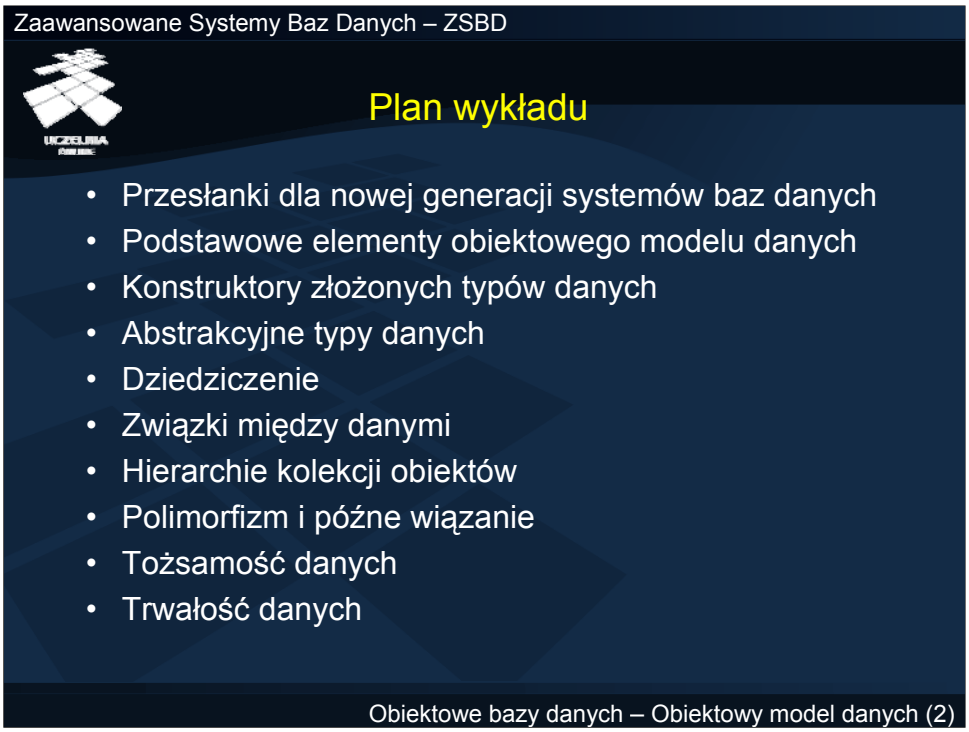
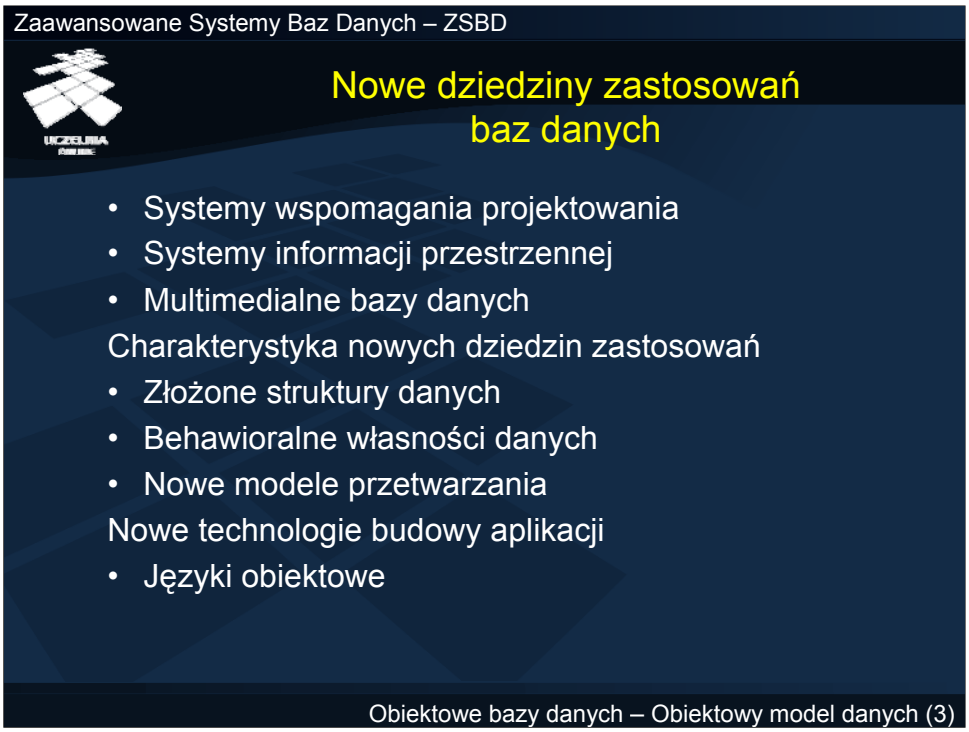
Tematyka obiektowych baz danych obejmuje trzy jednostki wykładowe. Pierwszy wykład dotyczy obiektowego modelu danych. Na drugim wykładzie zostaną przedstawione dwa alternatywne rozwiązania: obiektowe i obiektowo-relacyjne bazy danych. Trzeci wykład jest poświęcony implementacji obiektowych systemów baz danych.

Niniejszy wykład będzie wprowadzeniem do tematyki obiektowych baz danych. Obiektowe bazy danych są propozycją nowego, uniwersalnego i rozszerzalnego modelu danych dla systemów baz danych. W momencie ich pojawienia się miały zastąpić relacyjny model danych, jako lepiej przystosowane do nowych dziedzin zastosowań systemów baz danych.



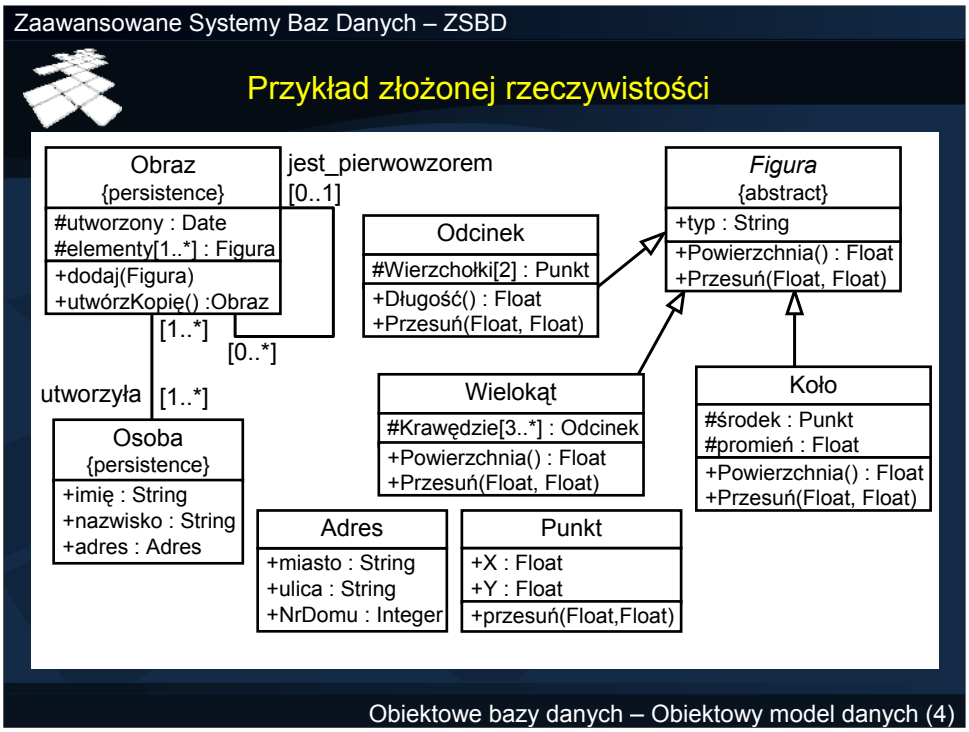
Celem wykładu jest poznanie własności nowego modelu danych zastosowanego w obiektowych bazach danych. Najpierw zastaną przedstawione przesłanki pojawienia się nowej generacji baz danych. Poznamy nowe dziedziny zastosowań systemów baz danych wymagające silniejszego i bardziej elastycznego modelu danych niż model relacyjnych baz danych. W trakcie wykładu przedstawione zostaną podstawowe koncepcje obiektowego modelu danych: możliwość modelowania abstrakcyjnych typów danych, mechanizm dziedziczenia typów danych, konstruktory złożonych typów danych, jawne związki między danymi, hierarchiczne zależności między kolekcjami obiektów oraz mechanizmy polimorfizmu i późnego wiązania. Na koniec zostaną przedstawione rozwiązania służące do zapewnienia obiektom przechowywanym w bazie danych systemowej tożsamości i trwałości.



W czasie powstawania relacyjnego modelu danych typowymi dziedzinami zastosowań systemów baz danych były bankowość, ubezpieczenia, finanse, gospodarka magazynowa, itp. Wspólna charakterystyka systemów tej klasy obejmuje proste struktury danych i prosty model ich przetwarzania. Typowymi wartościami atrybutów danych są teksy, liczby i daty.

Na początku lat osiemdziesiątych, rozpoczęły się próby stosowania systemów baz danych w nowych dziedzinach, takich jak, systemy wspomagania projektowania, systemy informacji przestrzennej lub systemy multimedialne. Charakterystyka tej klasy zastosowań jest diametralnie odmienna. Przetwarzane i składowane dane są złożone strukturalnie. Typowe są hierarchicznie złożone struktury danych oraz liczne i intensywnie przetwarzane powiązania między danymi. Powiązanie te mają złożoną semantykę: referencji, agregacji lub kompozycji. Również semantyka danych jest bardziej złożona. Informacje, które są przetwarzane w nowych dziedzinach zastosowań to długie dokumenty tekstowe, obrazy, animacje, dane wielowymiarowe, itp.

Lata osiemdziesiąte to również okres, kiedy rozpowszechniły się języki obiektowe. Chętnie i powszechnie stosowanymi narzędziami programowymi stosowanymi do budowy aplikacji stały się języki, takie jak: C++, Delphi i Java. Integracja aplikacji baz danych pisanych za pomocą języków obiektowych z relacyjnymi bazami danych była trudna i nienaturalna ponieważ system typów bazy danych i system typów aplikacji są całkowicie odmienne.



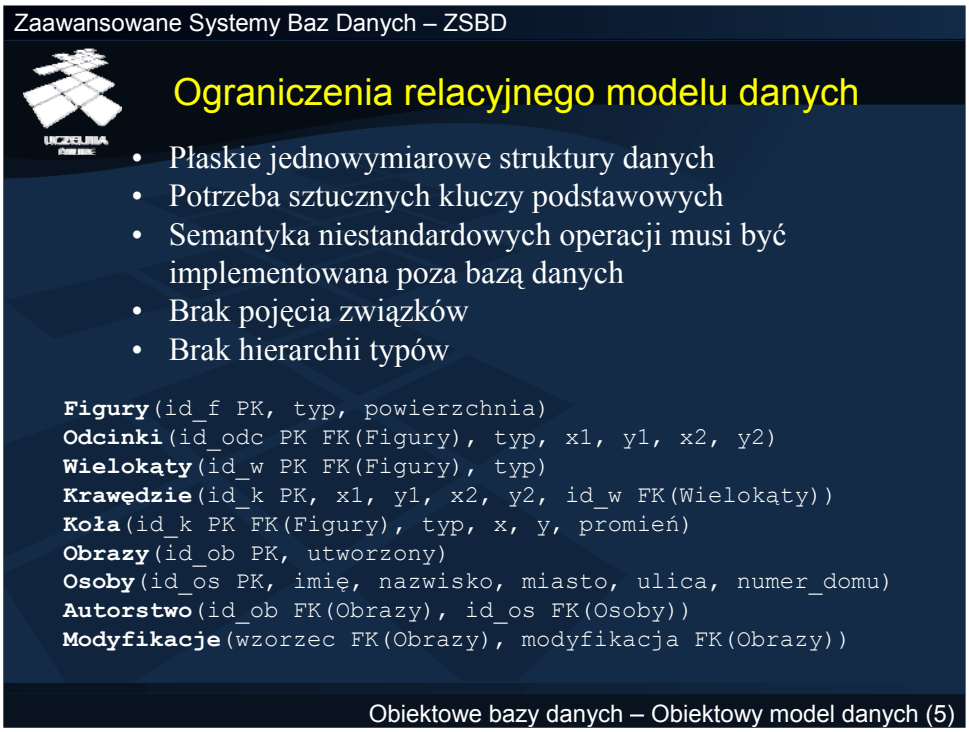
Na rysunku przedstawiono model fragmentu świata rzeczywistego o charakterystyce reprezentatywnej dla wymienionych nowych dziedzin zastosowań. Przykład został zamodelowany za pomocą diagramu klas języka UML.

Pierwszą charakterystyczną cechą zamodelowanej rzeczywistości są złożone struktury danych. Przykładem jest klasa „*Obraz*”, która jest nieograniczoną kolekcją elementów składowych, którymi są „*Figury*”. Poszczególne typy figur również mają złożoną konstrukcję. Na przykład „*Wielokąty*” są kolekcjami krawędzi, które z kolei są parami „*Wierzchołków*”. Na koniec typami danych „*Wierzchołków*” jest klasa „*Punkt*”.

Kolejną cechą przykładu jest zdefiniowana przez użytkownika semantyka operacji dostępnych dla zdefiniowanych klas, wykraczająca poza operacje predefiniowanych typów danych. Przykładem są operacje dostępne dla klasy „*Figura*”: wyznaczanie powierzchni i przesuwanie figur na płaszczyźnie.

Demonstrowany przykład obejmuje również hierarchię klas, której korzeniem jest abstrakcyjna klas „*Figura*”, a liśćmi klasy reprezentujące różne typy figur: „*Odcinki*”, „*Koła*” i „*Wielokąty*”. Dzięki temu atrybut „*Elementy*” klasy „*Figura*” na charakter polimorficzny. Wartościami kolekcji „*Elementy*" są obiekty o różnej strukturze i semantyce operacji.

Ostatnią specyficzną własnością ilustrowaną przez przykład są jawne związki między klasami. Są to: binarny związek między klasą „*Obraz*” i klasą „*Osoba*” reprezentujący autorstwo poszczególnych „*Obrazów*” oraz unarny związek między „*Obrazami*” reprezentujący zapożyczenia między „*Obrazami*”.



Reprezentacja przedstawionego fragmentu rzeczywistości za pomocą relacyjnego modelu danych nie jest ani prosta, ani naturalna.

W przykładzie występują złożone struktury danych, a jedyną strukturą dostępną w relacyjnym modelu danych jest krotka, czyli płaska lista prostych wartości. Relacyjny model danych nie umożliwia zagnieżdżania konstruktorów typów danych. W związku z tym, relacyjna reprezentacja przykładu będzie rozproszonym zbiorem niepowiązanych i jednowymiarowych struktur danych. Przedstawiony na slajdzie schemat relacyjnej bazy danych nie potrafi odzwierciedlić hierarchicznych zależności między danymi. Metodyki poprawnego projektowania schematów relacyjnych baz danych wykluczają również składowanie w bazie danych złożonych wartości atrybutów w sposób transparentny dla modelu danych. Ograniczenie to jest zdefiniowane jako tak zwana pierwsza postać normalna.

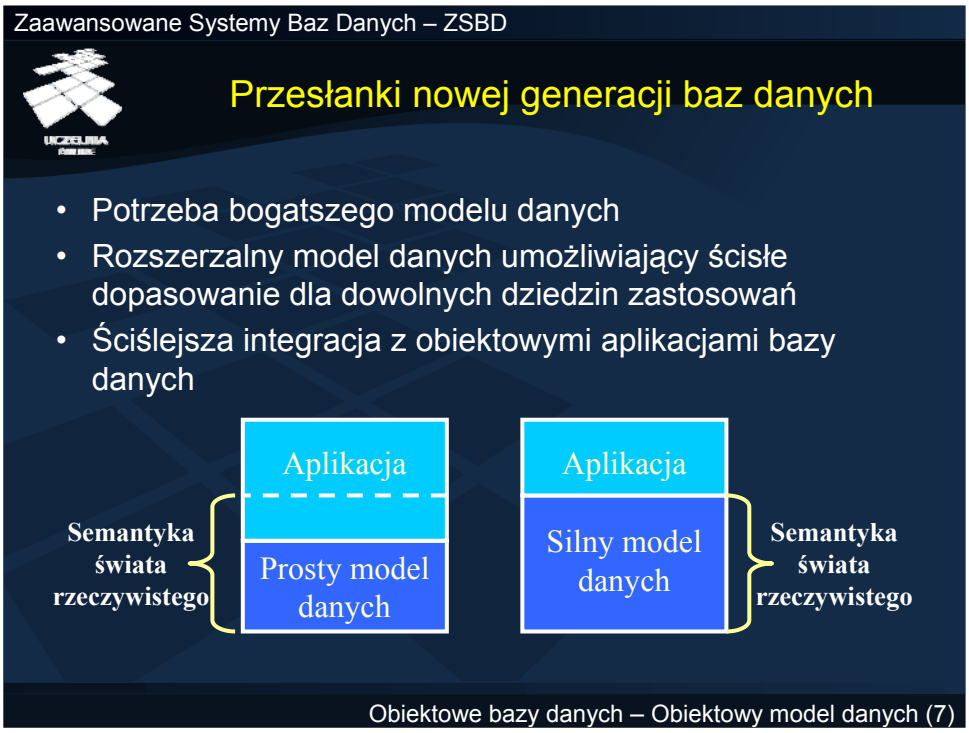
Potrzeba jednoznacznej identyfikacji pojedynczych danych przechowywanych w bazie danych wymaga rozszerzania schematów relacji o sztuczne klucze podstawowe. Dotyczy to przypadków, gdy zdefiniowane atrybuty nie gwarantują unikalności wartości poszczególnych danych. Ponieważ klasy Figura, Koło, Wielokąt, Odcinek nie posiadają atrybutów jednoznacznie identyfikujących ich wystąpienia transformacja tych klas do schematu relacyjnej bazy danych wymaga dodania sztucznych kluczy podstawowych do reprezentujących je relacji.

Relacyjny model danych pozwala na korzystanie jedynie z ograniczonego zbioru prostych predefiniowanych typów danych. Niestandardowa semantyka przetwarzania danych w rozwiązaniach relacyjnych musi być w całości zaimplementowana poza systemem bazy danych, w aplikacjach bazy danych.

Relacyjny model danych nie obejmuje pojęcia związków między danymi. Związki między danymi nie mogą, więc być składowane w bazie danych. Zależności klucz obcy – klucz podstawowy służą jedynie do weryfikacji poprawności danych i nie mogą być wykorzystane do nawigacji między danymi. Związki między danymi są kreowane dynamicznie przez operacje połączenia. Systemy relacyjne realizując operacje połączenia dopiero „w locie” ustalają powiązania między danymi.

Kolejnym ograniczeniem modelu relacyjnego jest brak możliwości modelowania hierarchicznych zależności między kolekcjami danych, między którymi zachodzi relacja podzbioru. Integracja danych reprezentujących różne podzbiory Figur będzie wymagać wykonywania operacji połączenia lub sumy.

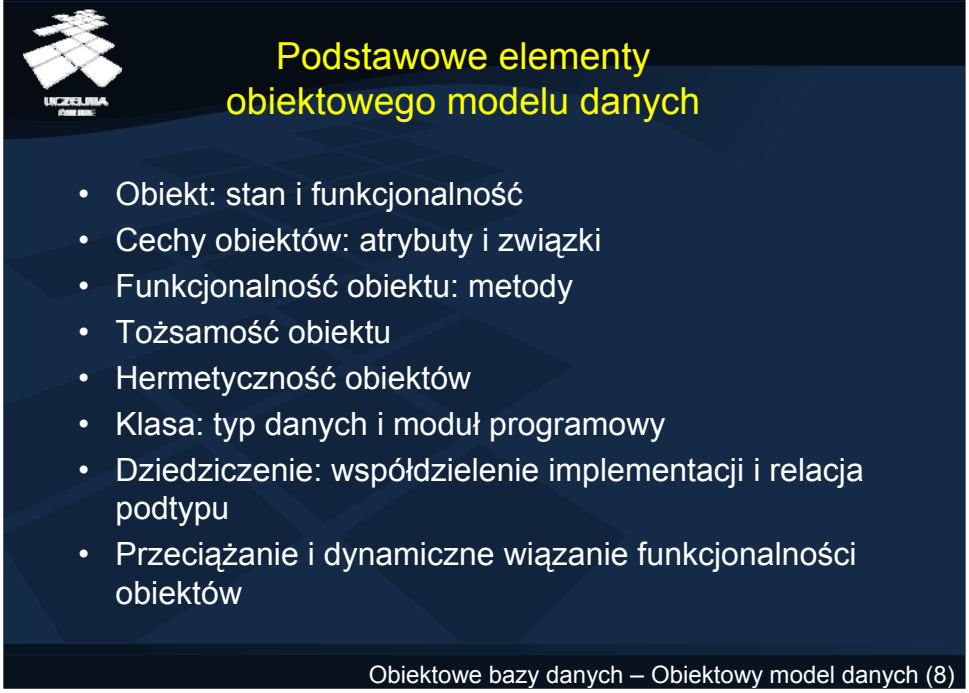
Podsumowując relacyjna implementacja bardziej złożonej rzeczywistości wymaga przeniesienia części jej semantyki do aplikacji bazy danych. Taka semantyka jest trudniej utrzymywana, bo informacji o niej nie ma w bazie danych, lecz trzeba ją utrzymywać w źródłowym kodzie aplikacji. Ponadto, odpowiedzialność za wydajne przetwarzanie semantycznych danych, musi być w tym wypadku przeniesiona z systemu zarządzania bazą danych na programistów tworzących aplikacje. W związku z tym będą to rozwiązania mniej skalowalne.



Reprezentacja świata rzeczywistego o złożonej semantyce wymaga odpowiednio silnego modelu danych. Takiego modelu danych, który pozwoli bezpośrednio wyrazić całą semantykę wybranego fragmentu rzeczywistości. Dzięki temu, aplikacje będą odpowiedzialne „tylko” za realizację logiki procesów biznesowych oraz interfejs z użytkownikami.

Idealny model danych powinien również być uniwersalny, po to by można go było dopasować do charakterystyki dowolnej dziedziny zastosowań. Pomysł ten prowadzi do idei „*rozszerzalnego modelu danych”*. To jest modelu, w którym użytkownicy mogą rozszerzać predefiniowany system typów danych, o dowolnie złożone własne typy danych. Pozwoli to w każdej sytuacji zastosować dokładnie taką siłę modelu, jaka będzie potrzebna dla konkretnego przypadku.

Dodatkowo poszukiwany model danych systemu bazy danych powinien być łatwo integrowalny z nowoczesnymi aplikacjami baz danych budowanymi za pomocą języków obiektowych. Idealna sytuacja polegałaby na przepływie danych między bazą danych a aplikacją bez konwersji niezbędnej dla niedopasowanych systemów typów danych.



Modelem danych, który spełnia większość wymienionych wymagań jest model wypracowany dla obiektowych języków programowania. Model ten został zaadoptowany do potrzeb systemów baz danych.

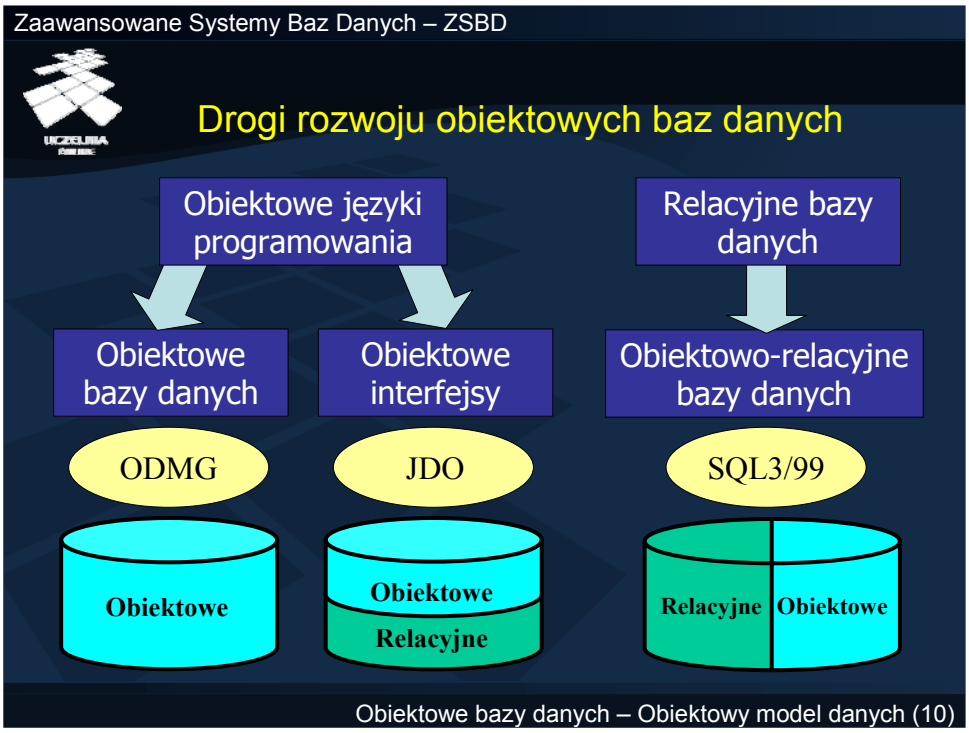
Podstawowym pojęciem tego modelu jest pojęcie obiektu, który umożliwia reprezentowanie cech strukturalnych i behawioralnych obiektów świata rzeczywistego. Struktura obiektu jest opisana przez zbiór atrybutów i związków nazywanych łącznie cechami obiektu. Wartościami atrybutów mogą być wystąpienia prostych typów danych lub obiekty składowe. Wartościami związków są referencje na inne, zewnętrzne obiekty. Zbiór wartości wszystkich atrybutów i związków tworzy stan obiektu. Z kolei własności behawioralne obiektu są reprezentowane przez zbiór dedykowanych procedur zwanych metodami.

Obiekty są jednoznacznie identyfikowane za pomocą systemowego atrybutu nazywanego identyfikatorem obiektu lub w skrócie - OID. Wartości tego atrybutu są unikalne i niezmienne.

Wewnętrzna struktura obiektu oraz implementacja metod są ukryte przed użytkownikami obiektu. Mówi się, że są to prywatne własności obiektów, niedostępne z zewnątrz. Dostęp do obiektów umożliwia ich publiczny interfejs, czyli wywołania metod obiektu. Metody obiektu są wywoływane przez wysyłanie do obiektu odpowiednich komunikatów.

Obiekty o tej samej strukturze i metodach należą do tej samej klasy obiektów. Klasy posiadają dualną naturę. Z jednej strony są odpowiednikami typów danych. Są definicjami własności obiektów. Z drugiej strony klasy są modułami programowymi, które zawierają implementację funkcjonalności typów danych.

Klasy mogą być definiowane jako specjalizacje innych klas. Klasa wyspecjalizowana jest nazywana podklasą i dziedziczy ona cechy i metody swojej nadklasy. Dziedziczenie umożliwia współdzielenie implementacji klas. Dodatkowo klasa wyspecjalizowana jest podtypem swojej nadklasy. Umożliwia to polimorficzne przetwarzanie kolekcji klas i dynamiczne wiązanie przesyłanych komunikatów z metodami klasy właściwej dla danego obiektu.

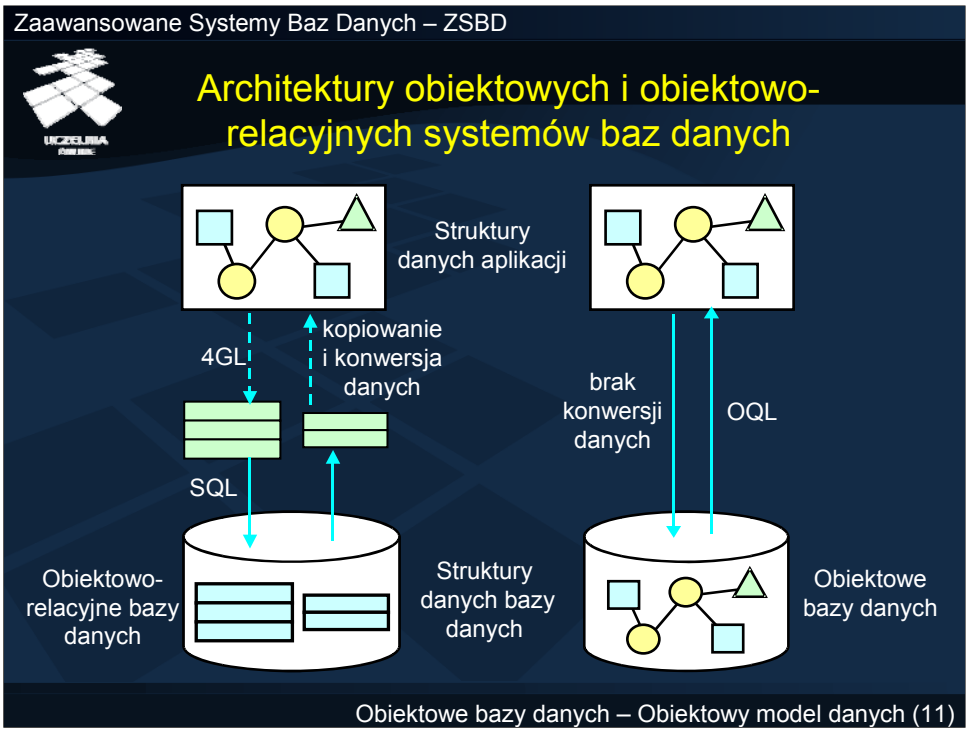


Istnieją dwie podstawowe strategie budowy systemów baz danych o obiektowym modelu danych: rewolucyjna, w której nowe systemy baz danych są budowane w całości od podstaw oraz ewolucyjna, w której obiektowe bazy danych powstają przez przyrostowe modyfikacje istniejących systemów baz danych.

Historycznie pierwszym rozwiązaniem była budowa systemów baz danych od nowa na bazie obiektowych języków programowania. Rozwiązanie to polega na rozszerzeniu funkcjonalności obiektów tworzonych i przetwarzanych przez języki obiektowe o własności typowe dla danych przechowywanych w systemach baz danych, to jest: trwałości, współdzielenia, synchronizacji dostępu, odtwarzania spójnego stanu po awarii, wydajnego przetwarzania dużych zbiorów obiektów, itp. Wynikiem tej strategii są "czysto" obiektowe bazy danych posiadające jednorodny obiektowy model danych. Standard dla modelu danych tej klasy systemów został opracowany przez organizację „*Object Database Management Group”* i od nazwy tej grupy jest nazwany „ODMG 3.0”.

Kontynuatorem tej strategii jest rozwiązanie, w którym funkcjonalność obiektowego modelu danych jest implementowana przez dodatkową warstwę programową budowaną na systemie bazy danych o dowolnym modelu danych. Standardem związanym z tą strategią jest ”*Java Data Objects”* - JDO.

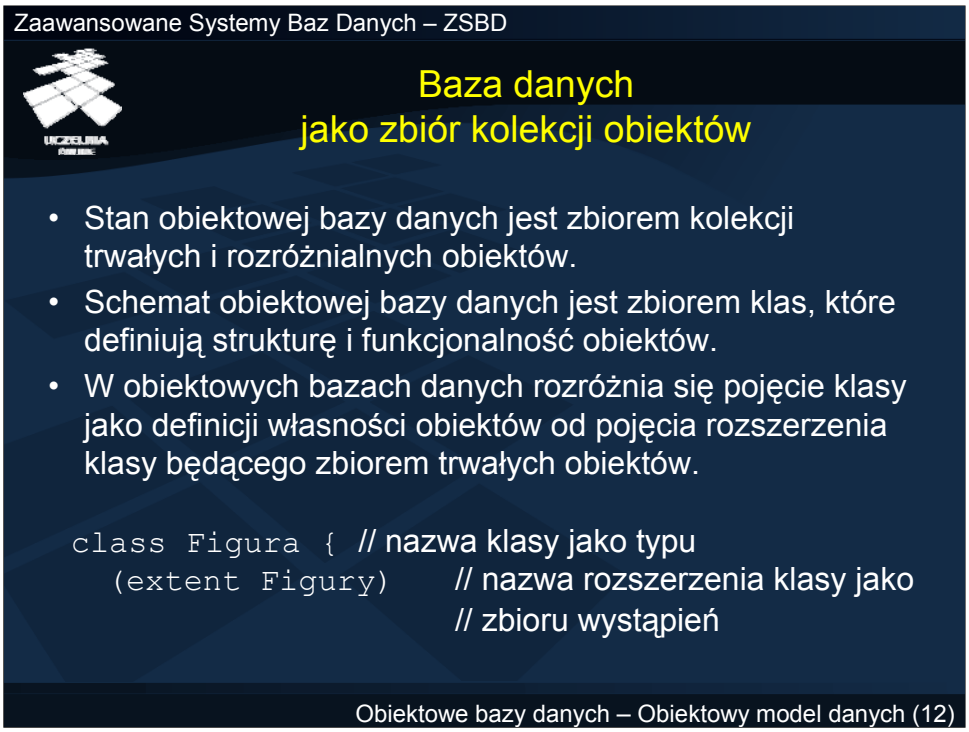
Całkowicie odmiennym rozwiązaniem jest ewolucyjna modyfikacja relacyjnych systemów baz danych polegająca na rozszerzeniu relacyjnego modelu danych o własności modelu obiektowego: hermetycznych obiektów zintegrowanych z metodami, dziedziczenia, polimorfizmu, dynamicznego wiązania, itp. Wynikiem jest obiektowo-relacyjny model danych, który jest konglomeratem cech relacyjnych i obiektowych. Własności modelu obiektowo-relacyjnego są opisane w standardzie SQL3 (SQL99).



Obiektowo-relacyjne systemy bazy danych dziedziczą po systemach relacyjnych cechę niedopasowania systemów typów danych bazy danych i aplikacji bazy danych, co zostało zilustrowane na rysunku po lewej stronie slajdu. Dane przechowywane w bazie danych mają inną strukturę i semantykę niż dane przetwarzane przez aplikacje. Ta sama informacja w ciągu swojego cyklu życia obejmującego przenoszenie jej między pamięcią dyskową, a pamięcią operacyjną zmienia swoją fizyczną reprezentację. W bazie danych jest krotką o określonej reprezentacji fizycznej poszczególnych typów danych, tekstowych, numerycznych i dat. Z kolei podczas wizualizowania na ekranie komputera przez aplikacje ta sama informacja jest kolekcją danych o odmiennych reprezentacjach fizycznych.

Na przykład, nazwisko studenta w bazie danych jest przechowywane w polu o typie VARCHAR(20), którego fizyczna reprezentacja jest dwuelementową tablicą, której pierwszy element pamięta faktyczną długość tekstu, a drugi jest łańcuchem kodów ASCII. Tymczasem aplikacja napisana w języku C++ przechowuje nazwisko studenta w zmiennej wskaźnikowej na tekst, który jest ciągiem kodów ASCII zakończonym wyróżnioną wartością zera binarnego. Przenoszenie informacji między bazą danych, a aplikacją wymaga więc bezustannej transformacji fizycznej reprezentacji tej informacji. Niedopasowanie systemów typów danych ogranicza wydajność działania oraz komplikuje tworzenie aplikacji.

Czysto obiektowe bazy danych nie posiadają tej wady. Obiekty w bazie danych i obiekty przetwarzane przez aplikacje mają dokładnie taką samą strukturę i semantykę. Przenoszenie danych między pamięcią operacyjną i dyskową nie wymaga żadnego przekształcania danych. Ilustruje to rysunek po prawej stronie slajdu.



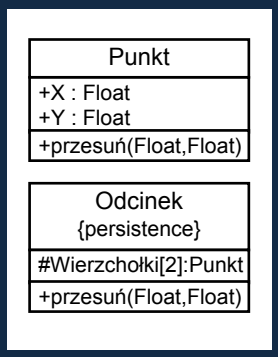
Obiekt jest podstawową jednostką danych składowanych w obiektowej bazie danych. W przeciwieństwie do obiektów przetwarzanych w zwykłych programach obiektowych, obiekty utworzone przez aplikacje obiektowych baz danych są autonomiczne w stosunku do tych aplikacji i trwałe. Autonomia obiektów polega na możliwości istnienia obiektu poza programem, który go utworzył. Natomiast trwałość oznacza, że czas życia obiektu jest dłuższy niż czas życia zmiennej, której był przypisany w momencie utworzenia. Aplikacja, która utworzyła obiekt może zostać wyłączona, a utworzone przez nią trwałe obiekty będą pamiętane w bazie danych.

Struktura i funkcjonalność obiektów bazy danych jest zdefiniowana za pomocą klas. Zbiór definicji klas tworzy schemat bazy danych. W przeciwieństwie do systemów relacyjnych, gdzie relacja pełniła jednocześnie rolę definicji struktury danych oraz zbioru wszystkich danych o tej strukturze, w obiektowych bazach danych te role są rozdzielone. Zbiór trwałych wystąpień danej klasy tworzy tak zwane rozszerzenie klasy. Ilustruje to przykład na dole slajdu. „*Figura*” jest nazwą klasy definiującej własności obiektów, a „*Figury*” są nazwą rozszerzenia, które jest zbiorem wszystkich trwałych wystąpień klasy „*Figura*”.

Abstrakcyjne typy danych

Możliwość definiowania nowych typów danych o dowolnej złożoności i funkcjonalności. Typy danych użytkownika mogą być podstawą definicji pojedynczych atrybutów klas jak również całych klas.

```
class Punkt {
  Float X, Y;
  void przesuń (in Float x,
                in Float y);};
class Odcinek {
  Punkt W1, W2;
  void przesuń (in Float a,
                in Float b) {
     W1.przesuń(a, b);
     W2.przesuń(a, b); }};
```

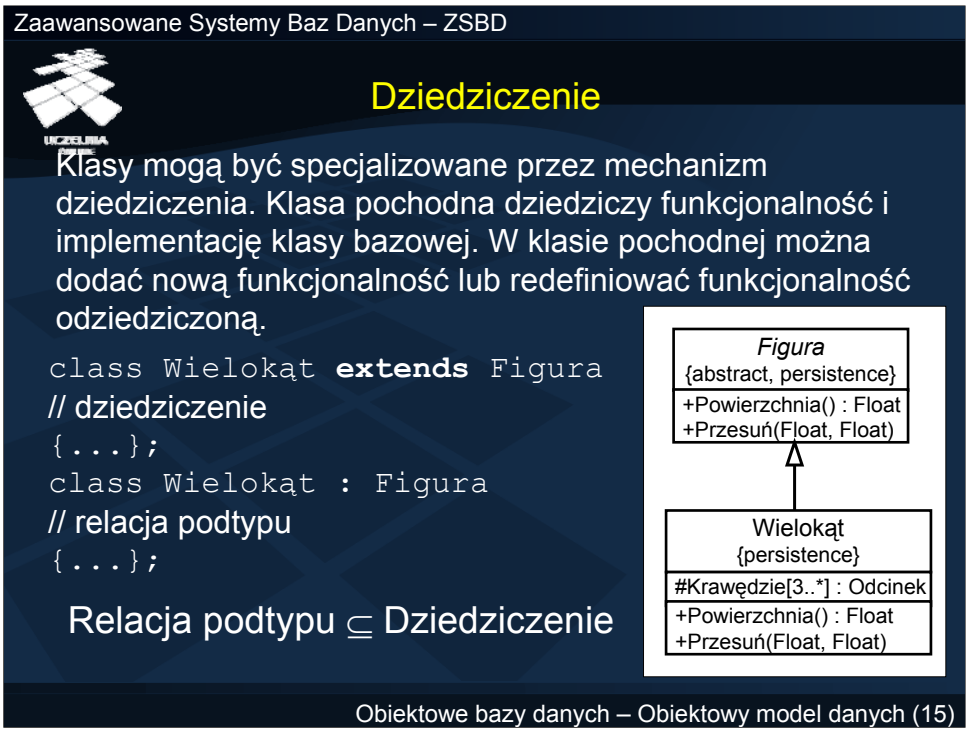


Relacyjne bazy danych oferują projektantowi schematu bazy danych ograniczony zbiór prostych, predefiniowanych typów danych dla atrybutów relacji. Przykładem mogą być systemowe typy tekstowe: varchar i char, numeryczne: int i float oraz temporalne: date. Typy danych niedostępne w relacyjnym modelu danych muszą być implementowane poza systemem bazy danych. Kod obsługujący ich semantykę musi być implementowany i powielany we wszystkich aplikacjach bazy danych.

Obiektowe bazy danych umożliwiają projektantom definiowanie nowych typów danych. Definiowanie własnego typu danych obejmuje definicję struktury i funkcjonalności typu. Definicje typów są składowane w systemie bazy danych. Sposób korzystania z typów danych zdefiniowanych przez użytkownika nie różni się niczym od korzystania z typów systemowych. Dzięki temu obiektowy model danych jest rozszerzalny. Można go elastycznie dopasowywać do dowolnej dziedziny zastosowania wymagającej unikalnych, specyficznych typów danych.

Rolę typów danych definiowanych przez użytkownika pełnią w obiektowych bazach danych klasy i interfejsy. Definicja interfejsu jest opisem funkcjonalności nowego typu danych, czyli opisem operacji dostępnych dla danego typu. Definicja klasy obejmuje dodatkowo implementację typu danych: to jest wewnętrzną strukturę danych służącą do przechowywania stanu wystąpień typu danych oraz kod operacji zdefiniowanych dla danego typu danych.

W zaprezentowanym przykładzie zdefiniowano dwa typy danych użytkownika: klasy Punkt i Odcinek. Część strukturalna klasy Punkt to dwa atrybuty X i Y reprezentujące lokalizację punktu na płaszczyźnie. Część funkcjonalna tej klasy jest ograniczona do metody Przesuń służącej do zmiany lokalizacji punktów. Klasa Punkt została wykorzystana jako typ danych dla atrybutów W1 i W2 klasy Odcinek. Wystąpienia klasy Punkt będą wartościami tych atrybutów. Klasa Odcinek służy do definiowania typu obiektów składowanych w bazie danych. Metoda przesuń jest zaimplementowana jako sekwencja wywołań operacji przesunięcia dla dwóch końców W1 i W2 danego odcinka.



W obiektowych bazach danych można definiować nowe klasy i interfejsy wywodząc je ze zdefiniowanych wcześniej klas lub interfejsów. Model ODMG rozróżnia dwa typy powiązań między klasami lub interfejsami: związek relacji podtypu i związek dziedziczenia. W językach obiektowych te dwa związki są traktowane jako tożsame.

Związek relacji podtypu może dotyczyć klas i interfejsów. Oznacza on dziedziczenie przez typ pochodny funkcjonalności typu bazowego. Klasy i interfejsy połączone związkiem podtypu tworzą sieć powiązań o topologii grafu acyklicznego skierowanego. Oznacza to, że pojedyncza klas lub interfejs może dziedziczyć funkcjonalność po wielu klasach lub interfejsach.

Związek dziedziczenia łączy jedynie klasy. Oznacza on dziedzicznie zarówno funkcjonalności jak i implementacji. Związek dziedziczenia obejmuje semantykę relacji podtypu. Klasy połączone związkiem dziedziczenia tworzą sieć powiązań o topologii hierarchii. Oznacza to, że pojedyncza podklasa może dziedziczyć zarówno funkcjonalność jak i implementację po dokładnie jednej nadklasie.

Dziedziczona funkcjonalność może być rozszerzona w stosunku do typu bazowego. Dziedziczona implementacja może być rozszerzana lub przesłaniana. Przez przesłaniane rozumie się definicje nowego kodu dla odziedziczonych metod, który zastąpi stary kod.

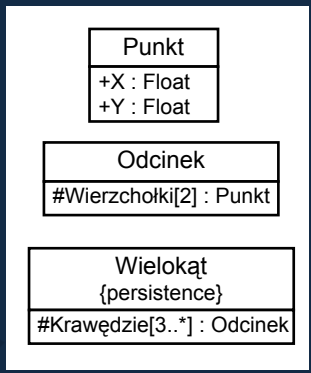
Przedstawiony na slajdzie przykład pokazuje dwa alternatywne rozwiązania. W pierwszym klasa Wielokąt dziedziczy po klasie Figura funkcjonalność i implementację, w drugim klasa Wielokąt przez relację podtypu dziedziczy funkcjonalność bez implementacji.

Złożone struktury danych

Możliwość naturalnego modelowania atrybutów **złożonych** i **wielowartościowych**. Przykład zastosowania konstruktorów typów złożonych w języku ODL.

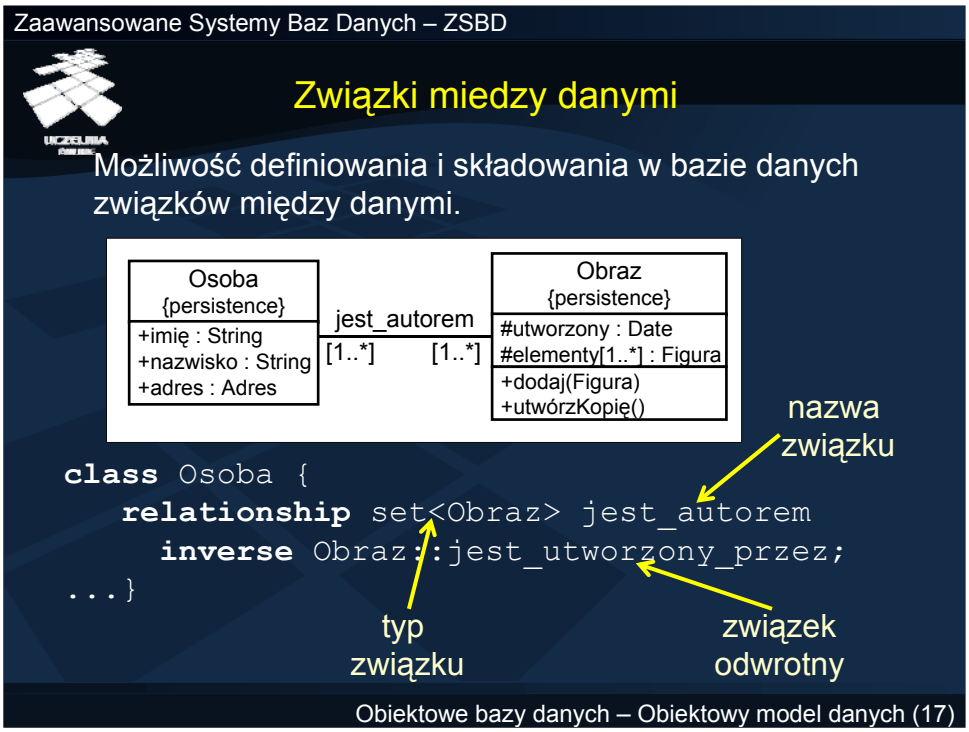
```
class Wielokąt {
   struct Punkt {
     Float X, Y; }
   struct Odcinek {
     Punkt Wierzchołek_1;
     Punkt Wierzchołek_2;}

   attribute set<Odcinek> krawędzie;};
```



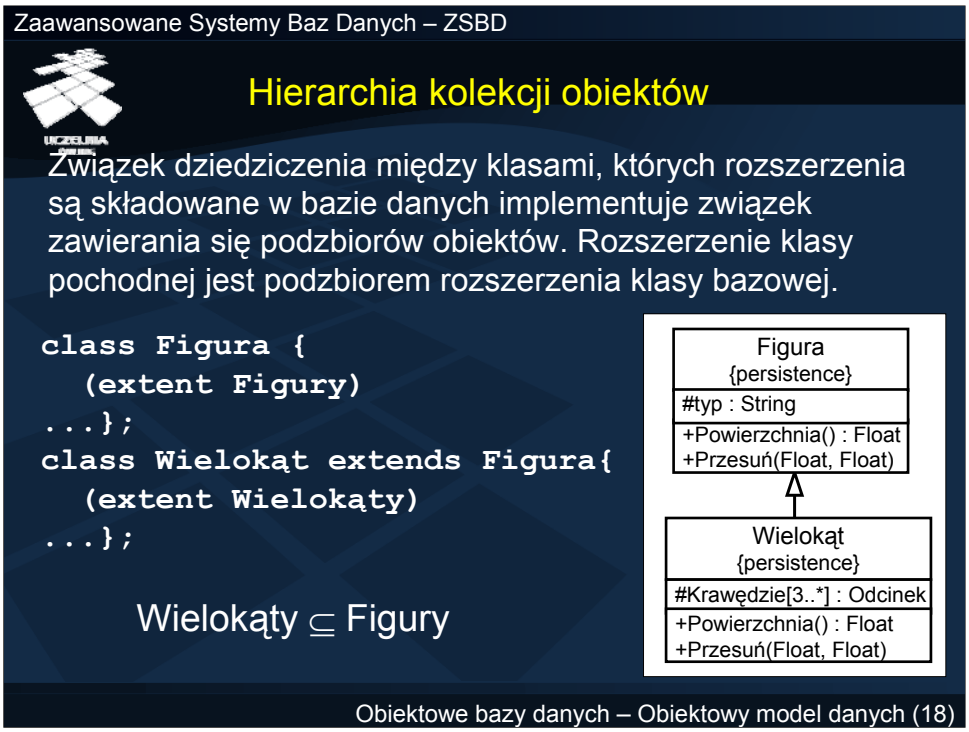
Kolejnym ograniczeniem przełamanym przez obiektowy model danych jest możliwość przechowywania w bazie danych, obiektów o złożonej strukturze. Elementem standardów ODMG i SQL3 jest bogaty zbiór konstruktorów złożonych typów danych. Kompletna lista obejmuje konstruktory: krotki, zbioru, wielozbioru, listy, tablicy i słownika. Konstruktory złożonych typów danych mogą być wielopoziomowo zagnieżdżane.

Przykład na slajdzie pokazuje transformację do schematu obiektowej bazy danych klasy Wielokąt o atrybucie wielowartościowym i złożonym. Atrybutem klasy Wielokąt jest zbiór Odcinków, z których każdy jest strukturą pary Punktów, które z kolei są strukturą pary współrzędnych. W przykładzie zastosowano dwa typy konstruktorów: konstruktora krotki – „*struct*” i konstruktora zbioru – „*set*”.



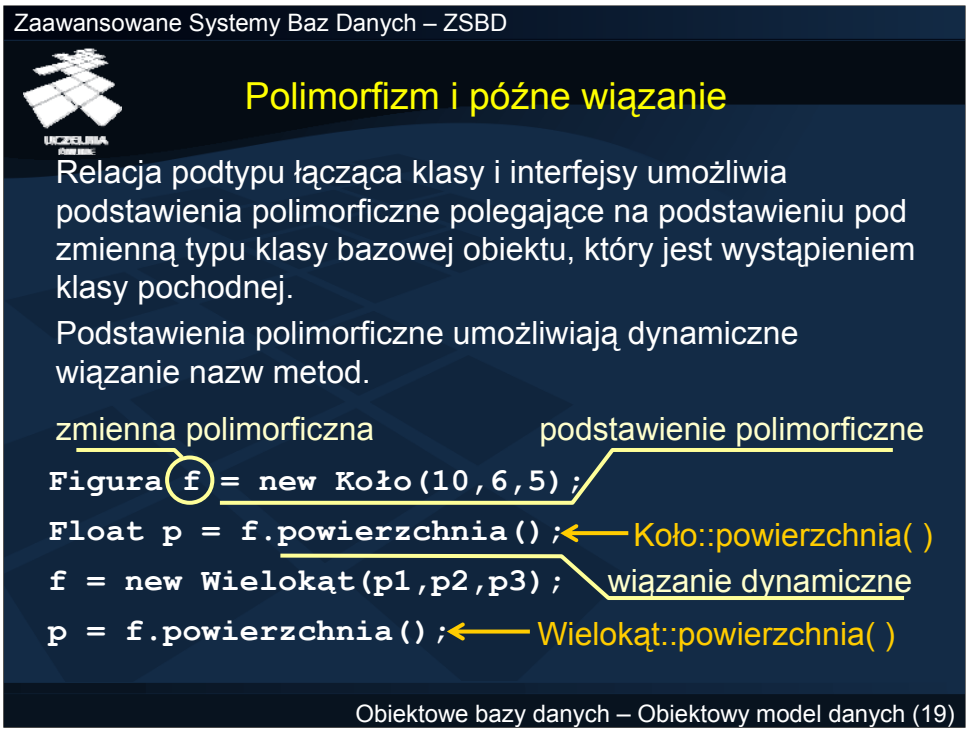
Obiektowy model danych pozwala na jawną reprezentację związków między danymi. W przeciwieństwie do relacyjnego modelu danych, gdzie związki są ustalane w sposób dynamiczny w momencie wykonywania zapytań, a dokładniej operacji połączenia (ang. join), w obiektowym modelu danych związki są pamiętane w bazie danych. Podczas przetwarzania powiązanych danych dostępna jest operacja nawigacji wzdłuż tych powiązań. W modelu ODMG przyjęto dodatkowo, że wszystkie związki są dwukierunkowe, co oznacza, że dla powiązanych obiektów A i B możliwa jest nawigacja od obiektu A do obiektu, jak i na odwrót, od obiektu B do obiektu A. Model obiektowy nie stawia ograniczeń na krotność związku. Dozwolone są powiązania jednokrotne i wielokrotne.

Na slajdzie pokazano przykład związku łączącego klasę *Obraz* z klasą *Osoba*, które je utworzyły. Obiekty obydwu tych klas przechowują informacje o obiektach, z którymi są powiązane. Po stronie osób związek ten nosi nazwę: „*jest_autorem*”, a po stronie obrazów: „*jest_utworzony_przez*”. Po obydwu stronach jest to powiązanie wielokrotne. Pojedyncza osoba może być autorem wielu obrazów, a pojedynczy obraz może mieć wielu autorów. Wartością związku „*jest_autorem*” jest zbiór identyfikatorów obiektów, które reprezentują obrazy utworzone przez daną osobę. Analogicznie wartością związku „*jest_utworzony_przez*” jest zbiór identyfikatorów obiektów, które reprezentują osoby będące autorami danego obrazu.



Związek dziedziczenia lub czysta relacja podtypu łączące klasy definiuje relację podzbiorów między ich rozszerzeniami. Rozszerzenie klasy pochodnej jest podzbiorem rozszerzenia klasy bazowej. Rozszerzenie klasy bazowej obejmuje rekurencyjnie obiekty należące do wszystkich rozszerzeń bezpośrednich i pośrednich klas pochodnych. Relacja ta pozwala na przetwarzanie heterogenicznych zbiorów obiektów. Operacje wykonywane na rozszerzeniu klasy bazowej są automatycznie propagowane na rozszerzenia wszystkich klas pochodnych.

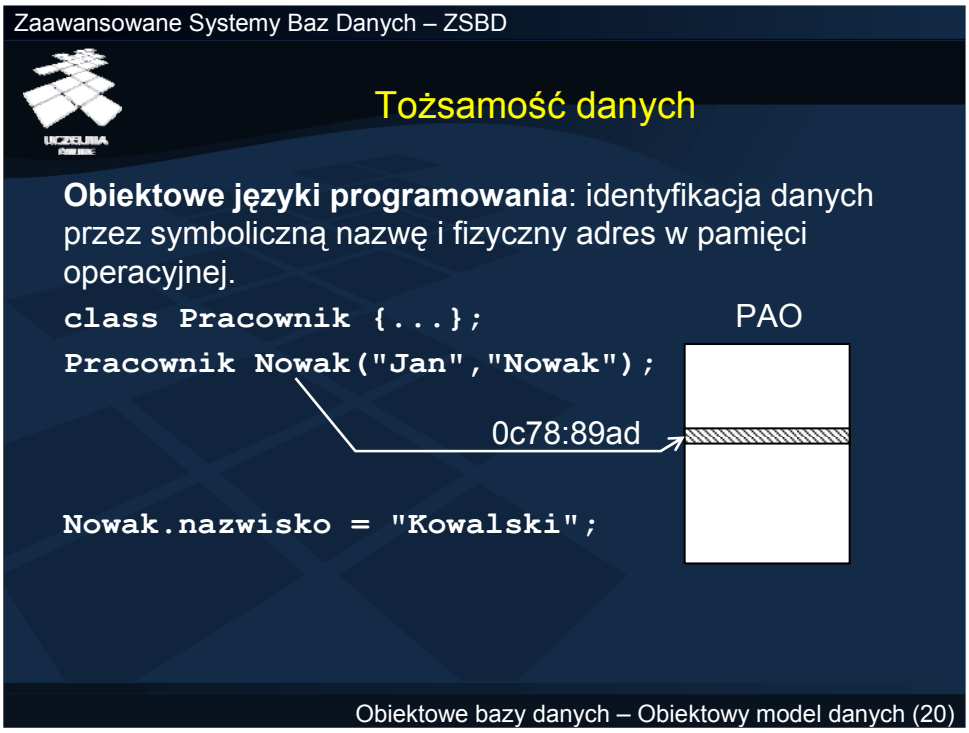
Na slajdzie jest to zilustrowane przykładem relacji podzbioru łączącej rozszerzenie „*Figury*” klasy „*Figura*” z rozszerzeniem „*Wielokąty*” klasy „*Wielokąt*”. Operacje wykonywane na zbiorze wystąpień klasy „*Figura*” będą wykonywane również na wystąpieniach klasy „*Wielokąt*”.



O obiektach składowanych w rozszerzeniach klas, które mają klasy pochodne mówi się, że są polimorficzne, bo mają różne cechy i metody. Na przykład wystąpienia klasy Figura mają określony atrybut „*Typ*” służący do przechowywania informacji o typie figury oraz metodę „*Powierzchnia*” służącą do wyznaczania powierzchni figur. Z kolei wystąpienia klasy pochodnej „*Wielokąt*” oprócz odziedziczonego atrybutu „*Typ*” mają dodatkowo wielowartościowy atrybut „*Krawędzie*”. Odziedziczona po klasie „*Figura*” metoda „*Powierzchnia*” jest w klasie „*Wielokąt*” redefiniowana ponieważ sposób wyznaczania powierzchni wielokątów jest inny niż uniwersalnych figur. Rozszerzenie klasy „*Figura*” obejmuje zarówno obiekty klasy „*Figura*”, jak również różniące się od nich obiekty, które są wystąpieniami klasy „*Wielokąt*”.

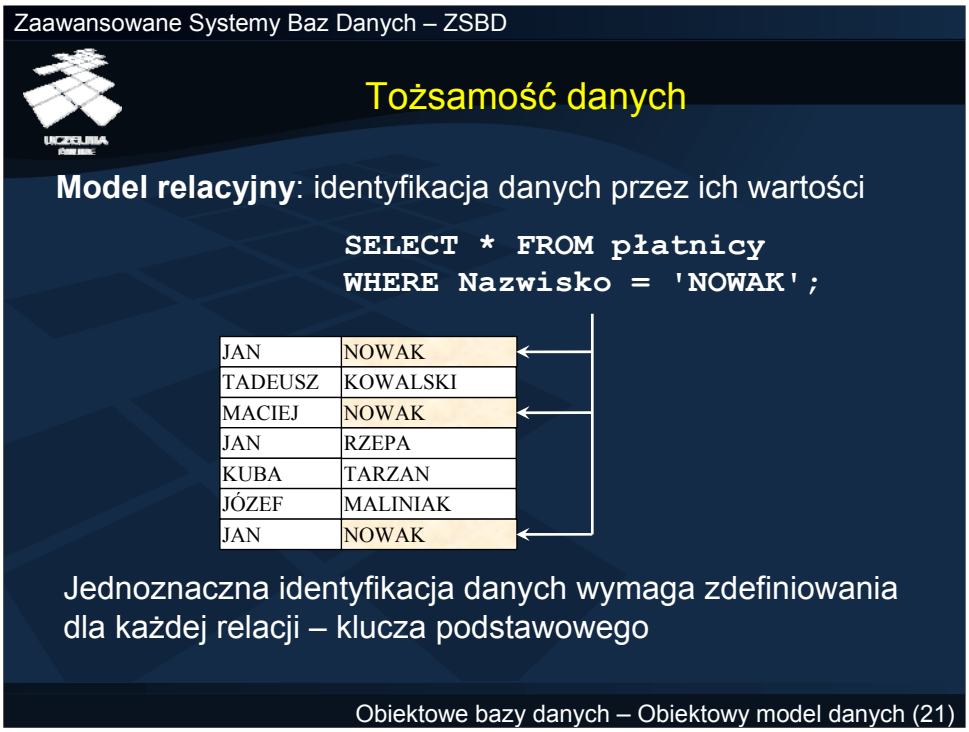
Na poziomie składni języków obiektowych polimorfizm obiektów wyraża się w występowaniu zmiennych polimorficznych i podstawień polimorficznych. Przez zmienną polimorficzną rozumie się taką zmienną, pod którą są podstawiane obiekty, które są wystąpieniami różnych klas, ale które występują w relacji podtypu z typem zmiennej polimorficznej. W podanym przykładzie, zmienną polimorficzną jest zmienna „*f*”. Mimo, że typem zmiennej „*f*” jest klasa „*Figura*”, to przypisywane są jej obiekty innych klas. Najpierw pod zmienną „*f*” jest podstawiony obiekt typu „*Koło*”, a później obiekt typu „*Wielokąt*”. Następnie przez zmienną „*f*” do przypisanych jej obiektów są przesyłane komunikaty o nazwie „*Powierzchnia*”. Właściwy kod metody wyznaczania powierzchni jest ustalany dynamicznie na podstawie typu obiektu przypisanego w danym momencie zmiennej „*f*”. W pokazanym przykładzie najpierw jest to kod metody „*Powierzchnia*” zdefiniowanej w klasie „*Koło*”, następnie kod metody „*Powierzchnia*” klasy „*Wielokąt*”.

Szczególnym przypadkiem zmiennych polimorficznych są nazwy rozszerzeń klas, które mają klasy pochodne. Nazwa rozszerzenia klasy bazowej pozwala przetwarzać obiekty należące do rozszerzeń wszystkich klas pochodnych.



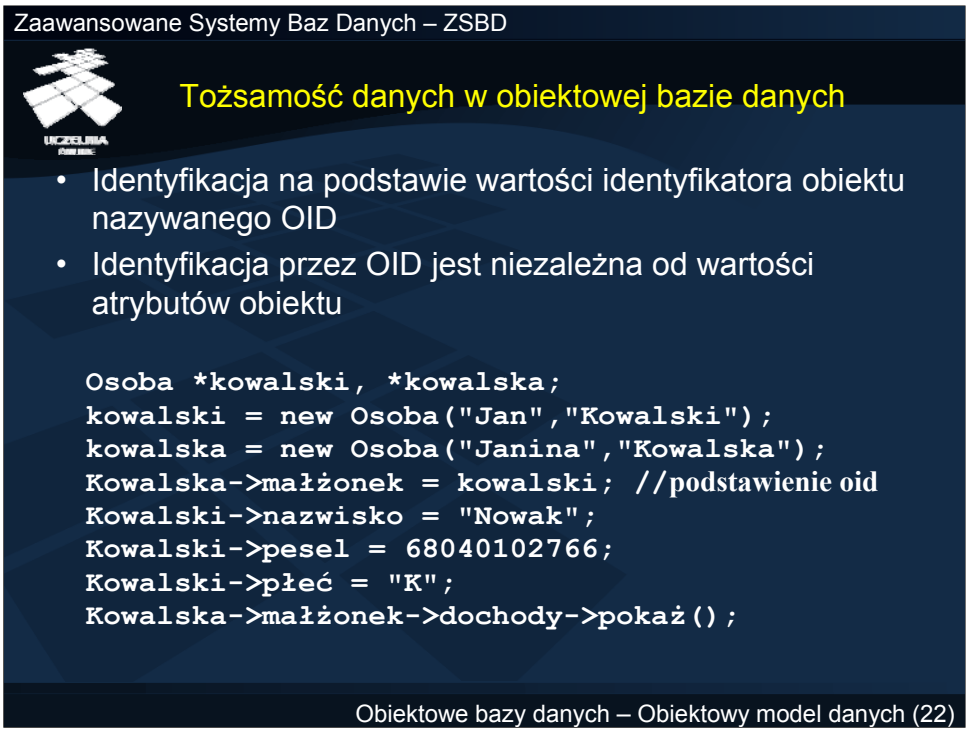
Dostęp do danych w bazie danych z poziomu aplikacji bazy danych wymaga mechanizmu identyfikacji danych. Programista musi dysponować jakimiś środkami wyrazu, które pozwolą mu wskazać te dane, które chce przetwarzać. Dostęp do danych lokalnych programów jest realizowany poprzez zmienne przypisane przetwarzanym danym. Nazwy zmiennych są środkiem jednoznacznej identyfikacji danych. W danej części programu nazwy zmiennych muszą być unikalne. W przedstawionym przykładzie zmienna o nazwie Nowak jednoznacznie określa podstawiony pod nią obiekt, który jest wystąpieniem klasy Osoba. Przesyłanie komunikatów do tego właśnie obiektu wymaga użycia nazwy zmiennej.

W wykonywalnej postaci programów nazwy zmiennych są transformowane do adresów w pamięci operacyjnej, pod którymi obiekty zostaną ulokowane. W naszym przykładzie symboliczna nazwa Nowak zostaje zamieniona na adres pamięci operacyjnej 0c78:89ad.



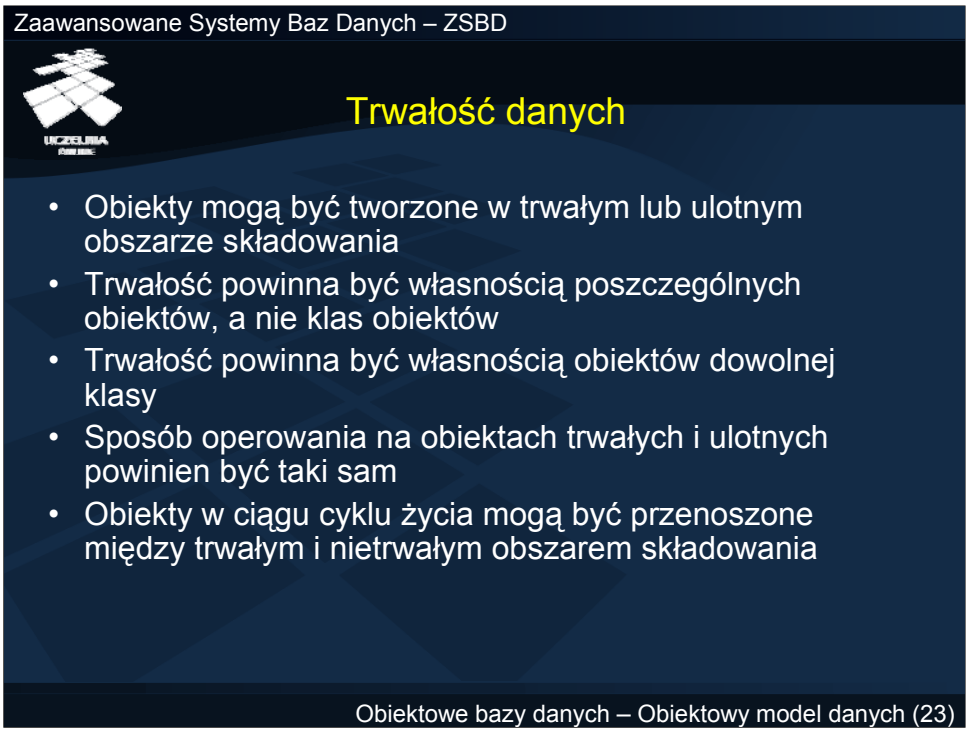
Cechą relacyjnego modelu danych jest identyfikacja danych poprzez wartość. W przeciwieństwie do danych przetwarzanych w proceduralnych językach programowania, pojedyncze krotki w bazie danych nie mają przypisanych żadnych symbolicznych nazw. Przez nazwę identyfikowalne są jedynie całe relacje. Również fizyczna lokalizacja krotek w pamięci dyskowej nie jest podstawą do ustalenia ich tożsamości. Fizyczna reorganizacja danych może się bowiem wiązać z realokacją krotek w pamięci dyskowej.

Jedyną cechą danych do której można się odwołać wyszukując ich w bazie danych są ich wartości. Zapytania w relacyjnych bazach danych wyrażane w języku SQL opierają się na wyrażeniach logicznych odwołujących się do wartości wybranych atrybutów krotek. Pokazuje to przykład na slajdzie, w którym w zbiorze Płatników są wyszukiwane osoby, dla których wartość atrybutu Nazwisko jest równa stałej tekstowej „Nowak”. W sytuacji gdy atrybuty, do których odwołuje się zapytanie nie mają cechy unikalności identyfikacja poprzez wartość nie jest jednoznaczna. W podanym przykładzie, aż trzy osoby noszą nazwisko Nowak. Dla jednoznacznej identyfikacji danych schematy relacji muszą posiadać atrybuty o własności klucza podstawowego. Dopiero odwołanie się w zapytaniu do wartości klucza podstawowego gwarantuje jednoznaczność identyfikacji i pozwala na dostęp do pojedynczych krotek.



W obiektowych bazach danych wprowadzono nowy mechanizm identyfikacji danych. Obiekty są identyfikowane za pomocą systemowego identyfikatora OID. Jest to dodatkowy atrybut każdego obiektu, którego wartości są generowane i utrzymywane w sposób transparentny dla użytkowników przez system zarządzania bazą danych. Wartość tego atrybutu jest generowana w momencie tworzenia każdego obiektu i niemodyfikowalna w ciągu całego życia obiektu. Identyfikatory obiektów są wykorzystywane do implementacji związków między obiektami. Powiązane obiekty pamiętają nawzajem swoje identyfikatory.

Na przykładzie zilustrowano własności identyfikatorów obiektów. Identyfikator obiektu reprezentującego osobę Jana Kowalskiego został zapamiętany w obiekcie reprezentującym żonę Kowalskiego Janinę Kowalską. Mimo zmiany wartości kluczowych atrybutów obiektu, takich jak nazwisko, numer PESEL i płeć, obiekt reprezentujący byłego Kowalskiego nie zmienił swojej tożsamości określonej za pomocą jego identyfikatora.



W obiektowych bazach danych powstałych jako rozwinięcie języków obiektów jest potrzebny dodatkowy mechanizm utrwalania w bazie danych wybranych obiektów tworzonych przez obiektowe aplikacje bazy danych. W świecie systemów relacyjnych programiści tworzący aplikacje bazy danych muszą korzystać z dwóch języków o diametralnie różnej filozofii. Po pierwsze z języka dostępu do trwałych danych składowanych w bazie danych, na przykład języka SQL i po drugie z języka służącego do budowy funkcjonalności i interfejsu aplikacji. Jednym z celów budowy obiektowych baz danych była jednorodność języka służącego do pisania logiki przetwarzania aplikacji i dostępu do bazy danych. Aby ten cel osiągnąć poszukiwane rozwiązania muszą spełniać pięć podstawowych wymagań.

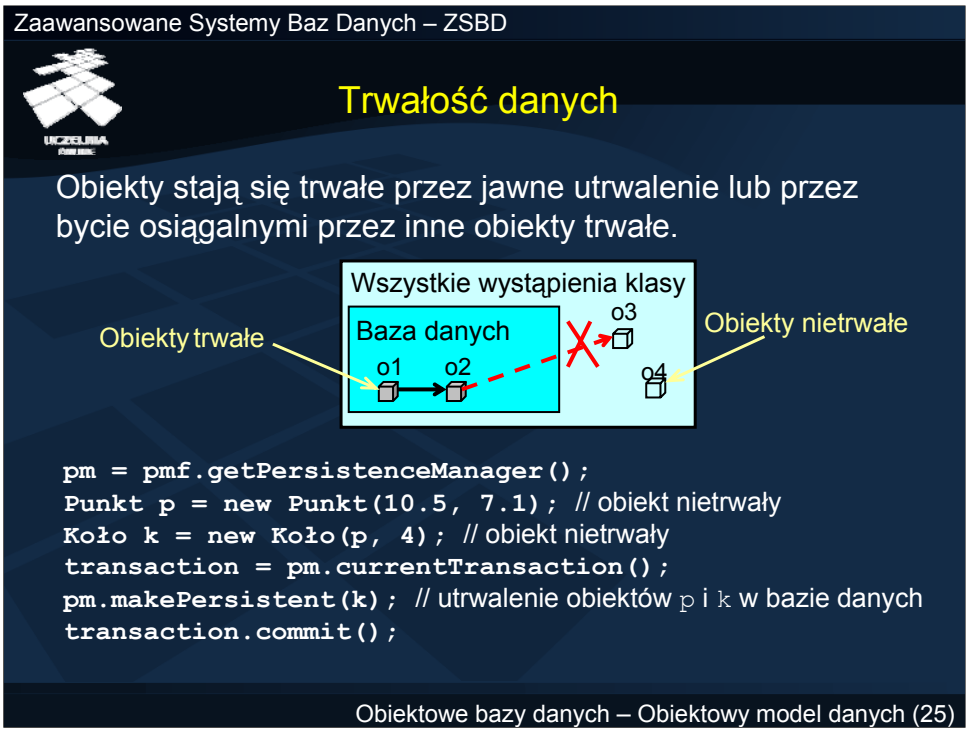
Pierwszym wymaganiem jest by aplikacje baz danych mogły równolegle tworzyć i przetwarzać zarówno zwykłe nietrwałe lokalne obiekty aplikacji, jak i obiekty trwałe, składowane w bazie danych.

Drugie wymaganie postuluje by własność trwałości dotyczyła pojedynczych obiektów, a nie całych klas obiektów. Oznacza, to że wystąpienia tej samej klasy mogą być zarówno trwałe jak i nietrwałe. Wymaganie to istotnie różni modele obiektowy i relacyjny. W modelu relacyjnym równolegle funkcjonują dwa systemy typów danych to jest trwałe relacje i lokalne dane aplikacji.

Trzecie wymaganie ma pozwolić by można było tworzyć trwałe wystąpienia dowolnych klas definiowanych przez użytkowników, a nie tylko wyróżnionych klas systemowych. W praktyce, klasy dla których chcemy tworzyć trwałe wystąpienia, muszą być wywiedzione z wyróżnionych klas systemowych o dodatkowej wymaganej funkcjonalności.

Czwarte wymaganie postuluje by konstrukcje języka obiektowego zastosowanego do budowy aplikacji bazy danych nie rozróżniały obiektów trwałych i nietrwałych. To znaczy, że sposób przetwarzania trwałych i nietrwałych wystąpień tej samej klasy ma być taki sam.

Ostatnim wymaganiem jest by obiekty w ciągu swojego życie mogły wielokrotnie przechodzić między trwałym i nietrwałym obszarem składowania.



W rozwiązaniach wypracowanych przez standardy ODMG i JDO wyróżnia się dwie klasy obiektów trwałych. Do pierwszej klasy należą obiekty, które stają się trwałe poprzez bezpośrednie wywołanie wyspecjalizowanej metody systemowej przesuwającej dany obiekt lub zbiór obiektów z nietrwałego do trwałego obszaru składowania. Na rysunku przykładem takiego obiektu jest obiekt *„o1”*. Do drugiej klasy należą obiekty, które są utrwalane w sposób pośredni, przez powiązanie ich z obiektami trwałymi. Obiekty należące do tej klasy są tak długo trwałe, jak długo są osiągalne z obiektów trwałych. Przykładem obiektu należącego do tej klasy trwałości jest obiekt *„o2”*. Tymczasem obiekt *„o3”*, który przez jakiś czas był trwały, w wyniku usunięcia powiązania z obiektem *„o2”* zostaje przesunięty do nietrwałego obszaru składowania. Jeżeli dodatkowo obiekt ten przestanie być osiągalny przez tymczasowe zmienne aplikacji bazy danych, zostanie z czasem usunięty przez systemowe procesy odśmiecania bazy danych.

Przedstawiony poniżej rysunku przykład wykorzystuje rozwiązania wypracowane w standardzie JDO. Obiekty *„p”* i *„k”* są w momencie utworzenia nietrwałe. Metoda systemowa *„MakePersistent”* przenosi obiekt *„k”* do bazy danych. Obiekt *„k”* należy do pierwszej klasy trwałości. Obiekt *„p”* zostanie również utrwalony przez związek łączący go z obiektem *„k”*. Obiekt *„p”* należy do drugiej klasy trwałości.