# Systemy Baz Danych

*Pl/SQL - język proceduralny serwera ORACLE – cz. 2*

# Materiał wykładu

Wykład ten jest kontynuacją wykładu poprzedniego. W dalszym ciągu omawiamy Pl/SQL - język proceduralny serwera bazy danych ORACLE. Po zapoznaniu się z podstawami składni oraz z instrukcjami sterującymi, przechodzimy obecnie do omówienia procedur, funkcji, pakietów i wyzwalaczy.

Wykład jest przeznaczony dla studentów przedmiotu:

**Systemy Baz Danych** – studia inżynierskie na Wydziale Informatyki PJWSTK,

oraz jako materiał uzupełniający dla słuchaczy przedmiotu

**Komunikacja z Bazami Danych** – studia podyplomowe na Wydziale Informatyki PJWSTK

# Procedury, funkcje, pakiety.

Procedury i funkcje to obiekty bazy danych, zapisywane w bazie danych. Mogą być wykorzystywane przez wszystkie procesy korzystające z bazy danych, pod warunkiem, że proces został uruchomiony przez użytkownika lub aplikację posiadającą uprawnienia do danego obiektu – tak jak w przypadku innych obiektów bazy danych.

Procedury i funkcje mogą być definiowane wewnątrz bloków Pl/SQL. Wówczas ich użycie jest ograniczone do tego bloku, w którym zostały zdefiniowane.

Procedury i funkcje mogą być obiektami niezależnymi, ale mogą także być umieszczane w większych strukturach zwanych pakietami.

Zasadniczą rolą tego typu obiektów jest utworzenie współdzielonego kodu, umieszczonego na serwerze. Mogą one zostać wykorzystane w zasadzie do wszystkich operacji wykonywanych na serwerze, eliminując konieczność sięgania bezpośrednio do tabel. Z uwagi na dużą elastyczność (możliwości programistyczne) użycie procedur może zastąpić widoki, a w dużej mierze także wyzwalacze.

W procedurach PL/SQL, podobnie jak w blokach anonimowych, niedopuszczalne jest uzycie poleceń DDL.

# Deklaracja procedury

```
CREATE [OR REPLACE] PROCEDURE nazwa_procedury
    (lista_parametrów_formalnych)
{AS | IS}
    blok Pl/SQL bez słowa kluczowego DECLARE
```

Użycie w składni deklaracji słów **OR REPLACE** oznacza, że jeśli istnieje już procedura o danej nazwie, zostanie ona zastąpiona przez nowy obiekt, ale nie zostanie podniesiony błąd (jak ma to miejsce w starszych wersjach T-SQL). Zapis ten jest bardzo wygodny przy testowaniu i wprowadzaniu poprawek do nowo tworzonych procedur, gdzie wielokrotnie zmieniamy treść kodu.

Słowa **AS IS** są wymienne (nie ma znaczenia, które zostanie użyte).

Lista parametrów (formalnych) procedury, to ujęte w nawiasy i oddzielone przecinkami deklaracje:

```
nazwa_parametru [typ] typ_danych [DEFAULT =
                                    wartość_domyślna]
(,...)
```

# Parametry procedury

Parametry procedury w Pl/SQL mogą być typu **IN** (domyślny), **OUT**, **IN OUT** .

Typ **IN** (domyślny) – parametry tego typu służą do odebrania z procesu wywołującego procedurę wartości i przekazania jej do procedury. Wartość przekazana przez parametr **IN** nie ulega (nie może ulec) zmianie w procedurze lub funkcji. Parametr **IN** nie może wystąpić po lewej stronie instrukcji podstawienia.

Typ **OUT** –wartość wyliczona w procedurze jest przekazywana poprzez parametr **OUT** do procesu, który procedurę uruchomił. Warunkiem przekazania wartości jest zwykłe (bez błędu) zakończenie procedury.

Typ **IN OUT**– poprzez parametr tego typu wartość może zostać przekazana do procedury z wywołującego ją procesu, a po poprawnym zakończeniu procedury może zostać użyty do przekazania wyliczonej w procedurze wartości do wywołującego ją procesu.

**W specyfikacji typu danych parametrów nie podaje się ich rozmiarów(!).** Podawany jest tylko typ np. `NUMBER`, `VARCHAR2`, albo odwołanie do typu kolumny w tabeli np. `emp.sal%TYPE`.

# Procedury

**Przykład**:

```
CREATE OR REPLACE PROCEDURE UpdSal
   (v_up NUMBER, v_deptno NUMBER)
/* Procedura w dziale o numerze podanym
przez v_deptno podnosi place o procent zadany przez v_up */
IS
BEGIN
  UPDATE emp
   SET sal = Sal * (1 + v_up/100)
   WHERE deptno = v_deptno;
END;
```

Wywołanie procedury:

```
CALL UpdSal (12, 20);
```
w Sqldeveloper

```
UpdSal (12, 20);
```
w kodzie Pl/SQL

```
EXECUTE UpdSal (12, 20);
```
w SQL*Plus

# Procedury

Zmodyfikujemy teraz poprzednią procedurę tak, żeby zwracała nowy (po podwyżce) budżet płac działu, poprzez parametr OUT.

**Przykład:**

```
CREATE OR REPLACE PROCEDURE UpdSal
 (v_up NUMBER, v_deptno NUMBER, v_Budzet OUT NUMBER)
/*Procedura podnosi place w dziale podanym przez v_deptno o
procent zadany przez v_up i zwraca budzet plac dzialu po
podwyżce*/
IS
BEGIN
  UPDATE emp
  SET    sal = Sal * (1 + v_up/100)
  WHERE  deptno = v_deptno;
  SELECT Sum(sal) INTO v_Budzet
  FROM   emp
  WHERE  deptno = v_deptno;
END;
```

# Procedury

… i wywołanie procedury z poprzedniego bloku, tym razem w bloku Pl/SQL:

```
SET Serveroutput ON;
DECLARE
   v_deptbudzet NUMBER(10);
   v_Nrdzialu NUMBER;
   V_proc NUMBER;
BEGIN
   v_Nrdzialu:= 30;
   V_proc := 5;
   UpdSal (V_proc, v_Nrdzialu, v_deptbudzet);
   dbms_output.put_line('Aktualny budzet dzialu ' || v_Nrdzialu
                        || ' ' || ' wynosi ' || v_deptbudzet);
END;
```

Proszę zwrócić uwagę na inną składnię wywołania procedury w bloku Pl/SQL (odwołanie do samej nazwy) i bezpośrednio w Sqldeveloper (poprzez instrukcję **CALL** pochodzącą ze zbioru instrukcji SQL*Plus).

# Wartości domyślne parametrów formalnych

Parametry zarówno funkcji jak i procedur mogą mieć nadawane wartości domyślne. Te wartości mogą być pomijane przy wywoływaniu procedur / funkcji, natomiast ich wartości podane w trakcie wywołania zastępują wartości domyślne. Parametry z wartościami domyślnymi umieszczane są na końcu listy parametrów.

```
CREATE OR REPLACE PROCEDURE nazwa_procedury
   (nazwa_parametru typ_danych DEFAULT wyrażenie (, ….))
IS
   (…);
```

**Przykład:**

```
CREATE OR REPLACE PROCEDURE XXX
  (v_deptno emp.deptno%TYPE DEFAULT 10,
  V_job emp.job%TYPE DEFAULT 'SALESMAN')
AS
BEGIN
  dbms_Output.put_line(v_deptno || ' ' || V_job);
END;
```

# Wartości domyślne parametrów formalnych

Jeżeli został zadeklarowany więcej niż jeden parametr formalny z wartościami domyślnymi, przy wywołaniu procedury wartości nadawane parametrom z wartościami domyślnymi wskazuje się explicite:

```
CALL nazwa_procedury (…, nazwa_parametru => wyrażenie);
```

Dla przykładowej procedury z poprzedniego slajdu (wartości domyślne deptno = 10, job = SALESMAN) wywołanie mogło by wyglądać następująco:

```
CALL XXX(V_job => 'MANAGER');
```

10 MANAGER

```
CALL XXX(v_deptno => 20);
```

20 SALESMAN

```
CALL XXX(v_deptno => 20, V_job => 'MANAGER');
```

20 MANAGER

# Funkcje

Składnia deklaracji funkcji jest zbliżona do deklaracji procedury:

```
CREATE [OR REPLACE] FUNCTION nazwa_funkcji
    (lista_parametrów_formalnych)
RETURN typ_danych
{AS | IS}
    blok Pl/SQL bez słowa kluczowego DECLARE
    z instrukcją RETURN wyrażenie;
```

Różnica pomiędzy funkcją a procedurą polega na sposobie zwracania wyliczonych wartości. Procedura może, ale nie musi zwracać wartości będących wynikiem wykonanych wewnątrz procedury obliczeń. Wartości zwracane przez procedurę przekazywane są „na zewnątrz" poprzez parametry **OUT**.

Funkcja zwraca wartość wyliczoną po słowie kluczowym **RETURN**. Po wyliczeniu tej wartości funkcja kończy działanie i przekazuje „na zewnątrz" wyliczoną wartość pod swoją nazwą.

# Funkcje

**Przykład**:

*Tym razem budżet działu o numerze podanym w parametrze odczytamy przez funkcję, odwołując się do jej nazwy.*

```
CREATE OR REPLACE FUNCTION YYY (v_deptno Int)
RETURN NUMBER
IS
V_Budzet Number;
BEGIN
   SELECT  Sum(sal)
   INTO    V_Budzet
   FROM    emp
   WHERE   deptno = v_deptno;
   RETURN  V_Budzet;
END;
```

# Funkcje

… i użycie funkcji w bloku Pl/SQl:

```
SET Serveroutput ON
DECLARE
  v_deptBudzet NUMBER(10,2);
BEGIN
  v_deptBudzet := YYY(10);
  dbms_output.put_line('Budzet departamentu 10 wynosi '
  || v_deptBudzet);
END;
```

… w SQL*Plus:

```
VARIABLE v_budzet NUMBER
EXECUTE  :v_budzet := YYY(10);
PRINT    v_budzet;
```

… w „czystym” SQL:

```
SELECT YYY(10) FROM dummy;
```

# Funkcje – użycie w poleceniach SQL

Jak widać z ostatniego przykładu na poprzednim slajdzie, funkcje definiowane w bazie danych mogą być używane w poleceniach SQL tak jak inne funkcje implementowane przez SZBD. Ograniczenie w ich wykorzystaniu wewnątrz poleceń SQL stanowi:

- zakaz użycia w ich treści instrukcji DDL i DML (funkcje nie mogą zmieniać stanu bazy danych),
- nie mogą posiadać parametrów wyjściowych **OUT**,
- nie powinny korzystać ze zmiennych nie – lokalnych w pakietach,
- wszystkie parametry muszą zostać wyspecyfikowane,
- nie wolno używać w nich notacji `parametr => wartość.`

# SQL*Plus vs Sqldeveloper

Zastąpienie narzędzia SQL*Plus uruchamianego w Command Line wyeliminowało konieczność użycia wielu instrukcji używanych w tym środowisku, takich jak:

**DESC[CRIBE]** nazwa_tabeli – wpisz podstawowe informacje o tabeli

**DESC[CRIBE]** nazwa_procedury (lub funkcji) – wpisz informacje o parametrach procedury (funkcji),

```
SELECT Line, Text
FROM User_Source
WHERE Name = 'NAZWA_PROCEDURY';
```
- wypisz skrypt procedury,

**Show Errors** – wypisz błędy nieudanej kompilacji procedury.

Wszystkie powyższe instrukcje zostały zastąpione operacjami wykonywanymi przy pomocy graficznego interfejsu. Jednak Sqldeveloper prawidłowo wykonuje większość z instrukcji Pl*SQl, w tym wszystkie wymienione powyżej.

# Przeciążenie nazw procedur i funkcji

Pl/SQL dopuszcza wielokrotne użycie tej samej nazwy funkcji czy procedury. Jest to wygodne, gdy ta sama (z logicznego punktu widzenia) procedura jest wykorzystywana różnie, zależnie od kontekstu, np. z różną liczbą parametrów lub różnymi typami danych parametrów.

Wersje tej samej procedury (funkcji) muszą różnić się liczbą parametrów, albo nazwą i typem parametrów. Jest to warunek konieczny dla rozróżnienia przez system tych wariantów i wybrania właściwej wersji.

Dopuszczalne jest zastosowanie takiego rozwiązania wyłącznie w procedurach utworzonych w ramach jednego bloku lub jednego pakietu, zatem niedopuszczalne jest przeciążanie nazw procedur niezależnych (standelone). Niedopuszczalne jest również zróżnicowanie tylko podtypów danych (np. Char i Varchar2, Int i Real itd.).

Na kolejnym slajdzie znajduje się przykład użycia tak zdefiniowanych procedur w bloku anonimowym, wykorzystanych do odczytania budżetu płac działu, jeśli podawana jest jego nazwa (dname) albo numer (deptno).

# Przeciążenie nazw procedur i funkcji

```
DECLARE
v_budzet NUMBER(8,2); v_num INT; v_name Varchar2(30);
PROCEDURE emp_dept (v_bud OUT NUMBER, v_deptno INT) IS
BEGIN
  SELECT SUM(sal) INTO v_bud FROM emp WHERE deptno = v_deptno;
END;
PROCEDURE emp_dept (v_bud OUT NUMBER, v_dname Varchar2) IS
BEGIN
  SELECT SUM(sal) INTO v_bud FROM emp WHERE deptno =
              (SELECT deptno FROM dept WHERE dname = v_dname);
END;
BEGIN
  v_num := 10;
  --v_name := 'SALES';
  emp_dept(v_budzet, v_name);
  emp_dept(v_budzet, v_num);
  dbms_output.put_line(v_budzet);
END;
```

# Procedury i funkcje w Sqldeveloper

Informacje na temat procedur i funcji dostępne są w Sqldeveloper w oknie **Connections** po rozwinięciu gałęzi **Procedures** lub **Functions** drzewa **Connection**. Spod prawego klawisza (menu kontekstowe) po wybraniu nazwy procedury lub funkcji w drzewie dostępne są operacje: **Edit**, **Debug**, **Compile**, **Compile for Debug**.

# Pakiety

Z uwagi na dużą liczbę procedur, funkcji, sekwencji, jakie zwykle powstają podczas tworzenia aplikacji, dużym ułatwieniem jest możliwość grupowania ich w większe jednostki zwane pakietami (packages). W ramach pakietu możemy zdefiniować:

- kursory
- zmienne i stałe
- wyjątki
- podprogramy (funkcje i procedury)

Pakiet zazwyczaj składa się z dwóch części – specyfikacji pakietu (specification) i ciała pakietu (body). Specyfikacja (część publiczna pakietu) stanowi interfejs dla aplikacji i zawiera deklaracje typów, zmiennych, stałych, wyjątków, kursorów i podprogramów możliwych do użycia przez aplikację korzystającą z pakietu.

Ciało pakietu (część prywatna) zawiera pełne definicje kursorów i podprogramów, czyli implementuje specyfikację.

Czas życia zmiennych i stałych pakietu ogranicza się do sesji, która pakiet wywołała.

# Pakiety

```
CREATE [OR REPLACE] PACKAGE nazwa_pakietu
AS | IS
   <deklaracja obiektów publicznych, w przypadku procedur i
   funkcji specyfikacja nagłówków>
END nazwa_pakietu;
CREATE [OR REPLACE] PACKAGE BODY nazwa_pakietu
AS | IS
<definicje obiektów publicznych i prywatnych>
[BEGIN]
<instrukcje inicjalizujące>
END nazwa_pakietu;
```

Część prywatna pakietu jest opcjonalna, pakiet może zawierać tylko deklaracje. Podprogramy deklarowane w części publicznej pakietu muszą znaleźć się na końcu – muszą być poprzedzone wszystkimi innymi deklaracjami.

Instrukcje inicjalizujące wykonywane są tylko jeden raz – przy pierwszym uruchomieniu pakietu w sesji.

# Pakiety

Przykład pakietu zawierającego dwie procedury – zatrudnienie i zwolnienie pracownika – prezentujemy poniżej. Z konieczności całość pakietu została przedstawiona na trzech slajdach.

```
CREATE OR REPLACE PACKAGE Obsluga_Prac
AS
  v_zat INT; v_zwol INT;
  PROCEDURE Zatrudnij (v_ename VARCHAR2
                      ,v_sal NUMBER
                      ,v_dname VARCHAR2);
  PROCEDURE Zwolnij (v_empno INT);
END Obsluga_Prac;
/
```

# Pakiety

```
CREATE OR REPLACE PACKAGE BODY Obsluga_Prac
AS
PROCEDURE Zatrudnij (v_ename VARCHAR2
                    ,v_sal NUMBER
                    ,v_dname VARCHAR2)
AS
  v_empId INT;
BEGIN
  SELECT NVL(Max(empno ), 0) + 1 INTO v_empId
   FROM emp;
  INSERT INTO emp (empno, ename, sal, deptno, hiredate)
  SELECT v_empId, v_ename,  v_sal, deptno, SYSDATE
   FROM dept
   WHERE dname = v_dname;
  COMMIT;
  v_zat := v_zat + 1;
END Zatrudnij;
```

# Pakiety

```
PROCEDURE Zwolnij (v_empno INT)
IS
BEGIN
  DELETE FROM emp
   WHERE empno = v_empno;
   COMMIT;
   v_zwol := v_zwol + 1;
END Zwolnij;
BEGIN
   v_zwol := 0;
   v_zat := 0;
END Obsluga_Prac;
```

Wywołanie procedury Zatrudnij :

```
BEGIN
   obsluga_prac.zatrudnij('Malinowski', 1225, 'SALES');
END;
```

# Wyzwalacze na tabelach bazy danych

Wyzwalacze w Pl/SQL są procedurami wiązanymi z jednym z obiektów:

- tabelą
- perspektywą
- schematem (kontem użytkownika)
- całą bazą danych

Wyzwalacze są uruchamiane przez SZBD w wyniku zaistnienia odpowiedniego zdarzenia, które może być zdarzeniem systemowym, albo jedną z instrukcji **INSERT**, **UPDATE**, **DELETE** zachodzącym na tabeli lub perspektywie, z którą związany jest wyzwalacz.

Rola wyzwalaczy w Pl/SQL, tabelowych i wyzwalaczy **INSTAED OFF** związanych z tabelami i perspektywami jest identyczna, jak rola wyzwalaczy w T-SQL. Jednak sama „filozofia" konstrukcji i działania wyzwalaczy różni się zasadniczo pomiędzy tymi językami.

# Wyzwalacze na tabelach bazy danych

Przy definiowaniu wyzwalacza tabelowego w Pl/SQL określa się:

- z jaką sekwencją instrukcji **INSERT**, **UPDATE**, **DELETE** wyzwalacz będzie związany (czyli jakie operacje na tabeli mogą go uruchomić),
- czy działanie wyzwalacza będzie dotyczyło pojedynczych wierszy (wyzwalacz wierszowy), czy też całej tabeli,
- czy wyzwalacz zostanie uruchomiony przed (**BEFORE**), czy też po (**AFTER**) wykonaniu instrukcji, która go uruchamia.

Składnia deklaracji wyzwalacza w ORACLE wygląda następująco:

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza
{BEFORE | AFTER} specyfikacja_instrukcji_DML
ON nazwa_tabeli
[FOR EACH ROW]
blok Pl/SQL
```

# Wyzwalacze na tabelach bazy danych

`specyfikacja_instrukcji_DML` to ciąg do trzech nazw instrukcji **`INSERT`**, **`UPDATE`**, **`DELETE`** połączonych spójnikiem **`OR`**. Kolejność specyfikacji instrukcji nie ma znaczenia.

W przypadku **`UPDATE`** można podać nazwy kolumn, których modyfikacja ma uruchamiać wyzwalacz:

```
UPDATE OF kolumna (, …)
```

Kolejność uruchamiania (odpalania) wyzwalaczy jest następująca:

1. Wyzwalacz przed instrukcją,
2. wyzwalacz przed pierwszym wierszem, na którym operuje instrukcja,
3. wyzwalacz po pierwszym wierszu, na którym operuje instrukcja,
4. …
5. wyzwalacz przed ostatnim wierszem,
6. wyzwalacz po ostatnim wierszu,
7. wyzwalacz po instrukcji.

# Wyzwalacze na tabelach bazy danych

W wyzwalaczu wierszowym (`FOR EACH ROW`) informacje o wartościach zmienianych w wyniku działania instrukcji DML dostępne są wewnątrz wyzwalacza przez odwołania:

`:OLD.wyrażenie` – wartość przed zmianą

`:NEW.wyrażenie` – wartość po zmianie

np.

```
IF :new.sal <= :old.sal THEN …
```

W Pl/SQL istnieje szereg ograniczeń w stosunku do operacji, które mogą być wykonywane w wyzwalaczach:

- Nie wolno używać instrukcji **`COMMIT`** ani **`ROLLBACK`**,
- W wyzwalaczu wierszowym (**`FOR EACH ROW`**) nie wolno odczytywać ani zmieniać wartości w tabeli zmienianej, z wyjątkiem wstawiania pojedynczego wiersza instrukcją **`INSERT`** …**`INTO`** … **`VALUES`**. Tabela zmieniana to tabela, z którą związany jest wyzwalacz, albo tabela odwołująca się do tej tabeli przez więzy referencyjne z opcją **`ON DELETE CASCADE`** lub **`ON DELETE SET NULL`**.

# Wyzwalacze na tabelach bazy danych

Z jedną tabelą można powiązać wiele wyzwalaczy. Jednak nie istnieje możliwość sterowania kolejnością ich uruchamiania (decyduje tutaj SZBD). W związku z tym nie należy tworzyć wyzwalaczy, których działanie wiąże się z określoną kolejnością uruchamiania.

Istnieje możliwość sprawdzenia, która instrukcja uruchomiła wyzwalacz. Służą do tego predykaty **INSERTING**, **UPDATING**, **DELETING**:

**IF** {**DELETING** | **INSERTING** | **UPDATING**} **THEN** … **END IF**;

Informacja ta może być istotna, jeżeli wyzwalacz może być uruchomiony przez więcej niż jedną instrukcję DML.

# Wyzwalacze na tabelach bazy danych

**Przykład:**

*Utwórz na tabeli **EMP** wyzwalacz, który po każdej zmianie w tabeli wpisze nowy rekord do tabeli **BUDZET (Wartosc, Data_aktualizacji)**.*

```
CREATE OR REPLACE TRIGGER Emp_budzet_trigg
AFTER INSERT OR UPDATE OR DELETE
ON emp
DECLARE
   v_sumsal emp.sal%type;
BEGIN
   SELECT  SUM(sal) INTO v_sumsal FROM emp;
   INSERT INTO Budzet (Wartosc, Data_aktualizacji)
   VALUES       (v_sumsal, Sysdate);
   dbms_output.put_line('Budzet na dzien ' || Sysdate ||
                        ':' || v_sumsal);
END;
```

# Wyzwalacze na tabelach bazy danych

**Przykład:**

*Utwórz na tabeli **EMP** wyzwalacz, który nie dopuści do takiej zmiany płacy (**sal**), która wychodziła by poza aktualną grupę zarobkową pracownika (**grade**).*

```
CREATE OR REPLACE TRIGGER Upd_Grade_Trigg
BEFORE UPDATE ON emp
FOR EACH ROW
DECLARE
  v_Ograde salgrade.grade%type; v_Ngrade salgrade.grade%type;
BEGIN
  SELECT grade INTO v_Ograde FROM Salgrade
  WHERE :old.sal Between Losal AND Hisal;
  SELECT grade INTO v_Ngrade FROM Salgrade
  WHERE :new.sal Between Losal AND Hisal;
    IF v_Ograde != v_Ngrade THEN
       :new.sal := :old.sal;
       dbms_output.put_line('Nie zmieniamy grupy zarobkowej!');
END IF;
END;
```

# Wyzwalacze na tabelach bazy danych

Wyzwalacz w przykładzie z poprzedniego slajdu musi być **BEFORE UPDATE**, ponieważ dokonuje sprawdzenia poprawności zmiany PRZED modyfikacją tabeli. Wyzwalacz **AFTER UPDATE** nie może dokonywać zmian już zmienionych wartości w tabeli.

Uruchomienie tego wyzwalacza instrukcją

```
UPDATE emp SET sal = 2500
WHERE empno IN (7369, 7900, 7782);
```

skutkuje wykonaniem sprawdzenia i ewentualnej korekty wartości na trzech rekordach, których dotyczy instrukcja **UPDATE**.

Odwołanie do wyrażenia `:old.nazwa_kolumny` w wyzwalaczu **BEFORE INSERT** lub `:new.nazwa_kolumny` w wyzwalaczu **AFTER DELETE** zwraca NULL, nie podnosząc błędu.

# Wyzwalacze INSTEAD OF

Podobnie jak w MS SQL Server, ORACLE oferuje możliwość prowadzenia operacji DML poprzez perspektywy utworzone na więcej niż jednej tabeli (patrz możliwość aktualizacji danych poprzez perspektywy). Realizacja tego typu operacji możliwa jest przy użyciu wyzwalaczy INSTEAD OF definiowanych dla perspektyw.

Składnia tego typu wyzwalaczy wygląda następująco:

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza
INSTEAD OF specyfikacja_instrukcji_DML
ON nazwa_perspektywy
[FOR EACH ROW]
blok Pl/SQL
```

Blok instrukcji Pl/SQL wraz z instrukcjami SQL pozwala na wykonanie niezależnych operacji DML na wszystkich tabelach, do których odwołuje się perspektywa, podczas gdy perspektywa izoluje szczegóły tych operacji (zatem także strukturę tabel) od użytkownika.

Z uwagi na brak innych różnic pomiędzy wyzwalaczami tabelowymi i Instead Of nie prezentuję osobnego przykładu.

# Wyzwalacze systemowe (bazy danych)

ORACLE pozwala zdefiniować wyzwalacze uruchamiane nie przez operacje DML związane z tabelą lub perspektywą, lecz przez zdarzenia zachodzące na bazie danych (zdarzenia bazodanowe) i zdarzenia DDL.

Zdarzenia bazodanowe to **SERVERERROR**, **LOGON**, **LOGOFF**, **STARTUP**, **SHUTDOWN**.

Zdarzenia DDL to nazwy instrukcji DDL i DCL takie jak **CREATE**, **ALTER**, **DROP**, **GRANT**, **REVOKE**.

Zdarzenia mogą być łączone w jednym wyzwalaczu systemowym za pomocą operatora OR.

Składnia tego typu wyzwalaczy wygląda następująco:

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza
[BEFORE | AFTER | INSTEAD OF][zdarzenie bazodanowe|DDL]
ON [DATABASE | SCHEMA]
blok Pl/SQL
```

# Operacje na wyzwalaczach

Wyzwalacze mogą być włączane i wyłączane za pomocą instrukcji :

```
ALTER TRIGGER nazwa_wyzwalacza {ENABLE | DISABLE];
```

Wyzwalacz może zostać usunięty za pomocą instrukcji:

```
DROP TRIGGER nazwa_wyzwalacza;
```

Informacja na temat wyzwalaczy dostępne są w Sqldeveloper w oknie **Connections** po rozwinięciu gałęzi **Triggers** drzewa **Connection**. Spod prawego klawisza (menu kontekstowe) po wybraniu nazwy wyzwalacza w drzewie dostępne są operacje: **Edit**, **Debug**, **Compile**, **Compile for Debug**.