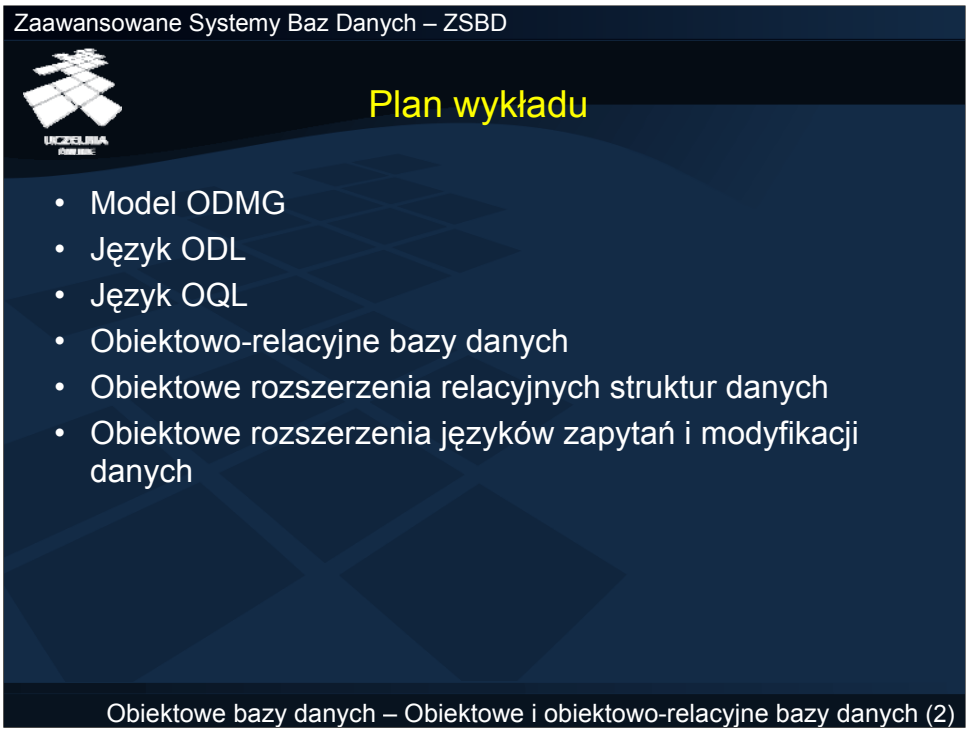
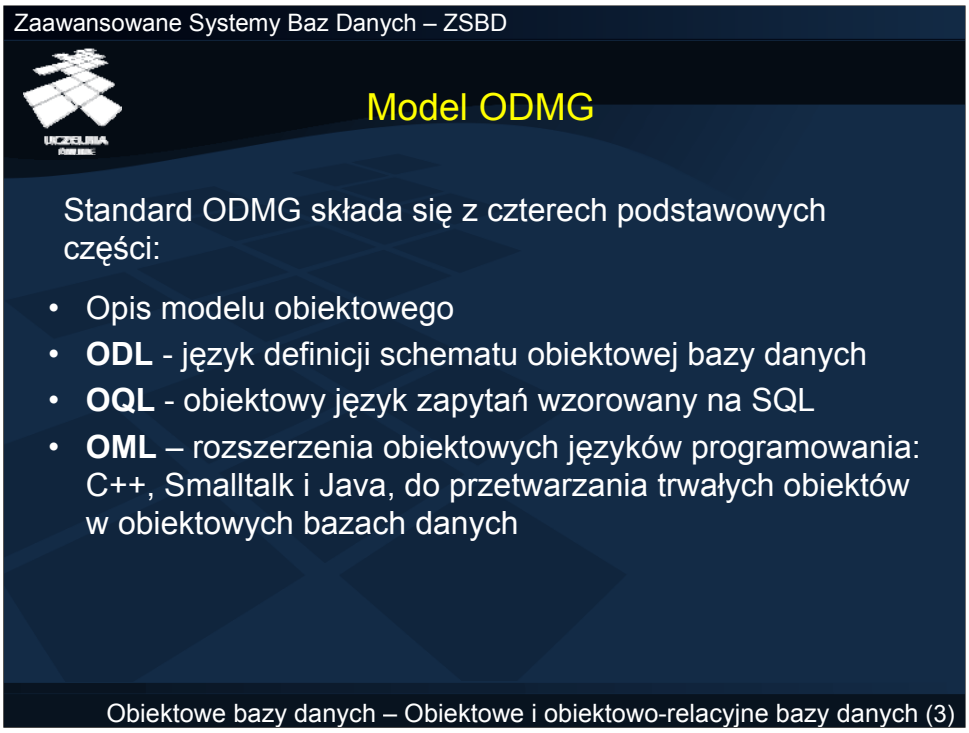
W ramach niniejszego wykładu zostaną przedstawione dwa alternatywne podejścia do budowy systemów obiektowych. Pierwsze z nich, polega na budowie nowego systemu bazy danych całkowicie od podstaw jako systemu obiektowego. Dalej takie systemy będziemy nazywali obiektowymi bazami danych. Drugie podejście, polega na ewolucji relacyjnych baz danych w kierunku modelu o własnościach obiektowych. Takie systemy będziemy nazywali obiektowo-relacyjnymi bazami danych.



Celem wykładu jest poznanie własności modelu danych obiektowych i obiektowo-relacyjnych baz danych. W ramach wykładu poznamy konstrukcje językowe umożliwiające tworzenie schematów baz danych oraz przetwarzanie danych.

Jako wzorzec obiektowych systemów baz danych został wykorzystany standard ODMG 3.0. Przedstawione zostaną dwie podstawowe części tego standardu: język definicji schematu bazy danych ODL oraz wzorowany na języku SQL język zapytań OQL. Z kolei jako przykład klasy obiektowo-relacyjnych systemów baz danych wybrano komercyjny systemem Oracle wzorowany na standardzie SQL3. Przedstawione zostaną konstrukcje językowe służące do tworzenia obiektowych struktur danych oraz do wyszukiwania i przetwarzania obiektów w bazach danych.



W drugiej połowie lat osiemdziesiątych powstało wiele prototypów i komercyjnych produktów obiektowych systemów baz danych tworzonych na bazie języków obiektowych. Modele obiektowe tych systemów różniły się znacznie między sobą. Uważano wtedy, że jest to podstawową przyczyną braku komercyjnych sukcesów obiektowych baz danych. W ramach wysiłków ujednolicenia obiektowego modelu danych powstała inicjatywa standaryzacyjna ukonstytuowana w grupie o nazwie ODMG (ang. Object Database Management Group). W trakcie swojego działania w latach 1993 do 2001 grupa ta tworzyła kolejne wersje standardu. Ostatnim był standard ODMG w wersji 3.0. Obiektowy model zdefiniowany w standardzie ODMG jest wzorcem dla czysto obiektowych baz danych.

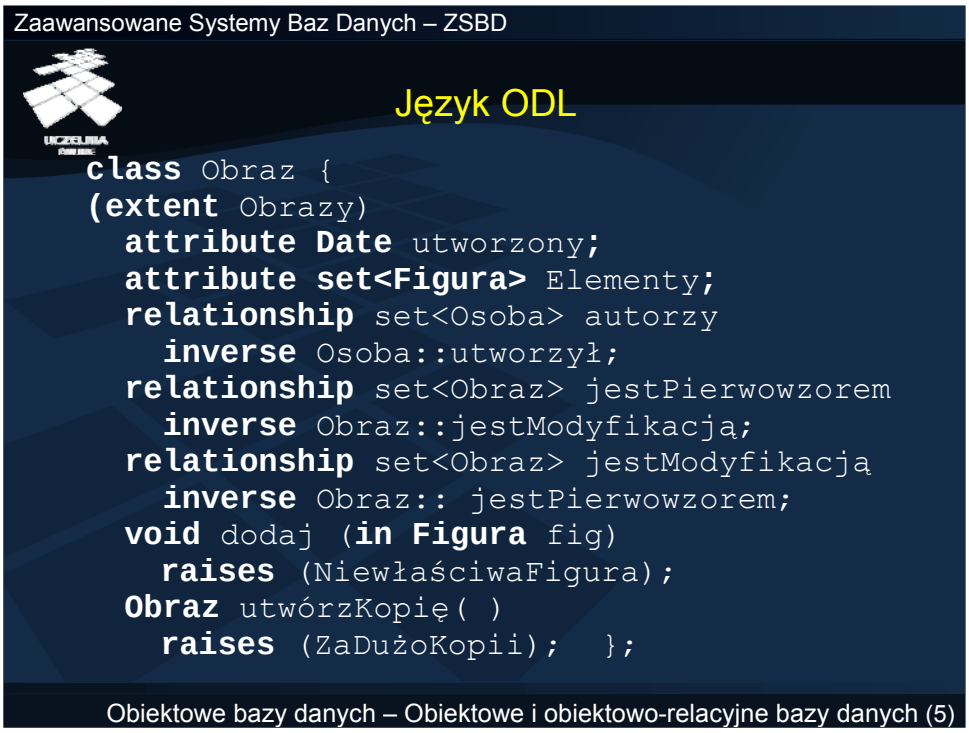
Standard ODMG składa się z czterech podstawowych części. Pierwszą częścią jest słowny opis własności i pojęć modelu obiektowej bazy danych. W stosunku do języków obiektowych model ten zawiera istotne rozszerzenia o własności charakterystyczne dla systemów baz danych. Rozszerzenia te dotyczą pojęć związków między obiektami, tożsamości obiektów, trwałości obiektów, rozszerzeń klas, struktur masowych, przetwarzania transakcyjnego, struktur metadanych, itp.

Druga część standardu zwiera definicję składni i semantyki języka ODL (ang. Object Definition Language) służącego do definiowania klas i interfejsów w schemacie obiektowej bazy danych. Język ODL jest wzorowany na języku IDL wypracowanym w ramach wcześniejszego standardu OMG (ang. Object Management Group) dedykowanego obiektowej integracji systemów informatycznych.

Trzecią częścią standardu jest definicja języka zapytań OQL (ang. Object Query Language). Jest to język wzorowany na języku SQL, ale wzbogacony o wszystkie koncepcje obiektowego modelu danych. W przeciwieństwie do języka SQL język OQL jest ograniczony do instrukcji zapytań. Z założenia nie obejmuje on instrukcji modyfikacji danych, które znajdują się w czwartej części standardu.

Ostatnia, czwarta część standardu obejmuje propozycje rozszerzeń służących do przetwarzania trwałych obiektów w obiektowych bazach danych. Rozszerzenia te są aplikowane do trzech języków obiektowych: C++, SmallTalk i Java.

Prace nad powiązaniem aplikacji baz danych pisanych za pomocą języka Java są kontynuowane w ramach standardu JDO (ang. Java Data Object). W roku 2006 powstała nowa grupa robocza Object Database Technology Working Group (ODBT WG), która zamierza opracować standard modelu danych nowej generacji obiektowych baz danych.



Język ODL nie jest językiem programowania. Służy on do definiowania klas w schemacie obiektowej bazy danych. Definicja funkcjonalności obejmuje specyfikację atrybutów, związków i metod.

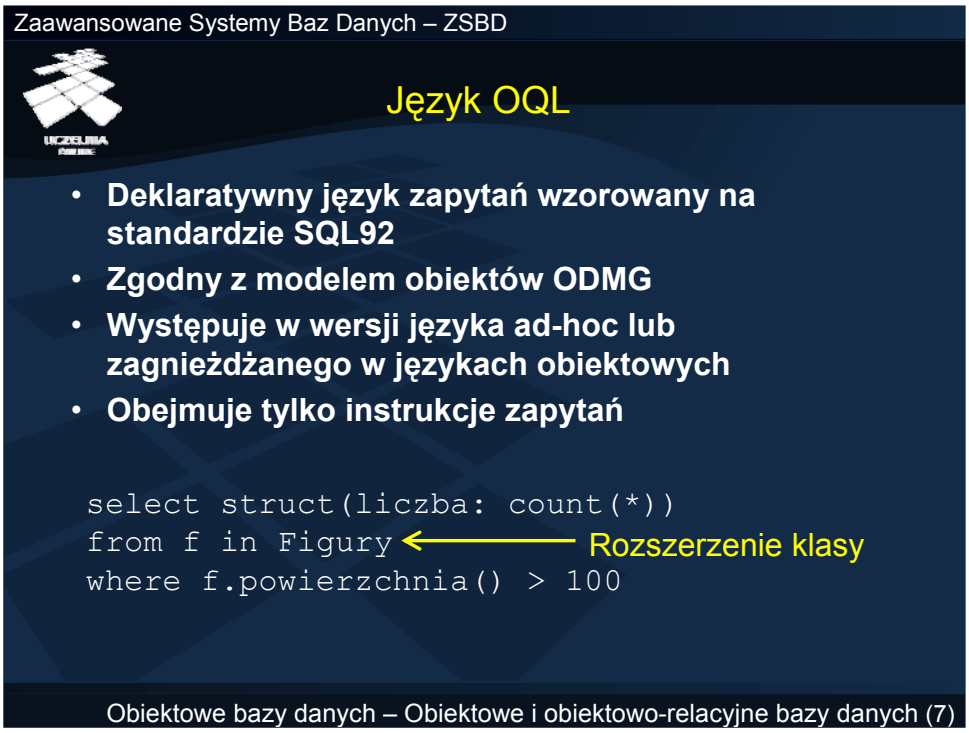
Definicja atrybutu klasy obejmuje nazwę i typ. Typem atrybutu mogą być proste predefiniowane typy danych lub typy złożone. Złożony typ danej może być skonstruowany na jeden z dwóch sposobów. Pierwszy sposób polega na modelowaniu złożonej struktury atrybutu za pomocą konstruktorów typów złożonych: krotki – „*struct*”, zbioru – „*set*”, wielozbioru – „*multiset*”, listy – „*list*”, tablicy - „*array*” i słownika – „*dictionary*”. Drugi sposób do definicji typu atrybutu wykorzystuje klasy zdefiniowane w schemacie bazy danych. Wartościami takich atrybutów będą obiekty tych klas. Atrybuty zdefiniowane w oparciu o klasy pozwalają modelować nie tylko złożoną strukturę atrybutów, ale również zdefiniowaną przez użytkownika nową semantykę.

Przykład zawiera definicję dwóch atrybutów: „*utworzony*” i „*Elementy*”. Pierwszy atrybut jest zdefiniowany na prostym typie danych „*Date*”. Drugi atrybut został zdefiniowany za pomocą konstruktora zbioru – „*set*” i klasy „*Figura*”. Wartościami tego atrybutu będą zbiory obiektów wystąpień klasy „*Figura*”, które nie będą niezależnie przechowywane w bazie danych, ale będą częścią składową obiektów wystąpień klasy „*Obraz*”.

Definicja związku obejmuje nazwę związku, typ związku i odwołanie do związku zwrotnego. Definicja typu związku obejmuje nazwę klasy, z którą obiekty danej klasy będą powiązane oraz krotność związku. Związki wielokrotne są modelowane za pomocą konstruktorów typów, tych samych co atrybuty złożone. Odwołanie do związku zwrotnego zawiera nazwę drugiego końca związku.

języków programowania.

Przedstawiony na slajdzie przykład zawiera dwie metody. Metoda „*dodaj”* nie zwraca wartości, posiada jeden parametr wejściowy „*fig”* typu „*Figura”* i może zgłosić jeden wyjątek o nazwie „*NiewłaściwaFigura”*. Metoda utwórz „*utwórzKopię”* jest bezparametrowa, natomiast zwraca wartości typu „*Obraz”*. Metoda ta może zgłosić wyjątek o nazwie „*ZaDużoKopii”*.

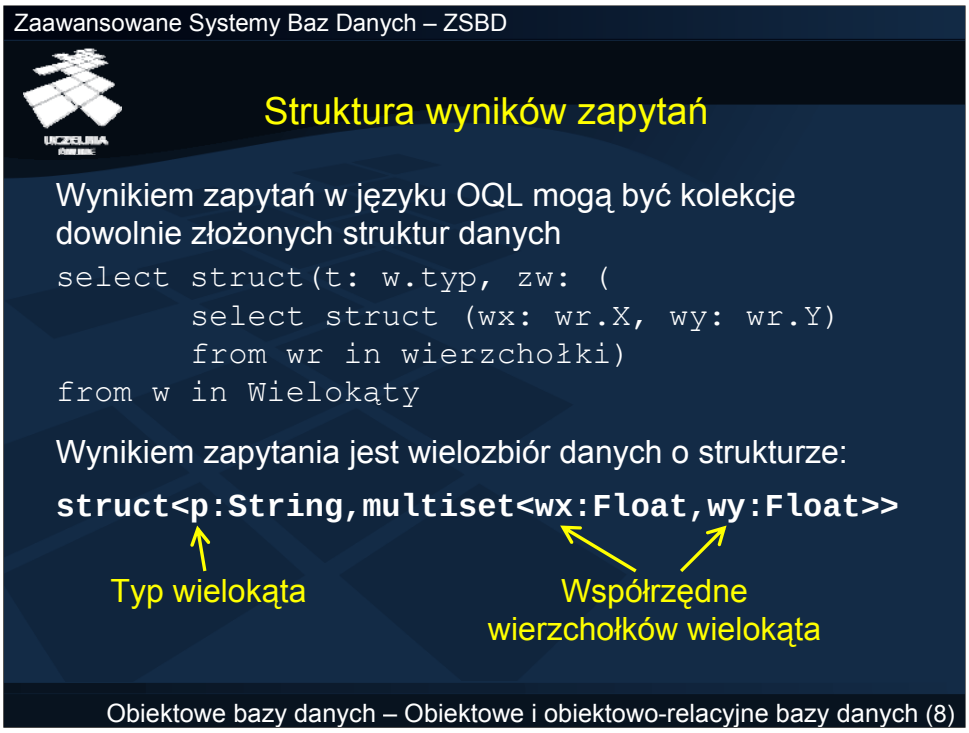


Język OQL jest deklaratywnym językiem zapytań wzorowanym na standardzie SQL92. Język ten powstał by umożliwić wykonywanie w obiektowych bazach danych zapytań ad-hoc. W pierwszych obiektowych systemach baz danych zapytania trzeba było implementować za pomocą języków obiektowych nieprzystosowanych do przetwarzania dużych zbiorów danych.

Język OQL istotnie różni się od klasycznego języka SQL możliwością wyszukiwania złożonych strukturalnie i semantycznie oraz powiązanych danych. Jest on w pełni zgodny z modelem obiektów standardu ODMG. Język ten może być zastosowany na dwa sposoby: jako język ad-hoc interpretujący interakcyjnie przekazywane zapytania i wizualizujący ich wyniki lub jako język zanurzony w środowisku obiektowych języków programowania dla ułatwienia tworzenia aplikacji bazy danych.

Język OQL jest ograniczony do języka zapytań. W przeciwieństwie do języka SQL nie zawiera operacji modyfikacji danych.

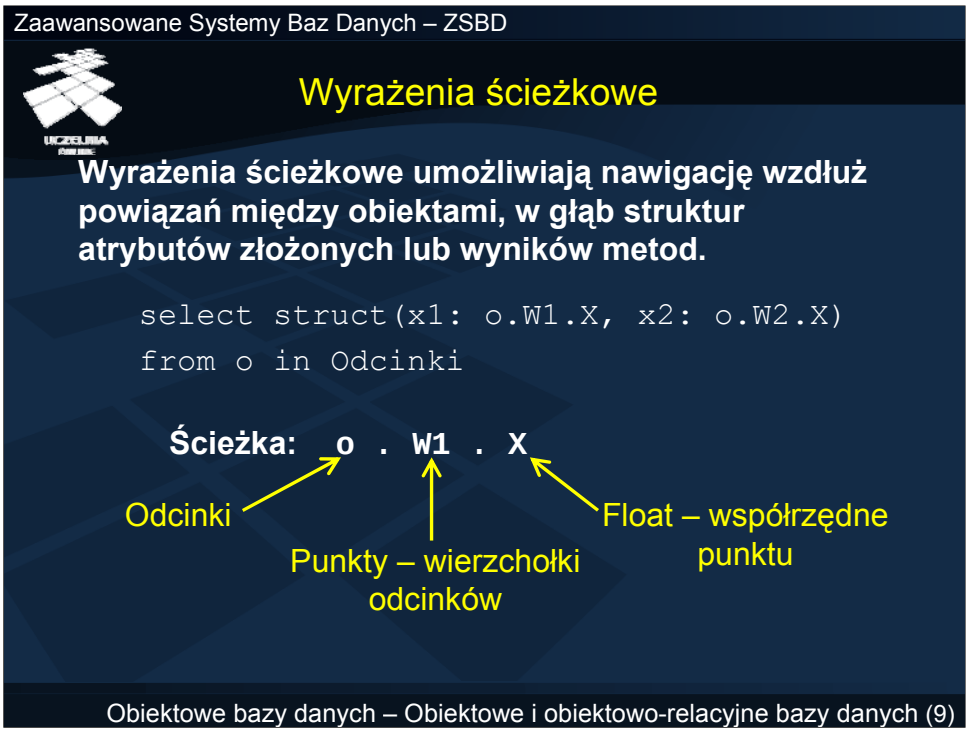
Na slajdzie przedstawiono przykładowe zapytanie w języku OQL. Konstrukcja klauzuli SELECT umożliwia ustalenie na podstawie składni zapytania - struktury wyniku zapytania. Argumentem klauzuli FROM jest nazwa rozszerzenia klasy. Warunek logiczny zawarty w klauzuli WHERE zwiera wywołanie metody „*powierzchnia”* dla obiektów należących do rozszerzenia „*Figury”*.



Wynikiem zapytań wyrażonych w języku OQL są kolekcje obiektów lub wartości. Struktura i funkcjonalność zwracanych obiektów jest zdefiniowana w schemacie bazy danych. Wynikiem mogą być obiekty odczytane z bazy danych lub obiekty dynamicznie utworzone w trakcie wykonywania zapytania za pomocą konstruktora klasy. Struktury wartości wynikowych są dynamicznie definiowane w zapytaniu. Struktura wartości, które są wynikiem zapytania może być dowolnie złożona. Wartościami składowymi złożonej wartości są wartości lub obiekty.

W podanym przykładzie struktura danych wynikowych jest krotką, co wynika z zastosowanego konstruktora typu – „*struct*”. Typem pierwszego atrybutu o nazwie „*t*” jest typ tekstowy – „*String*”. Wynika to z definicji atrybutu o nazwie „*Typ*” zdefiniowanego w klasie „*Wielokąty*”, składowanej w schemacie bazy danych. Drugi atrybut krotki o nazwie „zw” jest złożony. Jego wartościami są wielozbiory par liczb zmiennoprzecinkowych. Wartości tego atrybutu są generowane przez zagnieżdżone pod-zapytanie, które zwraca współrzędne wszystkich wierzchołków danego wielokąta.

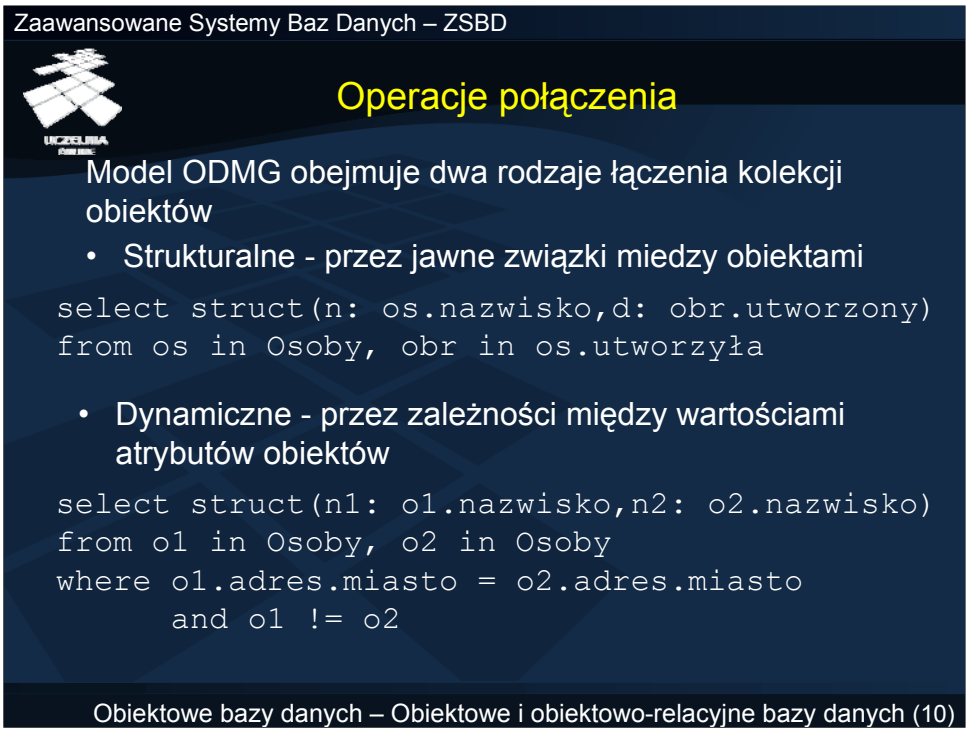
Zmienna "*w*" w zapytaniu reprezentuje obiekty klasy „*Wielokąt*”. Wynikiem zapytania, w którym argumentem klauzuli SELECT byłaby zmienna *w* będzie wielozbiór nie wartości, ale obiektów – wystąpień klasy „*Wielokąt*”.



Możliwość definiowania atrybutów złożonych i związków między obiektami spowodowała pojawienie się zupełnie nowych konstrukcji języka zapytań. Jedną z nowych możliwości są wyrażenia ścieżkowe reprezentujące wieloczłonową operację nawigacji. Wyrażenia ścieżkowe służą do specyfikacji operacji nawigacji w głąb obiektów złożonych, wzdłuż związków łączących obiekty lub do wyników zwróconych przez wywołania metod bezparametrowych. Pojedyncza, dowolnie długa ścieżka, może łączyć te trzy przypadki nawigacji.

Wartością wyrażenia ścieżkowego może być jedynie pojedynczy obiekt lub wartość o typie wynikającym z typu ostatniego elementu ścieżki. Wynika stąd ograniczenie stosowalności wyrażeń ścieżkowych do atrybutów jednowartościowych i związków jednokrotnych. Składniowym operatorem służącym do tworzenia ścieżek może być operator kropki: "." lub strzałki →. Wyrażenia ścieżkowe mogą być używane w klauzulach: SELECT, FROM i WHERE.

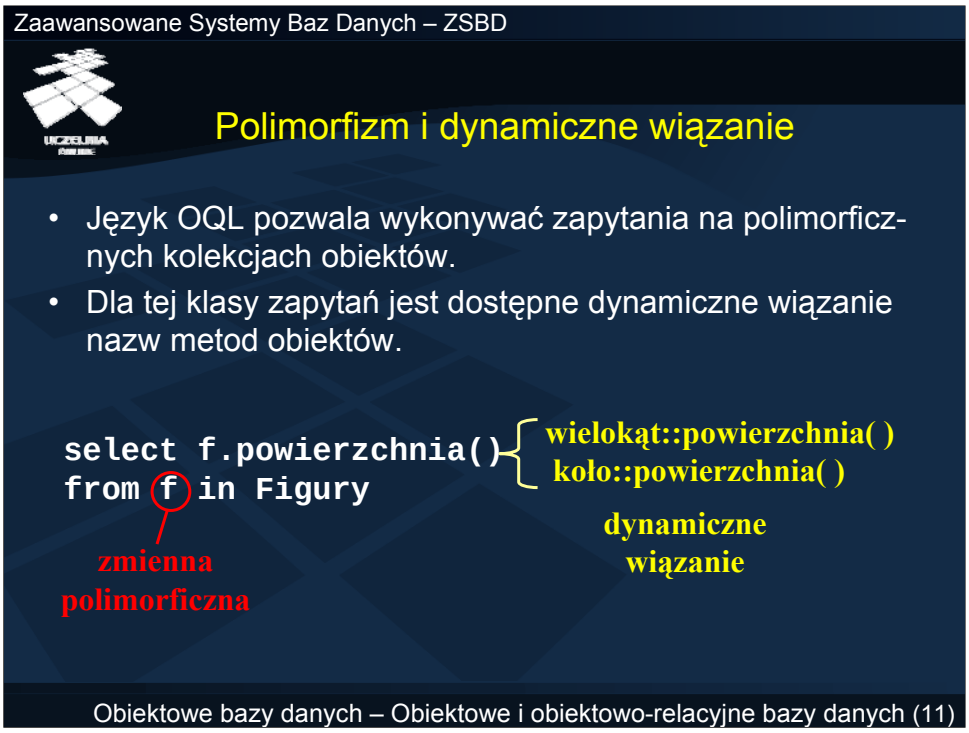
Na slajdzie pokazano przykład wyrażenia ścieżkowego trójczłonowego reprezentującego nawigację w głąb atrybutów złożonych obiektów. Nawigacja przechodzi w głąb obiektów klasy „*Odcinek*” przez wierzchołki odcinków do współrzędnych wierzchołków na osi X. Wynikiem zapytania są pary liczb zmiennoprzecinkowych.



Język OQL rozróżnia dwa rodzaje operacji łączenia (ang. join) kolekcji obiektów. Oprócz znanej z modelu relacyjnego operacji dynamicznego łączenia krotek, na podstawie zdefiniowanego warunku połączeniowego, wprowadzono nową klasę łączenia strukturalnego, na podstawie związków zdefiniowanych w bazie danych. Składniowo połączenia strukturalne są definiowane jako wielowartościowe wyrażenia ścieżkowe umieszczone w klauzuli FROM.

W podanym przykładzie łączenie obiektów rozszerzania „*Osoby”* z obiektami rozszerzenia „*Obrazy”* jest wykonywane przez nawigację wzdłuż związku o nazwie „*utworzyła”*. System zarządzania obiektową bazą danych w tym wypadku nie dopasowuje dynamicznie par obiektów na podstawie warunku połączeniowego, tylko nawiguje wzdłuż już ustalonych powiązań.

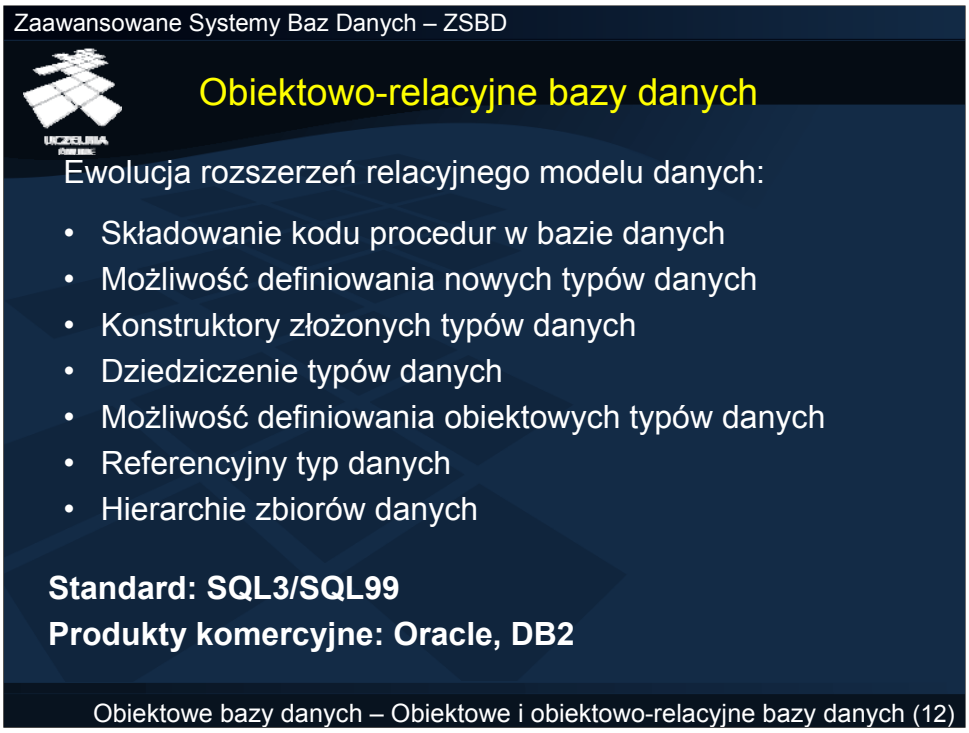
W języku OQL pozostawiono możliwość wykonywania relacyjnych operacji połączenia. Przykładem jest drugie zapytanie, w którym są łączone dwie kolekcje osób na podstawie dynamicznie weryfikowanego warunku zamieszkiwania w tym samym mieście. Dodatkowy warunek połączeniowy „o1” != „o2” wyklucza łączenie tych samych osób.



Język OQL umożliwia wykonywanie zapytań na kolekcjach obiektów polimorficznych. Kolekcje obiektów polimorficznych powstają w konsekwencji specjalizacji klas przez mechanizm dziedziczenia. Jak już to było mówione rozszerzenia klas bazowych obejmują rozszerzenia wszystkich klas pochodnych. W związku z tym, wynikiem zapytań mogą być kolekcje obiektów różnych klas tworzących jedną hierarchię dziedziczenia. Wszystkie te obiekty współdzielą funkcjonalność zdefiniowaną w klasie, która jest korzeniem takiej hierarchii.

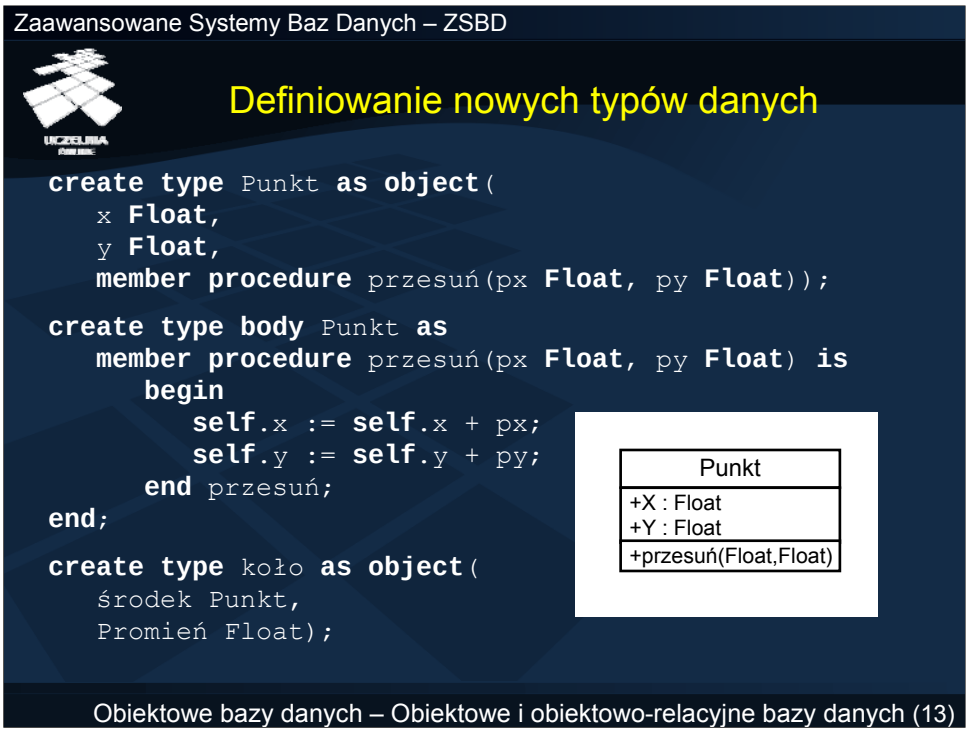
Dodatkowo język OQL realizuje mechanizm dynamicznego wiązania nazw komunikatów z różnymi metodami polimorficznej kolekcji obiektów. Umożliwia to pisanie zapytań, w których przypisanie występującym w zapytaniu komunikatom właściwych metod odbywa się dopiero w trakcie wykonywania tego zapytania. Ponadto, w ramach pojedynczego zapytania przypisanie to może być różne dla różnych polimorficznych obiektów.

W zapytaniu pokazanym na slajdzie jest przetwarzany heterogeniczny zbiór figur. Obiekty, które będą wynikiem zapytania mogą być wystąpieniami klas Wielokąt lub Koło, które różnią się strukturą i implementacją metod. Klasy te różni na przykład sposób wyznaczania powierzchni. Zmienna f zdefiniowana w zapytaniu jest więc zmienną polimorficzną, bo w trakcie wykonywania zapytania przypisywane jej będą zarówno wielokąty, jak i koła. Komunikat powierzchnia() przesyłany do zmiennej f będzie wiązany dynamicznie. Oznacz to, że metoda przypisywana do tego komunikatu będzie zależna od klasy obiektu, który aktualnie jest przypisany zmiennej f. Dla kół metodą tą będzie metoda zdefiniowana w klasie Koło, a dla wielokątów metoda z klasy Wielokąt.



Historia obiektowo-relacyjnych baz danych jest krótsza niż czysto obiektowych. Pierwsze prace na systemami tej klasy rozpoczęły się dopiero w połowie lat dziewięćdziesiątych. Równolegle trwały wysiłki standaryzacyjne, w ramach standardu SQL3 oraz prace nad systemami komercyjnymi, na przykład nad systemem Oracle lub DB2 oraz systemami klasy Open Source na przykład PostgreSQL.

Sposób konstrukcji systemów obiektowo-relacyjnych jest całkowicie odmienny od systemów czysto obiektowych. Punktem wyjścia jest tu zachowanie kompletnej funkcjonalności systemów relacyjnych i ewolucyjne modyfikacje wprowadzające dodatkowo oprócz własności relacyjnych - własności obiektowe. Pierwszym krokiem było wprowadzenie możliwości pamiętania w bazie danych oprócz danych, również procedur, początkowo słabo zintegrowanych z danymi. Następne kroki polegały na wprowadzeniu własności stricte obiektowych, takich jak możliwość definiowania nowych typów danych, dziedziczenia typów danych, definiowanie złożonych struktur danych, typów referencyjnych i hierarchii podzbiorów.



Obiektowo-relacyjny model danych umożliwia definiowanie użytkownikom bazy danych nowych typów danych, czyli odpowiednika klas w systemach czysto obiektowych. Definicja typu danych obejmuje definicję interfejsu typu oraz jego implementację. Interfejsem typu danych jest zbiór atrybutów oraz składnia operacji dostępnych dla danego typu. Implementacja typu danych jest implementacją operacji typu danych. Operacje są implementowane za pomocą proceduralnych języków bazy danych. Współczesne relacyjne bazy danych oferują proceduralne rozszerzenia dla języka SQL, stosowane do implementacji operacji typów danych oraz do implementacji niezależnych funkcji i procedur składowanych w bazie danych. W systemie Oracle proceduralnym językiem bazy danych jest język PL/SQL, w systemie DB2 jest to język o nazwie DB2 SQL.

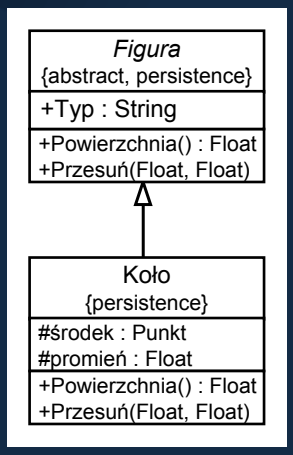
Zdefiniowane przez użytkowników bazy danych nowe typy danych mogą być wykorzystywane do definicji atrybutów innych złożonych typów danych lub do definicji atrybutów schematów relacji. Ten typ danych jest nazywany typem atrybutowym. Wystąpieniami typów atrybutowych nie są pełnoprawne obiekty, gdyż nie posiadają one tożsamości i mogą być przechowywane w bazie danych tylko jako obiekty składowe.

Na slajdzie przedstawiono przykład definicji typu danych „*Punkt*” w systemie obiektowo-relacyjnej bazy danych Oracle. Pierwsza konstrukcja przedstawiona na slajdzie definiuje interfejs typu, który obejmuje atrybuty „*x*” i „*y*” reprezentujące współrzędne punktu oraz składnię operacji „*przesuń*” realizującej przesunięcie punktu na płaszczyźnie o dany wektor. Następna konstrukcja zawiera implementację procedury realizującej operację przesuwania punktów. Ostania, trzecia konstrukcja pokazuje zastosowanie zdefiniowanego typu „*Punkt*” do definicji innego typu „*Koło*”. Typ „*Punkt*” został wykorzystany do definicji środka „*Koła*”.

Dziedziczenie

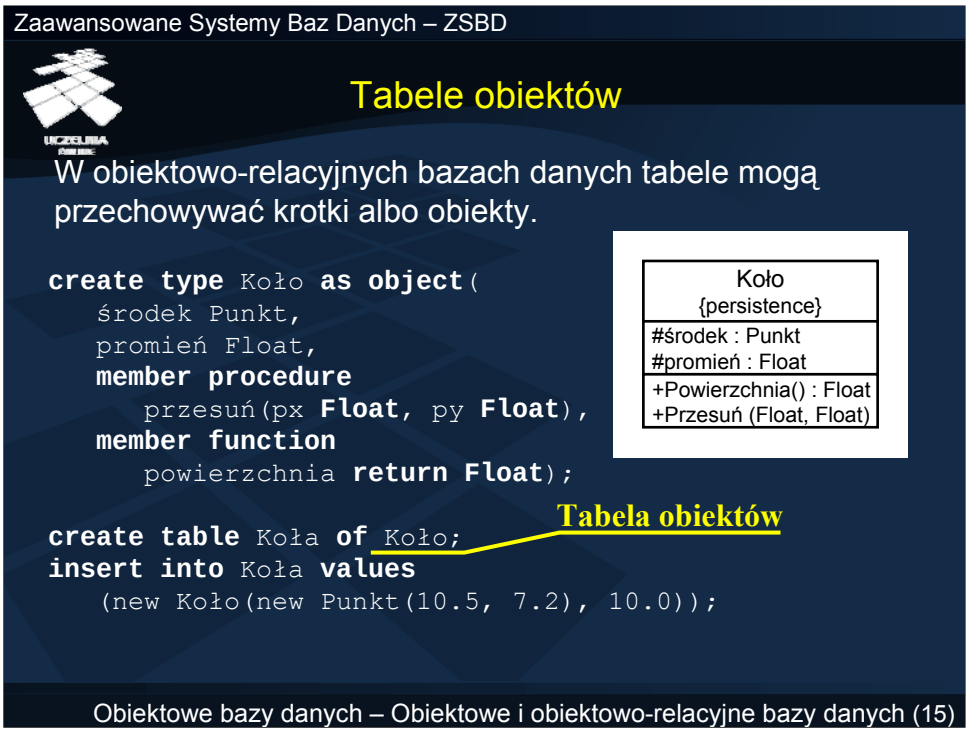
```
create type figura as object (
   typ varchar(10),
   not instantiable member procedure
      przesuń(px Float, py Float),
   not instantiable member function
      powierzchnia return Float)
not instantiable not final;

-- klasa koło dziedziczy po klasie figura
create type koło under figura (
   środek Punkt,
   promień Float,
   overriding member procedure
      przesuń(px Float, py Float),
   overriding member function
      powierzchnia return Float);
```



Zdefiniowane przez użytkownika typy danych mogą być specjalizowane za pomocą mechanizmu dziedziczenia. W typach pochodnych w sposób przyrostowy definiuje się różnice między typem pochodnym i bazowym. W typie pochodnym do cech odziedziczonych z typu bazowego można dodać nowe atrybuty, nowe operacje, redefiniować kod odziedziczonych operacji lub zaimplementować kod odziedziczonej operacji abstrakcyjnej. Operacje abstrakcyjne mają zdefiniowaną składnię, ale nie posiadają implementacji. Ich zastosowaniem są typy danych, które służą jedynie jako typy bazowe dla innych docelowych typów danych.

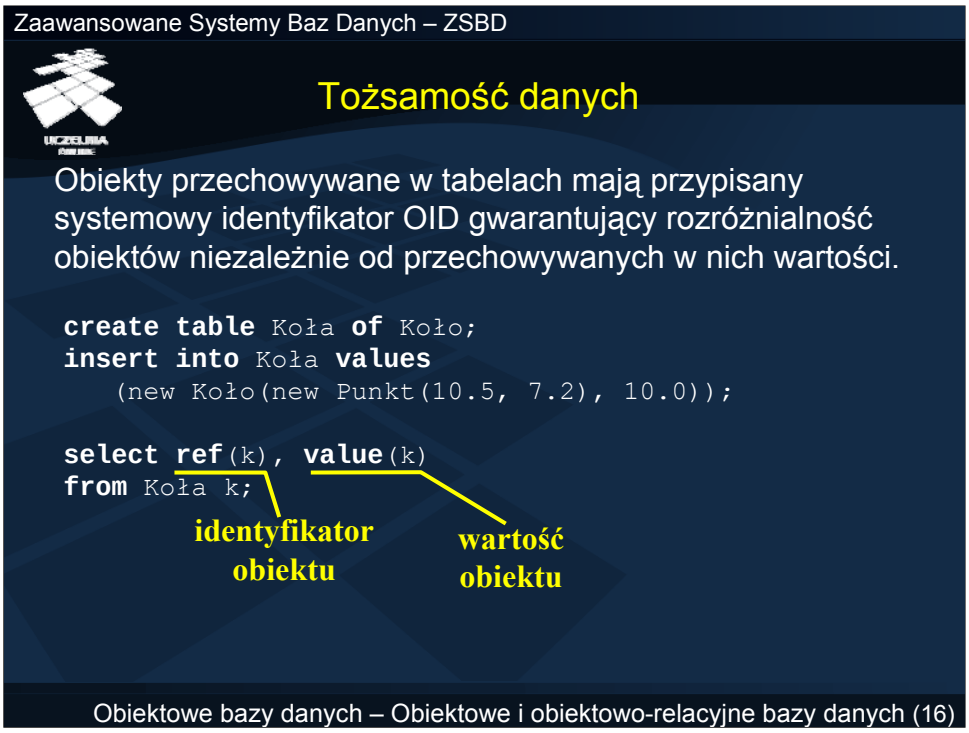
Na slajdzie przedstawiono definicje dwóch typów: typu bazowego „*Figura*” i dziedziczącego po nim typu pochodnego „*Koło*”. Typ bazowy „*Figura*” jest typem abstrakcyjnym, ma dwie abstrakcyjne operacje: „*przesuń*” i „*powierzchnia*”. Abstrakcyjność tych operacji, jak i całego typu deklaruje się w sposób jawny za pomocą słowa kluczowego: „*non instantiable*”. Z kolei słowo kluczowe „*not final*” oznacza, że typ ten może służyć jako typ bazowy. Typ *Koło* dziedziczy po typie „*Figura*” atrybut o nazwie „*typ*” i dwie wymienione metody abstrakcyjne. Definicja typu „*Koło*” zawiera dwa dodatkowe atrybuty: „*środek*” i „*promień*”, oraz implementację odziedziczonych operacji abstrakcyjnych. Za pomocą słowa kluczowego „*overriding*” definicja typu „*Koło*” jawnie informuje o redefinicji odziedziczonych operacji. Na slajdzie nie przedstawiono wymaganej w tym wypadku implementacji operacji.



Typy danych definiowane przez użytkownika bazy danych mogą być wykorzystywane do definiowania schematu całej tabeli. Dla rozróżnienia z typem atrybutowym, typ danych zastosowany do definicji schematu tabeli jest nazywany typem obiektowym lub krotkowym. Rozróżnienie między typami atrybutowymi i obiektowymi wprowadzone przez standard SQL3, nie dotyczy sposobu definiowania typu, ale jego zastosowania. Wartościami typów krotkowych są pełnoprawne obiekty posiadające systemowe identyfikatory obiektów OID, przechowywane jako niezależne dane w tabelach obiektów. Odwołując się do pojęć czysto obiektowych baz danych, tabela obiektów odpowiada rozszerzeniu klasy. Obiektowo-relacyjne bazy danych zawierają więc tabele krotek (relacje) i tabele obiektów.

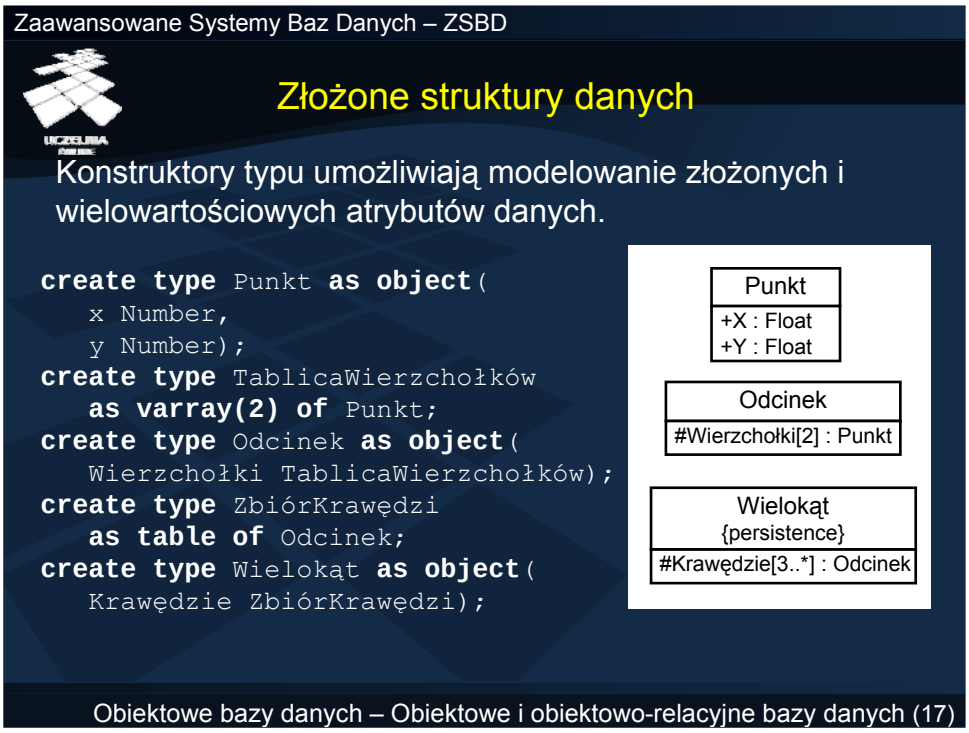
Na slajdzie przedstawiono definicję typu obiektowego *„Koło”*, analogiczną do poprzednich przykładów. Jednak w tym wypadku, typ danych *„Koło”* został użyty do definicji schematu tabeli obiektowej *„Koła”*. Kolejna pokazana instrukcja wstawia do tej tabeli utworzony za pomocą konstruktora obiekt, który jest wystąpieniem typu danych *„Koło”*. Wywołanie konstruktora typu danych *„Koło”* zawiera zagnieżdżone wywołanie konstruktora obiektu składowego, który jest wystąpieniem typu atrybutowego *„Punkt”*.

*) Od tego miejsca zamiast terminu relacja, będzie używany równoważny termin tabela, bo literatura nie zna określenia relacja obiektów na nazwę zbioru trwałych obiektów w bazie danych.



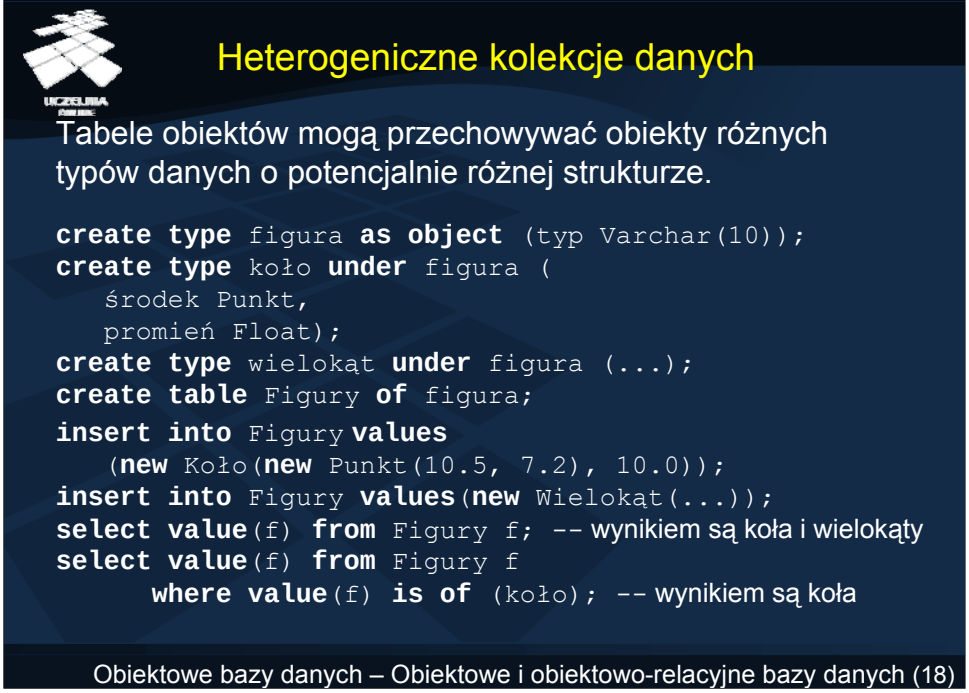
W odróżnieniu od krotek przechowywanych w relacjach, obiekty składowane w tabelach są identyfikowalne nie tylko poprzez wartości atrybutów, ale również przez systemowe identyfikatory obiektów OID. Wartość identyfikatora OID może być odczytana z obiektu za pomocą operatora referencji REF. Dostępna jest również operacja odwrotna, konwersji identyfikatora obiektu na wartość obiektu za pomocą operatora VALUE.

W przykładzie pokazano zastosowanie operatorów referencji i dereferencji. Do tablicy obiektów Koła wstawiono pojedynczy obiekt – koło o środku w punkcie o współrzędnych (10.5, 7.2) i promieniu o długości 10. Wynik zapytania wykonanego na tabeli Koła zawiera identyfikator obiektu i wartość obiektu.



Obiekty składowane w tabelach mogą mieć złożoną strukturę. Standard SQL3 obejmuje zbiór konstruktorów typów analogiczny to tego ze standardu ODMG. W komercyjnych produktach lista konstruktorów typów danych jest trochę uboższa. W systemie Oracle lista ta obejmuje trzy konstruktory: konstruktor krotki odziedziczony po relacyjnym modelu danych, konstruktor zbioru i konstruktor tablicy. W definicji typu danych konstruktory te mogą być wzajemnie zagnieżdżane.

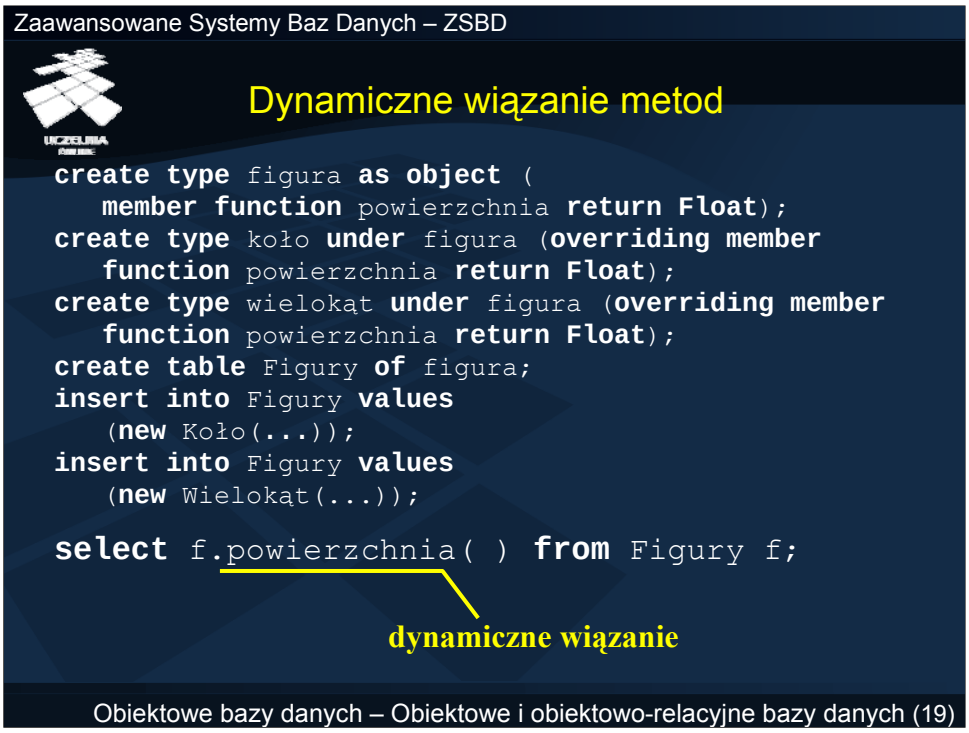
Na slajdzie pokazano zastosowanie konstruktorów typów na przykładzie złożonego typu danych Wielokąt. Najpierw do zdefiniowania typu danych Punkt zastosowano operator krotkowy. Następna definicja wykorzystuje operator tablicy VARRAY, do reprezentacji par punktów, zastosowany do definicji typu Odcinek. Kolejnym operatorem jest operator zbioru TABLE, zastosowany do reprezentacji zbioru odcinków. I na końcu definicja zbioru odcinków jest wykorzystana do modelowania zbioru krawędzi Wielokąta.



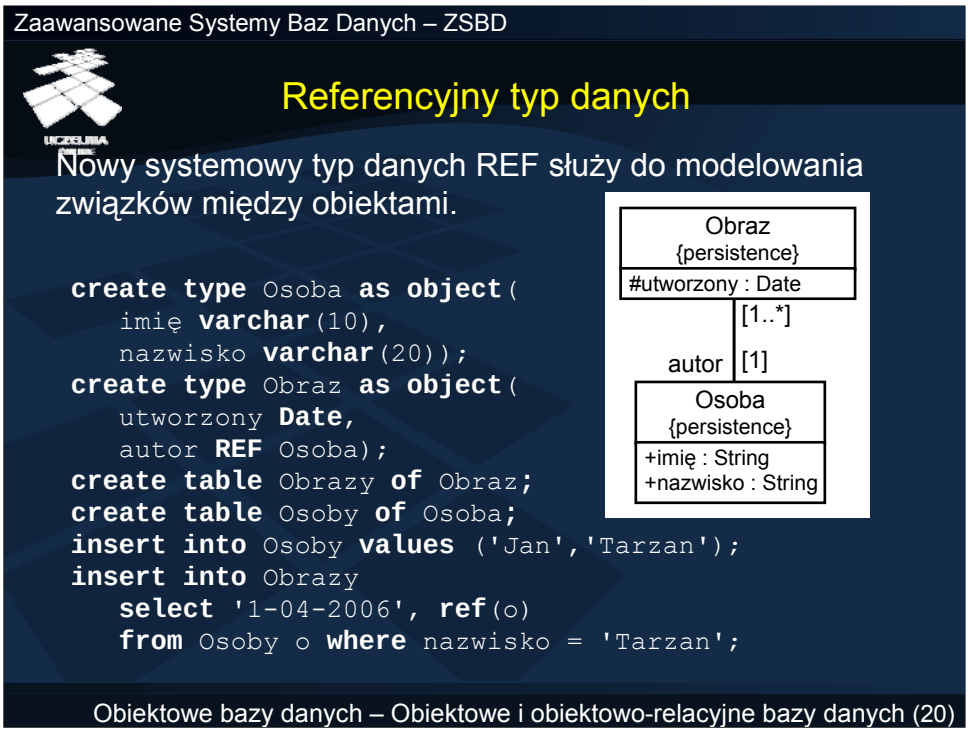
Tabele obiektów, których schemat jest zdefiniowany na podstawie bazowego typu danych z którego są wywiedzione typy pochodne, mogą być heterogeniczne. To znaczy, że mogą przechowywać obiekty polimorficzne. Obiekty przechowywane w tabelach heterogenicznych mogą być wystąpieniami zarówno typu bazowego jak i typów pochodnych. W związku z tym, mogą mieć różną strukturę. Wystąpienia typów pochodnych mogą posiadać dodatkowe atrybuty nie występujące w typie bazowym.

Na slajdzie pokazano przykład tabeli obiektów która może zawierać dwa typy obiektów: koła i wielokąty. Dodatkowo, jeżeli typ figura nie byłby typem abstrakcyjnym, tabela obiektów mogłaby jeszcze zawierać generyczne figury, które nie są ani kołami, ani wielokątami. Wystąpienia typu Figura mają jeden atrybut, a wystąpienia typu Koło dwa dodatkowe atrybuty: środek i promień.

Do heterogenicznej tabeli Figury zdefiniowanej na typie bazowym Figura wstawiono dwa obiekty, pierwszy typu Koło i drugi typu Wielokąt. Następnie na tabeli Figury wykonano dwa zapytania. Wynikiem pierwszego zapytania jest dwuelementowa kolekcja obiektów polimorficznych: jedno koło i jeden wielokąt. Wynik drugiego zapytania jest ograniczony do obiektów typu koło w wyniku zastosowania operatora "*is of*". W naszym konkretnym przypadku wynik zapytania będzie zawierał dokładnie jeden obiekt.



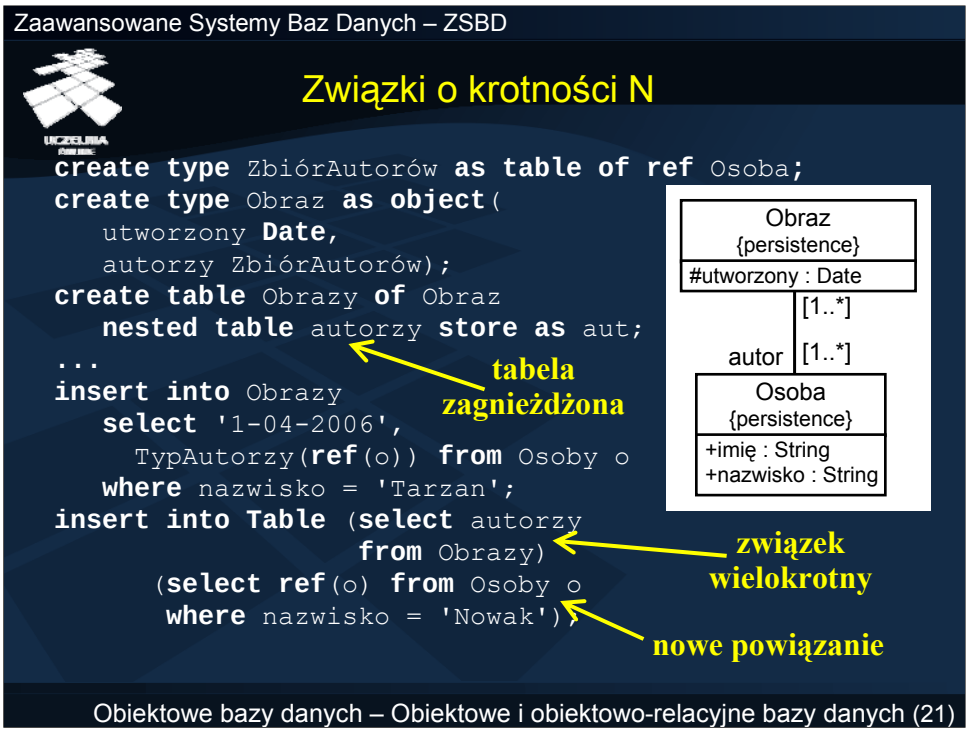
W zapytaniach wykonywanych na heterogenicznych tabelach obiektów można stosować dynamiczne wiązanie nazw operacji z implementującym je kodem, analogicznie jak wiązanie nazw komunikatów z kodem metod w czysto obiektowych bazach danych. Mechanizm ten zilustrowano na przykładzie, który jest rozszerzeniem przykładu z poprzedniego slajdu. Definicje typów Figura, Koło i Wielokąt rozszerzono o operację wyznaczania powierzchni figur. Kod tej operacji zdefiniowany dla bazowego typu Figura jest redefiniowany w typach pochodnych Koło i Wielokąt. Do heterogenicznej tabeli obiektów wstawiono dwa polimorficzne obiekty: koło i wielokąt. W zapytaniu wykonywanym na tabeli Figury wywołanie operacji powierzchnia( ) jest dynamicznie wiązane odpowiednio do typu obiektu, raz do kodu operacji powierzchnia( ) zdefiniowanej dla typu Koło i raz do kodu operacji powierzchnia( ) zdefiniowanej dla typu Wielokąt.



W obiektowo-relacyjnych bazach danych związki między obiektami są modelowane za pomocą nowego systemowego typu danych – typu referencyjnego. Wartościami referencyjnego typu danych są identyfikatory obiektów. Referencyjny typ danych służy do przechowywania identyfikatorów innych obiektów powiązanych z danym obiektem. Mechanizm referencyjnych typów danych w przeciwieństwie do związków w czysto obiektowym modelu danych reprezentuje jedynie związki jednokierunkowe. Modelowanie związków dwukierunkowych wymaga odrębnego użycia dwóch różnych referencji.

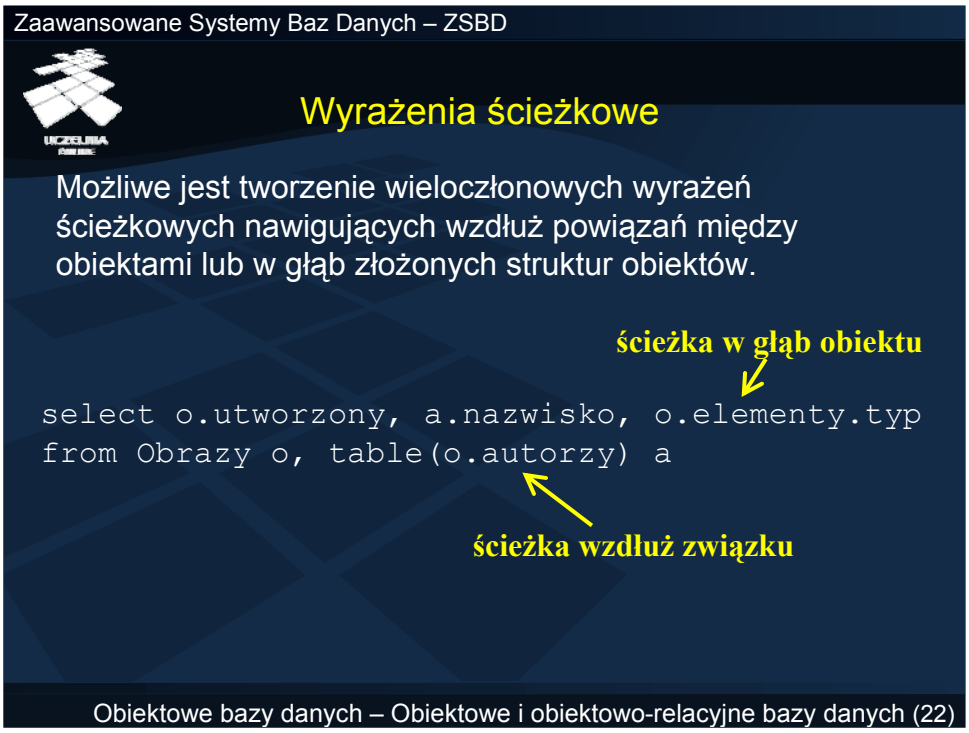
Na slajdzie pokazano przykład jednokierunkowego związki jednokrotnego łączącego obiekty typu Obraz z wystąpieniami typu Osoba. Definicja typy Obraz obejmuje atrybut autor zdefiniowany na referencyjnym typie danych. Wartościami tego atrybutu będą identyfikatory obiektów typu Osoba.

Na bazie typów Obraz i Osoba utworzono tabele obiektów: Obrazy i Osoby. Następnie do tabeli Osoby wstawiono obiekt reprezentujący Jana Tarzana, a do tabeli Obrazy wstawiono obiekt reprezentujący obraz utworzony w dniu 1-IV-2006 roku utworzony przez Jana Tarzana. Zapytanie zagnieżdżone w instrukcji INSERT wydobywa z tabeli osób identyfikator obiektu reprezentującego Jana Tarzana i umieszcza go w atrybucie referencyjnym.



Zamodelowanie związków wielokrotnych wymaga połączenia koncepcji konstruktora zbioru i typu referencyjnego. Ilustruje to przykład pokazany na slajdzie, w którym obraz może mieć więcej niż jednego autora. Do przechowywania zbioru identyfikatorów skonstruowano typ „*ZbiórAutorów*”, którego struktura służy do przechowywania zbioru identyfikatorów Osób. Typ został następnie użyty do zdefiniowania atrybutu autorzy w ramach typu „*Obraz*”.

Definicja tabeli obiektów „*Obrazy*” obejmuje konstrukcję definiującą zagnieżdżoną tabelę do przechowywania wielokrotnych referencji na autorów dla każdego z obrazów. Pierwsza z operacji wstawienia dotyczy tabeli „*Obrazy*”. Wstawiany obiekt zawiera pojedynczą referencję na osobę Tarzana. Kolejna operacja wstawiania dotyczy wielowartościowego atrybutu referencyjnego. Do zbioru autorów obrazu oprócz Tarzana jest dodawany identyfikator kolejnego autora (lub autorów) o nazwisku Nowak. Druga operacją INSERT nie tworzy więc nowego obiektu w tabeli „*Obrazy*”, ale dodaje kolejne powiązanie do związku wielokrotnego.



Atrybuty złożone i typy referencyjne umożliwiają wykonywanie operacji nawigacji w głąb obiektu lub wzdłuż referencji między obiektami. Opisujące takie nawigacje wyrażenia ścieżkowe mogą być wieloczłonowe i mogą być użyte w klauzulach: SELECT, FROM i WHERE.

Na slajdzie pokazano dwa wyrażenia ścieżkowe. Pierwsze w klauzuli SELECT nawiguje w głąb atrybutu złożonego, od obiektu typu Obraz do figur, które są elementami składowymi obrazu, i dalej do atrybut prostego typ figury.

Drugie wyrażenie ścieżkowe umieszczone w klauzuli FROM, służy do realizacji połączenia strukturalnego, analogicznie jak w modelu czysto obiektowym. Ścieżka nawigacji przechodzi w tym wypadku od obiektów z tabeli „*Obrazy*” wzdłuż wielokrotnych referencji związku autorzy do obiektów tabeli „*Osoby*”. Uzyskane w ten sposób połączenie łączy obrazy z ich autorami.