Tranzakciókezelés PL/SQL-ben

Tranzakció

- ACID tulajdonságok:
  - Atomosság, Konzisztencia, Izoláció, Tartósság
- A tranzakció állhat:
  - Több DML utasításból
  - Egy DDL utasításból

A tranzakció kezdete az első futtatható SQL utasítás, ami lehet:

- Egy DML utasítás
- Egy DDL utasítás
- SET TRANSACTION utasítás

Tranzakció

- Tranzakció vége:
  - `COMMIT;` vagy `ROLLBACK;` utasítás
  - DDL utasítás (implicit `COMMIT`)
  - Ha a felhasználó kilép egy Oracle eszközből vagy segédprogramtól (a normál kilépés általában implicit `COMMIT`-tal jár, azonban ez alkalmazásfüggő és konfigurálható lehet.)

DCL utasítások

- COMMIT
- ROLLBACK
- SAVEPOINT

Implicit visszagörgetés

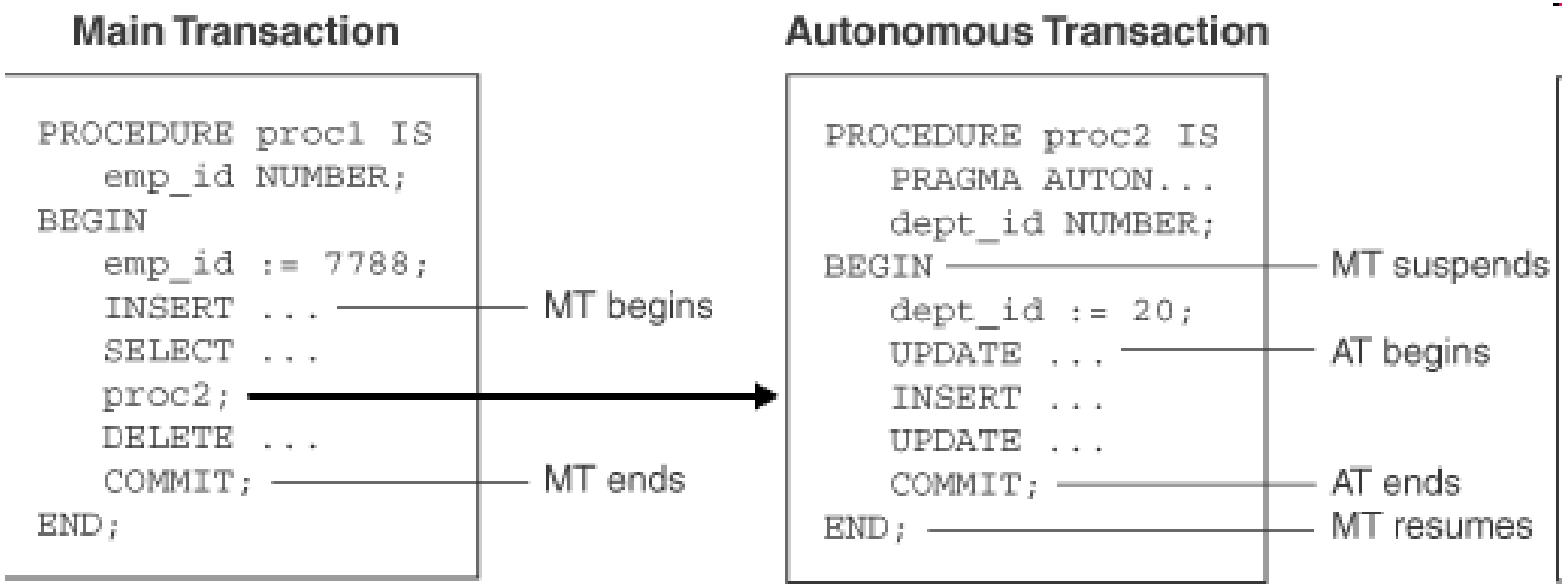
- SQL utasítás végrehajtása előtt az ABKR elhelyez egy (felhasználó számára hozzáférhetetlen) mentési pontot, ide kell visszagörgetni, ha az utasítás sikertelen (nem a teljes tranzakciót!)

SET TRANSACTION ... NAME név

- A tranzakciónak lehet nevet adni, így lehet követni később (monitorozás, auditálás)
- A SET TRANSACTION utasítás a tranzakcióban mindig az első utasításnak kell lennie

Autonóm tranzakciók

- Egy másik (a fő) tranzakció által elindított független tranzakció, amelyet a fő tranzakciótól függetlenül lehet véglegesíteni/visszagörgetni.



Autonóm tranzakciók

- Előnyök:
  - teljes függetlenség (nincsenek közös zárak, erőforrások vagy commit-függőségek a fő tranzakcióval)
  - moduláris, újrafelhasználható szoftverkomponens készíthető
- Megadása az AUTONOMOUS_TRANSACTION pragmával történik, az alábbiak deklarációs részében:
  - Legfelső szintű névtelen PL/SQL-blokk
  - Lokális, tárolt, vagy csomagban lévő alprogramok
  - SQL objektumtípus metódusai
  - Triggerek

```
CREATE OR REPLACE PACKAGE dolg_muveletek AS
  FUNCTION fizetesemeles (emp_id NUMBER, emeles NUMBER)
          RETURN NUMBER;
END dolg_muveletek ;
/
CREATE OR REPLACE PACKAGE BODY dolg_muveletek  AS
   FUNCTION fizetesemeles (emp_id NUMBER, emeles NUMBER)
     RETURN NUMBER IS
     PRAGMA AUTONOMOUS_TRANSACTION;
     uj_fizu NUMBER(8,2);
   BEGIN
     UPDATE employees SET salary =
       salary + emeles WHERE employee_id = emp_id;
     COMMIT;
     SELECT salary INTO uj_fizu FROM employees
       WHERE employee_id = emp_id;
     RETURN uj_fizu;
   END fizetesemeles;
END dolg_muveletek ;
/
```

Viszony a fő tranzakcióval

- Autonóm tranzakció ≠ beágyazott tranzakció!
  - nem függ a fő tranzakciótól, és nem osztozik vele az erőforrásokon
  - a véglegesített változások azonnal láthatók
  - az esetlegesen bekövetkező kivételek tranzakció szintű visszagörgetést vonnak maguk után (nem pedig utasítás szintűt)

```
CREATE TABLE a_tabla ( oszlop  NUMBER);

DECLARE

  PROCEDURE autonom(p NUMBER) IS
    PRAGMA AUTONOMOUS_TRANSACTION;
  BEGIN
    /* Első autonóm tranzakció kezdete - A1 */
    INSERT INTO a_tabla VALUES(p);
    COMMIT;

    /* Második autonóm tranzakció kezdete - A2 */
    INSERT INTO a_tabla VALUES(p+1);
    COMMIT;
  END;
…
```

```
…
 BEGIN
  /* Itt még a fő tranzakció fut - F */
  SAVEPOINT kezdet;

  /* Az eljárás autonóm tranzakciót indít */
  autonom(10);

  /* A fő tranzakció visszagörgetése */
  ROLLBACK TO kezdet;
END;
/

SELECT * FROM a_tabla;
```

- A főtranzakció a tulajdonságait megosztja a beágyazott rutinokkal, azonban az autonóm tranzakciókkal nem.

- Az autonóm tranzakció által végrehajtott változásokat a fő tranzakció alapértelmezés szerint látja. Ezen módosítani a következő utasítással lehet:
- `SET TRANSACTION ISOLATION LEVEL {READ COMMITTED | SERIALIZABLE};`
- READ COMMITTED (alapértelmezés): véglegesítés után a fő tranzakció látja a változtatásokat
- SERIALIZABLE: nem látja

Végrehajtás

- Egy autonóm rutinba belépéskor a fő tranzakció felfüggesztésre kerül, kilépéskor folytatódik
- Explicit módon kell véglegesíteni/visszagörgetni, ennek hiányában a végén kivétel váltódik ki, ami kezeletlenül ROLLBACK-et eredményez
- Tetszőleges számú COMMIT lehet
- A mentési pont hatásköre az őt definiáló tranzakció
- A SET TRANSACTION utasítás hatásköre az őt definiáló tranzakció

```
CREATE TABLE at_test
 (id NUMBER NOT NULL,
 description VARCHAR2(50) NOT NULL);

INSERT INTO at_test (id, description)
VALUES (1, 'Description for 1');
INSERT INTO at_test (id, description)
VALUES (2, 'Description for 2');

SELECT * FROM at_test;
```

```
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 FOR i IN 3 .. 10
   LOOP
    INSERT INTO at_test(id,description)
    VALUES (i, 'Description for `||i);
   END LOOP;
 COMMIT;
END;
/
SELECT * FROM at_test;
```

```
ROLLBACK;
SELECT * FROM at_test;

ID DESCRIPTION
---------- ------------------------------
3 Description for 3
4 Description for 4
5 Description for 5
6 Description for 6
7 Description for 7
8 Description for 8
9 Description for 9
10 Description for 10
```

```
CREATE TABLE error_logs
  (id NUMBER(10) NOT NULL,
   log_timestamp TIMESTAMP NOT NULL,
   error_message VARCHAR2(4000),
   CONSTRAINT error_logs_pk
       PRIMARY KEY (id) );

CREATE SEQUENCE error_logs_seq;
```

```
CREATE OR REPLACE PROCEDURE log_errors
  (p_error_message IN VARCHAR2) AS
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO error_logs
   (id,log_timestamp,error_message)
  VALUES (error_logs_seq.NEXTVAL,
  SYSTIMESTAMP, p_error_message);

COMMIT;
END;
/
```

```
BEGIN
   INSERT INTO at_test (id, description)
   VALUES (998, 'Description for 998');
-- Force invalid insert.
   INSERT INTO at_test (id, description)
   VALUES (999, NULL);


 EXCEPTION
    WHEN OTHERS
     THEN log_errors
       (p_error_message => SQLERRM);
          ROLLBACK;
END;
/
```

```
SELECT *
FROM at_test
WHERE id >= 998;
no rows selected

SELECT *
FROM error_logs;
ID LOG_TIMESTAMP ERROR_MESSAGE
------------------------------
1 28-FEB-2006 11:10:10.107625 ORA-01400:
  cannot insert NULL into
  ("TIM_HALL"."AT_TEST"."DESCRIPTION")

1 row selected.
```

Autonóm trigger

```
DROP TABLE emp;
CREATE TABLE emp AS SELECT * FROM employees;

DROP TABLE log;
CREATE TABLE log (
  log_id   NUMBER(6),
  up_date  DATE,
  new_sal  NUMBER(8,2),
  old_sal  NUMBER(8,2));
```

```
CREATE OR REPLACE TRIGGER log_sal
  BEFORE UPDATE OF salary ON emp FOR EACH ROW
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
INSERT INTO log (log_id,up_date,new_sal,old_sal)
VALUES (:old.employee_id,SYSDATE,:new.salary,:old.salary);
COMMIT;
END;
/
UPDATE emp SET salary = salary * 1.05
WHERE employee_id = 115;
COMMIT;

UPDATE emp SET salary = salary * 1.05
WHERE employee_id = 116;
ROLLBACK;

-- Show that both committed and rolled-back updates
```

```
-- Show that both committed and rolled-back updates
-- add rows to log table

SELECT * FROM log
WHERE log_id = 115 OR log_id = 116;
Result:
LOG_ID UP_DATE      NEW_SAL    OLD_SAL
---------- --------- ---------- ----------
       115 28-APR-10    3417.75       3255
       116 28-APR-10    3197.25       3045

2 rows selected.
```

```
DROP TABLE temp;
CREATE TABLE temp (temp_id NUMBER(6),up_date DATE);

CREATE OR REPLACE TRIGGER drop_temp_table
  AFTER INSERT ON log
DECLARE PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE temp';
  COMMIT;
END;
/
```

```
-- Show how trigger works
SELECT * FROM temp;

Result:
no rows selected

INSERT INTO log (log_id, up_date, new_sal, old_sal)
VALUES (999, SYSDATE, 5000, 4500);

1 row created.

SELECT * FROM temp;

Result:
SELECT * FROM temp
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```