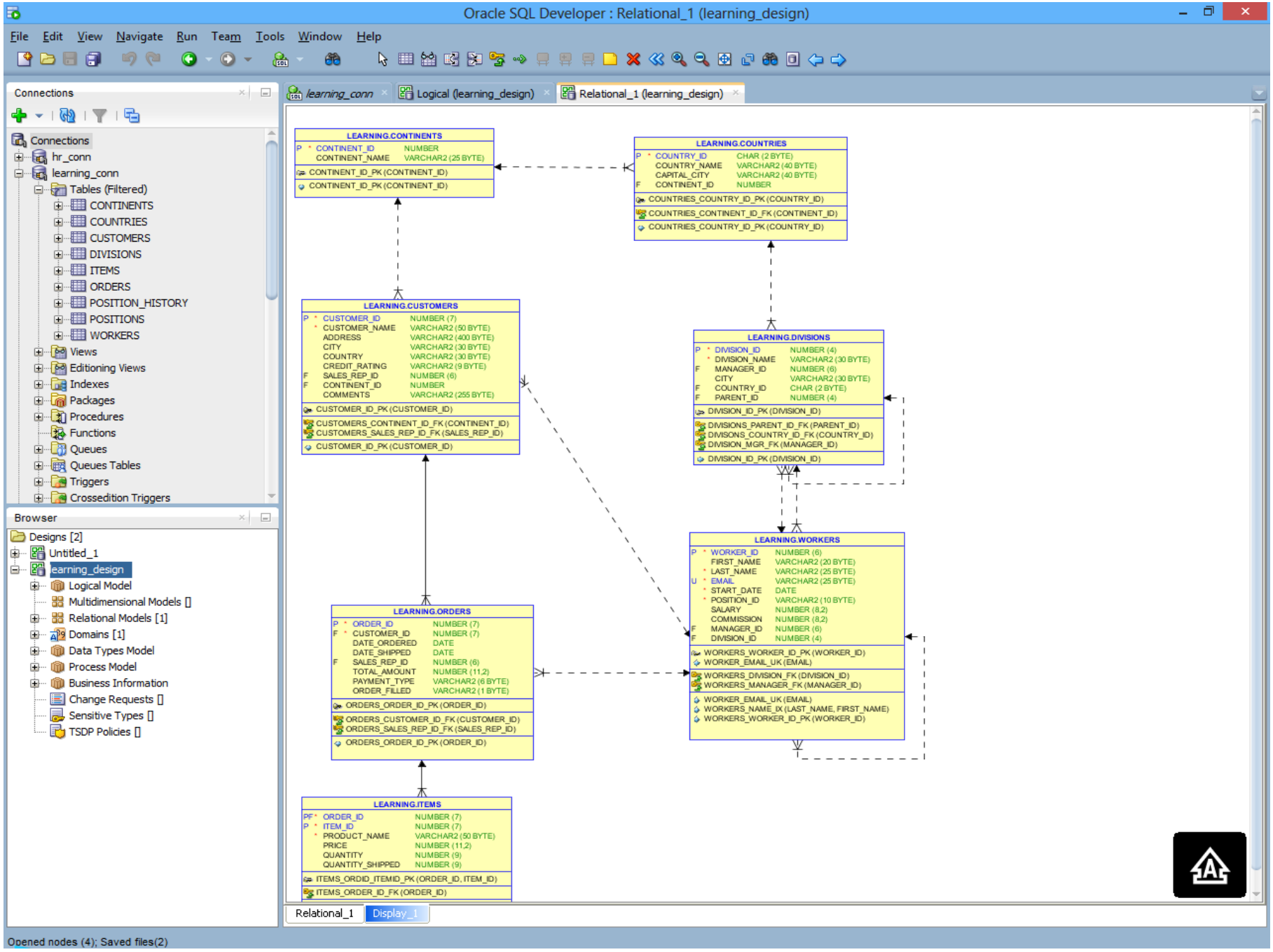
Kerepes Tamás – Czinkóczki László

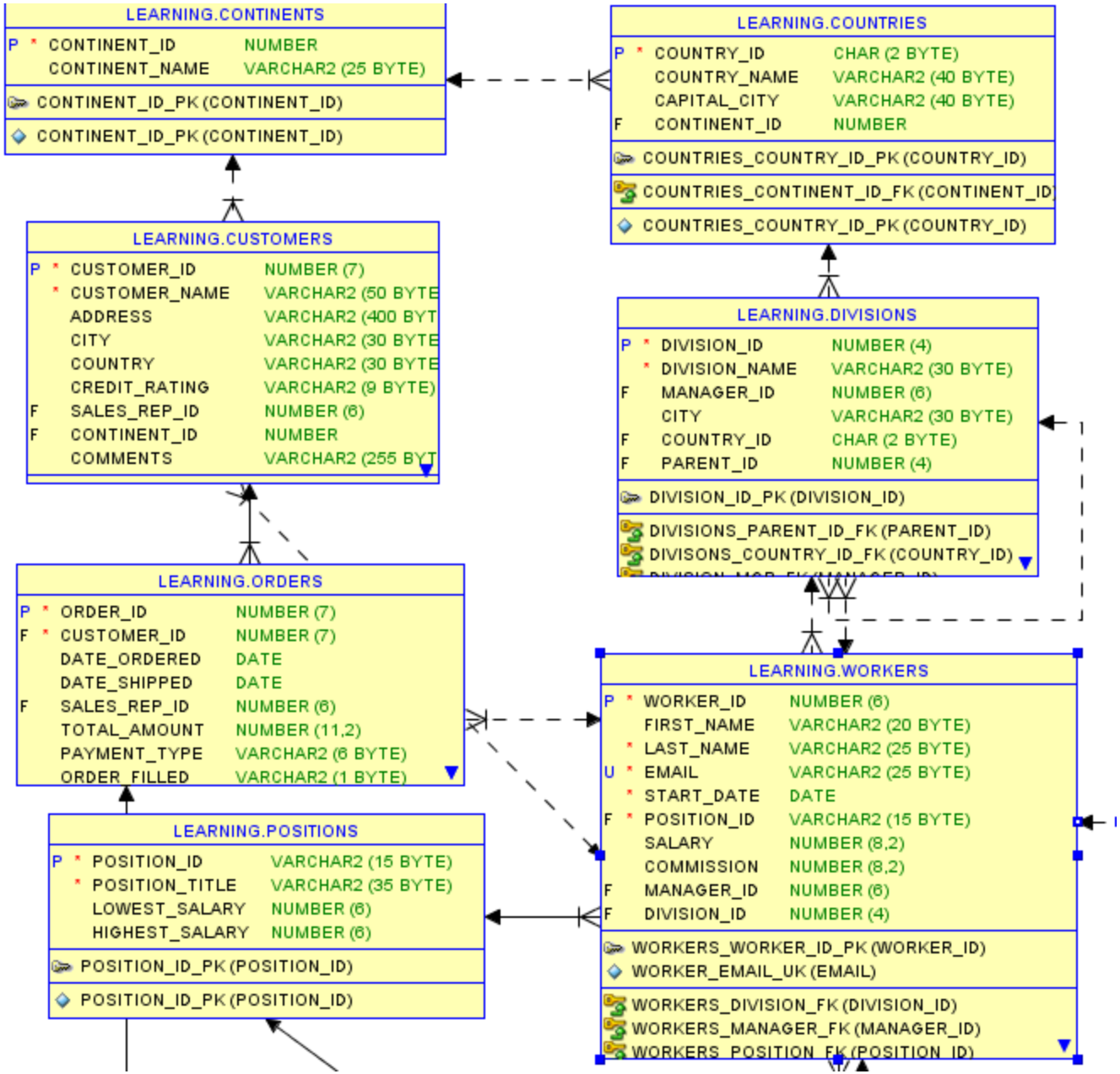
Adatbázisok

DML utasítások és tranzakciókezelés

A LEARNING séma



A LEARNING séma egy részlete



DML utasítások (Data Manipulation Language)

- A DML utasítások a meglévő táblák (általánosabban „egyes séma-objektumok”) adatait módosítják.
- DML utasítás az, amellyel:
  - Új sorokat „szúrunk be” a táblába – sorok beírása
  - A táblák meglévő sorait módosítjuk
  - A táblák meglévő sorait kitöröljük
  - Egyes adatbáziskezelőknél létezik még MERGE is…
- Van aki a SELECT-et is a DML utasítások közé sorolja
- A *tranzakció* DML utasítások sorozata, amely az adatmódosításaink egy logikai egysége. Mindezeket a változtatásokat (módosításokat) egységként vagy érvényesítjük a végén, vagy mindenestől visszavonjuk

Az INSERT utasítás és a szintaxisa

- Sorok beszúrása a táblába. INSERT utasítással sorokat adunk hozzá a táblához, a nézet alaptáblájához, esetleg egyéb, „hasonló” nem szabványos elemhez (pl. egy tábla-partícióhoz)

```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

- Ezzel a szintaxissal egy utasítással csupán egy sort szúrhatunk be.
- Megadjuk az oszlopokat, amelyek értéket kapnak:

```
INSERT INTO divisions (division_id,division_name,
manager_id,city,country_id,parent_id)
VALUES (2,'Headquarters',
100,'San Francisco','US',NULL);

1 rows inserted.
```

Null értékek beszúrása

- Implicit módszer: azokat az oszlopokat kihagyjuk az oszloplistából, amelyekbe NULL kell, hogy kerüljön

```
INSERT INTO divisions
(division_id,division_name,manager_id,city)
VALUES (2,'Headquarters',100,'San Francisco');
1 row created.
```

- Explicit módszer: `NULL` kulcsszót használunk a `VALUES` klauzulában:

```
INSERT INTO   divisions
VALUES   (100, 'Finance', NULL, NULL, NULL, NULL);
1 row created.
```

Az INSERT SELECT utasítás

- Írhatunk `INSERT` utasítást egy allekérdezéssel:

```
INSERT INTO preferred_customers
  SELECT *
  FROM   customers
  WHERE  credit_rating in ('GOOD','EXCELLENT');
105 row created.
```

- Ilyenkor nem szerepel a `VALUES` klauzula
- Az `INSERT` klauzula oszlopainak a száma megegyezik az allekérdezés oszlopainak a számával:

```
INSERT INTO preferred_customers
        (customer_id ,customer_name,city)
  SELECT customer_id ,customer_name,city
  FROM   customers
  WHERE  credit_rating in ('GOOD','EXCELLENT');
105 row created.
```

Az UPDATE utasítás és a szintaxisa

- A meglévő sorok értékeinek a módosítása
- Az UPDATE utasítás a tábla soraiban, vagy a nézet alaptáblája, esetleg egy materializált nézet soraiban lévő attribútumok értékeit módosítja:

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

```
UPDATE workers
SET    division_id = 70
WHERE  worker_id = 100;
1 row updated.
```

```
UPDATE workers
SET   last_name = UPPER(last_name);
53 rows updated.
```

Egyszerre tetszőleges számú sor is módosítható allekérdezéssel

Módosítsuk a 206-os dolgozó munkakörét és bérét úgy, hogy az megegyezzen a 205-ös dolgozó munkakörével és bérével

```
UPDATE   workers
SET (position_id, salary) = (SELECT  position_id, salary
FROM    workers   WHERE   worker_id = 205)
WHERE    worker_id    =  206;

1 rows updated.
```

Korrelált UPDATE utasítás

- A korrelált allekérdezés egy olyan SELECT utasítás, amely be van ágyazva a másik utasításba (itt pl. az UPDATE-be), és amely hivatkozik az őt körülvevő SQL utasítás oszlopára vagy oszlopaira.
- Amikor elképzeljük a működését, akkor úgy képzeljük, hogy a külső UPDATE utasítás által érintett minden sorra külön lefut a belső SELECT

A CUSTOMERS tábla COMMENTS mezőjébe szövegesen írjuk be, hogy az illető ügyfél összesen mekkora értékben rendelt terméket:

```
UPDATE customers c
 SET   comments =
         (SELECT TO_CHAR(SUM(total_amount))
          FROM   orders o
          WHERE  o.customer_id = c.customer_id);
135 rows updated.
```

A DELETE utasítás

- A táblák meglévő sorainak a törlése

```
DELETE [FROM]   table
[WHERE          condition];
```

```
DELETE FROM divisions
 WHERE  division_name = 'Contracting';
1 row deleted.
```

```
DELETE FROM divisions
 WHERE  division_name = 'Recruiting';
```

```
Error report -
SQL Error: ORA-02292: integrity constraint (LEARNING.WORKERS_DIVISION_FK) violated - child record found
02292. 00000 - "integrity constraint (%s.%s) violated - child record found"
```

Allekérdezés a DELETE-ben

- Egy DELETE segítségével törölhetők ki olyan sorok, amelyek megtalálhatóak egy másik táblában, vagy akár fordítva, azok, amelyek nem találhatóak a másikban sem.

```
DELETE FROM newworker
WHERE worker_id NOT IN
         (SELECT NVL(manager_id,-1)
          FROM   workers);
39 row deleted.
```

A tranzakciók fogalma

- A tranzakció SQL műveletek (DML műveletek) egy olyan sorozata, amely vagy teljes egészében érvényre jut, vagy annak minden módosítása visszavonásra kerül.
- A tranzakciókkal logikai csoportokba rendezzük a módosításainkat
- A fejlesztő (programozó) felelőssége, hogy pontosan a megfelelő egységekbe (tranzakciókba) csoportosítsa a DML műveleteket.

Az adatbáziskezelő ACID tulajdonsága

- ACID („savasság”): egy adatbázistól elvárt képességek halmaza
  - Atomicity (atomicitás): A fenti tulajdonság. Nem lehet fele véglegesített, a másik fele meg visszavont
  - Consistency (konzisztencia): a tranzakciók az adatbázis tartalmát egyik érvényes (konzisztens állapotából vezetik át egy másik érvényes állapotba
  - Isolation (izoláció): egyidőben több tranzakció hajtódhat végre, de az eredmény azonos, mint ha szekvenciálisan történtek volna a tranzakciól
  - Durability (tartósság): ha egy tranzakciót már jóváhagytak, akkor az „végleges”, tehát pl. egy áramszünet nem okozhatja, hogy elveszik a hatása

A különböző izolációs szintek

- Az ISO szabvány a következő lehetséges izolációs szinteket definiálja:
  - Read Uncomitted
  - Read Comitted
  - Serializable
  - Repeatable Read

A COMMIT és a ROLLBACK utasítás

- Ha egy tranzakciót véglegesíteni akarunk, azt a **COMMIT** utasítással tesszük
- Ha viszont menet közben meggondoltuk magunkat, és vissza akarjuk vonni az éppen folyamatban lévő tranzakciónkat, azt a **ROLLBACK** utasítással tesszük.

Egy egyszerű példa

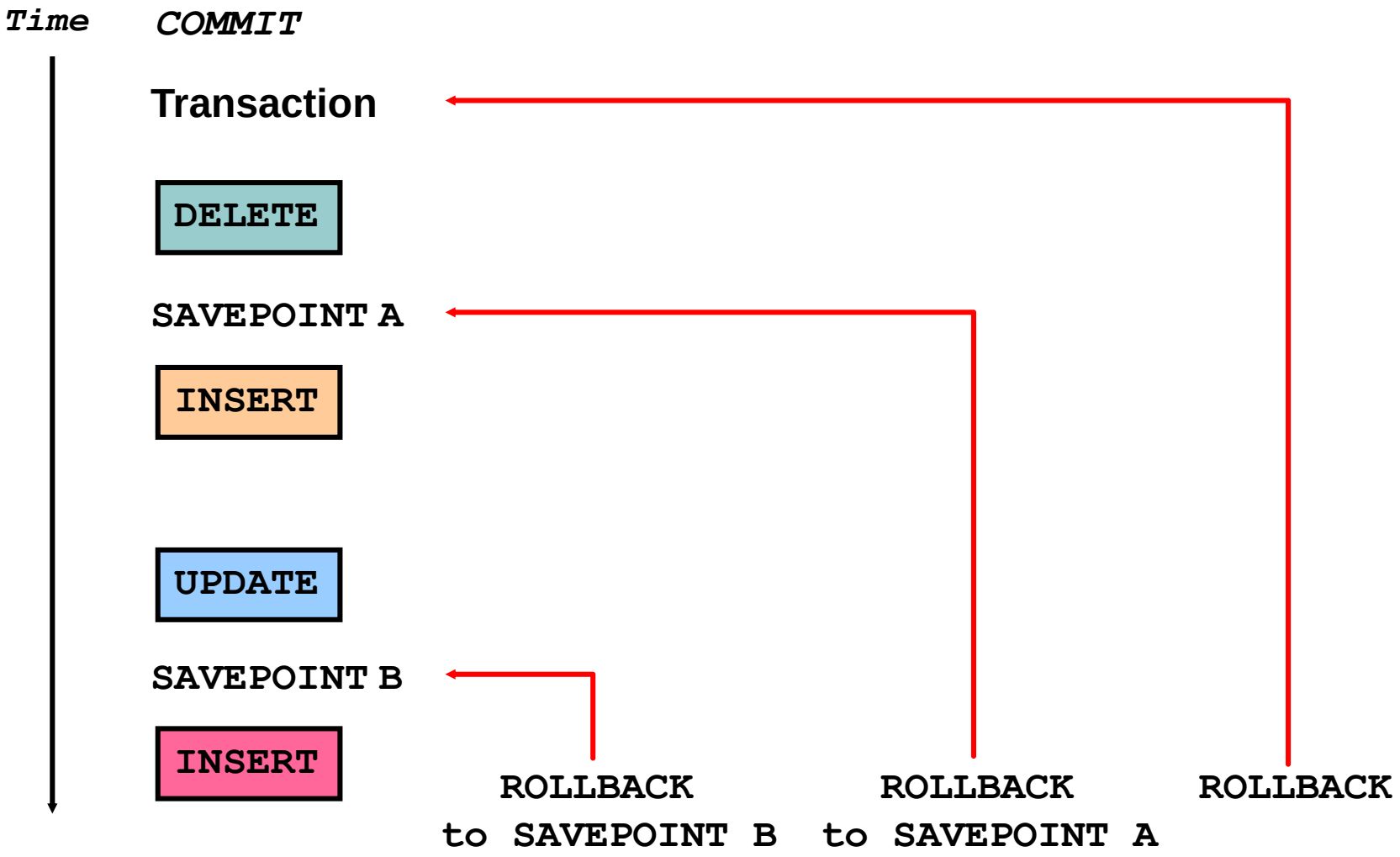
```
SELECT COUNT(*) FROM newworker;
COUNT(*)
----------
        53
INSERT INTO newworker SELECT * FROM newworker;
53 rows inserted.
SELECT COUNT(*) FROM newworker;
COUNT(*)
----------
       106
DELETE FROM newworker WHERE position_id LIKE 'SALES%';
28 rows deleted.
SELECT COUNT(*) FROM newworker;
COUNT(*)
----------
        78
ROLLBACK;
rollback complete.
SELECT COUNT(*) FROM newworker;
COUNT(*)
----------
        53
```

A SAVEPOINT utasítás

- Egyes adatbáziskezelőknél a tranzakció során elhelyezhetőek olyan pontok, amelyekre opcionálisan vissza lehet térni
- Ez a potenciális visszatérési pont a `SAVEPOINT`
- Ezekre a visszatérési pontokra a `ROLLBACK TO SAVEPOINT` utasítással térhetünk vissza

```
UPDATE...
SAVEPOINT update_done;
Savepoint created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.
```

Tranzakciók vezérlése



Az adatok értéke a `COMMIT` vagy `ROLLBACK` előtt

- Az adatok tranzakció előtti állapotát még mindig vissza lehet állítani (ROLLBACK)
- Az a munkamenet, amelyben folyamatban van a tranzakció, az SELECT utasításaiban látja a tranzakció során addig végrehajtott összes változtatást (DML)
- A többi munkamenet ekkor még nem látja a folyamatban lévő tranzakció által végzett módosításokat
- A folyamatban lévő tranzakció által módosított sorok le vannak zárva (foglalva). Más tranzakciók nem módosíthatják ezeket a sorokat mindaddig, amíg a folyamatban lévő tranzakció be nem fejeződik.

Az adatok értéke a `COMMIT` után

- Az adatok módosítása véglegessé válik az adatbázisban.
- A tranzakciós módosítások előtti értékek már nem visszaállíthatóak.
- Minden munkamenet látja az új értékeket.
- A zárak (lefoglalások) felszabadulnak.
- A tranzakció során esetleg elhelyezett mentési pontok (SAVEPOINT) elvesznek.

Az adatok értéke a `ROLLBACK` után

- A tranzakció által elvégzett minden módosítát elveszik
- A régi értékek visszaállnak
- A zárak (lefoglalások) felszabadulnak

```
DELETE FROM copy_emp;
22 rows deleted.
ROLLBACK ;
Rollback complete.
```

Utasítás-szintű ROLLBACK

- Ha egyetlen DML művelet a végrehajtása során hibára fut, akkor annak az 1 utasításnak az összes változtatása mindenestől visszavonásra kerül
- A tranzakció során előzőleg végrehajtott módosjtások megmaradnak
- Ha mégis a hiba miatt be kéne fejezni a tejes tranzakciót, azt a felhasználónak magának kell kezdeményeznie (`COMMIT` vagy `ROLLBACK` utasítások)

Kétfázisú jóváhagyás (two phase commit)

- Elosztott adatbázisokon igen nehéz biztosítani ezt az atomicitást
- Egy igen bonyolult algoritmus, a „two phase commit” algoritmus valósítja ezt meg:
  - Prepare fázis
  - Commit fázis
  - Utána elfelejthetők a részletek (forget fázis)

Phase	Description
Prepare phase	The initiating node, called the **global coordinator**, asks participating nodes other than the commit point site to promise to commit or roll back the transaction, even if there is a failure. If any node cannot prepare, the transaction is rolled back.
Commit phase	If all participants respond to the coordinator that they are prepared, then the coordinator asks the commit point site to commit. After it commits, the coordinator asks all other nodes to commit the transaction.
Forget phase	The global coordinator forgets about the transaction.

Olvasási konzisztencia

- Az adatbáziskezelő működési módja ez, amely azt biztosítja, hogy aki olvassa az adatokat, ne láthassa mások befejezetlen tranzakcióit
- Különbözőképpen implementálható, de mindenképpen rendkívül nehezen megvalósítható
- Például az Oracle adatbázisok esetén:
  - Aki csupán olvas (SELECT), az nem vár mások folyamatban lévő tranzakcióira
  - Aki módosít, nem vár azokra, akik csak olvasnak
  - Aki módosít, vár azokra, akik szintén módosítják ugyanazokat az adatokat
- Pl. az Oracle a tranzakciókezelést és az olvasási konzisztenciát lock-okkal és úgynevezett UNDO adatok vezetésével valósítja meg, de a konkrét megvalósítás – lévén rendkívül bonyolult – túlmutat az előadás keretein. A későbbiekben még megemlítjük majd röviden.