Własności obiektowo-relacyjne w Oracle9i

Instytut Informatyki, Politechnika Poznańska

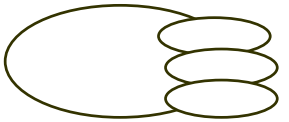
Zalety modelu obiektowego

- Ułatwia modelowanie złożonych obiektów świata rzeczywistego
- Pozwala na składowanie obiektów w bazie danych w postaci, w jakiej są one reprezentowane w nowoczesnych obiektowych aplikacjach (brak konieczności konwersji)
- Pozwala na związanie operacji na danych z danymi
  - Ułatwiona pielęgnacja aplikacji
  - Uniknięcie konieczności implementacji tych samych operacji w różnych aplikacjach
- Umożliwia naturalne reprezentowanie relacji kompozycji („bycia częścią całości”)

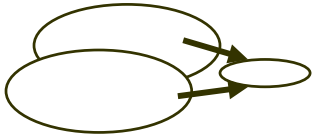
Model obiektowo-relacyjny

- Obsługa typów obiektowych zaimplementowana jako rozszerzenie relacyjnego modelu danych
- Oferuje intuicyjność i bogactwo środków modelu obiektowego
- Zachowuje własności, które przyczyniły się do sukcesu modelu relacyjnego
  - Wysoka współbieżność operacji i przepustowość systemu
    - efektywne mechanizmy transakcyjne
    - blokady na poziomie pojedynczych rekordów
  - Zapytania SQL w schemacie SELECT ... FROM ... WHERE ...
  - Obsługa kopii zapasowych i odtwarzanie po awarii
  - ...

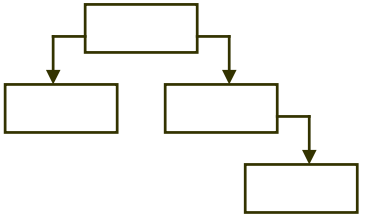
Własności obiektowo-relacyjne w Oracle9i



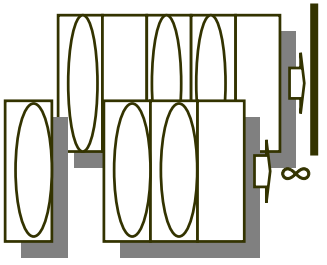
- Obiektowe typy danych użytkownika
  (przeciążanie metod, metody statyczne, konstruktory)



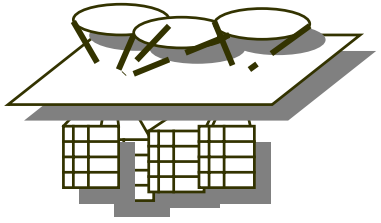
- Współdzielenie obiektów



- Dziedziczenie – z jednego nadtypu
  (przesłanianie metod,
  metody i typy abstrakcyjne, polimorfizm)



- Kolekcje
  (z możliwością zagnieżdżania)



- Perspektywy obiektowe

Typy obiektowe definiowane przez użytkownika

- Typy obiektowe składają się z dwóch elementów:
  - deklaracja – specyfikacja typu:
    - typ bazowy (gdy tworzony podtyp istniejącego typu)
    - atrybuty (publiczne)
    - metody - funkcje i procedury
  - definicja – ciało typu:
    - implementacja metod (domyślnie w PL/SQL)

Utworzenie przykładowego typu *Pracownik*

- Deklaracja typu


```
CREATE TYPE Pracownik AS OBJECT (
        nazwisko VARCHAR2(20),
        pensja NUMBER(6,2),
        etat VARCHAR2(15)
)
/
```

- Definicja - w przypadku typu bez metod jest opcjonalna

Tworzenie obiektów, dostęp do składowych

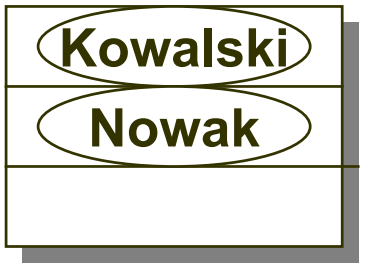
- Do tworzenia obiektów służą specyficzne metody zwane *konstruktorami*
- Automatycznie dostępny jest konstruktor *atrybutowy*, lista jego argumentów jest zgodna z zadeklarowanymi atrybutami typu
- Tworząc obiekt konstruktorem atrybutowym należy podać wartości dla wszystkich atrybutów typu
- Przy wywołaniu konstruktora można poprzedzić jego nazwę słowem NEW (od Oracle 9.2)
- Dostęp do składowych obiektu (atrybutów i metod): operator '.'

```
DECLARE
  prac1 Pracownik;
BEGIN
  prac1 := Pracownik('Kowalski',  1200, 'tokarz');
  /* prac1 := NEW Pracownik('Kowalski',  1200, 'tokarz');  */
  prac1.pensja := 1.1*prac1.pensja;
END;
/
```

Trwałość obiektów

- Obiekty składuje się trwale w bazie danych w dwóch postaciach:
  - obiektów krotkowych (ang. *row object)*,
  - obiektów atrybutowych (ang. column object)
- Obiekty krotkowe składuje się w *tabelach obiektów* (ang. object table)

Pracownicy1



- Obiekty atrybutowe składuje się jako atrybut tabeli

Pracownicy2

ost_mod	prac
30-10-98	Kowalski
3-11-98	Nowak

Obiekty krotkowe

- **Utworzenie tabeli obiektów:**

```
CREATE TABLE pracownicy1 OF Pracownik
/
```

- **Utworzenie obiektu typu *Pracownik* i wstawienie go do tabeli *pracownicy1*:**

```
INSERT INTO pracownicy1 VALUES
        (Pracownik('Kowalski',  1200, 'tokarz'));
COMMIT;
```

Dualny, obiektowo-relacyjny charakter tabel obiektów

- Dostęp obiektowy - funkcja VALUE

```
SELECT VALUE(p) FROM pracownicy1 p;

VALUE(P)(NAZWISKO, PENSJA, ETAT)
----------------------------------------
PRACOWNIK('Kowalski', 1200, 'tokarz')
```

- Dostęp relacyjny

```
SELECT * FROM pracownicy1;

NAZWISKO                       PENSJA ETAT
------------------------------ ---------- ---------------
Kowalski                             1200 tokarz
```

Obiekty atrybutowe

- Utworzenie tabeli z atrybutem typu *Pracownik*:

```
CREATE TABLE pracownicy2 (
        prac Pracownik,
        ost_mod DATE)
/
```

- Utworzenie obiektu typu *Pracownik* i wstawienie go do tabeli *pracownicy2*:

```
INSERT INTO pracownicy2 VALUES
        (Pracownik('Kowalski',  1200, 'tokarz'), SYSDATE);
COMMIT;
```

Dostęp do obiektów atrybutowych

- Dostęp do wartości obiektu atrybutowego

```
SELECT p.prac, p.ost_mod FROM pracownicy2 p;

PRAC(NAZWISKO, PENSJA, ETAT)                OST_MOD
------------------------------------------  ---------
PRACOWNIK('Kowalski', 1200, 'tokarz')       20-NOV-98
```

- Dostęp do składowych obiektu atrybutowego

```
SELECT p.prac.nazwisko, p.ost_mod FROM pracownicy2 p;

PRAC.NAZWISKO          OST_MOD
---------------------- ---------
Kowalski               23-OCT-98
```

Zmienna referencyjna (alias) dla tabeli wymagany przy dostępie do składowych obiektu

Metody typu

- Deklaracja – w specyfikacji typu

```
ALTER TYPE Pracownik REPLACE AS OBJECT(
        nazwisko VARCHAR2(20),
        pensja NUMBER(6,2),
        etat VARCHAR2(15),
        MEMBER FUNCTION jaka_pensja RETURN NUMBER,
        MEMBER PROCEDURE podwyzka (o_ile NUMBER)
)
/
```

- Rodzaje metod:
  - MEMBER – wołane na rzecz konkretnego obiektu
  - STATIC – wołane na rzecz typu
  - MAP / ORDER – wykorzystywane przy porównaniach obiektów i ich sortowaniu
  - CONSTRUCTOR – metody tworzące obiekty danego typu

Metody typu cd.

- Definicja – w ciele typu

```
CREATE OR REPLACE TYPE BODY Pracownik AS

  MEMBER FUNCTION jaka_pensja RETURN NUMBER IS
  BEGIN
    RETURN pensja;
  END;

  MEMBER PROCEDURE podwyzka (o_ile NUMBER) IS
  BEGIN
    pensja:=pensja+o_ile;
  END;

END;
/
```

Aktywowanie metod

- Aktywowanie metod z SQL

```
SELECT p.jaka_pensja() FROM pracownicy1 p;

P.JAKA_PENSJA()
---------------
           1200
```

- Aktywowanie metod z PL/SQL

```
DECLARE
  prac Pracownik;
BEGIN
  prac:=Pracownik('Malinowski',  1000, 'elektryk');
  prac.podwyzka(200);
  INSERT INTO pracownicy1 VALUES (prac); COMMIT;
END;
/
```

Tożsamość obiektów

- Tożsamość posiadają tylko obiekty składowane w tabelach obiektowych
- Do uzyskania referencji do obiektu (OID - Object Identifier) służy funkcja *REF*
- Standardowo OID są generowane przez system i mają rozmiar 16 bajtów, na kolumnie tabeli zawierającej OID automatycznie tworzony i pielęgnowany jest indeks (Oracle daje również możliwość oparcia OID o klucz główny relacji)
- Systemowe identyfikatory OID są lokalnie i globalnie unikalne

```
SELECT COUNT(*)
FROM pracownicy1 p
GROUP BY REF(p)
HAVING COUNT(*)>1;


no rows selected
```

Wartość obiektu

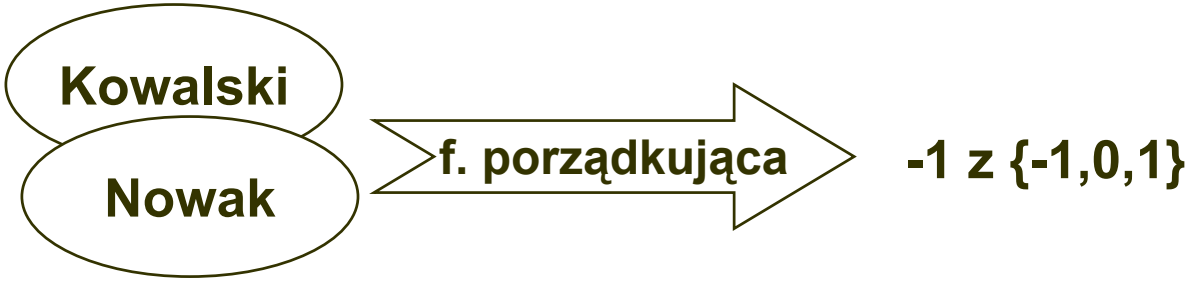
- Wartością obiektu jest zbiór wartości jego składowych
- Wartości dwóch obiektów są równe, gdy wartości wszystkich składowych tych obiektów są równe
- Wartości dwóch obiektów są różne, gdy wartości co najmniej jednej składowej tych obiektów są różne
- Standardowe operatory = i <> umożliwiają rozróżnianie wartości obiektów

```
SELECT p.nazwisko, q.prac.nazwisko
FROM pracownicy1 p, pracownicy2 q
WHERE VALUE(p)=q.prac;

NAZWISKO             PRAC.NAZWISKO
-------------------- --------------------
Kowalski             Kowalski
```

Porównywanie wartości obiektów

- W celu porównywania wartości obiektów za pomocą operatorów: <, >, <=, >=, between ... and ..., lub w celu użycia wartości obiektów w klauzulach ORDER BY, GROUP BY, DISTINCT można zdefiniować metodę:
  - odwzorowującą:

  - porządkującą:



Metoda odwzorowująca

- Deklaracja metody odwzorowującej *odwzoruj*

```
ALTER TYPE Pracownik REPLACE AS OBJECT(
...
  MAP MEMBER FUNCTION odwzoruj RETURN VARCHAR2
)
/
```

- Definicja metody odwzorowującej *odwzoruj*

```
CREATE OR REPLACE TYPE BODY Pracownik AS
...
  MAP MEMBER FUNCTION odwzoruj RETURN VARCHAR2 IS
  BEGIN
    RETURN nazwisko;
  END;
END;
/
```

Wykorzystanie metody odwzorowującej

- Jawne użycie metody odwzorowującej

```
SELECT p.nazwisko, q.prac.nazwisko
FROM pracownicy1 p, pracownicy2 q
WHERE p.odwzoruj()> q.prac.odwzoruj()
        AND p.nazwisko='Malinowski';

NAZWISKO                  PRAC.NAZWISKO
------------------------- -------------
Malinowski                Kowalski
```

- Niejawne użycie metody porządkującej lub odwzorowującej

```
SELECT p.nazwisko, q.prac.nazwisko
FROM pracownicy1 p, pracownicy2 q
WHERE VALUE(p)>q.prac;
```

Metoda porządkująca

- Deklaracja metody porządkującej *porządek*

```
ALTER TYPE Pracownik REPLACE AS OBJECT(
...
  ORDER MEMBER FUNCTION porzadek(p Pracownik) RETURN NUMBER
)
/
```

- Definicja metody porządkującej *porządek*

```
CREATE OR REPLACE TYPE BODY Pracownik AS
...
  ORDER MEMBER FUNCTION porzadek(p Pracownik) RETURN NUMBER IS
  BEGIN
    IF p.etat=etat THEN RETURN 0;
    ELSIF p.etat='tokarz' AND etat='elektryk' THEN RETURN -1;
    ELSE RETURN 1; END IF;
  END;
END;
```

Wykorzystanie metody porządkującej

- Jawne użycie metody

```
SELECT p.nazwisko, q.prac.nazwisko
FROM pracownicy1 p, pracownicy2 q
WHERE q.porządkuj(p)=-1 AND p.nazwisko='Nowak';
```

- Niejawne użycie metody porządkującej lub odwzorowującej

```
SELECT q.prac.nazwisko
FROM pracownicy1 p, pracownicy2 q
WHERE VALUE(p)>q;
```

Wykorzystanie metod porównujących wartości obiektów

- Wykorzystanie metody odwzorowującej jest w pewnych przypadkach bardziej efektywne, np.

```
SELECT q.prac.nazwisko
FROM pracownicy2 q
ORDER BY q.prac;

PRAC.NAZWISKO
-------------
Kowalski
```

- Metody porządkujące są bardziej elastyczne, umożliwiają bardziej zaawansowane porównywanie wartości

Konstruktory definiowane przez użytkownika

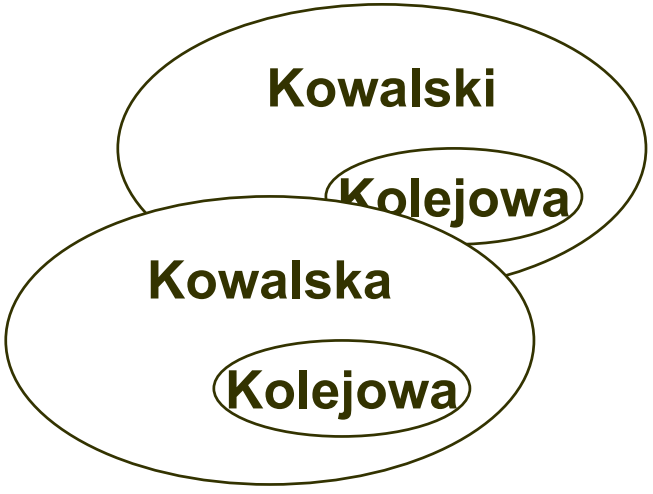
- Dostępne od wersji Oracle 9.2
- Nazwa taka jak nazwa typu, mogą być przeciążane
- Ułatwiają ewolucję typu (działają mimo np. dodania atrybutu)
- Domyślny konstruktor atrybutowy jest dostępny pod warunkiem, że nie zostanie zastąpiony (przykryty) konstruktorem użytkownika o tej samej sygnaturze

```
ALTER TYPE Pracownik REPLACE AS OBJECT(
  nazwisko VARCHAR2(20),
  pensja NUMBER(6,2),
  etat VARCHAR2(15),
  MEMBER FUNCTION jaka_pensja RETURN NUMBER,
  MEMBER PROCEDURE podwyzka (o_ile NUMBER),
  CONSTRUCTOR FUNCTION Pracownik (nazwisko VARCHAR2)
    RETURN SELF AS RESULT
)
/
```

Konstruktory definiowane przez użytkownika cd.

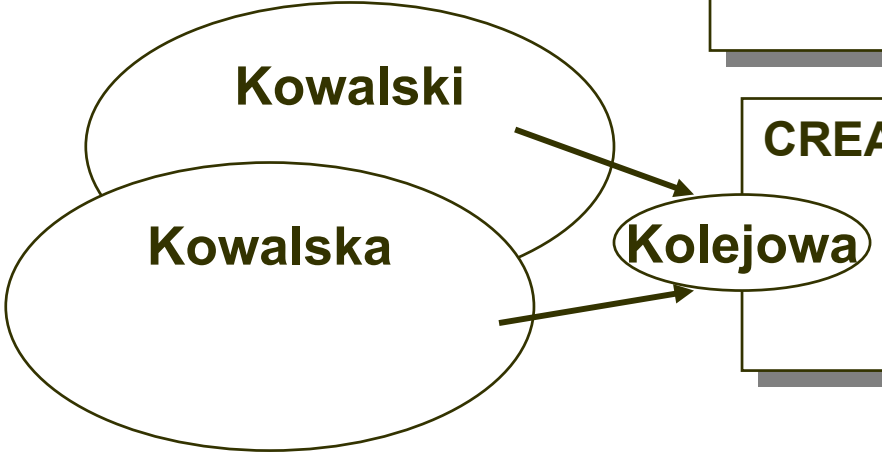
```
CREATE OR REPLACE TYPE BODY Pracownik AS

  MEMBER FUNCTION jaka_pensja RETURN NUMBER IS
  BEGIN  RETURN pensja;  END;

  MEMBER PROCEDURE podwyzka (o_ile NUMBER) IS
  BEGIN   pensja:=pensja+o_ile;  END;

  CONSTRUCTOR FUNCTION Pracownik (nazwisko VARCHAR2)
     RETURN SELF AS RESULT IS
  BEGIN
     SELF.nazwisko := nazwisko;
     /* pozostałe atrybuty ustawiane przez system na NULL */
     RETURN;
  END;
END;
/
```

Współdzielenie obiektów vs. zagnieżdżanie obiektów




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

```
CREATE TYPE Osoba AS OBJECT(
        nazwisko VARCHAR2(20),
        imie     VARCHAR2(15),
        adr Adres);
```




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

Wiązanie obiektów współdzielonych

- Utworzenie tabel obiektów

```
CREATE TABLE osoby OF Osoba;
CREATE TABLE adresy OF Adres;
```

- Utworzenie obiektów i ich powiązanie

```
DECLARE
  a REF Adres;
BEGIN
  INSERT INTO adresy adr VALUES(Adres('Kolejowa',1,5))
              RETURNING REF(adr) INTO a;
  INSERT INTO osoby VALUES(Osoba('Kowalski','Jan',a));
  INSERT INTO osoby VALUES(Osoba('Kowalska','Ala',a));
  COMMIT;
END;
/
```

Nawigowanie po referencjach obiektów

- Jawna referencja do obiektu

```
SELECT o.nazwisko, o.imie, DEREF(o.adr)
FROM osoby o;

NAZWISKO             IMIE       DEREF(O.ADR)(ULICA, ...)
-------------------- ---------- ------------------------
Kowalski             Jan        ADRES('Kolejowa', 1, 5)
Kowalska             Ala        ADRES('Kolejowa', 1, 5)
```

- Niejawna referencja do obiektu

```
SELECT o.nazwisko, o.imie, o.adr.ulica
FROM osoby o;

NAZWISKO             IMIE           ADR.ULICA
-------------------- -------------- ----------
Kowalski             Jan            Kolejowa
Kowalska             Ala            Kolejowa
```

Łączenie obiektów

```
SELECT o.nazwisko, o.imie, a.ulica
FROM osoby o, adresy a
WHERE o.adr=REF(a);

NAZWISKO             IMIE            ULICA
-------------------- --------------- ---------
Kowalski             Jan             Kolejowa
Kowalska             Ala             Kolejowa
```

Puste referencje

- Wstawienie pustej referencji

```
INSERT INTO osoby VALUES(Osoba('Nowak','Jerzy',NULL));
```

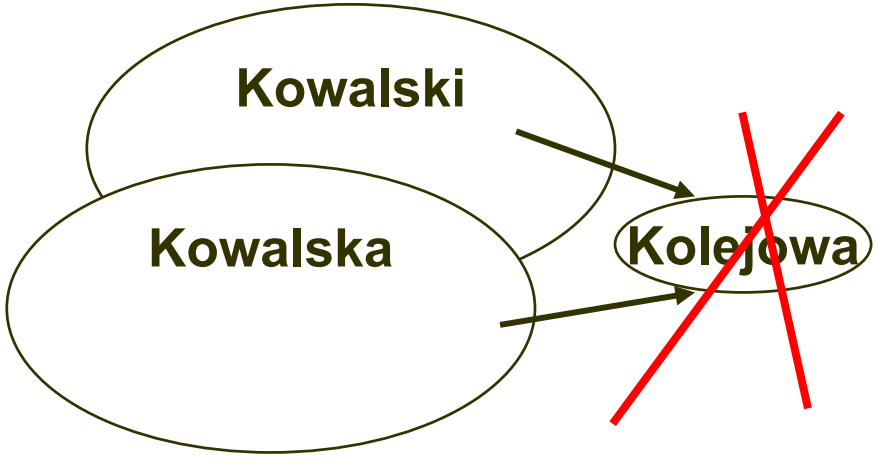
- Testowanie pustej referencji

```
SELECT o.nazwisko, o.imie FROM osoby o
WHERE adr IS NULL;

NAZWISKO                  IMIE
------------------------- -------------
Nowak                     Jerzy
```

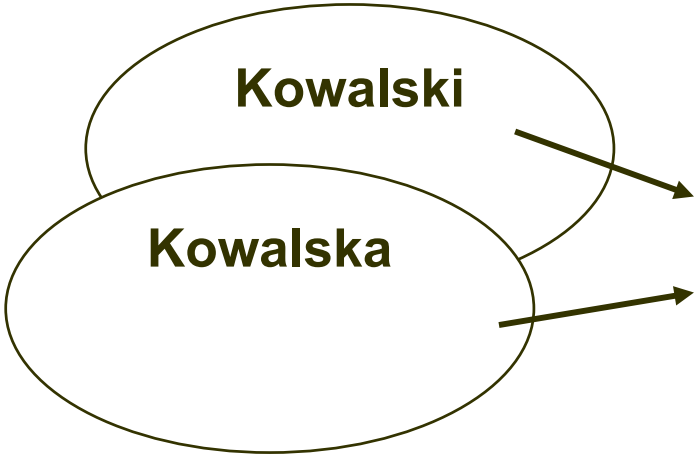
- Odwoływanie się do pustej referencji

```
SELECT o.adr.ulica FROM osoby o WHERE adr IS NULL;

ADR.ULICA
---------------
```

Wiszące referencje

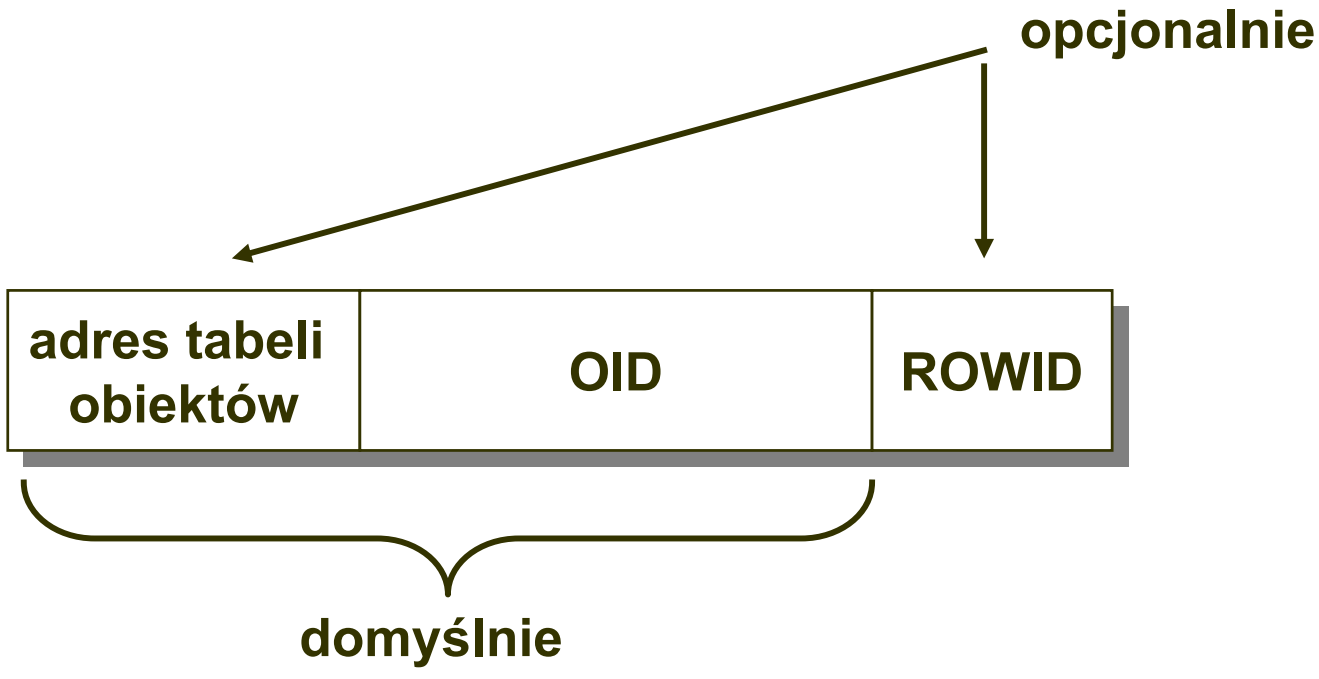



```
DELETE FROM adresy;
```




```
SELECT o.nazwisko, o.imie
FROM osoby o
WHERE adr IS DANGLING;

NAZWISKO                  IMIE
------------------------- ------
Kowalski                  Jan
Kowalska                  Ala
```

Fizyczna postać trwałych referencji



- OID - unikalny identyfikator obiektu,
- ROWID- logiczny adres dyskowy krotki

- Obecność ROWID poprawia efektywność dostępu do obiektu poprzez referencję (nie jest konieczne skorzystanie z indeksu na OID)

Zmiana fizycznej postaci trwałych referencji

- Dodanie ROWID do referencji

```
ALTER TABLE osoby
ADD (REF (adr) WITH ROWID)
/
```

- Zawężenie zakresu referencji

```
ALTER TABLE osoby
ADD (SCOPE FOR (adr) IS adresy)
/
```

UWAGA! Aby zmienić fizyczną postać referencji „adr” tabela „osoby” powinna być pusta.

Ograniczenia integralnościowe i obiekty

- Definicja ograniczeń integralnościowych możliwe jest tylko dla obiektów składowanych w tabelach obiektów
- Przykładowe ograniczenia dla obiektów typu *Osoba* składowanych w tabeli *osoby*:

```
ALTER TABLE osoby ADD
 (CONSTRAINT osoby_pk PRIMARY KEY (nazwisko,imie))
/
```

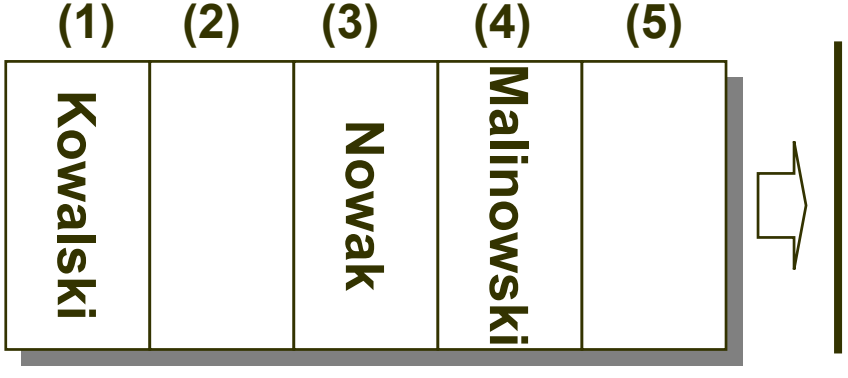
```
ALTER TABLE osoby ADD
(CONSTRAINT pensja_min CHECK(pensja>300))
/
```

```
ALTER TABLE osoby ADD
(CONSTRAINT nie_pusta_pensja pensja NOT NULL)
/
```

Kolekcje

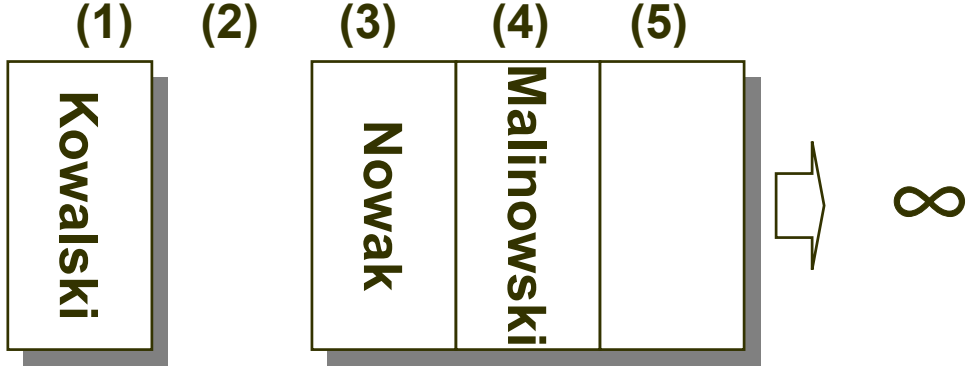
- Tablice o zmiennej długości (ang. varray)

```
...
kolekcja_prac(2):=
  Osoba('Nowak','Jan',null);
...
```



ograniczony zakres

- Zagnieżdżone tabele (ang. nested table)



Porównanie tabel o zmiennej długości i zagnieżdżonych tabel

własność	varray	nested table
Maksymalny rozmiar	TAK	NIE
Usuwanie elementów ze środka kolekcji	NIE	TAK
Składowanie w bazie danych	IN-LINE	OUT-LINE
Zachowanie fizycznego porządku	TAK	NIE
Manipulowanie na pojedynczych elementach w SQL	NIE	TAK

Deklaracja kolekcji

- Deklaracja PL/SQL typu tablicy o zmiennej długości

```
DECLARE
  TYPE Telefony IS VARRAY(5) OF VARCHAR2(20);
  tel Telefony;
...
```

- Utworzenie trwałego typu tablicy o zmiennej długości

```
CREATE TYPE Telefony AS VARRAY(5) OF VARCHAR2(20)
/
```

```
DECLARE
  tel Telefony;
...
```

Deklaracja kolekcji cd.

- Deklaracja PL/SQL typu zagnieżdżonej tabeli

```
DECLARE
TYPE Samochod AS OBJECT
(marka VARCHAR2(20), kolor VARCHAR2(10), nr_rej VARCHAR2(7));
TYPE Samochody AS TABLE OF Samochod;
auta Samochody;
...
```

- Utworzenie trwałego typu zagnieżdżonej tabeli

```
CREATE TYPE Samochod AS OBJECT
(marka VARCHAR2(20), kolor VARCHAR2(10), nr_rej VARCHAR2(7))
/
CREATE TYPE Samochody AS TABLE OF Samochod
/
```

```
DECLARE auta Samochody;
...
```

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

Tablice o zmiennej długości w PL/SQL

```
DECLARE
  tel Telefony:=Telefony();  -- deklaracja i inicjacja
  i INTEGER;
BEGIN
  tel.extend();              -- rozszerzenie kolekcji
  tel(1):='299909';          -- podstawienie wartości
  tel.extend(tel.limit()-tel.count(),1);
                    --rozszerzenie i skopiowanie
  FOR i IN 1..tel.last() LOOP
      dbms_output.put_line(tel(i));  --wyświetlenie
  END LOOP;
  tel.trim(1);               --przycięcie kolekcji
  tel(2):='246235';          --zmiana wartości
  tel.delete();              --usunięcie elementów kolekcji
END;
/
```

Zagnieżdżona tabela w PL/SQL

```
DECLARE auta Samochody:=Samochody();
i INTEGER;
BEGIN
auta.extend(1);
auta(1):=Samochod('Renault','biały', 'WAW1256');
auta.extend(5,1);
auta.delete(2,3);
 FOR i IN 1..auta.last() LOOP
   IF auta.exists(i) THEN
     dbms_output.put_line(auta(i).marka);  --wyświetlenie
   END IF;
  END LOOP;
  i:=auta.first();
  WHILE (i IS NOT NULL) LOOP
      auta(i).kolor:='zielony'; i:=auta.next(i);
  END LOOP;
END;
/
```

Przykładowa tablica o zmiennej długości w SQL

- Utworzenie typu *Adres_tel* wykorzystującego kolekcję „telefony”:

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

- Utworzenie tabeli obiektów z obiektami typu *Adres_tel*:

```
CREATE TABLE adresy_tel OF Adres_tel;
```

Operacje na tablicy o zmiennej długości w SQL

- Wstawienie tablicy o zmiennej długości

```
INSERT INTO adresy_tel VALUES (Adres_tel('Długa', 4,1,
  Telefony('8367227','8383821')));
```

- Odczytanie wartości tablicy o zmiennej długości

```
SELECT a.tel FROM adresy_tel a;

TELEFONY
------------------------------
TELEFONY('8367227', '8383821')
```

- Modyfikacja tablicy o zmiennej długości

```
UPDATE adresy_tel a SET tel = Telefony('8367227')
WHERE a.ulica = 'Długa';
```

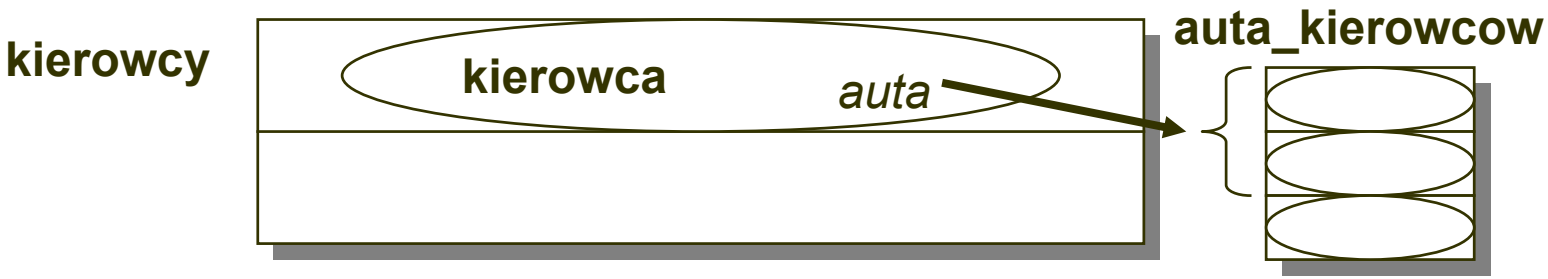
Przykładowa zagnieżdżona tabela w SQL

- Utworzenie typu *Kierowca* wykorzystującego kolekcję *Samochody*:

```
CREATE TYPE Kierowca AS OBJECT
(nazwisko VARCHAR2(20), wiek NUMBER, auta Samochody)
/
```

- Utworzenie tabeli obiektów z obiektami typu *kierowca*:

```
CREATE TABLE kierowcy OF Kierowca
        NESTED TABLE auta STORE AS auta_kierowcow;
```



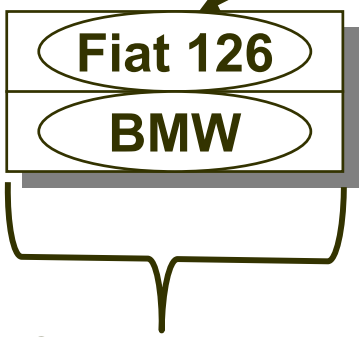
Operacje na zagnieżdżonej tabeli w SQL

- Wstawienie tablicy o zmiennej długości:

```
INSERT INTO kierowcy VALUES
(
    Kierowca('Ziutkiewicz', 25,
          Samochody(
            Samochod('Fiat 126', 'czerwony', 'PWG0978'),
            Samochod('VW', 'zielony', 'PWZ2345')
                          )
                )
);
```

„Rozpłaszczanie” zagnieżdżonych tabel - operator TABLE (THE w Oracle8)


```
SELECT auta FROM kierowcy
  WHERE nazwisko='Ziutkiewicz'
```




```
SELECT a.marka, a.kolor
FROM TABLE(.....) a
```

```
SELECT a.marka, a.kolor
 FROM TABLE (SELECT auta FROM kierowcy
    WHERE nazwisko='Ziutkiewicz') a;

MARKA                KOLOR
-------------------- ----------
Fiat 126             czerwony
BMW                  niebieski
```

„Rozpłaszczone” tabele w instrukcjach DML

- Wstawienie do rozpłaszczonej tabeli:

```
INSERT INTO
        TABLE (SELECT auta FROM kierowcy
              WHERE nazwisko='Ziutkiewicz')
        VALUES(Samochod('BMW', 'niebieski', 'WZZ9876'));
```

- Modyfikacja rozpłaszczonej tabeli:

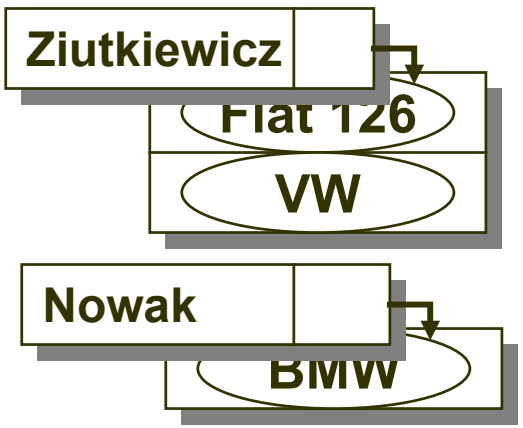
```
UPDATE TABLE (SELECT auta FROM kierowcy
          WHERE nazwisko='Ziutkiewicz') a
SET a.nr_rej='XYZ4570'
WHERE a.nr_rej='PWG0978';
```

- Usuwanie elementów z rozpłaszczonej tabeli:

```
DELETE TABLE (SELECT auta FROM kierowcy
            WHERE nazwisko='Ziutkiewicz') a
WHERE a.nr_rej='PWZ2345';
```

Korelowanie zagnieżdżonych tabel

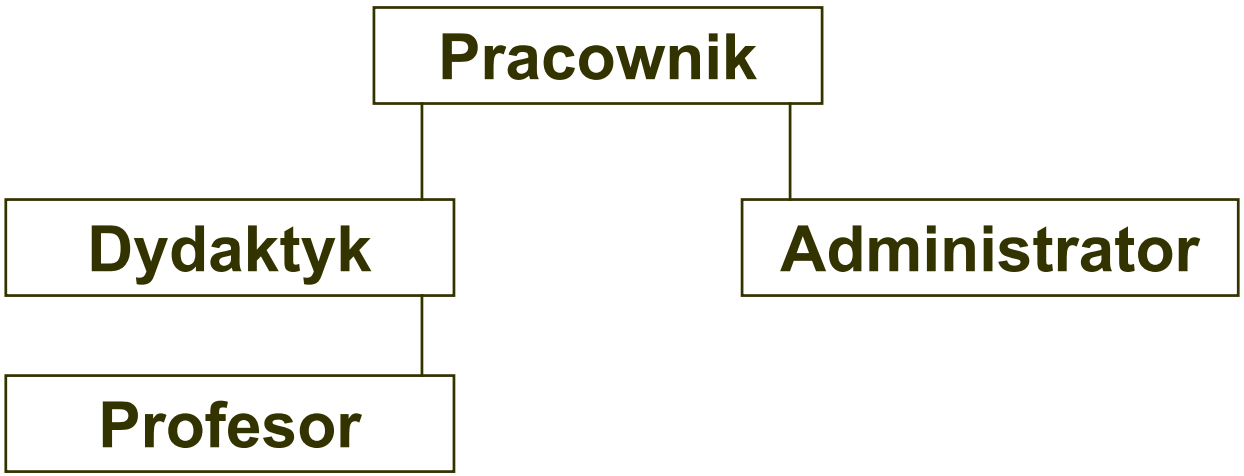
SELECT kr.nazwisko, CURSOR(...) FROM kierowcy
WHERE nazwisko='Ziutkiewicz';




```
SELECT kr.nazwisko,
     CURSOR ( SELECT a.marka, a.nr_rej
                FROM TABLE(kr.auta) a)
FROM kierowcy kr;

NAZWISKO             CURSOR(SELECTA.MARKA
-------------------- --------------------
Ziutkiewicz          CURSOR STATEMENT : 2
CURSOR STATEMENT : 2
MARKA                NR_REJ
-------------------- -------
Fiat 126             XYZ4570
VW                   PWZ2345
```

Dziedziczenie, hierarchie typów

- Nowy typ można utworzyć jako podtyp (specjalizację) istniejącego (jednego!) typu
- Podtyp dziedziczy z typu bazowego atrybuty oraz metody MEMBER i STATIC
- W podtypie można dodać nowe atrybuty i/lub metody oraz przesłaniać odziedziczone metody
- Typ może służyć jako bazowy dla nowych typów jeśli jest NOT FINAL (domyślnie FINAL!)
- Metody porządkujące mogą występować tylko w typie będącym korzeniem hierarchii



Dziedziczenie, hierarchie typów cd.

```
ALTER TYPE Pracownik NOT FINAL CASCADE;
```

Jeśli istnieją tabele obiektów typu

```
CREATE TYPE Dydaktyk UNDER Pracownik (
  dodatek_funkcyjny NUMBER(6,2),
  OVERRIDING MEMBER FUNCTION jaka_pensja RETURN NUMBER
);
```

```
CREATE TYPE BODY Dydaktyk AS
 OVERRIDING MEMBER FUNCTION jaka_pensja RETURN NUMBER IS
  BEGIN
    RETURN pensja + NVL(dodatek_funkcyjny, 0);
  END;
END;
/
```

Dziedziczenie, przeciążanie i przesłanianie metod

- Przeciążanie metod polega na dostępności w typie kilku metod o tej samej nazwie (zdefiniowanych w danym typie lub odziedziczonych) różniących się zestawem parametrów
- Przesłonięcie metody polega na redefinicji w podtypie implementacji metody odziedziczonej z nadtypu
  - niedozwolone dla metod zadeklarowanych jako FINAL
  - wymaga słowa kluczowego OVERRIDING w specyfikacji i ciele podtypu

Dziedziczenie, przeciążanie i przesłanianie metod cd

```
ALTER TYPE Dydaktyk
 ADD MEMBER PROCEDURE podwyzka(oilep NUMBER,oiled NUMBER)
```

```
CREATE OR REPLACE TYPE BODY Dydaktyk AS
  OVERRIDING MEMBER FUNCTION jaka_pensja
                    RETURN NUMBER IS
  BEGIN  RETURN pensja + NVL(dodatek_funkcyjny, 0);   END;
  MEMBER PROCEDURE podwyzka(oilep NUMBER,oiled NUMBER) IS
  BEGIN pensja := pensja + oilep;
          dodatek_funkcyjny := dodatek_funkcyjny + oiled;  END;
END;
```

```
DECLARE  p Dydaktyk;
BEGIN
  p := Dydaktyk('Kaczmarek',  1900, 'adiunkt', 300);
  p.podwyzka(100);       /* podwyzka(o_ile NUMBER) */
  p.podwyzka(100, 50); /* podwyzka(oilep NUMBER,oiled NUMBER) */
END;
```

Typy i metody abstrakcyjne

- Dany typ NOT FINAL można zadeklarować jako NOT INSTANTIABLE uniemożliwiając tworzenie obiektów typu
- Typy NOT INSTANTIABLE mogą mieć metody abstrakcyjne (NOT INSTANTIABLE - bez implementacji)
- Jeśli typ zawiera jakąś metodę abstrakcyjną, musi być zadeklarowany jako NOT INSTANTIABLE

```
CREATE TYPE Figura AS OBJECT (
  kolor VARCHAR2(20),
  NOT INSTANTIABLE MEMBER FUNCTION pole RETURN NUMBER
) NOT INSTANTIABLE NOT FINAL;
```

```
CREATE TYPE Kwadrat UNDER Figura (
  bok NUMBER,
  OVERRIDING MEMBER FUNCTION pole RETURN NUMBER );
```

```
CREATE TYPE BODY Kwadrat AS
  OVERRIDING MEMBER FUNCTION pole RETURN NUMBER IS
  BEGIN RETURN bok*bok; END;
END;
```

Polimorfizm

- Generalnie, w miejscu obiektu danego typu może wystąpić obiekt jego podtypu (choć można uniemożliwić lub ograniczyć zastępowanie typów)
- W przypadku operacji przypisania dopuszczalne są sytuacje:
  - Źródło i cel przypisania tego samego typu
  - Typ źródła będący specjalizacją typu celu
  - Typ źródła będący generalizacją typu celu, poprzez jawne użycie funkcji TREAT i pod warunkiem, że źródło w rzeczywistości zawiera obiekt typu celu lub jego podtypu

```
DECLARE
  p Pracownik; q Pracownik; d Dydaktyk;
BEGIN
  p := Pracownik('Kowalski',  1200, 'portier');
  q := Dydaktyk('Nowak',  1300, 'asystent', 100);
  d := Dydaktyk('Kaczmarek',  1900, 'adiunkt', 300);
  ... ???
END;
```

p := q;

q := p;

p := d;

d := p;

d := q;

d := TREAT (q as Dydaktyk);

Dynamiczne wiązanie metod

- Poszczególne typy w hierarchii mogą w różny sposób implementować te same metody
- Decyzja którą implementację danej metody wybrać jest podejmowana w trakcie działania programu, a nie w czasie kompilacji (metody są *wirtualne*)

```
CREATE TABLE pracownicy_uczelni OF Pracownik;
```

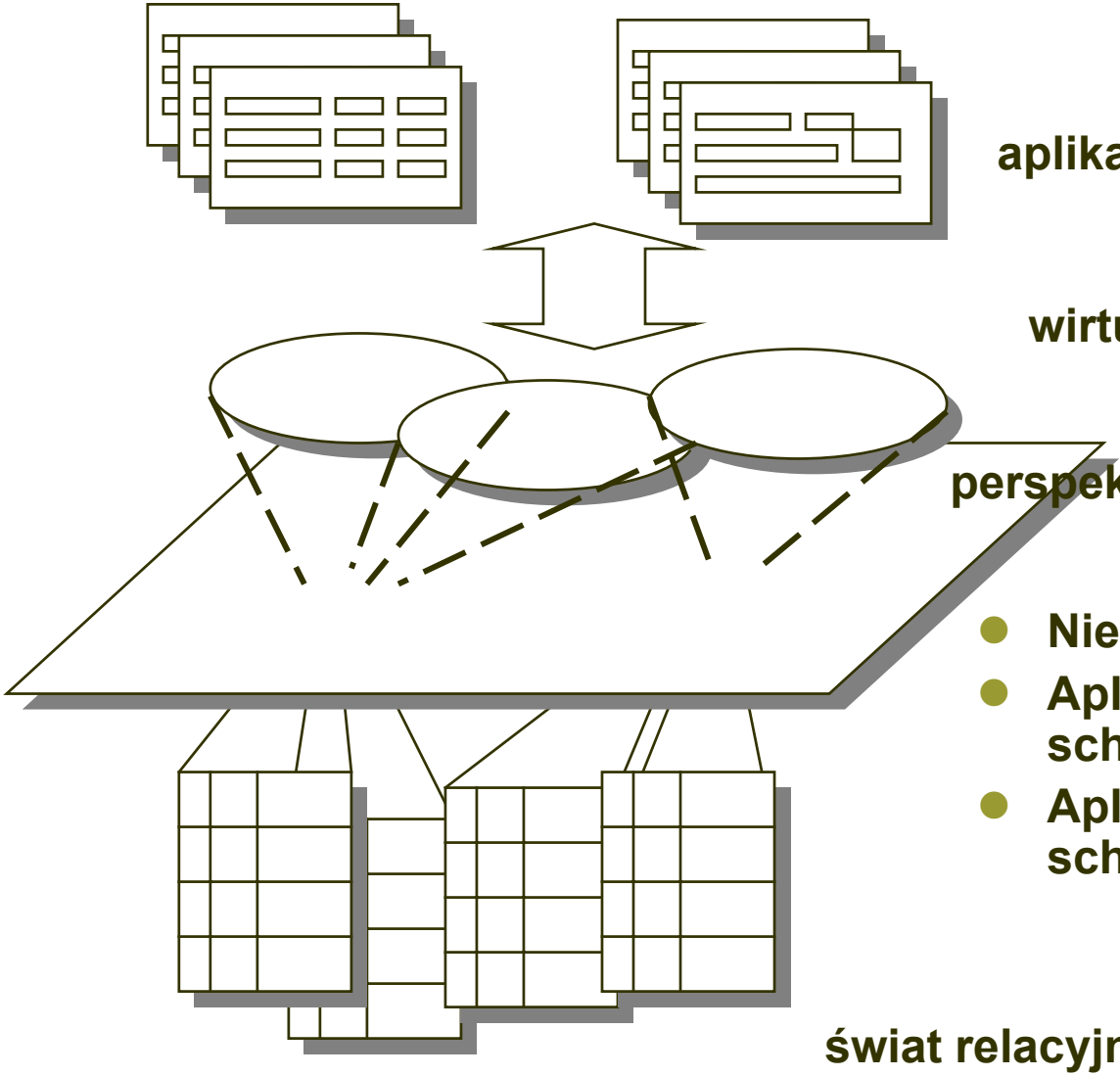
```
INSERT INTO pracownicy_uczelni VALUES
        (Pracownik('Kowalski',  1200, 'portier'));
INSERT INTO pracownicy_uczelni VALUES
        (Dydaktyk('Nowak',  1800, 'adiunkt', 300));
```

```
SELECT p.nazwisko, p.jaka_pensja()
FROM pracownicy_uczelni p;

P.NAZWISKO      P.JAKA_PENSJA()
--------------- ---------------
Kowalski                  1200
Nowak                     2100
```

Pracownik.jaka_pensja()

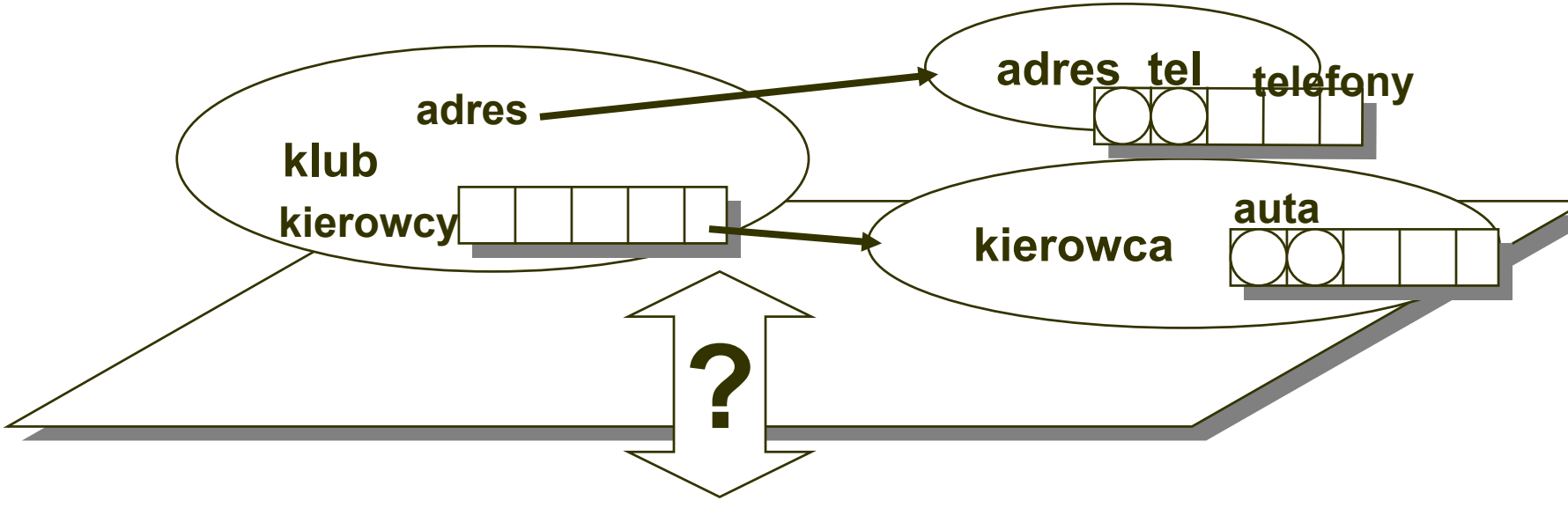
Dydaktyk.jaka_pensja()

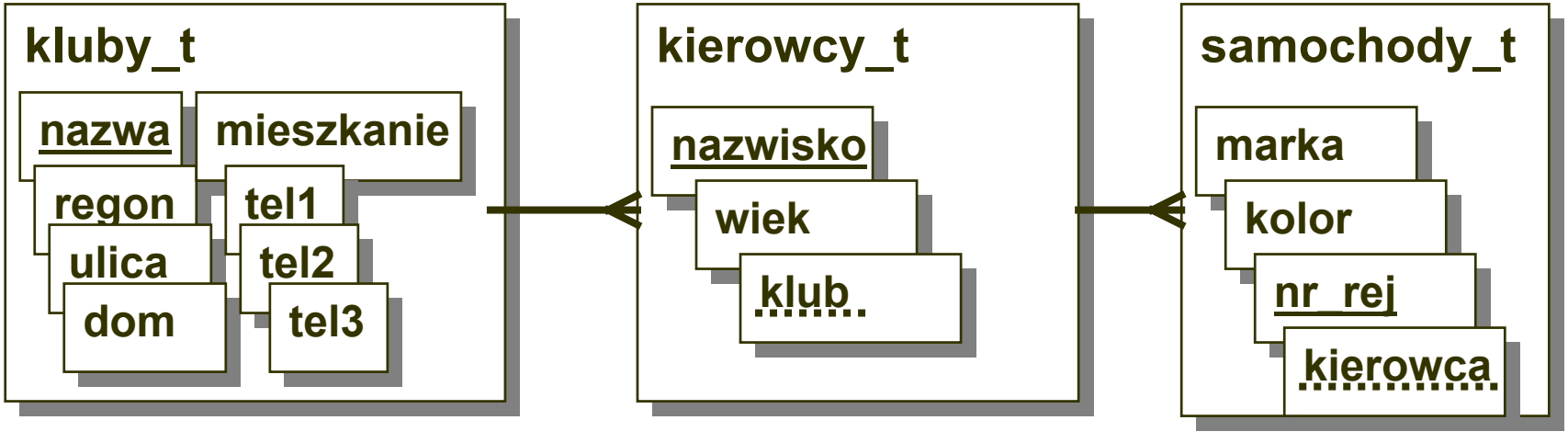
Perspektywy obiektowe



- Niezależność logiczna
- Aplikacje obiektowe na schemacie relacyjnym
- Aplikacje relacyjne na schemacie obiektowym

Perspektywy obiektowe - przykład





Przykład - świat relacyjny

```
CREATE TABLE kluby_t
(nazwa VARCHAR2(20) PRIMARY KEY, regon VARCHAR2(10),
  ulica VARCHAR2(15),dom NUMBER(4),mieszkanie NUMBER(3),
  tel1 VARCHAR2(20), tel2 VARCHAR2(20), tel3 VARCHAR2(20))
/

CREATE TABLE kierowcy_t
(nazwisko VARCHAR2(20) PRIMARY KEY, wiek NUMBER,
  nazwa_klubu VARCHAR2(20) REFERENCES kluby_t)
/

CREATE TABLE samochody_t
(marka VARCHAR2(20), kolor VARCHAR2(10),
  nr_rej VARCHAR2(7) PRIMARY KEY,
 naz_kier VARCHAR2(20) REFERENCES kierowcy_t)
/
```

Przykład - świat relacyjny cd.

```
INSERT INTO kluby_t VALUES
      ('Star','2345894', 'Wioślarska', 2,1,'345345','234564',NULL);

INSERT INTO kierowcy_t VALUES
      ('Piwowski', 40, 'Star');

INSERT INTO kierowcy_t VALUES
      ('Nowaczyński', 50, 'Star');

INSERT INTO samochody_t VALUES
      ('Fiat126p','czerwony','PWG2384','Nowaczyński');

INSERT INTO samochody_t VALUES
     ('VW','granatowy','PWY9033','Nowaczyński');
```

Przykład - świat obiektowy

```
CREATE TYPE Kierowcy_zt AS TABLE OF REF Kierowca
/
CREATE OR REPLACE TYPE Klub as object
( nazwa VARCHAR2(20), regon VARCHAR2(10),
adr_tel REF Adres_tel, kier Kierowcy_zt,
MEMBER FUNCTION ilu_czlonkow RETURN NUMBER,
PRAGMA RESTRICT_REFERENCES
              (ilu_czlonkow , WNDS, RNPS, WNPS)
)
/
CREATE OR REPLACE TYPE BODY Klub AS
  MEMBER FUNCTION ilu_czlonkow RETURN NUMBER IS
  BEGIN
    RETURN kier.count();
   END;
END;
/
```

Przykład - perspektywy obiektowe

```
CREATE OR REPLACE VIEW adresy_v
      OF Adres_tel WITH OBJECT OID(ulica,dom,mieszkanie)
AS SELECT ulica,dom,mieszkanie, Telefony(tel1,tel2,tel3)
      FROM kluby_t
/

CREATE OR REPLACE VIEW kierowcy_v
          OF Kierowca WITH OBJECT OID(nazwisko)
AS SELECT nazwisko, wiek,
    CAST (MULTISET (SELECT samochod (marka, kolor, nr_rej)
                           FROM samochody_t
                           WHERE naz_kier=k.nazwisko
                  ) AS samochody )
FROM kierowcy_t k
/
```

Przykład - perspektywy obiektowe cd.

```
CREATE OR REPLACE VIEW kluby_v
  OF Klub WITH OBJECT OID(nazwa)
  AS SELECT nazwa, regon,
       MAKE_REF(adresy_v,ulica,dom,mieszkanie),
       CAST (MULTISET (SELECT MAKE_REF(kierowcy_v, nazwisko)
                              FROM kierowcy_t
                              WHERE nazwa_klubu=k.nazwa
                     ) AS  kierowcy_zt)
FROM kluby_t k
/
```

Przeglądanie danych przez perspektywy obiektowe

```
SELECT a.marka, a.kolor
 FROM TABLE (SELECT auta FROM kierowcy_v
    WHERE nazwisko='Nowaczyński') a;

SELECT kr.nazwisko,
          CURSOR ( SELECT a.marka, a.nr_rej
                    FROM TABLE(kr.auta) a
          )
FROM kierowcy_v kr;

SELECT k.nazwa, k.ilu_czlonkow()
FROM kluby_v k;
```

Wyzwalacz INSTEAD-OF

```
CREATE OR REPLACE TRIGGER kierowcy_v_insert
 INSTEAD OF INSERT ON kierowcy_v
    FOR EACH ROW
DECLARE
a Samochody;
BEGIN
  a:=:NEW.auta;
  INSERT INTO kierowcy_t(nazwisko,wiek)
            VALUES (:NEW.nazwisko, :NEW.wiek);
  FOR i IN a.first .. a.last LOOP
          INSERT INTO samochody_t
          VALUES (a(i).marka, a(i).kolor,
                    a(i).nr_rej, :NEW.nazwisko);
  END LOOP;
END;
/
```

- Zawsze FOR EACH ROW
- Bez klauzuli WHEN

Modyfikowanie danych przez perspektywy obiektowe

```
INSERT INTO kierowcy_v
        VALUES (Kierowca('Ziutkiewicz', 25, Samochody(
                Samochod('Fiat 126', 'czerwony', 'PWG0978'),
                Samochod('VW', 'zielony', 'PWZ2345'))));

SELECT kr.nazwisko,
        CURSOR ( SELECT a.marka, a.nr_rej
                FROM TABLE(kr.auta) a
        )
FROM kierowcy_v kr;


SELECT nazwisko, nr_rej
FROM kierowcy_t,samochody_t
WHERE nazwisko=naz_kier;
```