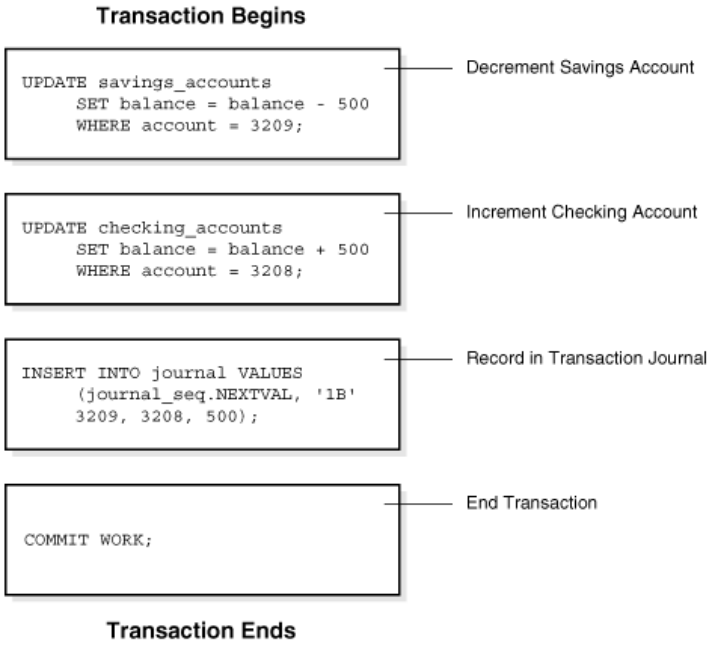
11. gyakorlat

I. Tranzakció kezelés a gyakorlatban

1. Ismétlés

Mi is az a tranzakció? A tranzakció egy vagy több SQL utasítást tartalmazó atomi egység. Ez azt jelenti, hogy a tranzakción belüli minden SQL utasítás „eredményes” („committed”) vagy minden utasítás visszavonásra kerül (pl. insert). Az **1. ábra** egy egyszerű banki tranzakciót mutat be, egyik számláról (account=3209) átrakunk pénzt egy másikra (account=3208). Ha a tranzakció sikeres, akkor minden utasítás végrehajtódik.



1. Ábra: példa egyszerű tranzakcióra. Forrás: http://docs.oracle.com/cd/B28359_01/server.111/b28318/transact.htm

A tranzakció végét a **`COMMIT`** paranccsal jelezzük. A **`ROLLBACK`** paranccsal lehet visszavonni a végrehajtott változásokat. Az oracle-ban van lehetőség nevesített köztes mentési pontokat definiálni, azaz nem muszáj a teljes tranzakciót eldobni, ha probléma van. Pl. **`savepoint sp1`**

2. Feladatok:

a) Hozzunk létre egy `person` táblát (`id, name`), majd töltsük fel random adatokkal. Commitoljuk a feltöltést.

b) Töltsünk be a táblába 3 új személyt minden egyes `INSERT` után egy `SAVEPOINT`ot létrehozva.

c) Az utolsó előtti nevet rosszul töltöttük fel, vonjuk vissza (`ROLLBACK`) a 2. savepointig az eredményeket.

3. Tranzakció kezelőn kívül eső műveletek:

a) CTAS (vs. CREATE TABLE + INSERT INTO páros)
   - Egyszerűbb, mint a create+insert, mert nem kell a create-et megírni.
   - Gyorsabb, mert nem kell a constraintekre figyelni, mint a sima insertnél (hiszen az alaptáblába beszúráskor ez már megtörtént, elég a végén rárakni)
   - Példa:

```
create table le_tmp_term_copy as
select *
from   go13.term
```

b) TRUNCATE (vs. DELETE FROM) bemutatása
   - truncate ddl utasítás
   - minden sort töröl
   - sokkal gyorsabb, mint a delete, mert triggerek, indexek... nem lassítják
   - jobb, mint a drop+recreate mert nem kell minden grantot, indexet... újracsinálni
   - mivel mindent töröl, nem mindig használható

4. Tranzakciók az oracle-ben

Kétféle izolációs szint használható az oracle-ben:

a) *Read commited* (default): Ha az adott tranzakció olyan DML utasítást tartalmaz, ami LOCK-olt sort tartalmaz (amit pl. egy másik tranzakció módosítja az adott sort), akkor a tranzakció várakozik a LOCK feloldásáig. Lehetséges phantom és non-repeatedable read-eket létrehozni.
b) *Serializable*: Ahogy a SQL92 szabványban megvan adva, azaz ha tranzakcióban olyan adatot szeretnék módosítani, amit egy másik tranzakcióban módosításra került, akkor az utasítás sikertelen lesz.

Valamint be lehet állítani, hogy az az adott tranzakció `READ ONLY`. Ez azt jelenti, hogy a tranzakcióban minden egyes query (`SELECT …`) csak azokat a változásokat látja, amik a tranzakció megkezdése előtt commitolva voltak.

Tranzakció megadása:

a) Új tranzakció kezdődik implicit módon a commit parancs után.
b) SET TRANSACTION paranccsal explicit módon meg lehet adni a tranzakció kezdetét. Pl. `SET TRANSACTION ISOLATION LEVEL READ COMMITTED; SET TRANSACTION READ ONLY;`
c) A teljes session-re be lehet állítani az izolációs szintet: `ALTER SESSION SET ISOLATION_LEVEL = READ COMMITTED;`

Példa:

```
COMMIT;
SET TRANSACTION READ ONLY NAME 'Toronto';
SELECT product_id, quantity_on_hand FROM inventories
   WHERE warehouse_id = 5
   ORDER BY product_id;
COMMIT;
```

5. **Feladatok. A feladatok megoldásait dokumentáld (sql-kód + screenshot)!**
   a) A szomszédod segítségével hozzál létre egy egyszerű bank adatbázist, aminek segítségével számlaegyenleg-változásokat lehet modellezni. A partnerednek az alábbi módon tudsz hozzáférést biztosítani a sémádban lévő tábládhoz:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON [tabla_nev] TO
PUBLIC;
```

   b) A partnered segítségével készíts egy olyan tranzakció sorozatot, amivel a következő jelenségeket sikerül produkálni:
      a. PHANTOM READ
      b. NON-REPEATABLE READS

      Az eredményeket részletesen dokumentáld! Milyen izolációs szintet kell beállítani ehhez?

   c) Tervezzetek egy olyan szcenáriót, amikor egy *read-only* tranzakciónak van értelme!

II. Optimalizálás – Indexek használata

Az indexek olyan (B-fa vagy hash alapú) adatszerkezetek, amik jelentősen meggyorsíthatják az egyes lekérdezéseket.

Indexek különösen hasznosak:

a) Teljes egyezések keresésénél (exact match)
b) Tartományok átfésülésénél (range query)
c) Szelektív lekérdezéseknél
d) Join esetében (join condition)
e) Rendezések esetében (sorted result)

Létrehozásuk:

```
CREATE INDEX index_neve ON táblaneve(attr1,atrr2….)
```

Feladatok:

Töltsd le a tárgy wiki oldaláról a Gyak11_01.sql fájlt, majd ezt a szkriptfájlt futtasd le az sql developerben! A feladat eredményeit (lekérdezési tervekhez – `explain plain-hez -`) tartozó screenshot-okat és az sql kódokat dokumentáld (pl. google docs segítségével) és ezeket küldd el pdf formtáumbam az adatbazismb2015@gmail.com email címre.

1. **Feladatsor**
   a) Kérdezd le az A és B betűvel kezdődő neveket.
   b) Készíts screenshotot a lekérdezési tervről és mentsd is le valahová (Explain plan).

c) Készíts egy student_name_ix indexet name indexelésével.
d) Készíts újra egy screenshotot a lekérdezési tervről. Hasonlítsd össze az index előttivel, mit tapasztalsz?
e) Melyik kategóriába esik a fenti lekérdezés? (exact match, range query, stb.)
f) Kérdezd le azokat a neveket, amelyekben szerepel az „an” sztring! Vizsgáld meg a lekérdezési tervet, mit tapasztalsz? Lehet-e itt jó indexet építeni?
g) Listázd ki azokat a hallgatókat, akik Strasbourgban laknak és a nevük A betűvel kezdődik! Vizsgált meg a lekérdezési tervet!
h) Készíts egy indexet, ami a fenti lekérdezést gyorsíthatja.

i) Nézd meg, a tábláról milyen statisztikákat tárol a DBMS!
   - ALL_TABLES and DBA_OBJECT_TABLES
   - ALL _TAB_STATISTICS and ALL _TAB_COL_STATISTICS
   - ALL _TAB_HISTOGRAMS
   - ALL _TAB_COLS
   - ALL _COL_GROUP_COLUMNS
   - ALL _INDEXES and ALL _IND_STATISTICS

2. Feladatsor (screenshot és explain plain minden esetben!)

a) Milyen attribútumok találhatóak a LUKACS.HISTORYITEMS_LARGE táblában, valamint hány recordot tartalmaz a tábla? Milyen indexek vannak a táblán? Hány blokkból állnak és mennyi helyet foglalnak? (1block - 8192byte)

b) Kérdezd le index használata nélkül az első 100 User-hez tartozó adatokat. SELECT /*+ NO_INDEX(HISTORYITEMS_LARGE) */ * FROM …

c) Végezd el ugyanezt a lekérdezést index használattal. A lekérdezés költsége hogyan változott?

d) Kérdezd le az elmúlt 13 hónapban indított számok adatait.

e) Kérdezd le az első 100 user azon adatait, amelyek az elmúlt 15 hónapban indított számokhoz tartoznak.

f) Melyik esetben nem használ indexet az adatbázis és miért?

g) Próbáld az alábbi lekérdezést különböző konstans értékekkel, így különböző lekérdezési szelektivitással!

```
SELECT COUNT(updated_at)
FROM lukacs.historyitems_large
WHERE user_id < 50
```

h) Milyen szelektivitásnál használja még az indexet a DBMS, milyennél már nem? A lekérdezések költsége (COST) hogyan változik? Készítsen táblázatot (legalább 10 sorral), melyben a lekérdezés szelektivitása, a költség és az szerepel, hogy a DBMS használja-e az indexet.