PL/SQL

Jazyk SQL je jazykem deklarativním, který neobsahuje procedurální příkazy jako jsou cykly, podmínky, procedury, funkce, atd.

Rozšířením jazyka SQL o proceduralitu od společnosti ORACLE je jazyk PL/SQL (Processing Language/Structured Query Language).

Jazyk PL/SQL umožňuje deklarovat konstanty, proměnné a kurzory, podporuje transakční zpracování, řeší chybové stavy pomocí výjimek. PL/SQL podporuje modularitu.

PL/SQL
praktické výhody

Šetření komunikačního kanálu

- Menší množství odesílaných povelů
  - V jednom povelu je větší množství příkazů
- Podstatně menší objem přenesených dat
  - Data se zpracují na serveru bez přenosu na klienta
- Odlehčení klienta
  - Možnost ukládat a vykonávat kód na serveru

Struktura jazyka PL/SQL

Typická struktura programového bloku se skládá ze tří částí:

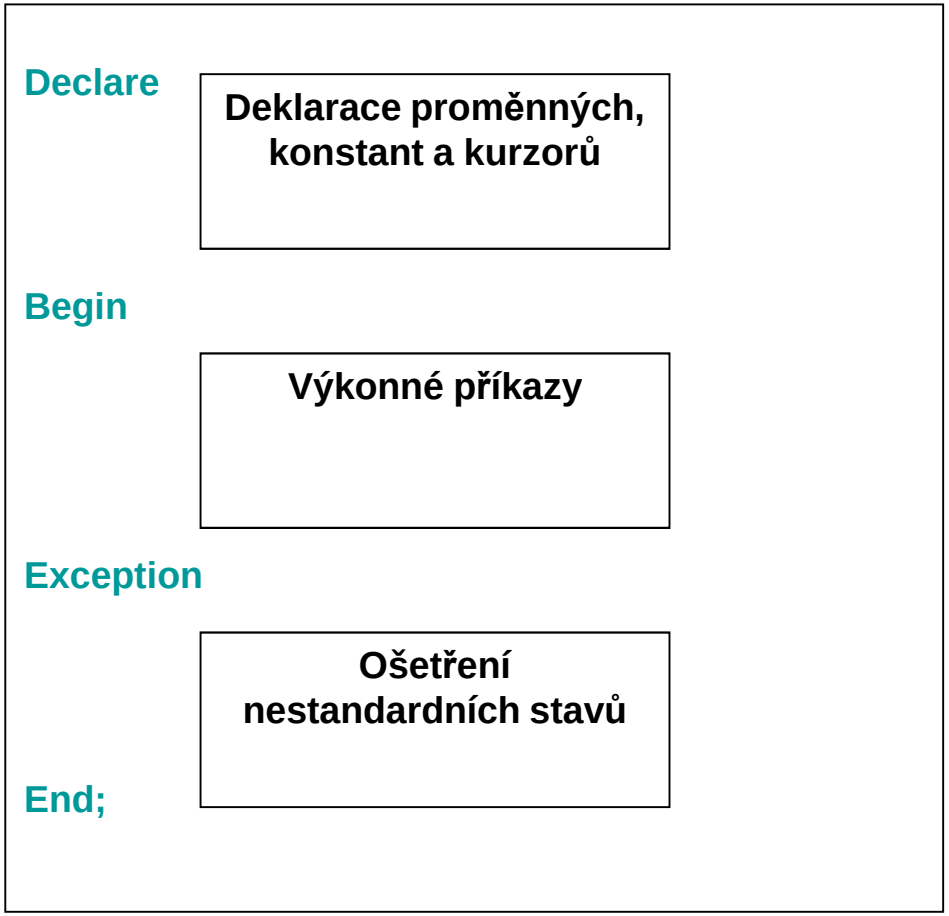
- deklarační část
- výkonná část
- část pro zpracování výjimek (ošetření chyb a nestandardních stavů)

Deklarační část obsahuje deklarace proměnných, konstant, kurzorů, atd. Výkonná část funkční logiku (algoritmus programového bloku), část výjimek řeší vzniklé chyby.
Povinná je pouze část výkonná.

Kurzor

- **Kurzor** je pracovní oblast obsahující data (výsledná množina, tzv. result set), které lze dále využívat prostřednictvím operací nad kurzory
- Existují implicitní a explicitní kurzory
- Implicitní jsou jednořádkové SQL (INTO)
- Explicitní můžeme deklarovat v části DECLARE pomocí klíčového slova CURSOR
- Práce s kurzory se podobá souborům

Struktura jazyka PL/SQL



Deklarační část

Deklarace proměnných a konstant

DECLARE
Jméno1 constant datový typ1:=hodnota1;
Jméno2 datový typ2;
Jméno3 datový typ3;
.
.
.
Jménon datový typn;

Příklad deklarace konstanty a proměnných
DECLARE
p_nasob constant number(3,2) :=1.15;
p_plat number(7,2);
p_bonus number(9,2);

Deklarační část

Deklarace kurzoru

```
DECLARE
CURSOR jméno1 IS
SELECT seznam FROM tabulka;
```

Příklad:

```
DECLARE
   CURSOR k1 IS
   SELECT jmeno  FROM zamestanec;
```

Výkonná část

```
BEGIN
Příkaz1;
Příkaz2;
.
.
.
Příkaz n;
END;
```

Výkonná část

Příklad:

```
BEGIN
   p_bonus := p_plat*p_nasob;
   .
   .
   .
END;
```

Příklad

```
DECLARE
p_jmeno                    char(15);        deklarace proměnné pro načítání jmen
    CURSOR k1 IS                            deklarace kurzoru na základě selectu
    SELECT upper(jmeno)  FROM zamestanec;

BEGIN                                       začátek výkonné části
OPEN k1;                                    otevření kurzoru
LOOP                                        začátek cyklu
    FETCH k1 INTO p_jmeno;          načtení řádku tabulky zamestnanec a uložení jmena do prom.
    dbms_output.put_line (p_jmeno);         výpis jména na konzolu
    EXIT WHEN k1% NOTFOUND;                 test na konec tabulky
END LOOP;                                   konec cyklu
CLOSE k1;                                   uzavření kurzoru
END;                                        konec výkonné části
```

Řízení průběhu programu

Syntaxe příkazu IF

IF podmínka THEN příkazy_1;
ELSIF podmínka THEN příkazy_2;
.
.
.
ELSE příkazy_n;
END IF;

IF - Příklad

```
DECLARE
        p_pohlavi                       char(1);
        p_jmeno                         char(15);
        p_stav                          char(1);
        CURSOR k2 IS
        SELECT jmeno, pohlavi, stav FROM osoby;
BEGIN
        OPEN k2;
        LOOP
                FETCH k2 INTO p_jmeno, p_pohlavi, p_stav;
                EXIT WHEN k2%NOTFOUND;
                IF p_pohlavi = ' M' THEN
                        INSERT INTO titultab (jmeno, pohlavi, stav, titul)
                        VALUES (p_jmeno,p_pohlavi,p_stav,'Pan');
                ELSEIF (p_pohlavi= ' Z' and p_stav= 'vdaná') then
                        INSERT INTO titultab (jmeno, pohlavi, stav, titul)
                        VALUES (p_jmeno,p_pohlavi,p_stav,'Paní');
                ELSE INSERT INTO titultab (jmeno, pohlavi, stav, titul)
                        VALUES (p_jmeno,p_pohlavi,p_stav,'Slečna');
                END IF;
        END LOOP;
COMMIT;
CLOSE k2;
END;
```

Řízení průběhu programu

Syntaxe příkazu CASE

```
CASE proměnná
   WHEN výraz_1 THEN příkazy_1;
   WHEN výraz_2 THEN příkazy_2;
   WHEN výraz_3 THEN příkazy_3;
   WHEN výraz_n THEN příkazy_n;
ELSE příkazy_n+1
END CASE
```

CASE - Příklad

```
SET SERVEROUT ON size 10000          přesměrování výstupu na konzolu
DECLARE
    znamka       constant          number(1):=1;
BEGIN
    CASE znamka
          WHEN 1 THEN dbms_output.put_line('Výborný');
          WHEN 2 THEN dbms_output.put_line('Chvalitebný');
          WHEN 3 THEN dbms_output.put_line('Dobrý');
          WHEN 4 THEN dbms_output.put_line('Dostatečný');
          WHEN 5 THEN dbms_output.put_line('Nedostatečný');
          ELSE dbms_output.put_line('Známka mimo stupnici');
    END CASE;
END;
```

ZÁKLADNÍ CYKLUS LOOP

```
LOOP
    příkaz_1;
    příkaz_2;
    příkaz_3;
    atd.
END LOOP;
```

Příklad:

```
LOOP
    pocet:= pocet +1
    IF pocet =100 THEN EXIT;
    END IF;
END LOOP;
```

CYKLUS FOR LOOP

```
FOR i IN start..konec
LOOP
      příkaz_1;
      příkaz_2;
      příkaz_3;
      atd.
END LOOP;
```

Příklad:

```
set serverout on size 10000
BEGIN
FOR i IN 1..5
LOOP
      if mod(i,2) = 0 then
      dbms_output.put_line (‘Cislo ‘ || i || ‘ je sude‘);
      else
      dbms_output.put_line (‘Cislo ‘ || i || ‘ je liche‘);
      end if;
END LOOP;
END;
```

CYKLUS WHILE LOOP

```
WHILE podmínka
LOOP
        příkaz_1;
        příkaz_2;
        příkaz_3;
        atd.
END LOOP;
```

Příklad:

```
DECLARE
        p_plat                  number(7,2);
        p_cissefa               number(4);
        start_cispra            constant        number(4):=7000;
        p_prijmeni              char(15);
BEGIN
        SELECT plat, cissefa, prijmeni INTO p_plat, p_cissefa, p_prijmeni
        FROM zamestnanec
        WHERE cispra=start_cispra;
        WHILE p_plat < 12000
LOOP
                SELECT plat, cissefa, prijmeni INTO p_plat, p_cissefa, p_prijmeni
                FROM zamestnanec
                WHERE cispra=p_cissefa;
END LOOP;
INSERT INTO  zamest VALUES (p_plat, p_prijmeni);
COMMIT;
END;
```

Tento příklad vyhledá zaměstnance, který je nejblíže nadřízený zaměstnanci 7000 a má plat nižší než 12000.

Přístupové proměnné

Přístupová proměnná je struktura, která obsahuje elementární části nazývané položky. Vybereme-li z tabulky, pohledu nebo pomocí kurzoru sloupce, ukládáme obsah těchto sloupců do proměnných, které byly všechny deklarovány v části DECLARE. Přístupová proměnná nahrazuje nutnost deklarace všech proměnných pro všechny položky z tabulek.

Definice přístupové proměnné

```
DECLARE
    jmeno_promenne        tabulka%ROWTYPE;
```

Příklad

Z tabulky zamestnanec vyberte všechny položky a uložte je do přístupové proměnné.

```
DECLARE
    zamest_zaznam                    zamestnanec%ROWTYPE
BEGIN
    SELECT * INTO zamest_zaznam FROM zamestnanec;
END;
```

KURZORY

- Deklarace kurzoru
- Otevření kurzoru
- Načtení záznamu do kurzoru
- Zavření kurzoru

Deklarace kurzoru

Tento krok přiřazuje kurzoru název a spojuje s ním příslušný příkaz SELECT. Deklarace kurzorů je součástí části DECLARE společně s deklaracemi proměnných.

Syntaxe:

CURSOR název_kurzoru IS příkaz_select;

Příklad

```
DECLARE
p_jmeno          char(15);
p_prijmeni       char(15);
p_datum          date;
CURSOR k1 IS SELECT * FROM zamestnanec;
```

Postup zpracování kurzoru

- Funguje stejně jako přístup k souborům
- Existují operace
  - OPEN cursor
  - FETCH cursor INTO record
  - CLOSE cursor
- Postup práce spočívá v otevření, postupném vyčtení jednotlivých záznamů (čtení s posunem) a uzavření kurzoru
- Pro zjištění stavu kurzoru existují atributy

Atributy (modifikátory) kurzoru

- Existují následující atributy za jménem kurzoru:
  - *cursor*%FOUND obsahuje záznamy?
  - *cursor*%NOTFOUND neobsahuje záznamy?
  - *cursor*%ISOPEN je otevřený?
  - *cursor*%ROWCOUNT dosud zpracováno ř.
- Existují také atributy pro definice typů:
  - *tab*%ROWTYPE záznam typu ř. tabul.
  - *tab.column*%TYPE typ záznamu

Otevření kurzoru

Tento krok provádí příkaz spojený s otevíráním kurzoru, který zakládá pracovní množinu n-tic (řádků), která může být dále naplněna příslušnými n-ticemi příkazem **FETCH**. Příkaz **OPEN** musí být umístěn v části výkonných příkazů (mezi **BEGIN** a **END**) nebo v části ošetření nestandardních stavů (**EXCEPTION**).

Syntaxe:

OPEN název_kurzoru

Příklad

OPEN k1;

Načtení záznamu do kurzoru

Načtení n-tic příslušného **SELECT**u do bloku PL/SQL se provádí příkazem **FETCH**. **Příkaz FETCH načte vždy jeden řádek příslušného SELECTu**. Z toho důvodu se příkaz **FETCH** vkládá do cyklu. Musí být zajištěno, aby vybraný seznam položek příkazem **SELECT** byl shodný se seznamem proměnných v příkaze **FETCH** (pořadí a odpovídající domény musí být shodné). **Příkaz FETCH je umístěn do části BEGIN nebo EXCEPTION.**

Načtení záznamu do kurzoru

Syntaxe:

```
BEGIN
OPEN název_kurzoru;
LOOP
FETCH název_kurzoru INTO seznam_proměnných
.
.
.
END LOOP;
END;
```

Načtení záznamu do kurzoru

Příklad

```
DECLARE
p_jmeno          char(15);
p_prijmeni       char(15);
p_datum          date;
CURSOR k1 IS SELECT * FROM zamestnanec;
BEGIN
    OPEN k1;
LOOP
    FETCH k1 INTO p_jmeno, p_prijmeni,p_datum;
    .
    .
    .
END LOOP;
CLOSE k1;
END;
```

Zavření kurzoru

Příkaz **CLOSE** uzavírá kurzor a nadále znepřístupňuje množinu dat vybranou příkazem **SELECT**. Příkaz **CLOSE** je součástí **BEGIN** nebo **EXCEPTION**. Kurzor rovněž uzavírá příkaz **EXIT** nebo **GOTO** (který vede výstup z cyklu).

Aktualizační operace s kurzorem

Přesunutá n-tice do kurzoru může být z databázové tabulky **vymazána**, resp. **aktualizována**. Pokud chceme využít této možnosti, je nutné, aby byl kurzor deklarován **FOR UPDATE OF** (položka pro aktualizaci) a v příkazu **FETCH** uvedena klauzule **WHERE CURRENT OF**.

Aktualizační operace s kurzorem

Příklad vymaže z databáze všechny záznamy, kde **datum je < 1.1.1930** a u všech záznamů, kde je **datum < 1.1.1940** změní hodnotu položky **plat na plat *1,2**

```
DECLARE
p_jmeno          char(15);
p_prijmeni       char(15);
p_datum          date;
CURSOR k1 IS SELECT * FROM zamestnanec WHERE datum < 1.1.1940
FOR UPDATE OF datum;
BEGIN
    OPEN k1;
LOOP
    FETCH k1 INTO p_jmeno, p_prijmeni,p_datum;
    IF p_datum < 1.1.1930 THEN DELETE zamestnanec WHERE CURRENT OF k1;
    ELSE UPDATE zamestnanec SET plat = plat *1,2 WHERE CURRENT OF k1;
    END IF;
END LOOP;
CLOSE k1;
END;
```

Atributy explicitních kurzorů

Atribut **%NOTFOUND** nabývá hodnoty **TRUE**, pokud právě provedený příkaz **FETCH** **nenalezl** další n-tici odpovědi. Opakem je atribut **%FOUND**, který v tomto případě nabývá hodnoty FALSE.

Příklad

```
OPEN k1;
LOOP
   FETCH k1 INTO x,y,z;
   EXIT WHEN k1%NOTFOUND;
END LOOP;
CLOSE k1;
```

Atributy explicitních kurzorů

Atribut %ROWCOUNT vrací počet řádků dosud načtených příkazem FETCH příslušným SELECTem.

Příklad

```
LOOP
    FETCH k1 INTO x,y,z;
    IF k1%ROWCOUNT < 15 THEN
    INSERT INTO jméno_tabulky VALUES (….);
    ELSE
    EXIT;
    END IF;
END LOOP;
```

Tento příklad opouští cyklus po načtení prvních 14 řádků tabulky.

Chyby a nestandardní stavy

Příkaz **RAISE** slouží k předání řízení do části **EXCEPTION** bloku PL/SQL. Nestandardní stav je třeba nejdříve v části DECLARE deklarovat.

```
DECLARE
   objednavka          EXCEPTION;
```

Jakmile je nestandardní stav definován, můžeme pomocí příkazu **RAISE** v části **BEGIN** tento stav vyvolat. V tomto okamžiku přejde řízení do části **EXCEPTION**.

Chyby a nestandardní stavy

```
BEGIN
   IF počet_na_sklade < limit THEN RAISE
   objednavka;
EXEPTION
   WHEN objednavka THEN
   INSERT INTO tab_obj ….
```

Zadání příkladu

Napište část programu, který přepíše načtené řádky do tabulky nove_oddeleni s výjimkou řádků s hodnotou cisodd=33. Nastavte EXCEPTION pro cisodd=33. Není-li cisodd=33, zapište cisodd, jmeno a misto do tabulky nove_oddeleni. Je-li cisodd=33, zapište vhodnou zprávu do tabulky zpravy.

Řešení

```
DECLARE
    p_cisodd        number;
    p_jmeno         varchar2(10);
    p_misto         varchar2(15);
    oddel_33        EXCEPTION;
BEGIN
IF p_cisodd = 33 THEN RAISE oddel_33;
END IF;
INSERT INTO nove_oddeleni (cisodd,,jmeno,misto) VALUES(p_cisodd,
    p_jmeno, p_misto);
    COMMIT;
EXCEPTION
    WHEN oddel_33 THEN
    INSERT INTO zpravy (text) VALUES (‘Pokus vytvorit oddeleni 33’);
    COMMIT;
END;
```

Procedury a funkce

Bloky příkazů jazyka PL/SQL lze pojmenovat a uložit ve spustitelné formě do databáze. Těmto blokům říkáme procedury, resp. funkce.

Vlastnosti procedur a funkcí:

- Jsou uloženy ve zkompilovaném tvaru v databázi.
- Mohou volat další procedury či funkce, či samy sebe.
- Lze je volat ze všech prostředí klienta.

Funkce, na rozdíl od procedury, vrací jedinou hodnotu (procedura může vracet hodnot více, resp. žádnou).

Procedury

CREATE PROCEDURE jméno_procedury
[(formální_parametry)] **AS**
[lokální deklarace]
BEGIN
[výkonné příkazy]
[**EXCEPTION**
ošetření nestandardních stavů]
END;

Příklad

```
Create procedure pln_cislo as
p_cislo number;
begin
for i in 1..20 loop
insert into Nic(cislo) values (i);
commit;
end loop;
end;
```

lokální deklarace bez kl. slova DECLARE

tabulka nic musí být předem vytvořena

Tato procedura zapíše do tabulky Nic čísla od 1 do 20.

Příklad

```
procedure prevod  as
p_jmeno varchar2(15);                    lokální deklarace bez kl. slova DECLARE
cursor k1 is                             deklarace kurzoru (tabulka Pacient musí existovat)
select upper(jmeno_p) from Pacient;
begin
open k1;
loop
fetch k1 into p_jmeno;                   načtení jmeno_p do kurzoru a uložení do p_jmeno
dbms_output.put_line(p_jmeno);           výstup na konzolu
exit when k1%notfound;
end loop;
close k1;
end;
```

Tato procedura vytiskne na konzolu jména všech pacientů z tabulky Pacient a převede všechna písmena na velká

Procedury

Formální parametry procedury

Jméno_parametru [IN OUT IN OUT] typ_paramertu [:= hodnota]

Specifikace jednotlivých parametrů v seznamu jsou odděleny čárkou.

Parametry mohou být vstupní, výstupní a vstupně-výstupní.

Pouze vstupní parametry mohou být inicializovány. K inicializaci můžeme použít buď klauzuli DEFAULT nebo přiřadit hodnotu (:=)

Příklad s parametry

```
create procedure deleni
(delenec IN number,delitel IN number)
as
begin
dbms_output.put_line(delenec/delitel);
end;
```

parametry

Procedura vytiskne na konzolu podíl hodnot zadaných jako skutečné parametry.

Kompilace a spuštění procedury

Zápis ukončíme znakem **.** (**tečka**) na novém řádku

Příkazem **RUN** přeložíme proceduru

Příkazem **EXECUTE[(seznam skutečných parametrů)]** proceduru vykonáme

Pro předchozí příklad nezapomeneme zadat příkaz **SET SERVEROUT ON** pro přesměrování výstupu na konzolu

Zápis, kompilace a spuštění v SQL*Plus

```
create procedure deleni
(delenec IN number,delitel IN number) as
begin
dbms_output.put_line(delenec/delitel);
end;
.
RUN
SET SERVEROUT ON
EXECUTE deleni(10,2);
```

Ošetření chyby při dělení nulou

```
create procedure deleni
(delenec IN number,delitel IN number) as
begin
dbms_output.put_line(delenec/delitel);
Exception
When zero_divide then
dbms_output.put_line('Chyba při dělení nulou');
end;
```

Funkce

CREATE FUNCTION jméno_funkce
[(formální_parametry)] **RETURN**
typ_návratové_proměnné **AS** [lokální
deklarace]

BEGIN

[výkonné příkazy]

[**EXCEPTION**

ošetření nestandardních stavů]

END;

Příklad funkce

```
create function f_deleni
(delenec IN number,delitel IN number) return number
As
Vysledek number;
begin
vysledek := delenec/delitel;
Return vysledek;
Exception
When zero_divide then
dbms_output.put_line('Chyba při dělení nulou');
end;
```

Vyvolání funkce

```
SET SERVEROUT ON
begin
dbms_output.put_line (f_deleni(12,4));
end;
```

Zrušení procedury či funkce

DROP PROCEDURE jméno_procedury

DROP FUNCTION jméno_funkce

Databázové triggery

Databázový trigger je uživatelsky definovaný blok PL/SQL sdružený s určitou tabulkou. Je implicitně spuštěn (proveden), jestliže je nad tabulkou prováděn aktualizační příkaz.

Databázový trigger má čtyři části:

- typ triggeru (**BEFORE** / **AFTER**)
- spouštěcí událost (**INSERT**/ **UPDATE**/ **DELETE**)
- omezení triggeru (nepovinná klauzule **WHEN**)
- akce triggeru (blok PL/SQL)

Databázové triggery

Na každou tabulku lze vytvořit až 12 různých databázových triggerů:

- INSERT / UPDATE / DELETE
- BEFORE / AFTER
- STATEMENT / ROW (příkazový/řádkový)

Příkazový trigger se spustí jedenkrát pro příkaz, bez ohledu na počet aktualizovaných řádků.
Řádkový trigger se spustí pro každý aktualizovaný řádek tabulky.

Vytvoření databázového triggeru

CREATE [OR REPLACE] **TRIGGER** jméno typ_triggeru spouštěcí_akce [**OF** sloupec, sloupec, …] **ON** tabulka [**FOR EACH ROW**] [**WHEN** podmínka]

BEGIN

.

.

.

END;

Příklad

```
Create trigger x_kontrola before insert or update
   on Vypujcka for each row
begin
   .
.
.
end;
```

Trigger před vložením nebo aktualizací dat do tabulky Vypujcka pro každý řádek provede kontrolu, uvedenou ve výkonné části

Prefixy v databázových triggerech

Pro odkazy na staré a nové hodnoty sloupců v řádkových triggerech se používají prefixy **:OLD** a **:NEW**.

Příklad

IF :NEW.plat < :OLD.plat THEN …

Poznámky:

- hodnoty **:NEW** a **:OLD** jsou použitelné pouze v řádkových triggerech
- obě hodnoty jsou použitelné v příkazu **UPDATE**
- **:OLD** u příkazu **INSERT** je **NULL**
- **:NEW** u příkazu **DELETE** je **NULL**
- v klauzulích **WHEN** se vynechává středník

Příklad

```
Create trigger x_kontrola before insert or update
   on vypujcka for each row
begin
  if :new.dat_vra < :old.dat_vyp
 then raise_application_error(-20500,'chybné
   datumy! ');
 end if;
end;
```

Trigger dělá kontroly datumů výpůjčky a vrácení. V případě chyby vypíše „chybné datumy!“

Příklad

Mějme schéma databázové tabulky vytvořené následujícím příkazem create:

```
CREATE TABLE pece_cenik (
      pece_cenik_id        SMALLINT NOT NULL,
      nazev_cenik          CHAR(18) NULL,
      priplatek            DECIMAL(7,2) NULL,
      popis_cenik          CHAR(18) NULL);
ALTER TABLE Pece_cenik
ADD  ( PRIMARY KEY (pece_cenik_id) ) ;
```

Dále mějme tabulku dodatecna_pece, která je tabulkou podřízenou, obsahuje FK, který je v tabulce pece_cenik PK.

Trigger pro zajištění referenční integrity při mazání z nadřazené tabulky

```
create trigger tD_Pece_cenik after DELETE on pece_cenik for each row
declare numrows INTEGER;
begin
select count(*) into numrows
      from dodatecna_pece
      where
            dodatecna_pece.pece_cenik_id = :old.pece_cenik_id;
    if (numrows > 0)
    then
      raise_application_error(-20001,
        'Cannot DELETE pece_cenik because dodatecna_pece exists.'
      );
    end if;
end;
/
```

Trigger pro zajištění referenční integrity při aktualizace nadřazené tabulky

```
create trigger tU_Pece_cenik after UPDATE on pece_cenik for each row
declare numrows INTEGER;
begin
if
    :old.pece_cenik_id <> :new.pece_cenik_id
  then
    select count(*) into numrows
      from dodatecna_pece
      where
        dodatecna_pece.pece_cenik_id = :old.pece_cenik_id;
    if (numrows > 0)
    then
      raise_application_error(-20005,
        'Cannot UPDATE pece_cenik because dodatecna_pece exists.'
      );
    end if;
  end if;
end;
/
```

Postup při spouštění databázového triggeru

- do ORACLE je předán příkaz INSERT, UPDATE nebo DELETE
- provede se příkazový trigger BEFORE
- pro každý řádek, kterého se příkaz SQL týká:
  - se provede řádkový trigger BEFORE
  - změní se řádek a provedou se kontroly integritního omezení
  - se provede řádkový trigger AFTER
- dokončí se odložené kontroly IO s ohledem na přechodná porušení
- provede se příkazový trigger AFTER
- návrat do aplikace

Aktivace a deaktivace triggerů

Po nadefinování triggerů jsou tyty implicitně aktivní. Je-li třeba trigger deaktivovat, resp. zpět aktivovat, lze použít příkazu:

ALTER TRIGGER jméno_triggeru ENABLE | DISABLE;

Resp. lze deaktivovat či aktivovat všechny triggery, definované nad konkrétní tabulkou:

ALTER TABLE jméno_tabulky ENABLE | DISABLE ALL TRIGGERS;