Rozszerzenie obiektowe w SZBD Oracle

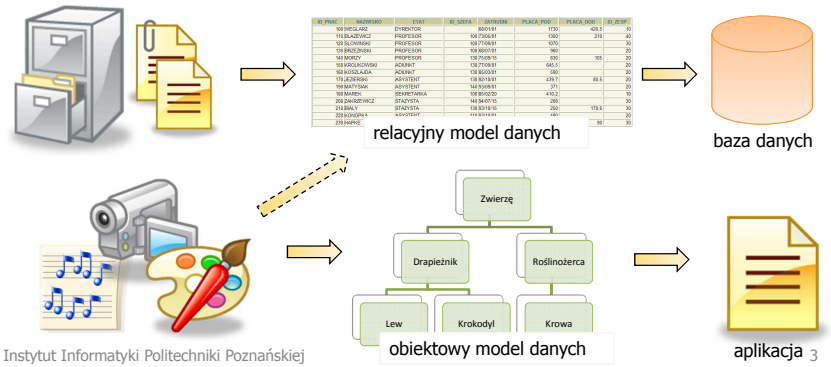
Cześć 1. Deklarowanie typów. Tworzenie instancji typów. Przegląd metod. Dziedziczenie. Polimorfizm.

Plan

- Wprowadzenie
  - obiektowy model danych
- Obiekty w bazie danych
  - definiowanie i przechowywanie
  - składowe i metody
  - konstruktory
  - referencje
  - dziedziczenie i polimorfizm

Model danych

- Model danych to abstrakcyjny opis sposobu **reprezentacji** i **wykorzystania** danych
- Na model danych składają się:
  - **struktura**: kolekcja struktur wykorzystywanych do przedstawiania obiektów i encji modelowanego mini-świata,
  - **ograniczenia**: reguły zapewniające spójność i poprawność danych
  - **operacje**: zbiór operatorów stosowalnych do struktur w celu czytania i pisania danych



Obiektowy model danych - pojęcia

- **Klasa** – abstrakcyjna reprezentacja "rzeczy", zawiera definicję struktury i zachowania w postaci składowych i metod.
- **Obiekt** – wystąpienie klasy, reprezentacja konkretnego bytu.
- **Składowa** – reprezentuje cechę obiektu.
- **Metoda** – zdolność obiektu do wykonania czynności.
- **Przekazywanie komunikatów** – sposób przekazywania danych lub wywoływania czynności między obiektami.
- **Dziedziczenie** – uproszczony sposób specjalizacji klas.
- **Enkapsulacja** – ukrywanie złożoności obiektu za interfejsem.
- **Polimorfizm** – zdolność występowania obiektów w kontekście ich klas-przodków.

Zalety modelu obiektowego

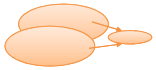
- Modelowanie:
  - naturalna reprezentacja złożonych obiektów świata rzeczywistego,
  - naturalna reprezentacja relacji kompozycji,
  - naturalna reprezentacja interakcji między obiektami,
  - bezpośrednie powiązanie operacji z danymi.
- Programowanie:
  - wielokrotne użycie kodu,
  - modularność i łatwość pielęgnacji,
  - zwiększone bezpieczeństwo i elastyczność,
  - brak konieczności konwersji między modelami danych na styku baza danych – aplikacja (ang. *O/R impedance mismatch*).

Model obiektowo-relacyjny

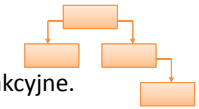
- Umożliwia wykorzystanie mechanizmów obiektowych wewnątrz relacyjnej bazy danych:
  - abstrakcyjne typy danych definiowane przez użytkownika,
  - predefiniowane typy obiektowe dostarczane przez producentów oprogramowania (multimedia, dane przestrzenne),
  - dziedziczenie, enkapsulacja, polimorfizm, klasy abstrakcyjne, …
  - dostępne mechanizmy relacyjne: SQL, przetwarzanie transakcyjne, zarządzanie współbieżnością, ścieżki dostępu, kopie bezpieczeństwa i protokoły odtwarzania spójności, …
- Alternatywa dla systemów odwzorowania obiektowo-relacyjnego (O/RM).

Własności obiektowo-relacyjne w Oracle

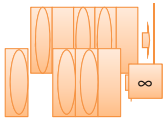
- Obiektowe typy danych użytkownika:
  - metody, składowe, konstruktory, przeciążanie.



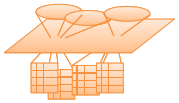
- Współdzielenie obiektów:
  - referencje, nawigacja.



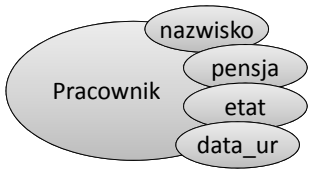
- Dziedziczenie:
  - polimorfizm, przesłanianie metod, typy abstrakcyjne.
- Kolekcje:
  - tabele o zmiennym rozmiarze, tabele zagnieżdżone.



- Perspektywy obiektowe.



Abstrakcyjne typy danych użytkownika



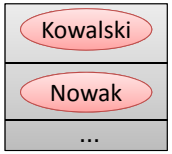
- Każdy typ obiektowy składa się z dwóch części:
  - **deklaracji** (interfejsu): zbioru składowych (pól i sygnatur metod),
  - **definicji** (ciała): implementacji metod.

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

Trwałość obiektów

- Obiekty składuje się trwale w dwóch postaciach:
  - **obiekty krotkowe** (ang. *row object*),
  - **obiekty atrybutowe** (ang. column object).
- Do przechowywania obiektów krotkowych wykorzystuje się **tabele obiektów** (ang. *object table*).
- Obiekty atrybutowe są przechowywane jako wartości atrybutu w zwyczajnej tabeli.

PracownicyObjTab



ProjektyTab

symbol		budzet	kierownik
AB 001	...	20 000	Kowalski
XY 222	...	15 000	Nowak

Tworzenie obiektów

- Do tworzenia obiektu służy specjalna metoda – **konstruktor:**
  - zwraca nową instancję typu obiektowego,
  - ustawia wartości pól w nowej instancji.
- Każdy typ obiektowy posiada **konstruktor atrybutowy:**
  - lista argumentów zgodna z listą atrybutów typu obiektowego,
  - podczas wywołania należy podać wartości wszystkich atrybutów.
- Nazwa konstruktora jest identyczna z nazwą typu obiektowego.
- Wywołanie konstruktora poprzedzamy słowem **NEW** (opcjonalnie).

```
NEW Pracownik('Kowalski', 1200, 'ASYSTENT',
              DATE '1963-10-03')
```

- Użytkownik może deklarować własne konstruktory.
- Sposób tworzenia obiektu jest identyczny dla obiektów krotkowych i obiektów atrybutowych.

Obiekty krotkowe (1)

- Utworzenie tabeli obiektów:

```
CREATE TABLE PracownicyObjTab OF Pracownik;
```

- Ograniczenia integralnościowe:

```
CREATE TABLE PracownicyObjTab OF Pracownik(
  nazwisko CONSTRAINT nazwisko_pk PRIMARY KEY);
```

- Indeksy:

```
CREATE INDEX PracObjTabEtatIdx
   ON PracownicyObjTab(etat);
```


Obiekty krotkowe (2)

- Wstawienie obiektu krotkowego:

```
INSERT INTO PracownicyObjTab VALUES (
    NEW Pracownik('Kowalski', 2500, 'ASYSTENT',
                  DATE'1965-07-01'));

INSERT INTO PracownicyObjTab VALUES (
    Pracownik('Nowak', 3100, 'ADIUNKT',
              DATE '1973-01-30'));
```

Obiekty krotkowe (3)

- Dostęp obiektowy – funkcja **VALUE**:
  - parametr – alias tabeli obiektowej,
  - wartość zwracana – instancje obiektów tabeli obiektowej.

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

- Dostęp relacyjny:

```
SELECT * FROM PracownicyObjTab;
```


Obiekty krotkowe (4)

```
SQL> SELECT VALUE(p) FROM PracownicyObjTab p;

VALUE(P)
----------------------------------------
PRACOWNIK('Kowalski',2500,'ASYSTENT','1965-07-01')
PRACOWNIK('Nowak',3100,'ADIUNKT','1973-01-30')

SQL> SELECT * FROM PracownicyObjTab;

NAZWISKO                 PENSJA ETAT            DATA_UR
-------------------- ---------- --------------- --------
Kowalski                   2500 ASYSTENT        65/07/01
Nowak                      3100 ADIUNKT         73/01/30
```


Obiekty krotkowe (5)

- Modyfikowanie obiektu krotkowego:
  - modyfikowanie wartości pól:

```
UPDATE PracownicyObjTab p
SET p.etat = 'PROFESOR' WHERE p.nazwisko = 'Kowalski';
```

  - zamiana całego obiektu:

```
UPDATE PracownicyObjTab p
SET p = Pracownik('Andrzejak',1500,'ASYSTENT',DATE'1980-11-03')
WHERE p.nazwisko = 'Kowalski';
```

- Usuwanie obiektu krotkowego:

```
DELETE PracownicyObjTab p
WHERE p.nazwisko = 'Andrzejak';
```


Obiekty atrybutowe (1)

- Utworzenie tabeli z obiektami atrybutowymi:

```
CREATE TABLE ProjektyTab (
  symbol CHAR(6), nazwa VARCHAR(100),
  budzet NUMBER, kierownik Pracownik);
```

- Ograniczenia integralnościowe:

```
CREATE TABLE ProjektyTab (
  symbol CHAR(6) CONSTRAINT proj_pk PRIMARY KEY,
  nazwa VARCHAR(100), budzet NUMBER,
  kierownik Pracownik,
  CONSTRAINT etat_chk
    CHECK(kierownik.etat in ('ADIUNKT','PROFESOR')));
```

- Indeksy:

```
CREATE INDEX ProjektyTabEtatIdx
  ON ProjektyTab(kierownik.etat);
```

Obiekty atrybutowe (2)

- Wstawienie rekordu z obiektem atrybutowym:

```
INSERT INTO ProjektyTab VALUES
  ('AB 001','Projekt X',20000,
    NEW Pracownik('Nowak',3200,'ADIUNKT',null));

INSERT INTO ProjektyTab VALUES
  ('XY 003','Projekt Y',45000,
    Pracownik('Kowalski',2500,'PROFESOR',null));
```

- Obiekt pusty a obiekt z pustymi wartościami pól:

```
INSERT INTO ProjektyTab VALUES
  ('1','Projekt 1',100, null);

INSERT INTO ProjektyTab VALUES
  ('2','Projekt 2',200,
     PRACOWNIK(null, null, null, null));
```


Obiekty atrybutowe (3)

- Dostęp obiektów atrybutowych i ich składowych:

```
SELECT nazwa, kierownik FROM ProjektyTab;

SELECT p.kierownik.nazwisko FROM ProjektyTab p;
```

alias jest wymagany w przypadku dostępu do składowych obiektu atrybutowego

Obiekty atrybutowe (4)

```
SQL> SELECT symbol, kierownik FROM ProjektyTab;

SYMBOL KIEROWNIK
------ ----------------------------------------------------
AB 001 PRACOWNIK('Nowak',3200,'ADIUNKT',NULL)
XY 003 PRACOWNIK('Kowalski',2500,'PROFESOR',NULL)

SQL> SELECT symbol, p.kierownik.nazwisko AS szef
     FROM ProjektyTab p;

SYMBOL SZEF
------ --------------------
AB 001 Nowak
XY 003 Kowalski
```


Obiekty atrybutowe (5)

- Modyfikowanie obiektów atrybutowych:

```
UPDATE ProjektyTab p
SET budzet = budzet * 1.1,
    p.kierownik.pensja = p.kierownik.pensja * 1.1
WHERE symbol = 'AB 001';

UPDATE ProjektyTab
SET kierownik =
         Pracownik('Andrzejak',1700,'ADIUNKT',null)
WHERE symbol = 'XY 003';
```

Definiowanie metod typu obiektowego

- Typ obiektowy może posiadać następujące rodzaje metod:
  - metody zwykłe (**MEMBER**) – wołane na rzecz konkretnego obiektu (zarówno funkcje jak i procedury),
  - metody statyczne (**STATIC**) – wołane na rzecz całego typu obiektowego (zarówno funkcje jak i procedury),
  - metody porównujące (**MAP/ORDER**) – wykorzystywane w sytuacji konieczności porównania obiektów (tylko funkcje),
  - konstruktory (**CONSTRUCTOR**) – tworzące nowe obiekty (tylko funkcje).
- Metoda jest definiowana dwukrotnie:
  - w deklaracji typu: **sygnatura metody** (nazwa i lista argumentów),
  - w definicji typu: **ciało metody** (implementacja, kod źródłowy).

Definiowanie metod typu obiektowego

```
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 podwyzka(p_kwota NUMBER));
```

```
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;
END;
```


Parametr SELF metody (1)

- Niejawny parametr każdej metody (z wyłączeniem metod statycznych).

```
MEMBER PROCEDURE podwyzka(pensja NUMBER) ...

to de facto

MEMBER PROCEDURE podwyzka(SELF IN OUT Pracownik,
                          pensja NUMBER) ...
```

- Umożliwia odwołanie z ciała metody do obiektu, dla którego metoda została wywołana.

```
MEMBER PROCEDURE podwyzka(pensja NUMBER) IS
  BEGIN
    SELF.pensja := SELF.pensja + pensja;
  END podwyzka;
```


Parametr SELF metody (2)

- Parametr SELF jest domyślnie deklarowany jako przekazywany w trybie IN OUT:
  - wartość parametru jest kopiowana do metody,
  - dla dużych obiektów może to spowolnić wykonanie metody,
  - rozwiązanie: jawna deklaracja parametru SELF z klauzulą NOCOPY:

```
MEMBER PROCEDURE podwyzka(SELF IN OUT NOCOPY Pracownik,
                          pensja NUMBER) ..
```

Wywoływanie metod składowych

- Metody składowe mogą być uruchamiane:
  - z poziomu SQL (tylko metody będące funkcjami),
  - z poziomu PL/SQL (wszystkie metody).

```
SELECT p.nazwisko, p.data_ur, p.wiek() FROM PracownicyObjTab p;
```

```
DECLARE
  l_kierownik Pracownik;
BEGIN
  SELECT kierownik INTO l_kierownik
  FROM ProjektyTab WHERE symbol = 'AB 001';

  l_kierownik.podwyzka(200);

  UPDATE ProjektyTab
  SET kierownik = l_kierownik WHERE symbol = 'AB 001';
END;
```


Tożsamość obiektów (1)

- Tożsamość obiektu to unikalny i niezmienny identyfikator związany z obiektem przez cały cykl jego życia:
  - tożsamość posiadają tylko obiekty krotkowe,
  - tożsamość jest zapewniana przez OID (ang. *object identifier*),
  - rodzaje OID:
    - OID generowany automatycznie (domyślny), unikalny globalnie, zajmuje 16B,
    - OID generowany na podstawie klucza podstawowego, unikalny lokalnie, rozmiar zależy od rozmiaru wartości klucza podstawowego.

```
CREATE TABLE … OF typ
OBJECT IDENTIFIER IS [SYSTEM GENERATED | PRIMARY KEY];
```


Tożsamość obiektów (2)

- Aplikacja może wykorzystywać referencje do identyfikatorów, odczytywane za pomocą funkcji **REF( )**
  - funkcja zwraca referencję do obiektu krotkowego.

```
SELECT REF(p) FROM PracownicyObjTab p
WHERE p.nazwisko = 'Kowalski';
```


Porównywanie wartości obiektów

- Wartością obiektu jest zbiór wartości jego składowych:
  - dwa obiekty są równe gdy mają te same wartości składowych,
  - standardowo, operatory = i != umożliwiają porównywanie obiektów.
- Porównywanie obiektów najczęściej wymaga sortowania:
  - operatory relacyjne: <, >, <=, >=,
  - operator **BETWEEN ... AND ...**,
  - klauzule **ORDER BY**, **GROUP BY**, **DISTINCT.**
- Metody porównujące: metoda odwzorowująca i metoda porządkująca.
- Cechy:
  - typ obiektowy może mieć tylko jedną metodę porównującą,
  - metody odwzorowujące są bardziej efektywne, metody porządkujące są bardziej elastyczne,
  - metody porównujące są wywoływane automatycznie (niejawnie), ale mogą być również wywoływane jawnie.

Metoda odwzorowująca (1)

- Funkcja zwracająca dla obiektu wartość, która może służyć do porównywania obiektu z innymi obiektami tego samego typu obiektowego.
  - zwracana wartość jest najczęściej wyliczana na podstawie wartości pól obiektu.
- Wywołanie niejawne:

```
obiektA > obiektB
```

jest równoważne

```
obiektA.metoda_odwzorujaca() > obiektB.metoda_odwzorujaca()
```


Metoda odwzorowująca (2)

- 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;
```


Metoda odwzorowująca (3)

- Przykład użycia

```
SQL> SELECT nazwisko, pensja, p.odwzoruj() FROM pracownicyobjtab p;

NAZWISKO                 PENSJA P.ODWZORUJ()
-------------------- ---------- ------------
Kowalski                   2500         3048
Nowak                      3100         3040

SQL> SELECT * FROM pracownicyobjtab p ORDER BY VALUE(p);

NAZWISKO                 PENSJA ETAT            DATA_UR
-------------------- ---------- --------------- --------
Nowak                      3100 ADIUNKT         73/01/30
Kowalski                   2500 ASYSTENT        65/07/01

SQL> SELECT * FROM PracownicyObjTab p
     WHERE VALUE(p) > (SELECT VALUE(r) FROM PracownicyObjTab r
                       WHERE r.nazwisko = 'Nowak');

NAZWISKO                 PENSJA ETAT            DATA_UR
-------------------- ---------- --------------- --------
Kowalski                   2500 ASYSTENT        65/07/01
```


Metoda porządkująca (1)

- Porównuje zawsze dwa obiekty:
  - obiekt, dla którego metoda jest wywoływana (obiekt_1),
  - z obiektem przekazanym do metody przez parametr (obiekt_2).
- Wartości zwracane:
  - jeśli obiekt_1 > obiekt_2: wartość dodatnia,
  - jeśli obiekt_1 < obiekt_2: wartość ujemna,
  - jeśli obiekt_1 = obiekt_2: wartość = 0.
- Stosowana zwykle wtedy, gdy algorytm porównania obiektów jest zbyt złożony dla funkcji odwzorowującej.

Metoda porządkująca (2)

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

ALTER TYPE Pracownik ADD ORDER MEMBER FUNCTION porzadkuj(prac
Pracownik) RETURN NUMBER CASCADE INCLUDING TABLE DATA;
```

```
CREATE OR REPLACE TYPE BODY Pracownik AS
  MEMBER FUNCTION wiek RETURN NUMBER IS
  ...
  MEMBER PROCEDURE podwyzka(p_kwota NUMBER) IS
  ...
  ORDER MEMBER FUNCTION porzadkuj(prac Pracownik) RETURN NUMBER IS
    v_wart1 NUMBER; v_wart2 NUMBER;
  BEGIN
    v_wart1 := ROUND(pensja,-3) + wiek();
    v_wart2 := ROUND(prac.pensja,-3) + prac.wiek();
    IF v_wart1 > v_wart2 THEN return 1;
    ELSIF v_wart1 < v_wart2 THEN return -1;
    ELSE return 0;
    END IF;
  END porzadkuj;
END;
```


Metoda porządkująca (3)

```
SQL> SELECT p1.nazwisko AS p1, p2.nazwisko AS p2,
     p1.porzadkuj(VALUE(p2)) AS porownanie
     FROM pracownicyobjtab p1 CROSS JOIN pracownicyobjtab p2;

P1                   P2                   POROWNANIE
-------------------- -------------------- ----------
Kowalski             Kowalski                      0
Kowalski             Nowak                         1
Nowak                Kowalski                     -1
Nowak                Nowak                         0

SQL> SELECT * FROM PracownicyObjTab p
     WHERE VALUE(p) > (SELECT VALUE(r) FROM PracownicyObjTab r
                       WHERE r.nazwisko = 'Nowak');

NAZWISKO                PENSJA ETAT            DATA_UR
-------------------- ---------- --------------- --------
Kowalski                  2500 ASYSTENT        65/07/01
```


Konstruktor (1)

- Konstruktor to specjalna metoda tworząca nowe obiekty:
  - nazwa konstruktora jest taka sama jak nazwa typu obiektowego,
  - zawsze dostępny domyślny **konstruktor atrybutowy**:
    - liczba parametrów równa liczbie pól typu obiektowego,
    - ustawia wartości pól wartościami parametrów.
  - użytkownik może definiować własne konstruktory, może także przesłonić domyślny konstruktor atrybutowy,
  - konstruktory ułatwiają inicjalizację i umożliwiają ewolucję obiektów.
- Sygnatura konstruktora:

```
CONSTRUCTOR FUNCTION nazwa_typu(lista parametrów)
    RETURN SELF AS RESULT;
```


Konstruktor (2)

```
ALTER TYPE Pracownik ADD CONSTRUCTOR FUNCTION Pracownik(p_nazwisko
VARCHAR2) RETURN SELF AS RESULT CASCADE INCLUDING TABLE DATA;
```

```
CREATE OR REPLACE TYPE BODY Pracownik AS
  MEMBER FUNCTION wiek RETURN NUMBER IS
  ...
  MEMBER PROCEDURE podwyzka(p_kwota NUMBER) IS
  ...
  ORDER MEMBER FUNCTION porzadkuj(prac Pracownik) RETURN NUMBER IS
  ...

  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 Pracownik;
END;
```

```
INSERT INTO PracownicyObjTab VALUES (NEW Pracownik('Kowalski'));
```

Metoda statyczna (1)

- Wywoływana dla typu obiektowego a nie instancji obiektu.
- Może być funkcją lub procedurą.
- Nie posiada parametru **SELF**.
- Sygnatura metody statycznej:

```
STATIC FUNCTION | PROCEDURE nazwa_metody(lista
parametrów) [RETURN typ];
```

- Składnia wywołania: **nazwa_typu.nazwa_metody()**

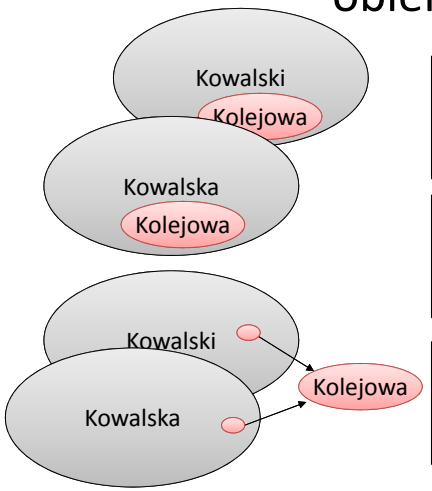
Metoda statyczna (2)

```
ALTER TYPE Pracownik REPLACE AS OBJECT (
  nazwisko VARCHAR2(20),
  ...
  STATIC PROCEDURE dodaj_nowego(nazwisko VARCHAR2));
```

```
CREATE OR REPLACE TYPE BODY Pracownik AS
  ...
  STATIC PROCEDURE dodaj_nowego(nazwisko VARCHAR2) IS
  BEGIN
    INSERT INTO pracownicyobjtab
    VALUES (NEW Pracownik(nazwisko));
  END dodaj_nowego;
END;
```

```
BEGIN
  Pracownik.dodaj_nowego('Jackowski');
END;
```


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);
```


Wiązanie obiektów współdzielonych

- 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

Wykorzystanie referencji do nawigacji

- Referencje można wykorzystać w następujący sposób:
  - nawigacja jawna przez wywołanie funkcji **DEREF( )**:

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

  - nawigacja niejawna przez notację kropkową:

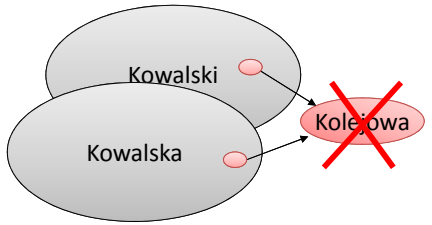
```
SELECT o.imie, o.nazwisko, o.adres.ulica, o.adres.dom
FROM OsobyObjTab o;
```

  - operacja połączenia tabel przez porównanie referencji:

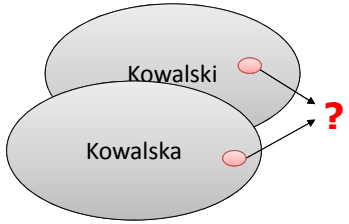
```
SELECT o.imie, o.nazwisko, a.ulica, a.dom, a.mieszkanie
FROM OsobyObjTab o JOIN AdresyObjTab a
  ON o.adres = REF(a);
```


Wiszące referencje

- 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;
```


Słownik bazy danych

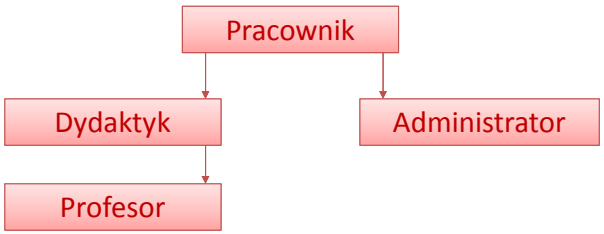
- Perspektywy **USER_TYPES** i **USER_SOURCE** słownika

```
SELECT type_name,typecode,attributes,methods
FROM user_types;

SELECT text FROM user_source
WHERE name = 'PRACOWNIK';
```


Dziedziczenie (1)

- Dziedziczenie polega na definiowaniu nowego typu obiektowego w oparciu o istniejący typ obiektowy:
  - nowy typ stanowi podtyp (specjalizację) swojego nadtypu (przodka),
  - podtyp dziedziczy wszystkie składowe i metody **MEMBER** i **STATIC,**
  - podtyp może dodawać nowe składowe i przesłaniać metody,
  - każdy podtyp może mieć tylko jeden nadtyp,
  - podtyp może dziedziczyć tylko z nadtypu który został zadeklarowany jako **NOT FINAL** (uwaga: domyślnie każdy typ jest **FINAL**).



Dziedziczenie (2)

- Składnia:

```
CREATE TYPE podtyp UNDER nadtyp (...)
[FINAL | NOT FINAL];
```

- Przykład:

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

```
CREATE TYPE Dydaktyk UNDER Pracownik (
  tytul VARCHAR2(10),
  ...
```


Metody w podtypie (1)

- Metoda definiowana w podtypie:
  - nie ma odpowiednika w żadnej z metod nadtypu.
- Metoda przesłaniania w podtypie:
  - nadpisują metodę z nadtypu,
  - jej sygnatura jest identyczną z sygnaturą metody z nadtypu,
  - deklarowana z dodatkową klauzulą OVERRIDING.
- Metoda przeciążana w podtypie:
  - ma taką samą nazwę jak metoda z nadtypu ale inną sygnaturę.

Metody w podtypie (2)

przesłonięcie metody

przeciążenie metody

```
CREATE TYPE Dydaktyk UNDER Pracownik (
  tytul VARCHAR2(10),
  OVERRIDING MEMBER FUNCTION wiek RETURN NUMBER,
  MEMBER FUNCTION wiek(l_data DATE) RETURN NUMBER);
```

```
CREATE TYPE BODY Dydaktyk AS
  OVERRIDING MEMBER FUNCTION wiek RETURN NUMBER IS
  BEGIN
    RETURN ROUND(MONTHS_BETWEEN(CURRENT_DATE, data_ur));
  END wiek;

  MEMBER FUNCTION wiek(l_data DATE) RETURN NUMBER IS
  BEGIN
    RETURN EXTRACT(YEAR FROM l_data) –
           EXTRACT(YEAR FROM data_ur);
  END wiek;
END;
```


Metody w podtypie (3)

- Ograniczenia mechanizmu przesłaniania:
  - redefiniowane mogą być tylko metody zadeklarowane w nadtypie jako NOT FINAL (domyślnie),
    - przykład:

```
CREATE TYPE Pracownik (
  ...
  FINAL MAP MEMBER FUNCTION ...);
```

  - metoda porządkująca może być zdefiniowana tylko w korzeniu hierarchii – nie może być redefiniowana w podtypach,

Metody w podtypie (4)

- Ograniczenia mechanizmu przesłaniania (cd):
  - metoda statyczna w podtypie nie może redefiniować zwykłej metody nadtypu,
  - zwykła metoda w podtypie nie może redefiniować metody statycznej nadtypu,
  - jeśli nadpisywana metoda ma wartość domyślną dla jakiegokolwiek parametru, metoda nadpisująca musi zapewnić tą samą wartość domyślną w podtypie.

Konstruktor podtypu

- Domyślny konstruktor atrybutowy: wszystkie pola nadtypu + pola podtypu.

```
DECLARE
  dydaktyk_1 Dydaktyk := Dydaktyk(
     'Kowalski',
     3500,                    pola nadtypu (Pracownik)
     'ADIUNKT',
     DATE '1972-02-27',
     'dr inż.');              pole podtypu (Dydaktyk)
BEGIN
  …
```

- Konstruktory użytkownika z nadtypu nie są dziedziczone do podtypu – dlatego nie mogą być przesłaniane w podtypie.

Wywołanie metody z nadtypu (1)

- Pozwala dla obiektu-podtypu wywołać metodę z nadtypu
- Konstrukcja: **(obiekt AS nadtyp).metoda**
- Przykład 1.:

```
DECLARE
  d1 Dydaktyk := new Dydaktyk('Kowalski',3500,'ADIUNKT',
                 DATE '1972-02-27','dr inż.');
BEGIN
  DBMS_OUTPUT.PUT_LINE('wiek w miesiącach: '||d1.wiek());
  DBMS_OUTPUT.PUT_LINE('wiek w latach: '||
                        (d1 AS Pracownik).wiek());
END;
```


Wywołanie metody z nadtypu (2)

- Przykład 2.:

```
CREATE TYPE BODY Dydaktyk AS
  OVERRIDING MEMBER FUNCTION wiek RETURN NUMBER IS
    v_wiek_w_latach NUMBER(3);
  BEGIN
    v_wiek_w_latach := (SELF AS Pracownik).wiek();
    RETURN 12 * v_wiek_w_latach;
  END wiek;
  ...
END;
```

Abstrakcyjny typ obiektowy

- Typ, będący korzeniem hierarchii, deklaruje się często jako typ abstrakcyjny:
  - nie można tworzyć instancji tego typu,
  - jest szablonem zbioru pól i metod dla podtypów,
  - deklarowany z klauzulą **NON INSTANTIABLE**,
  - musi mieć dodatkowo klauzulę **NOT FINAL**.

```
CREATE TYPE Osoba AS OBJECT (
  pesel char(11), imie VARCHAR2(100),
  nazwisko VARCHAR2(100), data_ur date,
  MEMBER FUNCTION wiek RETURN number)
NOT INSTANTIABLE NOT FINAL;
```

```
CREATE TYPE BODY Osoba (
  MEMBER FUNCTION wiek RETURN number ...);
```


Abstrakcyjna metoda

- Metoda bez implementacji:
  - klauzula **NON INSTANTIABLE** w sygnaturze.
- Typ zawierający metodę abstrakcyjną sam musi być typem abstrakcyjnym.
- Jeśli podtyp nie redefiniuje metody abstrakcyjnej odziedziczonej z nadtypu, sam musi zostać zadeklarowany jako abstrakcyjny.

```
CREATE TYPE Osoba AS OBJECT (
  pesel char(11), imie VARCHAR2(100),
  nazwisko VARCHAR2(100), data_ur date,
  NOT INSTANTIABLE MEMBER FUNCTION wiek RETURN number)
NOT INSTANTIABLE NOT FINAL;
```


Polimorfizm (1)

- Polimorfizm powoduje, że obiekty podtypu mogą zachowywać się jak obiekty swojego nadtypu:
  - zamiast obiektu nadtypu można wykorzystać obiekt podtypu,
  - wybór wersji metody przeciążonej następuje w momencie wykonania programu (metody są wirtualne).

```
CREATE TABLE pracownicyIdydaktycyObjTab OF Pracownik;

INSERT INTO pracownicyIdydaktycyObjTab VALUES
(NEW Pracownik('Bolek',2000,'ASYSTENT',DATE '1969-09-03'));
INSERT INTO pracownicyIdydaktycyObjTab VALUES
(NEW Dydaktyk('Lolek',5000,'PROF.',DATE '1949-11-13','dr'));

SELECT p.nazwisko, p.wiek()
FROM pracownicyIdydaktycyObjTab p;
```

```
NAZWISKO     P.WIEK()
---------- ----------
Bolek              44
Lolek             768
```


Polimorfizm (2)

- Funkcja **TREAT** - umożliwia zmianę typu wyrażenia do innego typu, cele stosowania:
  - odwołanie do obiektu, będącego wystąpieniem nadtypu, w kontekście podtypu (zawężenie),
  - umożliwienie dostępu do pól lub metod z podtypu zadeklarowanego typu.

```
SELECT p.nazwisko, p.wiek(), p.tytul
FROM pracownicyIdydaktycyObjTab p; -- BŁĄD

SELECT p.nazwisko, p.wiek(), TREAT(VALUE(p) as Dydaktyk).tytul
FROM pracownicyIdydaktycyObjTab p; -- OK

NAZWISKO               P.WIEK() TYTUL
-------------------- ---------- ----------
Bolek                        44            <- Pracownik
Lolek                       768 dr         <- Dydaktyk
```

Polimorfizm (3)

- Pseudokolumna **OBJECT_VALUE** – umożliwia identyfikację wartości obiektów:
  - działa podobnie do funkcji VALUE.

```
SELECT OBJECT_VALUE
FROM pracownicyIdydaktycyObjTab p;

OBJECT_VALUE
------------------------------------------------------------
PRACOWNIK('Bolek',2000,'ASYSTENT','1969-09-03 00:00:00.0')
DYDAKTYK('Lolek',5000,'PROF.','1949-11-13 00:00:00.0','dr')
```


Ewolucja typu (1)

- Obiekty zależne – odwołują się do typu bezpośrednio lub pośrednio.
- Podczas zmiany typu:
  - zależne: podprogramy, perspektywy, operatory i typy indeksowe są unieważniane, muszą zostać zrekompilowane przed ponownym użyciem,
  - zależne indeksy funkcyjne zostają usunięte lub wyłączone (zależnie od zmiany), muszą zostać odbudowane,
  - zależne tabele obiektowe lub z kolumnami obiektowymi: dodanie pola do obiektu dodaje pustą kolumnę, usunięcie pola usuwa skojarzoną z polem kolumnę, modyfikacja typu pola powoduje modyfikację wartości kolumn (o ile to możliwe).

Ewolucja typu (2)

- Modyfikacje danych po modyfikacji typu:

```
ALTER TYPE nazwa ...
  INVALIDATE
  CASCADE [INCLUDING TABLE DATA | NOT INCLUDING TABLE DATA]
```

  - INVALIDATE – unieważnienie obiektów zależnych bez kontroli (szybkie działanie),

Ewolucja typu (3)

- Modyfikacje danych po modyfikacji typu (cd):
  - CASCADE – modyfikacje mają być propagowane do obiektów zależnych:
    - proces przerywany w przypadku błędu, wymuszenie działania opcją FORCE.
    - INCLUDING TABLE DATA – dane w tabelach zależnych mają zostać skonwertowane do nowej postaci typu (domyślne),
    - NOT INCLUDING TABLE DATA – dane w tabelach zależnych nie są modyfikowane,
      - modyfikacja może zostać zrealizowana później poleceniami ALTER TABLE … UPGRADE INCLUDING DATA i ALTER TABLE … DROP UNUSED COLUMNS