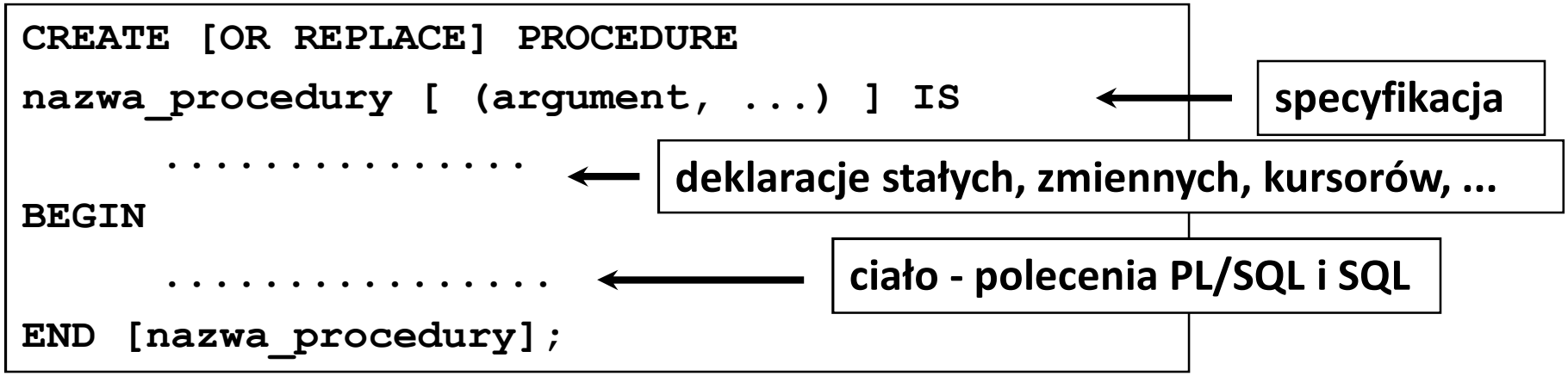
Język PL/SQL. Rozdział 4. Procedury i funkcje składowane

Podprogramy, procedury składowane, funkcje składowane, wywoływanie podprogramów.

Podprogramy

- Przez podprogramy rozumiemy głównie:
  - procedury (wykonują określone akcje),
  - funkcje (wykonują obliczenia i zwracają wartości) i
  - pakiety (grupują logicznie powiązane procedury, funkcje, zmienne i kursory)
- Własności:
  - są trwale przechowywane w bazie danych w postaci zarówno skompilowanej jak i źródłowej,
  - dzięki postaci skompilowanej osiągają większą szybkość działania niż kod wykonywany ad-hoc (bloki anonimowe),
  - mogą być współdzielone przez wielu użytkowników.
- Zalety:
  - rozszerzalność
  - modularność
  - łatwość pielęgnowania kodu
  - możliwość wielokrotnego użycia kodu
  - ukrycie szczegółów implementacji

Definiowanie procedury



- nazwa procedury musi być unikalna w ramach schematu (lub pakietu)
- między słowami kluczowymi IS i BEGIN umieszczamy deklaracje wszystkich zmiennych i kursorów lokalnych
- między słowami kluczowymi BEGIN i END umieszczamy kod PL/SQL, który wykonuje dana procedura

Parametry procedur i funkcji

```
nazwa [ IN | [ OUT | IN OUT [ NOCOPY ] ] ] typ
[ DEFAULT wartość ]
```

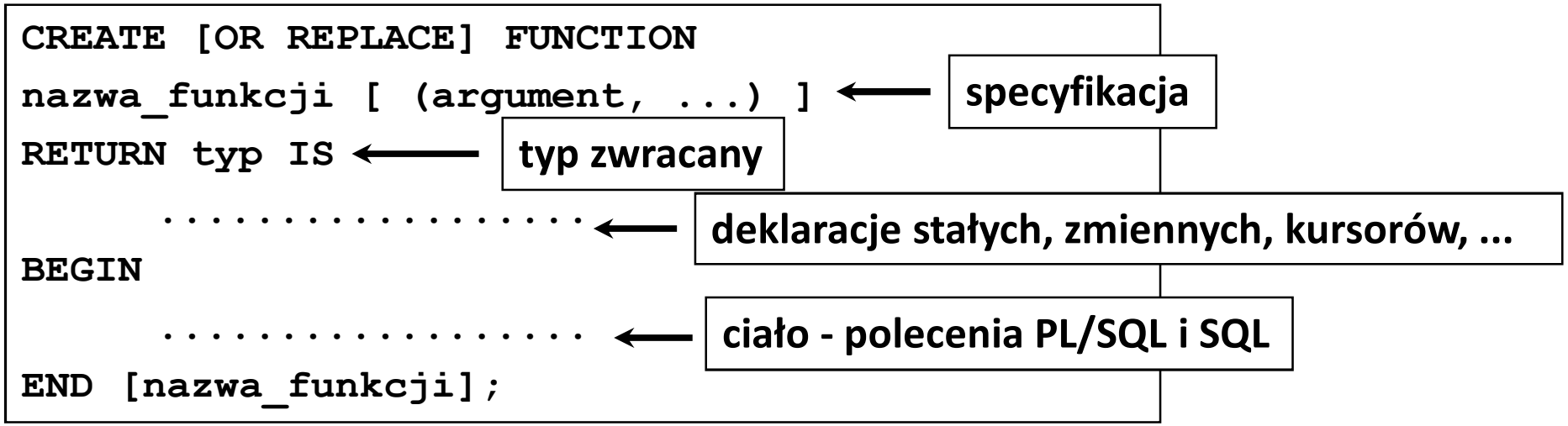
- na liście parametrów nie podajemy rozmiaru (tylko typ),
- parametr ***formalny***: używany w deklaracji procedury i w części wykonywalnej PL/SQL,
- parametr ***aktualny***: używany przy wywoływaniu procedury.

IN	OUT	IN OUT
Wartość przekazywana do programu przez referencję	Wartość zwracana do środowiska przez kopiowanie (domyślnie) lub referencję (kl. NOCOPY)	Wartość przekazywana do programu i zwracana do środowiska przez kopiowanie (domyślnie) lub referencję (kl. NOCOPY)
W programie zachowuje się jak stała	W programie zachowuje się jak nie zainicjalizowana zmienna	W programie zachowuje się jak zainicjalizowana zmienna
Musi być literałem, wyrażeniem, stałą lub zmienną	Musi być zmienną	Musi być zmienną

Przykład procedury

```
CREATE OR REPLACE PROCEDURE
  sprawdz_asystentow (p_id_zesp IN NUMBER DEFAULT 10,
                      p_ilu_asystentow OUT NUMBER) IS
BEGIN
  SELECT COUNT(*) INTO p_ilu_asystentow
  FROM pracownicy
  WHERE id_zesp = p_id_zesp AND etat = 'ASYSTENT';
  IF p_ilu_asystentow > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Asystenci zespołu ' || to_char(p_id_zesp) || ':');
    FOR cur_rec IN
      (SELECT * FROM pracownicy
       WHERE id_zesp = p_id_zesp AND etat = 'ASYSTENT'
       ORDER BY nazwisko)
       LOOP
          DBMS_OUTPUT.PUT_LINE(cur_rec.nazwisko);
       END LOOP;
  ELSE
   DBMS_OUTPUT.PUT_LINE('W zespole ' || p_id_zesp || ' nie ma asystentów!');
  END IF;
END sprawdz_asystentow;
/
```

Definiowanie funkcji



- nazwa funkcji musi być unikalna w ramach schematu (lub pakietu)
- po słowie kluczowym RETURN umieszczamy typ zwracany przez funkcję
- między słowami kluczowymi IS i BEGIN umieszczamy deklaracje wszystkich zmiennych i kursorów lokalnych
- między słowami kluczowymi BEGIN i END umieszczamy kod PL/SQL, który wykonuje dana funkcja
- przetwarzanie funkcji musi zakończyć się instrukcją RETURN (w kodzie PL/SQL **musi** się znaleźć instrukcja RETURN)

Przykład funkcji

```
CREATE OR REPLACE FUNCTION podatek (p_id_prac IN NUMBER) RETURN NUMBER IS
  CURSOR c_pracownik IS
    SELECT * FROM pracownicy
    WHERE id_prac = p_id_prac;
  v_pracownik pracownicy%ROWTYPE;
  v_roczne_zarobki NUMBER;
  v_podatek NUMBER;
BEGIN
  OPEN c_pracownik;
  FETCH c_pracownik INTO v_pracownik;
  CLOSE c_pracownik;
  v_roczne_zarobki := 12 * v_pracownik.placa_pod +
                      NVL(v_pracownik.placa_dod, 0);
  IF (v_roczne_zarobki > 5000) THEN v_podatek := 0.40 * v_roczne_zarobki;
  ELSIF (v_roczne_zarobki > 3000) THEN v_podatek := 0.30 * v_roczne_zarobki;
  ELSE v_podatek := 0.19 * v_roczne_zarobki;
  END IF;
  RETURN v_podatek;
END podatek;
/
```

Wyświetlanie informacji o błędach

```
create or replace procedure proc_test is
begin
  a := 10;
end;
```

```
PROCEDURE proc_test compiled
Warning: wykonywanie ukończono z ostrzeżeniem
```

- w narzędziach firmy Oracle (iSQL*Plus, SQL Developer, …)

```
show errors
```

```
3/3             PLS-00363: wyrażenie 'A' nie może być użyte jako cel przypisania
3/3             PL/SQL: Statement ignored
```

nr linii/nr pozycji

- bezpośrednio ze słownika bazy danych

```
SELECT * FROM user_errors
WHERE type = 'PROCEDURE'
AND   name = 'PROC_TEST';
```

NAME	TYPE	SEQUENCE	LINE	POSITION	TEXT
PROC_TEST	PROCEDURE	1	3	3	PLS-00363: wyrażenie 'A' nie może być użyte jako cel przypisania
PROC_TEST	PROCEDURE	2	3	3	PL/SQL: Statement ignored

Wywołanie procedur i funkcji

- Wywołania procedur mogą następować tylko w blokach PL/SQL
- Wywołania funkcji mogą następować zarówno w blokach PL/SQL jak i poleceniach SQL

```
SELECT PODATEK(id_prac)
FROM pracownicy;
```

```
DECLARE
  v_liczba_asystentow NUMBER;
  v_kwota_podatku NUMBER;
BEGIN
   SPRAWDZ_ASYSTENTOW (20, v_liczba_asystentow);
   v_kwota_podatku := PODATEK(100);
END;
/
```

- Funkcja wywołana z poziomu PL/SQL musi posiadać cel przypisania zwracanej przez siebie wartości
- Wywoływanie funkcji bez parametrów może odbyć się zarówno przy użyciu nawiasów jak i bez ich użycia

```
DECLARE
  v_data DATE;
BEGIN
  v_data := SYSDATE;
  v_data := SYSDATE();
END;
```

- Wywołanie funkcji z poziomu SQL jest możliwe tylko wtedy, gdy funkcja spełnia określone zasady – posiada odpowiedni poziom „czystości”

Wywołanie procedur i funkcji
Notacja parametrów aktualnych

- Podczas wywołania procedur i funkcji wartości parametrów można określać za pomocą:
  - notacji pozycyjnej (piąta wartość – piąty parametr),
  - notacji nazewniczej (wartości mają określone nazwy parametrów którym odpowiadają),
  - notacji mieszanej (pierwszych *n* parametrów zgodnie z pozycją, pozostałe na podstawie nazwy).
- Jedynie parametry posiadające wartości domyślne mogą być pominięte podczas wywołania

```
DECLARE
  v_liczba_asystentow NUMBER;
  v_kwota_podatku NUMBER;
BEGIN
   SPRAWDZ_ASYSTENTOW(20, v_liczba_asystentow);
   SPRAWDZ_ASYSTENTOW(p_ilu_asystentow => v_liczba_asystentow, p_id_zesp => 30);
   SPRAWDZ_ASYSTENTOW(40, p_ilu_asystentow => v_liczba_asystentow);
   SPRAWDZ_ASYSTENTOW(p_ilu_asystentow => v_liczba_asystentow);
END;
/
```

Czystość funkcji

- Aby funkcja mogła być wywoływana z poziomu polecenia SQL, musi posiadać odpowiedni poziom czystości.
  Poniżej wymieniono najbardziej podstawowe reguły:
  - funkcja wywoływana z polecenia SELECT
    - nie może modyfikować danych relacji bazy danych,
  - funkcja wywoływana z poleceń INSERT, UPDATE i DELETE
    - nie może odczytywać i modyfikować danych relacji, której dotyczy polecenie,
  - funkcja wywoływana z poleceń SELECT, INSERT, UPDATE i DELETE
    - nie może zawierać poleceń sterujących sesją i transakcjami (np. COMMIT, ALTER SESSION) oraz instrukcji DDL.

Kompilowanie procedur i funkcji

- Podobnie jak w przypadku perspektyw, podprogramy (w tym procedury i funkcje)
  - mogą odwoływać się do innych obiektów bazy danych
  - posiadają tzw. status
- Każda modyfikacja obiektu bazy danych skutkuje utratą statusu VALID obiektów odwołujących się do modyfikowanego obiektu,
- Aby przywrócić ten status, i potwierdzić w ten sposób poprawność obiektów odwołujących się, należy przeprowadzić ich rekompilację

```
CREATE OR REPLACE PROCEDURE
  sprawdz_asystentow (...) IS
BEGIN
  SELECT COUNT(*) INTO p_ilu_asystentow
  FROM pracownicy
  WHERE id_zesp = p_id_zesp AND etat = 'ASYSTENT';
...
```

```
SELECT object_name, status
FROM   user_objects
WHERE  object_type = 'PROCEDURE'
AND    object_name = 'SPRAWDZ_ASYSTENTOW'
```

OBJECT_NAME	STATUS
SPRAWDZ_ASYSTENTOW	VALID

```
SELECT referenced_name, referenced_type
FROM   user_dependencies
WHERE  name = 'SPRAWDZ_ASYSTENTOW'
AND    type = 'PROCEDURE';
```

REFERENCED_NAME	REFERENCED_TYPE
STANDARD	PACKAGE
SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
DBMS_OUTPUT	SYNONYM
PRACOWNICY	TABLE

```
alter table pracownicy
modify nazwisko VARCHAR2(15);
```

OBJECT_NAME	STATUS
SPRAWDZ_ASYSTENTOW	INVALID

```
ALTER PROCEDURE | FUNCTION nazwa COMPILE;
```

```
ALTER PROCEDURE SPRAWDZ_ASYSTENTOW COMPILE;
```

OBJECT_NAME	STATUS
SPRAWDZ_ASYSTENTOW	VALID

Słownik bazy danych

- USER_OBJECTS – informacja o wszystkich obiektach w schemacie użytkownika (w tym procedurach i funkcjach)

```
SELECT object_name, object_type, status
FROM    user_objects
WHERE  object_type IN ('PROCEDURE','FUNCTION');
```

- USER_SOURCE – kod źródłowy podprogramów użytkownika składowanych w bazie danych (w tym procedur i funkcji)

```
SELECT text
FROM   user_source
WHERE  name = 'SPRAWDZ_ASYSTENTOW'
AND    type = 'PROCEDURE'
ORDER BY line;
```

Usuwanie procedur i funkcji

- Usuwanie funkcji oraz procedur możliwe jest za pomocą polecenia DROP

```
DROP PROCEDURE | FUNCTION nazwa;
```

Zagadnienia uzupełniające

- Modele uprawnień wykorzystywane podczas wykonywania procedur i funkcji
- Kilka uwag dotyczących funkcji
  - Klauzula DETERMINISTIC
  - Klauzula RESULT_CACHE
- Lokalne procedury i funkcje
  - Przeciążanie
  - Deklaracje
- Zmienne środowiskowe i ich wykorzystanie

Modele uprawnień wykorzystywane podczas wykonywania procedur i funkcji

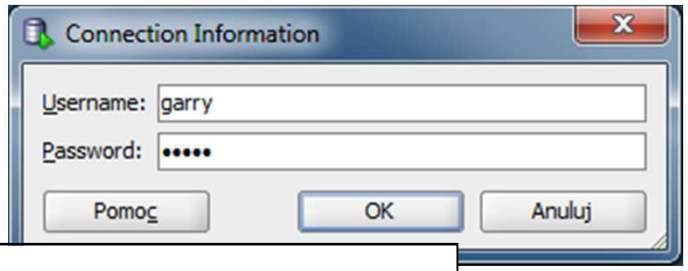
- Standardowo procedura (funkcja) wykonywana jest zgodnie z modelem uprawnień użytkownika definiującego procedurę (funkcję)

```
CREATE OR REPLACE PROCEDURE [schemat.]nazwa[( parametr[, parametr...] ) ]
[AUTHID DEFINER | CURRENT_USER] IS
```

- Klauzula *AUTHID* pozwala określić czy procedura lub funkcja ma być wykonywana wg modelu uprawnień właściciela (definiującego) czy też z użytkownika, który z procedury lub funkcji korzysta.

```
CREATE OR REPLACE PROCEDURE
  sprawdz_asystentow (p_id_zesp IN NUMBER DEFAULT 10,
                      p_ilu_asystentow OUT NUMBER)
  AUTHID CURRENT_USER IS
BEGIN
  SELECT COUNT(*) INTO p_ilu_asystentow
  FROM pracownicy
```

```
grant execute on SPRAWDZ_ASYSTENTOW to garry;
```

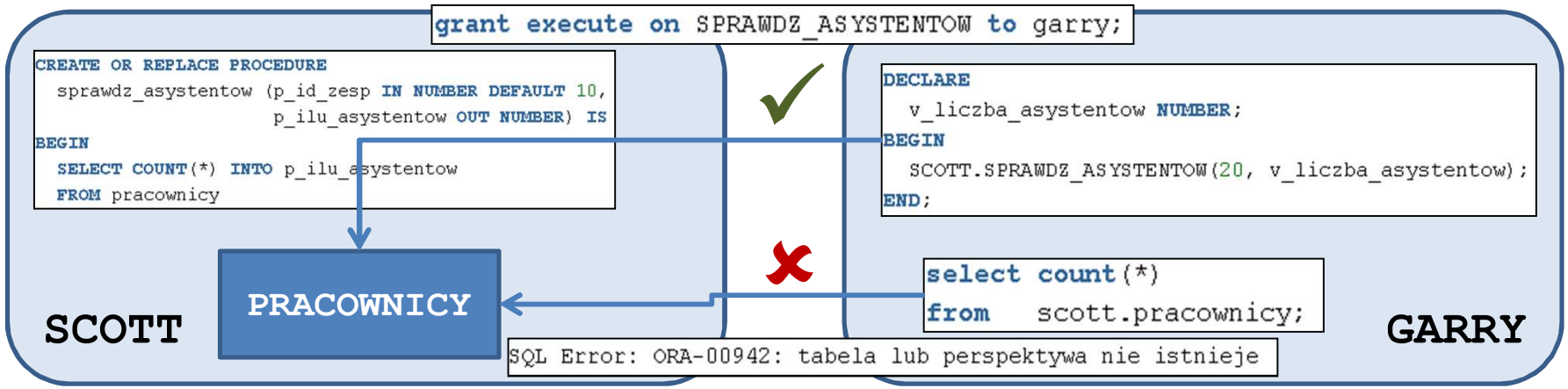



```
DECLARE
  v_liczba_asystentow NUMBER;
BEGIN
  SCOTT.SPRAWDZ_ASYSTENTOW(20, v_liczba_asystentow);
END;
```

```
Error report:
ORA-00942: tabela lub perspektywa nie istnieje
ORA-06512: przy "SCOTT.SPRAWDZ_ASYSTENTOW", linia 6
ORA-06512: przy linia 4
```

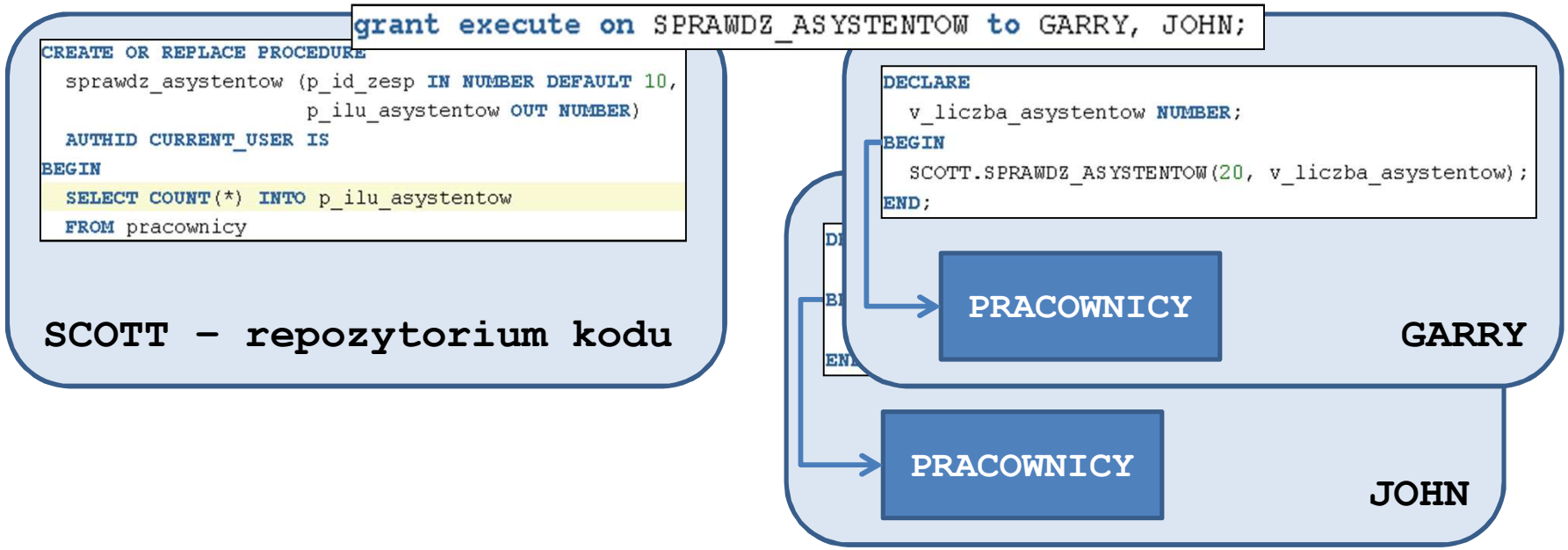
Model uprawnień właściciela obiektu

- Reguły:
  - każda referencja w podprogramie jest wyznaczana na etapie kompilacji z wykorzystaniem bezpośrednich uprawnień (uprawnienia nadane poprzez role są ignorowane)
  - wykonywanie podprogramu odbywa się pod kontrolą schematu właściciela
- Zalety:
  - umożliwia implementację wyrafinowanej (proceduralnej) kontroli dostępu
  - większa wydajność – silnik PL/SQL nie weryfikuje uprawnień podczas wykonywania
- Wady:
  - konieczność posiadania bezpośrednich uprawnień
  - konieczność propagacji tego samego kodu do wielu schematów wykorzystujących takie same zestawy obiektów



Model uprawnień użytkownika obiektu

- Reguły:
  - każda referencja w podprogramie jest wyznaczana na etapie wykonywania
  - oprócz uprawnień nadanych bezpośrednio wykorzystywane są także uprawnienia nadane poprzez role (wyjątek to wywołanie z podprogramu opartego o model uprawnień właściciela)



Kilka uwag dotyczących funkcji

```
CREATE OR REPLACE FUNCTION [schemat.]nazwa[( parametr[, parametr...] ) ]
RETURN typ
[DETERMINISTIC] [RESULT_CACHE [RELIES_ON (tabela, tabela, …)]] IS
```

- Klauzula DETERMINISTIC deklaruje, że funkcja zwraca taki sam wynik dla tych samych wartości parametrów (IN, IN OUT)
  - konieczna podczas definiowania indeksów funkcyjnych
  - konieczne w przypadku perspektyw materializowanych odświeżanych przyrostowo lub wykorzystywanych podczas przepisywania zapytań
- Klauzula RESULT_CACHE
  - pozwala serwerowi bazy danych buforować wyniki funkcji w SGA i wykorzystywać je ponownie dla tych samych parametrów wywołania.
    Dotyczy to nawet różnych poleceń w ramach tej samej lub różnych sesji.
  - uzupełniona o klauzulę RELIES_ON pozwala wskazać tabele, których zatwierdzone zmiany wymuszają usunięcie zbuforowanych wyników funkcji

```
create or replace FUNCTION
  roczny_dodatek(dodatek NUMBER)
  RETURN VARCHAR2 RESULT_CACHE
IS
BEGIN
  DBMS_OUTPUT.put_line
    ('roczny_dodatek=' || dodatek);
  IF dodatek IS NULL THEN
    RETURN 0;
  ELSE
    RETURN dodatek*12;
  END IF;
END;
```

```
select sum(roczny_dodatek(p.placa_dod))
from pracownicy p
```

```
SUM(ROCZNY_DODATEK(P.PLACA_DOD))
--------------------------------
                         12919,2
```

```
roczny_dodatek=420,5
roczny_dodatek=210
roczny_dodatek=
roczny_dodatek=105
roczny_dodatek=80,5
roczny_dodatek=170,6
roczny_dodatek=90
```

Lokalne procedury i funkcje

- Istnieje możliwość definiowania lokalnych procedur i funkcji w ramach części deklaracyjnej bloku PL/SQL (anonimowego lub nazwanego – składowanego). Podstawowe zalety takiego rozwiązania:
  - redukcja kodu poprzez wyodrębnienie powtarzających się fragmentów
  - poprawa czytelności kodu
- Lokalne procedury i funkcje mogą być przeciążane
- W przypadku gdy podprogramy wywołują się wzajemnie konieczna może być ich deklaracja. Deklaracja składa się z samego nagłówka.

```
DECLARE
  v_liczba_prac_id    NUMBER;
  v_liczba_prac_nazwa NUMBER;
  FUNCTION liczba_prac(p_nazwa_zesp VARCHAR2)
    RETURN NUMBER IS
    v_liczba_prac NUMBER;
  BEGIN
    SELECT COUNT(*) INTO v_liczba_prac
    FROM pracownicy
    WHERE id_zesp = (SELECT id_zesp FROM zespoly
                     WHERE nazwa = p_nazwa_zesp);
    RETURN v_liczba_prac;
  END;
  FUNCTION liczba_prac(p_id_zesp NUMBER)RETURN NUMBER IS
    v_liczba_prac NUMBER;
  BEGIN
    SELECT COUNT(*) INTO v_liczba_prac
    FROM pracownicy
    WHERE id_zesp = p_id_zesp;
    RETURN v_liczba_prac;
  END;
BEGIN
  FOR r_zesp IN (SELECT * FROM zespoly)
  LOOP
    v_liczba_prac_id    := liczba_prac(r_zesp.id_zesp);
    v_liczba_prac_nazwa := liczba_prac(r_zesp.nazwa);
    dbms_output.put_line(
       v_liczba_prac_id||' = '||v_liczba_prac_nazwa);
  END LOOP;
END;
```

Podprogram jako osobna transakcja

- Istnieje możliwość wykonania pod-programu jako osobnej transakcji, tzw. transakcji autonomicznej.
- Dyrektywa kompilatora: AUTONOMOUS_TRANSACTION.
- Transakcja autonomiczna powinna zostać jawnie zatwierdzona lub wycofana,
  - w przeciwnym przypadku zostaje automatycznie wycofana.
- Transakcja główna zostaje zawieszona na czas wykonania podprogramu jako transakcji autonomicznej.
- Status zakończenia transakcji autonomicznej jest niezależny od statusu transakcji głównej.

```
CREATE OR REPLACE PROCEDURE
  usun_pracownika(p_id_prac NUMBER) is
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  DELETE pracownicy
  WHERE id_prac = p_id_prac;
  COMMIT;
END usun_pracownika;
```

```
DECLARE
  v_lp NUMBER;
BEGIN
  SELECT COUNT(*) INTO v_lp FROM pracownicy;
  dbms_output.put_line('przed proc.: '||v_lp);
  usun_pracownika(210);
  SELECT COUNT(*) INTO v_lp FROM pracownicy;
  dbms_output.put_line('po proc.: '||v_lp);
  ROLLBACK;
  SELECT COUNT(*) INTO v_lp FROM pracownicy;
  dbms_output.put_line('po rollback: '||v_lp);
END;
/

przed proc.: 12
po proc.: 11
po rollback: 11
```

Zmienne środowiskowe

- W niektórych środowiskach programistycznych istnieje możliwość deklarowania własnych zmiennych, a następnie ich wykorzystywania podczas uruchamiania kodu PL/SQL czy SQL.
- Z reguły wyróżniamy następujące typy zmiennych środowiskowych:
  - Zmienne podstawienia
    - uzupełniają treść polecenia
    - integrowane z treścią polecenia zanim polecenie zostanie wysłane do serwera bazy danych, a to oznacza, że mogą być użyte np. w miejsce nazw relacji, atrybutów, fragmentów poleceń lub po prostu jako wartości atrybutów
    - nie mogą rozpoczynać polecenia
    - są zawsze typu znakowego
    - nie mogą być celem przypisania w PL/SQL
  - Zmienne wiązania
    - w kodzie PL/SQL lub SQL zachowują się jak zwykłe zmienne
    - zmienne PL/SQL użyte w kodzie SQL, z punktu widzenia polecenia SQL, również traktowane są jak zmienne wiązania
    - integrowane są z poleceniem po jego parsowaniu, po stronie serwera bazy danych
    - mogą być różnego typu

Zmienne środowisk SQL*Plus i SQL Developer

Zmienne podstawienia

- Definiowane za pomocą polecenia DEFINE
- Polecenie DEFINE bez parametrów wyświetla listę wszystkich zmiennych podstawienia.
- UNDEFINE usuwa zmienną
- Wyłączenie lub zmiana znaku wyznaczającego użycie zmiennej podstawiania odbywa się za pomocą polecenia SET DEFINE OFF|znak

```
SET DEFINE $

DEFINE myTable = 'PRACOWNICY'
DEFINE myValue = 'PROFESOR'

SELECT * FROM $myTable
WHERE etat = '$myValue';

DEFINE
```

```
DEFINE MYVALUE =  "PROFESOR" (CHAR)
DEFINE MYTABLE =  "PRACOWNICY" (CHAR)
```

```
UNDEFINE myTable
UNDEFINE myValue
```

Zmienne wiązania

- Definiowane za pomocą polecenia VARIABLE
- Polecenie VARIABLE bez parametrów wyświetla wszystkie zmienne wiązane.
- PRINT pozwala na wyświetlenie zawartości zmiennej

```
VARIABLE x NUMBER

BEGIN
  SELECT COUNT(*) INTO :x
  FROM pracownicy;
END;

PRINT x
```

```
X
--
14
```

```
VARIABLE
```

```
VARIABLE X  DATATYPE NUMBER
```

Kursorowe zmienne wiązania

- W niektórych środowiskach programistycznych oprócz prostych zmiennych wiązania dostępne są także kursorowe zmienne wiązania.
- W środowisku SQL*Plus i SQL Developer ich zachowanie jest podobne do zmiennych kursorowych dostępnych w PL/SQL
- Pozwalają na odczytanie w SQL*Plus wyniku zapytania umieszczonego w bloku PL/SQL.
- Mogą być stosowane zarówno w anonimowych blokach PL/SQL, jak i jako parametr lub typ wynikowy procedury lub funkcji.

```
VARIABLE x REFCURSOR

BEGIN
  OPEN :x FOR SELECT * FROM pracownicy;
END;

PRINT x
```

Ułatwienia wywoływania kodu PL/SQL w środowiskach SQL*Plus i SQL Developer

- SQL*Plus i SQL Developer ułatwia wywoływanie procedur i funkcji między innymi poprzez:
  - możliwość wykorzystania zmiennych środowiskowych
  - udostępnienie polecenia execute automatyzującego utworzenie anonimowego bloku PL/SQL

```
VARIABLE liczba_asystentow NUMBER
VARIABLE kwota_podatku NUMBER

execute SPRAWDZ_ASYSTENTOW (20, :liczba_asystentow);
execute :kwota_podatku := PODATEK(100);

PRINT liczba_asystentow
PRINT kwota_podatku
```

```
anonymous block completed
anonymous block completed

LICZBA_ASYSTENTOW
-----------------
3

KWOTA_PODATKU
-------------
8472
```