

Popis programovacího jazyka PL/SQL

Zdeněk Pavlas

Kapitola 1 - Přehled

V této kapitole jsou shrnuty hlavní vlastnosti PL/SQL, a je poukázáno na přednosti, které přinášejí. Jste seznámeni se základními koncepty, užitými v PL/SQL, a obecnou strukturou PL/SQL programů. Uvidíte, jak PL/SQL přemostňuje mezeru mezi SQL a procedurálními programovacími jazyky.

Bloková struktura

PL/SQL je jazyk s blokovou strukturou. To znamená, že základní jednotky (procedury, funkce, nepojmenované bloky), které tvoří program, jsou logickými bloky, které mohou obsahovat libovolný počet vnořených bloků. Deklarace jsou lokální vzhledem k bloku, a nejsou mimo něj dostupné.

```
[DECLARE
  -- deklarace]
BEGIN
  -- příkazy
[EXCEPTION
  -- zpracování výjimek]
END;
```

výpis 1 - Struktura bloku

PL/SQL blok má tři části: deklarační, část s příkazy, a část s obsluhou výjimek. Pouze část s příkazy je povinná. Pořadí částí je logické. Začíná se deklarační částí, ve které jsou nadeklarovány objekty. Poté je s nimi možno manipulovat v příkazové části. Výjimky, vzniklé během provádění příkazové části mohou být zpracovány v části zpracování výjimek.

Bloky je možno vnořovat v příkazové části, a v části zpracování výjimek, nikoliv v deklarační části. Lze v ní však definovat lokální podprogramy. Ty budou dostupné pouze v bloku, v kterém jsou definovány.

Proměnné a konstanty

V deklarační části bloku lze nadeklarovat konstanty a proměnné, a poté je používat v příkazech jako výrazy. Dopředné odkazy v deklarační části nejsou povoleny.

Deklarace proměnných

Proměnné mohou mít některý datový typ z SQL, jako je CHAR, DATE a NUMBER, nebo některý PL/SQL datový typ, jako je BOOLEAN, VARCHAR2 a BINARY_INTEGER.

Deklarace proměnné se skládá z názvu proměnné, následované jejím typem a středníkem. Je možno volitelně proměnnou inicializovat, a případně i specifikovat, že proměnná nesmí být NULL. Pokud není při deklaraci proměnné použita žádná inicializace, je proměnná zpočátku nastavena na NULL.

```
counter NUMBER(4);
in_stock BOOLEAN;
ratio REAL := 1.0;
ration REAL NOT NULL := 1.5;
```

výpis 2 - Příklady deklarace proměnných

Přiřazení hodnoty do proměnné

Do proměnné lze přiřadit hodnotu dvěma způsoby. Jednak lze použít přiřazovací příkaz ':=' , kdy je na levé straně proměnná, a napravo je výraz. Dále lze přiřadit hodnotu proměnné SQL příkazy SELECT, nebo FETCH.

Deklarace konstant

Deklarace konstant je stejná jako deklaráce proměnné, pouze je třeba doplnit klíčové slovo CONSTANT, a přímo v deklaraci do proměnné přiřadit hodnotu. Další přiřazení do konstanty nejsou povoleny.

```
credit_limit CONSTANT REAL := 5000.00;
max_count CONSTANT BINARY_INTEGER := 100;
```

výpis 3 - Příklady deklarace konstant

Kurzory

Existují dva druhy kurzorů, implicitní a explicitní. PL/SQL implicitně deklaruje kurzory pro všechny SQL příkazy, které pracují s databází, včetně dotazů, které vyberou pouze jeden řádek. Pro dotazy, které vrátí více řádků, lze explicitně nadeklarovat kurzor, a zpracovat všechny vrácené řádky individuálně.

```
CURSOR c1 IS SELECT * FROM emp WHERE deptno = 20;
```

výpis 4 - Příklad deklarace kurzoru

Množina řádků, vrácená dotazem na databázi, se nazývá 'result set'. Každý kurzor, odpovídající nějakému result setu, má také ukazatel na aktuální řádek. Tím je umožněna iterace result setu po jednotlivých řádcích.

Práce s kurzorem je podobná jako práce se klasickými soubory. Kurzor je třeba nejprve otevřít pomocí OPEN. Tím je proveden databázový dotaz s kurzorem spojený, a ukazatel na aktuální řádek je nastaven na první řádek. Příkaz FETCH přečte aktuální řádek do proměnné, a posune ukazatel kurzoru na další řádek. Po zpracování posledního řádku je třeba kurzor zrušit příkazem CLOSE.

Kurzorové FOR smyčky

V mnoha situacích je možno obejít užití explicitního kurzoru a příkazů OPEN, FETCH a CLOSE pomocí implicitního kurzoru smyčky FOR. Kurzorová smyčka FOR implicitně nadeklaruje indukční proměnnou jako záznam, reprezentující jeden řádek tabulky, otevře kurzor, opakovaně provede FETCH do indukční proměnné a provede tělo cyklu, a po zpracování všech řádků kurzoru skončí a kurzor uzavře. Test na poslední položku kurzoru lze provádět pomocí atributu %NOTFOUND, který bude true, pokud předchozí FETCH skončil neúspěšně.

```
DECLARE
  CURSOR c1 IS SELECT * FROM emp;
  i emp%ROWTYPE;
  total REAL := 0;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO i;
    EXIT WHEN c1%NOTFOUND;
    total := total + i.salary;
  END LOOP;
  CLOSE c1;
END;
```

```
DECLARE
  CURSOR c1 IS SELECT * FROM emp;
  total REAL := 0;
BEGIN
  FOR i IN c1 LOOP
    total := total + i.salary;
  END LOOP;
```

```

END;

DECLARE
  total REAL := 0;
BEGIN
  FOR i IN (SELECT * FROM emp) LOOP
    total := total + i.salary;
  END LOOP;
END;

```

výpis 5 - Příklady kurzorové FOR smyčky

Kurzorové proměnné

Kurzor je při své deklaraci pevně navázán na určitý dotaz. Kurzorová proměnná však může být navázána na libovolný typově kompatibilní dotaz. Navíc se chová jako proměnná, lze do ní přiřazovat nové hodnoty, nebo ji předat podprogramům.

```

DECLARE
  generic GenericCurTyp;
BEGIN
  OPEN generic FOR SELECT * FROM emp;
  -- generic je nyní kurzorovou proměnnou
END;

```

výpis 6 - Příklad deklarace a nastavení kurzorové proměnné

Parametrické kurzory

Kurzor lze nadefinovat s parametry, které budou určeny teprve při jeho otevření příkazem OPEN.

```

DECLARE
  emp_name emp.ename%TYPE;
  salary emp.sal%TYPE;
  CURSOR c1(name VARCHAR2, salary NUMBER) IS
    SELECT * FROM emp WHERE ename = name AND sal = salary;
BEGIN
  OPEN c1('ATTLEY', 1500);
  -- kurzor je otevřen s parametry ATTLEY a 1500
END;

```

výpis 21 - Použití parametrického kurzoru

Atributy

Jak proměnné tak kurzory mají speciální atributy, které zjednodušují opakované zápisy typů. Tabulky a sloupce tabulek mají podobné atributy.

Atribut %TYPE vrací datový typ proměnné, nebo sloupce tabulky. Pokud např. tabulka 'book' obsahuje sloupec 'title', lze použít při specifikaci typu proměnné syntax 'book.title%TYPE', a nadeklarovat tak proměnnou totožného typu, jako je sloupec 'title'. Podobně atribut %ROWTYPE vrací datový typ záznam, reprezentující jeden řádek tabulky, nebo kurzoru. Například 'book%ROWTYPE' je datový typ, reprezentující jeden řádek tabulky 'book'.

Řídící struktury

Protože je PL/SQL procedurální jazyk, obsahuje také řídicí struktury pro větvení, iteraci, skoky, a pro vytváření a zpracování výjimek.

IF-THEN-ELSE

Příkaz IF vyhodnotí podmínku, a pokud je splněna, provede klauzuli THEN. Pokud není podmínka splněna, provede se klauzule ELSE. Pokud větev ELSE tvoří pouze další příkaz IF, je možno dvojici ELSE-IF nahradit na ELSIF, a vypustit jeden koncový END IF. Vznikne tak konstrukce IF-THEN-ELSIF

```
IF balance > debit THEN
  -- balance > debit
END IF;
```

```
IF balance > debit THEN
  -- balance > debit
ELSE
  -- balance <= debit
END IF;
```

```
IF balance > debit THEN
  -- balance > debit
ELSIF balance = debit THEN
  -- balance = debit
END IF;
```

výpis 7 - Příklad IF-THEN, IF-THEN-ELSE, a IF-THEN-ELSIF příkazu

LOOP

Pomocí příkazu LOOP lze opakovaně provádět některé příkazy. Nejjednodušší je nekonečná smyčka, jejíž syntax je:

```
LOOP
  -- příkazy
END LOOP;
```

výpis 8 - Nekonečná LOOP smyčka

Dále existují FOR smyčky, a WHILE smyčky. Smyčka FOR specifikuje rozsah, a je provedena pro hodnoty z daného rozsahu. Pokud je za klíčovým slovem IN uvedeno REVERSE, proběhne iterace daného intervalu pozpátku. Smyčka WHILE vždy nejprve vyhodnotí podmínku, a dokud platí, provádí opakovaně tělo smyčky. Jakmile podmínka přestane platit, je tělo smyčky přeskočeno. Uvnitř smyčky lze použít příkaz EXIT WHEN, který specifikuje podmínku, která, pokud bude platit, způsobí ukončení provádění smyčky. Příkaz lze použít například pro předčasné ukončení smyčky.

```
FOR i IN [REVERSE] 1..count LOOP
  -- příkazy
END LOOP;
```

```
WHILE salary < 4000 LOOP
  -- příkazy
END DO;
```

```
LOOP
  -- příkazy
  total := total + salary;
  EXIT WHEN total > 2500;
  -- příkazy
END LOOP;
```

výpis 9 - Příklad smyček FOR, WHILE, a použití EXIT WHEN

GOTO

Příkaz GOTO způsobí nepodmíněný skok programu na daný label. Label se definuje jako identifikátor uzavřený mezi znaky <<>. Label musí předcházet některý příkaz nebo PL/SQL blok.

```
IF rating > 90 THEN
  GOTO calc;
END IF;
-- příkazy
<<calc>>
-- příkazy
```

výpis 10 - Příklad použití GOTO

PL/SQL Tabulky

PL/SQL Tabulka je něco jiného, než SQL tabulka. Jedná se o uspořádaný seznam elementů stejného typu. Každý element má unikátní index, který určuje jeho pozici v seznamu. Na rozdíl od klasického pole však PL/SQL tabulka není omezená, ale může dynamicky růst. Navíc, hodnoty indexů položek nemusí být spojitě, ale mohou to být libovolné celočíselné posloupnosti. PL/SQL tabulky lze naplnit například pomocí kurzorové smyčky FOR.

```
DECLARE
  TYPE TabTyp IS TABLE OF dept%ROWTYPE INDEX BY BINARY_INTEGER;
  a_tab TabTyp;
  n BINARY_INTEGER := 0;
BEGIN
  FOR i IN (SELECT * FROM dept) LOOP
    n := n + 1;
    a_tab(n) := i;
  END LOOP;
  -- a_tab je PL/SQL tabulka, naplněná řádky z 'SELECT * from dept'
END;
```

výpis 11 - Naplnění PL/SQL tabulky

```
DECLARE
  TYPE TabTyp IS TABLE OF dept%ROWTYPE INDEX BY BINARY_INTEGER;
  a_tab TabTyp;
  i BINARY_INTEGER := 0;
BEGIN
  FOR i IN a_tab.FIRST..a_tab.LAST LOOP
    INSERT INTO dept (dept_no, name)
      VALUES (a_tab(i).dept_no, a_tab(i).name);
  END LOOP;
END;
```

výpis 12 - Uložení PL/SQL tabulky

Pomocí syntaxe a_tab(i) se lze odkazovat na položku tabulky a_tab s indexem i. Krom toho existují i další operace, které se dají s PL/SQL tabulkou provádět.

```
a_tab(i).EXISTS      -- TRUE, pokud položka s indexem i existuje
a_tab.COUNT         -- vrátí počet položek tabulky
a_tab.FIRST         -- vrací nejmenší index prvku
```

```

a_tab.LAST          -- vrací největší index prvku
a_tab.PRIOR(i)     -- vrací index, který je hned před i
a_tab.NEXT(i)      -- vrací index, který je hned za i
a_tab.DELETE       -- vymaže všechny prvky tabulky
a_tab.DELETE(i)    -- vymaže prvek s indexem i
a_tab.DELETE(i, j) -- vymaže prvky s indexy od i do j

```

výpis 13 - Operace s PL/SQL tabulkou

Uživatelsky definovatelné záznamy

Pro deklaraci proměnné typu záznam lze použít syntax 'tabulka%ROWTYPE'. Lze však také nadefinovat vlastní datový typ typu záznam, pomocí klíčového slova TYPE a RECORD. Při definici záznamů lze používat také jiné, dříve definované záznamy. Proměnné typu záznam lze vzájemně přiřazovat operátorem ':='.

```

DECLARE
  TYPE Meeting IS RECORD (
    day DATE,
    place VARCHAR2(20),
    purpose VARCHAR(50)
  );

```

výpis 14 - Definice uživatelského záznamu

Definice podprogramů

PL/SQL rozlišuje podprogramy na procedury, a funkce. Procedura je definována pomocí klíčového slova PROCEDURE, funkce pomocí FUNCTION. Podobně jako PL/SQL blok je podprogram tvořen deklarační částí, příkazovou částí, a částí zpracování výjimek. Argumenty, které jsou posílány pouze do podprogramu jsou označeny jako IN, a ty, které mají být po skončení podprogramu kopírovány také směrem zpět jako OUT.

```

PROCEDURE award_bonus (emp_id IN NUMBER) IS
  bonus REAL;
  missing EXCEPTION;
BEGIN
  SELECT comm * 0.15 INTO bonus FROM emp WHERE empno = emp_id;
  IF bonus IS NULL THEN
    RAISE missing;
  ELSE
    UPDATE payroll SET pay = pay + bonus WHERE empno = emp_id;
  END IF;
EXCEPTION
  WHEN missing THEN
    -- zpracování výjimky missing
  END;
END award_bonus;

FUNCTION get_comm (emp_id IN NUMBER) RETURN REAL IS
  emp_comm REAL;
BEGIN
  SELECT comm INTO emp_comm FROM emp WHERE empno = emp_id;
  RETURN emp_comm;
END;

```

výpis 15 - Definice procedury a funkce

Zpracování chyb

PL/SQL umožňuje snadnou obsluhu chyb, pomocí výjimek. Když vznikne chyba, je možno vytvořit výjimku pomocí příkazu RAISE. Tím je ukončeno normální provádění příkazové části bloku, a začne se provádět část, která zpracovává výjimky. V této části je vyhledán blok, uvozený klíčovými slovy WHEN exc THEN, kde exc je jméno výjimky. Pokud je takový blok nalezen, je proveden. Pokud ne, bude se spoustupně prohledávat zpracování výjimek v nadřazených blocích, z nichž byl blok s výjimkou vyvolán. V bloku, který zpracovává výjimku, je možno také provést ne zpracování výjimky, ale pouze nějaké úklidové práce, a poté opět aktivovat poslední výjimku příkazem RAISE bez argumentu.

Runtime systém vytváří implicitně některé predefinované výjimky. Pokud je např. provedeno dělení nulou, je automaticky vytvořena výjimka ZERO_DIVIDE. Uživatelsky definované výjimky je třeba nadeklarovat pomocí datového typu EXCEPTION v deklarační části bloku, a vytvářet je explicitně pomocí příkazu RAISE.

Výhody PL/SQL

PL/SQL je plně portabilní, vysoce výkonný jazyk na provádění transakcí, který nabízí následující výhody.

- Podpora SQL
- Vyšší produktivita
- Vyšší výkon
- Portabilita
- Integrace s Oracle

SQL se stal standardním databázovým jazykem, protože je flexibilní, mocný, a snadno naučitelný. Několik příkazů jako je SELECT, INSERT, UPDATE a DELETE umožňuje snadnou manipulaci s daty v relační databázi.

PL/SQL rozšiřuje možnosti nástrojů jako je Oracle Forms, a Oracle Reports. PL/SQL blok lze například použít v triggerech v Oracle Forms, a zvýšit tak produktivitu práce.

Bez PL/SQL musí Oracle zpracovávat dotazy jeden po druhém. Každý SQL příkaz způsobuje nový dotaz na server, což zatěžuje síť. Při použití PL/SQL lze naráz poslat na server celý PL/SQL blok, a provést tak naráz několik SQL příkazů, a velmi tak snížit zatížení sítě.

Aplikace napsané v PL/SQL jsou portabilní na libovolný systém, na kterém běží Oracle, není je třeba upravovat pro nová prostředí. Lze psát portabilní PL/SQL knihovny, které jsou použitelné v různých prostředích.

Integrací s Oracle je míněna podpora atributů %TYPE a %ROWTYPE.

Kapitola 2 - Základy

Znaková sada

- Písmena A..Z, a..z
- Číslice 0..9
- Tabulátory, mezery, CR
- Symboly ()+*/<>=!:;.'@%,"#\$^&_[]{}?[]

výpis 16 - Znaková sada PL/SQL

Komentáře

Single-line komentáře začínají dvojitou pomlčkou, tedy znaky --, a jsou ukončeny koncem řádku. Multi-line komentáře začínají podobně jako v jazyku C znaky /*, a končí znaky */. Tyto komentáře nelze vnořovat.

Identifikátory

Identifikátory se používají pro označení PL/SQL podprogramů, konstant, proměnných, výjimek, kurzorů, a kurzorových proměnných. Identifikátor začíná písmenem, volitelně následovaným dalšími písmeny, číslicemi, nebo znaky '\$', '#', a '_'. Znaky jako pomlčky, lomítka nebo mezery nejsou povoleny. Nejsou také povoleny

identifikátory, které jsou klíčovými slovy. PL/SQL nerozlišuje u identifikátorů malá a velká písmena. Doporučuje se používat pro klíčová slova VELKÝCH písmen, pro identifikátory malých písmen.

Literály

Literál je explicitní hodnota nějakého typu. Může být numerický, znakový, řetězcový, nebo logický.

Numerický literál může být celočíselný (030 6 -14 0 +32767), nebo reálný (6.6667 0.0 -12.0 3.14159 +8300.00 .5 25. 2e5 1.0e-7). Hodnota 25.0 je považována za reálný literál, přestože je jeho hodnota celočíselná.

Znakové literály jsou jednotlivé znaky, uzavřené v apostrofech. ('Z' '%' '7' ' ' 'z' '('). Literály 'a' a 'A' jsou rozdílné. Znakové literály obsahující '0'..'9' lze používat v numerických výrazech, protože jsou implicitně převáděny na číselné hodnoty. Řetězcové literály jsou tvořeny posloupností 0 nebo více znaků, uzavřených v apostrofech ('Hello!' '\$1,000,000' '10-NOV-91'). Pokud je uvnitř řetězcového literálu znak apostrof, je třeba jej při zápisu zdvojit ('don"t leave'). Také v řetězcových literálech jsou rozlišována malá a velká písmena.

Logické literály tvoří předdefinované hodnoty TRUE, FALSE, a NULL, která je používána pro označení neznámé hodnoty.

Symboly

+	operátor sčítání
-	operátor odčítání nebo negace
*	operátor násobení
/	operátor dělení
**	operátor umocnění
=, <, >, <>, !=, ~, ^, <=, >=	relační operátory
(,)	oddělovače výrazu nebo seznamu
;	ukončení příkazu
%	atributový operátor
,	oddělovač položek
.	selektor komponenty
@	indikátor vzdáleného přístupu (?)
'	oddělovač znakového a řetězcového literálu
"	oddělovač speciálních identifikátorů
:	indikátor host variable
:=	přiřazovací operátor
=>	operátor asociace (?)
..	operátor rozsahu hodnot
	spojovací operátor
<<, >>	oddělovače návěští (label pro GOTO)
--	single-line komentář
/*, */	multi-line komentář

výpis 17 - Seznam symbolů PL/SQL

Datové typy

Každá konstanta nebo proměnná má svůj datový typ, které určuje formát, v kterém je hodnota uložena, a specifikuje také omezení této hodnoty. Skalární typ nemá žádné další interní komponenty. Kompozitní typ obsahuje interní komponenty, s kterými jde manipulovat individuálně. Referenční typ obsahuje hodnoty, nazývané ukazatele, které určují jiné objekty.

Předefinované datové typy

BINARY_INTEGER

Tento datový typ lze použít pro celá čísla se znaménkem (32 bitů). Stejně jako PLS_INTEGER, i BINARY_INTEGER vyžaduje méně místa než typ NUMBER. Většina operací s BINARY_INTEGER je pomalejší než s PLS_INTEGER.

Předdefinovány jsou také další odvozené typy, s názvy NATURAL, NATURALN, POSITIVE, POSITIVEN. Typy NATURAL* mohou obsahovat pouze nezáporná, typy POSITIVE* pak pouze kladná čísla. Pokud tento odvozený typ končí *N, je navíc definován jako NOT NULL, jeho hodnota tedy nesmí být NULL.

NUMBER

Datový typ NUMBER může obsahovat celá nebo reálná čísla s volitelnou přesností. Lze specifikovat přesnost, která představuje celkový počet cifer čísla, a dále měřítko, které určuje počet cifer, které budou od desetinné tečky směrem napravo. Syntaxe je NUMBER(přesnost), nebo NUMBER(přesnost, měřítko).

Přesnost i měřítko musejí být číselné literály. Maximální přesnost je 38, pokud není uvedena, je jako default použita maximální přesnost, podporovaná na daném systému. Měřítko může nabývat hodnot -84 až 127. Pokud není měřítko uvedeno, je jako default 0.

PL/SQL podporuje mnoho synonym pro typ NUMBER, a to: DEC, DECIMAL, DOUBLE PRECISION, INTEGER, INT, NUMERIC, REAL, SMALLINT. Mají stejný rozsah hodnot jako NUMBER, a jsou pro ANSI/ISO/IBM kompatibilitu.

Dalším podobným typem je FLOAT, u něj však nelze specifikovat měřítko, pouze přesnost v bitech. Maximální přesnost je 126 bitů.

PLS_INTEGER

Datový typ PLS_INTEGER obsahuje celá čísla se znaménkem (32 bitů), podobně jako BINARY_INTEGER. Při výpočtech je užívána strojová aritmetika, jsou proto rychlejší než NUMBER, nebo BINARY_INTEGER, které používají knihovnu.

Mezi PLS_INTEGER a BINARY_INTEGER je také rozdíl v tom, že PLS_INTEGER generuje vždy výjimku při přetečení, zatímco BINARY_INTEGER a NUMBER nikoliv. Tato sémantika typu BINARY_INTEGER je způsobena snahou o kompatibilitu.

CHAR

Typ CHAR může obsahovat řetězce pevné délky. Pokud jsou kratší, jsou automaticky rozšířeny mezerami na potřebnou délku. Lze specifikovat volitelný parametr délka, který určuje maximální délku řetězce. Ta může být maximálně 32767 byte. Syntaxe je CHAR(délka). Délka musí být číselný literál. Pokud není délka uvedena, je jako default hodnota 1.

Pokud je datový typ CHAR použit pro sloupec v některé tabulce, je délka omezena na maximálně 255. Synonymem pro typ CHAR je typ CHARACTER, opět z důvodu ANSI/ISO/IBM kompatibility.

LONG

Tento typ může obsahovat řetězce proměnné délky. Maximální délka je 32760 byte. Pokud je však typ LONG použit pro sloupec v tabulce, je maximální délka řetězce 2*31 byte. Typ je vhodný pro ukládání textů, nebo krátkých dokumentů. V některých SQL příkazech nelze používat sloupce typu LONG v některých výrazech.

RAW

Typ RAW slouží pro ukládání binárních dat. Je prakticky totožný s typem CHAR, ovšem Oracle při práci s ním řetězce nijak neinterpretuje, a neprovádí žádné konverze znakových sad.

LONG RAW

Typ LONG RAW slouží pro ukládání větších binárních dat. Je prakticky totožný s typem LONG, ovšem Oracle při práci s ním řetězce nijak neinterpretuje, a neprovádí žádné konverze znakových sad.

ROWID

Každá tabulka obsahuje pseudo-sloupec, nazývaný rowid. Tyto hodnoty jsou typu ROWID, a unikátně identifikují daný řádek tabulky. Jsou nejrychlejší metodou, jak zpřístupnit určitý řádek tabulky. Proměnnou typu ROWID lze naplnit přečtením pseudo-sloupce rowid z tabulky, a poté ji porovnávat s pseudo-sloupcem rowid ve WHERE klauzuli příkazů UPDATE, DELETE.

VARCHAR2

Typ VARCHAR2 může obsahovat řetězce proměnné délky. Vyžaduje povinný parametr délka, který určuje maximální délku řetězce, a to až do 32767 byte. Syntaxe je VARCHAR2(délka). Délka musí být numerický literál. Pokud je v tabulce sloupec typu VARCHAR2, je maximální délka pouze 2000 byte.

Typy VARCHAR2 a CHAR mají krom maximálních délek také některé další sémantické odlišnosti. K typu VARCHAR2 jsou definována synonyma STRING, a VARCHAR. Specifikace typu VARCHAR se však v budoucnu možná bude měnit, a proto je lepší jej nevyužívat.

BOOLEAN

Datový typ BOOLEAN může obsahovat hodnoty TRUE, FALSE, a ne-hodnotu NULL, která znamená chybějící informaci. Proměnnou typu BOOLEAN nelze přecíst ani zapsat do databázového sloupce.

DATE

Typ DATE nemá žádné parametry, může obsahovat data od 1-1-4712 BC do 12-31-4712 AD. Pokud je DATE v databázovém sloupci, obsahuje také čas s rozlišením na vteřiny. Default hodnota je první den aktuálního měsíce, default čas je půlnoc.

MLSLABEL

Tento datový typ používá Trusted Oracle pro definici sloupců, k nimž je regulovaný přístup. Ve standardním Oracle sloupec typu MLSLABEL obsahuje pouze hodnoty NULL.

Uživatelsky definované typy

V deklarační části PL/SQL bloku lze definovat tzv. podtypy. Nejedná se o nové typy, spíše o přejmenování a případně další omezení již existujícího typu. Subtyp je definován pomocí klíčového slova SUBTYPE.

```
SUBTYPE MY_CHAR IS CHAR(25);  
SUBTYPE COUNTER IS NATURAL;  
SUBTYPE EMP_REC IS emp%ROWTYPE;
```

výpis 18 - Příklady definic podtypů

Konverze datových typů

PL/SQL provádí implicitní konverze typů. Pokud je řetězcový typ použit pro numerickou operaci, je automaticky převeden na číslo. Pokud je sloupec tabulky jiného typu, než proměnná do/z něj ukládaná, je také provedena implicitní konverze. Krom toho lze provádět explicitní konverze za použití těchto knihovnických funkcí. Je vhodné je používat, a nespolehat na implicitní konverze, protože jsou kontextové a proto špatně předvídatelné.

```

my_char := TO_CHAR(my_date);
my_char := TO_CHAR(my_number);
my_char := RAWTOHEX(my_raw);
my_char := ROWIDTOCHAR(my_rowid);

```

```

my_date := TO_DATE(my_char);
my_date := TO_DATE(my_number);

```

```

my_number := TO_NUMBER(my_char);
my_raw := HEXTORAW(my_char);
my_rowid := CHARTOROWID(my_rowid);

```

výpis 19 - Explicitní konverze datových typů

Funkce TO_CHAR(my_date) může mít volitelný další parametr formát, který určuje způsob převodu. Například TO_CHAR(SYSDATE, 'SSSS') vrátí řetězec, který obsahuje počet vteřin od půlnoci.

Zpracování výrazů

Výrazy jsou tvořeny operátory a operandy. Operand je proměnná, konstanta, literál, nebo volání funkce. Operátory jsou vyhodnocovány v pořadí podle jejich priority.

1. x ** y -- x na y
 NOT x -- logická negace x

2. +x -- identita
 -x -- negace

3. x * y -- násobení
 x / y -- dělení

4. x + y -- sčítání
 x - y -- odčítání
 x || y -- spojení řetězců x a y

5. =, !=, <, >, <=, >=
 - relační operátory pro srovnání, obvyklá sémantika
 - x IS NULL -- test, zda je proměnná x NULL
 - x LIKE y -- test, zda x odpovídá patternu y.
 - '_' uvnitř aby představuje libovolný znak
 - '%' pak více libovolných znaků
 - x BETWEEN y AND z
 - stejné jako x >= y AND x <= z
 - x IN (y1, y2, .., yn)
 - stejné jako x = y1 OR x = y2 OR .. OR x = yn
 - lze použít ve WHERE klauzuli
 - složených SELECT příkazů

6. x AND y -- logický AND
 x OR y -- logický OR

výpis 20 - Seznam operátorů podle priority

Pokud je některý z operandů NULL, je výsledek operátoru také NULL. Výjimkou je spojování řetězců, kde je NULL chápán jako prázdný řetězec, a TRUE OR NULL je TRUE, a FALSE AND NULL je FALSE. NULL lze také použít jako prázdný příkaz v místech, kde syntaxe PL/SQL nějaký příkaz vyžaduje, např. v menu či v triggerech.