Další vlastnosti PL/SQL

EXECUTE IMMEDIATE I.

- Slouží k dynamickému konstruování a následnému spuštění SQL příkazů, může také sestavit a spustit anonymní blok PL/SQL:

```
create or replace procedure aero.trunc_table
                        (p_table_name varchar2) is
  begin
    execute immediate 'truncate table ' || p_table_name;
  end;
```

EXECUTE IMMEDIATE II.

- Použití EXECUTE IMMEDIATE:
  - Uvnitř PL/SQL není možné volat DDL operace (CREATE, ALTER, DROP, TRUNCATE).
  - Pokud potřebujete dynamicky zkonstruovat SQL dotaz.

- Zase na druhou stranu pozor na:
  - Pokud by se takový příkaz volal často, pak bude pomalejší.
  - Možnost zavedení SQL injection.
  - Daleko větší šance chyb, protože při kompilaci se neprovádí validace.
  - Je méně čitelný.

EXECUTE IMMEDIATE III. - NEE!!!

- Tohle bude fungovat, ale je blbost to tak dělat (vytvoří to bordel v kurzorech a navíc je to náchylné na SQL injection):

```
declare
  nazev varchar(255);
begin
  nazev := 'test destinace ' || dbms_random.random;
  execute immediate 'insert into destinace (nazev) values ('''
                    || nazev || ''')';
end;
```

EXECUTE IMMEDIATE IV. - INPUT

- Jak správně zavolat EXECUTE IMMEDIATE se vstupem (imunní vůči SQL injection):

```
declare
  nazev varchar(255);
begin
  nazev := 'test destinace ' || dbms_random.random;
  execute immediate 'insert into destinace (nazev) values (:nazev)'
     using nazev;
end;
```

EXECUTE IMMEDIATE V.

- Ještě hezčí je toto:

```
declare
  nazev varchar(255);
  sql_command varchar(2000);
begin
  sql_command := 'insert into destinace (nazev) values (:nazev)';
  nazev := 'test destinace ' || dbms_random.random;
  execute immediate sql_command using nazev;
end;
```

EXECUTE IMMEDIATE VI.

- EXECUTE IMMEDIATE s výstupem (jeden řádek):

```
declare
  zamestnanec_row zamestnanec%rowtype;
  sql_command varchar(2000);
  zam_id number;
begin
  zam_id := 1;
  sql_command := 'select * from zamestnanec where zamestnanec_id = :zam_id';
  execute immediate sql_command into zamestnanec_row using zam_id;
  dbms_output.put_line('zamestnanec s ID ' || zam_id || ': '
    || zamestnanec_row.jmeno || ' ' || zamestnanec_row.prijmeni);
end;
```

EXECUTE IMMEDIATE VII.

- EXECUTE IMMEDIATE s výstupem (více řádků):

```
declare
  type zamestnanec_type is table of zamestnanec%rowtype;
  zamestnanci zamestnanec_type;
begin
  execute immediate 'select * from zamestnanec'
    bulk collect into zamestnanci;
  for i in zamestnanci.first .. zamestnanci.last loop
    dbms_output.put_line(zamestnanci(i).jmeno || ' '
        || zamestnanci(i).prijmeni);
  end loop;
end;
```

EXECUTE IMMEDIATE VIII.

- Předchozí SELECT bez hvězdičky:

```
declare
  type zamestnanec_rec is record (jmeno zamestnanec.jmeno%type,
                                  prijmeni zamestnanec.prijmeni%type);
  type zamestnanec_type is table of zamestnanec_rec;
  zamestnanci zamestnanec_type;
begin
  execute immediate 'select jmeno, prijmeni from zamestnanec'
    bulk collect into zamestnanci;
  for i in zamestnanci.first .. zamestnanci.last loop
    dbms_output.put_line(zamestnanci(i).jmeno ||
                         ' ' || zamestnanci(i).prijmeni);
  end loop;
end;
```

DBMS_SQL

- EXECUTE IMMEDIATE neumožňuje volání s variabilním počtem vstupů. K tomu musíme použít procedury z DBMS_SQL balíčku:

```
cur integer := dbms_sql.open_cursor;
stmt varchar(2000) := 'SQL PŘÍKAZ / DOTAZ';
dbms_sql.parse(cur, stmt, dbms_sql.native);
dbms_sql.bind_variable(cur, NAME, VALUE);
rows_processed := dbms_sql.execute(cur);
dbms_sql.close_cursor(cur);
```

  - Pro následující příklad je nutné zavolat:

```
alter table zamestnanec add last_changed date;
```

  - Příklad je v souboru: `test_dbms_sql_example_1.sql`

AUTHID DEFINER vs. CURRENT_USER

- Co když bude proceduru `trunc_table()` chtít volat uživatelé z jiného schématu? Například uživatel SYSTEM bude chtít zavolat:

```
aero.trunc_table('nazev_system_tabulky');
```

- Jenže toto mu vyhodí chybu, že taková sekvence neexistuje. Proč? Protože ve výchozím nastavení je procedura vykonána s oprávněními schématu, ve kterém je **definována**.
- Pro změnu je nutné mít následující hlavičku procedury:

```
CREATE OR REPLACE PROCEDURE AERO.trunc_table
    (p_seq_name IN VARCHAR2) authid current_user IS
```

Informace o uživateli

- Jméno přihlášeného uživatele: **SELECT USER FROM** dual;
- Informace o tabulkách vlastněných přihlášeným uživatelem (ve schématu uživatele): **SELECT** * **FROM** SYS.user_tables;
- Informace o všech tabulkách, ke kterým má přístup přihlášený uživatel: **SELECT** * **FROM** SYS.all_tables;
- Informace o názvech tabulek, pohledů, sekvencí, synonym vlastněných přihlášeným uživatelem (ve schématu uživatele): **SELECT** * **FROM** SYS.user_catalog;
- Role přiřazené přihlášenému uživateli: **SELECT** * **FROM** SYS.user_role_privs;
- Popis dalších užitečných tabulek z datového slovníku Oracle poskytne dotaz: **SELECT** * **FROM** dict;

Transakce v Oracle a PL/SQL

- Začátek transakce:
  - Automaticky při vykonání prvního příkazu (po připojení k databázi, po ukončení předchozí transakce).
- Konec transakce:
  - Při provedení příkazu COMMIT nebo ROLLBACK.
  - Před příkazem DDL (CREATE, ALTER, ...) nebo DCL (GRANT, REVOKE, ...) se provede automaticky COMMIT. Rovněž po úspěšném provedení příkazu z těchto skupin se provede automaticky COMMIT.
  - Při odpojení od databáze je aktuální transakce automaticky potvrzena (COMMIT).
  - Při korektním ukončení konzole SQL*Plus (příkazem exit, quit) se provede automaticky COMMIT, při nekorektním ukončení se provede automaticky ROLLBACK.
  - Při provedení příkazu DML, který selže, se provede automaticky ROLLBACK tohoto konkrétního dotazu.

Transakce v PL/SQL

- **Blok PL/SQL kódu za normálních okolností pokračuje v započaté transakci na úrovni volajícího kódu.**
- Ve funkcích není běžné používat příkaz COMMIT, protože volání funkce se obvykle provádí v rámci nějakého jiného SQL příkazu, který se provádí v rámci určité transakce – transakce se řídí na úrovni volajícího kódu. Pokud se jedná o funkci ukládající data/nějaký výsledek, může být uložení, pokud je to vhodné, hned potvrzeno příkazem COMMIT (před příkazem RETURN).
- V proceduře může/nemusí být použit příkaz COMMIT, v závislosti na tom, zda chceme určitou skupinu vykonaných příkazů hned potvrdit.

ROLLBACK při vyvolání výjimky

- Při vyvolání výjimky během provádění DML příkazů můžeme chtít provést ROLLBACK, a odvolat tak jen částečně provedené DML příkazy...

```
DECLARE
    emp_id INTEGER;
    ...
BEGIN
   SELECT empno, ... INTO emp_id, ... FROM new_emp WHERE ...
   ...
   INSERT INTO emp VALUES (emp_id, ...);
   INSERT INTO tax VALUES (emp_id, ...);
   INSERT INTO pay VALUES (emp_id, ...);
   ...
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
     ROLLBACK;
    ...
END;
```

ROLLBACK při vyvolání výjimky

- Nebo odvolat jen určitou skupinu DML příkazů použitím SAVEPOINTu a ROLLBACK TO SAVEPOINT...

```
DECLARE
  emp_id emp.empno%TYPE;
BEGIN
  UPDATE emp SET ... WHERE empno = emp_id;
  DELETE FROM emp WHERE ...
  ...
  SAVEPOINT do_insert;
  INSERT INTO emp VALUES (emp_id, ...);
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ROLLBACK TO do_insert;
END;
```

Autonomní transakce

- **Autonomní transakce** (autonomous transaction) je nezávislá transakce, která může být vyvolána z jiné (hlavní) transakce.
- Autonomní transakce je na hlavní transakci zcela nezávislá (nevidí dosud nepotvrzená data z hlavní transakce, nesdílí s ní žádné zámky na datech).
- COMMIT autonomní transakce provede potvrzení změn v rámci autonomní transakce, nikoliv změn v hlavní transakci. Pokud je v hlavní transakci někdy po vyvolání autonomní transakce vyvolán ROLLBACK, ROLLBACK se nebude týkat kódu běžícího v rámci autonomní transakce (ten může být potvrzen vlastním COMMITem, případně odvolán vlastním ROLLBACKem).

Autonomní transakce a PL/SQL

- V autonomní transakci mohou běžet následující bloky:
  - Uložená procedura nebo funkce (v balíčku, mimo balíček),
  - trigger,
  - balíček,
  - metoda objektového typu,
  - anonymní blok na top-level úrovni (nikoliv vnořený blok).
- Stačí hned za DECLARE v anonymním bloku/za hlavičkou funkce/procedury uvést direktivu překladače PL/SQL (direktivy se píší pomocí klíčového slova pragma):
  pragma AUTONOMOUS_TRANSACTION;
- Např.:

```
CREATE PROCEDURE close_account (acct_id INTEGER, OUT balance) AS
  PRAGMA AUTONOMOUS_TRANSACTION;
  my_bal REAL;
BEGIN ... END;
```

Autonomní transakce a PL/SQL

- Příklad anonymního bloku v autonomní transakci:

```
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
  my_empno NUMBER(4);
BEGIN ... END;
```

- Příklad triggeru v autonomní transakci:

```
CREATE TRIGGER TR_logging
BEFORE INSERT ON parts FOR EACH ROW
DECLARE
   PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
   INSERT INTO parts_log VALUES(:new.pnum, :new.pname);
   COMMIT;
END;
```

  - Výhoda tohoto triggeru spočívá v tom, že může zalogovat do tabulky parts_log data i tehdy, kdy provádění příkazu INSERT v hlavní transakci selže (a je proveden ROLLBACK).