Beugrók:

1. Tranzakció formális leírása

Tranzakciónak nevezzük, azt a
$T=(T_e,\rightarrow)$
párt, melyben Te a műveletek halmaza és → a megelőzési reláció, azaz
$T_e = \{r(x), w(x'), a, c \mid x,x' \in DB\}$
$\rightarrow \subseteq T_e \times T_e$
ahol DB jelenti az adatbázisbeli objektumok halmazát, úgy hogy a T
tranzakcióra teljesülnek a következők:

- $a \in Te \Leftrightarrow c \notin Te$
- $\forall p \in Te \Rightarrow p \rightarrow a$, ha $a \in Te$ és $p \rightarrow c$, ha $c \in Te$
- $\forall p1, p2 \in Te$, p1 és p2 konfliktusban áll $\Rightarrow p1 \rightarrow p2$ vagy $p2 \rightarrow p1$

2. Meg volt adva egy Árú tábla, elemei: Áru(vonalkód,…,ár,szavatosság) (a 2. mező nem rémlik de nem is kellett) és az volt a lényeg hogy olyan függvényt kellett írni ami visszaad hogy mennyi olyan árú van ami a paraméterként megadott árnál kisebb.

```
CREATE OR REPLACE FUNCTION szamlalo(a aru.ar%TYPE) RETURN NUMBER IS
      db NUMBER;
BEGIN
      SELECT count(*) INTO db FROM aru WHERE ar<a;
      RETURN db;
END;
```

3. Egy triggert kellett létrehozni ami azt nézi hogyha egy lejárt szavatosságú áru kerül az adatbázisba azt egy naplo fájlba mentse el.

```
CREATE OR REPLACE TRIGGER arutrigger AFTER INSERT ON aru FOR EACH ROW
BEGIN
      IF :NEW.ar<SYSDATE THEN
            INSERT INTO naplo VALUES('Lejárt szavatosságú áru:  ' ||
            :NEW.vonalkod , USER, SYSDATE);
      END IF;
END;
```

4. Kapcsolódás JDBC-vel.

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
   Connection c = DriverManager.getConnection("jdbc:oracle:thin:username/password@host:port/info");
   c.close();
}
catch (Exception e)
{
   System.out.println(e);
}
```

5. Sorted Merge módszere és költsége

A sorted-merge algoritmus esetén előbb rendezik mindkét táblát a kapcsolódó mezők alapján. Ezután a merge módszerhez hasonlóan egy-egy pointer vezetnek be a két táblához, indulásként mindkettőt a megfelelő tábla elejére pozicionálva. Ezután megnézik, hogy a pointerek által kijelölt rekordok illeszkednek-e. Ha nem, akkor valamelyik kisebb értékű, mint a másik. A továbblépéshez a kisebb értékre mutató pointert mozgatják eggyel előre, miáltal az az előzőnél nagyobb vagy ismét vele egyenlő értékre fog mutatni. E két új elemre újból elvégezzük az illeszkedés vizsgálatot. Ha a két érték illeszkedi, akkor kiírjuk őket az eredmény táblába, majd továbblépünk. A továbblépésnél figyelni kell azonban arra, hogy egyetlen egy lehetséges párosítást sem vétsünk el. Ez abban az esetben jelenthet problémát, ha mindkét táblában több azonos kulcsértékű rekord tárolódik. Így ezen rekordok mindegyik párosítását be kell hozni az eredmény táblába. Ennek megoldására egy

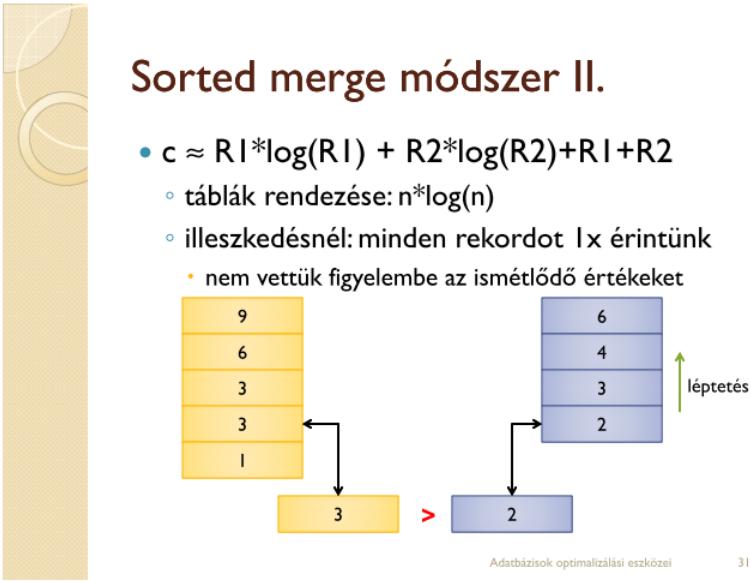
kis nested-loop algoritmust kell megvalósítani az azonos értékű rekordok mindennemű párosítására.Az algoritmus költségében a legnagyobb tételt a táblák rendezése jelenti. A rendezés optimális költsége, mint ismert az n*log(n) hatékonyságosztályba esik. Mivel a rendezés után az illeszkedés vizsgálatnál minden rekordot nagyjából egyszer érintjük, így a join költsége

$$c \approx R1*\log(R1) + R2*\log(R2) + R1 + R2$$

alakú lesz. A valós költség ettől mindíg nagyobb, hiszen itt nem vettük figyelembe a többszörös érték-előfordulás hatását az illeszkedés vizsgálatnál. Ha a költséget az összehasonlítások műveletére vonatkoztatjuk, akkor kimarad a rendezésre vonatkozó rész:
$c \approx R1 + R2$

Előadásból (gondolom 3 pontért nem regényt kért volna):

11. előadás 30. oldala



11. előadás 31. oldala

6. Cursor kezelés.

A kurzor létrehozása a PL/SQL nyelvben a deklarációs részben, s nem a műveleti részben történik, mint a beágyazott SQL esetén. A kurzor deklarációjának formátuma:

DECLARE

...

CURSOR kurzornév (paraméterlista) IS SELECT_utasítás;

Az utasításban a zárójelek között megadott paraméterlista opcionális. A deklarációs utasításban egy azonosító nevet rendelünk a kurzor szerkezethez, s e név segítségével hajtjuk végre a lekérdezést, s e név segítségével

férhetünk hozzá az eredményhez is. A kurzor a PL/SQL nyelvben paraméteresen is deklarálható. A paraméterek a kapcsolódó SELECT utasításban kerülnek felhasználásra. A paraméterlista a paraméter azonosító neve mellett a paraméter típusát is tartalmazza. A listaelemek vesszővel vannak elválasztva egymástól. A PL/SQL szabályai szerint minden felsorolt paraméternek meg kell jelennie a kapcsolódó SELECT utasításban is. A SELECT utasításnak itt a normál, INTO opció nélküli formátuma használható csak. A következő példában egy kurzort deklarálunk a paraméterként megadott dátum előtt született személyek nevének és lakcímének a lekérdezésére:

```
DECLARE
    ...
    CURSOR lista (datum DATE) IS SELECT nev, lakcim
            FROM szemelyek WHERE szuldat < datum;
```

A kurzor megnyitása után a kapott adatokat egyenként lekérdezhetjük, s módosíthatjuk is. Módosítás esetén azonban a kurzor deklarációjánál jelezni kell, hogy nemcsak olvasásra hozzuk létre a kurzort. A módosítási igényt a SELECT utasítás végén álló, már ismert

```
FOR UPDATE OF mezőlista
```

opcióval jelezzük. A kurzor által visszaadott rekordot a

```
CURRENT OF kurzornév
```

feltétellel jelölhetjük a megfelelő UPDATE utasításban.

A kurzor megnyitása, vagyis a kijelölt lekérdezés elvégzése az

```
OPEN kurzornév (aktuális paraméterlista);
```

utasítással lehetséges, ahol a zárójelek között megadott paraméterlista opcionális elem. A paraméterlista most konkrét értékeket tartalmaz, melyek sorra behelyettesítődnek a deklarációkor kijelölt formális paraméterekbe, s ezen értékekkel fog a lekérdezés végrehajtódni. Ugyanaz a kurzor többször is végrehajtható különböző aktuális paraméterértékek mellett.

Az eredményrekordok egyenkénti lekérdezésére a

```
FETCH kurzornév INTO változólista;
```

utasítás szolgál. A változólistának vagy annyi elemi változót kell tartalmaznia, ahány elemű az eredménytáblázat, vagy olyan rekordváltozót kell megadni, melynek struktúrája megegyezik az eredménytábla struktúrájával. A vizsgált PL/SQL változatban a kurzorpointer csak előre léptethető, mégpedig egy rekorddal.

A kurzor felhasználása után célszerű a kurzor által lefoglalt erőforrásokat felszabadítani.

Ehhez ki kell adni a

```
CLOSE kurzornév;
```

utasítást.

Az eredménytábla több rekordot is tartalmazhat, ezért a FETCH utasítást többször egymásután is ki kell adni a teljes választábla feldolgozásához. Az eredménytábla végének figyelését, vagyis annak ellenőrzését, hogy az összes rekordot érintettük-e már, egy kurzor attribútum segítségével végezhetjük el. Az attribútum, melynek alakja

```
kurzornév%NOTFOUND
```

akkor tartalmaz igaz értéket, ha elértük az eredménytábla végét, s nincs már további feldolgozásra váró rekord a kurzorban. A lekérdező ciklus, melynek magjában a megfelelő FETCH utasítás áll, egy alap LOOP ciklus segítségével is megoldható, melybe a kilépéshez beteszünk egy

```
EXIT WHEN kurzornév%NOTFOUND;
```

utasítást is.

Ebből is kérdéses mire is gondolt 2-3 pontért. Én leírtam hogy hozom létre, hogy nyitom meg, leírtam hogy egy ciklusban végigmegyek minden rekordon (ide lehet a FETCH kellett volna) és leírtam hogy zárom le. Erre kaptam fél pontot…