Databázové systémy 2

Michal Krátký, Radim Bača

Katedra informatiky
Fakulta elektrotechniky a informatiky
VŠB – Technická univerzita Ostrava

2024/2025

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

Obsah přednášky

Procedurální rozšíření SQL – PL/SQL

1. Jazyk PL/SQL[1] představuje rozšíření jazyka SQL o procedurální rysy.
2. Syntaxe PL/SQL je založena na jazyku ADA.
3. Podobná procedurální rozšíření pak byla vyvinuta i pro další relační DBS: Transact-SQL pro Sybase a Microsoft SQL Server, PL/pgSQL pro PostgreSQL a SQL PL pro IBM DB2.

[1] `https://docs.oracle.com/en/database/oracle/oracle-database/21/lnpls/`

Vlastnosti PL/SQL

Výhody:

- Kombinace procedurální logiky a SQL.
- Možné nižší množství přenášených dat (přenáší se jen konečné výsledky) – PL/SQL kód je uložen v DBS.
- Sdílení kódu mezi aplikacemi.
- Nezávislost na platformě (nikoli na DBS).

Nevýhody:

- Nepřenositelnost aplikace mezi DBS různých výrobců (to je ovšem problém i na úrovni SQL).
  - **Potřebujeme přenositelnost mezi DBS?**
- Umožňuje vyhnout se složitějším dotazům dekompozicí na více dotazů, což má ale dopad na efektivitu.

Základní struktura PL/SQL bloku[2]

1. `DECLARE` – nepovinná deklarace lokálních proměnných,
2. `BEGIN` – povinné otevření bloku příkazů,
3. `EXCEPTION` – nepovinné zachytávání výjimek,
4. `END` – povinné ukončení bloku.

[2] Někdy také mluvíme o anonymní proceduře.

Proměnné

V sekci `DECLARE` můžeme deklarovat proměnné, jenž mohou být použity v bloku, deklarace má následující tvar:

```
jmeno_promenne typ_promenne [NOT NULL := hodnota];
```

Kde:

- `jmeno_promenne` je název proměnné, který má obvykle prefix `v_`,
- `typ_promenne` je datový typ proměnné,
- `hodnota` je nepovinná část, která definuje výchozí hodnotu proměnné.

Proměnné 1/2

- Proměnné můžeme využít pro uložení dočasných hodnot a pro manipulaci s nimi.
- Hodnota muže být do proměnné zapsána dvěma způsoby:

Syntaxe	**Příklad**
`jmeno_promenne := hodnota`	`v_vek := 20`
`SELECT sloupec INTO jmeno_promenne FROM jmeno_tabulky`	`SELECT vek INTO v_vek FROM student WHERE login LIKE 'bon007'`

- SELECT musí vracet právě jeden záznam, jinak je vygenerována výjimka `NO_DATA_FOUND` resp. `TOO_MANY_ROWS`.

Proměnné 2/2

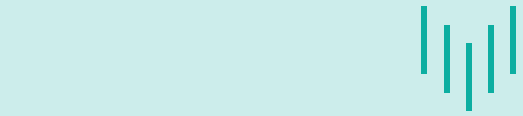
- Při práci s proměnnou můžeme využít standardní aritmetické operátory v případě čísel.
- V případě řetězců můžeme použít operátor || pro konkatenaci řetězců a SQL funkce (`TO_CHAR`, `TO_DATE`, `SUBSTR`, `LENGTH`, atd.).

Proměnné, příklad 1/4

```
CREATE TABLE Student
(
   login CHAR(6) CONSTRAINT Student_PK PRIMARY KEY,
   fname VARCHAR(30) NOT NULL,
   lname VARCHAR(50) NOT NULL,
   email VARCHAR(50) NOT NULL,
   grade INTEGER  NOT NULL,
   date_of_birth DATE NOT NULL
);

INSERT INTO Student VALUES('kra28', 'Stanislav', 'Krátoška',
   'kra28@vsb.cz', 1, to_date('2004-01-01', 'YYYY-MM-DD'));
INSERT INTO Student VALUES('bon007', 'James', 'Bond',
   'bon007@vsb.cz', 1, to_date('1940-01-01', 'YYYY-MM-DD'));
```

Proměnné, příklad 2/4



```
DECLARE
   v_fname VARCHAR2(20);
   v_lname VARCHAR2(20);
   v_email VARCHAR2(60);
BEGIN
   SELECT fname, lname INTO v_fname, v_lname
          FROM student WHERE login = 'bon007';

   v_email := v_fname || '.' || v_lname || '@vsb.cz';

   UPDATE student set email = v_email
          WHERE login = 'bon007';
END;
```

Proměnné, příklad 3/4

Jedná se o ukázkový příklad, stejnou funkcionalitu napíšeme jedním příkazem update:

```
BEGIN
  UPDATE Student
    SET email = fname || '.' || lname || '@vsb.cz'
  WHERE login = 'bon007';
END;
```

Proměnné, příklad 4/4

- Dlouhé řádky ve výstupu select?

```
select * from Student;
```

```
LOGIN  FNAME                          LNAME                                          EMAIL
------ ------------------------------ ---------------------------------------------- --------------
kra28  Stanislav                      Krátoška                                       kra28@vsb.cz
bon007 James                          Bond                                           James.Bond@vsb
```

- Řešení bez úpravy dotazu:

```
column fname format a14;
col lname format a14;
col email format a18;
```

```
LOGIN  FNAME          LNAME          EMAIL                 GRADE DATE_OF_B
------ -------------- -------------- ------------------ ---------- ---------
kra28  Stanislav      Krátoška       kra28@vsb.cz               1 01-JAN-04
bon007 James          Bond           James.Bond@vsb.cz          1 01-JAN-40
```

Operátor %TYPE

- Zadávání datových typů proměnných muže být problematické: datové typy proměnných velmi často **kopírují datové typy atributů** tabulek.
- Při změně datového typu atributu v tabulce, pak **musíme měnit** kód.
- Proto namísto konkrétního datového typu používáme operátor `%TYPE`.

Příklad:

```
v_lg Student.login%TYPE; ...
```

Kde proměnná `v_lg` je stejného typu jako atribut `login` tabulky `Student`.

Operátor %TYPE, příklad

```
DECLARE
   v_email VARCHAR(30);                -- neni vhodne
   v_email_2 Student.email%TYPE;       -- je vhodne
BEGIN
   SELECT email INTO v_email from Student
      where login = 'kra228';

   SELECT email INTO v_email_2 from Student
      where login = 'kra228';
END;
```

Operátor %ROWTYPE

- Operátor `%ROWTYPE` deklaruje **strukturovaný datový typ**, který obsahuje proměnné stejných datových typů jako jsou datové typy atributů tabulek.
- Instance takového strukturovaného datového typu **reprezentuje záznam z tabulky**.

Příklad:

```
v_st Student%ROWTYPE; ...
```

Kde proměnná `v_st` reprezentuje záznam, obsahuje stejné proměnné a datové typy jako tabulka `Student`.

Operátor %ROWTYPE, příklad

```
DECLARE
   v_login CHAR(6);                          -- neni vhodne
   v_fname Student.fname%TYPE;               -- lepsi, ale musime deklarovat
   v_lname Student.lname%TYPE;               -- promennou pro kazdy stribut.
   v_email Student.email%TYPE;
   v_grade Student.grade%TYPE;
   v_date_of_birth Student.date_of_birth%TYPE;
   v_student Student%ROWTYPE;                -- je vhodne
BEGIN
   SELECT *
     INTO v_login, v_fname, v_lname, v_email, v_grade, v_date_of_birth
   FROM Student
   WHERE login = 'bon007';                   -- neni vhodne

   SELECT * INTO v_student from Student
   WHERE login = 'bon007';                   -- je vhodne

   dbms_output.put_line(v_student.login);
END;
```

Proměnné, deklarace, definice, příklad

```
DECLARE
   C_VCHAR_MAXLEN CONSTANT NUMBER := 32767;
   v_date DATE := SYSDATE;
   v_number NUMBER NOT NULL := 1;
   v_student Student%ROWTYPE;
   v_name Student.name%TYPE;
BEGIN
...
END;
```

Poznámky

```
BEGIN
  -- jednoradkova poznamka
  INSERT INTO Person VALUES('sob28', 'jan.sobota@vsb.cz',
    'heslo', 'Jan', NULL, 'Sobota', NULL, NULL);
  /*
   * viceradkova poznamka
   */
  INSERT INTO PersonRole VALUES('sob28', 1);
END;
```

Výjimky

- **Výjimka je chyba**, která se vyskytne během provádění PL/SQL kódu.
- Jazyk PL/SQL nabízí vlastní mechanismus pro zpracování výjimek.
- Výjimka může vzniknout jak v samotném Oracle serveru (chyba provádění nějakého SQL dotazu), tak může být vytvořena samotným PL/SQL kódem.

Část EXCEPTION

- Ke zpracování výjimek v PL/SQL bloku slouží část `EXCEPTION`.

```
...
BEGIN
   ...
EXCEPTION
  WHEN jmeno_vyjimky THEN
     zpracování vyjímky
END;
```

- V případě chyby, program automaticky skočí do sekce `EXCEPTION`, konkrétně do části, která zpracovává danou výjimku (pokud taková část existuje).

Část EXCEPTION

```
...
BEGIN
    ...
EXCEPTION
   WHEN jmeno_vyjimky THEN
      zpracování vyjímky
END;
```

- V případě úspěšného zpracování, se výjimka **již dále nepropaguje** do části aplikace, která PL/SQL blok volala.
- V případě, že chceme zpracovat jakoukoli výjimku (kromě těch co již jsou zpracovány jinými `WHEN` příkazy), pak namísto `jmeno_vyjimky` dáme klíčové slovo `OTHERS`.

Definition

Transakce je databázová operace, obsahující více elementárních databázových operací (INSERT, UPDATE, SELECT, DELETE), která proběhne celá nebo není proveden žádný příkaz transakce.

- Ukončení transakce:
  - **COMMIT** - úspěšné ukončení transakce, aktualizace jsou potvrzeny v databázi.
  - **ROLLBACK** - neúspěšné ukončení transakce, všechny aktualizace transakce jsou zrušeny z databáze.

Příklad, Transakce v PL/SQL 1/2

```
CREATE TABLE Person(
   login char(5) PRIMARY KEY,
   email VARCHAR(20) NOT NULL,
   password VARCHAR(15) NOT NULL,
   fname VARCHAR(15) NOT NULL,
   mname VARCHAR(15),
   lname VARCHAR(15) NOT NULL,
   street VARCHAR(30),
   city VARCHAR(30));

CREATE TABLE Role (
   idRole INT PRIMARY KEY,
   role VARCHAR(30) NOT NULL);

CREATE TABLE PersonRole (
   login CHAR(5) REFERENCES Person,
   idRole INT REFERENCES Role,
   PRIMARY KEY(login, idRole));

INSERT INTO Role VALUES(1, 'Author');
```

Příklad, Transakce v PL/SQL 2/2

Při vložení nové osoby chceme zároveň přidat této osobě roli 'Autor' (která má `id=1`). Transakce tedy bude vypadat takto:

```
BEGIN
   INSERT INTO Person VALUES('sob28', 'jan.sobota@vsb.cz',
      'heslo', 'Jan', NULL, 'Sobota', NULL, NULL);
   INSERT INTO PersonRole VALUES('sob28', 1);
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      ROLLBACK;
END;
```

Pokud **selže** vložení osoby (např. osoba je už v systému evidována), pak vložení role osoby nedává smysl a celá transakce bude zrušena.

SET AUTOCOMMIT ON/OFF

- Po spuštění `SET AUTOCOMMIT ON`, bude každý SQL příkaz proveden v jedné transakci. **Tzn. operace COMMIT a ROLLBACK pozbývají významu.**
- Po spuštění `SET AUTOCOMMIT OFF`, začíná transakce po ukončení transakce předchozí a končí operacemi **COMMIT** nebo **ROLLBACK**.

Výjimky balíku STANDARD

Některé výjimky balíku STANDARD:

Jméno výjimky	**Číslo chyby**	**Popis**
ACCESS_INTO_NULL	ORA-06530	Pokus o přiřazení hodnoty do neinicializovaného objektu
DUP_VAL_ON_INDEX	ORA-00001	Pokus vložit duplicitní hodnotu
INVALID_CURSOR	ORA-01001	Neplatná operace s kurzorem
INVALID_NUMBER	ORA-01722	Selhala konverze čísla na řetězec
NO_DATA_FOUND	ORA-01403	Příkaz SELECT nevrátil data
TOO_MANY_ROWS	ORA-01422	Příkaz SELECT INTO vrátil více než jeden řádek
VALUE_ERROR	ORA-06502	Chybná manipulace s hodnotou

Zachycení výjimky

Následující procedura vypíše zprávu 'Hodnota atributu login musí být unikátní!' v případě výjimky `DUP_VAL_ON_INDEX`. V případě jiné výjimky vypíše chybovou hlášku dané chyby.

```
BEGIN
  INSERT INTO Student (login, fname, lname)
    VALUES ('bon007', 'James', 'Bond');
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    DBMS_OUTPUT.put_line ('Hodnota atributu login musí
      být unikátní!');
  WHEN OTHERS THEN
    DBMS_OUTPUT.put_line(DBMS_UTILITY.FORMAT_ERROR_STACK);
END;
```

Vyvolání výjimky

- PL/SQL umožňuje **vyvolat** výjimku v případě chyby.
- K tomuto účelu se používá klíčové slovo `RAISE`.
- Je možné tak vyvolat standardní nebo uživatelem definovanou výjimku.

Výjimka definovaná uživatelem

- Stejně jako proměnné nebo kurzory je možné v PL/SQL bloku **deklarovat také výjimku**:

  ```
  jmeno_vyjimky EXCEPTION;
  ```

- **Rozsah platnosti výjimky** je jen pro danou proceduru.
- Pokud budeme chtít výjimku deklarovanou v proceduře odchytit mimo tuto proceduru, pak je to možné jen s použitím `OTHERS`.
- Vlastní výjimku můžeme zveřejnit pomocí **balíčku** (package).

Výjimka definovaná uživatelem, příklad

V následujícím výpise dojde k vyvolání výjimky `too_many_records`, která není v proceduře ošetřena. Výjimka tedy bude propagována do nadřazeného kódu.

```
DECLARE
   too_many_records EXCEPTION;
   v_records INT;
BEGIN
   SELECT count(*) INTO v_records FROM student;
   IF v_records > 20 THEN
      RAISE too_many_records
   ELSE
      INSERT INTO student (login, fname, lname)
             VALUES ('bon007', 'James', 'Bond');
   END IF;
END;
```

Příklad, kill_locked_session 1/7

- Pokud uživatel zapomene aktualizace tabulky potvrdit příkazem **commit** nebo zrušit příkazem **rollback**, tabulka je stále uzamčena.
- Při další práci s tabulkou, může Oracle vyhazovat výjimku: **ORA-04021: timeout occurred while waiting to lock object**
- V některých případech nepomůže commit, např. když se odhlásíme a znovu přihlásíme, potřebujeme ukončit proces držící objekt:
  `ALTER SYSTEM KILL SESSION '<sid>, <serial#>'`
  kde `sid` a `serial#` jsou identifikátory procesu, např.:

  ```
  ALTER SYSTEM KILL SESSION '<1699, 20315>';
  ```

- Úkolem je napsat uloženou proceduru `kill_locked_sessions(p_user_name varchar)`, která ukončí všechny procesy blokující objekty uživatele `p_user_name`, např. takto: `exec sys.kill_locked_sessions('KRA28');`

Příklad, kill_locked_session 2/7

- Začneme systémovým pohledem v$locked_object, který obsahuje všechny aktuální zámky transakcí.
- Nejprve zjistíme schéma pohledu v$locked_object:

```
describe v$locked_object;
desc v$locked_object;

Name                Null? Type
------------------- ----- -------------
XIDUSN                    NUMBER
XIDSLOT                   NUMBER
XIDSQN                    NUMBER
OBJECT_ID                 NUMBER
SESSION_ID                NUMBER
ORACLE_USERNAME           VARCHAR2(128)
OS_USER_NAME              VARCHAR2(128)
PROCESS                   VARCHAR2(24)
LOCKED_MODE               NUMBER
CON_ID                    NUMBER
```

Příklad, kill_locked_session 3/7

- Nyní můžeme například vypsat object_id uzamčených objektů pro uživatele KRA28:

```
-- nastavení počtu znaků sloupce
col oracle_username format a15;

select lo.oracle_username, lo.object_id
from v$locked_object lo
where lo.oracle_username='KRA28';

ORACLE_USERNAME  OBJECT_ID
--------------- ----------
KRA28                637025
```

- Stále ovšem nemáme sid a serial#

Příklad, kill_locked_session 4/7

- sid a serial# najdeme v pohledu v$session, který obsahuje informace o aktuálním přihlášení - session:

```
select sid, serial#
from v$session
where username = 'KRA28'
```

```
       SID    SERIAL#
---------- ----------
      2426       17516
```

- Mezi atributy ovšem nenajdeme object_id, nemáme tedy jistotu, že tento proces je blokující.

Příklad, kill_locked_session 5/7

- Musíme tedy dále využít pohled systémového katalogu dba_objects (user_objects) obsahující seznam objektů (uživatele):

```
col owner format a15;
col object_name format a15;

select ob.owner, ob.object_name, ob.object_id
from dba_objects ob
where ob.owner='KRA28';
```

```
OWNER           OBJECT_NAME      OBJECT_ID
--------------- --------------- ----------
KRA28           ABSENCE             629423
KRA28           BODY                629405
KRA28           BODY_PK             629406
KRA28           BODY_TYPE           629402
...
KRA28           session             629411
116 rows selected.
```

Příklad, kill_locked_session 6/7

- A teď to dáme dohromady: zjisti sid a serial# procesů blokujících databázové objekty uživatele KRA28:

```
select vs.sid,vs.serial#
from v$locked_object lo
inner join dba_objects ob on lo.object_id=ob.object_id
inner join v$session vs on vs.sid=lo.session_id
where lo.ORACLE_USERNAME='KRA28';
```

```
       SID    SERIAL#
---------- ----------
      2426       17516
```

- Můžeme tedy blokující proces ukončit:

```
ALTER SYSTEM KILL SESSION '2426, 17516';
System KILL altered.
```

Příklad, kill_locked_session 7/7

- Nyní nejsou v DBS žádné procesy blokující databázové objekty uživatele KRA28:

```
select lo.oracle_username, lo.object_id
from v$locked_object lo
where lo.oracle_username='KRA28';

no rows selected
```

- Nicméně, zatím se nejedná o uloženou proceduru (splňující zadání).

Reference

- Portál materiálu k Oracle:
  `https://docs.oracle.com/en/database/oracle/oracle-database/21/books.html`
  - PL/SQL Language Reference
  - PL/SQL Packages and Types Reference