menedzser taszkok létrehozását végzi.
- *DIA0 (diagnosability process 0):* holtpont detektálás és feloldás a feladata.
- *DIAG (diagnosability):* diagnosztikai memóriakiíratás és globális "oradebug" parancsok futtatása.
- *EMNC (event monitor coordinator):* adatbázis eseménykezelés és értesítések.
- *FBDA (flashback data archiver process):* flashback adatarchívumok menedzselése.
- *GMON:* karbantartja az ASM diszkcsoportjainak taglistáját.
- *GTX0-j (global transaction):* Oracle RAC környezetben használatos XA globális tranzakcióknak nyújt transzparens támogatást.
- *KATE:* ASM háttérfolyamat, amely végrehajtja a proxy I/O-t egy ASM metafájlon, ha egy merevlemez offline lesz.
- *MARK:* megjelöli az ASM allokációs egységeit egy offline lemezre történő sikertelen írási kísérletet követően.
- *MMAN:* belső adatbázis taszkok számára.
- *MMNL:* a gyakori és jelentéktelenebb menedzseléssel kapcsolatos feladatokat látja el.
- *MMON:* különböző menedzseléssel kapcsolatos háttérfeladatok végrehajtása.
- *ARBn:* egy ASM példányon belüli aktuális kiegyenlítő extent mozgatásokat végzi.

- *PSP0 (process spawner):* Oracle processzek létrehozása.
- *RBAL:* egy ASM példányon belül a diszkcsoportok közötti kiegyenlítést koordinálja.
- *SMCO (space management coordinator):* tárhely kezeléssel kapcsolatos taszkok koordinálása. Dinamikus létrehoz szolga processzeket (Wnnn) a taszkok implementálására.
- *VKTM (virtual keeper of time):* felel a falióra idejének (másodpercenkénti frissítés) és a referenciaidő számlálónak (20ms-enkénti frissítés) a karbantartásáért.

8.4. Trace fájlok és Alert log

Az Oracle 11g újdonságaként a problémák megelőzésére, észlelésére, diagnosztizálására és megoldására egy fejlett hibadiagnosztizáló infrastruktúra lett a rendszerbe beépítve. Főként a kritikus hibák észlelése volt a cél, amiket például adatbázis programhibák, meta- vagy ügyféladat sérülések okozhatnak.

Kritikus hiba fellépése esetén egy incidens számot rendelünk a hibához és a hozzá tartozó diagnosztikai adatokat (pl. trace fájlok) azonnal begyűjtjük és megjelöljük ezzel az incidens számmal. Az adatokat az Automatic Diagnostic Repositoryban (ADR) – adatbázison kívüli, fájl alapú tárhely – mentjük el, ahonnan később az incidens szám alapján visszakereshető és analizálható a hiba.

Minden szerver- és háttérfolyamat belső hiba észlelése esetén kiírja a hibához kapcsolódó információkat a saját **trace fájl**jába. Továbbá a háttérfolyamatok kiegészítő információkat is írhatnak a trace fájlba, ami segítheti az alkalmazás vagy az adatbázis példány hangolását.

Minden adatbázishoz tartozik egy **alert.log fájl** is, mely időrendi sorrendben tartalmazza a következő üzeneteket és hibákat:

- Minden belső hiba (ORA-600), blokk meghibásodás hiba (ORA-1578) és holtpont hiba (ORA-60).
- Adminisztratív műveletek, úgy mint `CREATE`/ `ALTER`/ `DROP DATABASE`/ `TABLESPACE` SQL utasítások, valamint az Enterprise Manager/SQL*Plus `STARTUP`/ `SHUTDOWN`/ `ARCHIVE LOG`/ `RECOVER` utasításai.
- Különböző elosztott szerver és ütemező folyamatok működéséhez kapcsolódó üzenetek és hibák.
- Materializált nézeteket automatikus frissítése során fellépő hibák.

8.5. Elosztott szerveres architektúra

Az elosztott szerveres architektúrában nem szükséges minden kapcsolathoz dedikálnunk egy-egy külön szerver processzt. Az ütemező a bejövő kéréseket az elosztott szerverfolyamatok készletéhez irányítja, ahonnan egy éppen tétlen folyamat fogja a kérést lekezelni.

Így tulajdonképpen néhány elosztottan működő szerverrel tudjuk ugyanazt a teljesítményt produkálni, mint sok dedikált szerverrel együtt. Ezen felül mivel így az egy felhasználó számára szükséges memória is relatív kicsi, kevesebb memória és processz menedzsment szükséges, s egyidejűleg több felhasználót tudunk kiszolgálni.

Elosztott szerveres rendszerek esetén a következő processzekre van szükség:

- egy hálózati listener processz, amely létrehozza a kapcsolatot a felhasználói processzek és az ütemezők vagy a dedikált szerverek között.
- egy vagy több ütemező processz
- egy vagy több elosztott szerver processz

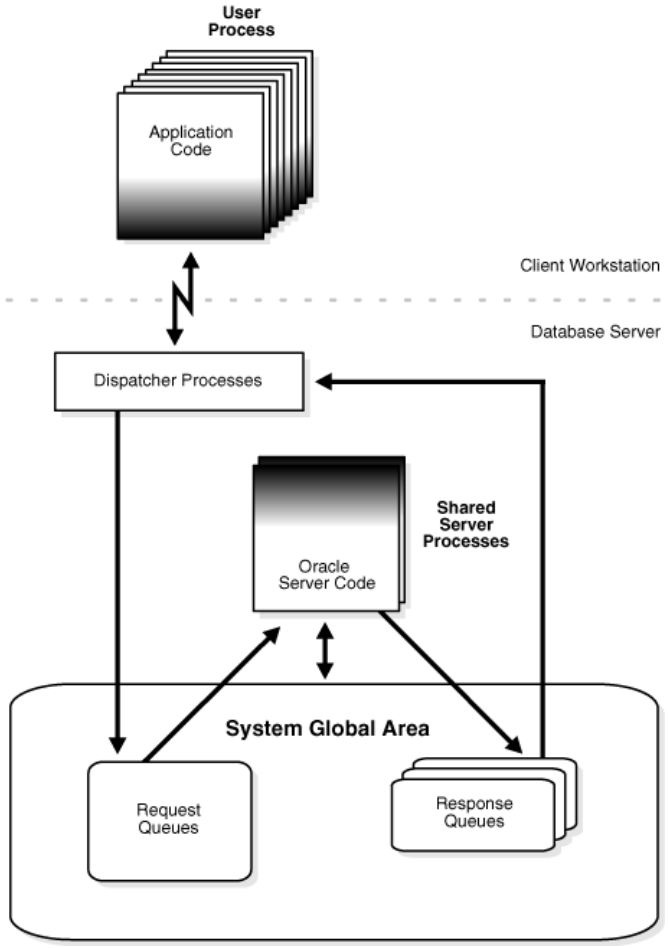
Az adatbázispéldány indulását követően a listener processz létrehozza a kommunikációs portot, amin keresztül a felhasználók csatlakozhatnak az adatbázishoz, majd minden ütemező processz megadja a listernek azt a címet, amin várja a kapcsolat-felépítési kérelmeket. Használt hálózati protokollonként szükség van legalább egy megfelelően konfigurált ütemező processzre.

A felhasználói processz kapcsolat-felépítési kérelme után a listener megvizsgálja, hogy a felhasználó processze használhat-e elosztott szerver processzt. Amennyiben igen, úgy a listener visszaadja számára a legkevésbé terhelt ütemező címét, amihez aztán a felhasználói processz közvetlenül csatlakozhat. Néhány felhasználói processz azonban nem képes az ütemezővel kommunikálni, dedikált szerverre van szüksége. Ebben az esetben a listener létrehozza a dedikált szervert és felépíti a megfelelő kapcsolatot.

8.5.1. Ütemező kérés- és válaszsorai

Egy felhasználói hívást követően az ütemező a kérést a kéréssorba (**request queue**) helyezi, ahonnan a következő rendelkezésre álló elosztott szerver processz azt kiveszi. A kéréssort az SGA-ban tároljuk. Minden adott adatbázispéldányhoz tartozó ütemezőnek egy közös kéréssora van, ahonnan a szabad elosztott szerver processzek a kéréseket FIFO módon szedik ki. A kérés kiszolgálását követően a kiszolgálást végző elosztott szerver processz a választ a hívást kezdeményező ütemező saját válaszsorába helyezi el (**response queue**). Minden ütemező saját válaszsort tart fenn az SGA-ban, s innen továbbítja a teljesített kérésekre kapott választ a megfelelő felhasználói processznek.

A kérés kiszolgálását követően tehát a felhasználó kapcsolatban maradhat, azonban nem kell számára egy külön processzt továbbra is fenntartani. A kérést kiszolgáló processz miután a választ elhelyezte a megfelelő válaszsorban, hozzáfoghat új, akár más felhasználóktól származó kérések kiszolgálásához is:



[22]

8.5.2. Ütemező folyamatok (Dnnn)

Az ütemező folyamatok teszik lehetővé az elosztott szerveres konfiguráció számára, hogy a felhasználói folyamatok néhány limitált számú szerverfolyamaton osztozzanak. Ezáltal mivel kevesebb szerverfolyamatra van szükség, mint felhasználói folyamatra, egyidejűleg jóval több felhasználó szolgálható ki.

Egy adatbázis példányhoz akár több ütemező folyamatot is létrehozhatunk, az adatbázis által használt hálózati protokollonként egy azonban mindenképpen szükséges. Ütemező folyamatokat akár az adatbázis futása közben is létrehozhatunk vagy eltávolíthatunk.

Elosztott szerveres beállítások esetén a listener processz fogadja a felhasználói alkalmazásoktól beérkező kapcsolat-felépítési kérelmeket, s irányítja őket az ütemezőkhöz. Ha az alkalmazás nem képes ütemezőhöz csatlakozni, akkor a listener létrehoz számára egy dedikált szerver processzt. A listener processz nem az Oracle adatbázispéldány része, hanem az adatbázissal együttműködő hálózati processzekhez tartozik.

[22] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/process.htm#BEIDDFDG`

8.5.3. Elosztott szerverfolyamatok (Snnn)

Elosztott szerveres architektúra esetén minden szerverfolyamat több klienst szolgálhat ki (de nem egyidejűleg). Funkcionalitását tekintve nincs különbség dedikált és elosztott szerverfolyamatok között, csupán annyiban térnek el, hogy az elosztott szerverfolyamatok nincsenek egy megadott felhasználói folyamathoz hozzárendelve, hanem bármelyik kliens kéréseit kiszolgálhatják. Ebből kifolyólag az elosztott szerverfolyamatok PGA-ja nem is tartalmaz semmilyen felhasználói információt, csak egy veremet és processz-specifikus változókat. A sessionökhöz tartozó információkat az SGA-ban tároljuk, így minden szerverfolyamat által hozzáférhetőek. Adott sessionökhöz tartozó terület méretét a `PRIVATE_SGA` paraméter beállításával limitálhatjuk.

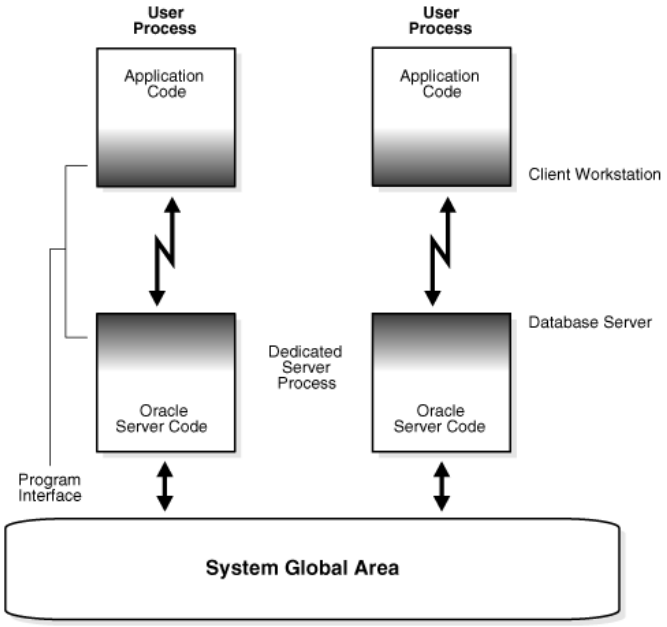
A szerverfolyamatok számát az adatbázis dinamikusan állítja egy, a `SHARED_SERVERS` és `MAX_SHARED_SERVERS` inicializálási paraméterek között megadott értékre a kéréssor hosszának függvényében.

8.5.4. Az elosztott szerver korlátozott műveletei

Bizonyos adminisztratív feladatok (pl. adatbázis leállítása, indítása, adathordozó helyreállítás) nem hajthatóak végre, ha ütemező folyamathoz csatlakozunk – hibaüzenetet fogunk kapni. Ezért ha adminisztrátorként csatlakozunk egy adatbázishoz, célszerű a connection stringben explicit megadni, hogy dedikált szerverfolyamatot (SERVER=DEDICATED) szeretnénk használni.

8.6. Dedikált szerveres architektúra

Dedikált szerveres architektúra esetén minden felhasználói folyamathoz külön szerverfolyamatot rendelünk:



[23]

Ezeket a processzeket hívjuk dedikált szerverfolyamatoknak, mivel csak a hozzá tartozó felhasználói folyamat nevében cselekszenek. Mindig ugyanannyi szerverfolyamatunk lesz, mint amennyi felhasználói - a szerverfolyamat akkor is megmarad inaktívként, ha a hozzá tartozó felhasználói folyamat éppen nem intéz kérést az adatbázis felé.

Bizonyos operációsrendszerek esetén (pl. UNIX) akkor is szükség van külön szerverfolyamatokra, ha a kliensalkalmazás és a szerver ugyanazon a gépen fut, ugyanis ezek az operációsrendszerek nem lennének képesek külön kezelni a két programot, ha egy közös processz tartozna csak hozzájuk.

8.7. Database Resident Connection Pooling

DRCP[24]

A DRCP egy connection pool[25]-t nyújt tipikusan web alkalmazások számára. Különösen hasznos olyan többprocesszes, egyszálas alkalmazásszerverek (pl. PHP és Apache szerverek) skálázhatóságának javításában, melyek nem képesek középső rétegbeli connection poolingra. A webes alkalmazások egy szálon, általában csak rövid ideig használják az adatbázis-kapcsolatot. Ennek támogatására a DRCP tulajdonképpen azt teszi lehetővé, hogy az egyes processzek közösen osztozzanak a dedikált szervereken, azaz nem kell minden egyes kapcsolatot újra kiépíteni, aminek következményeként nagyszámú klienskapcsolatot tudunk kiszolgálni jóval szerényebb erőforrásokkal is (csökkenti a szükséges memóriát, növeli az adatbázisszerver és a középső réteg skálázhatóságát, valamint csökkenti újbóli kapcsolat-felépítésekhez szükséges időt).

[23] http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/process.htm#BEIJAEDA
[24] http://www.oracle.com/technology/tech/oci/pdf/oracledrcp11g.pdf
[25] http://en.wikipedia.org/wiki/Connection_pool

A pooled szerver modell nagyban hasonlít az Oraclenél alapértelmezésben használt dedikált modellhez. A különbség csupán annyi, hogy csökkenti a szervert csak rövid ideig igénylő kapcsolatoknál a szerverdedikálásból fakadó overheadet. A kliensek a "connection broker"-hez csatlakoznak, ami a pool működését implementálja és multiplexálja a pooled szervereket a kliens processzektől bejövő kapcsolatok között (ábra[26]). Ha egy kliens adatbázis-műveleteket szeretne végrehajtani, akkor a connection broker kiszed a poolból egy pooled szervert, s hozzárendeli a klienshez. Innentől a kliens már közvetlenül csatlakozik a pooled szerverhez, melynek a működése teljesen azonos lesz egy dedikált szerverével. A kérés kiszolgálását követően azonban a szerver visszakerül a pool-ba, s a kliens is visszacsatlakozik a connection brokerhez.

DRCP használatához az adatbázis adminisztrátorának explicit el kell indítania a poolt. Az alapértelmezett connection pool-t `SYS_DEFAULT_CONNECTION_POOL`-nak nevezik, így ennek elindítása SYSDBA-ként bejelentkezve a következő paranccsal hajtható végre:

```
EXECUTE DBMS_CONNECTION_POOL.START_POOL('SYS_DEFAULT_CONNECTION_POOL');
```

A megosztott pool-hoz történő csatlakozáshoz továbbá a server típusát is `POOLED`-ra kell állítani a connection stringben. Erre egy példa:

```
ServerPool = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=somehost)
(PORT=1521))(CONNECT_DATA=(SERVICE_NAME=testdb)(SERVER=POOLED)))
```

Vagy egyszerű connect parancs használatával:

```
CONNECT joeuser@myhost.mydomain.com:1521/mydb:POOLED
```

8.7.1. Kapcsolatosztályok

Logikai neveket definiálnak különböző alkalmazások által igényelt kapcsolattípusokra. Két különböző felhasználó nem oszthat meg egymás között kapcsolatot vagy sessiont. Továbbá lehetőség van egy felhasználón belül az alkalmazások közötti elkülönítésre is. A DRCP biztosítja, hogy egy kapcsolatosztályhoz tartozó session nem osztozkodik a kapcsolatosztályon kívüli sessionökkel.

8.7.2. Session Purity

Meghatározza, hogy az alkalmazás egy teljesen új sessiont igényel-e (`PURITY=NEW`), vagy pooled sessiont kívánja használni (`PURITY=SELF`). Utóbbi esetben az alkalmazás egy igényelt kapcsolatosztályú szabad sessiont kap.

A kapcsolatosztályokat és a session purityt a kliens a DRCP kapcsolat attribútumaiban határozhatja meg. Alapértelmezésben a kapcsolatosztály értéke *username*.`SHARED`, illetve a purity értéke `NEW`. Ezek azonban alkalmazások esetén eltérhetnek, ezért célszerű az alkalmazás manualjében utánanézni.

[26] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/process.htm#CIHDHHEF`

8.8. Program interfész

A program interfész egy szoftver réteg az alkalmazás és az Oracle adatbázis között, mely a következőket végzi:

- biztonsági falat képez, amely megakadályozza a felhasználói processzek destruktív hozzáférését az SGA-hoz.
- kommunikációs mechanizmusként viselkedik: formázza az információkéréseket, továbbítja az adatokat, valamint elkapja és visszaadja a hibákat.
- konvertálja és lefordítja az adatot, különösen különböző típusú számítógépek között, vagy külső felhasználói adattípusok számára.

Bővebben: link[27]

[27] http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/process.htm#i18680

Hivatkozások

[1] Oracle Library - Concepts
http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/part_2.htm#sthref167
2008. május 5., 7:42

[2] http://www.oracle.com/technology/oramag/oracle/03-sep/o53business.html
2008. május 5., 7:43

[3] Oracle Library - Data Warehousing Guide
http://download.oracle.com/docs/cd/B28359_01/server.111/b28313/basicmv.htm#i1006519
2008. május 5., 7:44