

Dodanie metody do istniejącego typu obiektowego

Dodanie sygnatury metody do deklaracji typu obiektowego

```
ALTER TYPE Pracownik ADD MAP MEMBER FUNCTION odwzoruj  
RETURN NUMBER CASCADE INCLUDING TABLE DATA;
```

Dodanie implementacji metody do definicji typu obiektowego

```
CREATE OR REPLACE TYPE BODY Pracownik AS  
  MEMBER FUNCTION wiek RETURN NUMBER IS  
  BEGIN  
    RETURN EXTRACT (YEAR FROM CURRENT_DATE) -  
           EXTRACT (YEAR FROM data_ur);  
  END wiek;  
  MEMBER PROCEDURE podwyzka(p_kwota NUMBER) IS  
  BEGIN  
    pensja := pensja + p_kwota;  
  END podwyzka;  
  MAP MEMBER FUNCTION odwzoruj RETURN NUMBER IS  
  BEGIN  
    RETURN ROUND(pensja,-3) + wiek();  
  END odwzoruj;  
END;
```

Konstruktor – specjalna metoda tworząca nowe obiekty

- nazwa konstruktora jest taka sama jak nazwa typu obiektowego
- użytkownik może definiować własne konstruktory, może także przesłonić domyślny konstruktor atrybutowy
- konstruktory ułatwiają inicjalizację i umożliwiają ewolucję obiektów

```
ALTER TYPE Pracownik REPLACE AS OBJECT (  
    nazwisko VARCHAR2(20),  
    pensja    NUMBER(6,2),  
    etat      VARCHAR2(15),  
    data_ur   DATE,  
    MEMBER FUNCTION wiek RETURN NUMBER,  
    MEMBER PROCEDURE podwyżka(p_kwota NUMBER),  
    MAP MEMBER FUNCTION odwzoruj RETURN NUMBER,  
    CONSTRUCTOR FUNCTION Pracownik(p_nazwisko VARCHAR2)  
        RETURN SELF AS RESULT );
```

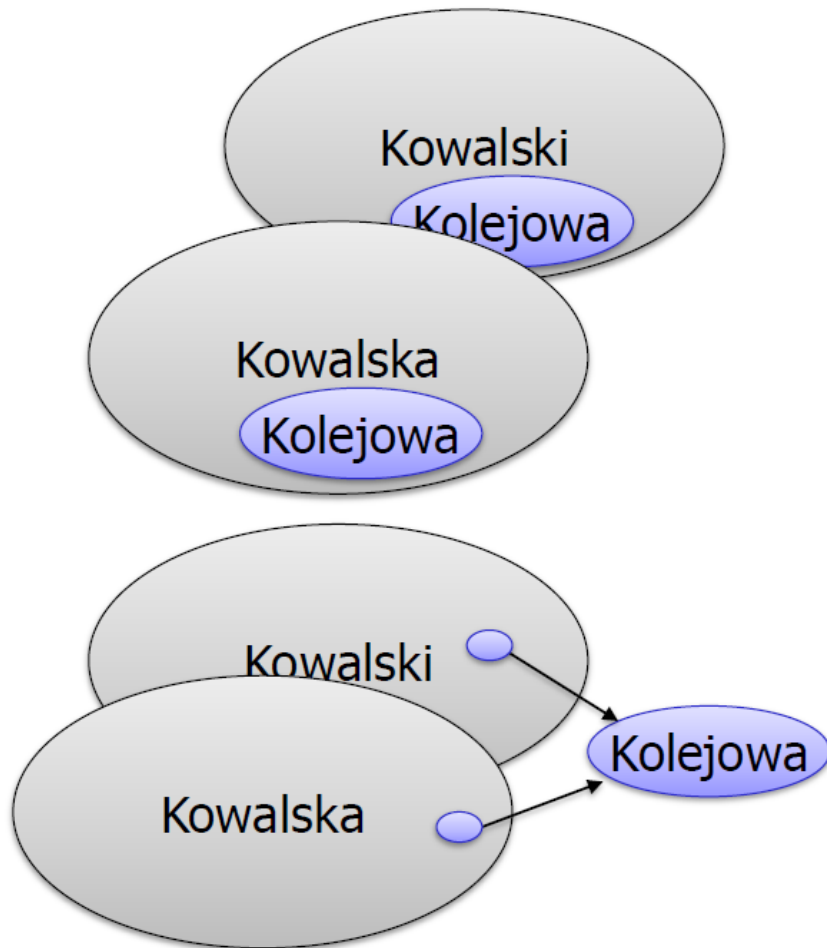
Konstruktor – implementacja

```
CREATE OR REPLACE TYPE BODY Pracownik AS  
  MEMBER FUNCTION wiek RETURN NUMBER IS  
  BEGIN  
    RETURN EXTRACT (YEAR FROM CURRENT_DATE) -  
      EXTRACT (YEAR FROM data_ur);  
  END wiek;  
  MEMBER PROCEDURE podwyzka(p_kwota NUMBER) IS  
  BEGIN  
    pensja := pensja + p_kwota;  
  END podwyzka;  
  MAP MEMBER FUNCTION odwzoruj RETURN NUMBER IS  
  BEGIN  
    RETURN ROUND(pensja,-3) + wiek();  
  END odwzoruj;  
  
  CONSTRUCTOR FUNCTION Pracownik(p_nazwisko VARCHAR2)  
    RETURN SELF AS RESULT IS  
  BEGIN  
    SELF.nazwisko := p_nazwisko; SELF.pensja := 1000;  
    SELF.etat := null; SELF.data_ur := null;  
    RETURN;  
  END;  
END;
```

Konstruktor – specjalna metoda tworząca nowe obiekty

- Typy obiektowe zwykle udostępniają konstruktor domyślny.
- Takie konstruktory zazwyczaj nie mają parametrów formalnych.
- W obiektach, których egzemplarze wymagają parametrów formalnych, konstruktor domyślny wywołuje konstruktor z parametrem domyślnym
- Odbywa się to w czterech etapach:
 1. Najpierw należy przygotować zmienną lokalną typu obiektowego
 2. Następnie trzeba utworzyć lokalny (wewnętrzny) egzemplarz klasy za pomocą domyślnej wartości argumentu
 3. Trzeci krok to przypisanie tymczasowego obiektu lokalnego do egzemplarza klasy
 4. Czwarty etap polega na zwróceniu uchwytu do obiektu

Współdzielenie i zagnieżdżanie obiektów



```
CREATE TYPE TAdres AS OBJECT (  
    ulica VARCHAR2(15),  
    dom NUMBER(4),  
    mieszkanie NUMBER(3));
```

```
CREATE TYPE Osoba AS OBJECT (  
    nazwisko VARCHAR2(20),  
    imie VARCHAR2(15),  
    adres TAdres);
```

```
CREATE TYPE Osoba AS OBJECT (  
    nazwisko VARCHAR2(20),  
    imie VARCHAR2(15),  
    adres REF TAdres);
```

Współdzielenie i zagnieżdżanie obiektów

- Utworzenie tabel obiektowych

```
CREATE TABLE AdresyObjTab OF TAdres;  
CREATE TABLE OsobyObjTab OF Osoba;
```

```
ALTER TABLE OsobyObjTab ADD  
SCOPE FOR(adres) IS AdresyObjTab;
```

- Wstawienie obiektów


```
INSERT INTO AdresyObjTab VALUES  
(NEW TAdres('Kolejowa',2,18));
```

```
INSERT INTO OsobyObjTab VALUES  
(NEW Osoba('Kowalska','Anna',null));
```

```
INSERT INTO OsobyObjTab VALUES  
(NEW Osoba('Kowalski','Jan',null));
```

```
UPDATE OsobyObjTab o  
SET o.adres = (  
    SELECT REF(a) FROM AdresyObjTab a  
    WHERE a.ulica = 'Kolejowa' );
```

aby zawęzić zakres referencji tabela
OsobyObjTab musi być pusta



Referencje – sposoby wykorzystania

- nawigacja jawna przez wywołanie funkcji **DEREF** ()
- nawigacja niejawna przez notację kropkową
- operacja połączenia tabel przez porównanie referencji

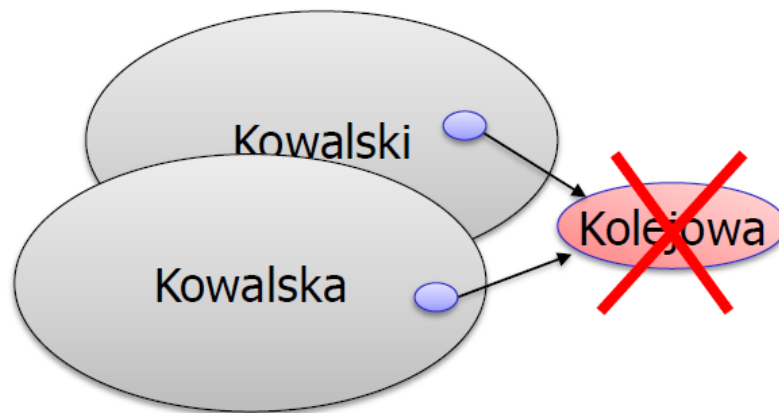
```
SELECT o.imie, o.nazwisko, DEREF(o.adres)  
FROM OsobyObjTab o;
```

```
SELECT o.imie, o.nazwisko, o.adres.ulica, o.adres.dom  
FROM OsobyObjTab o;
```

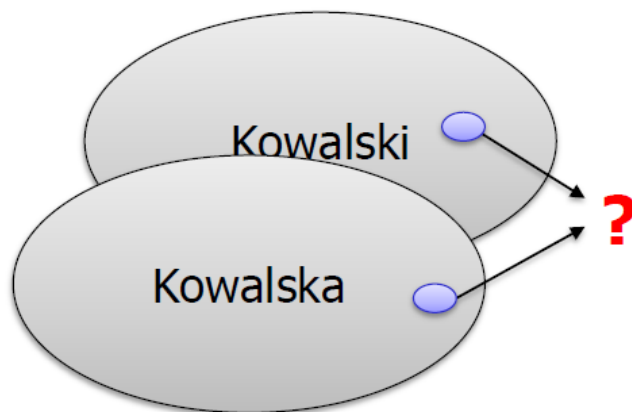
```
SELECT o.imie, o.nazwisko, a.ulica, a.dom, a.mieszkanie  
FROM OsobyObjTab o JOIN AdresyObjTab a  
    ON o.adres = REF(a);
```


Referencje – sposoby wykorzystania

- Usunięcie obiektu nie usuwa referencji do obiektu



```
DELETE FROM AdresyObjTab a  
WHERE a.ulica = 'Kolejowa';
```



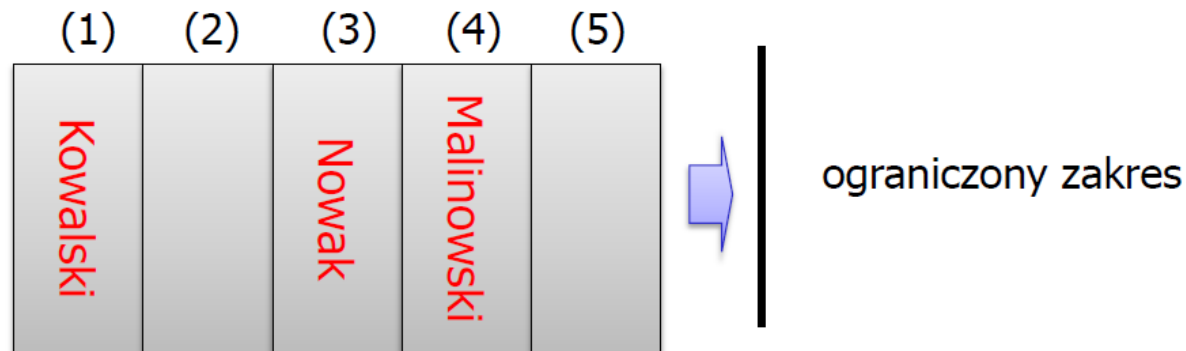
```
SELECT * FROM OsobyObjTab o  
WHERE o.adres IS NULL;
```

```
SELECT * FROM OsobyObjTab o  
WHERE o.adres IS Dangling;
```

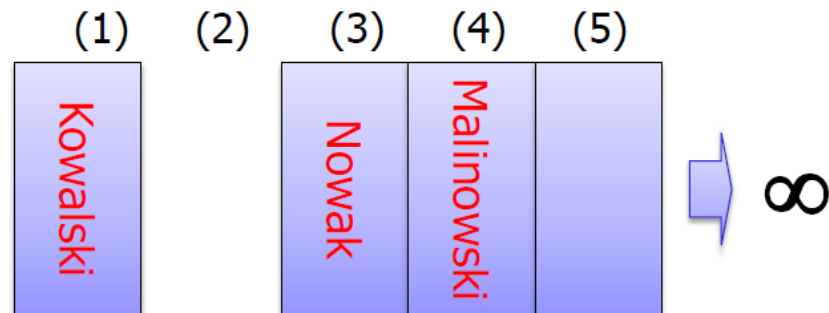
KOLEKCJE

Kolekcje to zbiory obiektów

Tablica o zmiennym rozmiarze (varray)



Tablica zagnieżdżona (nested table)



Metody kolekcji

metoda	opis
kolekcja(wartość,...)	konstruktor kolekcji, opcjonalnie wstawia wartości jako kolejne elementy kolekcji
EXTEND([n],[i])	rozszerza kolekcję o n pustych elementów, opcjonalnie wypełnia wartością i-tego elementu
TRIM([n])	usuwa n elementów od końca kolekcji
DELETE([n],[m])	usuwa wszystkie elementy kolekcji, n-ty element, lub elementy od n-tego do m-tego
NEXT(n), PRIOR(n)	zwraca indeks elementu następującego (poprzedzającego) elementu o indeksie n
EXISTS(n)	testuje istnienie elementu o indeksie n
FIRST, LAST	zwraca indeks pierwszego (ostatniego) elementu
LIMIT	zwraca maksymalny zakres kolekcji
COUNT	zwraca liczbę elementów kolekcji

Tabela o zmiennym rozmiarze w PL/SQL

```
create or replace type cisnienia as  
varray(5) of numeric(4);
```

```
declare  
    pomiary cisnienia;  
begin  
    pomiary:= cisnienia(995,1020,1009);  
    dbms_output.put_line(pomiary.limit()||' '||pomiary.count());  
    dbms_output.put_line('*****');  
for x in pomiary.first()..pomiary.last() loop  
    dbms_output.put_line(pomiary(x));  
end loop;  
    dbms_output.put_line('*****');  
    pomiary.extend(2,3);  
for x in pomiary.first()..pomiary.last() loop  
    dbms_output.put_line(pomiary(x));  
end loop;  
    dbms_output.put_line('*****');
```

```
5 3  
*****  
  
995  
1020  
1009  
*****  
  
995  
1020  
1009  
1009  
1009  
*****
```

Tabela o zmiennym rozmiarze w PL/SQL

```
create or replace type cisnienia as  
varray(5) of numeric(4);
```

```
pomiary.trim(1);  
for x in pomiary.first()..pomiary.last() loop  
  dbms_output.put_line(pomiary(x));  
end loop;  
dbms_output.put_line('*****');  
pomiary.extend();  
pomiary(5):=1000;  
for x in pomiary.first()..pomiary.last() loop  
  dbms_output.put_line(pomiary(x));  
end loop;  
dbms_output.put_line('*****');  
pomiary.delete();  
dbms_output.put_line( pomiary.count);  
end;
```

995
1020
1009
1009

995
1020
1009
1009
1000

0

Tabela o zmiennym rozmiarze w SQL

Jeśli kolekcja jest tabelą o zmiennym rozmiarze to:

- nie można manipulować pojedynczymi elementami w SQL
- elementy zachowują fizyczny porządek

```
create or replace type DZIENNY_POMIAR as object  
(  
  data_pomiaru DATE,  
  wartosci_pomiarow cisnienia);
```

```
CREATE TABLE POMIARY OF DZIENNY_POMIAR;
```

```
INSERT INTO POMIARY VALUES(DZIENNY_POMIAR(  
  TO_DATE('028/04/2017','DD/MM/YYYY'), CISNIENIA(995,1020,1009)));
```

```
INSERT INTO POMIARY VALUES(DZIENNY_POMIAR(  
  TO_DATE('04/05/2017','DD/MM/YYYY'), CISNIENIA(990,1000)));
```

Tabela o zmiennym rozmiarze w SQL

```
SELECT DATA_POMIARU, WARTOSCI_POMIAROW  
FROM POMIARY;
```

	DATA_POMIARU	WARTOSCI_POMIAROW
1	17/04/28	II372.CISNIENIA(995,1020,1009)
2	17/05/04	II372.CISNIENIA(990,1000)

Tabela o zmiennym rozmiarze w SQL

```
UPDATE POMIARY  
SET WARTOSCI_POMIAROW=CISNIENIA(1100,1001,1015,999)  
WHERE  
DATA_POMIARU=TO_DATE('04/05/2017','DD/MM/YYYY');
```

```
SELECT DATA_POMIARU, WARTOSCI_POMIAROW  
FROM POMIARY;
```

jest

	DATA_POMIARU	WARTOSCI_POMIAROW
1	17/04/28	II372.CISNIENIA(995,1020,1009)
2	17/05/04	II372.CISNIENIA(1100,1001,1015,999)

było

	DATA_POMIARU	WARTOSCI_POMIAROW
1	17/04/28	II372.CISNIENIA(995,1020,1009)
2	17/05/04	II372.CISNIENIA(990,1000)

Przykład kolekcji - tabela o zmiennym rozmiarze w SQL

Przykład

1. Utwórz typ tablicy o zmiennym rozmiarze przechowujący maksymalnie 50 liczb, który będzie reprezentował listę ocen studenta
2. Utwórz typ obiektowy Student, o atrybutach:
 - nr_indeksu (number),
 - nazwisko (varchar2)
 - Imie(vvarchar2)
 - oceny (typ wcześniej zdefiniowanej kolekcji).
3. Utwórz tabelę obiektową STUDENCI, która będzie przechowywała obiekty typu STUDENT.
4. Wstaw za pomocą polecenia INSERT kilku studentów, razem z ocenami, do tabeli.
5. Odczytaj tabelę za pomocą polecenia SELECT.
6. Zmień listę ocen jednego ze studentów za pomocą polecenia UPDATE

Tabela o zmiennym rozmiarze w PL/SQL

```
create or replace type oceny_studenta  
as varray(30) of numeric(2,1)
```

```
create or replace type student_ as object(  
indeks numeric (5),  
nazwisko varchar2(20),  
imie varchar2(20),  
oceny oceny_studenta  
);
```

```
CREATE TABLE STUDENCI OF STUDENT_;
```

Tabela o zmiennym rozmiarze w PL/SQL

```
INSERT INTO STUDENCI VALUES  
(STUDENT_(23451,'Nowakowski','Piotr',OCENY_STUDENTA(3,4,3,5,2,2)));
```

```
INSERT INTO STUDENCI VALUES  
(STUDENT_(34512,'Iksinski','Radek',OCENY_STUDENTA(4,5,4,2,3,3)));
```

```
INSERT INTO STUDENCI VALUES  
(STUDENT_(12345,'Kowalski','Jan',OCENY_STUDENTA(2,3,2,4,5,5)));
```

```
SELECT * FROM STUDENCI;
```

	INDEKS	NAZWISKO	IMIE	OCENY
1	23451	Nowakowski	Piotr	II372.OCENY_STUDENTA(3,4,3,5,2,2)
2	34512	Iksinski	Radek	II372.OCENY_STUDENTA(4,5,4,2,3,3)
3	12345	Kowalski	Jan	II372.OCENY_STUDENTA(2,3,2,4,5,5)

Tabela o zmiennym rozmiarze w PL/SQL

```
UPDATE STUDENCI SET  
OCENY=OCENY_STUDENTA(4,5,4,2,5,5,5,5)  
WHERE NAZWISKO='Iksinski';
```

```
SELECT * FROM STUDENCI;
```

jest

	INDEKS	NAZWISKO	IMIE	OCENY
1	23451	Nowakowski	Piotr	II372.OCENY_STUDENTA(3,4,3,5,2,2)
2	34512	Iksinski	Radek	II372.OCENY_STUDENTA(4,5,4,2,5,5,5,5)
3	12345	Kowalski	Jan	II372.OCENY_STUDENTA(2,3,2,4,5,5)

było

	INDEKS	NAZWISKO	IMIE	OCENY
1	23451	Nowakowski	Piotr	II372.OCENY_STUDENTA(3,4,3,5,2,2)
2	34512	Iksinski	Radek	II372.OCENY_STUDENTA(4,5,4,2,3,3)
3	12345	Kowalski	Jan	II372.OCENY_STUDENTA(2,3,2,4,5,5)

Tabela zagnieżdżona w PL/SQL

```
create or replace type zakupy as  
table of varchar2(50);
```

```
declare  
koszyk zakupy;  
x numeric;  
begin  
koszyk:=zakupy('chleb', 'maslo', 'mleko', 'ser');  
koszyk.delete (2,3);  
for x in koszyk.first()..koszyk.last() loop  
if koszyk.exists(x) then  
dbms_output.put_line(koszyk(x));  
end if;  
end loop;  
dbms_output.put_line('*****');
```

```
chleb  
ser  
*****
```

Tabela zagnieżdżona w PL/SQL

```
create or replace type zakupy as  
table of varchar2(50);
```

```
koszyk.delete(1);  
koszyk.extend(2);  
koszyk(5):='smietana';  
koszyk(6):='kasza';  
x:=koszyk.first();  
while x is not null loop  
  dbms_output.put_line(koszyk(x));  
  x:=koszyk.next(x);  
end loop;  
end;
```

```
ser  
smietana  
kasza
```

Tabela zagnieżdżona w SQL

Jeśli kolekcja jest tabelą zagnieżdżoną to:

- można manipulować pojedynczymi elementami w SQL
- należy wskazać tabelę out-line do przechowywania kolekcji

```
CREATE OR REPLACE TYPE KUPNO AS OBJECT(  
data_kupna date,  
cena numeric(4),  
zawartosc_koszyka zakupy);
```

```
CREATE TABLE ZAKUP OF KUPNO NESTED TABLE  
ZAWARTOSC_KOSZYKA STORE AS TOWAR_W_KOSZYKU;
```

```
INSERT INTO ZAKUP VALUES(KUPNO (TO_DATE('04/05/2017',  
'DD/MM/YYYY'), 50, ZAKUPY('chleb', 'maslo', 'mleko')));
```

```
INSERT INTO ZAKUP VALUES(KUPNO(TO_DATE('04/05/2017',  
'DD/MM/YYYY'), 30, ZAKUPY('kasza', 'ryz' )));
```

Tabela zagnieżdżona w SQL

```
SELECT DATA_KUPNA, CENA, ZAWARTOSC_KOSZYKA  
FROM ZAKUP;
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')

```
INSERT INTO ZAKUP VALUES(KUPNO(TO_DATE('28/04/2017',  
'DD/MM/YYYY'), 30, ZAKUPY('dzem', 'miod' )));
```

```
SELECT VALUE(X) FROM TABLE(  
SELECT ZAWARTOSC_KOSZYKA FROM ZAKUP  
WHERE  
DATA_KUPNA=TO_DATE('28/04/2017','DD/MM/YYYY')) X;
```

	VALUE(X)
1	dzem
2	miod

Operator Table

```
INSERT INTO TABLE(SELECT ZAWARTOSC_KOSZYKA  
FROM ZAKUP WHERE  
DATA_KUPNA=TO_DATE('28/04/2017',  
'DD/MM/YYYY') ) VALUES ('miod');
```

```
SELECT DATA_KUPNA, CENA, ZAWARTOSC_KOSZYKA  
FROM ZAKUP;
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','miod','miod')

```
INSERT INTO TABLE(SELECT ZAWARTOSC_KOSZYKA  
FROM ZAKUP WHERE DATA_KUPNA=TO_DATE('28/04/2017',  
'DD/MM/YYYY') ) VALUES ('ziemniaki');
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','miod','miod','ziemniaki')

Tabela zagnieżdżona w SQL

```
UPDATE TABLE(SELECT ZAWARTOSC_KOSZYKA FROM  
ZAKUP  
WHERE  
DATA_KUPNA=TO_DATE('28/04/2017','DD/MM/YYYY') ) X  
SET VALUE(X)='marchewka' WHERE VALUE(X)='miod';
```

```
SELECT DATA_KUPNA, CENA, ZAWARTOSC_KOSZYKA  
FROM ZAKUP;
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','marchewka','marchewka','ziemniaki')

było

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','miod','miod','ziemniaki')

Tabela zagnieżdżona w SQL

```
DELETE FROM TABLE(SELECT ZAWARTOSC_KOSZYKA  
FROM ZAKUP  
WHERE  
DATA_KUPNA=TO_DATE('28/04/2017','DD/MM/YYYY')) X  
WHERE VALUE(X)='ziemniaki';
```

```
SELECT DATA_KUPNA, CENA, ZAWARTOSC_KOSZYKA  
FROM ZAKUP;
```

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','marchewka','marchewka')

było

	DATA_KUPNA	CENA	ZAWARTOSC_KOSZYKA
1	17/05/04	50	II372.ZAKUPY('chleb','maslo','mleko')
2	17/05/04	30	II372.ZAKUPY('kasza','ryz')
3	17/04/28	30	II372.ZAKUPY('dzem','marchewka','marchewka','ziemniaki')

Tabela zagnieżdżona w SQL

```
SELECT DATA_KUPNA,VALUE(X)
FROM ZAKUP CROSS JOIN
TABLE(ZAWARTOSC_KOSZYKA) X;
```

	DATA_KUPNA	VALUE(X)
2	17/05/04	maslo
3	17/05/04	mleko
4	17/05/04	kasza
5	17/05/04	ryz
6	17/04/28	dzem
7	17/04/28	marchewka
8	17/04/28	marchewka