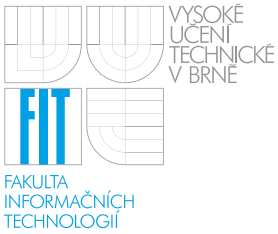


Tvorba databázových aplikací v prostředí Oracle Form Builder

IDS

Studijní opora

Ing. Jaroslav Ráb
4. října 2006

Tento učební text vznikl za podpory projektu „Zvýšení konkurenceschopnosti IT odborníků – absolventů pro Evropský trh práce“, reg. č. CZ.04.1.03/3.2.15.1/0003. Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky.

Obsah

Kapitola 1

Úvod

0:10

1.1 Slovo autora

Tato studijní opora slouží pro potřeby cvičení v předmětu „Databázové systémy“. Představuje pomůcku pro tvorbu databázových aplikací v prostředí databázového systému Oracle s využitím vývojového prostředí Oracle Form Builder.

Obsahem této studijní opory je úvod do tvorby databázových aplikací v prostředí databázového systému Oracle. Je psána tak, aby pokryla základní požadavky kladené na projekt z předmětu „Databázové systémy“. Obsahuje pouze popis nejpoužívanějších funkcí vývojového prostředí Oracle Form Builder. Popis všech funkcí toto vývojového nástroje lze najít v příslušné dokumentaci. Studijní opora si neklade za cíl popis základních principů databázových systémů, jako např. jazyk SQL, transakční zpracování atd. Tyto jsou součástí přednášek předmětu „Databázové systémy“. Cílem cvičení předmětu „Databázové systémy“ je seznámení se z databázovým systémem Oracle, vývojem aplikací v prostředí Oracle Form Builderu a aplikace znalostí získaných v rámci přednášek při tvorbě jednoduché databázové aplikace.

Následující text je výběrem těch nejdůležitějších částí cvičení, které jsou doplněny o podrobné komentáře i o prvky vyžadované pro samostudium. Jedná se především o zavedení grafických symbolů (tzv. piktogramů) pro označení specifických částí textu. Význam jednotlivých piktogramů je v tabulce 1.1.

	Čas potřebný pro studium		Otázka, příklad k řešení
	Cíl		Počítačové cvičení, příklad
	Definice		Příklad
	Důležitá část		Reference
	Rozšiřující látka		Správné řešení
	Obtížná část		Souhrn
			Slovo tutora, komentář
			Zajímavé místo

Tabulka 1.1: Význam používaných piktogramů

1.2 Cíl cvičení a struktura opory

Cílem cvičení je, aby se studenti seznámili a vyzkoušeli tvorbu databázových aplikací v prostředí Oracle. Aby se seznámili s procesem tvorby softwaru v prostředí Oracle Form Builder a procvičili znalosti jazyka SQL, získané v textu přednášek předmětu „Databázové systémy“.

Probíraná látka je organizována následovně. Nejdříve se seznámíme se základy jazyka PL/SQL, který je stavebním kamenem tvorby databázových aplikací v prostředí Oracle. Dále se zaměříme na tvorbu základních databázových objektů, které se používají pro tvorbu aplikací na straně serveru a to na uložené procedury a funkce, databázové triggery a sekvence. Třetí část je věnována tvorbě klientských aplikací v prostředí Oracle Form Builderu s podporou grafického uživatelského rozhraní.

Jak bylo avizováno v předchozím textu, tato studijní opora popisuje základy tvorby databázových aplikací v prostředí Oracle. Text studijní opory neobsahuje kompletní informace o diskutovaných tématech. Takový text by byl příliš rozsáhlý a náročnost přesahuje rámce předmětu. Pokud máte zájem o hlubší studium a prohloubení znalostí z této oblasti, použijte dokumentaci dodávanou k databázovému systému Oracle a vývojovému prostředí Oracle Form Builder.

[Oracle] Oracle10g Database Online Documentation. (Release 2 (10.2)). Oracle Corporation., dostupné na adrese http://otn.oracle.com.

[Lul96] Lulushi, A.: Developing Oracle Forms applications, Prentice Hall, 1996. ISBN 0-13-531229-9.

Kapitola 2

Úvod do jazyka PL/SQL

0:50

PL/SQL je procedurální jazyk (PL), který zahrnuje nativní podporu některých příkazů jazyka SQL (Structured Query Language). Jazyk PL/SQL můžeme použít k programování funkcí informačního systému pomocí uložených procedur a funkcí, balíků, trigerů událostí databázového systému nebo jej můžeme využít pro zvýšení programové logiky spouštěných SQL příkazů. Dále je tento jazyk používán v dalších vývojových prostředích Oracle jako Oracle Form Builder (nástroj pro tvorbu formulářových aplikací) nebo Oracle Report Builder (nástroj pro tvorbu tiskových sestav). Nejprve začneme přehledem základních programových konstrukcí a syntaxí použitých v jazyce PL/SQL.

2.1 Přehled PL/SQL

Jazyk PL/SQL nerozlišuje velká či malá písmena pro klíčová slova a identifikátory. Kód v jazyce PL/SQL je seskupen do struktur, které nazýváme *bloky*. Jestliže vytváříme uložené procedury, funkce, balíky, trigery přiřazujeme takto vytvořeným blokům PL/SQL kódu identifikátory (jména). Pokud PL/SQL blok nemá jméno, nazýváme takovýto blok *anonymní*. Příklady uvedené v této kapitole budou obsahovat anonymní bloky PL/SQL kódu. Příklady zabývající se vytvářením pojmenovaných bloků budou uvedeny později.
Blok PL/SQL kódu se skládá ze 3 částí, kde ne všechny musí být povinně uvedeny:

Část	**Výskyt**	**Popis**
Deklarační	Volitelný	Slouží pro deklaraci a inicializaci proměnných, kurzorů a výjimek použitých v bloku
Příkazová	Povinný	Obsahuje příkazy pro řízení prováděného toku kódu (jako je příkaz větvení **if** nebo cyklus), příkazy a přiřazení hodnot deklarovaným proměnným
Zpracování výjimek	Volitelný	Slouží pro zpracování chyb (výjimek) vzniklých v příkazové části

Uvnitř PL/SQL bloku, první část je deklarační část. V této deklarační části definujeme proměnné, kurzory a výjimky, které budou v bloku použity. Deklarační část začíná klíčovým slovem **declare** a končí začátkem příkazové části, které je indikováno klíčovým slovem **begin**. Za příkazovou částí následuje část pro zpracování výjimek, jejíž začátek je identifikován klíčovým slovem **exception**. Celý PL/SQL blok je ukončen klíčovým slovem **end**.

Struktura typického PL/SQL bloku je následující:


```
DECLARE
  <deklarační část>
BEGIN
  <příkazová část>
EXCEPTION
  <část zpracování výjimek>
```

```
END;
```

V následujícím textu této kapitoly bude uveden popis jednotlivých částí PL/SQL bloku.

2.2 Deklarační část

Deklarační část není povinná. Pokud je uvedena, začíná touto částí daný PL/SQL blok. Deklarační čast začíná klíčovým slovem **DECLARE** pro anonymní blok, následované seznamem deklarací proměnných, kurzorů a výjimek. Můžeme deklarovat proměnné jednoduchých datových typů, kompozitních (složených) datových typů, deklarovat konstanty (proměnné které musí být v deklarační části inicializovány hodnotou a tato hodnota se niž nedá změnit). Dále v deklarační části lze deklarovat kurzory, kurzorové proměnné, výjimky, nové datové typy a pojmenované PL/SQL bloky (procedury nebo funkce). Syntaxe deklarace proměnné je následující:


```
  jméno_identifikátoru datový_typ [CONSTANT] [[:= | DEFAULT]
    hodnota|výraz];
```

Jméno identifikátoru musí začínat alfa znakem, dále může obsahovat alfanumerické znaky nebo speciální znaky _,$,#. Maximální délka jména identifikátoru je 30 znaků. Tyto vlastnosti platí pro všechny identifikátory v jazyce PL/SQL. Každá deklarace musí být ukončena znakem **;**. Následující příklad složí k výpočtu obsahu kruhu. Výsledek tohoto výpočtu je uložen do tabulky KRUH. Tabulka KRUH [1] obsahuje dva sloupce pro uložení hodnot poloměru a vypočítaného obsahu.


```
DECLARE
  pi CONSTANT NUMBER(9,7) := 3.1415927;
  v_polomer INTEGER(5) := 3;
  v_obsah NUMBER(14,2);
BEGIN
  v_obsah := pi*power(v_polomer,2);
  INSERT INTO KRUH(polomer,obsah)
    VALUES(v_polomer,v_obsah);
END;
```

Příkaz **END;** ukončuje PL/SQL blok. V závislosti na konzoli, kterou jsme připojeni k databázovému serveru, přes kterou tento blok spouštíme, bude pravděpodobně nutné na konec tohoto bloku uvést znak / na novém řádku, abychom blok spustili. V případě použití konzole SQL*Plus tomu tak bude a po spuštění obdržíme následující odpověď:

```
PL/SQL procedure successfully completed.
```

Abychom se ujistili o správném provedení PL/SQL bloku, můžeme provést dotaz, kterým vybereme řádky z tabulky kruh, které byly vloženy PL/SQL kódem. Tento dotaz spolu s výsledkem zobrazuje řádek tabulky kruh vytvořený PL/SQL blokem.


```
select * from kruh;

   POLOMER      OBSAH
---------- ----------
         3      28,27
```

[1] Vytvořit tabulku lze příkazem: CREATE TABLE KRUH(polomer INTEGER(5,0) PRIMARY KEY, obsah NUMBER(14,2));

Výsledek ukazuje vložení jednoho řádku do tabulky KRUH PL/SQL blokem.

V první části PL/SQL bloku, byly deklarovány tři proměnné. Musíme deklarovat proměnné, které budou použity v příkazové části PL/SQL bloku. První deklarovaná proměnná je *pi*, které je přiřazena hodnota (inicializována). Pomocí klíčového slova **CONSTANT** má proměnná vlastnost konstanty. Protože je tato proměnná deklarována jako konstanta, její hodnota nemůže být změněna v příkazové části. Hodnota je přiřazena pomocí přiřazovacího operátoru **:=** :


```
pi     CONSTANT NUMBER(9,7):=3.1415927;
```

Následující dvě proměnné jsou deklarovány, z nichž první je inicializována, druhá nikoliv.


```
v_polomer INTEGER(5) := 3;
v_obsah   NUMBER(14,2);
```

Každá proměnná, která nebyla v deklarační části inicializována, obsahuje hodnotu **NULL**. Tato hodnota je prázdná, nedefinovaná. V deklarační části místo operátoru přiřazení pro inicializaci proměnné nebo konstanty můžeme použít klíčové slovo **DEFAULT**. Zápis by byl potom následující:


```
v_polomer INTEGER(5) default 3;
```

V uvedených příkladech jsme použili datové typy INTEGER a NUMBER. Datové typy jazyka PL/SQL zahrnují všechny platné datové typy jazyka SQL stejně jako komplexní datové typy, které jsou nejčastěji vytvořené podle struktury použitých dotazů. Tabulka 2.1 obsahuje všech podporovaných datových typů v PL/SQL. Zaměříme se pouze na některé nejčastěji používané datové typy. Ze skalárních to jsou typy NUMBER, VARCHAR2, CHAR, DATE a BOOLEAN. Z kompozitních to budou typy záznam (Record) a indexovaná tabulka (Index-by Table).

2.2.1 Skalární datové typy

Datový typ NUMBER je základním číselným datovým typem, od kterého jsou odvozeny další podtypy jako INTEGER, FLOAT, atd. Při deklaraci proměnná typy NUMBER můžeme uvést velikost a přesnost pro hodnotu proměnné. První parametr v kulaté závorce představuje velikost, která znamená počet číslic, které budou uloženy v paměti včetně oddělovače desetinné části, kterým je tečka. Druhý parametr představuje přesnost, který znamená kolik číslic je vyhrazeno z velikosti pro desetinnou část.

Datový typ VARCHAR2 je základním datovým typem pro řetězce proměnné délky. Při deklaraci proměnné tohoto typu musíme v závorce uvést počet znaků, který bude možné do paměťového místa této proměnné uložit. Maximum je 32767 znaků.

Datový typ CHAR je datový typ pro řetězce pevné délky. Pokud uložíme hodnotu s menším počtem znaků, než je deklarovaná velikost, je řetězec doplněn znaky mezera zprava do plné délky řetězce.

Datový typ DATE v sobě zahrnuje informaci o datu a čase s přesností na 1 sekundu. Hodnota pro typ DATE i v paměti uložena jako číselná hodnota, kde celá část slouží k vyjádření data, desetinná k vyjádření času. Pokud k hodnotě typu DATE přičteme číselnou hodnotu, pak celá část představuje dny, h/24 hodiny, m/1440 minuty, s/86400 sekundy.

Datový typ BOOLEAN je datový typ pro vyjádření pravdivostní hodnoty. Tento typ není v jazyce SQL zastoupen. Je až součástí jazyka PL/SQL. Hodnoty, kterých může nabývat jsou TRUE, FALSE a NULL. Atribut %TYPE se používá pro deklaraci proměnné typu shodného s typem proměnné nebo sloupce tabulky, již deklarované. Tato

konstrukce nám umožňuje vytvářet typy proměnných, které jsou odvozeny od předchozích deklarací (převážně databázových tabulek) a dává nám možnost opětovné deklarace těchto objektů a následnou správnou funkci PL/SQL kódu bez nutnosti jej přepisovat.


```
DECLARE
  prom1 kruh.polomer%TYPE;
  prom2 NUMBER(38);
  prom3 prom2%TYPE;
```

2.2.2 Kompozitní datové typy

Typ záznam je datový typ, který obsahuje jednu nebo více položek skalárního datového typu nebo typu záznam. Nejdříve v deklarační části bloku musíme deklarovat nový datový typ a to příkazem TYPE následujícím způsobem:


```
  TYPE identifikátor_typu IS RECORD (
    identifikátor_položky datový_typ,
    identifikátor_položky datový_typ,
    ...
  );
```

Potom můžeme deklarovat proměnnou tohoto nově vytvořeného typy.

Typ indexové tabulky se používá pro práci s polem. Indexová tabulka je indexována indexem nejčastěji číselného typu (BINARY_INTEGER). Další typ indexu tabulky, který lze použít je datový typ VARCHAR2. Pak hovoříme o hash tabulce. Velikost tabulky je omezena pouze datovým typem indexu a velikostí dostupné paměti. Tabulka může být řídká tzn: nemusíme naplnit všechny položky tabulky na jednotlivých indexech. Indexová tabulka je uložená v paměti prováděcí jednotky PL/SQL. Nejedná se o databázovou tabulku, jejíž řádky jsou permanentně uloženy v databázi. Deklaraci typu indexové tabulky zapisujeme následujícím způsobem:


```
TYPE identifikátor_typu IS TABLE OF
  [identifikátor_jednoduchého_datového_typu |
   identifikátor_kompozitního_datového_typu]
  INDEX BY [BINARY_INTEGER | VARCHAR2];
```

Na základě deklarace typu indexové tabulky lze deklarovat proměnnou tohoto typu.


```
DECLARE
  TYPE tab_typ IS TABLE OF VARCHAR2(64) INDEX BY BINARY_INTEGER;
  jmena tab_typ;
  ...
```

Atribut %ROWTYPE se používá podobně jako atribut %TYPE s tím rozdílem, že vytváří kompozitní datový typ proměnné, odvozený od definice tabulky (položky tohoto typu mají shodný název i typ jako jednotlivé sloupce tabulky). Dále se tento atribut používá pro deklaraci proměnné odvozené od předem deklarovaného kurzoru.

2.2.3 Kurzory

V deklarační části PL/SQL bloku deklarujeme kurzory, které použijeme v příkazové části. Deklarovat můžeme kurzor bez parametrů nebo s parametry. Příkaz jazyka SQL SELECT, který je použit při deklaraci kurzoru se shoduje se syntaxí tohoto příkazu podporovaného databázovým serverem Oracle. Syntaxe deklarace kurzoru je následující:

Skalární datové typy			
BINARY_INTEGER	INT	NVARCHAR2	STRING
BOOLEAN	INTEGER	NUMBER	TIMESTAMP
CHAR	INTERVAL DAY TO SECOND	NUMERIC	TIMESTAMP WITH LOCAL TIMEZONE
CHARACTER	INTERVAL YEAR TO MONTH	PLS_INTEGER	TIMESTAMP WITH TIME ZONE
DATE	LONG	POSITIVE	UROWID
DEC	LONG RAW	POSITIVEN	VARCHAR
DECIMAL	NATURAL	REAL	VARCHAR2
DOUBLE PRECISION	NATURALN	SIGNTYPE	RAW
FLOAT	NCHAR	SMALLINT	ROWID
Kompozitní datové typy			
RECORD	TABLE	VARRAY	
Referenční datové typy			
REF CURSOR	REF *object_type*		
LOB typy			
BFILE	BLOB	CLOB	NCLOB

Tabulka 2.1: Přehled datových typů

```
CURSOR jméno_kurzoru[(jméno_parametru datový_typ, ...)] IS
  SELECT ...;
```

V následujícím příkladu je deklarován a použit kurzor pro získání řádku tabulky POLOMER. POLOMER je tabulka složená z jednoho sloupce, který se jmenuje Polomer, který obsahuje hodnotu poloměru, která se používá v následujících příkladech. Kurzor je deklarován v deklarační části a proměnná pojmenovaná *hodnota_polomeru* je deklarována s datovým typem odvozeným od výsledku kurzoru.


```
DECLARE
  pi CONSTANT NUMBER(9,7) := 3.1415927;
  vypocitany_obsah NUMBER(14,2);
  CURSOR  polomer_cursor IS SELECT * FROM kruh ORDER BY polomer;
  radek polomer_cursor%ROWTYPE;
```

Kromě %ROWTYPE deklarace můžeme také použít %TYPE deklaraci k deklaraci proměnné stejného datového typu jakého je proměnná uvedená před %TYPE.

```
DECLARE
  CURSOR polomer_cursor IS SELECT * FROM kruh;
  radek polomer_cursor%ROWTYPE;
  v_polomer radek.polomer%TYPE;
```

Výhody odvození datového typu proměnné pomocí %ROWTYPE nebo %TYPE od jiné proměnné je v umožnění definice datových typů nezávisle na základních datových strukturách. Jestliže změníme definici sloupce polomer tabulky POLOMERY z datového typu NUMBER(5) na NUMBER(4,2), nepotřebujeme modifikovat PL/SQL kód; datové typy deklarovaných proměnných budou odvozeny dynamicky z definice sloupce tabulky v době běhu programu.

2.3 Příkazová část

V příkazové části pracujeme s proměnnými a kurzory deklarovanými v deklarační části PL/SQL kódu. Příkazová část PL/SQL bloku začíná klíčovým slovem **BEGIN**. V příkazové části PL/SQL bloku používáme SQL a PL/SQL příkazy. Použití SQL příkazů se budeme věnovat v kapitole Interakce s databázovým serverem. Základní PL/SQL příkazy budou popsány v kapitole Příkazy jazyka PL/SQL. Příkazová část PL/SQL bloku nesmí být prázdná, musí obsahovat alespoň jeden příkaz. Dále umožňuje zanořování PL/SQL bloků, kde každý vložený PL/SQL blok je příkazem. Příklad zanoření PL/SQL bloků:


```
DECLARE
  i INTEGER;
BEGIN
  i:=i+15;
  DECLARE
    j INTEGER;
  BEGIN
    j:=i;
    i:=10;
  END;
END;
```

2.3.1 Zanoření bloků a rozsah platnosti identifikátorů

V souvislosti se zanořením PL/SQL bloků je třeba brát v úvahu rozsah platnosti identifikátorů proměnných, kurzorů atp. Proměnná je platná v bloku, ve kterém byla deklarována a ve všech blocích zanořených v tomto bloku. Pokud budeme pracovat s proměnnou, která není v bloku platná, ukončí se provádění kódu s výjimkou. Pokud v zanořeném bloku deklarujeme proměnnou se stejným identifikátorem jako je v nadřazeném bloku, pracujeme potom s proměnnou deklarovanou v tomto zanořeném bloku, nikoliv v nadřazeném bloku. K žádnému konfliktu (vyvolání výjimky) z důvodu shodného identifikátoru nedojde.

2.4 Zpracování výjimek

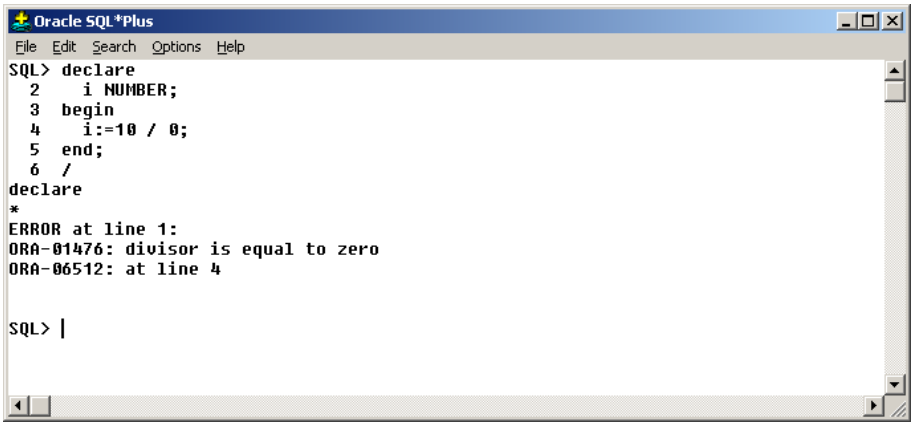
Třetí volitelná část PL/SQL bloku se týká zpracování výjimek. Začíná klíčovým slovem EXCEPTION a musí obsahovat alespoň jeden identifikátor zpracovávané výjimky. Při provádění příkazů v příkazové části PL/SQL bloku může dojít k vyvolání výjimky. V takovém případě se ukončuje provádění příkazů v příkazové části a přechází se při provádění

kódu do části pro zpracování výjimek a hledá se identifikátor vyvolané výjimky a provedou se příkazy, které reagují na vzniklou výjimku. Pokud není nalezen identifikátor vyvolané výjimky (hovoříme o nezpracování vyvolané výjimky) je výjimka propagována do volajícího prostředí. Volající prostředí může být nadřazený blok, konzolové prostředí SQL Plus, prostředí programu, které PL/SQL kód vyvolalo např: C/C++, Cobol, prostředí .NET, Java atd. Příklad části pro zpracování výjimek:

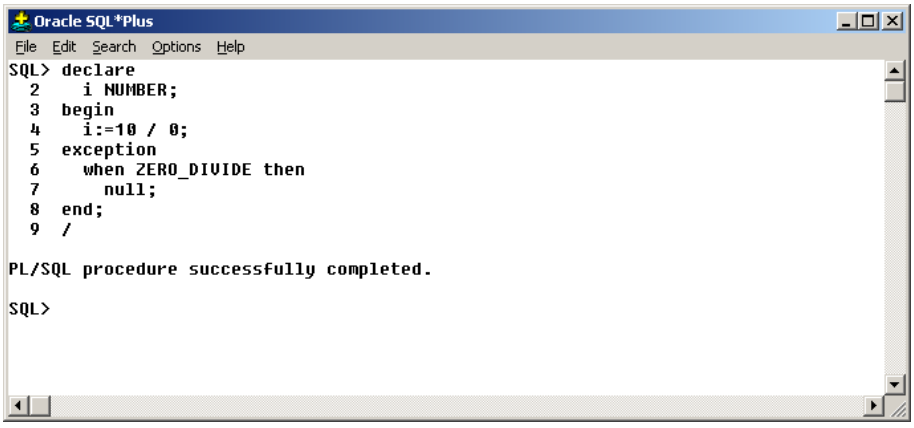
x+y

```
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    dbms_output.put_line('Deleni nulou !')
END;
```

Příklad propagace nezpracované výjimky je obrázku 2.1. P říklad zpracování výjimky je obrázku 2.2.



Obrázek 2.1: Propagace výjimky do prostředí SQL Plus



Obrázek 2.2: Propagace výjimky do prostředí SQL Plus

2.5 Komentáře zdrojového kódu

Komentáře zdrojového kódu může zapsat pomocí dvou konstrukcí. První je jedno řádkové komentáře, která začínají posloupností znaků – následované textem komentá ře až

do konce řádku. Druhá varianta je shodná s jazykem C, kde víceřádkové komentáře zapisuje mezi začínající sekvenci znaků /* a ukončovací sekvenci */. Víceřádkové komentáře nelze do sebe zanořovat.

2.6 Pojmenované PL/SQL bloky - procedury a funkce

V deklarační části PL/SQL bloku lze na konci uvést definici procedur a funkcí. Definice procedury nebo funkce se skládá z deklarace rozhraní procedury nebo funkce a zápisu těla funkce. Při deklaraci rozhraní uvádíme pouze rodinu datového typu (řetězec, číselná hodnota, datum) bez omezení na velikost a přesnost. Takový kód pojmenovaného PL/SQL bloku můžeme opakovaně provádět jeho voláním v příkazové části PL/SQL bloku, v němž byl definován. Jsou to tedy objekty dočasné, vytvořené v paměti PL/SQL interpretru. Syntaxe definice funkce je následující:


```
FUNCTON jmeno_funkce([parametr1 [IN|OUT|IN OUT] datovy_typ, ...])
  RETURN datovy_typ { IS|AS }
BEGIN
  <PL/SQL prikazy ...>
  RETURN hodnota;
END [jmeno_funkce];
```

Syntaxe definice procedury je následující:


```
PROCEDURE jmeno_procedury([parametr1 [IN|OUT|IN OUT] datovy_typ,
                          ... ]) { IS | AS }
BEGIN
  <PL/SQL prikazy ...>
END [jmeno_procedury];
```

Příklad PL/SQL funkce:


```
DECLARE
  i NUMBER;
  FUNCTION secti(a IN NUMBER, b IN NUMBER) RETURN NUMBER IS
  BEGIN
    RETURN a+b;
  END secti;
BEGIN
  i:=secti(1,3);
END;
```

Modifikátory IN, OUT a IN OUT se používají pro zápis pro způsobu předání hodnoty argumentu procedury nebo funkce. Modifikátor IN povoluje pouze čtení hodnoty v PL/SQL bloku (předání hodnotou), OUT povoluje zápis a na začátku ukládá hodnotu NULL do parametru, IN OUT umožňuje čtení a zápis hodnoty (předání odkazem).

2.7 Uložené procedury a funkce a další databázové objekty

Uložené procedury a funkce jsou perzistentní objekty zapsány v jazyce PL/SQL a jsou uloženy ve schématu databáze. Spouštět takovéto objekty můžeme z jiných uložených procedur a funkcí, z anonymních PL/SQL bloků, balíčků, databázových triggerů, prostředí konzoly SQL Plus pomocí příkazu EXECUTE (zkráceně EXEC). Syntaxe vytváření

uložených procedur a funkcí je shodná se syntaxí uvedenou v předchozí podkapitole. Pouze pro vytvoření uloženého objektu v databázi je třeba použít příkaz CREATE jazyka SQL. Syntaxe pro vytvoření takového objektu je následující:


```
CREATE [OR REPLACE] { procedura | funce PL/SQL definice ... }
```

Kapitola 3

Interakce s databázovým serverem

0:20

V této kapitole se budeme věnovat použití příkazů jazyka SQL v prostředí PL/SQL. Jazyk PL/SQL nativně podporuje pouze část příkazů a to příkazy jazyka DML, příkaz SELECT, který je modifikovaný pro potřeby práce s proměnnými.

3.1 Příkaz SELECT

Tento příkaz slouží stejně jako v jazyce SQL pro získání dat (řádků, vět) z tabulky nebo pohledu uloženého v databázi. Jeho syntaxe však modifikována a rozšířena o klauzuli **INTO**, která slouží pro přiřazení získaných hodnot do proměnných v PL/SQL prostředí. Syntaxe je následující:


```
SELECT sloupec1[, sloupec2, ...] INTO promenna1[, promenna2, ...]
  FROM tabulka1 [,tabulka2, ...]
  [WHERE klauzule]
  [GROUP BY klauzule]
  [HAVING klauzule]
  [ORDER BY klauzule];
```

Pro takový příkaz SELECT v prostředí PL/SQL platí jedno významné pravidlo. Příkaz při spuštění musí vracet právě jeden řádek. Pokud nevrací žádný, je příkaz ukončen s výjimkou NO_DATA_FOUND. Pokud je tímto příkazem vybráno více řádků, je ukončen s chybou TOO_MANY_ROWS. Pokud potřebujeme zpracovat předem neurčený počet řádků z tabulky/pohledu, používáme kurzory, kurzorové cykly. Takovýto příkaz SELECT nejčastěji používáme pro dotazy, kde se v klauzuli WHERE omezujeme na podmínku vyhledání řádku podle primárního klíče nebo kde používáme agregační funkce. Příklad pro zjištění počtu řádků tabulky POLOMER:


```
DECLARE
  pocet INTEGER;
BEGIN
  SELECT count(*) INTO pocet FROM kruh;
END;
```

V příkazu SELECT můžeme též pracovat s proměnnými kompozitního datového typu.


```
DECLARE
  vysledek kruh%ROWTYPE;
begin
  SELECT * INTO vysledek FROM kruh WHERE polomer=1;
end;
```

3.2 Příkaz INSERT

Příkaz INSERT v prostředí PL/SQL je shodný se zápisem jazyce SQL. Syntaxe je následující:


```
INSERT INTO tabulka [(sloupec1, sloupec2, ...)]
VALUES (hodnota1, hodnota2, ...);
```

3.3 Příkaz DELETE

Příkaz DELETE v prostředí PL/SQL je shodný se zápisem jazyce SQL. Syntaxe je následující:


```
DELETE [FROM] tabulka
[WHERE podmínka ...];
```

3.4 Příkaz UPDATE

Příkaz UPDATE v prostředí PL/SQL je shodný se zápisem jazyce SQL. Syntaxe je následující:


```
UPDATE tabulka SET sloupec1=hodnota1 [, sloupec2=hodnota2, ...]
[WHERE podmínka ...];
```

3.5 Příkaz MERGE

Příkaz MERGE v prostředí PL/SQL je shodný se zápisem jazyce SQL. Syntaxe je následující:


```
MERGE INTO tabulka1 USING tabulka2 ON (podmínka)
  WHEN MATCHED THEN
    UPDATE SET sloupec1=hodnota1 [,sloupec2=hodnota2, ...]
  WHEN NOT MATCHED THEN
    INSERT (sloupec1 [,sloupec2, ...]) VALUES (hodnota1 [,hodnota2, ...]);
```

Kapitola 4

Příkazy jazyka PL/SQL

1:20

4.1 Kurzory

4.1.1 Explicitní kurzory

Pro práci s explicitními kurzory (deklarovanými v deklarační části PL/SQL bloku) se používají následující příkazy: **OPEN, CLOSE, fetch**. Příkaz OPEN složí k otevření kurzoru. Otevření kurzoru představuje spuštění SELECT dotazu definovaného pro kurzor. Tím se identifikuje množina řádků (tzv: Result Set) a nastaví se ukazatel na první řádek této množiny. Poté příkazem FETCH získáme hodnoty aktuálního řádku, na který ukazuje kurzor, do PL/SQL proměnných a dále s nimi pracujeme. Příkaz FETCH automaticky přesune ukazatel na další řádek. Opakovaným voláním příkazu FETCH získáme hodnoty všech řádků. Pro ukončení práce s kurzorem použijeme příkaz CLOSE. Po uzavření kurzoru je možné jej znovu otevřít příkazem OPEN a pracovat s ním. Syntaxe příkazů OPEN, FETCH a CLOSE je následující:


```
OPEN jmeno_kurzoru[(hodnota_parametru, ...)];
FETCH jmeno_kurzoru INTO promenna[, promenna, ...];
CLOSE jmeno_kurzoru;
```

Příklad práce s kurzorem bez parametrů:

```
pi      constant NUMBER(9,7) := 3.1415927;
v_obsah   NUMBER(14,2);
cursor  kruh_cursor IS SELECT * FROM kruh;
radek kruh_cursor%ROWTYPE;
BEGIN
  OPEN kruh_cursor;
  FETCH kruh_cursor INTO radek;
  v_obsah := pi*power(kruh.polomer,2);
  UPDATE kruh SET obsah=v_obsah WHERE polomer=radek.polomer;
  CLOSE kruh_cursor;
END;
```

4.1.2 Atributy kurzorů

Atributy kurzorů se používají pro získání důležitých informací o kurzoru. Ty zahrnují zda je kurzor otevřen, počet řádků, který byl již získán operací FETCH. Atributy vrací hodnotu určitého typu a zapisují se za jméno kurzoru spolu s operátorem %. Následující tabulka obsahuje přehled jednotlivých atributů kurzorů.

Jméno atributu	Návratová hodnota	Popis
%ISOPEN	**Boolean**	Vrací hodnotu TRUE pokud je kurzor otevřen, jinak FALSE.
%ROWCOUNT	**Number**	Vrací počet řádků získaných operací FETCH z kurzoru.
%FOUND	**Boolean**	Vrací hodnotu TRUE pokud poslední operace FETCH vrátila hodnotu řádku. Před prvním voláním FETCH vrací hodnotu NULL. Po získání poslední hodnoty řádku kurzoru vrací hodnotu FALSE.
%NOTFOUND	**Boolean**	Vrací hodnotu FALSE pokud poslední operace FETCH vrátila hodnotu řádku. Před prvním voláním FETCH vrací hodnotu NULL. Po získání poslední hodnoty řádku kurzoru vrací hodnotu TRUE.

4.1.3 Implicitní kurzory

Implicitní kurzor se používá pro příkazy SELECT (s INTO klauzulí), INSERT, UPDATE a DELETE přímo zapsané v PL/SQL bloku. Implicitní kurzor má identifikátor SQL. Atributy pro implicitní kurzor můžeme použít stejně jako pro explicitní kurzory. Atribut %ISOPEN má vždy hodnotu FALSE, %FOUND a %NOTFOUND podle toho, zda se poslední volaný SQL příkaz provedl alespoň nad jedním řádkem tabulky nebo pohledu. Atribut %ROWCOUNT vrací počet řádků, se kterými se pracovalo s posledním SQL příkazem. Následující příklad ukazuje práci s implicitním kurzorem a zjišťuje počet řádků, které byly smazány příkazem DELETE.


```
DECLARE
  pocet NUMBER;
BEGIN
  DELETE FROM kruh;
  pocet:=SQL%ROWCOUNT;
  dbms_output.put_line('Bylo smazano '||to_char(pocet)||
                       ' radku z tabulky kruh.');
END;
```

4.2 Podmíněný příkaz IF

V PL/SQL kódu můžeme použít příkazy *IF*, *ELSE*, *ELSIF* a *CASE* k řízení běhu programu. Formát příkazu **IF** je následující:


```
IF <podmínka> THEN <PL/SQL příkaz(y)>
[ELSIF <podmínka> THEN <PL/SQL příkaz(y)>]
[ELSE <PL/SQL příkaz(y)>]
END IF;
```

Podmíněné příkazy **IF** můžeme do sebe libovolně zanořovat, jak uvedeno v následujícím příkladu:


```
IF i>30 THEN
  i:=i+20;
  IF x<10 THEN
    BEGIN
```

```
      x:=0;
    END;
  END IF;
ELSE
  i:=i-20;
END IF;
```

4.2.1 Relační operátory

Relační operátory jsou následují:

Operátor	Význam
=	Rovnost
<>, !=, ~=, ^=	Nerovnost
<	Menší
>	Větší
<=	Menší nebo roven
<=	Větší nebo roven

4.2.2 Operátor IS NULL

Operátor IS NULL slouží pro zjištění, zda má daný výraz nebo proměnná NULL hodnotu. Vrací hodnotu TRUE pokud je výraz roven NULL hodnotě, jinak vrací FALSE. Nelze použít operátor porovnání = pro zjištění NULL hodnoty.

4.2.3 Operátor LIKE

Podobně jako v jazyce SQL můžeme použít operátor LIKE pro porovnání řetězců podle vzoru. Pro libovolný znak použijeme znak _. pro libovolnou (i prázdnou) sekvenci znaků použijeme znak %.

4.2.4 BETWEEN

Tento operátor používáme pro zjištění, zda je hodnota výrazu v intervalu.

4.2.5 IN

Tento množinový operátor slouží pro zjištění, zda je hodnota výrazu shodná s některou ze seznamu. Hodnota NULL je seznamu ignorována a je nutné použít operátor IS NULL pro zjištění této hodnoty.

Následují pravdivostní tabulky pro logické operátory AND, OR a NOT.

AND	TRUE	FLASE	NULL
TRUE	TRUE	FLASE	NULL
FALSE	FALSE	FLASE	FALSE
NULL	NULL	FLASE	NULL

OR	TRUE	FLASE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FLASE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

4.3 Cykly

Pro opakované provádění PL/SQL příkazů používáme cyklus (např.: zpracování všech řádků které vrací kurzor). PL/SQL podporuje 3 typy cyklů:

Typ cyklu	Popis
jednoduchý cyklus	cyklus, který se opakuje dokud není vykonán příkaz **EXIT** nebo podmíněný příkaz **EXIT WHEN**
FOR cyklus	cyklus, který se vykoná n-krát (tzv.: cyklus s pevným počtem iterací)
WHILE cyklus	cyklus, který se opakuje dokud je splněna podmínka (tzv.: podmíněný cyklus)

4.3.1 Jednoduchý cyklus

Jednoduchý cyklus slouží pro opakované provádění sekvence PL/SQL příkazů dokud není zavolán příkaz **EXIT**, který cyklus ukončuje a provádění kódu přechází na následující příkaz za koncem cyklu. Cyklus začíná klíčovým slovem **LOOP**, příkaz **EXIT** nebo **EXIT WHEN** ukončuje cyklus, pokud je splněna zadaná podmínka. Klíčové slovo **END LOOP** identifikuje konec cyklu. Pokud cyklus příkaz EXIT nikdy nevyvolá, provádí se do nekonečna. Syntaxe jednoduchého cyklu je následující:

DEF

```
LOOP
  [EXIT;]
  [EXIT WHEN <podmínka>;]
END LOOP;
```

x+y

```
DECLARE
  pi CONSTANT NUMBER(9,7) := 3.1415927;
  v_polomer INTEGER(5);
  v_obsah NUMBER(14,2);
BEGIN
  v_polomer := 3;
  LOOP
    v_obsah := pi*power(v_polomer,2);
    INSERT INTO kruh VALUES(polomer,v_obsah);
    v_polomer:=v_polomer+1;
    EXIT WHEN obsah>100;
  END LOOP;
END;
```

4.3.2 FOR cyklus

Cyklus FOR používáme pokud předem známe počet iterací, které má cyklus provést. Syntaxe příkazu je postavena na konstrukci jednoduchého cyklu. Klíčové slovo **FOR** následované deklarací intervalu, ve kterém se bude pro danou proměnnou cyklus provádět je prefixem klíčového slova LOOP jednoduchého cyklu. Příkaz EXIT nebo podmíněný EXIT WHEN můžeme použít pro předčasné ukončení cyklu. Proměnná, která je použita pro řízení cyklu nemusí být předem deklarována. Do proměnné nelze v cyklu přiřadit jinou hodnotu. Lze ji pouze použít ve výrazu. Je deklarována cyklem automaticky na potřebný typ. Syntaxe FOR cyklu je následující:

DEF

```
FOR promenna IN dolni_mez .. horni_mez LOOP
  <PL/SQL prikazy;>
END LOOP;
```

Následující příklad je modifikací předchozího příkladu.


```
DECLARE
  pi constant NUMBER(9,7) := 3.1415927;
  obsah NUMBER(14,2);
BEGIN
  FOR polomer IN 3 .. 100 LOOP
    obsah := pi*power(polomer,2);
    INSERT INTO kruh VALUES(polomer,obsah);
  END LOOP;
END;
```

4.3.3 Podmíněný cyklus WHILE

Podmíněný cyklus WHILE se používáme, pokud počet iterací cyklu není předem známy a lze jej určit nějakou podmínkou. Syntaxe opět vychází ze syntaxe jednoduchého cyklu a je následující:


```
WHILE podminka LOOP
  <PL/SQL prikazy;>
END LOOP;
```

Opět platí, že předčasné ukončení cyklu lze provést voláním příkazu EXIT nebo EXIT WHEN. Následující příklad ukazuje použití podmíněného cyklu spolu s kurzorem s parametrem.


```
pi      constant NUMBER(9,7) := 3.1415927;
v_obsah   NUMBER(14,2);
cursor  kruh_cursor(p_polomer NUMBER) IS
    SELECT * FROM kruh WHERE polomer<p_polomer;
radek kruh_cursor%ROWTYPE;
BEGIN
  OPEN polomer_cursor(10);
  FETCH kruh_cursor INTO radek;
  WHILE kruh_cursor%found LOOP
    v_obsah := pi*power(radek.polomer,2);
    UPDATE kruh SET obsah=v_obsah WHERE polomer=radek.polomer;
    FETCH kruh_cursor INTO radek;
  END LOOP;
  CLOSE kruh_cursor;
END;
```

4.3.4 Kurzorové FOR cykly

Předchozí příklad pracuje se všemi řádky přes explicitní kurzor. Kód je sice jednoduchý, ale rozsáhlý. Pro zjednodušení byl jazyk PL/SQL rozšířen o práci s kurzory ve FOR cyklu. Následující příklad ukazuje použití kurzorového FOR cyklu:


```
pi      CONSTANT NUMBER(9,7) := 3.1415927;
v_obsah   NUMBER(14,2);
cursor  kruh_cursor(p_polomer NUMBER) IS
```

```
    SELECT * FROM kruh WHERE polomer<p_polomer;
BEGIN
  FOR rec in kruh_cursor(10) LOOP
    v_obsah := pi*power(rec.polomer,2);
    UPDATE kruh SET obsah=v_obsah WHERE polomer=rec.polomer;
  END LOOP;
END;
```

Kurzorový FOR cyklus automaticky provádí operaci OPEN, FETCH a CLOSE nad kurzorem. Použití může být ještě zjednodušeno a to tak, že lze použít kurzorový FOR cyklus přímo pro příkaz SELECT.


```
pi      constant NUMBER(9,7) := 3.1415927;
v_obsah   NUMBER(14,2);
BEGIN
  FOR rec in (SELECT * FROM kruh WHERE polomer<10) LOOP
    v_obsah := pi*power(rec.polomer,2);
    UPDATE kruh set obsah=v_obsah WHERE polomer=rec.polomer;
  END LOOP;
END;
```

4.4 Funkce pro práci s rětězci

Pro práci s řetězci (typu VARCHAR2 i CHAR) je dostupná celá řada funkcí. Operátor pro zřetězení dvou řetězců je ||. Následující tabulka obsahuje pouze základní funkce. Úplný seznam najdete v dokumentaci k jazyku PL/SQL.

Definice funkce	**Popis**
CONCAT(řetězec, řetězec)	Vrací řetězec zřetězením dvou řetězců.
LENGTH(řetězec)	Vrací délku řetězce.
SUBSTR(řetězec,začátek,délka)	Vrací podřetězec z řetězce od pozice znaku začátek v délce znaků.
INSTR(řetězec,pozice,výskyt)	Vrací pozici n-tého výskytu podřetězce v řetězci od zadané pozice.
LOWER(řetězec)	Vrací řetězec kde všechny velké znaky byly konvertovány na malé.
UPPER(řetězec)	Vrací řetězec kde všechny malé znaky byly konvertovány na velké.
INITCAP(řetězec)	Vrací řetězec kde všechny počáteční znaky slov byly konvertovány na velké, ostatní na malé. Oddělovačem slov v řetězci jsou mezera, „„“ , tabulátor a „.“.

4.5 Funkce pro práci s číselnými typy

Mezi základní funkce pro práci s číselnými typy patří následující:

Definice funkce	Popis
ROUND(hodnota, přesnost)	Vrací hodnotu zaokrouhlenou na zadanou přesnost.
TRUNC(hodnota)	Vrací hodnotu ořezanou na zadanou přesnost.
SQRT(hodnota)	Vrací odmocninu zadané hodnoty.
POWER(hodnota, mocnina)	Vrací n-tou mocninu hodnoty.
FLOOR(hodnota, přesnost)	Vrací nejbližší nižší hodnotu zadané přesnosti.
SIN, COS, TAN, ...	Vrací hodnotu dané goniometrické funkce.

4.6 Práce s indexovou tabulkou

Deklarace typu indexové tabulky a proměnné tohoto typu je uvedeno v kapitole **??**. Pro přístup k jednotlivým položkám na daném indexu indexové tabulky používáme index podle typu definice tabulky uvedený v (). Pokud je typ indexové tabulky typu záznam, k jednotlivým položkám záznamu přistupujeme pomocí kvantifikátoru „.“. Pro práci s indexovou tabulkou požíváme metody, které jsou uvedeny v následující tabulce.

Metoda	Popis
EXISTS(n)	Vrací hodnotu TRUE existuje na daném indexu indexové tabulky položka.
COUNT	Vrací počet položek indexové tabulky.
FIRST	Vrací index první položky, která se nachází v indexové tabulce.
LAST	Vrací index poslední položky, která se nachází v indexové tabulce.
PRIOR(n)	Vrací index nejbližší předchozí položky. NULL pokud je index n první v tabulce.
NEXT(n)	Vrací index nejbližší následující položky. NULL pokud je index n poslední v tabulce.

EXTEND[(n,i)]	Má 3 varianty. Zvětšuje tabulku buď o jednu NULL položku (volání bez parametru) nebo zvětšuje tabulku o n NULL položek (volání s jedním parametrem) na konec tabulky. Poslední varianta kopíruje i-tou položku n-krát na konec tabulky (volání s dvěma parametry).
TRIM[(n)]	Odstraňuje poslední položku (bez parametru) nebo posledních n položek z tabulky(volání s jedním parametrem). Do těchto položek se zahrnují i položky smazané metodou DELETE.
DELETE[(m,n)]	Maže celou tabulku (bez parametru), maže n-tou položku tabulky (volání s jedním parametrem) nebo maže n položek od pozice m (volání s dvěma parametry).

Následující příklad ukazuje práci s indexovou tabulkou.


```
DECLARE
  TYPE typ_tabulka IS TABLE OF kruh.obsah%TYPE
    INDEX BY BINARY_INTEGER;
  tabulka typ_tabulka;
BEGIN
  FOR rec in (SELECT * FROM kruh) LOOP
    tabulka(rec.polomer):=rec.obsah;
  END LOOP;
```

```
  j:=tabulka.FIRST;
  FOR i IN 1..tabulka.count LOOP
    dbsm_output.put_line('Polomer '||j||': obsah = '||tabulka(j));
    j:=tabulka.NEXT(j);
  END LOOP;
END;
```

Kapitola 5

Zpracování výjimek

0:40

Třetí část PL/SQL bloku slouží ke zpracování chyb (výjimek), které mohou být vyvolány v době provádění PL/SQL kódu. Výjimka je zpracována pomocí identifikátoru výjimky. Prostředí PL/SQL obsahuje několik předdefinovaných výjimek. Ne všechny chyby, ke kterým může dojít mají předdefinovaný identifikátor výjimky. Důvod je ten, že takových chyb může být několik desítek tisíc. Rozlišujeme 3 typy výjimek:

- Uživatelem definované výjimky
- Předefinované Oracle výjimky
- Nepředddefinované Oracle výjimky

K vyvolání výjimky může dojít automaticky systémem, pokud nastane taková situace. Příklad může být dělení nulou. Pokud v příkazové části PL/SQL bloku se objeví výraz kde je prováděno dělení nulou, je automaticky vyvolána předdefinovaná výjimka ZERO_DIVIDE. Pokud chceme vyvolat uživatelem definovanou výjimku, musíme nejprve v deklarační části výjimku deklarovat. Pak v příkazové části ji explicitně vyvolat příkazem **RAISE**. Provádění kódu se v případě vyvolání výjimky (explicitní nebo implicitní) přeruší a řízení se předává do části pro zpracování výjimek.

5.1 Uživatelem definované výjimky

Deklarace výjimky v deklarační části je následující:


```
jmeno_vyjimky exception;
```

Uživatelem deklarované výjimky se používají nejčastěji pro řízení běhu programu vyvoláním příkazem RAISE.


```
declare
  e exception;
begin
  if length('slovo')=5 then
    raise e;
  end if;
exception
  when e then
    dbms_output.put_line('Podminka byla vyhodnocena jako pravdiva.');
 end;
```

5.2 Předefinované Oracle výjimky

Tyto výjimky jsou deklarovány prostředím PL/SQL pro nejčastější chyby, které mohou být vyvolány. Zde je jejich výčet: ACCESS_INTO_NULL, COLLECTION_IS_NULL, CURSOR_ALREADY_OPEN, DUP_VAL_ON_INDEX, INVALID_CURSOR, INVALID_NUMBER,

LOGIN_DENIED, NO_DATA_FOUND, NOT_LOGGED_ON, PROGRAM_ERROR, ROWTYPE_MISMATCH, SELF_IS_NULL, STORAGE_ERROR, SUBSCRIPT_BEYOND_COUNT, SUBSCRIPT_OUTSIDE_LIMIT, SYS_INVALID_ROWID, TIMEOUT_ON_RESOURCE, TOO_MANY_ROWS, VALUE_ERROR, ZERO_DIVIDE.

Největší význam mají pro nás následující výjimky. Výjimka TOO_MANY_ROWS je vyvolána pokud příkaz SELECT s INTO klauzulí vrací více než jeden řádek. Výjimka NO_DATA_FOUND je vyvolána pokud příkaz SELECT s INTO klauzulí nevrací žádný řádek. Výjimka DUP_VAL_ON_INDEX je vyvolána pokud příkazem INSERT jsme se pokusili vložit řádek do tabulky s hodnotou primárního klíče, který již v tabulce je.

5.3 Nepředddefinované Oracle výjimky

Ostatní chyby, ke kterým může dojít nemají deklarovaný identifikátor výjimky. Tyto chyby lze zpracovat dvěma způsoby. První je univerzální identifikátor výjimky **OTHERS**. Ten zpracuje všechny výjimky, které nebyly zpracovány předchozími identifikátory výjimek v části pro zpracování výjimek. Druhý způsob je deklarace uživatelem definované výjimky a asociace této výjimky s konkrétní chybou systému. Každá chyba, která v systému může nastat má unikátní číselný kód. tento kód je záporné číslo. V textu chyby je prefixováno řetězcem ORA. Například chyba pro dělení nulou má kód ORA-01476. Tato asociace se provádí pomocí pragmy překladače **EXCEPTION_INIT**.


```
DECLARE
   uvaznuti EXCEPTION;
   PRAGMA EXCEPTION_INIT(uvaznuti, -60);
BEGIN
   ...
EXCEPTION
   WHEN uvaznuti THEN
      -- zpracování deadlocku
END;
```

V části pro zpracování výjimek lze pomocí operátoru OR spojovat část pro zpracování jednotlivých výjimek. Výjimka OTHERS by měla být uvedena jako poslední, pokud je použita.


```
EXCEPTION
   WHEN TOO_MANY_ROWS OR NO_DATA_FOUND THEN
     dbms_output.put_line('SELECT musi vracet prave jeden radek');
   WHEN OTHERS THEN
     NULL; -- tento blok nebude nikdy propagovat vyjimku
END;
```

5.4 RAISE_APPLICATION_ERROR

Procedura RAISE_APPLICATION_ERROR(code IN INTEGER, message IN VARCHAR2) se používá v uložených procedurách a funkcích pro vyvolání uživatelem definovaných výjimek. V prostředí databázového systému Oracle máme možnost použít 1000 chybových hlášení, které vypadají jako systémové. A to v rozsahu kódu od -20000 az po -20999.


```
CREATE OR REPLACE PROCEDURE vloz_kruh(r IN integer) IS
  pi CONSTANT NUMBER(9,7) := 3.1415927;
```

```
  v_obsah NUMBER(14,2);
BEGIN
  IF r<0 THEN
    RAISE_APPLICATION_ERROR(-20100,
    'Nelze vlozit zapornou hodnotu polomeru');
  END IF;
  v_obsah:=pi*power(r,2);
  INSERT INTO kruh(polomer,obsah) VALUES(r,v_obsah);
END;
```

Kapitola 6

Práce v prostředí Form Builderu

1:30

Vývojové prostředí Oracle Form Builder se vývojem do dnešní podoby několikrát transformovalo. Jako první se objevilo v únoru 1981 pro verzi 2 databázové systému Oracle v podobě nástroje Interactive Application Facility (IAF) tehdy vyvíjen společností Relational Software Inc. (RSI). Dnes je toto prostředí součástí Oracle Developer Suite. Aplikace vytvořené v tomto prostředí zahrnují grafické uživatelské rozhraní. Aplikace je možné provozovat v prostředí Microsoft Windows, X Window a dalších, pro který musí být dostupný interpreter. Vytvořený kód je přenositelný na libovolná grafická prostředí (pokud neobsahují speciální volání funkcí operačního systému).

6.1 Charakteristika vývojového prostředí a vytvořených aplikací

Vývojové prostředí slouží pro rychlý vývoj aplikací (Rapid Application Development) pro databázový systém Oracle. Programovacím jazykem je jazyk PL/SQL. Toto prostředí dále umožňuje spolupráci s dalšími prostředími jako Java (možnost použití JavaBeans). Program je řízen událostmi a kód v PL/SQL se zapisuje v podobě trigerů, které jsou těmito událostmi vyvolány. Kód trigeru je anonymní PL/SQL blok, který může obsahovat volání rozšířené množiny procedur a funkcí než bylo doposud představeno. Tyto procedury a funkce se týkají převážně práce s prvky grafického uživatelského rozhraní a interakce s databázovým systémem. Vytvořený kód aplikace je přeložen do mezikódu, který je potom při spuštění interpretován runtime softwarem. Prvky grafického uživatelského prostředí jsou běžné, jak se vyskytují v běžných operačních systémech (tlačítka, list/combo boxy, edit boxy, ...). Dále je možné přidávat grafy, které jsou generovány na základě dotazu nad daty v databázi. Zdrojový kód aplikace se ukládá do souboru s koncovkou **FMB** a přeložený kód pro spuštění do souboru s koncovkou **FMX**. Vytvořená aplikace představuje modul formuláře, který musí obsavat minimálně jedno okno. Formulář může obsahovat více oken. Záleží na přístupu, který si programátor osvojí a bude používat.

6.1.1 Hlavní okna prostředí Form Builderu

Po spustění Form Builderu se zobrací dialogové okno pro spuštění průvodce. Form Builder obsahu celou řadu průvodců, kteří slouží pro generování kódu a objektů aplikace. Automatizují tak některé činnosti, které jsme jinak nuceni dělat sami. Cokoliv, co je schopen vytvořit průvodce, jsme schopni sami vytvořit v prostředí Form Builderu.

6.1.2 Okno Navigátor objektů

Toto okno je nejdůležitější v celém systému. Umožňuje práci se všemy objety, které formulář může obsahovat. Zpřístupňuje kódy trigerů, které editujeme v okně PL/SQL editoru. Dále umožňuje zpřístupnit plátna, na kterých se nachází prvky grafického uživatelského rozhraní.

6.1.3 Okno pro editaci rozložení

Toto okno slouží pro editaci plátna (canvas), kde rozmisťujeme prvky grafického uživatelského rozhraní na plochu plátna. Na levé straně se nachází sada prvků, které můžeme na plochu plátna umístit.

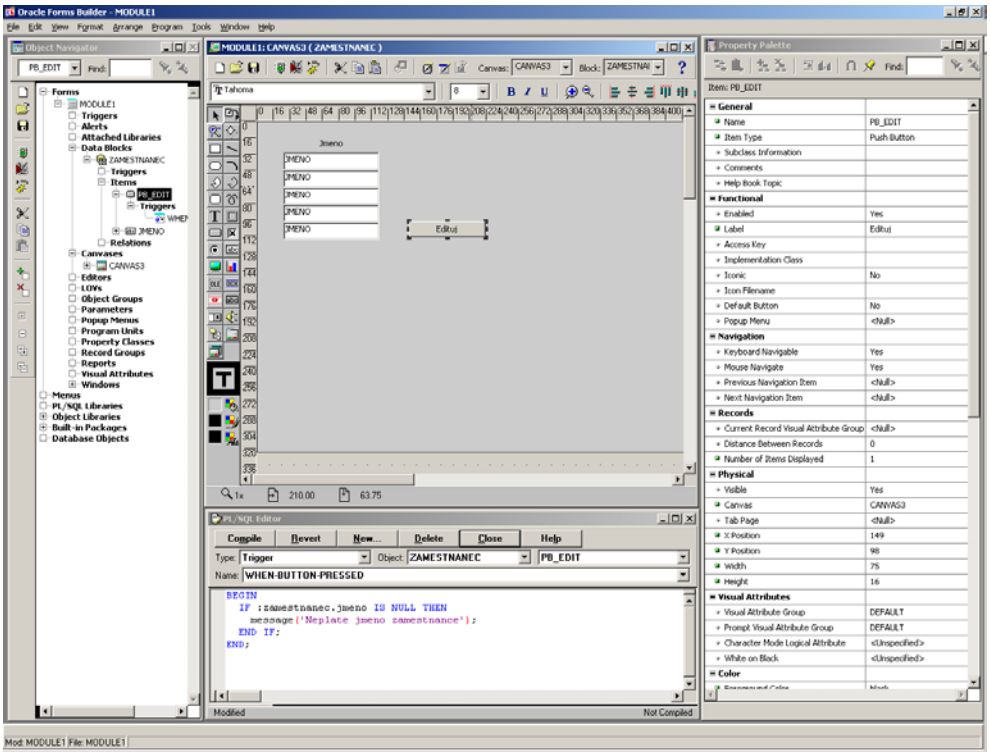
6.1.4 Okno pro editaci vlastností

Každý objekt ve formuláři (i samotný formulář) má nějaké vlastnosti, které lze měnit. Některé jsou statické, jiné dynamické. Dynamické vlastnosti lze měnit za běhu programu pomocí funkcí přes trigery zapsané v PL/SQL kódu. Hodnoty těchto vlastností editujeme v okně vlastností (property palette).

6.1.5 Okno pro editaci PL/SQL kódu

Toto okno slouží pro editaci kódu trigerů. Obsahuje tlačítka pro překlad kódu, návrat zpět, vytvoření nového trigeru a smazání trigeru.

Obrázek 6.1 zobrazuje prostředí Form Builderu spolu s jednotlivými okny pro práci s formulářem.



Obrázek 6.1: Vývojové prostředí Form Builderu

6.2 Struktura aplikace

Základní strukturou je datový blok. Datový blok obsahuje prvky grafického uživatelského rozhraní. Těmito prvky jsou textové položky, tlačítka, list boxy a další prvky, které

nalezneme v liště prvků okna pro editaci rozložení. Každý funkční prvek grafického uživatelského patří do datových bloku, jehož vlastnosti ovlivňují i tyto prvky. Kromě datového bloku jsou to plátno, které se seskupuje prvky grafického uživatelského rozhraní, které jsou současně zobrazeny na jednom plátně. Obsah plátna je zobrazován přes okno, které patří do základní množiny struktur, které tvoří aplikaci. Tyto struktury umožňují tvorbu aplikací s jedním oknem, tzv: Single Document Interface a také aplikace s více okny s modálním či nemodálním chováním tzv: Multiple Document Interface.

6.2.1 Datový blok

Základní charakteristika datové bloku se týká vazby na databázový systém a to buď na tabulku nebo pohled. Podle této vlastnosti rozlišujeme mezi dvěma typy datových bloků a to mezi **databázovým** a **řídicí** datovým blokem.

Databázový datový blok

Databázový datový blok je spojen s datovým zdrojem (tabulka, pohled nebo uložená procedura), jehož data (řádky) zpřístupňuje aplikaci a automatizuje proces práce s těmito daty. Tyto data odpovídají řádkům datového zdroje a označují se jako záznamy (RECORDS). Tato automatice zahrnuje vkládání nových dat, změnu a mazání. Příkazy, které se používají pro tyto operace jsou CREATE_RECORD, DELETE_RECORD. Změna je provedna přiřazením nové hodnoty do položky, která zpřístupňuje data. Položka v databázovém datovém bloku může definovat vazbu se sloupcem datového zdroje. Data z toho sloupce jsou pak zobrazovány přes tyto položky. Potvrzení změn nad všemi záznamy lze provést příkazem COMMIT_FORM.

Řídicí datový blok

Tento blok není spojen s žádným datovým zdrojem. Používá se především pro tlačítka pro navigaci a řízení toku dat v aplikaci. Dále se používá pro položky, jejichž hodnoty se neukládají do databáze. V pokročilejších programovacích technikách se používají také pro data, která se do databáze ukládají ale tyto operace musí programátor zabezpečit sám příkazy SQL.

6.2.2 Položky

Mezi nejpoužívanější položky patří textové položky, které umožňují data zobrazit a modifikovat. Displejové položky slouží pouze pro zobrazení dat a uživateli není umožněno tyto hodnoty měnit. Každá položka má z rozsáhlé kolekce vlastností také vlastnost datového typu hodnoty, kterou lze položce přiřadit. Databázové položky lze vytvořit pouze v databázovém datovém bloku a to asociací se sloupce datového zdroje. Toto je nastaveno přes vlastnost položky dostupné přes okno vlastností.

Dalšími položkami jsou tlačítka. Tlačítka se používají k řízení aplikací a to přes trigery, zapsané v PL/SQL kódu. Vlastnostmi tlačítka lze nastavit jeho chování, text nebo obrázek zobrazený na tlačítku a celá řada dalších vlastností. Každá položka je asociována s plátnem, které se stará o její vykreslení. Pokud není plátno nastaveno, nikde se nezobrazuje a používá se jako globální sdílená proměnná. Tohoto principu se využívá také pro master-detail data. Takové vazby mezi databázovými datovými bloky, kde jeden je master a druhý detail lze jednoduše vytvořit s využitím průvodce pro vytvoření datového bloku.

6.2.3 Plátna a okna

Okna se starají o vykreslování obsahu jednotlivých pláten. Pro MDI aplikace se nej častěji používá přístup takový, že každé plátno je asociováno s unikátním oknem. Jiný přístup je sdílení jednoho okna a přepínání plátna do okna programově. Vazba mezi plátnem a oknem se nastavuje ve vlastnostech těchto objektů.

6.3 Realizace projektu v prostředí Form Builderu

Tento text nepokrývá kompletní popis vývojového prostředí, funkcí a aspektů tvorby aplikací. Pro další studium je potřebné využít nápovědy, kde jsou příklady použití jednotlivých trigerů. Dále referenční popis všech procedur a funkcí, vlastností jednotlivých objektů pro tvotbu aplikace. Další materiály jsou dostupné jako multimediální prezentace dostupné přes stránky kurzu IDS.