

Na prawach rękopisu

ZAKŁAD SYSTEMÓW INFORMACYJNYCH
WYDZIAŁ INFORMATYKI I ZARZĄDZANIA
POLITECHNIKA WROCŁAWSKA

Raport serii PRE nr 34

DYNAMIKA DANYCH W MODELU OBIEKTOWYM

rozprawa doktorska

Dariusz Król

Promotor: prof. dr hab. inż. Czesław Daniłowicz

Słowa kluczowe: dynamika danych,
 migracja danych,
 model danych,
 obiektowa baza danych

WROCŁAW 2001

Składam serdeczne podziękowania promotorowi Panu prof. Czesławowi Daniłowiczowi za niezwykle cierpliwość, życzliwość i pomoc okazaną mi przy przygotowywaniu niniejszej pracy.

Specjalne podziękowania należą się dr. inż. Ngoc Thanh Nguyenowi za cenne uwagi.

Wyrażam wdzięczność wszystkim pracownikom Zakładu Systemów Informacyjnych za miłą i przyjemną atmosferę pracy.

Dziękuję także mojej żonie za wyrozumiałość i wykazaną samodzielność w wychowywaniu Krzysia, Alicji i najmniejszego Maciusia.

SPIS TREŚCI

STRESZCZENIE	6
---------------------	----------

WAŻNIEJSZE OZNACZENIA I SKRÓTY	7
---------------------------------------	----------

OZNACZENIA ZWIĄZANE Z OBIEKTEM	7
OZNACZENIA ZWIĄZANE Z ALGEBRĄ OBIEKTÓW	7
OZNACZENIA ZWIĄZANE Z MIGRACJĄ OBIEKTÓW	8
SKRÓTY ZWIĄZANE Z BAZAMI DANYCH	8

WPROWADZENIE	10
---------------------	-----------

1. DYNAMIKA W OBIEKTOWYCH BAZACH DANYCH	14
--	-----------

1.1. ZMIANA SCHEMATU BAZY DANYCH	14
1.2. WERSJONOWANIE	15
1.3. EWOLUCJA OBIEKTÓW	15
1.4. MIGRACJA OBIEKTÓW	16
1.5. DZIEDZICZENIE	16
1.6. PERSPEKTYWY	18
1.7. MECHANIZM RÓL	18
1.8. STAN OBIEKTU I CZAS	19
1.9. WNIOSKI	19

2. ALGEBRA OBIEKTÓW WIELOKLASOWYCH	21
---	-----------

2.1. MODEL DANYCH	21
2.1.1. Wieloklasowość obiektów	22
2.1.2. Obiekt wieloklasowy	23
2.1.3. Przykładowa baza obiektów	28
2.1.4. Równość obiektów wieloklasowych	30
2.1.5. Związki między obiektami wieloklasowymi	31
2.1.6. Schemat obiektów, baza obiektów	32
2.2. ALGEBRA OBIEKTÓW	32
2.2.1. Selekcja	33
2.2.2. Asocjacja	33
2.2.3. Projekcja	34
2.2.4. Złączenie	35
2.2.5. Suma, różnica i przecięcie	35
2.2.6. Spłaszczenie	36
2.2.7. Zagnieżdżenie, rozgnieżdżenie	37
2.2.8. Usunięcie duplikatów	38
2.3. WNIOSKI	38

3.	<u>MIGRACJA OBIEKTÓW</u>	40
3.1.	PODSTAWOWE ZMIANY W UNISQL	40
3.2.	OPERATORY ZMIAN – PODEJŚCIE ALGEBRAICZNE	42
3.3.	WŁASNOŚCI MIGRACJI OBIEKTÓW	45
3.3.1.	Schemat migracji obiektów	46
3.3.2.	Podstawowe operacje migracji obiektów	48
3.4.	ALGORYTMY OPERATORÓW MIGRACJI	51
3.4.1.	Algorytm realizacji operatora sprawdź	51
3.4.2.	Algorytm realizacji operatora dodaj	52
3.4.3.	Algorytm realizacji operatora usuń	53
3.4.4.	Algorytm realizacji operatora przenieś	53
3.5.	WŁASNOŚCI OPERATORÓW MIGRACJI	54
3.6.	ROZSZERZONE OPERATORY MIGRACJI	55
3.6.1.	Algorytm realizacji operatora przenieś+	55
3.6.2.	Algorytm realizacji operatora dodaj+	56
3.7.	WNIOSKI	57
4.	<u>INTEGRALNOŚĆ OBIEKTÓW WIELOKLASOWYCH</u>	58
4.1.	OGRANICZENIA DZIEDZINY	60
4.2.	OGRANICZENIA ZWIĄZKU OBIEKT-PODOBIĘT	61
4.2.1.	Wzajemne wykluczanie obiektów	62
4.2.2.	Wzajemne występowanie obiektów	62
4.3.	ZALEŻNOŚCI MIĘDZY OGRANICZENIAMI	63
4.4.	OBIEKT-OGRANICZENIE	65
4.5.	PRZYKŁADY OBIEKTÓW-OGRANICZEŃ	67
4.6.	INTEGRALNOŚĆ MIGRACJI OBIEKTÓW	69
4.7.	WNIOSKI	71
5.	<u>JĘZYK MANIPULACJI OBIEKTAMI WIELOKLASOWYMI</u>	72
5.1.	WYRAŻENIA ŚCIEŻKOWE	73
5.2.	INSTRUKCJA WYSZUKIWANIA OBIEKTÓW	74
5.2.1.	Klauzula celu	74
5.2.2.	Klauzula zakresu	75
5.2.3.	Klauzula kwalifikacji	75
5.2.4.	Operatory mnogościowe i agregujące	75
5.3.	INSTRUKCJA WPROWADZANIA OBIEKTÓW	76
5.4.	INSTRUKCJA USUWANIA OBIEKTÓW	77
5.5.	INSTRUKCJA AKTUALIZACJI OBIEKTÓW	77
5.6.	INSTRUKCJE KONFIGURACJI MIGRACJI OBIEKTÓW	79
5.7.	PRZYKŁADOWA BAZA UTWORZONA W JĘZYKU MOW	79
5.8.	TRANSLACJA INSTRUKCJI JĘZYKA MOW DO WYRAŻEŃ ALGEBRY OBIEKTÓW	80
5.8.1.	Translacja instrukcji wprowadzania obiektów	80
5.8.2.	Translacja instrukcji usuwania obiektów	81
5.8.3.	Translacja instrukcji aktualizacji obiektów	81

5.9. WNIOSKI	82
6. ASPEKTY IMPLEMENTACYJNE MODELU	83
6.1. REPREZENTACJA FIZYCZNA OBIEKTU WIELOKLASOWEGO I BAZY DANYCH	83
6.2. IMPLEMENTACJA ZAPYTAŃ JĘZYKA MOW	84
6.2.1. Obiekty wieloklasowe	85
6.2.2. Związek obiekt-podobiekt	85
6.3. WNIOSKI	86
7. PODSUMOWANIE	87
7.1. KIERUNKI DALSZYCH BADAŃ	89
BIBLIOGRAFIA	91
ZAŁĄCZNIK	105

Streszczenie

Rozprawa jest poświęcona dynamice danych w obiektowej bazie danych. Celem rozprawy jest opracowanie modelu obiektów wieloklasowych odzwierciedlającego dynamikę danych. Bazą analizowanych problemów są możliwości zmiany struktury i zachowania obiektu, możliwości zmiany miejsca pobytu obiektu oraz możliwości zmiany powiązań z innymi obiektami. Wymienione cechy odnajdujemy w klasycznym podejściu obiektowym pod postacią czasu, stanu obiektu, migracji obiektów, obiektów kooperacyjnych, zmiany schematu bazy danych, perspektyw, wersjonowania, dziedziczenia klas, ról i ewolucji obiektów. W pracy szczegółowo przeanalizowano dwa aspekty dynamiki obiektów. Pierwszym z nich jest wieloklasowość obiektów, a drugim migracja obiektów wieloklasowych.

Zmodyfikowana definicja obiektu poza identyfikatorem, wartością i zbiorem metod została rozszerzona o zbiór identyfikatorów nadobiektów, zbiór identyfikatorów podobiektów i zbiór atrybutów dla podobiektów. Taka struktura zapewnia homogeniczność obiektów, klas i ograniczeń. Zależności między obiektami wyraża związek obiekt-podobiekt, który jest uogólnieniem klasycznego związku dziedziczenia. Ze względu na wieloklasowość obiektu wprowadzono pojęcie pochodzenia atrybutu oraz pojęcia sumy i przecięcia atrybutów.

Opisana algebra obiektów zawiera operatory znane z algebry relacyjnej i algebry obiektów złożonych, które zostały zaadaptowane do wymagań stawianych przed obiektami wieloklasowymi. Do zbioru operatorów należą: selekcja, asocjacja, projekcja, złączenie, suma, przecięcie, różnica, spłaszczenie, zagnieżdżenie, rozgnieżdżenie i usunięcie duplikatów. Wykazano, że za pomocą operatorów algebry obiektów można dokonać następujących zmian w obiekcie: dodać lub usunąć atrybut, dodać lub usunąć metodę, dodać lub usunąć obiekt oraz dodać obiekt do lub usunąć obiekt ze zbioru podobiektów lub zbioru nadobiektów. Przedyskutowano konsekwencje, jakie wieloklasowość wprowadza do migracji danych. Podano algorytmy dla operatorów migracji obiektów: dodaj, usuń i przenieś.

Przeanalizowano możliwe ograniczenia obiektów i wprowadzono obiekt-ograniczenie z następującymi atrybutami: Dotyczy, Nie_dotyczy, Przesłania oraz Przesłaniane_przez.

Język manipulacji obiektami wieloklasowymi (MOW) jest językiem bazującym na operatorach algebry obiektów. MOW jest rozszerzeniem typowego strukturalnego języka SQL uwzględniającym mechanizmy wynikające z migracji. Podano sposób transformacji instrukcji języka MOW do wyrażeń algebry obiektów wieloklasowych.

W pracy zaproponowano struktury implementacyjne dla obiektu wieloklasowego i bazy danych.

W podsumowaniu wskazano najważniejsze kierunki dalszych badań.

Ważniejsze oznaczenia i skróty

Oznaczenia związane z obiektem

$\oplus$	operator sumy atrybutów
$\otimes$	operator przecięcia atrybutów
$=$	operator identyczności obiektów
$\cong$	operator równości powierzchniowej obiektów
$\angle$	związek obiekt-podobiekt
$\leq$	związek dziedziczenia
$A(x)$	zbiór atrybutów obiektu x
$\bar{A}(x)$	zbiór atrybutów lokalnych obiektu x
$A_{\text{Sub}}(x)$	zbiór atrybutów opisujących podobiektu obiektu x
$\bar{A}_{\text{Sub}}(x)$	zbiór atrybutów lokalnych opisujących podobiektu obiektu x
$\text{id}(x)$	identyfikator lub nazwa obiektu x
$M(x)$	zbiór metod obiektu x
$O(x)$	zbiór ograniczeń obiektu x
OBJ	zbiór obiektów
OGR	zbiór ograniczeń
$\text{Sub}(x)$	zbiór identyfikatorów podobiektów obiektu x
$\text{Sup}(x)$	zbiór identyfikatorów nadobiektów obiektu x
$v(x)$	wartość obiektu x
$V(x)$	zbiór wartości atrybutów obiektu x
$V_a(x)$	zbiór wartości atrybutu a obiektu x
$x_1.x_2....x_n.A$	wyrażenie ścieżkowe

Oznaczenia związane z algebrą obiektów

σ	operator selekcji
τ	operator asocjacji
π	operator projekcji
χ	operator złączenia

$\cup$	operator sumy
$-$	operator różnicy
$\cap$	operator przecięcia
ε	operator spłaszczenia
γ	operator zagnieżdżenia
μ	operator rozgnieżdżenia
θ	operator usunięcia duplikatów

Oznaczenia związane z migracją obiektów

$\text{id}(x_1), \text{id}(x_2), \dots, \text{id}(x_n) \xrightarrow[\text{przenieś}]{\text{id}(x)} \text{id}(y_1), \text{id}(y_2), \dots, \text{id}(y_m)$	operator przenieś
$\xrightarrow[\text{dodaj}]{\text{id}(x)} \text{id}(y_1), \text{id}(y_2), \dots, \text{id}(y_m)$	operator dodaj
$\text{id}(x_1), \text{id}(x_2), \dots, \text{id}(x_n) \xrightarrow[\text{usuń}]{\text{id}(x)}$	operator usuń
$\text{id}(x_1), \text{id}(x_2), \dots, \text{id}(x_n) \xrightarrow[\text{sprawdź}]{\text{id}(x)} \text{id}(y_1), \text{id}(y_2), \dots, \text{id}(y_m)$	operator sprawdź
$\equiv$	operator równoważności

Skróty związane z bazami danych

ACM	<i>Association for Computing Machinery</i>
ANSI	<i>American National Standards Institute</i>
BNF	<i>Backus-Naur form</i>
DBMS	<i>database management system</i>
DDL	<i>data definition language</i>
DML	<i>data manipulation language</i>
ISO	<i>International Organization for Standardization</i>
NF ²	<i>non-first normal form</i>
OID	<i>object identifier</i>
OMG	<i>Object Management Group</i>
OODB	<i>object-oriented database</i>
OOPL	<i>object-oriented programming language</i>

OSQL	<i>object structured query language</i>
RDBMS	<i>relational database management system</i>
SIGMOD	<i>Special Interest Group on Management of Data</i>
SQL	<i>structured query language</i>
UML	<i>unified modeling language</i>
VLDB	<i>very large data bases</i>
X3H2	<i>ANSI database committee</i>

Wprowadzenie

Obserwując dotychczasowy rozwój w dziedzinie systemów baz danych można stwierdzić, że upływa co najmniej 15 lat od pierwszej publikacji dotyczącej nowych idei do chwili, w której nowa teoria zajmie przysługujące jej miejsce. Obiektowo-zorientowane języki programowania pojawiły się w latach siedemdziesiątych. W roku 1983 na konferencji SIGMOD odbyła się pierwsza specjalna sesja poświęcona nierelacyjnemu podejściu, a mianowicie semantycznym modelom danych. Jednakże dopiero w 1989 roku referat „The Object-Oriented Database System Manifest” [Atk89] dostarczył podstaw do tworzenia obiektowo-zorientowanych baz danych. W tym samym roku powstała grupa robocza OODBTG, skupiająca specjalistów z dziedziny baz danych, mająca na celu opracowanie jednolitych zasad dla systemów obiektowo-zorientowanych. Zarówno terminologia, jak i elementy modelu, opisane w rozprawie, będą bazować na rezultatach prac tej grupy [Mos93, Tho93] oraz na pracach [Abi95, Bha91, Dog94, Kim96 i Nie89].

Ze względu na fakt, iż obiektowo-zorientowane bazy danych pojawiły się w końcu lat osiemdziesiątych wydaje się być oczywistym, że nie można jeszcze dzisiaj oczekiwać pełnej teorii modelu obiektowego. Analizując aktualny stan teorii i praktyki obiektowego podejścia, można stwierdzić, że duża liczba specjalistów w świecie pracuje nad poszczególnymi aspektami lub też całościowymi projektami systemów z obiektową bazą danych. Jako rezultat tych prac niech posłużą pierwsze komercyjne obiektowo-zorientowane bazy danych, np. Object Store, O₂ czy UniSQL.

Do tej pory podstawowym obszarem zastosowań obiektowych baz danych było komputerowe wspomaganie projektowania (*computer-aided design*), komputerowe wspomaganie wytwarzania (*computer-aided manufacturing*), komputerowe wspomaganie inżynierii oprogramowania (*computer-aided software engineering*) oraz automatyzacja przetwarzania dokumentów (*office information system*). Dziedziny te charakteryzują się dużą ilością oraz złożonością przetwarzanych danych. Tradycyjne podejście powoduje w takich przypadkach niepotrzebną redundancję. Dla wymienionych obszarów podejście obiektowe jest najlepszym rozwiązaniem.

W znanych modelach obiektowych w małym stopniu odzwierciedlona jest dynamika danych. Najczęściej dynamikę można wyrazić za pomocą ewolucji klas i ewolucji obiektów. W podstawowym modelu obiektowym klasa jest kolekcją obiektów o tej samej strukturze. Wówczas ewolucja na poziomie klas polega na modyfikacji klasy, np. dodaniu/usunięciu klasy, zmianie dziedziny atrybutów klasy, czy też aktualizacji metod klasy. Ewolucja na poziomie obiektów najczęściej polega na usunięciu obiektu z klasy lub dodaniu obiektu do klasy. Zauważmy, że pomiędzy klasami i obiektami występują różnego rodzaju powiązania, takie jak związek dziedziczenia pomiędzy klasami, związek referencyjny pomiędzy obiektami i związek przynależności obiektu do klasy. Obiektowa baza danych tworzy więc skomplikowaną i zagnieżdżoną sieć, w której węzłami są poszczególne klasy i obiekty, a krawędzie odzwierciedlają różne związki między nimi. W takiej sieci każda zmiana polegająca na usunięciu lub przeniesieniu węzła na ogół powoduje jej bardzo kosztowną rekonstrukcję.

Powstaje zatem pytanie, czy nie odchodząc od ideologii obiektowości, można inaczej zdefiniować pojęcia klasy i obiektu oraz związki pomiędzy nimi tak, aby dynamika danych mogła być sprawniej realizowana?

W niniejszej pracy przedstawiono próbę pozytywnej odpowiedzi na to pytanie. W tym celu ujednolicono dwa podstawowe pojęcia modelu obiektowego, jakimi są klasa i obiekt. Aby to uzyskać przyjęto następujące założenia:

1. Obiekt może należeć jednocześnie do więcej niż jednej klasy (wieloklasowość obiektu).
2. Obiekt i klasa mają jednolitą strukturę, co powoduje, że klasa może być traktowana jako obiekt i odwrotnie (homogeniczność obiektu i klasy).

W dotychczasowych modelach wieloklasowość obiektu dotyczy tylko przynależności obiektu do klas w hierarchii podklasa-nadklasa. Na przykład, obiekt klasy Student jest także obiektem klasy Osoba. Natomiast niedopuszczalne jest, aby obiekt klasy Student należał jednocześnie do klasy Pracownik, jeśli między tymi klasami nie istnieje związek podklasa-nadklasa. Wówczas, aby zachować informację o tym, że jakiś student pracuje (lub jakiś pracownik studiuje) definiuje się w systemie dwa różne obiekty (reprezentujące ten sam obiekt rzeczywisty), z których jeden należy do klasy Student, a drugi do klasy Pracownik. Zwróćmy uwagę, że jeżeli wśród atrybutów opisujących te obiekty nie ma atrybutu kluczowego, wówczas wyszukiwanie odpowiedzi na pytanie o studiujących pracowników (lub pracujących studentach) może być niejednoznaczne. Istnieje wprawdzie rozwiązanie pośrednie nazywane dziedziczeniem wielokrotnym, które jednak nie jest szeroko stosowane. Opis problemów związanych z dziedziczeniem wielokrotnym przedstawiono w rozdziale pierwszym.

Przyczyny, dla których dobrze byłoby potraktować klasę jako obiekt wyjaśnili Won Kim [Kim96] i Oscar Nierstrasz [Nie89]. Przypomnijmy dwie z nich. Po pierwsze, cecha enkapsulacji modelu obiektowego powoduje, że wszystkie operacje muszą być wykonane na obiektach, także operacja tworzenia (usuwania) klasy. Po drugie klasa jest zbiorem obiektów (instancji) do niej należących i informacje, które dotyczą tego zbioru jako całości (np. średni wiek pracowników) są często potrzebne i trzeba je gdzieś w systemie przechować. Na przykład informację o średnim wieku pracowników możemy przechować w każdym obiekcie klasy Pracownik, lecz spowoduje to duży nadmiar informacji, i co gorsze, za każdym razem kiedy obiekty są dodawane lub usuwane, trzeba we wszystkich pozostałych obiektach aktualizować tę informację.

W podstawowym modelu obiektowym obiekt jest definiowany jako para (identyfikator, wartość), natomiast klasa jest trójką (nazwa, zbiór atrybutów, zbiór metod). Pierwszy element obiektu jest niepowtarzalny i niezmienny. Dynamika obiektu jest reprezentowana przez drugi element, wartość obiektu. Jednakże takie obiekty nie mogą w pełni zmieniać swojej struktury i zachowania, nie mogą migrować między klasami, nie mogą występować w kilku klasach, także w klasach nie powiązanych związkiem dziedziczenia (brakuje możliwości dodania/usunięcia obiektu do/z dowolnej klasy). Migracja obiektu jest ograniczona jedynie do usunięcia z jednej klasy i utworzenia go w innej klasie. Namiastką wieloklasowości jest przynależność obiektu do wszystkich nadklas zgodnie z hierarchią dziedziczenia.

Prowadzone badania musiały dać odpowiedź na następujące pytania:

1. W jaki sposób zapewnić możliwość ewolucji struktury i zachowania obiektu?

2. W jaki sposób rozwiązać problem niejednorodności obiektu i klasy?

Przez dynamikę danych rozumiemy wszelkie zmiany zachodzące w strukturze i zawartości informacyjnej bazy danych. W modelu obiektowym do tych zmian należą: dodanie (usunięcie) atrybutu, metody, obiektu, klasy; modyfikacja atrybutu, metody, obiektu, klasy, hierarchii klas; modyfikacja nazw atrybutu, metody, klasy. Zmiany te można podzielić na dwie grupy: zmiany dotyczące schematu bazy danych oraz zmiany dotyczące obiektów. Dynamiką danych w aspekcie zmian schematu bazy danych zajmowało się wielu autorów [Bjo89, Bra92, Dij93, Kim96, Kos93, Och92, Odb95], natomiast brak jest formalnego opisu takiego obiektowego modelu danych, który uwzględnia zmiany dotyczące obiektów wieloklasowych.

Celem pracy jest opracowanie modelu obiektów wieloklasowych odzwierciedlającego dynamikę obiektów.

Aby zrealizować cel ujednolicono definicję obiektu i klasy poprzez umieszczenie w obiekcie (klasie) informacji, do której klasy on należy jako obiekt, oraz które obiekty należą do niego jako klasy.

Szczegółowe cele realizowane w pracy, to:

1. Zdefiniowanie obiektu wieloklasowego i zbadanie jego własności.
2. Zdefiniowanie algebry obiektów wieloklasowych.
3. Opracowanie operatorów zmian w obiektach, w szczególności operatorów migracji.
4. Określenie ograniczeń dla obiektów wieloklasowych.
5. Zaproponowanie strukturalnego języka manipulacji danymi umożliwiającego wieloklasowość oraz migrację obiektów.

Model obiektów wieloklasowych został zaprojektowany jako rozszerzenie podstawowego obiektowego modelu danych, w którym podkreślono „dynamiczny” charakter obiektów, a nie tylko ich obiektową strukturę. Tradycyjna, statyczna więź pomiędzy klasami a obiektami została wymieniona na związek dynamiczny obiekt-podobiekt. Obiekty bazy danych mogą być specjalizowane inaczej niż pozwala na to klasyczna struktura klas zdefiniowanych w bazie danych. Następujący przykład ilustruje taką sytuację: Kowalski jest studentem. W czasie studiów podejmuje pracę, a więc zostaje pracownikiem, jednakże nie przestaje być studentem. Automatycznie stał się także podatnikiem. Ponadto po pewnym czasie awansuje w pracy do funkcji kierownika. Kończy studia, przestaje być studentem. Żeni się i po pewnym czasie zostaje ojcem itd.

Zaletą języka manipulacji danymi odzwierciedlającego dynamikę danych w porównaniu z innymi językami jest możliwość posługiwania się wieloklasowością i operatorami migracji obiektów. Dynamicznie modyfikowane obiekty-instancje oraz obiekty-klasy są tak samo traktowane przez język zapytań jak obiekty tradycyjne. W algebrze obiektów zostały dodane specjalne operatory służące tworzeniu nowych klas dla obiektu.

Wymienione cele zrealizowano w kolejnych rozdziałach pracy.

W rozdziale pierwszym szczegółowo przedyskutowano najważniejsze aspekty dynamiki danych w obiektowych bazach danych. Na początku rozdziału zdefiniowano postulaty odzwierciedlające dynamikę danych. Dokonano przeglądu różnych typów zmian

zachodzących w strukturze i zachowaniu obiektów. Na tle aktualnego stanu badań w obszarze obiektowych baz danych uzasadniono potrzebę rozszerzenia obiektowego modelu danych o wieloklasowość obiektów.

W drugim rozdziale opisano podstawowe pojęcia modelu obiektów wieloklasowych oraz zdefiniowano operatory algebry obiektów.

Rozdział trzeci zawiera opis możliwych zmian dokonywanych w obiektach wieloklasowych, zarówno elementarnych jak i złożonych, związanych z migracją danych. Przedstawiono definicje tych zmian za pomocą operatorów algebry obiektów.

W rozdziale czwartym omówiono konflikty, jakie mogą wystąpić w obiektach. Wyróżniono atrybuty obiektu-ograniczenia oraz pokazano ich własności.

Rozdział piąty zawiera propozycję strukturalnego języka manipulacji danymi związanego bezpośrednio z algebrą modelu. Pokazano, w jaki sposób dokonuje się translacja wyrażań języka do wyrażań algebry obiektów.

Zagadnieniu opracowania struktur danych dla obiektu wieloklasowego poświęcono rozdział szósty. Zaproponowano fizyczną strukturę obiektu oraz strukturę pliku danych. Omówiono również możliwości zwiększenia efektywności wykonywania niektórych operacji.

Pracę kończy podsumowanie prezentujące najważniejsze wyniki pracy, płynące z nich wnioski oraz propozycje kierunków dalszych badań.

Bibliografia zawiera publikacje podstawowe, cytowane w pracy i piśmiennictwo pomocnicze, oznaczone symbolem *.

Załącznik prezentuje syntaktykę podstawowych instrukcji języka manipulacji obiektami wieloklasowymi w notacji BNF.

1. Dynamika w obiektowych bazach danych

Najważniejszym aspektem obiektowej bazy danych analizowanym w pracy jest dynamika obiektów. Dynamika obiektów jest definiowana przez atrybuty, metody i powiązania z innymi obiektami.

Dynamikę obiektów w obiektowej bazy danych odzwierciedlają następujące postulaty:

1. Obiekt może zmieniać swoją strukturę i zachowanie przez cały okres swojego istnienia.
2. Obiekt może zmienić miejsce swojego pobytu.
3. Obiekt może zmieniać powiązania z innymi obiektami.
4. Obiekt może dostosować się do potrzeb informacyjnych użytkownika.

Odnosząc się do klasycznych pojęć z teorii baz danych postulat pierwszy dotyczy przede wszystkim zmiany schematu bazy danych, wersjonowania, ewolucji obiektów oraz zmiany wartości atrybutów i zmiany implementacji metod. Postulat drugi jest odpowiednikiem migracji obiektów. Trzeci postulat wyraża się m. in. w hierarchii dziedziczenia klas. Czwarty postulat jest związany z perspektywami i rolami obiektów.

Spełnienie postulatów 1-4 umożliwia obiektowi zmianę swojego stanu. W tym przypadku stan obiektu byłby określany przez zbiór atrybutów, metod, przynależność obiektu do klas, przez powiązania z innymi obiektami oraz przez zbiór atrybutów i metod udostępnionych dla konkretnego użytkownika. Wymienione postulaty wiążą się bezpośrednio również z pojęciem czasu, ponieważ każdą z wymienionych zmian określa czas jej przeprowadzenia.

Dynamikę obiektów w dotychczas znanych modelach wyrażają przede wszystkim takie pojęcia jak: zmiana schematu bazy danych, wersjonowanie, ewolucja obiektów, migracja obiektów, dziedziczenie, perspektywy, mechanizm ról, stan obiektu i czas.

1.1. Zmiana schematu bazy danych

Schemat bazy danych kojarzy się z pewnym wzorcem działania systemu bazy danych. Zawiera on informacje o strukturze obiektów, powiązaniach, dostępnych interfejsach, które mogą się zmieniać, a więc powinien w pełni realizować postulaty pierwszy, trzeci i czwarty. Ze względu na mechanizm dziedziczenia wymagane jest wprowadzenie zmienionych w stosunku do modelu relacyjnego operacji modyfikacji schematu [Och92, Odb95]. Schemat bazy danych jest dynamiczny, to znaczy umożliwia modyfikacje już istniejących obiektów. Najczęściej taka modyfikacja odbywa się w sposób następujący: nowym atrybutom nadaje się wartości puste a starym atrybutom nie zmienia się wartości.

Innym rozwiązaniem dla wprowadzenia zmian w schemacie bazy danych jest mechanizm **wersyjności schematu** [Kos93]. Możemy rozróżnić dwa typy zmian schematu: zmiana lokalna (dodanie i usuwanie atrybutów/metod) oraz zmiana globalna (tworzenie i

usuwanie klasy, zmiana związku dziedziczenia) [Klo96]. Obie te zmiany muszą zachować spójność atrybutów i metod zgodnie z tak zwanymi **niezmiennikami schematu**, np. zasadą zachowania poprawności hierarchii dziedziczenia, zasadą pojedynczego pochodzenia czy zasadą modyfikacji definicji **atrybutów kompozytowych** [Kim96]. Niezmienniki schematu dotyczą struktury schematu bazy danych. Nie ma natomiast uniwersalnego rozwiązania dla metod. Jeśli jakiś atrybut zostanie usunięty, metody odwołujące się do niego także powinny zostać zmodyfikowane [Fig96].

1.2. Wersjonowanie

Poprawne i efektywne działanie systemu bazy danych wiąże się z właściwą interpretacją zawartych w nim danych oraz z możliwością dokonywania zarówno strukturalnych jak i funkcjonalnych zmian w systemie. Jednym ze stosowanych rozwiązań jest mechanizm wersjonowania danych. Mechanizm ten powinien umożliwić użytkownikom bazy danych tworzenie i zarządzanie wersjami danych. Modele danych umożliwiające wersjonowanie danych nazywane są **wielowersyjnymi modelami danych**. Możliwość wykorzystania mechanizmu wersjonowania danych w modelu obiektowym uzyskuje się przez wprowadzenie do modelu pojęcia **wersji obiektu** [Bjo89]. Istnienie wielu wersji danych rodzi jednak problem ich spójności. Wprowadza się więc np. pojęcie **porcji** jako grupy wspólnie ewoluujących obiektów oraz hierarchię kompozycji porcji [Kos93].

Zbiór wersji jest interpretowany w zależności od tego czy wersje posiadają atrybut czasu utworzenia czy też posiadają atrybut semantycznej przynależności. Pojedyncza wersja obiektu reprezentuje konkretny **stan obiektu** [Dij93]. Wersje obiektów są porządkowane zgodnie z funkcją czasu lub z funkcją semantycznej przynależności. Pojęcie wersyjności jest wykorzystywane w dwóch przypadkach: po pierwsze do reprezentowania **historii obiektu** – wersjonowanie obiektu, po drugie przy rozwiązywaniu problemu zmian w definicji typu lub klasy – wersjonowanie schematu bazy danych [Fig96].

1.3. Ewolucja obiektów

Ewolucja obiektów, często nazywana **zmianą typu** obiektu, jest pojęciem należącym do modelowania konceptualnego, spotykanym w systemach, które posiadają zdolność reprezentowania ewolucji świata rzeczywistego [Alb95a, Ber98b, Ngu98, Ska86]. Zmiany w systemie są stymulowane przez zmiany w środowisku lub zmiany potrzeb informacyjnych użytkowników [Dan92].

Pojęcie ewolucji obejmuje wersje obiektów oraz wersje schematu bazy danych [Pro94]. Pozwala to na swobodę wyboru typu wprowadzanych zmian w zależności od tego czy są to zmiany lokalne dotyczące poszczególnych obiektów, czy zmiany globalne dotyczące klas lub ich powiązań. Problem ewolucji obiektów został zauważony w każdym ze znanych modeli baz danych.

Ewolucja obiektów obejmuje też zmiany w **obiektach kooperacyjnych**. Koncepcje zachowania kooperacyjnego grupy obiektów pojawiły się już w roboczych dokumentach standardu ISO/IECJTC1 SC21 Working Group 7 [Gri82]. Istota tego pojęcia jest widoczna w szczególnie sposób podczas projektowania i funkcjonowania systemów obiektowych, w których najważniejszą rolę pełni komunikacja między obiektami, np. system zarządzania ruchem na lotnisku. Wprowadzając abstrakcyjne pojęcie klas kooperacyjnych unika się stosowania metod pozaklasowych. Klasy kooperacyjne łączą ze sobą różne klasy podobnie

jak relacja łączy ze sobą różne tabele w modelu relacyjnym. Jako jedno z rozwiązań implementacyjnych dla klas kooperacyjnych może posłużyć system transakcji [Liu96].

Ewolucja obiektów jest koncepcją najbliższą realizacji postulatów pierwszego umożliwiającego modyfikację struktury i zachowania obiektu oraz postulatu drugiego pozwalającego na migrację obiektów.

1.4. Migracja obiektów

Miejsce pobytu obiektu możemy określić przez **przynależność do klasy** lub badając aktualny **stan obiektu**. Oba mechanizmy są wykorzystywane do **migracji obiektów**. Mając na uwadze ograniczenia schematu bazy danych rozróżniamy dwa typy migracji obiektów: pierwszy prostszy z klasy do podklasy lub nadklasy, drugi do klasy nie związanej hierarchią dziedziczenia [Pap97]. Ponadto rozróżnia się dwie strategie implementacyjne przemieszczania obiektów do klas, do jakich należy obiekt: opóźnioną i natychmiastową. Pierwsza z nich polega na przeprowadzaniu zmian w obiekcie dopiero na etapie ponownego jego użycia już w nowym środowisku, natomiast zgodnie z drugą strategią zmiany wprowadza się zaraz po zarejestrowaniu obiektu w nowej klasie. W przypadku, gdy obiekt dodajemy do klasy mamy do czynienia z pewnym rodzajem migracji obiektów, co możemy nazywać **replikacją obiektów** analogicznie do replikacji danych w bazach relacyjnych. Replikacja obiektu zakłada, że obiekt może występować w kilku klasach jednocześnie. Zmiana przynależności obiektu do klasy bez konieczności jego usunięcia i ponownego utworzenia, a więc z zachowaniem istniejącego unikalnego identyfikatora oraz wszystkich jego cech jest operacją złożoną. Większość obecnie istniejących systemów nie dostarcza tej cechy [Fig96].

Problem zachowania integralności danych w procesie migracji ze względu na referencyjne powiązania obiektów można rozwiązać w modelu danych, natomiast problemy związane z utratą części cech obiektu czy migracji powiązanych ze sobą obiektów są najczęściej rozwiązywane dopiero na etapie konkretnej implementacji systemu. W istniejących systemach czasami dopuszcza się przemieszczanie obiektów do podklas klasy, do której obiekty aktualnie należą lub do klasy o tych samych dziedzinach atrybutów.

1.5. Dziedziczenie

Dziedziczenie jest podstawowym pojęciem paradygmatu obiektowego. Na nim opiera się m. in. hierarchia klas. Zależność dziedziczenia znana przede wszystkim z języków programowania obiektowo-zorientowanego została przeniesiona na płaszczyznę obiektowych baz danych [Abi95]. Dziedziczenie obejmuje także zmiany w obiekcie po wprowadzeniu zmian w klasie nadrzędnej zgodnie z zasadą dziedziczenia atrybutów i metod [Kim96]. Jednakże, jeśli obiekt miałby obejmować wszystkie lokalne aspekty zmian swojej struktury i zachowania to należałoby ciągle rozbudowywać schemat bazy danych, co prowadziłoby do szybkiej utraty kontroli nad systemem. Często te lokalne aspekty mają charakter tymczasowy i nie powinno się ich utrzymywać. Obiektem można posługiwać się nie znając jego typu, tzn. każdy z jego podobiektów określa metody manipulowania i odpowiada za właściwe ich używanie (**enkapsulacja**). Wbudowany w model obiektowy **polimorfizm** metod zapewnia różne zachowanie się obiektów w zależności od kontekstu. Z pojęciem dziedziczenia związanych jest wiele problemów. Dwa najważniejsze z nich to dziedziczenie wielokrotne oraz zasięg dziedziczenia.

W większości istniejących systemów każdy obiekt należy tylko do jednej klasy lub do zbioru klas powiązanych relacją dziedziczenia. Czasami zezwala się na to, aby obiekt należał do więcej niż jednej klasy nie powiązanych ze sobą relacją dziedziczenia. Takim rozwiązaniem jest **dziedziczenie wielokrotne**, czyli wtedy, gdy klasa może dziedziczyć atrybuty i metody od więcej niż jednej nadklasy. Przykład: `Pracujący_Student`. Jest on jednocześnie `Studentem`, jak i `Pracownikiem`, posiada atrybuty wspólne dla obydwu klas. Może pojawić się jednak konflikt nazw atrybutów czy metod – między klasą a jej nadklasą i między nadklasami danej klasy. Wtedy najczęściej wprowadza się priorytet, np. nazwa użyta w klasie o większym priorytecie ma pierwszeństwo w stosunku do nazw z innej nadklasy. W większości systemów uporządkowanie pierwszeństwa nadklas jest określone dla każdej klasy. Np. klasa `Pracujący_Student` może odziedziczyć atrybut `Wydział` na podstawie określonego priorytetu. Jednakże tracimy wtedy informację z klas o mniejszym priorytecie (`Wydział` odziedziczony od `Pracownika` zakrywa `Wydział` „dziedziczony” od `Studenta`). Przykład ten jednoznacznie pokazuje, iż mechanizm wielokrotnego dziedziczenia jest niewystarczający dla dokładnego określenia stanu obiektu. W przypadku braku możliwości dziedziczenia wielokrotnego, musi być skonstruowana alternatywna hierarchia klas zawierająca wszystkie klasy pośrednie: `Osoba` – `Student` – `Pracownik` – `Pracujący_Student` lub `Osoba` – `Pracownik` – `Student` – `Pracujący_Student`. Pojawia się wtedy inny problem. Podklasa dziedziczy atrybuty i metody od semantycznie nie związanej klasy. `Pracownik` dziedziczy atrybuty i metody od `Studenta` lub na odwrót `Student` dziedziczy atrybuty i metody od `Pracownika`. Problem ten można rozwiązać wprowadzając dwa rodzaje `Pracującego_Studenta`. Powoduje to jednak, że rozrastający się w ten sposób schemat bazy danych staje się dodatkowo mało czytelny.

Czasami dziedziczenie wielokrotne jest zastępowane lub uzupełniane tzw. **interfejsami** [Mul00]. Interfejsy stanowią na diagramach UML-owych specjalny rodzaj klasyfikatora. W języku Java słowo kluczowe *implements* pozwala na dołączenie interfejsu do klasy bez konieczności dziedziczenia go po klasie macierzystej. Mechanizm ten umożliwia łączenie ze sobą wielu zachowań, które mogą być współdzielone przez różne klasy. Nie posiada on jednak atrybutów, interfejsy składają się jedynie z zestawów operacji. Główną wadą interfejsów jest niemożność dziedziczenia za ich pomocą kodu – dziedziczyć można jedynie deklaracje operacji. Wszystkie operacje wchodzące w skład danego interfejsu muszą posiadać swoje implementacje w dziedziczącej go klasie.

Drugim znanym problemem jest **zasięg dziedziczenia** [Kim96]. Najczęściej stosowane jest **dziedziczenie pełne**, w którym podklasa dziedziczy wszystkie atrybuty i metody od swoich nadklas, za wyjątkiem tych, których dotyczą konflikty nazw. Dziedziczenie pełne w wielu przypadkach ogranicza możliwości modelowania świata rzeczywistego, często specjalizowane obiekty nie posiadają wszystkich cech swoich rodziców, ale za to dziedziczenie pełne jest łatwiejsze w implementacji. **Dziedziczenie wybiórcze**, w którym podklasa może nie dziedziczyć pewnych atrybutów i metod od swoich nadklas nie jest praktycznie spotykane [Kim96]. Przy dziedziczeniu atrybutów o tej samej nazwie i różnych dziedzinach natrafiamy na kolejny problem. Podobnie jak w przypadku wyboru atrybutu z nadklasy należy zdefiniować uporządkowanie dziedzin według ich uszczegółowienia – o ile jest to możliwe – ostatecznie wybierając najbardziej wyspecjalizowaną dziedzinę. Alternatywnym rozwiązaniem jest arbitralny wybór dziedziny.

1.6. Perspektywy

Przez dostosowywanie się obiektów do potrzeb informacyjnych użytkowników rozumie się m. in. możliwość występowania obiektów o odmiennym kształcie i zachowaniu, czyli np. możliwość stosowania wielu **perspektyw** do obiektów. W przeciwieństwie do zmian w schemacie danych mechanizm perspektywy nie powoduje straty informacji z bazy danych [Ber92, Bra92]. Inną różnicą jest to, że obiekt utworzony w jakiejś wersji schematu jest widoczny jedynie w tym schemacie, natomiast w bazie z perspektywami obiekt jest widoczny zawsze wtedy i tylko wtedy, jeśli spełnia warunki perspektywy. Dotychczas w pracach nad wykorzystaniem perspektywy stworzono podstawy modelowania i wykorzystania perspektyw w obiektowych bazach danych [Wie91] oraz skonstruowano odpowiedni język zapytań [Yoo93].

W najprostszym przypadku perspektywa jest **klasą wirtualną**, czyli abstrakcyjną klasą zbudowaną na podstawie innych klas. Perspektywa umożliwia m.in. autoryzację dostępu do obiektów, prezentację tych samych obiektów w różny sposób oraz możliwość wprowadzenia dodatkowego poziomu ograniczeń. Do definiowania perspektywy służą następujące operatory obiektowe: operator **selekcji**, operator **projekcji**, operator **złączenia**, operator **sumy**, operator **przecięcia** i operator **różnicy**. Perspektywy mogą być konstruowane na bazie jednej klasy lub wielu klas. Taki sposób definiowania perspektywy powoduje jednak kilka problemów związanych z dodawaniem, aktualizowaniem i usuwaniem obiektów w utworzonej przez perspektywę klasie wirtualnej. Opisany mechanizm wielu interfejsów do obiektów przypomina **mechanizm ról**. Perspektywa może być postrzegana jako element aktywnego interfejsu użytkownika.

1.7. Mechanizm ról

Szukając rozwiązania dla problemu wielokrotnego dziedziczenia powodowanego konfliktem nazw, typów i implementacji metod wprowadzono pojęcie **roli obiektu** [Alb93, Got96, Per90, Won97a]. Odrzucono więc m. in. rozwiązania polegające na ustaleniu priorytetów dziedziczenia, lokalnej zmianie nazwy lub zmianie schematu bazy danych. Do konfliktu nie dochodzi, gdyż obiekt w danym kontekście może występować tylko w jednej roli. Rola obiektu przejmuje odpowiedzialność za dynamiczną **specjalizację obiektu** [Pap91]. Rola posiada unikalny identyfikator, prywatne atrybuty i metody, dziedziczy atrybuty i metody obiektu [Jod98]. Typowymi systemami, w których role mogą przynieść wiele korzyści są: systemy projektowe, systemy zarządzania czy systemy z bazą wiedzy. Obiekty mogą nabywać lub tracić role niezależnie od stanu obiektu. Role jednego obiektu mogą cechować różne własności. Zachowana jest enkapsulacja, gdyż jest ograniczony dostęp do atrybutów i metod w ramach jednej roli. Wprowadzenie **zależności ról** jednego obiektu wprawdzie pozwala uniknąć ewentualnych konfliktów, ale za to nakłada dodatkowe trudności przy kontroli spójności danych. Pogląd o niezależności hierarchii klas i hierarchii ról wydaje się być dyskusyjny w aspekcie utrwalania się ról – otrzymywania niezbywalnych cech - i przekształcania w klasy. Niezależność dotyczy tylko poziomu definiowania dziedziczenia – dziedziczenie ról istnieje na poziomie konkretnych obiektów, natomiast dziedziczenie klas występuje na poziomie hierarchii klas. Zaproponowano dwie równorzędne metody implementacji ról jako **zagnieżdżone obiekty** [Jun96] i jako samodzielne **obiekty-aspekty** związane z obiektem podstawowym [Ric91]. Najczęściej role są odpowiedzialne za sterowanie dostępem do obiektów [Koł01]. Ponadto mogą one odzwierciedlać preferencje użytkowników końcowych [Dan92].

1.8. Stan obiektu i czas

Pojęcie **stan obiektu** zostało wprowadzone w pracy [Atk89] i jest jednym z podstawowych pojęć obiektowego modelu danych. Obiekt może się zmieniać, może ukryć atrybuty, może zmienić swoje miejsce. Może zostać usunięty lub utworzony przez inny obiekt. Własności te zwykle określamy jednym terminem – zmianie uległ stan obiektu. Analizę własności związanych ze zmianą stanu obiektu przedstawiono w pracach [Kro95d, Kro96]. Zmiana stanu może być realizowana jako przesunięcie obiektu w schemacie klas, czyli dodanie obiektu do nowej klasy i usunięcie go z poprzedniej klasy [Kro00c]. Wymienione własności będą analizowane w rozdziale trzecim niniejszej pracy.

Także **czas** jest jednym z podstawowych pojęć obiektowego modelu danych, pomimo iż został wymieniony na końcu. Czas w bazach danych służy do chronologicznego uszeregowania zdarzeń. Najczęściej wyróżniamy trzy rodzaje czasu [Lai94]:

1. **Czas właściwy** określający moment, w którym zdarzenie zaistniało w otoczeniu.
2. **Czas transakcyjny** określający moment rejestracji zdarzenia przez system bazy danych.
3. **Czas modalny** określający przypuszczalny czas właściwy zajścia zdarzenia w przyszłości.

Czas jest wykorzystywany do budowania **scenariuszy**, czyli ustalenia jednoznacznego przebiegu zdarzeń [Gor93]. Problem ten jest rozwiązywany przez definiowanie grafu skierowanego, którego wierzchołkami są zdarzenia, a krawędziami zależności czasowe pomiędzy nimi. Wyodrębniono trzynaście zależności czasowych: **before, after, meets, met-by, overlaps, overlapped-by, during, includes, starts, started-by, finishes, finished-by, equals** [Ngu96]. Każdą z wzajemnych relacji dwóch przedziałów czasowych można zdefiniować jako alternatywę podanych trzynastu zależności. Na tej podstawie zbudowano temporalną algebrę [Su98] i temporalny język zapytań [Ros93]. Bazy danych, w których stosuje się w ten sposób zdefiniowany czas zdarzeń określa się jako temporalne bazy danych.

W pracy pojęcie czasu nie będzie analizowane, ale nie prowadzi to do jakichkolwiek ograniczeń. Zmiany w bazie danych mogą być zapamiętywane w taki sam sposób jak to jest realizowane w tradycyjnej bazie danych np. przez wprowadzenie atrybutu zawierającego czas dokonywanych zmian lub też czas ten może być zapamiętywany automatycznie przez system transakcyjny bazy danych.

1.9. Wnioski

Istnieje kilka zasadniczych różnic między proponowaną w pracy migracją obiektu wieloklasowego a typową zmianą schematu bazy danych. Pierwsza polega na tym, że zmiana schematu odbywa się na poziomie klas i dopiero jej skutki dotyczą obiektów. Taki mechanizm znajdujemy w systemach ObjectStore czy UniSQL. Migracja natomiast jest ściśle związana z obiektem. Zwykle zmiana dotyczy pojedynczego obiektu. Obiekt nabywa lub traci atrybuty i metody. Jeśli zmiana dotyczy wszystkich obiektów lub prawie wszystkich to zmiana schematu jest uzasadniona. Częściej jednak obiekt nabywa lub traci cechy w sposób indywidualny. Indywidualizowanie (specjalizowanie) schematu bazy danych nie jest korzystne dla systemu i powinno się tego unikać. Pojawiające się zmiany w świecie rzeczywistym mają charakter ewolucyjny i dopiero po dłuższej obserwacji jesteśmy w stanie określić własności wspólne dla grupy obiektów. Dlatego też zastosowanie migracji dla

pojedynczego obiektu jest rozwiązaniem lepszym. Kolejna różnica dotyczy nakładanych ograniczeń na oba mechanizmy. Przy zmianie schematu muszą być zachowane reguły utrzymywania poprawności schematu m. in. zasada pełnego dziedziczenia oraz zasada zachowania poprawności dziedziny [Fig96]. Jak pokazano wcześniej zarówno przy dziedziczeniu pojedynczym jak i wielokrotnym zachowanie tych reguł prowadzi do szybkiego rozbudowywania schematu bazy danych i w rezultacie do nadmiernego skomplikowania systemu bazy danych. Stosując mechanizm migracji obiektu nie jesteśmy ograniczeni tymi regułami. Po trzecie tradycyjne podejście do zmian schematu bazy danych nie pozwala na przyjmowanie pojedynczego obiektu do wielu klas jednocześnie. Jedynie hierarchia dziedziczenia pozwala przez specjalizację obiektów na ich istnienie „jakby” w wielu klasach. W rzeczywistości taka sytuacja jest najczęstsza. Wieloklasowość obiektu zapewnia możliwość wielokrotnego definiowania odmiennych cech i zachowań w pojedynczym obiekcie.

W niniejszej pracy przedstawiamy elementy modelu danych, który spełnia wymienione wymagania dynamiki danych obejmujące migrację obiektów, wielokrotne dziedziczenie oraz zmianę schematu bazy danych. Wyłączenie z modelu wersjonowania, mechanizmu perspektyw i ról, obiektów kooperacyjnych podyktowane zostało opublikowaniem rozwiązań tych problemów w wielu pracach lub przynajmniej ich częściową realizacją w projektach informatycznych [Bjo89, Bra92, Dij93, Kim96, Kos93, Och92, Odb95]. Z drugiej strony formalne uwzględnienie np. wersjonowania nie wpłynęłoby na prezentowany w pracy sposób rozwiązania problemu wieloklasowości [Kos93]. Podobnie nie wprowadzamy explicite pojęcia czasu, ponieważ problemem tym zajmuje się wielu badaczy [Gor93, Lai94, Ros93] i wydaje się, że najważniejsze problemy związane z przetwarzaniem czasu zostały rozwiązane.

Wszystkie postulaty z początku rozdziału z wyjątkiem drugiego postulatu są realizowane w mniejszym lub większym stopniu za pomocą ról. Proponowane w pracy pojęcie obiektu wieloklasowego intuicyjnie najbliższe jest pojęciu roli, jaką może pełnić obiekt.

Model obiektów wieloklasowych jest próbą rozwiązania najważniejszych problemów omówionych w niniejszym rozdziale i stwarza podstawy do integracji także pozostałych pojęć związanych z dynamiką danych.

2. Algebra obiektów wieloklasowych

Model danych oraz algebra są podstawowymi elementami każdego systemu zarządzania bazą danych. Możliwości operatorów algebry determinują transformacje przeprowadzane w bazie danych.

Jako punkt wyjścia do opracowania modelu obiektów wieloklasowych wybrano obiektowy model ENCORE [Mit94], który powstał na Uniwersytecie Brown w 1986 roku, oraz algebrę EQUAL (ang. *Encore QUery ALgebra*) [Urb94a]. W pracy wykorzystano także doświadczenia w tworzeniu obiektowej bazy danych OB! OK!¹ [Kro95c, Pia96]. Nie mniejszą rolę odegrały badania empiryczne związane z implementacjami systemów obiektowych [Bie97, Bie01, Kro00a].

2.1. Model danych

W modelu przyjęto następujące założenia.

1. Obiekt.

Dane są obiektami, dla których można określić pewien skończony zbiór cech bezpośrednio pochodzących ze świata zewnętrznego [Atk89]. Jest to główne założenie podejścia obiektowego.

2. Tożsamość obiektu.

Obiekt posiada niepowtarzalny identyfikator [Atk89]. Identyfikator jest automatycznie nadawany w momencie tworzenia obiektu i nie jest potem modyfikowany.

3. Dynamika obiektu.

Obiekt posiada własności dynamiczne [Kro94]. Struktura obiektu może być zmieniana. Obiekt może należeć do wielu klas [Kro00c]. Istnieje możliwość specyfikacji ograniczeń dotyczących wartości atrybutów dla klasy i nakładanych na związki, w których obiekt uczestniczy [Ull00].

4. Metoda.

Obiekt posiada metody, pozwalające w pełni zarządzać wszystkimi cechami obiektu [Bal90]. Metody mogą komunikować się ze środowiskiem, w którym obiekt istnieje.

5. Przesłanie atrybutów i przeciążanie metod.

Istnieje możliwość definiowania specjalizowanych atrybutów i metod [Wie93, Pia96]. Ma to szczególne zastosowanie w systemach technicznych.

¹ Skrót od nazwy "Object Base! OK!"

6. Klasa.

Zbiory jednakowych obiektów, tzn. takich, w których zbiory atrybutów i metod są takie same, tworzą klasy [Kim96, Spo91].

7. Homogeniczność obiektów i klas.

Obiekt i klasa są definiowane a więc i przetwarzane w taki sam sposób [Kro00c].

8. Hierarchia i dziedziczenie.

Klasa może dziedziczyć atrybuty lub metody od innej klasy, tzn. klasa może przejąć, uzupełniając swój zbiór cech, wybrane własności i metody. Klasy tworzą hierarchię [Abi95, Kim96].

Autorzy prezentowanych w literaturze modeli obiektów zajmują się głównie zmianami dotyczącymi klas, a nie obiektów [Abi95] i dopiero przez zmiany w klasach wpływają na zmiany w obiektach [Alh93a, Gou91].

Alhajj i Arkun [Alh93a] opisują algebrę obiektowej bazy danych jako nadzbiór algebry relacyjnej nie uwzględniając dynamiki obiektów. W pracy zdefiniowano obiekty i metody. Obiekt posiada identyfikator i wartość. Zdefiniowano tożsamość obiektów, dziedziczenie, operatory selekcji, projekcji, iloczynu kartezjańskiego, sumy i różnicy. Tak zdefiniowana algebra obiektowa obejmuje algebrę relacyjną, jednakże sam obiekt jest statyczny. Raz utworzony nie zmienia się.

W modelu regułowym [Gou91] obiekt posiada atrybuty i reguły, które nie są obiektami. Nie spełniony jest więc postulat homogeniczności. Natomiast interesujące jest wprowadzenie ograniczeń integralnościowych dla obiektów w postaci reguł złożonych z warunku i akcji.

W modelu opisanym w [Abi95] zdefiniowano typ, klasę, obiekt, hierarchię klas oraz metodę. Zaproponowano język zapytań. Ponieważ centralnym elementem jest klasa nie wprowadzono bezpośredniej możliwości modyfikacji obiektu.

Dopiero model ENCORE spełnia praktycznie wszystkie wyżej wymienione założenia za wyjątkiem założeń o dynamice obiektu i homogeniczności. W niżej prezentowanym rozszerzeniu tego modelu centralnym elementem jest obiekt. Ponadto szczególną uwagę zwrócono na zmiany w obiektach oraz homogeniczność obiektów i klas.

2.1.1. Wieloklasowość obiektów

Obiekt to każda konkretna lub abstrakcyjna rzecz zawierająca powiązania z innymi rzeczami. Zgodnie z normą ISO [Gri82] rozróżniamy trzy poziomy obiektowości:

1. Wystąpienie - właściwy obiekt, który może reprezentować rzecz lub problem.
2. Typ - zbiór własności statycznych i dynamicznych wspólnych dla różnych wystąpień.
3. Klasa - zbiór wystąpień jednego typu.

Najczęściej przyjmowaną w literaturze definicją obiektu zgodną z powyższą normą ISO jest para (identyfikator, wartość). Konsekwencją tego faktu jest to, że klasa jest reprezentowana przez trójkę (nazwa, zbiór atrybutów, zbiór metod). Identyfikator obiektu zwykle jest automatycznie generowany przez system i jest niezależny od wartości obiektu. Istnieje wiele metod generowania niepowtarzalnego identyfikatora [Del95]. Nie będziemy

tego problemu analizować, ponieważ wystarczy nam to, że identyfikator rozróżnia obiekty. Dlatego dla uproszczenia przyjmujemy zbiór liczb naturalnych jako zbiór identyfikatorów.

W rozszerzeniu modelu podstawowego, które proponujemy w niniejszej pracy, można wyróżnić dwie podstawowe cechy:

1. **Wieloklasowość** oznaczającą, że obiekt może należeć jednocześnie do więcej niż jednej klasy.
2. Klasa i obiekt mają jednolitą strukturę; klasa może więc być traktowana jako obiekt i odwrotnie.

Pierwszą cechę posiadają tylko nieliczne ze znanych modeli [Sub98] i dotyczy ona tylko przynależności obiektu do klas w hierarchii podklasa-nadklasa. Np. obiekt klasy Student jest także obiektem klasy Osoba. Niedopuszczalne jest, aby obiekt klasy Student należał jednocześnie do klasy Pracownik, jeśli między tymi klasami nie istnieje związek podklasa-nadklasa. Aby zachować informacje o tym, że jakiś student pracuje (lub jakiś pracownik studiuje) definiuje się w systemie dwa różne obiekty (reprezentujące ten sam obiekt rzeczywisty), z których pierwszy należy do klasy Student, a drugi do klasy Pracownik. Komplikuje to wyszukiwanie, a w przypadku braku atrybutu kluczowego wśród atrybutów opisujących te obiekty (np. PESEL), nie jest możliwe wyszukiwanie wśród studiujących pracowników lub pracujących studentów.

Druga cecha też wynika z jej braku w modelu podstawowym. Między innymi w pracy [Kim96] przedstawiono przyczyny, dla których należy traktować klasę jako obiekt. Przypomnijmy dwie z nich. Po pierwsze, wszystkie operacje muszą być wykonane na obiektach, także operacja tworzenia (usuwania) klasy. Po drugie klasa jest zbiorem obiektów (instancji) do niej należących i informacje, które dotyczą tego zbioru jako całości (np. średni wiek pracowników) są często potrzebne i należy je w systemie przechować. Co prawda, możemy informację o średnim wieku pracowników przechować w każdym obiekcie klasy Pracownik, lecz spowoduje to duży nadmiar informacji, i co gorsze, za każdy razem kiedy obiekty są dodawane lub usuwane, trzeba we wszystkich pozostałych obiektach aktualizować tę informację.

Proponujemy ujednolicić pojęcie obiektu i klasy przez umieszczenie w obiekcie (klasie) informacji, do której klasy on należy jako obiekt, oraz które obiekty należą do niego jako klasy. Zauważmy, że związek przynależności obiekt-klasa jest podobny do związku element-zbiór, a ten drugi możemy zawsze przekształcić w związek zbiór-zbiór. Np. związek $2 \in \{2,3\}$ implikuje związek $\{2\} \in \{\{2\}, \{3\}\}$. Sugeruje to, że związek obiekt-klasa może być przekształcony w związek klasa-klasa lub obiekt-obiekt, jeśli zamiast słowa „klasa” użyjemy słowa „obiekt”. Konsekwencje tego faktu zauważymy w definicji obiektu prezentowanej w następnym podrozdziale.

2.1.2. Obiekt wieloklasowy

Aby zdefiniować obiekt wieloklasowy wprowadzamy następujące oznaczenia.

Żałóżmy, że OID jest przeliczalnym zbiorem identyfikatorów, skończonym zbiorem atrybutów jest A oraz M jest skończonym zbiorem metod. Zbiór wartości obiektu wieloklasowego x oznaczamy przez $V(x)$, zbiór atrybutów obiektu wieloklasowego x przez

$A(x)$, a zbiór wartości atrybutu $a \in A(x)$ przez $V_a(x)$. Zbiór zawierający wszystkie obiekty wieloklasowe (występujące w bazie, systemie) oznaczmy przez OBJ.

Przyjmijmy dalej, że D będzie zbiorem wartości, składającym się z liczb całkowitych, liczb dziesiętnych, łańcuchów i wartości logicznych.

Wówczas zbiór wartości występujących w bazie oznaczany przez V może być określony w następujący sposób:

1. $\emptyset \in V$ (wartość pusta).
2. Jeżeli $v \in D$, to $v \in V$ (wartości atomowe).
3. Jeżeli $v \in \text{OID}$, to $v \in V$ (wartości referencyjne).
4. Jeżeli $a_i \in A$ i $v_i \in V$ dla $i=1, 2, \dots, n$ to $\langle a_1:v_1, a_2:v_2, \dots, a_n:v_n \rangle \in V$ (wartości krotkowe).
5. Jeżeli $v_1, v_2, \dots, v_n \in V$, $v_i \neq v_j$ dla $i \neq j$, to $\{v_1, v_2, \dots, v_n\} \in V$ (wartości zbiorowe).

Wartość krotkową $\langle a_1:v_1, a_2:v_2, \dots, a_n:v_n \rangle$ nazywać będziemy wyrażeniem, a składową wartości krotkowej $a_i:v_i$ podwyrażeniem. Kolejność składowych w wartości krotkowej nie ma znaczenia. Będziemy dopuszczać wartości krotkowe w postaci $\langle \emptyset \rangle$ i traktować je jako puste wartości krotkowe. Dwie wartości krotkowe będziemy uważać za identyczne, jeśli mają takie same składowe. Atrybut, który ma wartość referencyjną nazywamy atrybutem referencyjnym. Natomiast przez $\text{Ref}(x)$ oznaczamy zbiór wieloklasowych obiektów referencyjnych dla x , tzn. takich, których identyfikatory są wartościami atrybutów referencyjnych obiektu wieloklasowego x .

Definicja

Obiektem wieloklasowym x (w skrócie obiektem x) nazywamy wyrażenie $(\text{id}(x), \text{Sub}(x), \text{Sup}(x), A_{\text{Sub}}(x), v(x), M(x))$, gdzie

$\text{id}(x)$ - identyfikator lub nazwa obiektu x ,

$\text{Sub}(x)$ - zbiór identyfikatorów podobiektów obiektu x ,

$\text{Sup}(x)$ - zbiór identyfikatorów nadobiektów obiektu x ,

$A_{\text{Sub}}(x)$ - zbiór atrybutów opisujących podobiekty obiektu x ,

$v(x)$ - funkcja odwzorowująca zbiór atrybutów obiektu x w zbiór wartości, tj. wartość obiektu x ,

$M(x)$ - zbiór metod obiektu x ,

przy czym:

- (1) $(\forall x, y \in \text{OBJ}) \left(\text{id}(x) = \text{id}(y) \Leftrightarrow v(x) = v(y) \wedge \text{Sub}(x) = \text{Sub}(y) \wedge \text{Sup}(x) = \text{Sup}(y) \right)$,
- (2) $(\forall x, y \in \text{OBJ}) (\text{id}(y) \in \text{Sub}(x) \Rightarrow \text{id}(x) \notin \text{Sub}(y))$,
- (3) $(\forall x, y \in \text{OBJ}) (\text{id}(y) \in \text{Sup}(x) \Rightarrow \text{id}(x) \notin \text{Sup}(y))$,
- (4) $(\forall x, y \in \text{OBJ}) (\text{id}(y) \in \text{Sub}(x) \Leftrightarrow \text{id}(x) \in \text{Sup}(y))$,

$$(5) \quad (\forall x, y \in \text{OBJ})(\text{id}(y) \in \text{Sub}(x) \Rightarrow A(y) \supseteq A_{\text{Sub}}(x) \vee A_{\text{Sub}}(y) \supseteq A_{\text{Sub}}(x)),$$

$$(6) \quad v(x): A(x) \rightarrow V(x) \text{ gdzie } a \in A(x), v \in V_a(x).$$

Wieloklasowość obiektów narzuca dodatkowe wymagania. Do sformułowania $A(x)$ i $A_{\text{Sub}}(x)$ konieczne jest wprowadzenie dodatkowych pojęć: atrybutu lokalnego, pochodzenia atrybutu i sumy atrybutów.

Atrybutem lokalnym obiektu x nazywamy atrybut $a \in A(x)$ taki, że:

$$\neg \exists y \in \text{OBJ}, \text{id}(y) \in \text{Sup}(x) \text{ gdzie } a \in A(y).$$

Zbiór atrybutów lokalnych obiektu x oznaczamy przez $\bar{A}(x)$. Na podstawie definicji atrybutu lokalnego wnioskujemy, że $\bar{A}(x)$ oznacza zbiór $A(x)$ ograniczony do atrybutów nie występujących w nadobiektych obiektu x . W taki sam sposób określamy atrybuty lokalne dla podobiektów i oznaczamy je przez $\bar{A}_{\text{Sub}}(x)$.

Teraz możemy zdefiniować pojęcie pochodzenia atrybutu. Intuicyjnie, pochodzenie atrybutu oznacza obiekt, w którym dany atrybut został zdefiniowany po raz pierwszy, czyli nie jest on dziedziczony z żadnego nadobiektu. W przypadku obiektów wieloklasowych mogą jednakże istnieć dwa lub więcej nadobiektów, w których ten atrybut został zdefiniowany. W takiej sytuacji określamy jeden z nich jako właściwy nadobiekt dla danego atrybutu.

Pochodzeniem atrybutu $a \in A$ obiektu x nazywamy obiekt y , jeśli $a \in \bar{A}(y)$.

Odpowiednio pochodzeniem atrybutu $a \in A_{\text{Sub}}$ obiektu x nazywamy obiekt y , jeśli $a \in \bar{A}_{\text{Sub}}(y)$. Pochodzenie atrybutu a należącego do obiektu x będziemy oznaczać symbolem (a, x) . Konsekwencją takiej definicji pochodzenia atrybutu jest to, że pochodzenie zawsze istnieje i jest jedyne. Teraz możemy sprecyzować zbiór atrybutów obiektu wieloklasowego. Wcześniej jednak zdefiniujemy operator sumy dla zbiorów atrybutów.

Sumą atrybutów obiektów x i y , którą oznaczamy $A(x) \oplus A(y)$, nazywamy sumę następujących zbiorów:

1. $\{A(x) \div A(y)\}$
2. $\{A(x) \cap A(y) \wedge (a, x) = (a, y)\}$
3. Dla każdego $a \in A(x) \cap A(y)$, jeśli $(a, x) \neq (a, y)$ to tworzymy dwa nowe atrybuty, który oznaczamy odpowiednio $x \perp a$ i $y \perp a$.

W punkcie 3 definicji sumy atrybutów zakładamy, że gdy następuje zmiana nazwy atrybutu dziedziczonego to zmiana ta jest automatycznie propagowana do wszystkich obiektów w górę hierarchii obiektów aż do obiektu, w którym atrybut został zdefiniowany pierwszy raz. Ponadto znak ' $\perp$ ' jest znakiem specjalnym, który nie może być używany w nazwach atrybutów nadawanych przez projektanta bazy danych.

Definicja sumy atrybutów odnosi się także do sumy atrybutów dla podobiektów. Zapis takiego wyrażenia jest następujący: $A_{\text{Sub}}(x) \oplus A_{\text{Sub}}(y)$.

W przypadku, gdy suma atrybutów dotyczy wielu zbiorów atrybutów będziemy stosować skrócony zapis $\oplus A(y_i)$, co oznacza $A(y_1) \oplus A(y_2) \oplus \dots \oplus A(y_n)$.

Atrybuty obiektu x spełniają następujący warunek:

$$(7) \quad A(x) = \bigoplus_{id(y) \in Sup(x)} A_{Sub}(y) \oplus \bar{A}(x).$$

Gdy x jest obiektem-klasą to zachodzi również warunek:

$$(8) \quad A_{Sub}(x) = \bigoplus_{id(y) \in Sup(x)} A_{Sub}(y) \oplus \bar{A}_{Sub}(x).$$

Warunki te kończą definicję obiektu.

◆

W pracy nie będziemy zajmować się problemami związanymi z metodami. Przyjmujemy definicję metody przedstawioną w pracy [Abi95].

Z definicji wynika, że obiekt wieloklasowy może występować w bazie jako obiekt-instancja lub jako obiekt-klasa w tradycyjnym rozumieniu tych pojęć.

Definicja

Obiekt wieloklasowy x nazywamy **obiektem-instancją**,

jeżeli $A_{Sub}(x) = \emptyset$ oraz $Sub(x) = \emptyset$.

Obiekt wieloklasowy x nazywamy **obiektem-klasą**,

jeżeli $A_{Sub}(x) \neq \emptyset$.

◆

Teraz na przykładach wyjaśnimy interpretację przedstawionych definicji.

Składniki definicji obiektu $id(x)$ i $v(x)$ opisują obiekt jako instancję klasy, natomiast składniki $id(x)$, $A_{Sub}(x)$ i $M(x)$ charakteryzują obiekt jako klasę. Ponadto, definicja obiektu zawiera jeszcze dwa składniki: $Sub(x)$ i $Sup(x)$. Każdy z nich jest zbiorem identyfikatorów. Konkretnie, zbiór $Sub(x)$ zawiera identyfikatory tych obiektów, które są instancjami obiektu x jako klasy, natomiast zbiór $Sup(x)$ zawiera identyfikatory obiektów będących klasami, do których obiekt x należy jako instancja.

Przykład

Posłużymy się przykładem obiektowej bazy dla kuratorium (Rysunek 2.1). W bazie można wyróżnić dwa rodzaje obiektów: obiekt-klasa i obiekt-instancja.

Pierwszy rodzaj obiekt-klasa może zawierać podobiekty i nadobiekty oraz posiadać własne atrybuty, np.

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M
Szkoła_11	Klasa_2a, Klasa_2b, Koło_mat,	Dolny_Śląsk	Nazwa_jedn, Liczba_uczniów	<Nr_szkoły:11, Dyrektor:Nauczyciel_1, Liczba_uczniów:122>	Śred arob

Przykładem obiektu-klasy, który nie posiada wartości jest:

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M
Dolny_Śląsk	Szkoła_11, Szkoła_12	∅	Nr_szkoły, Dyrektor, Liczba_uczniów	<∅>	∅

Drugi rodzaj obiekt-instancja nie zawiera podobiektów i zbiór atrybutów dla podobiektów jest pusty, np.

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M
Uczeń_1	∅	Klasa_2a, Uczeń, Osoba	∅	<Imię:Maciej, Nazwisko:Kruk, Rok_ur:1990, Płeć:M, Ocena:5.0, 4.0, 3.5>	Średn cen

W obiektach możemy wyróżnić zbiór atrybutów i ich wartości, np. w obiekcie Uczeń_1 mamy <Imię:Maciej, Nazwisko:Kruk, Rok_ur:1990, Płeć:M, Ocena:5, 4.0, 3.5> oraz zbiór atrybutów dla jego podobiektów, np. w obiekcie Dyrekcja mamy {Imię, Nazwisko, Przedmiot, Dodatek}.



W przypadku, gdy obiekt jest klasą i wartość obiektu jest zbiorem pustym mamy do czynienia z klasycznym dziedziczeniem. Analizując zbiór atrybutów A_{Sub} opisujących podobiektu możemy poklasyfikować obiekty.

Atrybuty muszą spełniać trzy warunki: (5), (7) i (8) podane w definicji obiektu. Pierwszy warunek specyfikuje zależność między zbiorem atrybutów obiektu i zbiorem atrybutów podobiektu. Mówi, że zbiór atrybutów podobiektów jest nadzbiorem zbiorów atrybutów odpowiednich obiektów. Warunki drugi i trzeci wynikają z wieloklasowości oraz własności pochodzenia atrybutu. Drugi dotyczy obiektu i określa zbiór atrybutów jako sumę atrybutów lokalnych i atrybutów przekazanych z nadobiektów. Trzeci dotyczy klas i specyfikuje obowiązkowe atrybuty w wartościach podobiektów.

Aby ustalić czy zbiory podobiektów i nadobiektów są poprawne należy sprawdzić czy spełnione są warunki wyspecyfikowane w formalnej definicji obiektu. Wprost z definicji obiektu wynikają dwie własności.

1. Obiekt nie może być swoim podobiektem ani nadobiektem.
2. Jeżeli obiekt x jest podobiektu y to y jest nadobiektu x .

Gdy obiekt jest podobiektu wielu obiektów to może wystąpić konflikt atrybutów. Aby rozwiązać ten problem wprowadziliśmy pojęcie pochodzenia atrybutu. Intuicyjnie, jeśli atrybut należy do przecięcia zbiorów atrybutów nadobiektów to możemy mieć do czynienia z takim samym pochodzeniem atrybutu lub dwoma różnymi pochodzeniami atrybutu. W pierwszym przypadku przechowujemy tylko jeden atrybut natomiast w drugim przypadku będziemy przechowywać oba i to ze zmienionymi nazwami, np. dodamy przedrostek w postaci nazwy obiektu źródłowego do nazwy konfliktowego atrybutu.

Przykład

$A_{\text{Sup}}(\text{Osoba}) = \{\text{Imię, Nazwisko}\}.$

$A_{\text{Sup}}(\text{Uczeń}) = \{\text{Imię, Nazwisko, Płeć}\}.$

$\bar{A}(\text{Uczeń}_1) = \{\text{Ocena}\}.$

$A(\text{Uczeń}_1) = A_{\text{Sup}}(\text{Osoba}) \oplus A_{\text{Sup}}(\text{Uczeń}) \oplus \bar{A}(\text{Uczeń}_1) = \{\text{Imię, Nazwisko, Płeć, Ocena}\}.$

Zbiór atrybutów obiektu Uczeń_1 zawiera atrybuty podane w obiekcie Osoba oraz obiekcie Uczeń (pomijamy w przykładzie przynależność do Klasy_2a). Ponadto zawiera atrybut

lokalny Ocena. Atrybuty Imię i Nazwisko występują jednokrotnie, gdyż ich pochodzenie dla obiektu Osoba i obiektu Uczeń jest identyczne.

◆

Przykład

$$A_{\text{Sup}}(\text{Osoba}) = \bar{A}_{\text{Sup}}(\text{Osoba}) = \{\text{Imię, Nazwisko}\}.$$

$$A_{\text{Sup}}(\text{Student}) = \{\text{Nr, Imię, Nazwisko, Kierunek}\}.$$

$$A_{\text{Sup}}(\text{Pracownik}) = \{\text{Nr, Imię, Nazwisko}\}.$$

$$\bar{A}_{\text{Sup}}(\text{Prac_Student}) = \emptyset \text{ oraz } \text{Osoba} \leq \text{Student i Osoba} \leq \text{Pracownik wtedy}$$

$$A_{\text{Sup}}(\text{Prac_Student}) = A_{\text{Sup}}(\text{Student}) \oplus A_{\text{Sup}}(\text{Pracownik}) = \{\text{Imię, Nazwisko, Student_Nr, Kierunek, Pracownik_Nr}\}.$$

Zbiór atrybutów obiektu Prac_Student zawiera m. in. dwa atrybuty: Student_Nr i Pracownik_Nr powstałe z atrybutów Nr obiektów: Student i Pracownik. Zmiana nazwy nastąpiła, ponieważ atrybuty te miały różne pochodzenie.

◆

Można odwrócić problem wieloklasowości i rozpatrywać go jako możliwość wielokrotnego występowania obiektu w tej samej klasie. Np. wśród pracowników naukowych uwzględniając przynależność instytutową lub zakładową niektórzy pracownicy mogą wystąpić kilka razy. Jednakże, takie postawienie problemu jest niepotrzebne ze względu na małą możliwość wykorzystania. W związku z tym, w pracy nie będziemy zajmować się problemem wielokrotnego występowania tego samego obiektu w jednej klasie.

2.1.3. Przykładowa baza obiektów

W prezentowanym przykładzie obiektowej bazy danych dla kuratorium (rys. 2.1) widzimy różne zależności występujące między obiektami.

Obiektem z podobiektami nie posiadającym nadobiektów jest obiekt Dolny_Śląsk. Z opisu jego atrybutów dla podobiektów wynika, że jego podobiekty: Szkoła_11 i Szkoła_12 muszą posiadać dyrektora. Wartością atrybutu Dyrektor w szkołach jest identyfikator innego obiektu, a więc mamy do czynienia z atrybutem referencyjnym.

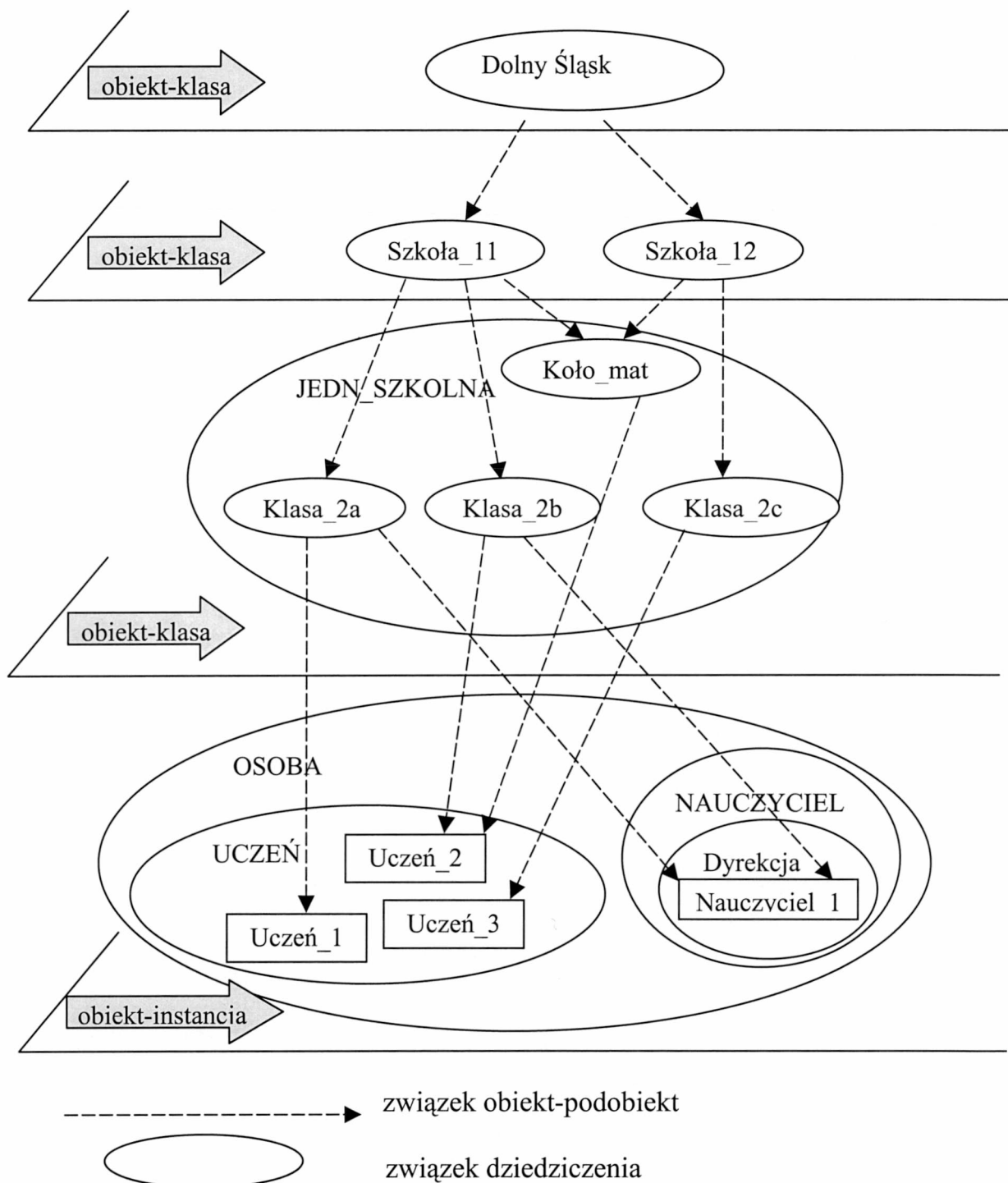
Innymi przykładami obiektów bez nadobiektów są Jedn_szkolna i Osoba. Obiekty te nie posiadają również wartości. Mają jednak podobiekty.

Występują jeszcze obiekty-instancje, np. nauczyciele i uczniowie.

Analizując zbiory podobiektów, zbiory nadobiektów, wartości referencyjne dochodzimy do wniosku, że obiekty mogą być wzajemnie powiązane. Aby zbadać te zależności wprowadzimy kilka pomocniczych pojęć, a mianowicie: równość obiektów, związek obiekt-podobiekt, związek dziedziczenia, schemat obiektów wieloklasowych oraz bazę obiektów wieloklasowych.

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M
Dolny_Śląsk	Szkoła_11, Szkoła_12	∅	Nr szkoły, Dyrektor, Liczba_uczniów	<∅>	∅
Szkoła_11	Klasa_2a, Klasa_2b, Koło_mat	Dolny_Śląsk	Nazwa_jedn, Liczba_uczniów	<Nr_szkoly:11, Dyrektor:Nauczyciel_1, Liczba_uczniów:2>	Średn arobk
Szkoła_12	Klasa_2c, Koło_mat	Dolny_Śląsk	Nazwa_jedn, Liczba_uczniów	<Nr_szkoly:12, Dyrektor:Nauczyciel_1, Liczba_uczniów:1>	Średn arobk
Jedn_szkolna	Klasa_2a, Klasa_2b, Klasa_2c, Koło_mat	∅	Liczba_uczniów	<∅>	∅
Klasa_2a	Uczeń_1, Nauczyciel_1	Szkoła_11, Jedn_szkolna	Imię, Nazwisko, Rok_ur	<Nazwa_jedn:2a, Liczba_uczniów:1, Liczba_chłopców:1>	Średn cen
Klasa_2b	Uczeń_2, Nauczyciel_1	Szkoła_11, Jedn_szkolna	Imię, Nazwisko, Rok_ur	<Nazwa_jedn:2b, Liczba_uczniów:1>	Średn cen
Klasa_2c	Uczeń_3	Szkoła_12, Jedn_szkolna	Imię, Nazwisko	<Nazwa_jedn: 2c, Liczba_uczniów:1>	Średn cen
Koło_mat	Uczeń_2	Szkoła_11, Szkoła_12, Jedn_szkolna	Imię, Nazwisko	<Nazwa_jedn: Koło_matematyczne, Liczba_uczniów:1>	∅
Osoba	Uczeń, Nauczyciel, Dyrekcja, Uczeń_1, Uczeń_2, Uczeń_3, Nauczyciel_1	∅	Imię, Nazwisko	<∅>	Wiek
Uczeń	Uczeń_1, Uczeń_2, Uczeń_3	Osoba	Imię, Nazwisko, Płeć	<∅>	∅
Nauczyciel	Dyrekcja, Nauczyciel_1	Osoba	Imię, Nazwisko, Przedmiot	<∅>	Liczb lonis
Nauczyciel_1	∅	Klasa_2a, Klasa_2b, Dyrekcja, Nauczyciel, Osoba	∅	<Imię:Beata, Nazwisko:Bąk, Rok_ur:1960, Przedmiot:język polski, Dodatek:100>	∅
Dyrekcja	Nauczyciel_1	Nauczyciel, Osoba	Imię, Nazwisko, Przedmiot, Dodatek	<∅>	∅
Uczeń_1	∅	Klasa_2a, Uczeń, Osoba	∅	<Imię:Maciej, Nazwisko:Kruk, Rok_ur:1990, Płeć:M, Ocena:5.0, 4.0, 3.5>	Średn cen
Uczeń_2	∅	Klasa_2b, Koło_mat, Uczeń, Osoba	∅	<Imię:Jan, Nazwisko:Kula, Rok_ur:1990, Płeć:M, Ocena:4.5>	Średn cen
Uczeń_3	∅	Klasa_2c, Uczeń, Osoba	∅	<Imię:Alicja, Nazwisko:Nowak, Ocena:3.0, 2.0, Płeć:K>	Średn cen

Rysunek 2.1. Przykłady obiektów bazy Kuratorium



Rysunek 2.2. Schemat powiązań obiektów w bazie Kuratorium

2.1.4. Równość obiektów wieloklasowych

Wprowadzamy dwa typy równości obiektów: identyczność i powierzchowną równość.

Definicja

Obiekty x i y nazywamy **identycznymi** wtedy i tylko wtedy, gdy $\text{id}(x) = \text{id}(y)$ i **powierzchownie równymi** wtedy i tylko wtedy, gdy $v(x) = v(y)$.

Identyczność obiektów x i y oznaczamy przez $x = y$ a powierzchnią równość przez $x \cong y$.



Na podstawie powyższych definicji można stwierdzić, że identyczność obiektów implikuje ich powierzchnią równość. Zwróćmy uwagę, że powierzchnią równość obiektów nie wymaga, by te obiekty miały wspólne nadobiekty, ani też to samo pochodzenie atrybutów. Wystarczy równość atrybutów i ich wartości.

2.1.5. Związki między obiektami wieloklasowymi

Definicja obiektu oprócz identyfikatora, wartości i metod zawiera również identyfikatory powiązanych z nim nadobiektów i podobiektów. W ten sposób została określona nowa hierarchia pomiędzy obiektami-klasami i obiektami-instancjami (rys. 2.2). Związek ten nie należy do związku całość-część ani do związku podklasa-nadklasa.

Definicja

Mówimy, że obiekty x i y występują w **związku obiekt-podobiekt** jeżeli $y \in \text{Sub}(x)$.

Związek obiekt-podobiekt między obiektami x i y oznaczamy symbolem $x \angle y$.



Przykład

W bazie danych Kuratorium mamy obiekty:

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
Dolny_Śląsk	Szkoła_11, Szkoła_12	∅	Nr_szkoły, Dyrektor, Liczba_uczniów	<∅>	∅

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
Szkoła_11	Klasa_2a, Klasa_2b, Koło_mat,	Dolny_Śląsk	Nazwa_jedn, Liczba_uczniów	<Nr_szkoły:11, Dyrektor:Nauczyciel_1, Liczba_uczniów:122>	Średnia_z arobków

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
Klasa_2a	Uczeń_1	Szkoła_11 Jedn_szkolna	Imię, Nazwisko, Rok_ur	<Nazwa_jedn:2a, Liczba_uczniów:19, Liczba_chłopców:10>	Średnia_o cen

Zatem: Dolny_Śląsk jest nadobiektom obiektu Szkoła_11, a Szkoła_11 jest nadobiektom obiektu Klasa_2a, czyli możemy zapisać $\text{Dolny_Śląsk} \angle \text{Szkoła_11} \angle \text{Klasa_2a}$.



Jeżeli związek obiekt-podobiekt zostanie ograniczony do związku między obiektami-klasami to otrzymamy klasyczny związek dziedziczenia.

Definicja

Związek obiekt-podobiekt nazywamy **związkiem dziedziczenia** wtedy i tylko wtedy,

gdy $(x \angle y)$ i $A_{\text{Sub}}(y) \supseteq A_{\text{Sub}}(x)$.

Związek dziedziczenia oznaczamy symbolem $x \leq y$.

♦

Jeżeli między obiektami x i y występuje związek dziedziczenia $x \leq y$ to mówimy, że y **dziedziczy** od x i x jest **dziedziczone przez** y .

Z definicji związku dziedziczenia wynikają następujące własności:

1. $x \leq y \Rightarrow \text{Sub}(x) \subseteq \text{Sub}(y)$
2. $x \leq y \Rightarrow \text{id}(x) \in \text{Sup}(y)$
3. $x \leq y \Rightarrow x$ jest obiektem-klasą i y jest obiektem-klasą
4. $(x \leq y) \wedge (y \leq z) \Rightarrow (x \leq z)$

Własność 4 nazywamy **przechodnością dziedziczenia**.

Zatem klasyczne wielokrotne dziedziczenie to nic innego jak wielokrotne występowanie związku obiekt-podobiekt, związku jednego podobiektu z kilkoma nadobiektami. Wnosząc dalej wielokrotne dziedziczenie jest równoważne obiektowi wieloklasowemu, a uogólnione homogeniczne pojęcie obiektu i klasy, w szczególności związek obiekt-podobiekt, daje możliwość realizacji wieloklasowości, tj. występowania tego samego obiektu w różnych klasach.

2.1.6. Schemat obiektów, baza obiektów

Związek obiekt-podobiekt służy (między innymi) definicjom schematu i bazy obiektów wieloklasowych.

Definicja

Schematem obiektów wieloklasowych S nazywamy parę $(\text{OBJ}, \angle)$,

gdzie OBJ jest zbiorem obiektów wieloklasowych, natomiast $\angle$ jest związkiem obiekt-podobiekt zdefiniowanym dla obiektów należących do OBJ .

Bazą obiektów wieloklasowych nazywamy parę (S, MOW) ,

gdzie S jest schematem obiektów wieloklasowych, natomiast MOW jest językiem manipulacji obiektami wieloklasowymi.

♦

Język MOW będzie opisany w rozdziale piątym.

2.2. Algebra obiektów

Operatory algebraiczne możemy podzielić na dwie grupy:

1. Operatory nie zmieniające struktury obiektów: selekcja, asocjacja, złączenie, suma, przecięcie, różnica i usunięcie duplikatów.
2. Operatory wykorzystywane do zmiany struktury obiektów: projekcja, spłaszczenie, zagnieżdżenie i rozgnieżdżenie.

Operatory znane z algebry relacyjnej i algebry obiektów złożonych zostały zaadoptowane do wymogów stawianych przed modelem obiektów wieloklasowych.

2.2.1. Selekcja

Operacja selekcji tworzy zbiór obiektów spełniających warunek zdefiniowany za pomocą predykatu.

Definicja

Operatorem **selekcji** nazywamy operator σ spełniający następujący warunek:

$$\sigma(X, p) = \{x \mid x \in X \wedge p(x)\},$$

gdzie $X \subseteq \text{OBJ}$ i p jest predykatem języka logiki pierwszego rzędu.

♦

Termami predykatu mogą być stałe, zmienne i wyrażenia ścieżkowe. Termy łączymy za pomocą operatorów boolowskich $\wedge$, $\vee$ i $\neg$. Mogą być uogólniane $\forall x$ lub uszczegółowiane $\exists x$.

Przykład

$W = \sigma(\text{Sub}(\text{Osoba}), \text{Sub}(\text{Osoba}).\text{Nazwisko} = \text{"Bąk"})$.

W przykładzie warunek $\text{Sub}(\text{Osoba}).\text{Nazwisko} = \text{"Bąk"}$ oznacza, że obiekt, który go spełnia zostanie wybrany do zbioru wyjściowego W . Wynikiem operatora selekcji w bazie Kuratorium jest zbiór jednoelementowy $\{\text{Nauczyciel}_1\}$.

♦

Prezentowana algebra umożliwia przechowywanie wyników operacji algebraicznych w tymczasowych zbiorach obiektów, które mogą być wykorzystane w argumentach innych operacji.

Przykład

$W = \sigma(\text{Sub}(\text{Osoba}), \text{Sub}(\text{Osoba}).\text{Rok_ur} = 1990)$.

$Z = \sigma(\text{Sub}(\text{Klasa_2a}), g.\text{Ocena} \in \{5.0, 4.0\} \wedge g \in W)$.

Wynik pierwszej selekcji jest przechowywany w zbiorze W . W drugiej operacji przeszukujemy uczniów klasy 2a z ocenami 5.0 i 4.0 a ponadto sprawdzamy czy znajdują się w zbiorze W . Wynikiem jest zbiór jednoelementowy $\{\text{Uczeń}_1\}$.

♦

2.2.2. Asocjacja

Operator asocjacji zwraca dla każdego obiektu obiekty związane referencyjnie. Powiązanie obiektów określa atrybut a . Jeśli w zbiorze wynikowym pojawią się identyczne

obiekty to są one pomijane. Znajdujemy, więc podobieństwo operatora asocjacji z relacyjnym operatorem projekcji ograniczonym do jednego atrybutu wynikowego.

Definicja

Operatorem **asocjacji** nazywamy operator τ spełniający następujący warunek:

$$\tau(X, a) = \{y \mid y = V_a(x) \wedge x \in X\},$$

gdzie $X \subseteq \text{OBJ}$ i $a \in A(x)$.

◆

Przykład

$W = \tau(\text{Sub}(\text{Dolny Śląsk}), \text{Dyrektor})$.

◆

Wynikiem asocjacji będą dyrektorzy szkół z Dolnego Śląska, czyli dla bazy Kuratorium {Nauczyciel_1}. Jak widzimy, podstawowym zbiorem przeszukiwań był zbiór szkół, natomiast w wyniku otrzymaliśmy obiekty dyrektorów powiązane przez atrybut referencyjny Dyrektor. W przypadku powtórzenia elementu (obiekt identyczny lub wartość pusta) operator asocjacji usuwa go ze zbioru wynikowego.

◆

2.2.3. Projekcja

Operator projekcji rozszerza możliwości operatora asocjacji przez umożliwienie użycia wielu atrybutów.

Definicja

Operatorem **projekcji** nazywamy operator π spełniający następujący warunek:

$$\pi(X, a_1, a_2, \dots, a_n) = \{x \mid A(x) = \{a_1, a_2, \dots, a_n\} \wedge x \in X\},$$

gdzie $X \subseteq \text{OBJ}$ i $a_i \in A(x)$.

◆

Projekcja zwraca zbiór obiektów, w którym zmieniono krotkę wartości obiektu, dla każdego obiektu ze zbioru X . Obiekty wynikowe mogą być powierzchownie równe. Na końcu rozdziału zostanie opisany operator do usuwania tego rodzaju duplikatów.

Jeśli dla dowolnego $x \in X$ zbiór atrybutów $\{a_1, a_2, \dots, a_n\}$ nie jest podzbiorem zbioru wymaganych atrybutów dla obiektów wynikających z własności np. związku obiekt-podobieństwo to operator projekcji zwróci błąd.

Projekcja odpowiada relacyjnemu operatorowi projekcji ograniczając atrybuty obiektów. Moc zbioru wejściowego i wynikowego po zastosowaniu operatora projekcji jest taka sama.

Przykład

$W = \pi(\text{Sub}(\text{Uczeń}), \text{Imię}, \text{Nazwisko}, \text{Płeć})$.

W tym przypadku ze zbioru atrybutów obiektów $\text{Sub}(\text{Uczeń})$ zostały usunięte wszystkie atrybuty uczniów poza atrybutem Imię, Nazwisko i Płeć.

◆

2.2.4. Złączenie

Operator złączenie jest operatorem używanym do łączenia obiektów z dwóch kolekcji.

Definicja

Operatorem **złączenia** nazywamy operator χ spełniający następujący warunek:

$$\chi(X, Y, p) = \{\{x, y\} \mid x \in X \wedge y \in Y \wedge p(x, y)\},$$

gdzie $X, Y \subseteq \text{OBJ}$ i p jest predykatem języka logiki pierwszego rzędu.

◆

Przykład

$W = \chi(\text{Sub}(\text{Nauczyciel}), \text{Sub}(\text{Uczeń}), \text{Sub}(\text{Nauczyciel}).\text{Imię} = \text{Sub}(\text{Uczeń}).\text{Imię})$.

Operacja połączy nauczycieli z uczniami o tych samych imionach. Wynikiem operatora złączenia jest w tym przypadku zbiór par obiektów spełniających warunek identyczności imion.

◆

2.2.5. Suma, różnica i przecięcie

Operatory suma, różnica i przecięcie są operatorami używanymi do tworzenia nowych kolekcji obiektów bazując na identyczności obiektów. Ich pochodzeniem jest teoria zbiorów.

Definicja

Operatorem **sumy** nazywamy operator $\cup$ spełniający następujący warunek:

$$X \cup Y = \{x \mid x \in X \vee x \in Y\},$$

gdzie $X, Y \subseteq \text{OBJ}$.

◆

Przykład

$W = \text{Sup}(\text{Klasa_2a}) \cup \text{Sup}(\text{Klasa_2b}) = \{\text{Szkola_11}, \text{Jedn_szkolna}\}$.

◆

Definicja

Operatorem **przecięcia** nazywamy operator $\cap$ spełniający następujący warunek:

$$X \cap Y = \{x \mid x \in X \wedge x \in Y\},$$

gdzie $X, Y \subseteq \text{OBJ}$.

◆

Przykład

$$W = \text{Sub}(\text{Szkoła_11}) \cap \text{Sub}(\text{Szkoła_12}) = \{\text{Koło_mat}\}.$$

◆

Definicja

Operatorem **różnicy** nazywamy operator – spełniający następujący warunek:

$$X - Y = \{x \mid x \in X \wedge x \notin Y\},$$

gdzie $X, Y \subseteq \text{OBJ}$.

◆

Przykład

$$W = \text{Sub}(\text{Osoba}) - \text{Sub}(\text{Nauczyciel}) = \{\text{Uczeń}, \text{Nauczyciel}, \text{Uczeń_1}, \text{Uczeń_2}, \text{Uczeń_3}\}.$$

◆

Podobnie jak w teorii zbiorów, także dla obiektów spełniony jest warunek:

$$X \cap Y = X - (X - Y).$$

Przykład

$$W = \text{Sub}(\text{Szkoła_11}) \cap \text{Sub}(\text{Szkoła_12}) = \{\text{Koło_mat}\}.$$

$$Z = \text{Sub}(\text{Szkoła_11}) - (\text{Sub}(\text{Szkoła_11}) - \text{Sub}(\text{Szkoła_12})) = \{\text{Koło_mat}\}.$$

Zatem $W=Z$.

◆

2.2.6. Spłaszczenie

Operator spłaszczenia jest używany do rekonstrukcji zagnieżdżonych struktur takich jak zbiór zbiorów obiektów. Operator ten najczęściej występuje po operatorze złączenia lub asocjacji, gdy te zwracają zbiór zbiorów obiektów. Spłaszczenie usuwa nawiasy, przez co zezwala na pojedyncze przetwarzanie elementów zbioru.

Definicja

Operatorem **spłaszczenia** nazywamy operator ε spełniający następujący warunek:

$$\varepsilon(X) = \{x \mid y \in X \wedge x \in y\},$$

gdzie $\varepsilon(X) \subseteq \text{OBJ}$.

◆

Przykład

$$W = \chi(\text{Sub}(\text{Nauczyciel}), \text{Sub}(\text{Uczeń}), (\text{Sub}(\text{Nauczyciel}).\text{Rok_ur}+18) < \text{Sub}(\text{Uczeń}).\text{Rok_ur}).$$

Wynik złączenia jest zbiorem zbiorów $\{\{\text{Nauczyciel_1}, \text{Uczeń_1}\}, \{\text{Nauczyciel_1}, \text{Uczeń_2}\}\}$.

Po zastosowaniu operatora spłaszczenia otrzymamy:

$$Z = \varepsilon(W) = \{\text{Nauczyciel_1}, \text{Uczeń_1}, \text{Uczeń_2}\}.$$

Wynik operatora spłaszczenie nie zawiera powtórzeń. Ponadto warunek predykatu dla złączenia nie jest spełniony dla obiektu Uczeń_3 , więc został on pominięty w zbiorze wynikowym.

◆

2.2.7. Zagnieżdżenie, rozgnieżdżenie

Operatory zagnieżdżenie i rozgnieżdżenie są operatorami dla relacji nie spełniających warunku pierwszej postaci normalnej zaadaptowanymi dla obiektów. Operatory działają na wartościach obiektów. W przypadku pierwszego operatora krotka wartości zawiera obiekty, natomiast w przypadku drugiego operatora krotka wartości zawiera zbiory obiektów. Operator zagnieżdżenie porównuje wartości atrybutów i tworzy nowe niepowtarzalne krotki. Drugi operator na ogół odtwarza sytuację przed zagnieżdżeniem. Oba operatory mogą tworzyć nowe obiekty powierzchownie równe do istniejących.

Definicja

Operatorem **zagnieżdżenia** nazywamy operator γ spełniający następujący warunek:

$$\gamma(X, a_i) = \{x \mid \langle a_1:s.a_1, \dots, a_i:t, \dots, a_n:s.a_n \rangle \wedge \forall r \exists s (r \in t \wedge s \in X \wedge s.a_i = r)\},$$

gdzie $X \subseteq \text{OBJ}$, x poza wartością posiada $\text{id}(x)$, pozostałe elementy są równe $\emptyset$.

◆

Definicja

Operatorem **roznieżdżenia** nazywamy operator μ spełniający następujący warunek:

$$\mu(X, a_i) = \{x \mid \langle a_1:s.a_1, \dots, a_i:t, \dots, a_n:s.a_n \rangle \wedge s \in X \wedge t \in s.a_i\},$$

gdzie $X \subseteq \text{OBJ}$, x poza wartością posiada $\text{id}(x)$, pozostałe elementy są równe $\emptyset$.

◆

Drugi operator na ogół odtwarza sytuację przed zagnieżdżeniem, choć nie jest operatorem odwrotnym do operatora zagnieżdżenie, tworzy inne obiekty.

Przykład

$$X = \{x_1, x_2, x_3, x_4\}$$

$$v(x_1) = \langle a: y_1, b: y_2 \rangle$$

$$v(x_2) = \langle a: y_1, b: y_3 \rangle$$

$$v(x_3) = \langle a: y_2, b: y_4 \rangle$$

$$v(x_4) = \langle a: y_1, b: y_2 \rangle$$

$$\gamma(X, b) = \{x_5 \mid \langle a: y_1, b: \{y_2, y_3\} \rangle, x_6 \mid \langle a: y_2, b: \{y_4\} \rangle\} = \{x_5, x_6\} = Y$$

$$\mu(Y, b) = \{x_7 \mid \langle a: y_1, b: y_2 \rangle, x_8 \mid \langle a: y_1, b: y_3 \rangle, x_9 \mid \langle a: y_2, b: y_4 \rangle\} = \{x_7, x_8, x_9\} \neq X.$$

◆

Jeśli zbiór X zawiera obiekty powierzchownie równe to $\mu(\gamma(X, a), a)$ nie jest powierzchownie równe z X . Zakładamy, że równość powierzchowna zbiorów oznacza

równość powierzchowną par obiektów, w których jeden element pochodzi z pierwszego zbioru, a drugi pochodzi odpowiednio z drugiego zbioru.

Podobnie dla kolejnego przykładu wyrażenie $\gamma (\mu (Y, a), a)$ nie jest powierzchownie równe z Y .

Przykład

$$v(x_5) = \langle a: y_1, b: \{y_2, y_3\} \rangle$$

$$v(x_2) = \langle a: y_1, b: y_3 \rangle$$

$$Y = \{x_5, x_2\}.$$

$$\mu (Y, b) = \{x_{10} \mid \langle a: y_1, b: y_2 \rangle, x_{11} \mid \langle a: y_1, b: y_3 \rangle\} = \{x_{10}, x_{11}\} = X.$$

$$\gamma (X, b) = \{x_{12} \mid \langle a: y_1, b: \{y_2, y_3\} \rangle\} \neq Y.$$

♦

2.2.8. Usunięcie duplikatów

Operator usunięcia duplikatów zapewnia, że tylko jedna kopia obiektu ze zbioru obiektów znajduje się w zbiorze. Kopia obiektu odnosi się do definicji równości powierzchownej obiektów. Usuwanie obiektów identycznych jest zapewnione automatycznie – zbiór wynikowy dla operatorów algebraicznych nie posiada dwóch takich samych elementów. W szczególności operator usunięcia duplikatów ma zastosowanie przy tworzeniu nowych obiektów, a więc np. po zastosowaniu operatora projekcji czy roznieżdzenia.

Definicja

Operatorem **usunięcia duplikatów** nazywamy operator θ spełniający następujący warunek:

$$\theta (X) = \{x \mid \forall x \neg \exists y (x \cong y)\},$$

gdzie $X \subseteq \text{OBJ}$, $x \in X$.

♦

Przykład

Niech $\gamma (X, b) = \{x_7 \mid \langle a: y_1, b: y_2 \rangle, x_8 \mid \langle a: y_1, b: y_2 \rangle, x_9 \mid \langle a: y_2, b: y_4 \rangle\} = Y$, więc

$$\theta (Y) = \{x_7, x_9\}.$$

♦

2.3. Wnioski

Podana definicja obiektu wieloklasowego umożliwia jednakowe traktowanie obiektu i klasy oraz pozwala na wielokrotną przynależność obiektu do klasy.

Taką strukturę, jak się okaże w następnym rozdziale, można wykorzystać do wprowadzania zmian w obiektach a w szczególności przy migracji obiektów.

Prezentowana algebra obiektowa charakteryzuje się tym, że wynikiem zapytania jest zbiór obiektów. Elementami odpowiedzi mogą być zarówno istniejące obiekty jak i nowe obiekty, utworzone podczas przetwarzania.

Operatory algebry obiektowej są podobne pod względem syntaktyki do operatorów w algebrze relacyjnej. Semantycznie jednakże różnią się one w zasadniczy sposób. Podstawą wszystkich operatorów algebraicznych jest identyfikator obiektu. Poza nim dużą rolę odgrywa wartość obiektu oraz zbiory podobieństw i zbiory nadobiektów. Wprowadzono też dwie definicje równości między obiektami: identyczność i równość powierzchowną, która jest wykorzystywana przy usuwaniu duplikatów.

Z definicji podstawowych operatorów wynika, że obiekty będące wynikiem sumy, różnicy, przecięcia, selekcji, asocjacji, projekcji, złączenia, spłaszczenia, zagnieżdżenia, rozgnieżdżenia i usunięcia duplikatów są obiektami wieloklasowymi.

Porównując operatory algebry obiektów oraz operatory relacyjne widzimy, że operatory algebry obiektów po dokonaniu niewielkich zmian mogłyby być wykorzystane w systemach obiektowo-relacyjnych.

Kolejnym problemem, który będziemy analizować jest zdefiniowanie podstawowego zbioru operatorów zmian w bazie danych za pomocą przyjętych operatorów algebry.

3. Migracja obiektów

Podstawowe problemy występujące przy zmianach w klasach i obiektach opiszemy na przykładzie bazy danych UniSQL [Uni95a, Uni95b]. Wybraliśmy bazę UniSQL z kilku powodów. Po pierwsze, jest jednym z najpopularniejszych systemów obiektowo-relacyjnych [Klo00]. Po drugie, zawiera kompletny zestaw podstawowych operacji niezbędnych do dokonywania zmian w tradycyjnym schemacie bazy danych. Aby uniknąć nieporozumień będziemy w tym przypadku używać klasycznej nazwy klasy i obiektu, ponieważ baza danych UniSQL nie jest przystosowana do prezentowanego w pracy homogenicznego podejścia do obiektu i klasy.

W drugim podrozdziale, powracamy do modelu obiektów wieloklasowych, porządkujemy możliwe zmiany obiektów i definiujemy je formalnie za pomocą opisanych w rozdziale drugim operatorów algebry.

W kolejnych podrozdziałach wyodrębniamy ze zbioru zmian te, które dotyczą migracji obiektów jako najbardziej charakterystycznej funkcji realizowanej w modelu obiektów wieloklasowych.

3.1. Podstawowe zmiany w UniSQL

W bazie danych UniSQL mogą być stosowane dwa rodzaje zmian. Pierwszy rodzaj zmian dotyczy obiektów. Do tej kategorii zalicza się dodawanie obiektu, usuwanie obiektu, dodawanie czy usuwanie atrybutu/metody oraz zmianę atrybutu lub metody. Drugi rodzaj zmian obejmuje modyfikacje klas bazy danych, takie jak: dodawanie klasy, usuwanie klasy oraz wprowadzanie zmian w hierarchii klasa-podklasa.

Do podstawowych zmian w UniSQL zaliczamy:

1. Instrukcje dotyczące obiektów.

1.1. Tworzenie obiektu.

```
insert into nazwa_klasy lista_atrybutów  
values lista_wartości;
```

1.2. Usunięcie obiektu.

```
delete from nazwa_klasy where warunek;
```

1.3. Dodanie atrybutu.

```
alter class nazwa_klasy  
add attribute nazwa_atrybutu typ_atrybutu;
```

1.2. Usunięcie atrybutu.

```
alter class nazwa_klasy  
drop nazwa_atrybutu;
```

1.3. Zmiana nazwy atrybutu.

```
alter class nazwa_klasy  
rename attribute nazwa_atrybutu as nazwa_atrybutu;
```

1.4. Zmiana typu atrybutu.

```
alter class nazwa_klasy  
drop nazwa_atrybutu;  
alter class nazwa_klasy  
add attribute nazwa_atrybutu typ_atrybutu;
```

1.5. Dodanie metody.

```
alter class nazwa_klasy  
add method nazwa_metody;
```

1.6. Usunięcie metody.

```
alter class nazwa_klasy  
drop method nazwa_metody;
```

1.7. Zmiana nazwy metody.

```
alter class nazwa_klasy  
rename method nazwa_metody as nazwa_metody;
```

2. Instrukcje dotyczące klas bazy danych.

2.1. Tworzenie klasy.

```
create class nazwa_klasy lista_atrybutów;
```

2.2. Usunięcie klasy.

```
drop only nazwa_klasy;
```

2.3. Zmiana nazwy klasy.

```
rename class nazwa_klasy as nazwa_klasy;
```

2.4. Dodanie nadklasy.

```
alter class nazwa_klasy  
add superclass nazwa_klasy;
```

2.5. Usunięcie nadklasy.

```
alter class nazwa_klasy  
drop superclass nazwa_klasy;
```

W wykazie podstawowych instrukcji języka UniSQL służących do przeprowadzania zmian celowo pomijamy zmianę kodu realizacji metody jako zależną jedynie od języka implementacji metod. Pomijamy ponadto instrukcje zmian złożonych, np. podział klasy, połączenie klas, gdyż można je zrealizować za pomocą instrukcji zmian podstawowych.

3.2. Operatory zmian – podejście algebraiczne

Na podstawie analizy możliwych zmian w obiektach i klasach przedstawionych w poprzednim rozdziale oraz po uwzględnieniu homogeniczności obiektu i klasy w modelu obiektów wieloklasowych wyróżniamy następujące operatory zmian:

1. Operatory dotyczące obiektu.

- 1.1. Dodanie atrybutu.
- 1.2. Usunięcie atrybutu.
- 1.3. Zmiana nazwy atrybutu.
- 1.4. Zmiana typu atrybutu.
- 1.5. Dodanie metody.
- 1.6. Usunięcie metody.
- 1.7. Zmiana nazwy metody.

2. Operatory dotyczące schematu obiektów.

- 2.1. Tworzenie obiektu.
- 2.2. Usunięcie obiektu.
- 2.3. Zmiana nazwy (identyfikatora) obiektu.
- 2.4. Dodanie obiektu do zbioru podobiektów.
- 2.5. Usunięcie obiektu ze zbioru podobiektów.
- 2.6. Dodanie obiektu do zbioru nadobiektów.
- 2.7. Usunięcie obiektu ze zbioru nadobiektów.

Z powyższego zbioru operatorów zmian **podstawowymi operatorami zmian** określimy:

- 1.1. Dodanie atrybutu.
- 1.2. Usunięcie atrybutu.
- 1.5. Dodanie metody.
- 1.6. Usunięcie metody.
- 2.1. Tworzenie obiektu.
- 2.2. Usunięcie obiektu.
- 2.4. Dodanie obiektu do zbioru podobiektów.
- 2.5. Usunięcie obiektu ze zbioru podobiektów.
- 2.6. Dodanie obiektu do zbioru nadobiektów.
- 2.7. Usunięcie obiektu ze zbioru nadobiektów.

Pozostałe zmiany, pominięte w zbiorze podstawowym można zastąpić sekwencją zmian ze zbioru podstawowego. I tak:

- 1.3 można zastąpić sekwencją 1.2 i 1.1.

1.4 można zastąpić sekwencją 1.2 i 1.1.

1.7 można zastąpić sekwencją 1.6 i 1.5.

2.3 można zastąpić sekwencją 2.2 i 2.1.

Teraz podstawowe operatory zmian zostaną zdefiniowane za pomocą operatorów algebry obiektów wieloklasowych opisanych w rozdziale drugim. Ze względu na określanie zbioru atrybutów dla podobiektów musimy wprowadzić operatory umożliwiające modyfikację tego zbioru atrybutów. Operatory tworzenia i usunięcia obiektu zostały zdefiniowane na końcu, ponieważ przy ich realizacji wykorzystujemy operatory opisane wcześniej.

Definicja

1. Operator **dodania atrybutu (lokalnego)** a_i do obiektu x jest to podstawienie

$$x = \gamma(x, a_i).$$

2. Operator **usunięcia atrybutu (lokalnego)** a_i z obiektu x jest to podstawienie

$$x = \pi(x, a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n).$$

3. Operator **dodania atrybutu** a_i **opisujących podobiektu** obiektu x jest to podstawienie

$$A_{\text{Sub}}(x) = \{a_i\} \oplus A_{\text{Sub}}(x) \text{ oraz dla każdego podobiektu } y_i \text{ obiektu } x \text{ podstawienie}$$

$$A(y_i) = \{a_i\} \oplus A(y_i).$$

4. Operator **usunięcia atrybutu** a_i **opisujących podobiektu** obiektu x jest to podstawienie

$$A_{\text{Sub}}(x) = A_{\text{Sub}}(x) - \{a_i\} \text{ oraz dla każdego podobiektu } y_i \text{ obiektu } x \text{ podstawienie}$$

$$A(y_i) = A(y_i) - \{a_i\}.$$

5. Operator **dodania metody** m_i do obiektu x jest to podstawienie

$$M(x) = \{m_i\} \cup M(x).$$

6. Operator **usunięcia metody** m_i z obiektu x jest to podstawienie

$$M(x) = M(x) - \{m_i\}.$$

7. Operator **dodania obiektu** x **do zbioru podobiektów** $\text{Sub}(y)$ jest to podstawienie

$$\text{Sub}(y) = \{\text{id}(x)\} \cup \text{Sub}(y) \text{ oraz podstawienie}$$

$$\text{Sup}(x) = \{\text{id}(y)\} \cup \text{Sup}(x).$$

8. Operator **usunięcia obiektu** x **ze zbioru podobiektów** $\text{Sub}(y)$ jest to podstawienie

$$\text{Sub}(y) = \text{Sub}(y) - \{\text{id}(x)\} \text{ oraz podstawienie}$$

$$\text{Sup}(x) = \text{Sup}(x) - \{\text{id}(y)\}.$$

9. Operator **dodania obiektu** x **do zbioru nadobiektów** $\text{Sup}(y)$ jest to podstawienie

$$\text{Sup}(y) = \{\text{id}(x)\} \cup \text{Sup}(y) \text{ oraz podstawienie}$$

$$\text{Sub}(x) = \{\text{id}(y)\} \cup \text{Sub}(x).$$

10. Operator **usunięcia obiektu** x **ze zbioru nadobiektów** $\text{Sup}(y)$ jest to podstawienie

$$\text{Sup}(y) = \text{Sup}(y) - \{\text{id}(x)\} \text{ oraz podstawienie}$$

$$\text{Sub}(x) = \text{Sub}(x) - \{\text{id}(y)\}.$$

11. Operator **tworzenia obiektu**² x jest to sekwencja operatorów

- a) dodania atrybutów lokalnych $x = \gamma (\gamma (\gamma (\gamma (\text{object}, a_1), a_2), \dots) a_n),$
gdzie $a_1, a_2, \dots, a_n$ są atrybutami obiektu x , a *object* jest obiektem systemowym będącym nadobiektem wszystkich obiektów o pustym zbiorze identyfikatorów nadobiektów,
- b) dodania obiektów do zbioru podobiektów obiektu x ,
- c) dodania obiektów do zbioru nadobiektów obiektu x ,
- d) dodania atrybutów opisujących podobiektu obiektu x ,
- e) nadania wartości atrybutom obiektu x ,
- f) dodania metod do obiektu x .

12. Operator **usunięcia obiektu** x jest to sekwencja operatorów

- a) usunięcia obiektu x ze zbioru podobiektów dla każdego nadobiektu obiektu x ,
- b) usunięcia obiektu x ze zbioru nadobiektów dla każdego podobiektu obiektu x ,
- c) usunięcia identyfikatora obiektu x z wartości innych obiektów,
- d) podstawienia $v(x) = \text{Sup}(x) = \text{Sub}(x) = \emptyset$.



Przykład

Zmodyfikujemy bazę Kuratorium dodając obiekt *Koło_fiz*, które zostało utworzone dla uczniów klasy 2a i klasy 2c, a usunięto *Koło_mat*. Zmiany możemy przeprowadzić w następujący sposób:

1. Tworzenie obiektu *Koło_fiz*.

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
Koło_fiz	Uczeń_1, Uczeń_3	Szkoła_11, Szkoła_12, Jedn_szkolna	Imię, Nazwisko	<Nazwa_jedn: Koło_fizyczne, Liczba_uczniów:2>	∅

2. Usunięcie obiektu *Koło_mat*.

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
Koło_mat	Uczeń_2	Szkoła_11, Szkoła_12, Jedn_szkolna	Imię, Nazwisko	<Nazwa_jedn: Koło_matematyczne, Liczba_uczniów:1>	∅

3. W wyniku utworzenia obiektu *Koło_fiz* zostały zmodyfikowane obiekty *Uczeń_1*, *Uczeń_3*, *Szkoła_11*, *Szkoła_12* i *Jedn_szkolna*. W wyniku usunięcia obiektu *Koło_mat* zostały zmodyfikowane obiekty *Uczeń_2*, *Szkoła_11*, *Szkoła_12* i *Jedn_szkolna*. Ostateczna postać tych obiektów jest następująca.

² W pracy zakładamy, że do pobrania i ustawienia wartości danego atrybutu używamy metod atrybutowych typu get i set [Abi95].

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
Szkoła_11	Klasa_2a, Klasa_2b, Koło_fiz	Dolny_Śląsk	Nazwa_jedn, Liczba_uczniów	<Nr_szkoły:11, Dyrektor:Nauczyciel_1, Liczba_uczniów:2>	Średnia_z arobków
Szkoła_12	Klasa_2c, Koło_fiz	Dolny_Śląsk	Nazwa_jedn, Liczba_uczniów	<Nr_szkoły:12, Dyrektor:Nauczyciel_1, Liczba_uczniów:1>	Średnia_z arobków
Jedn_szkolna	Klasa_2a, Klasa_2b, Klasa_2c, Koło_fiz	∅	Liczba_uczniów	<∅>	∅
Uczeń_1	∅	Klasa_2a, Uczeń, Osoba, Koło_fiz	∅	<Imię:Maciej, Nazwisko:Kruk, Rok_ur:1990, Płeć:M, Ocena:5.0, 4.0, 3.5>	Średnia_o cen
Uczeń_2	∅	Klasa_2b, Uczeń, Osoba	∅	<Imię:Jan, Nazwisko:Kula, Rok_ur:1990, Płeć:M, Ocena:4.5>	Średnia_o cen
Uczeń_3	∅	Klasa_2c, Uczeń, Osoba, Koło_fiz	∅	<Imię:Beata, Nazwisko:Nowak, Ocena:3.0, 2.0, Płeć:K>	Średnia_o cen



3.3. Własności migracji obiektów

Operatory zmian w obiektach są dwojakiego rodzaju. Operatory należące do pierwszej grupy służą do przeprowadzania zmian wartości atrybutów oraz do przeprowadzania zmian w zbiorze metod obiektu. Druga grupa operatorów zmian pozwala na zmianę położenia obiektu w schemacie obiektów, czyli na **migrację obiektu**. Przez migrację obiektu rozumiemy zmianę jego przynależności do klasy [Li94, Li98, Men94, Rad94, Su91, Wie95]. Migracja jest ważnym elementem zbioru możliwych zmian w obiektach. W przypadku obiektów wieloklasowych na ogół migracja sprowadza się do aktualizacji zbioru identyfikatorów nadobiektów zmienianego obiektu i ewentualnie do zmiany atrybutów obiektu wynikającej z wymagań narzuconych przez związek obiekt-podobiekt. Wtedy, jeżeli obiekt uzyskuje nowe własności, oznacza to pojawienie się obiektu w nowej klasie bez potrzeby usuwania go z innych klas. Natomiast w przypadku, gdy obiekt traci własności, najczęściej jest on usuwany z danej klasy. Migracja może prowadzić do przejścia obiektu do nowych klas, pozostania w dotychczasowych klasach lub ich opuszczenia.

Aby poznać zasady migracji obiektów przeanalizujemy trzy charakterystyczne przykłady.

Przykład. Figury geometryczne

Każdy wielokąt składa się z wierzchołków i krawędzi. Klasa wielokątów posiada atrybut wierzchołek reprezentowany przez parę liczb rzeczywistych. Ponadto klasa ta posiada metody: dodanie wierzchołka i obliczenie pola powierzchni. Klasa prostokątów jest podklasą wielokątów. Warunkiem przynależności wielokąta do tej klasy jest posiadanie czterech wierzchołków, których poziome i pionowe współrzędne są parami równe. Klasa ta posiada

atrybuty automatyczne [Kro00b]: długość i szerokość (wyliczane na podstawie współrzędnych wierzchołków) oraz metodę skalowania prostokąta. Ponadto posiada własną implementację metody obliczenia pola powierzchni. Klasa kwadratów jest podklasą prostokątów. Jedynym warunkiem przynależności jest to, aby długość i szerokość prostokąta były sobie równe. Kiedy definiujemy wielokąt, może on trafić do klasy wielokątów, prostokątów lub kwadratów w zależności od jego własności geometrycznych. Na przykład, jeśli mamy prostokąt i dodamy do niego wierzchołek automatycznie przestaje być prostokątem, i staje się wielokątem. Traci też atrybuty długość i szerokość. I odwrotnie dodanie wierzchołka do trójkąta może spowodować jego przejście do klasy prostokątów lub kwadratów. Podobnie proces migracji obiektów może przebiegać po wykonaniu metody skalowania dla prostokątów i kwadratów.



Przykład. Stan cywilny osób

Rozważmy teraz hierarchię osób ze względu na ich stan cywilny. Klasa osób posiada cztery rozłączne podklasy: stan paniński, stan kawalerski, stan małżeński i stan wdowi. Osoby zmieniają swój stan cywilny. Przechodzą ze stanu kawalerskiego czy panińskiego do stanu małżeńskiego. Nie wolno natomiast zmienić stanu wdowiego na stan kawalerski czy na stan paniński. Widzimy tutaj ważną cechę migracji, jaką jest kierunek możliwych zmian. Ograniczenie tego typu zostanie zdefiniowane w następnym rozdziale.



Przykład. Ochrona środowiska

Tworząc bazę danych o przedsiębiorstwie i ich poprodukcyjnych odpadach umożliwiamy monitorowanie zmian w środowisku na pewnym terytorium. W szczególności zbieramy informacje o lokalizacji przedsiębiorstwa, ilości i jakości materiałów odpadowych (nietoksyczne, toksyczne, wysoce niebezpieczne). Dla każdego obszaru zapamiętujemy liczbę mieszkańców. Następnie klasyfikujemy obszar jako bezpieczny, zagrożony czy skażony w zależności od odległości od przedsiębiorstwa produkującego toksyczne odpady oraz od ilości i jakości produkowanych materiałów. W tym przypadku kilka zdarzeń może być przyczyną zmian struktur i zachowania istniejących obiektów. Już utworzenie nowego przedsiębiorstwa czy zmiana rodzaju i ilości produkowanych odpadów może spowodować migrację z obszaru bezpiecznego do grupy zagrożonych lub skażonych.



3.3.1. Schemat migracji obiektów

Klasyczne podejście do migracji obiektów ogranicza się do operacji **przeniesienia** obiektu z jednej klasy źródłowej do innej klasy. Ogólny schemat takiego przeniesienia jest następujący:

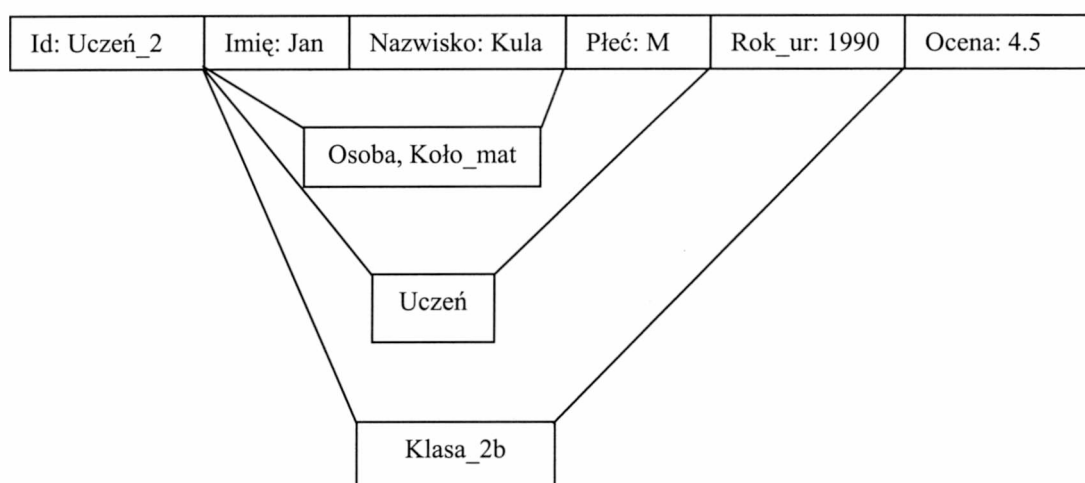
1. Przeniesienie w dół hierarchii dziedziczenia obiektów polegające na dodaniu atrybutów i metod klasy specjalizującej oraz modyfikacji powiązań z klasą źródłową i docelową. Taką operację nazywamy **specjalizacją** obiektu.
2. Przeniesienie w górę hierarchii dziedziczenia obiektów polegające na usunięciu atrybutów i metod klasy specjalizującej oraz modyfikacji powiązań z klasą źródłową i docelową. Taką operację nazywamy **generalizacją** obiektu.

3. W pozostałych przypadkach, czyli niezgodnie z hierarchią dziedziczenia przeniesienie polega na dodaniu, następnie usunięciu atrybutów i metod oraz modyfikacji powiązań z klasami.

Przesunięcie obiektu w górę hierarchii obiektów może być postrzegana jako usunięcie obiektu. W systemach obiektowych są znane dwa podejścia do usuwania obiektów [Fig96]. W pierwszym użytkownik decyduje o usuwaniu obiektów. W drugim ograniczono możliwości użytkownika jedynie do usuwania powiązań między obiektami. Obiekt jest automatycznie usuwany przez system o ile nie posiada żadnego powiązania.

Podobna sytuacja pojawia się przy przesunięciu obiektu w dół hierarchii obiektów. W tym przypadku należy poza przeprowadzeniem modyfikacji atrybutów i metod zmienianego obiektu dokonać modyfikacji obiektów powiązanych. Dużym ułatwieniem jest oparcie zależności między obiektami na niezmiennych identyfikatorach. W ten sposób nie trzeba dokonywać żadnych zmian w otoczeniu samego obiektu.

Lokalizację obiektu w schemacie obiektów możemy określić przez zbadanie jego związków obiekt-podobiekt, w tym związków dziedziczenia (rys. 3.1).



Rysunek 3.1. Schemat obiektu Uczeń_2 i powiązanych z nim obiektów-klas

Z punktu widzenia zarządzania danymi przy migracji obiektów należy przyjąć dwa podstawowe założenia:

1. Założenie o niezmienności identyfikatora obiektu.

Pierwsze założenie mówi, że migracji obiektu nie można uprościć do usunięcia obiektu w jednej klasie by następnie utworzyć go w innej. Takie postępowanie prowadzi do utworzenia innego obiektu. System bazy danych automatycznie dba o nadanie niepowtarzalnego identyfikatora. Jedynie zachowanie tego samego identyfikatora umożliwia jednoznaczną interpretację, że to właśnie istniejący obiekt zmienił miejsce swojego pobytu.

2. Założenie zachowania informacji.

Drugie założenie zapewnia przeniesienie odpowiadających wartości atrybutów z obiektu źródłowego do obiektu docelowego. Taką odpowiedniość gwarantuje pojęcie pochodzenia atrybutu.

Ograniczenia nakładane na migrację obiektów wynikają z definicji obiektu oraz z dodatkowych ograniczeń zdefiniowanych dla tego obiektu. Problemem dodatkowych ograniczeń będziemy zajmować się w rozdziale poświęconym integralności obiektów wieloklasowych.

3.3.2. Podstawowe operacje migracji obiektów

W modelu obiektów wieloklasowych klasyczny operator przenieś zgodny ze schematem migracji z poprzedniego rozdziału jest niewystarczający. Konieczne jest wprowadzenie dwóch dodatkowych operatorów dodaj oraz usuń. Poniżej przedstawiono definicje podstawowych operatorów migracji obiektów.

Definicja

Operatorem **przeniesienia** nazywamy operator $\frac{id(x)}{\text{przenieś}}$ taki, że

jeżeli $id(x_1), id(x_2), \dots, id(x_n) \xrightarrow{\text{przenieś}} id(y_1), id(y_2), \dots, id(y_m)$,

gdzie x jest migrującym obiektem, $id(x_1), id(x_2), \dots, id(x_n) \in \text{Sup}(x)$ i $y_1, y_2, \dots, y_m \in \text{OBJ}$, to

- usuń obiekty $x_1, x_2, \dots, x_n$ ze zbioru nadobiektów obiektu x ,
- usuń atrybuty opisujące podobiekty obiektów $x_1, x_2, \dots, x_n$,
- dodaj obiekty $y_1, y_2, \dots, y_m$ do zbioru nadobiektów obiektu x ,
- dodaj atrybuty opisujące podobiekty obiektów $y_1, y_2, \dots, y_m$,
- pobierz wartości dla każdego atrybutu obiektu x .



W przypadku, gdy obiekt x należy już do zbioru podobiektów obiektu y_i operacja przenieś jest przerywana dla danego obiektu i kontynuowana dla obiektu y_{i+1} .

Przykład

Szkoła_11 $\xrightarrow[\text{przenieś}]{\text{Klasa_2a}}$ Szkoła_12.



Operator przenieś uzupełnia zbiór identyfikatorów nadobiektów obiektu x o obiekty $y_1, y_2, \dots, y_m$, usuwa ze zbioru identyfikatorów nadobiektów obiekty $x_1, x_2, \dots, x_n$, pyta o wartości dla nowych atrybutów pochodzących z obiektów $y_1, y_2, y_3, \dots, y_m$ oraz usuwa atrybuty przynależne do obiektów $x_1, x_2, x_3, \dots, x_n$.

W przypadku, gdy obiekt x nie należy do zbioru podobiektów obiektu x_i operacja może być przerywana lub może być kontynuowana dla następnego obiektu x_{i+1} . Wybraliśmy

kontynuację, gdyż w naszej ocenie nie jest to błąd krytyczny a tylko brak związku obiekt-podobiekt. Aby uniknąć takiej sytuacji można np. wprowadzić pomocniczy operator boolowski sprawdź.

Definicja

Operatorem **sprawdzenia** nazywamy operator $\frac{id(x)}{\text{sprawdź}}$ taki, że

jeżeli $id(x_1), id(x_2), \dots, id(x_n) \frac{id(x)}{\text{sprawdź}} id(y_1), id(y_2), \dots, id(y_m)$,

gdzie x jest sprawdzanym obiektem, $id(x_1), id(x_2), \dots, id(x_n) \in \text{Sub}(x)$ i $id(y_1), id(y_2), \dots, id(y_m) \in \text{Sup}(x)$, to zwróć wartość logiczną true. W przeciwnym przypadku operator zwraca wartość logiczną false.

♦

Przykład

$\text{Uczeń_1}, \text{Nauczyciel_1} \frac{\text{Klasa_2a}}{\text{sprawdź}} \text{Szkoła_11}, \text{Jedn_szkolna}$.

Po wykonaniu operatora w wyniku otrzymamy wartość true, ponieważ Uczeń_1 i Nauczyciel_1 są podobiektami obiektu Klasa_2a, natomiast Szkoła_11 i Jedn_szkolna są nadobiektami sprawdzanego obiektu.

♦

Definicja

Operator przeniesienia obiektu x nazywamy **generalizacją** wtedy i tylko wtedy,

gdy $id(y) \frac{id(x)}{\text{przenieś}} id(z)$ i $z \leq y$,

gdzie $x, y, z \in \text{OBJ}$.

♦

Przykład

$\text{Nauczyciel} \frac{\text{Nauczyciel_1}}{\text{przenieś}} \text{Osoba}$.

♦

Definicja

Operator przeniesienia obiektu x nazywamy **specjalizacją** wtedy i tylko wtedy,

gdy $id(y) \frac{id(x)}{\text{przenieś}} id(z)$ i $y \leq z$,

gdzie $x, y, z \in \text{OBJ}$.

♦

Przykład

Nauczyciel $\frac{\text{Nauczyciel_1}}{\text{przenieś}}$ Dyrekcja.

◆

Operacja specjalizacji jest operacją odwrotną do generalizacji. Zachowują one ten sam identyfikator, jednakże nie są to operacje zachowujące kompletność informacyjną obiektu, gdyż po utracie atrybutu tracimy także jego wartość.

Definicja

Operatorem **dodania** nazywamy operator $\frac{\text{id}(x)}{\text{dodaj}}$ taki, że

jeżeli $\frac{\text{id}(x)}{\text{dodaj}}$ $\text{id}(y_1), \text{id}(y_2), \dots, \text{id}(y_m)$,

gdzie x jest migrującym obiektem i $y_1, y_2, \dots, y_m \in \text{OBJ}$, to

- dodaj obiekty $y_1, y_2, \dots, y_m$ do zbioru nadobiektów obiektu x ,
- dodaj atrybuty opisujące podobiekty obiektów $y_1, y_2, \dots, y_m$,
- pobierz wartości dla każdego atrybutu obiektu x .

◆

W przypadku, gdy obiekt x należy już do zbioru podobiektów obiektu y_i operacja dodaj jest przerywana dla danego obiektu i kontynuowana dla obiektu y_{i+1} .

Operator dodaj uzupełnia zbiór identyfikatorów nadobiektów obiektu x o obiekty $y_1, y_2, y_3, \dots, y_m$ i pyta o wartości dla nowych atrybutów.

Przykład

$\frac{\text{Uczeń_1}}{\text{dodaj}}$ Koło_mat.

◆

Przed użyciem operatora dodaj możemy sprawdzić czy obiekt x nie jest już podobiektom obiektów $y_1, y_2, y_3, \dots, y_m$.

Przykład

$\frac{\text{Klasa_2a}}{\text{sprawdź}}$ Szkoła_12, Jedn_szkolna.

Po wykonaniu operatora sprawdź otrzymamy wartość false, ponieważ Szkoła_12 nie jest nadobiektom sprawdzanego obiektu.

◆

Definicja

Operatorem **usunięcia** nazywamy operator $\frac{\text{id}(x)}{\text{usuń}}$ taki, że

jeżeli $id(x_1), id(x_2), \dots, id(x_n) \xrightarrow{\text{usuń}} id(x)$

gdzie x jest migrującym obiektem i $id(x_1), id(x_2), \dots, id(x_n) \in \text{Sup}(x)$, to

- a) usuń obiekty $x_1, x_2, \dots, x_n$ ze zbioru nadobiektów obiektu x ,
- b) usuń atrybuty opisujące podobiekty obiektów $x_1, x_2, \dots, x_n$.

◆

Operator usuń usuwa ze zbioru identyfikatorów nadobiektów obiektu x obiekty $x_1, x_2, x_3, \dots, x_n$ oraz usuwa atrybuty przynależne do obiektu x_i , z którego obiekt został usunięty.

W przypadku, gdy obiekt x nie należy do zbioru identyfikatorów podobiektów x_i operacja jest przerywana i kontynuowana dla klasy x_{i+1} .

Przykład

$\text{Koło_mat}, \text{Klasa_2b} \xrightarrow{\text{usuń}} \text{Uczeń_2}$.

◆

Podobnie jak w dwóch poprzednich przypadkach przed użyciem operatora usuń możemy sprawdzić czy obiekt x jest podobiektami obiektów $x_1, x_2, x_3, \dots, x_n$.

Przykład

$\text{Uczeń_1}, \text{Nauczyciel_1} \xrightarrow{\text{sprawdź}} \text{Klasa_2a}$.

Po wykonaniu operatora w wyniku otrzymamy wartość true, ponieważ Uczeń_1 i Nauczyciel_1 są podobiektami obiektu Klasa_2a.

◆

Zdefiniowane powyżej operatory mogłyby zostać rozszerzone o możliwość stosowania ich dla grupy obiektów. Nie jest to jednak konieczne. W sytuacji zmian dotyczących wszystkich podobiektów wystarczy dokonać zmiany w ich nadobiektach.

3.4. Algorytmy operatorów migracji

3.4.1. Algorytm realizacji operatora sprawdź

Algorytm operatora sprawdź składa się z n sprawdzeń czy x_i jest podobiektami obiektu x i m sprawdzeń czy y_i jest nadobiektami obiektu x . Ponadto sprawdzamy poprawność zbioru atrybutów obiektu x wynikającą z wymagań definicyjnych.

Algorytm operatora sprawdź**Parametry wejściowe:** $x, x_1, x_2, x_3 \dots x_n, y_1, y_2, y_3 \dots y_m$.**Stan wyjściowy:** true lub false.

```

for i=1 to n do
    begin
        if  $x \notin \text{Sup}(x_i)$  then
            return false;
        if  $A_{\text{Sub}}(x) \neq \emptyset$  then
            if  $A_{\text{Sub}}(x_i) \supseteq A_{\text{Sub}}(x)$  then
                continue;
            else
                return false;
        if  $A(x_i) \not\subset A_{\text{Sub}}(x)$  then
            return false;
    end
for i=1 to m do
    begin
        if  $x \notin \text{Sub}(y_i)$  then
            return false;
        if  $A_{\text{Sub}}(x) \neq \emptyset$  then
            if  $A_{\text{Sub}}(x) \supseteq A_{\text{Sub}}(y_i)$  then
                continue;
            else
                return false;
        if  $A(x) \not\subset A_{\text{Sub}}(y_i)$  then
            return false;
    end

```

3.4.2. Algorytm realizacji operatora dodaj

Algorytm operatora dodaj składa się z m elementów typu dodanie obiektu y_i do zbioru $\text{Sup}(x)$ oraz typu dodanie obiektu x do zbioru $\text{Sub}(y_i)$. Następnie musimy uzupełnić atrybuty zgodnie z warunkiem definicji operatora.

Algorytm operatora dodaj**Parametry wejściowe:** $x, y_1, y_2, y_3 \dots y_m$.**Stan wyjściowy:** x .

```

for i=1 to m do
    begin
        if  $x \in \text{Sub}(y_i)$  then continue;
        Dodanie obiektu  $y_i$  do zbioru nadobiektów  $\text{Sup}(x)$ ;
         $k = \text{count}(\text{A}_{\text{Sub}}(y_i))$ ;
        for j=1 to k do
            if  $a_j \in \text{A}_{\text{Sub}}(y_i)$  and  $a_j \notin A(x)$  then
                Dodanie atrybutu  $a_j$  do obiektu  $x$ ;
        end
    end

```

W algorytmie występuje funkcja *count()* zliczająca elementy zbioru atrybutów. Ponadto w żadnym algorytmie nie zamieściliśmy wywołania funkcji pobrania wartości dla nowych atrybutów, gdyż to zadanie możemy pozostawić instrukcjom języka manipulacji danymi.

3.4.3. Algorytm realizacji operatora usuń

Algorytm operatora usuń składa się z n elementów typu usunięcie obiektu x_i ze zbioru $\text{Sup}(x)$ oraz typu usunięcie obiektu x ze zbioru $\text{Sub}(x_i)$. Następnie musimy usunąć atrybuty zgodnie z warunkiem definicji operatora.

Algorytm operatora usuń**Parametry wejściowe:** $x, x_1, x_2, x_3 \dots x_n$.**Stan wyjściowy:** x .

```

for i=1 to n do
    begin
        if  $x \notin \text{Sub}(x_i)$  then
            continue;
        Usunięcie obiektu  $x_i$  ze zbioru nadobiektów  $\text{Sup}(x)$ ;
         $k = \text{count}(\text{A}_{\text{Sub}}(x_i))$ ;
        for j=1 to k do
            if  $a_j \in \text{A}_{\text{Sub}}(x_i)$  and  $a_j \in A(x)$  then
                Usunięcie atrybutu  $a_j$  z obiektu  $x$ ;
        end
    end

```

3.4.4. Algorytm realizacji operatora przenieś

Algorytm operatora przenieś składa się z sekwencji algorytmów usuń i dodaj.

Algorytm operatora przenieś

Parametry wejściowe: $X, X_1, X_2, X_3 \dots X_n, Y_1, Y_2, Y_3 \dots Y_m$.

Stan wyjściowy: x .

1. Algorytm operatora usuń.
2. Algorytm operatora dodaj.

3.5. Własności operatorów migracji

Jeśli przyjmiemy powyższe definicje operatorów migracji obiektów to możemy stwierdzić, że

sekwencja operatorów (przenieś, dodaj) oraz sekwencja (dodaj, usuń) są równoważne.

Własność ta wynika z tego, że operator przenieś został zdefiniowany jako sekwencja operatorów usuń i dodaj, czyli

$$(id(x_1), id(x_2), \dots, id(x_n) \xrightarrow[\text{przenieś}]{id(x)} id(y_1), id(y_2), \dots, id(y_m)) \equiv$$

$$(id(x_1), id(x_2), \dots, id(x_n) \xrightarrow[\text{usuń}]{id(x)}, \xrightarrow[\text{dodaj}]{id(x)} id(y_1), id(y_2), \dots, id(y_m))$$

a więc

- (1) z łączności tego samego operatora

$$(X \xrightarrow[\text{operator}]{id(x)} Y, W \xrightarrow[\text{operator}]{id(x)} Z) \equiv (X \cup W) \xrightarrow[\text{operator}]{id(x)} (Y \cup Z)$$

- (2) z przemienności operatorów

$$(X \xrightarrow[\text{operator 1}]{id(x)} Y, W \xrightarrow[\text{operator 2}]{id(x)} Z) \equiv (W \xrightarrow[\text{operator 2}]{id(x)} Z, X \xrightarrow[\text{operator 1}]{id(x)} Y)$$

wynika, że

(przenieś, dodaj) $\equiv$

$$(id(x_1), id(x_2), \dots, id(x_n) \xrightarrow[\text{przenieś}]{id(x)} id(y_1), id(y_2), \dots, id(y_m), \xrightarrow[\text{dodaj}]{id(x)} id(z_1), id(z_2), \dots, id(z_k)) \equiv$$

$$\equiv (id(x_1), id(x_2), \dots, id(x_n) \xrightarrow[\text{usuń}]{id(x)}, \xrightarrow[\text{dodaj}]{id(x)} id(y_1), id(y_2), \dots, id(y_m)),$$

$$\xrightarrow[\text{dodaj}]{id(x)} id(z_1), id(z_2), \dots, id(z_k)) \equiv$$

$$\equiv (id(x_1), id(x_2), \dots, id(x_n) \xrightarrow[\text{usuń}]{id(x)}, \xrightarrow[\text{dodaj}]{id(x)} id(y_1), id(y_2), \dots, id(y_m), id(z_1), id(z_2), \dots, id(z_k)) \equiv$$

$$\equiv (\xrightarrow[\text{dodaj}]{id(x)} id(y_1), id(y_2), \dots, id(y_m), id(z_1), id(z_2), \dots, id(z_k), id(x_1), id(x_2), \dots, id(x_n) \xrightarrow[\text{usuń}]{id(x)}) \equiv$$

$\equiv (\text{dodaj}, \text{usuń}).$

Przykład

Sekwencja operatorów

$(\text{Klasa_2a} \xrightarrow[\text{przenieś}]{\text{Uczeń_1}} \text{Klasa_2b}, \xrightarrow[\text{dodaj}]{\text{Uczeń_1}} \text{Koło_mat})$

jest równoważna sekwencji operatorów

$(\xrightarrow[\text{dodaj}]{\text{Uczeń_1}} \text{Koło_mat}, \text{Klasa_2b}, \text{Klasa_2a} \xrightarrow[\text{usuń}]{\text{Uczeń_1}}).$

♦

3.6. Rozszerzone operatory migracji

Jedną z najistotniejszych cech obiektów wieloklasowych jest pochodzenie atrybutów. Podane operatory migracji nie uwzględniają zachowania wartości atrybutów przy przenoszeniu obiektów lub rozróżnienia atrybutów przy dodawaniu obiektów. Wykorzystując pojęcie pochodzenia atrybutów zminimalizujemy usuwanie i pobieranie wartości jedynie do atrybutów tego wymagających.

Zmodyfikowane algorytmy nazwiemy algorytmami operatorów **przenieś+** i **dodaj+**. Algorytm operatora usuń nie wymaga modyfikacji.

3.6.1. Algorytm realizacji operatora przenieś+

Najpierw wprowadzimy pomocnicze pojęcie przecięcia atrybutów.

Przecięciem atrybutów obiektów x i y , które oznaczamy $A(x) \otimes A(y)$, nazywamy następujący zbiór:

$$\{a: a \in A(x) \wedge a \in A(y) \wedge (a, x) = (a, y)\}.$$

Algorytm **przenieś+** zawiera wyrażenie $(\oplus A_{\text{Sub}}(x_i)) \otimes (\oplus A_{\text{Sub}}(y_i))$, które oznacza zbiór wspólnych atrybutów. Te atrybuty nie będą usuwane, gdyż następnie konieczne by było ich dodanie. Nie są usuwane także atrybuty lokalne.

Algorytm operatora przenieś+**Parametry wejściowe:** $x, x_1, x_2, x_3 \dots x_n, y_1, y_2, y_3 \dots y_m$.**Stan wyjściowy:** x .

```

for i=1 to n do
    begin
        if  $x \notin \text{Sub}(x_i)$  then
            continue;
        Usunięcie obiektu  $x_i$  ze zbioru nadobiektów  $\text{Sup}(x)$ ;
         $k = \text{count}(\text{A}_{\text{Sub}}(x_i))$ ;
        for j=1 to k do
            // usunięcie atrybutu  $a_j$  z obiektu  $x$ , który nie będzie dodany
            if  $a_j \in \text{A}_{\text{Sub}}(x_i) \wedge a_j \notin (\oplus \text{A}_{\text{Sub}}(x_i)) \otimes (\oplus \text{A}_{\text{Sub}}(y_i))$  then
                Usunięcie atrybutu  $a_j$  z obiektu  $x$ ;
        end
    end
for i=1 to m do
    begin
        if  $x \in \text{Sub}(y_i)$  then
            continue;
        Dodanie obiektu  $y_i$  do zbioru nadobiektów  $\text{Sup}(x)$ ;
        // dodanie wymaganych atrybutów dla podobiektów  $y_i$ 
         $\text{A}(x) = \text{A}(x) \oplus \text{A}_{\text{sub}}(y_i)$ ;
    end

```

3.6.2. Algorytm realizacji operatora dodaj+

Algorytm operatora dodaj został zmodyfikowany ze względu na operację sumowania atrybutów obiektów wieloklasowych.

Algorytm operatora dodaj+**Parametry wejściowe:** $x, y_1, y_2, y_3 \dots y_m$.**Stan wyjściowy:** x .for $i=1$ to m do

begin

if $x \in \text{Sub}(y_i)$ then continue;Dodanie obiektu y_i do zbioru nadobiektów $\text{Sup}(x)$;// dodanie wymaganych atrybutów dla podobiektów y_i $A(x) = A(x) \oplus A_{\text{sub}}(y_i)$;

end

Rozszerzone operatory migracji nie posiadają własności przemienności, stąd wynika, że sekwencja operatorów (przenieś+, dodaj+) nie jest równoważna sekwencji (dodaj+, usuń).

3.7. Wnioski

Klasyczny zbiór zmian dotyczących obiektów zawiera: dodanie/usunięcie obiektu/atributu/metody, zmianę nazwy atrybutu/metody oraz zmianę typu atrybutu. Natomiast zbiór zmian dotyczących klas bazy danych składa się z: dodania/usunięcia klasy/nadklasy oraz zmiany nazwy klasy. Dla obiektów wieloklasowych podstawowy zbiór zmian jest prostszy i obejmuje: dodanie/usunięcie obiektu/atributu/metody oraz dodanie/usunięcie obiektu do/ze zbioru nadobiektów/podobiektów.

Jak wynika z powyższych definicji podstawowy zbiór zmian w obiektach wieloklasowych jest prosty i przejrzysty. Na tej podstawie można stwierdzić, że wprowadzenie wieloklasowości oraz ujednolicenie pojęcia obiektu i klasy pozwoliło na uproszczenie operacji zmian w bazie danych.

Migracja może być interpretowana wraz z operacją dodania/usuwania obiektów jako podstawowy mechanizm zmiany schematu bazy danych. Np. łączenie klas może być zastąpione przez przeniesienie obiektów z jednej klasy do drugiej i usunięcie jednej z nich.

Zestaw operatorów migracji dla obiektów wieloklasowych składa się z trzech operatorów: przenieś, dodaj i usuń. Operatorem pomocniczym dla operatorów podstawowych jest operator sprawdź. Po uwzględnieniu pochodzenia atrybutów skonstruowaliśmy rozszerzone operatory przenieś+ i dodaj+, których ogólna idea jest analogiczna jak dla operatorów podstawowych, z tą jednak różnicą, że o ile to możliwe przy tym samym pochodzeniu atrybutów zachowujemy ich wartość i rozróżniamy atrybuty o tych samych nazwach i różnym pochodzeniu przy operacji dodawania.

Podejście prezentowane w pracy w pełni zezwala na migrację danych i to zarówno rozumianą w klasyczny sposób jak i w rozszerzony sposób dla obiektów wieloklasowych.

4. Integralność obiektów wieloklasowych

Problem zachowania integralności obiektów, czyli poprawności i spójności w obiektowej bazie danych ma takie samo duże znaczenie jak w bazach tradycyjnych [Bas94, Sla93]. W pracy mówiąc o poprawności danych mamy na myśli obiekty spełniające warunki wynikające z definicji podanych w rozdziale drugim. Ponadto bierzemy pod uwagę poprawność wartości atrybutów (np. wymaganie by wartości niektórych atrybutów były zbiorem niepustym) oraz wymaganie właściwych relacji między atrybutami obiektu (np. wymóg, aby data rozpoczęcia konferencji była wcześniejsza niż data jej zakończenia). Przez spójność danych rozumiemy zachowywanie odpowiednich zależności między obiektami, np. zapewnienie by po usunięciu obiektu nie pozostały odwołania do tego obiektu z innych obiektów.

Obiektowe systemy baz danych oferują podstawowe mechanizmy kontroli spójności i poprawności bazy danych, np. nie można jawnie usuwać obiektów, do których istnieją odwołania z innych obiektów. Procedury typu *garbage collection* usuwają obiekty, do których nie ma żadnych odwołań [Fig96]. Stosowane są ponadto: statyczne nakładanie bezpośrednich ograniczeń na daną klasę (np. długość nie może być ujemna) i dynamiczne dotyczące zmian poprzez definiowanie specjalnych funkcji tzw. triggers (np. nie można zmienić daty urodzenia w przód poza aktualną datę). W modelu obiektowym dzięki występowaniu niepowtarzalnego identyfikatora każdego obiektu nie ma potrzeby konstruowania ograniczeń dotyczących klucza czy referencji na inny obiekt tak jak ma to miejsce w modelu relacyjnym [Dat00].

Najczęściej w systemach obiektowych wymaga się stosowania następujących zasad związanych z integralnością danych:

1. Zachowanie poprawności atrybutów, np. żądanie wartości dla atrybutu.
2. Zachowanie zależności pomiędzy atrybutami, np. wartość jednego atrybutu mniejsza od wartości drugiego.
3. Zachowanie spójności między obiektami, np. po usunięciu obiektu.
4. Zachowanie reguł przy przechodzeniu obiektu z klasy do klasy, np. zasada przemieszczania obiektu tylko w obrębie hierarchii dziedziczenia.
5. Zachowanie reguł przynależności obiektu do klasy, np. wzajemne wykluczanie klas.
6. Zachowanie reguł definiowania przy dziedziczeniu, np. blokada redefiniowania atrybutów.

Nie rozwiązują one jednak wszystkich pojawiających się problemów. Niezbędne staje się wyspecyfikowanie dodatkowych ograniczeń nakładanych na bazę danych w celu zapewnienia integralności danych w niej zawartych. W przypadku obiektów wieloklasowych oraz migracji danych ma to szczególne znaczenie.

W bazach danych mamy do czynienia z dwoma rodzajami ograniczeń: ograniczeniami zachowania i ograniczeniami strukturalnymi [Tet91]. **Ograniczenia zachowania** dotyczą zależności występujących między obiektami świata rzeczywistego. Są to w szczególności zależności między atrybutami. Opisują one semantykę świata rzeczywistego i dlatego będziemy je nazywali **ograniczeniami semantycznymi** lub krótko **ograniczeniami**. Określamy je w trakcie projektowania bazy danych. Natomiast ograniczenia strukturalne, które nazywać będziemy **ograniczeniami integralnościowymi**, są częścią składową modelu. Są więc związane z modelem danych i są niezależne od świata zewnętrznego. W pracy zostały one podane w rozdziale drugim.

W modelu obiektów wieloklasowych musimy uwzględnić ograniczenia nie występujące w innych modelach danych, bądź występujące w marginalny sposób. Należą do nich:

1. **Poprawność związku obiekt-podobiekt**, np. obiekt Student może być też obiektem Absolwent, natomiast obiekt Uczeń już nie.
2. **Poprawność migracji obiektów**, np. obiekt Student może w pewnym momencie stać się obiektem Absolwent, a następnie obiektem Wykładowca. Z kolei nie jest możliwa sytuacja, aby obiekt Absolwent został obiektem Student chyba, że jednocześnie będzie Absolwentem i Studentem. Zatem w ogólności obiekt Absolwent może się zmienić tylko w obiekt należący do określonego zbioru obiektów, a nie w dowolny obiekt.

Możliwe są trzy podejścia do rozwiązania problemu przechowywania ograniczeń w obiektowej bazie danych. W modelu obiektów wieloklasowych ograniczenia możemy umieścić w: obiektach-instancjach, obiektach-klasach lub obiektach-ograniczeniach.

W dalszej części pokażemy ograniczenia definiowane w **obektach-ograniczeniach**. Rozwiązanie to posiada dwie podstawowe zalety w stosunku do znanych rozwiązań:

1. Obiekty-ograniczenia są obiektami. Powoduje to, że są one bezpieczniejsze.
2. Obiekty-ograniczenia są proste w implementacji, także dla ograniczeń obejmujących obiekty wieloklasowe czy związek obiekt-podobiekt a więc i związek dziedziczenia.

Można teraz przystąpić do konstruowania wyrażeń odpowiedzialnych za ograniczanie obiektów.

Przykład

Ograniczenie wieku ucznia do zakresu (6 lat, 20 lat):

$\forall x \in \text{Sub}(\text{Uczeń})(x.\text{Wiek} > 6 \wedge x.\text{Wiek} < 20).$



Przedstawione tu wyrażenie ukazuje problem bezpośredniego odwoływania się do atrybutów obiektu (atrybut Wiek). Taki sposób odwoływania się do atrybutów łamie jedno z podstawowych założeń obiektowości tj. hermetyzację. Ograniczenia powinny odwoływać się do atrybutów poprzez odpowiednie metody. Problem ten możemy rozwiązać wprowadzając tzw. **metody atrybutowe**. Ponadto stosując metody łatwiej i bardziej intuicyjnie można ograniczać dziedzinę bazy danych. Wynika to z faktu, że nie wszystkie dane są umieszczane w obiektach w postaci atrybutów. Niektóre są po prostu wyliczane, np. wiek obliczamy na podstawie porównania daty urodzin i daty systemowej.

Zmodyfikowane ograniczenie, w którym używamy metod atrybutowych ma następującą postać:

$$\forall x \in \text{Sub}(\text{Uczeń})(x.\text{get}(\text{Wiek}) > 6 \wedge x.\text{get}(\text{Wiek}) < 20).$$

W pozostałych przykładach ze względu na większą czytelność zapisu będziemy opuszczać nazwę metody atrybutowej `get` i będziemy posługiwać się bezpośrednio atrybutem oraz jego wartością. Podobne uproszczenie zapisów będziemy stosować w przypadku strukturalnego języka manipulacji obiektami wieloklasowymi w rozdziale piątym.

Celem tego rozdziału będzie przedstawienie różnych ograniczeń modelujących dynamikę obiektowej bazy danych, wzbogacone o liczne przykłady oraz podstawy teoretyczne. W dalszej części omówione zostaną ograniczenia dotyczące obiektów wieloklasowych i problemy integralności migracji danych. W przykładach będziemy posługiwać się notacją języka logiki pierwszego rzędu.

Ograniczenia w modelu obiektów wieloklasowych można podzielić na:

1. **Ograniczenia dziedziny.**
2. **Ograniczenia związku.**

4.1. Ograniczenia dziedziny

Ograniczenia dziedziny związane są z pojęciem dziedziny bazy danych, ponieważ są rodzajem ograniczeń narzuconych na wartości atrybutów obiektów. Modelując dziedzinę bazy danych powinniśmy zawsze kierować się zasadą: „ograniczaj dziedzinę bazy danych (usuвай jak najwięcej niedopuszczalnych wartości), pamiętając o maksymalizacji liczby dostępnych elementów świata rzeczywistego”.

Możemy wyróżnić cztery rodzaje ograniczeń dziedziny:

1. Ograniczenia niezależne.

Nie zależą od stanu bazy. Nakładają ograniczenia na pojedyncze atrybuty.

2. Ograniczenia wewnętrzne.

Nałożone na atrybut w obiekcie x zależą od jednej lub większej liczby wartości atrybutów tego samego obiektu.

3. Ograniczenia zewnętrzne.

Nałożone na atrybut w obiekcie x zależą od jednej lub większej liczby wartości atrybutów z obiektów $y_1, y_2, \dots, y_n$.

4. Ograniczenia agregacji.

Nakładane na atrybut w obiekcie x zależą od jednej lub większej liczby agregacji. Klasyczne operatory agregacji to: **count**, **sum**, **avg**, **min**, **max** [Kro00b].

Poniżej znajdują się przykłady ograniczeń dziedziny.

Przykład. Ograniczenie niezależne

$$\forall x \in \text{Sub}(\text{Uczeń})(x.\text{Ocena} \in \{2.0, 3.0, 3.5, 4.0, 4.5, 5.0\}).$$

W tym przypadku ograniczenie dotyczy pojedynczego atrybutu – oceny, która musi mieć jedną z podanych wartości.



Przykład. Ograniczenie wewnętrzne

Rok urodzenia nauczyciela musi być mniejszy lub równy od roku zatrudnienia pomniejszonego o 18.

$$\forall x \in \text{Sub}(\text{Nauczyciel})(x.\text{Rok_urodzenia} \leq x.\text{Rok_zatrudnienia} - 18).$$

Ograniczenie to zależy tylko od atrybutów w pojedynczym obiekcie.



Przykład. Ograniczenie zewnętrzne

$$\forall x \in \text{Sub}(\text{Szkoła}) \exists y \in \text{Sub}(\text{Dyrekcja})(x.\text{Dyrektor} = \text{id}(y)).$$

W tym przypadku występuje ograniczenie związane ze związkiem referencji. Dla obiektu Szkoła identyfikator Dyrektora musi się znaleźć w zbiorze podobiektów obiektu Dyrekcja.



Przykład. Ograniczenie agregacji

$$\forall x \in \text{Sub}(\text{Nauczyciel}) \wedge x \in \text{Sub}(\text{Szkoła_11}) \\ (x.\text{Wynagrodzenie} \leq \text{Szkoła_12}.\text{Srednia_zarobków}).$$

Przykład pokazuje sposób ograniczania dziedziny atrybutów. Nauczyciel Szkoły_11 nie może zarabiać więcej niż średnia zarobków nauczycieli Szkoły_12.



4.2. Ograniczenia związku obiekt-podobiekt

Możliwe są następujące związki pomiędzy obiektami: 1:1, 1:N, N:1, N:M i tak:

1. Związek 1:1 odpowiada powiązania obiektu-klasy z jednym obiektem.
2. Związek 1:N odpowiada związkowi przynależności do obiektu-klasy N obiektów-instancji.
3. Związek N:1 odpowiada sytuacji, kiedy obiekt-instancja jest podobiektom N obiektów-klas.
4. Związek N:M odpowiada współwystępowaniu związków 1:N i N:1, czyli M obiektów-instancji przynależy do N obiektów-klas.

W modelu obiektów wieloklasowych związek obiekt-podobiekt to związek N:M i jak każdy związek może być ograniczany. Ograniczenie dotyczy wtedy powiązania nadobiektów i podobiektów, wzajemnego wykluczania obiektów w związku obiekt-podobiekt lub wzajemnego występowania obiektów w związku. Warunki wzajemnego wykluczania czy wzajemnego występowania mogą być specyfikowane dla nadobiektów jak i podobiektów.

Przykład

Założmy, że nauczyciele prowadzą jednostki szkolne (klasy i koła zainteresowań). Każdy nauczyciel musi prowadzić co najmniej jedną jednostkę szkolną, ale nie więcej niż dwie jednostki. Ograniczenie takie można zapisać w następujący sposób:

$$\forall x \in \text{Sub}(\text{Nauczyciel}) \wedge \exists y \in \text{Sub}(\text{Jedn_szkolna}) \wedge \exists z \in \text{Sub}(\text{Jedn_szkolna}) \wedge y \neq z \\ \Rightarrow (y \in \text{Sup}(x)) \vee (z \in \text{Sup}(x)).$$

◆

4.2.1. Wzajemne wykluczanie obiektów

W proponowanym podejściu obiekt może przynależeć do kilku obiektów-klas jednocześnie. Jednakże istnieją obiekty-klasy, które nie powinny posiadać wspólnych podobiektów. Np. pojedynczy obiekt nie może być uczniem oraz nauczycielem w tej samej szkole. Rozwiązaniem tego problemu może być ograniczenie polegające na przyporządkowaniu obiektów tylko do określonych nadobiektów. Najczęściej nadobiekty będą odpowiadały wspólnemu poddrzewu w schemacie obiektów, ale czasami może to być obiekt także z innego poddrzewa.

Należy więc wprowadzić możliwość definiowania **obiektów wykluczających**. W przykładowym schemacie obiektów z bazy Kuratorium, choć Klasa_2a i Klasa_2b są podobiektami Szkoły_11 to nie powinny one posiadać wspólnych uczniów. Pojedynczy uczeń nie może być podobiektom obu klas. Ograniczenie to można zdefiniować w następujący sposób:

Przykład

$$\forall x \in \text{Sub}(\text{Uczeń})(x \in \text{Sub}(\text{Klasa_2a}) \Rightarrow x \notin \text{Sub}(\text{Klasa_2b}). \\ \forall x \in \text{Sub}(\text{Uczeń})(x \in \text{Sub}(\text{Klasa_2b}) \Rightarrow x \notin \text{Sub}(\text{Klasa_2a}).$$

Gdyby istniał obiekt Uczeń należący do jednej i drugiej klasy, zostanie wygenerowany komunikat o błędzie. W zależności od umiejscowienia reakcji systemu albo nie zostanie dopuszczona taka sytuacja na etapie zarządzania obiektami lub też użytkownik będzie decydował jak rozwiązać ten problem (odwołanie transakcji, usunięcie obiektu).

◆

4.2.2. Wzajemne występowanie obiektów

Podobna sytuacja dla obiektów wieloklasowych jak w przypadku wzajemnego wykluczania istnieje dla **obiektów wzajemnie występujących**.

Przykład

$$\forall x \in \text{Sub}(\text{Uczeń})(x \in \text{Sub}(\text{Kolo_mat}) \Rightarrow x \in \text{Sub}(\text{Klasa_2a}).$$

Warunek ten jest niejako dopełnieniem warunku dla obiektów wzajemnie się wykluczających. W tym przypadku związek obiekt-podobiekt (uczeń uczestnik koła matematycznego) wymusza występowanie innego związku obiekt-podobiekt (ten sam obiekt ma być uczniem klasy 2a).

◆

4.3. Zależności między ograniczeniami

W bazie o rozbudowanej strukturze mogą występować różnego rodzaju zależności, także niezgodności lub sprzeczności przy definiowaniu ograniczeń dla obiektów. Do typowych zależności między warunkami ograniczającymi należą: wykluczanie ograniczeń, zawieranie ograniczeń, odwoływanie ograniczeń, specjalizacja ograniczeń, przesłanianie ograniczeń i wyłączanie obiektów.

a) Wykluczanie ograniczeń

Wykluczające się ograniczenia będą usiłować ustawić wartości w obiekcie na przeciwne. Przypuśćmy, że mamy następujące ograniczenia:

Przykład

Ograniczenie o_1

$\forall x \in \text{Sub}(\text{Dyrekcja})(x. \text{Wynagrodzenie} \geq 2500)$.

Ograniczenie o_2

$\forall x \in \text{Sub}(\text{Nauczyciel})(x. \text{Wynagrodzenie} \leq 2500)$.

Gdy spróbujemy wprowadzić wynagrodzenie 3500 zł dla nauczyciela-dyrektora pojawi się problem. Ograniczenie o_1 będzie spełnione. Natomiast ograniczenie o_2 nie będzie spełnione. Wartość wynagrodzenia zostanie więc usunięta. Podobna sytuacja zajdzie, gdy podamy wartość mniejszą od 2500. W rezultacie nie wprowadzimy żadnej wartości.



b) Zawieranie ograniczeń

Zawieranie ograniczeń pojawia się, gdy zbiór wartości atrybutu dopuszczalny przez ograniczenie jest podzbiorem zbioru wartości atrybutu dopuszczalnych przez inne ograniczenie.

Przykład

Ograniczenie o_3

$\forall x \in \text{Sub}(\text{Dyrekcja})(x. \text{Wynagrodzenie} \geq 2500)$.

Ograniczenie o_4

$\forall x \in \text{Sub}(\text{Nauczyciel})(x. \text{Wynagrodzenie} \geq 1500)$.

Wprawdzie ograniczenie o_4 pozwala na mniejsze od 2500 zł wynagrodzenie, ale ograniczenie o_3 jest bardziej restrykcyjne i to ono przejmuje kontrolę nad o_4 . Istnieją dwa możliwe scenariusze:

1. Wynagrodzenie ≥ 2500 .

Dominuje ograniczenie o_3 . Sprawdzanie o_4 powoduje tylko redundancję i obniża efektywność systemu.

2. Wynagrodzenie < 2500 .

Ograniczenie o_4 spełnione. Dominuje jednak ograniczenie o_3 i to ono decyduje o odrzuceniu wartości.



c) Odwoływanie ograniczeń

Może się tak zdarzyć, że tylko część podobiektów powinna być ograniczona. Zwykle ograniczenia nakładamy na wszystkie podobiekty.

Przykład

Ograniczenie o_5

$$\forall x \in \text{Sub}(\text{Jedn_szkolna}) (x.\text{Liczba_uczniów} \geq 30).$$

Takie ograniczenie spowoduje nałożenie minimalnej liczby uczniów dla wszystkich jednostek szkolnych, chociaż koła szkolne nie mają określanego minimum.



d) Specjalizacja ograniczeń

Polega na tworzeniu ograniczeń dotyczących obiektów-klas.

Przykład

Ograniczenie o_6

$$\forall x \in \text{Sub}(\text{Osoba}) (x.\text{Wiek} > 0 \wedge x.\text{Wiek} < 150).$$

Ograniczenie o_7

$$\forall x \in \text{Sub}(\text{Nauczyciel}) (x.\text{Wiek} > 25 \wedge x.\text{Wiek} < 60).$$



Ograniczenia o_6 dotyczy obiektu-klasy a o_7 jego podobiektu. Ograniczenie o_7 jest bardziej restrykcyjne i tylko ono powinno być sprawdzane w przypadku nauczycieli. Ze względu na efektywność przetwarzania ograniczenie o_6 nie powinno być wtedy sprawdzane.



e) Przesłanianie ograniczeń

Polega na usuwaniu jednego ograniczenia przez drugie, które dotyczy obiektów dziedziczonych z obiektu, do którego odnosi się pierwsze ograniczenie. Charakterystyczne jest wykluczanie się tych ograniczeń. W przeciwnym razie mielibyśmy do czynienia ze specjalizacją.

Przykład

Ograniczenie o_8

$$\forall x \in \text{Sub}(\text{Nauczyciel}) (x.\text{Pensum} \geq 18).$$

Ograniczenie o_9

$$\forall x \in \text{Sub}(\text{Dyrekcja}) (x.\text{Pensum} = 10).$$

W tym przypadku tylko jedno ograniczenie powinno być sprawdzane, ograniczenie o_9 powinno zasłonić ograniczenie o_8 dla podobiektów Dyrekcji.



f) Wyłączanie obiektów

Ograniczenia nałożone na podobiektory nie rozwiązują wszystkich problemów. Mogą pojawić się pojedyncze obiekty, na które nie nakłada się ograniczeń, pomimo że reszta takich wymaga. Musimy więc mieć możliwość wyłączania niektórych obiektów spod działania ograniczeń.

Przykład

Ograniczenie o_{10}

$$\forall x \in \text{Sub}(\text{Jedn_szkolna}) (x.\text{Liczba_uczniów} \geq 30 \\ \wedge x.\text{Nazw_jedn} \neq \text{Kolo_matematyczne}).$$

W przykładzie z punktu c) określając minimalną liczbę uczniów dla jednostki szkolnej nałożyliśmy także ograniczenie dla koła matematycznego. Możemy tego uniknąć precyzując odpowiednio warunek ograniczający, np. $\text{Nazw_jedn} \neq \text{Kolo_matematyczne}$.



Problemy przedstawione w punktach od a) do f) można rozwiązać przez wprowadzenie nowego pojęcia modelu, tj. **obiektu-ograniczenia**.

4.4. Obiekt-ograniczenie

Przyjmijmy, że ograniczenie będzie reprezentowane przez obiekt z czterema obligatoryjnymi atrybutami: *Dotyczy*, *Nie_dotyczy*, *Przesłania* i *Przesłaniane_przez*, a OGR oznacza zbiór obiektów-ograniczeń. Dla uproszczenia będziemy stosować zapis V_a zamiast zapisu $V_a(x)$.

1. Atrybut **Dotyczy** taki, że $V_{\text{Dotyczy}} \subseteq \text{OBJ}$.

Wartością atrybutu jest zbiór obiektów, których dotyczy ograniczenie.

Przyjmujemy, że jeśli w zbiorze obiektów występuje obiekt-klasa to ograniczenie dotyczy wszystkich jego podobiektów. Jeśli takim podobiektom jest obiekt-klasa, z którym zachodzi związek dziedziczenia to ograniczenie dotyczy także jego wszystkich podobiektów. I tak postępując rekurencyjnie ograniczenie przechodzi na wszystkie obiekty-instancje powiązane przez dziedziczenie lub bezpośrednio przez związek obiekt-podobiekt.

Ponieważ uwzględniane są podobiektory, więc musi być spełniony następujący warunek:

$$\forall x, y \in V_{\text{Dotyczy}} \Rightarrow \neg(x \leq y \vee x \geq y \vee x \angle y \vee y \angle x).$$

Oznacza to, że zbiór obiektów będący wartością atrybutu *Dotyczy* nie jest wzajemnie powiązany ani związkiem dziedziczenia ani też związkiem obiekt-podobiekt.

2. Atrybut **Nie_dotyczy** taki, że $V_{Nie_dotyczy} \subseteq OBJ$.

Wartością atrybutu jest zbiór obiektów, które są wyłączone z ograniczania. Podobnie jak dla atrybutu Dotyczy ograniczenie wyłącza ze sprawdzania podane obiekty oraz wszystkie dziedziczone obiekty.

Spełnione muszą być następujące warunki:

- a. Podobnie jak dla atrybutu Dotyczy zbiór identyfikatorów obiektów będący wartością atrybutu Nie_dotyczy nie jest wzajemnie powiązany ani relacją dziedziczenia ani relacją obiekt-podobiekt.

$$\forall x, y \in V_{Nie_dotyczy} \Rightarrow \neg(x \leq y \vee x \geq y \vee x \angle y \vee y \angle x).$$

- b. Jeśli obiekt znajduje się w zbiorze wartości jednego z atrybutów Dotyczy lub Nie_dotyczy to nie może występować w drugim atrybucie. W tym sensie są one rozłączne.

$$\forall x \in V_{Dotyczy} \Rightarrow x \notin V_{Nie_dotyczy}.$$

$$\forall x \in V_{Nie_dotyczy} \Rightarrow x \notin V_{Dotyczy}.$$

- c. Jeśli obiekt znajduje się w zbiorze wartości atrybutu Nie_dotyczy to musi istnieć obiekt, od którego dziedziczy, który należy do zbioru wartości atrybutu Dotyczy.

$$\forall x \in V_{Nie_dotyczy} \Rightarrow \exists y \in V_{Dotyczy} \wedge (y \leq x \vee y \angle x).$$

3. Atrybut **Przesłania** taki, że $V_{Przesłania} \subseteq OGR$.

Wartością atrybutu jest zbiór obiektów-ograniczeń zastąpionych przez ten obiekt-ograniczenie.

4. Atrybut **Przesłaniane_przez** taki, że: $V_{Przesłaniane_przez} \in OGR$.

Wartością atrybutu jest obiekt-ograniczenie, który zastępuje definiowany obiekt-ograniczenie.

$$\forall x \exists y x \in V_{Przesłania}(y) \Rightarrow V_{Przesłaniane_przez}(x) = y.$$

Atrybuty Przesłania i Przesłaniane_przez muszą być wzajemnie powiązane.

Dlaczego atrybuty Przesłania i Przesłaniane_przez są potrzebne? W celu uniknięcia przypadku ominięcia sprawdzania ograniczeń dla obiektu, zakłada się, że kiedy ograniczenie jest wyłączone dla pewnych obiektów, to przesłonięte przez niego ograniczenie tychże obiektów dotyczy. Można to osiągnąć jedynie dzięki użyciu atrybutu Przesłania, ponieważ gdyby go nie było, należałoby przeglądać wszystkie obiekty ograniczeń żeby znaleźć jeden, który zawiera odpowiedni identyfikator w swoim atrybucie Przesłaniane_przez. Jest to bardzo nieefektywne. Z drugiej strony, atrybut Przesłaniane_przez może być wykorzystany w algorytmie sprawdzającym poprawność obiektów po przeprowadzeniu zmian.

Biorąc pod uwagę określone powyżej atrybuty obiekt-ograniczenie definiujemy w następujący sposób.

Definicja

Obiektem-ograniczeniem x nazywamy wyrażenie $(id(x), Sub(x), Sup(x), A_{Sub}(x), v(x), M(x))$, gdzie

$$Sub(x) = Sup(x) = A_{Sub}(x) = \emptyset,$$

$v(x)$ jest funkcją, taką że $v(x): A(x) \rightarrow V(x)$, gdzie $a \in \{Dotyczy, Nie_dotyczy, Przesłania, Przesłaniane_przez\} \wedge V_{Dotyczy} \subseteq OBJ \wedge V_{Nie_dotyczy} \subseteq OBJ \wedge V_{Przesłania} \subseteq OGR \wedge V_{Przesłaniane_przez} \in OGR$,

przy czym:

$$(1) \quad \forall x, y \in V_{Dotyczy} \Rightarrow \neg(x \leq y \vee x \geq y \vee x \angle y \vee y \angle x).$$

$$(2) \quad \forall x, y \in V_{Nie_dotyczy} \Rightarrow \neg(x \leq y \vee x \geq y \vee x \angle y \vee y \angle x).$$

$$(3) \quad \forall x \in V_{Dotyczy} \Rightarrow x \notin V_{Nie_dotyczy}.$$

$$(4) \quad \forall x \in V_{Nie_dotyczy} \Rightarrow x \notin V_{Dotyczy}.$$

$$(5) \quad \forall x \in V_{Nie_dotyczy} \Rightarrow \exists y \in V_{Dotyczy} \wedge (y \leq x \vee y \angle x).$$

$$(6) \quad \forall x \exists y x \in V_{Przesłania}(y) \Rightarrow V_{Przesłaniane_przez}(x) = y.$$

♦

Z definicji obiektu i obiektu-ograniczenia wynika, że $OGR \subseteq OBJ$.

4.5. Przykłady obiektów-ograniczeń

Zdefiniujemy teraz obiekty-ograniczenia dla zależności podanych w rozdziale 4.3.

a) Wykluczanie ograniczeń

Aby rozwiązać problem wykluczania ograniczeń wybieramy, które z ograniczeń jest ważniejsze.

Przykład

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
o_1	∅	∅	∅	<Dotyczy: {Dyrekcja}, Nie_dotyczy: {Nauczyciel}, Przesłania: ∅, Przesłaniane_przez: ∅>	o ₁
o_2	∅	∅	∅	<Dotyczy: {Nauczyciel}, Nie_dotyczy: ∅, Przesłania: ∅, Przesłaniane_przez: ∅>	o ₂

Ograniczenia o_1 i o_2 są zdefiniowane dla różnych obiektów, ale oba są dziedziczone przez obiekt Dyrekcja. Aby rozwiązać niejednoznaczność, projektant bazy danych musi zadeklarować, które z ograniczeń dominuje. W przykładzie wskazano, że obiektu Dyrekcja dotyczy ograniczenie o_1 .

♦

b) Zawieranie ograniczeń

Podobnie jak w punkcie a) problem wyboru ograniczenia nadrzędnego spoczywa na projektancie bazy danych. Rozwiązanie jest następujące: Dyrekcja musi znaleźć się w wartości atrybutu Nie_dotyczy.

Przykład

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
o_3	∅	∅	∅	<Dotyczy: {Dyrekcja}, Nie_dotyczy: ∅, Przesłania: ∅, Przesłaniane_przez: ∅>	o ₃
o_4	∅	∅	∅	<Dotyczy: {Nauczyciel}, Nie_dotyczy: {Dyrekcja}, Przesłania: ∅, Przesłaniane_przez: ∅>	o ₄

Ograniczenia o₃ i o₄ są zdefiniowane w różnych gałęziach schematu, ale „spotykają się” w klasie Dyrekcja. Problemem jest to, że o₃ dominuje nad o₄ i niejednoznaczność, które z ograniczeń powinno być użyte, powinna być rozwiązana przez projektanta bazy danych. Tu zdecydowano, że o₃ jest odpowiednim ograniczeniem dla obiektu Dyrekcja, dlatego w o₄ obiekt Dyrekcja znajduje się w wartości atrybutu Nie_dotyczy.

**c) Odwołanie ograniczeń****Przykład**

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
o_5	∅	∅	∅	<Dotyczy: {Jedn_szkolna}, Nie_dotyczy: {Kolo_mat}, Przesłania: ∅, Przesłaniane_przez: ∅>	o ₅

Dla ograniczenia o₅ nakładanego na obiekt Jedn_szkolna i jej podobiektów wyłączony jest obiekt Kolo_mat.

**d) Specjalizacja ograniczeń**

Ograniczenie o₆ powinno być zastąpione przez o₇ w odniesieniu do klasy Nauczyciel i jej podobiektów.

Przykład

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
o_6	∅	∅	∅	<Dotyczy: {Osoba}, Nie_dotyczy: ∅, Przesłania: ∅, Przesłaniane_przez: o_7>	o ₆
o_7	∅	∅	∅	<Dotyczy: {Nauczyciel}, Nie_dotyczy: ∅, Przesłania: {o_6}, Przesłaniane_przez: ∅>	o ₇



Ograniczenia o₆ i o₇ dotyczą obiektów Osoba i Nauczyciel. Jednakże tylko ograniczenie o₇ powinno być sprawdzane, ponieważ jest bardziej restrykcyjne. Sprawdzanie ograniczenia o₆ nie jest niepoprawne, ale jest nieefektywne. Dlatego obiekt-ograniczenie o_6 powinien być zastąpiony przez o_7 w klasie Nauczyciel.

**e) Przesłanianie ograniczeń**

W tym przypadku tylko ostatnie ograniczenie powinno być uwzględnione, ponieważ przesłania ono poprzednie. Gdy oba będą dostępne jednocześnie nic nie zostanie zapisane.

Przykład

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
o ₈	∅	∅	∅	<Dotyczy: {Nauczyciel}, Nie_dotyczy: ∅, Przesłania: ∅, Przesłaniane_przez: o ₉ >	o ₈
o ₉	∅	∅	∅	<Dotyczy: {Dyrekcja}, Nie_dotyczy: ∅, Przesłania: {o ₈ }, Przesłaniane_przez: ∅>	o ₉

W tym przypadku ograniczenie o₉ powinno być wymuszane dla obiektu Dyrekcja, ponieważ przesłania ograniczenie o₈. Wymuszanie ograniczenia o₈ jest błędne, ponieważ niedozwolona byłaby aktualizacja klasy Dyrekcja. Dlatego też ograniczenie o₈ musi być zastąpione przez o₉ w obiekcie-klasie Dyrekcja.

**f) Wyłączanie obiektów****Przykład**

Id(x)	Sub(x)	Sup(x)	A _{Sub} (x)	v(x)	M(x)
o ₁₀	∅	∅	∅	<Dotyczy: {Jedn_szkolna}, Nie_dotyczy: {Kolo_mat}, Przesłania: ∅, Przesłaniane_przez: ∅>	o ₁₀

Ograniczenie o₁₀ odnosi się do wszystkich jednostek szkolnych oprócz koła matematycznego. Wyłączenie ograniczenia dla tego obiektu można uzyskać dzięki atrybutowi Nie_dotyczy.

**4.6. Integralność migracji obiektów**

Czasami migracja obiektów jest bezsensowna. W szczególności może przebiegać:

- (a) z klasy do nadklasy,
- (b) z klasy do podklasy,
- (c) z klasy do innej klasy.

Rozważmy trzy obiekty-klasy: Osoba, Mężczyzna, Kobieta. Migracja obiektów typu (a), (b) dla Osoby i Kobiety nie ma sensu, ponieważ klasa Osoba nie posiada obiektów nie należących do klas Mężczyzna lub Kobieta. W powyższym przykładzie także migracja typu (c) nie ma sensu, ponieważ obiekty klas Mężczyzna i Kobieta są znaczeniowo rozłączne.

Ponadto problemem może być kierunek zmian, np. obiekty-klasy Dzieci i Dorośli. Obiekt-instancja Krzyś z klasy Dzieci może migrować do klasy Dorośli, ale nie na odwrót.

Podstawowe pytanie, jakie więc pojawia się przy migracji obiektów jest następujące: czy po przeprowadzeniu migracji zostanie zachowana integralność danych? Najczęściej stosowane rozwiązanie tego problemu opiera się na specyfikacji zmian podanej przez projektanta systemu [Alb95b, Ric91].

W modelu obiektów wieloklasowych integralność migracji musimy rozpatrywać biorąc pod uwagę operatory Dodaj, Usuń i Przenieś.

Przy migracji obiektów występują dwa ogólne przypadki: pojedyncze przeniesienie, dodanie czy usunięcie obiektu oraz sekwencja przeniesienia, dodania lub usunięcia kilku

obiektów. W obu przypadkach możemy mieć do czynienia z niedozwoloną zmianą. Zabronioną operacją jest np. migracja ucznia do klasy Nauczyciel. Taka operacja nie powinna być zaprojektowana. Sekwencje operacji mogą być kontrolowane przez obiekty-ograniczenia. Możemy utworzyć taki obiekt, w którym warunek ograniczający będzie sprawdzał czy wystąpienie obiektu w jednej klasie jest równoznaczne z wystąpieniem innego obiektu w odpowiedniej klasie lub wykluczeniem go z określonej klasy. Przypadek ten analizowaliśmy wcześniej.

Opisane rozwiązanie nie zabezpiecza nas przed wszystkimi problemami powstającymi po migracji obiektów. Rozważmy np. przeniesienie obiektów.

Przykład

Mamy następujące obiekty-klasy:

$(K, \text{Sub}(K), \{Z\}, \{\text{Nazwisko}, \text{Rok_ur}\}, \emptyset, M(K)),$

$(S, \text{Sub}(S), \text{Sup}(S), \{\text{Nazwisko}, \text{Adres}\}, \emptyset, M(S)),$

$(M, \text{Sub}(M), \{Z\}, \{\text{Nazwisko}, \text{Rok_ur}\}, \emptyset, M(M)),$

$(Z, \{K, M\}, \text{Sup}(Z), \{\text{Nazwisko}, \text{Rok_ur}\}, \emptyset, M(Z)),$

oraz obiekt

$(\text{Uczeń_1}, \emptyset, \{K\}, \emptyset, \langle \text{Imię:Maciej}, \text{Nazwisko:Kruk}, \text{Rok_ur:1990}, \text{Płeć:M}, \text{Ocena:5}, 4.0, 3.5 \rangle, \emptyset).$

Założmy ponadto, że

$(\text{Nazwisko}, K) = (\text{Nazwisko}, S) = (\text{Nazwisko}, M)$

oraz

$(\text{Rok_ur}, K) = (\text{Rok_ur}, M) = (\text{Rok_ur}, Z).$

Porównajmy operację $(K \xrightarrow[\text{przenieś}]{\text{Uczeń_1}} M)$

z sekwencją operacji $(K \xrightarrow[\text{przenieś}]{\text{Uczeń_1}} S), (S \xrightarrow[\text{przenieś}]{\text{Uczeń_1}} M).$

Po wykonaniu sekwencji przeniesienia obiekt Uczeń_1 jest następujący:

$(\text{Uczeń_1}, \emptyset, \{M\}, \emptyset, \langle \text{Imię:Maciej}, \text{Nazwisko:Kruk}, \text{Rok_ur:1991}, \text{Płeć:M}, \text{Ocena:5}, 4.0, 3.5 \rangle, \emptyset).$

Wartość atrybutu Rok_ur mogła ulec zmianie (poprzednia wartość 1990 została zmieniona na 1991), ponieważ przenosząc obiekt do obiektu-klasy S usunęliśmy ten atrybut. Musimy ponownie określić wartość dla tego atrybutu, gdyż pojawił się on po przeniesieniu do obiektu-klasy M . Możemy więc podać inną wartość niż początkowo.

Podczas gdy operacja $(K \xrightarrow[\text{przenieś}]{\text{Uczeń_1}} M)$ zachowuje tą wartość, ponieważ atrybut Rok_ur nie był usuwany.

W obu przypadkach uzyskano ten sam skutek pod względem przynależności obiektu do klasy, jednakże z innymi wartościami. Rezultat taki możemy otrzymać zawsze wtedy, gdy w

sekwencji operacji przeniesienia usuniemy atrybut, który następnie dodamy (dotyczy atrybutów o tym samym pochodzeniu).



Na tej podstawie nasuwa się następujący wniosek: im mniej operacji przeniesienia tym większe prawdopodobieństwo zachowania informacji przeprowadzanej migracji. Stwierdzenie to ma duże znaczenie dla projektanta bazy danych. Szczególnie przy przenoszeniu obiektów z jednej klasy do drugiej należy zwrócić uwagę czy zostanie zachowana kompletność informacyjna obiektu.

4.7. Wnioski

Integralność w bazie obiektów wieloklasowych nabiera nowego znaczenia. Aby zapewnić prawidłowe działanie systemu musimy nałożyć pewne ograniczenia na obiekty. Problem modelowania dziedziny bazy danych z uwzględnieniem ograniczeń został przedstawiony na początku tego rozdziału. Podano przykłady ograniczeń sformułowane za pomocą języka logiki pierwszego rzędu. Modelowaniu podlegała nie tylko dziedzina bazy danych w postaci wartości atrybutów, ale również relacje łączące obiekty. Ponieważ wprowadzane ograniczenia są często niewystarczające ze względu na pojawiające się wyjątki, w drugiej części przedstawiono problemy zależności między ograniczeniami, jakie mogą się pojawić w bazie danych oraz sposób ich rozwiązania.

Analizując problem konstruowania ograniczeń dla obiektów znaleźliśmy rozwiązanie jak je definiować. W tym celu używamy obiektu, w którym umieszczamy cztery obligatoryjne atrybuty: `Dotyczy`, `Nie_dotyczy`, `Przesłania`, `Przesłaniane_przez`. Taki obiekt nazwaliśmy obiektem-ograniczeniem. Metoda w tym obiekcie określa warunek, który mają spełniać odpowiednie obiekty. Definicja obiektu-ograniczenia zapewnia powiązanie ograniczeń z obiektami.

5. Język manipulacji obiektami wieloklasowymi

W systemach obiektowych języki manipulacji służą do opisu przekształceń obiektów w bazie danych. W wyniku tych przekształceń otrzymujemy obiekty zawierające informacje, które nas interesują. Często język manipulacji danymi nazywany jest językiem zapytań. Nie jest to jednak właściwe określenie, gdyż w rzeczywistości poza pytaniami język taki umożliwia przeprowadzanie zmian w bazie danych.

Możemy wyróżnić następujące grupy języków w obiektowych bazach danych: języki proceduralne [Bla93, Kri93, Kro95c, Kro01, Pia96], języki strukturalne [Bus93, Uni95a, Uni95b], języki oparte na algebrze obiektowej [Alh93b, Bee94, Bee93, Mit94], języki wzorowane na języku naturalnym [Bro91]. Języki proceduralne są konstruowane na podstawie języków programowania obiektowo-zorientowanego, takich jak język C++ czy język Java. Używanie tych języków do manipulacji obiektami polega na rozszerzeniu standardowego zestawu operacji języka programowania o operacje umożliwiające działanie na obiektach bazy danych. Języki proceduralne są szczególnie przydatne do formułowania operacji złożonych, często trudno realizowalnych w innych grupach języków manipulacji danymi. Największą grupę języków stanowią języki strukturalne wywodzące się z języka SQL, np. OSQL, ORL, SQL3. W językach opartych na algebrze obiektowej np. OQL, użytkownik podaje jedynie kolejność wykonywania operacji oraz argumenty każdej z nich. Argumentami najczęściej są zbiory obiektów, zbiory atrybutów i wyrażenia predykatowe. Ostatnią grupę, języków opartych na językach naturalnych, charakteryzuje to, że żądania są formułowane w języku naturalnym.

Algebra obiektów opisana w rozdziale drugim, pozwala na formułowanie zapytań w obiektowej bazie danych. Jednakże posługiwanie się operatorami algebry nie jest łatwe, dlatego też proponujemy bardziej przyjazny sposób do wyrażania zapytań przez użytkownika.

W tym rozdziale zaprezentujemy język rozszerzony w stosunku do języka SQL 92 [SQL92, Gru96]. Nazwiemy go MOW (Manipulacja Obiektami Wieloklasowymi). Ze względu na popularność języka SQL takie podejście jest w pełni uzasadnione. Definiowanie języka MOW polega na tym, że odpowiednie instrukcje języka SQL przekształcamy tak, aby odzwierciedlały zarządzanie obiektami wieloklasowymi. Do opisu składni języka MOW używać będziemy notacji BNF [Igl84].

W klasycznym podejściu wyszukiwania w bazie danych każdy zbiór musi być jednorodny, tj. musi zawierać elementy (obiekty, rekordy, krotki) o takiej samej strukturze. W algebrze obiektów wieloklasowych a w konsekwencji w języku MOW dopuszczamy do przetwarzania także niejednorodne zbiory obiektów.

Istnieją dwa rodzaje instrukcji w języku MOW: instrukcja wyszukiwania obiektów oraz instrukcje zmiany w obiektach. Drugi rodzaj obejmuje: wprowadzanie, usuwanie i aktualizację obiektów. Te zmodyfikowane względem standardu języka SQL instrukcje zmiany w obiektach w konsekwencji umożliwiają migrację obiektów wieloklasowych.

Dalej opiszemy kolejno instrukcję wyszukiwania, instrukcję wprowadzania, instrukcję usuwania i instrukcję aktualizacji obiektów. Pokażemy też, w jaki sposób instrukcje języka MOW są przekształcane na odpowiednie operacje zmian w obiektach.

5.1. Wyrażenia ścieżkowe

Złożona struktura obiektów (zarówno logiczna, jak i fizyczna) wymaga od obiektowego języka zapytań specjalnych mechanizmów wyszukiwania w takich strukturach. Co więcej, język który pozwala na tworzenie obiektów, potrzebuje mechanizmów do budowania nowych struktur. Jednym z najczęściej stosowanych pojęć wykorzystywanych w formułowaniu zapytań do obiektowych baz danych jest **wyrażenie ścieżkowe** [Pan00, Ull00]. Przeszukiwanie i tworzenie obiektów prowadzi do regularnego używania wyrażeń ścieżkowych w nawigacji po schemacie obiektów. Przetwarzanie złożonych struktur obiektów powoduje również to, że w językach używane są zapytania zagnieżdżone, w których zmienne z zewnętrznych wyrażeń są używane w wyrażeniach zagnieżdżonych. Definicja wyrażenia ścieżkowego musi być dostosowana do struktury obiektu wieloklasowego.

Definicja

Wyrażeniem ścieżkowym lub **ścieżką** nazywamy wyrażenie o postaci $x_1.x_2...x_n.A$,

gdzie x_i dla $1 \leq i \leq n$ jest zmienną (oznaczającą obiekt lub zbiór obiektów), A jest atrybutem zdefiniowanym bądź w obiekcie wskazywanym przez x_n , albo w podobiektach x_n , albo w obiektach referencyjnych wskazywanych przez x_n , przy czym pomiędzy obiektami x_{i-1} i x_i zachodzi związek obiekt-podobiekt.



Symbol $x.A$ oznacza wartość atrybutu A w obiekcie x . Wartością wyrażenia ścieżkowego jest pojedynczy element (obiekt lub wartość), jeśli atrybut występujący w ścieżce jest jednowartościowy, albo zbiór elementów, jeśli x jest zbiorem podobiektów lub atrybut A jest wielowartościowy.

Ścieżki są obiektowym odpowiednikiem wyrażenia relacyjnego obejmującego złączenia i rzut. Ścieżki nie wprowadzają nowych obiektów, ale pozwalają na nawigację po bazie danych w poszukiwaniu obiektów, które już istnieją.

Przykład

1. `Uczeń_1.Rok_ur` – wartość atrybutu.
2. `Sub(Klasa_2a).Płeć` – zbiór wartości atrybutu dla podobiektów.
3. `Dolny Ślask.Szkoła_12.Dyrektor` – obiekt referencyjny.
4. `Dolny Ślask.Szkoła_12.Dyrektor.Dodatek` – wartość atrybutu obiektu referencyjnego.



Wyrażenia ścieżkowe można rozpatrywać w dwóch aspektach: po pierwsze ze względu na związek obiekt-podobiekt, po drugie ze względu na związek dziedziczenia.

Przykład

1. Związek obiekt-podobiekt.
`Szkoła_11.Klasa_2a.Uczeń_1.Płeć`.

2. Związek dziedziczenia.

Jedn_szkolna.Klasa_2a.Liczba_uczniów.



5.2. Instrukcja wyszukiwania obiektów

Składnia instrukcji wyszukiwania obiektów **select-from-where** w MOW jest następująca:

<Instrukcja wyszukiwania> :: =

select <Klauzula celu> |

select <Klauzula celu> **from** <Klauzula zakresu> |

select <Klauzula celu> **where** <Klauzula kwalifikacji> |

select <Klauzula celu> **from** <Klauzula zakresu> **where** <Klauzula kwalifikacji>

Przykład

1. Podaj nazwiska osób.

select f.Nazwisko

2. Podaj nazwiska osób, które są w klasie 2a.

select f.Nazwisko **from** f **in** Sub(Klasa_2a)

3. Podaj nazwiska osób, które są jednocześnie mężczyznami.

select f.Nazwisko **where** f.Płeć="M"

4. Podaj nazwiska osób, które są jednocześnie mężczyznami w klasie 2a.

select f.Nazwisko **from** f **in** Sub(Klasa_2a) **where** f.Płeć="M"



Zadając pytania w obiektowym systemie bazy danych ważną rolę pełni możliwość pytania przez użytkownika o dowolną wartość wybranego atrybutu. Podobną rolę w języku manipulacji obiektami wieloklasowymi pełni możliwość pytania o zbiór nadobiektów lub podobiektów.

Przykład

Podaj nazwiska nauczycieli klasy 2a.

select f.Nazwisko **from** f **in** Sub(Klasa_2a) **where** f **isin** Sub(Nauczyciel)



5.2.1. Klauzula celu

Klauzula celu w języku MOW różni się tym od swojego pierwowzoru z języka SQL, że wyrażenie ścieżkowe, które tam się pojawia może dotyczyć atrybutów wielowartościowych.

Przykład

Podaj imiona osób.

select f.Imię



Podobnie jak w języku SQL, możemy używać słowa kluczowego **distinct**, które usuwa duplikaty w odpowiedzi.

Przykład

Podaj wszystkie różne nazwiska osób z klasy 2a i z klasy 2b.

select distinct f.Nazwisko **where** f **isin** Sub(Klasa_2) **and** Sub(Klasa_2)



5.2.2. Klauzula zakresu

Klauzula ta zezwala na użycie zmiennych zakresowych.

Przykład

Określ zakres wyszukiwania do podobiektów klasy 2a.

select f.Nazwisko **from** f **in** Sub(Klasa_2a)



5.2.3. Klauzula kwalifikacji

Podobnie jak w języku SQL w klauzuli **where** specyfikujemy predykat (wyrażenie boolowskie), który muszą spełnić obiekty otrzymane w wyniku wyszukiwania. Przykładami z zastosowaniem predykatu typu porównanie są odpowiednio:

Przykład

1. Podaj osoby i uczniów o tych samych imionach.

select f, f.Uczeń **from** f **in** Sub(Osoba) **where** f.Imię=f.Uczeń.Imię

2. Podaj nazwiska osób, które są jednocześnie mężczyznami w klasie 2a.

select f.Nazwisko **from** f **in** Sub(Klasa_2a) **where** f.Płeć="M"

3. Podaj osoby z klasy 2a nie będące uczniami.

select Osoba **from** f **in** Sub(Klasa_2a) **where** Osoba $\neq$ **all** (**select** Sub(Uczeń))



5.2.4. Operatory mnogościowe i agregujące

Aby rozszerzyć możliwości wyszukiwania wprowadzamy pomocnicze operatory mnogościowe: **union**, **difference** i **intersect**, które mają takie samo znaczenie jak operatory w algebrze obiektów wieloklasowych. Instrukcje, w którym uczestniczą operatory mnogościowe mają następującą składnię:

<Instrukcja wyszukiwania> :: = <Instrukcja wyszukiwania> <Operator mnogościowy>
<Instrukcja wyszukiwania>

Przykład

Podaj osoby z klasy 2a i z klasy 2b.

```
(select Osoba from f in Sub(Klasa_2a))
union
(select Osoba from g in Sub(Klasa_2b))
```



Bardzo przydatnym operatorem jest operator agregujący: **count**, **avg**, **sum**, **max**, **min** [Kro00a]. Jego argumentem jest zbiór obiektów.

Przykład

Ile uczniów liczy klasa 2a?

```
count ((select Osoba from f in Sub(Klasa_2a))
```

**Przykład**

Podaj uczniów z klasy 2a o średniej ocen mniejszej niż 4.0.

```
select Osoba from f in Sub(Klasa_2a) where avg(f.ocena)<4.0
```



Ponieważ instrukcja wyszukiwania obiektów wieloklasowych jest wzorowana na odpowiedniej instrukcji języka ORL [Urb94a] nie będziemy dalej dogłębnie jej analizować. Kompletny opis tej instrukcji w notacji BNF znajduje się w Załączniku. Bardziej interesujące z punktu widzenia modelu obiektów wieloklasowych jest analiza instrukcji wprowadzania, usuwania i aktualizacji obiektów.

5.3. Instrukcja wprowadzania obiektów

Składnia instrukcji wprowadzania obiektów w MOW jest następująca:

```
<Instrukcja wprowadzania> ::= insert <Obiekt> | insert <Oid> into <Oid> {, <Oid>}
| insert <Oid> into <Instrukcja wyszukiwania>
```

Zatem wyróżniamy dwie sytuacje:

1. Tworzenie obiektu:

```
insert <Obiekt>
```

Przykład

```
insert (Dolny Śląsk, Ø, Ø, {Nr_szkoły, Dyrektor, Liczba_uczniów}, <Ø>, Ø)
```

```
insert (Nauczyciel_1, Ø, Ø, Ø, <Imię:Beata, Nazwisko:Bąk, Rok_ur:1960, Przedmiot:język
polski, Dodatek:100>, Ø)
```

```
insert (Szkoła_11, Ø, {Dolny Śląsk}, {Nazwa_jedn, Liczba_uczniów}, <Nr_szkoły:11,
Dyrektor:Nauczyciel_1, Liczba_uczniów:2>, {Średnia_zarobków})
```

insert (Klasa_2a, $\emptyset$, {Szkoła_11}, {Imię, Nazwisko, Rok_ur}, <Nazwa_jedn:2a, Liczba_uczniów:1, Liczba_chłopców:1>, {Średnia_ocen})

insert (Uczeń_1, $\emptyset$, {Klasa_2a}, $\emptyset$, <Imię:Maciej, Nazwisko:Kruk, Rok_ur:1990, Płeć:M, Ocena:5, 4.0, 3.5>, $\emptyset$)

◆

2. Dodanie obiektu do zbioru podobiektów innego obiektu lub obiektów:

insert <Oid> **into** <Oid> {, <Oid>} | **insert** <Oid> **into** <Instrukcja wyszukiwania>

Przykład

insert Uczeń_1 **into** Koło_mat

insert Uczeń_1 **into**

select Jedn_Szkolna **from** f **in** Sub(Szkoła_11)

◆

5.4. Instrukcja usuwania obiektów

Składnia instrukcji usuwania obiektów w MOW jest następująca:

<Instrukcja usuwania> ::= **delete** <Oid> | **delete** <Oid> **from** <Oid> {, <Oid>}
| **delete** <Oid> **from** <Instrukcja wyszukiwania>

Także w tym przypadku wyróżniamy dwie sytuacje:

1. Usunięcie obiektu:

delete <Oid>

Przykład

delete Uczeń_1

◆

2. Usunięcie obiektu ze zbioru podobiektów innego obiektu lub obiektów:

delete <Oid> **from** <Oid> {, <Oid>} | **delete** <Oid> **from** <Instrukcja wyszukiwania>

Przykład

delete Uczeń_2 **from** Koło_mat, Klasa_2b

delete Uczeń_1 **from**

select Jedn_Szkolna **from** f **in** Sub(Szkoła_11)

◆

5.5. Instrukcja aktualizacji obiektów

Składnia instrukcji aktualizacji obiektów w MOW jest następująca:

<Instrukcja aktualizacji> ::= <Instrukcja aktualizacji wartości> | <Instrukcja aktualizacji
atrybutów> | <Instrukcja aktualizacji obiektu>

<Instrukcja aktualizacji wartości> ::= **update** <Oid> **set** <<Atrybut> : <Wartość>

{,<Atrybut> : <Wartość>}>

<Instrukcja aktualizacji atrybutów> :: = **update** <Oid> **del** <Atrybut> {,<Atrybut>}

<Instrukcja aktualizacji obiektu> :: = **update** <Oid> **from** <Oid> {,<Oid>}

into <Oid> {, <Oid>} | **update** <Oid> **from** <Oid> {,<Oid>} **into** <Instrukcja wyszukiwania> | **update** <Oid> **from** <Instrukcja wyszukiwania> **into** <Oid> {, <Oid>} | **update** <Oid> **from** <Instrukcja wyszukiwania> **into** <Instrukcja wyszukiwania>

W przypadku aktualizacji obiektów mamy trzy sytuacje:

1. Aktualizacja wartości lub dodanie atrybutu i jego wartości:

update <Oid> **set** <<Atrybut> : <Wartość> {,<Atrybut> : <Wartość>}>

Przykład

update Uczeń_1 **set** <Nazwisko:Król, Rok_ur:1991, Wzrost:140>

♦

2. Usunięcie atrybutu z obiektu:

update <Oid> **del** <Atrybut> {,<Atrybut>}

Przykład

update Uczeń_1 **del** Wzrost

♦

3. Aktualizacja związku obiekt-podobiekt:

<Instrukcja aktualizacji obiektu> :: = **update** <Oid> **from** <Oid> {,<Oid>}

into <Oid> {, <Oid>} | **update** <Oid> **from** <Oid> {,<Oid>} **into** <Instrukcja wyszukiwania> | **update** <Oid> **from** <Instrukcja wyszukiwania> **into** <Oid> {, <Oid>} | **update** <Oid> **from** <Instrukcja wyszukiwania> **into** <Instrukcja wyszukiwania>

Przykład

update Klasa_2a **from** Szkoła_11 **into** Szkoła_12

♦

Przykład. Generalizacja obiektu

update Nauczyciel_1 **from** Nauczyciel **into** Osoba

♦

Przykład. Specjalizacja obiektu

update Nauczyciel_1 **from** Nauczyciel **into** Dyrekcja

♦

5.6. Instrukcje konfiguracji migracji obiektów

Składnia instrukcji konfiguracji migracji obiektów w MOW jest następująca:

<Instrukcja konfiguracji> :: = **set migration checked** | **set migration unchecked**

Instrukcja **set migration checked** powoduje, że przed wykonaniem operacji migracji wywołamy operator sprawdź i w przypadku:

- instrukcji **insert** $id(x)$ **into** $id(y_1), id(y_2), \dots, id(y_m)$ sprawdzamy czy podane obiekty nie są już nadobiektami zmienianego obiektu,
- instrukcji **delete** $id(x)$ **from** $id(x_1), id(x_2), \dots, id(x_n)$ sprawdzamy czy podane obiekty są nadobiektami zmienianego obiektu,
- instrukcji **update** $id(x)$ **from** $id(x_1), id(x_2), \dots, id(x_n)$ **into** $id(y_1), id(y_2), \dots, id(y_m)$ sprawdzamy czy obiekty występujące po słowie **into** nie są już nadobiektami zmienianego obiektu oraz czy obiekty występujące po słowie **from** są nadobiektami zmienianego obiektu.

W wszystkich przypadkach, gdy operator sprawdź zwróci wartość logiczną **true**, to operacja migracji nie zostanie wykonana.

Instrukcja **set migration unchecked** powoduje, że operacje migracji nie będą wcześniej sprawdzane operatorem sprawdź.

5.7. Przykładowa baza utworzona w języku MOW

Utworzymy teraz bazę danych Kuratorium za pomocą instrukcji języka MOW.

Przykład

set migration checked

insert (Dolny Ślask, $\emptyset$, $\emptyset$, {Nr_szkoły, Dyrektor, Liczba_uczniów}, $\langle \emptyset \rangle$, $\emptyset$)

insert (Szkoła_11, $\emptyset$, {Dolny Ślask}, {Nazwa_jedn, Liczba_uczniów}, <Nr_szkoły:11, Dyrektor: $\emptyset$, Liczba_uczniów:2>, {Średnia_zarobków})

insert (Szkoła_12, $\emptyset$, {Dolny Ślask}, {Nazwa_jedn, Liczba_uczniów}, <Nr_szkoły:12, Dyrektor: $\emptyset$, Liczba_uczniów:1>, {Średnia_zarobków})

insert (Nauczyciel_1, $\emptyset$, $\emptyset$, $\emptyset$, <Imię:Beata, Nazwisko:Bak, Rok_ur:1960, Przedmiot:język polski, Dodatek:100>, $\emptyset$)

insert Szkoła_11, Szkoła_12 **set** <Dyrektor:Nauczyciel_1>

insert (Jedn_szkolna, $\emptyset$, $\emptyset$, {Liczba_uczniów}, $\langle \emptyset \rangle$, $\emptyset$)

insert (Klasa_2a, $\emptyset$, {Szkoła_11, Jedn_szkolna}, {Imię, Nazwisko, Rok_ur}, <Nazwa_jedn:2a, Liczba_uczniów:1, Liczba_chłopców:1>, {Średnia_ocen})

insert (Klasa_2b, $\emptyset$, {Szkoła_11, Jedn_szkolna}, {Imię, Nazwisko, Rok_ur}, <Nazwa_jedn:2b, Liczba_uczniów:1>, {Średnia_ocen})

insert (Klasa_2c, $\emptyset$, {Szkoła_12, Jedn_szkolna}, {Imię, Nazwisko}, <Nazwa_jedn:2c, Liczba_uczniów:1>, {Średnia_ocen})

insert (Koło_mat, $\emptyset$, {Szkola_11, Szkola_12, Jedn_szkolna}, {Imię, Nazwisko}, <Nazwa_jedn:Koło_matematyczne, Liczba_uczniów:1>, $\emptyset$)

insert (Osoba, $\emptyset$, $\emptyset$, {Imię, Nazwisko}, < $\emptyset$ >, {Wiek})

insert (Uczeń, $\emptyset$, {Osoba}, {Imię, Nazwisko, Płeć}, < $\emptyset$ >, $\emptyset$, $\emptyset$)

insert (Nauczyciel, $\emptyset$, {Osoba}, {Imię, Nazwisko, Przedmiot}, < $\emptyset$ >, {Liczba_polonistów})

insert Nauczyciel_1 **into** Nauczyciel

insert (Dyrekcja, {Nauczyciel_1}, {Nauczyciel, Osoba}, {Imię, Nazwisko, Przedmiot, Dodatek}, < $\emptyset$ >, $\emptyset$)

insert (Uczeń_1, $\emptyset$, {Klasa_2a, Uczeń, Osoba}, $\emptyset$, <Imię:Maciej, Nazwisko:Kruk, Rok_ur:1990, Płeć:M, Ocena:5, 4.0, 3.5>, {Średnia_ocen})

insert (Uczeń_2, $\emptyset$, {Klasa_2b, Koło_mat, Uczeń, Osoba}, $\emptyset$, <Imię:Jan, Nazwisko:Kula, Rok_ur:1990, Płeć:M, Ocena:4.5>, {Średnia_ocen})

insert (Uczeń_3, $\emptyset$, {Klasa_2c, Uczeń, Osoba}, $\emptyset$, <Imię:Alicja, Nazwisko:Nowak, Ocena:3.0, 2.0, Płeć:K>, {Średnia_ocen})

Jest to jeden z możliwych scenariuszy tworzenia bazy Kuratorium. Inny scenariusz mógłby powstać gdyby zamiast instrukcji **insert** użylibyśmy instrukcji **update**, gdyby zamiast instrukcji **update from into** użylibyśmy instrukcji **insert into** i instrukcji **delete from**. Na uwagę zasługuje tworzenie wartości referencyjnej instrukcją **update set** oraz uzupełnienie podobiektów obiektu Nauczyciel instrukcją **insert into**.



5.8. Translacja instrukcji języka MOW do wyrażeń algebry obiektów

5.8.1. Translacja instrukcji wprowadzania obiektów

Wyróżniamy dwie sytuacje:

1. Tworzenie obiektu:

Instrukcji **insert** ($id(x)$, $Sub(x)$, $Sup(x)$, $A_{Sub}(x)$, $v(x)$, $M(x)$) odpowiada operator tworzenia obiektu x - operator 11 z listy podstawowych operatorów zmian.

2. Dodanie obiektu do zbioru podobiektów innego obiektu lub obiektów:

Instrukcji **insert** $id(x)$ **into** $id(y_1)$, $id(y_2)$, ..., $id(y_m)$ odpowiada operator migracji obiektów $\frac{id(x)}{dodaj+}$ o następującej postaci:

$$\frac{id(x)}{dodaj+} id(y_1), id(y_2), \dots, id(y_m).$$

W przypadku wcześniejszego wywołania instrukcji **set migration checked** miejsce operatora $\frac{id(x)}{dodaj+}$ zajmie para operatorów $\frac{id(x)}{sprawdź}$, $\frac{id(x)}{dodaj+}$ o następującej postaci:

$$\frac{id(x)}{\text{sprawdź}} id(y_1), id(y_2), \dots, id(y_m), \quad \frac{id(x)}{\text{dodaj}+} id(y_1), id(y_2), \dots, id(y_m).$$

Drugi operator zostanie wykonany tylko w przypadku zwróconej przez pierwszy operator wartości logicznej false.

5.8.2. Translacja instrukcji usuwania obiektów

Także teraz wyróżniamy dwie sytuacje:

1. Usunięcie obiektu:

Instrukcji **delete** $id(x)$ odpowiada operator usunięcia obiektu x - operator 12 z listy podstawowych operatorów zmian.

2. Usunięcie obiektu ze zbioru podobiektów innego obiektu lub obiektów:

Instrukcji **delete** $id(x)$ **from** $id(x_1), id(x_2), \dots, id(x_n)$ odpowiada operator migracji obiektów $\frac{id(x)}{\text{usuń}}$ o następującej postaci:

$$id(x_1), id(x_2), \dots, id(x_n) \frac{id(x)}{\text{usuń}}.$$

W przypadku wcześniejszego wywołania instrukcji **set migration checked** miejsce operatora $\frac{id(x)}{\text{usuń}}$ zajmie para operatorów $\frac{id(x)}{\text{sprawdź}}, \frac{id(x)}{\text{usuń}}$ o następującej postaci:

$$\frac{id(x)}{\text{sprawdź}} id(x_1), id(x_2), \dots, id(x_n), \quad id(x_1), id(x_2), \dots, id(x_n) \frac{id(x)}{\text{usuń}}.$$

Drugi operator zostanie wykonany tylko w przypadku zwróconej przez pierwszy operator wartości logicznej true.

5.8.3. Translacja instrukcji aktualizacji obiektów

Przy aktualizacji obiektów mamy trzy sytuacje:

1. Aktualizacja wartości lub dodanie atrybutu i jego wartości:

W przypadku instrukcji **update** $id(x)$ **set** $\langle a_1:v_1, a_2:v_2, \dots, a_n:v_n \rangle$ dla każdego atrybutu a_i wykonujemy:

- a) w przypadku istnienia atrybutu a_i wywołujemy metodę $set(a_i)$,
- b) w przeciwnym przypadku dodajemy atrybut (lokalny) a_i do obiektu x - operator 1 z listy podstawowych operatorów zmian, a następnie wywołujemy metodę $set(a_i)$.

2. Usunięcie atrybutu z obiektu:

W przypadku instrukcji **update** $id(x)$ **del** $a_1, a_2, \dots, a_n$ dla każdego atrybutu a_i wykonujemy:

- a) w przypadku atrybutu lokalnego a_i wywołujemy operator 2 z listy podstawowych operatorów zmian,

- b) w przeciwnym przypadku wywołujemy operator 4 z listy podstawowych operatorów zmian dla odpowiedniego nadobiektu.

3. Aktualizacja związku obiekt-podobiekt:

Instrukcji **update** $\text{id}(x)$ **from** $\text{id}(x_1), \text{id}(x_2), \dots, \text{id}(x_n)$ **into** $\text{id}(y_1), \text{id}(y_2), \dots, \text{id}(y_m)$ odpowiada

operator migracji obiektów $\frac{\text{id}(x)}{\text{przenieś} +}$ o następującej postaci:

$$\text{id}(x_1), \text{id}(x_2), \dots, \text{id}(x_n) \frac{\text{id}(x)}{\text{przenieś} +} \text{id}(y_1), \text{id}(y_2), \dots, \text{id}(y_m).$$

W przypadku wcześniejszego wywołania instrukcji **set migration checked** miejsce operatora

$\frac{\text{id}(x)}{\text{przenieś} +}$ zajmie trójka operatorów $\frac{\text{id}(x)}{\text{sprawdź}}$, $\frac{\text{id}(x)}{\text{sprawdź}}$, $\frac{\text{id}(x)}{\text{przenieś} +}$ o następującej postaci:

$$\frac{\text{id}(x)}{\text{sprawdź}} \text{id}(x_1), \text{id}(x_2), \dots, \text{id}(x_n), \frac{\text{id}(x)}{\text{sprawdź}} \text{id}(y_1), \text{id}(y_2), \dots, \text{id}(y_m),$$

$$\text{id}(x_1), \text{id}(x_2), \dots, \text{id}(x_n) \frac{\text{id}(x)}{\text{przenieś} +} \text{id}(y_1), \text{id}(y_2), \dots, \text{id}(y_m).$$

Drugi operator zostanie wykonany tylko w przypadku zwróconej przez pierwszy operator wartości logicznej false, natomiast trzeci zostanie wykonany, gdy drugi zwróci wartość logiczną true.

5.9. Wnioski

Język MOW obejmuje operacje manipulowania obiektami umożliwiające realizację instrukcji wyszukiwania, wprowadzania, usuwania i aktualizacji obiektów. Dodatkowo wprowadzono operatory mnogościowe i agregujące jako ważne rozszerzenie klasycznej instrukcji wyszukiwania. Instrukcje języka MOW odzwierciedlają strukturę obiektu wieloklasowego i umożliwiają zarządzanie takimi obiektami. Syntaktyka podstawowych instrukcji języka MOW znajduje się w Załączniku.

Instrukcja wyszukiwania została zdefiniowana jedynie w podstawowej formie. Nie uwzględniono w niej fraz *group by*, *having* i *order by*. Frazy te, z punktu widzenia konstrukcji językowych nie wnoszą nic dodatkowego [Ull00].

Instrukcje wprowadzania, usuwania i aktualizacji obiektów oprócz standardowych możliwości języka SQL 92 realizują migrację obiektów zgodną z opisem z rozdziału trzeciego.

6. Aspekty implementacyjne modelu

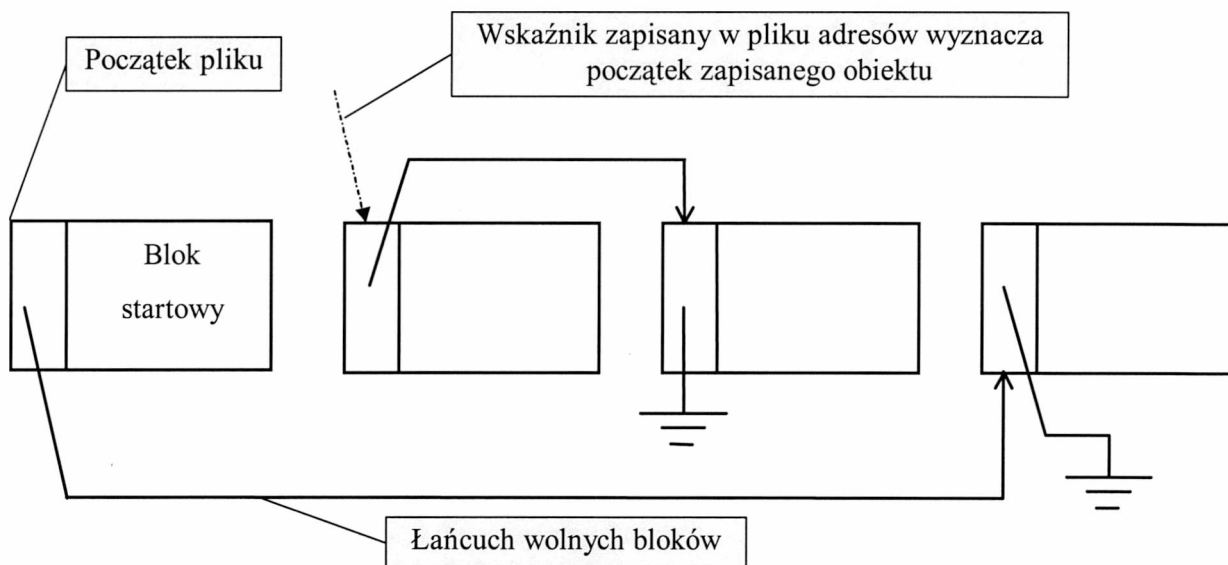
W niniejszej pracy przedstawiono i przeanalizowano aktualny stan badań w zakresie obiektowych modeli danych. Na tej podstawie zaproponowano zmiany w podstawowym modelu obiektowym. Najważniejszą cechą modelu stała się dynamika obiektów reprezentowana przez obiekt wieloklasowy oraz migrację obiektów.

W rozdziale zostaną przedstawione rozwiązania implementacyjne dla modelu obiektów wieloklasowych. Można je wykorzystać także w innych obiektowych bazach danych.

6.1. Reprezentacja fizyczna obiektu wieloklasowego i bazy danych

Jako podstawę rozważań implementacyjnych przyjmiemy system OB OK [Kro95c, Pia96]. System ten spełnia przyjęte w pracy założenia modelowe, chociaż nie występuje w nim wieloklasowość oraz homogeniczność obiektu i klasy.

Obiekty są przechowywane w pliku danych, natomiast dostęp do nich zapewnia plik adresów. Plik danych przechowujący informacje o obiektach wewnętrznie stanowi ciąg kolejnych łańcuchów bloków. Rozmiar bloku w bajtach jest zapisany w pierwszym bloku pliku, tzw. bloku startowym. Zalecany rozmiar bloku to wielokrotność rozmiaru sektora fizycznego dysku.



Rysunek 6.1. Struktura fizyczna pliku danych

W każdym bloku pierwsze 4 bajty wykorzystujemy do przechowania wskaźnika na następny blok w danym łańcuchu. Każdy łańcuch jest zakończony blokiem, którego wskaźnik jest ustawiony na wartość Null.

Pierwszy blok, zaczynający się od zerowej pozycji w pliku, jest blokiem startowym, wykorzystywanym do przechowywania danych konfiguracyjnych, np. wielkość bloku, liczba wolnych bloków, liczba bloków z danymi. Blok startowy jest tworzony w momencie tworzenia bazy danych. Wtedy jego wskaźnik jest ustawiany na wartość Null. Oznacza to, że łańcuch wolnych bloków jest pusty i nowy blok należy zapamiętać na końcu pliku danych.

Podczas działania systemu wskaźnik bloku startowego zawsze wskazuje na łańcuch wolnych bloków, zwolnionych na skutek wykonania wcześniejszych operacji – usuwania lub modyfikacji obiektu. Jeżeli jego wartość wynosi Null oznacza to, iż kolejne bloki danych muszą zostać dopisane na koniec pliku danych.

Każdy obiekt danych jest przechowywany w pliku danych w postaci łańcucha bloków. Początek łańcucha jest wyznaczony przez wskaźnik zapisany w pliku adresów pod pozycją odpowiadającą identyfikatorowi danego obiektu. Sposób generowania identyfikatora polega na pobraniu kolejnej liczby naturalnej względem ostatnio użytego identyfikatora. Taki system zapewnia niepowtarzalność identyfikatorów obiektów. Z każdym identyfikatorem wiążemy 4-bajtowy adres początku obiektu w pliku danych.

Obiekt został podzielony na dwie części: opis i dane. W pierwszej części występują elementy definicyjne. Każdy element jest opisany przez długość elementu oraz offset początku w części dane. Kolejno więc: identyfikator obiektu $id(x)$, zbiór identyfikatorów podobiektów $Sub(x)$, zbiór identyfikatorów nadobiektów $Sup(x)$, zbiór atrybutów dla podobiektów $A_{Sub}(x)$, zbiór atrybutów obiektu $A(x)$, zbiór wartości dla atrybutów obiektu $V(x)$ oraz zbiór metod $M(x)$. Po części opisowej następują dane. Struktura zapisu wymaga użycia separatora dla rozróżnienia elementów obiektu oraz separatora dla rozróżnienia wielu wartości dla jednego atrybutu. W przypadku zagnieżdżenia wartości definiujemy obiekty zagnieżdżone i wykorzystamy ich identyfikatory w wartości obiektu podstawowego.

Plik adresów składa się z zapisanych jeden za drugim, dla każdego obiektu, adresów wskazujących na początek łańcucha bloków w pliku danych. Znając identyfikator obiektu możemy odczytać adres obiektu. Wystarczy pomnożyć wartość identyfikatora przez wielkość adresu obiektu.

6.2. Implementacja zapytań języka MOW

Rozważmy jeszcze problemy związane z efektywnym przetwarzaniem zapytań języka MOW. Zapytanie jest wyrażeniem opisującym informacje, które użytkownik zamierza uzyskać z bazy danych. Takie wyrażenie jest zwykle tłumaczone przez optymalizator zapytań w ciąg operacji algebraicznych, które zwracają oczekiwaną informację.

Zdolność do automatycznej optymalizacji zapytań jest jedną z podstawowych cech systemów relacyjnych baz danych. Optymalizatory relacyjnych zapytań wykorzystują semantykę modelu. Podstawowa strategia dla relacyjnych zapytań skupia się na zapytaniach dotyczących operatorów selekcji, projekcji i złączenia [Dat00].

Kolejno zostaną zaprezentowane problemy, które pojawiają się przy wieloklasowości obiektu oraz wprowadzeniu związku obiekt-podobiekt. Każdy problem zostanie zilustrowany przykładem.

6.2.1. Obiekty wieloklasowe

Obiektowy model powinien umożliwiać optymalizację obiektów-klas i obiektów-instancji. Ponadto powinien umieć stosować optymalizację dla związków typu obiekt-podobiekt. Powinien mieć możliwość stosowania transformacji.

Na przykład, rozważmy zapytanie: "Znajdź uczniów klasy 2b urodzonych w 1990 roku, którzy są członkami koła matematycznego".

Pierwszy nasuwający się sposób realizacji takiego zapytania jest następujący:

$$\sigma(\text{Sub}(\text{Koło_mat}), v \in \sigma(\text{Sub}(\text{Klasa_2b}), \text{Sub}(\text{Klasa_2b}).\text{Rok_ur} = 1990))$$

Wewnętrzna operacja selekcji tworzy zbiór wszystkich uczniów klasy 2b, którzy urodzili się w 1990 roku. Druga operacja selekcji wybiera ze wszystkich członków koła matematycznego tych, którzy należą do zbioru obiektów Uczniowie.

Można uprościć ostatnią operację selekcji w następujący sposób:

$$\sigma(\sigma(\text{Sub}(\text{Klasa_2b}), \text{Sub}(\text{Klasa_2b}).\text{Rok_ur} = 1990), v \in \text{Sub}(\text{Koło_mat}))$$

To uproszczenie, wynika z faktu, że w naszym modelu warunek określający zawieranie się elementów w zbiorze jest przemienny ze zbiorem specyfikowanym jako pierwszy argument operatora selekcji. Obiekty mogą być zamieniane z predykatem „element należy do zbioru”. Relacje obiekt-podobiekt też są relacjami zawierania się zbiorów. Na przykład, zbiór Dyrekcja jest podzbiorem zbioru Nauczyciele oraz obiekty: Klasa_2a i Klasa_2b są elementami zbioru, jakim jest Szkoła_11.

Analizowane zapytanie można także sformułować za pomocą operatora przecięcia oraz operatora selekcji.

$$\sigma(\text{Sub}(\text{Klasa_2b}) \cap \text{Sub}(\text{Koło_mat}), \text{Sub}(\text{Klasa_2b}).\text{Rok_ur} = 1990)$$

Jak wynika z powyższego przykładu, znalezienie uczniów z koła matematycznego można uprościć do przecięcia odpowiednich zbiorów obiektów i dopiero na jego wyniku dokonać selekcji zgodnej z zadaniem warunkiem.

6.2.2. Związek obiekt-podobiekt

Informacja pochodząca ze związku obiekt-podobiekt może również wpływać na użyteczność transformacji. Na przykład, wykonanie zapytania dla sumy dwóch zbiorów nie jest, w szczególnym przypadku, równoważne wykonaniu zapytań dla każdego zbioru z osobna i następnie utworzenia ich sumy. Formalnie: operator $(S_1, p) \cup \text{operator}(S_2, p) \neq \text{operator}((S_1 \cup S_2), p)$, gdyż predykat p może nie być zdefiniowany dla $S_1 \cup S_2$.

Dla przykładu, rozważmy zapytanie, które wyszukuje lata urodzin wszystkich uczniów i nauczycieli. Zapytanie to może być wyrażone następująco:

$$(\tau(\text{Sub}(\text{Uczeń}), \text{Rok_ur})) \cup (\tau(\text{Sub}(\text{Nauczyciel}), \text{Rok_ur}))$$

ale nie w ten sposób:

$$\tau((\text{Sub}(\text{Uczeń}) \cup \text{Sub}(\text{Nauczyciel})), \text{Rok_ur}),$$

gdyż atrybut Rok_ur nie jest zdefiniowany dla obiektu Osoba. Jednakże, jeśli część wspólna wynikowego zbioru (to znaczy uczniów i nauczycieli) jest duża, to ostatnie zapytanie może

być prawidłowe i efektywne. Wykonując asocjację po operacji sumy uniknęłoby się dwukrotnego stosowania operatora asocjacji na obiektach części wspólnej.

Powyższe przykłady ilustrują fakt, że wiedza na temat związków: obiekt-podobiekt i dziedziczenia może wpływać w modelu obiektowym na transformacje zapytań. Optymalizator musi uwzględniać rozszerzalność, która jest nieodzowna w konstrukcji danych obiektowych. Dodanie nowych obiektów może powodować zmiany w procesie optymalizacji zapytań. Optymalizator powinien umieć stosować odpowiednie przekształcenia dla znalezienia lepszych rozwiązań, ale musi zarazem być zdolny do określania, kiedy takie transformacje są do przyjęcia. To może dotyczyć umiejętności sprawdzenia wyniku zapytania i porównania go z wynikiem po przeprowadzonym przekształceniu.

6.3. Wnioski

Na podstawie przeprowadzonej dyskusji na temat optymalizacji można stwierdzić, że optymalizacja obiektowych zapytań jest złożonym problemem i wymaga prowadzenia dalszych badań. Zaproponowane przykładowe transformacje zapytań nie powinny zadowolić konstruktora optymalizatora, który poza statycznymi transformacjami powinien reagować także na zmiany powiązań obiektów i na implementacje metod. Strategia postępowania w takim przypadku, podobnie jak przy problemie adaptacyjnego grupowania obiektów [Klo98], winna opierać się na odpowiedniej funkcji kosztu.

Wyczerpujące informacje na temat sposobu postępowania w przypadku optymalizacji obiektowego języka zapytań znajdziemy w pracach [Plo00, Mit93].

7. Podsumowanie

Systemy baz danych funkcjonują zwykle wiele lat. W tym czasie zachodzą zmiany. Najpierw dotyczy to kilku obiektów, potem kilkunastu. Wreszcie biorąc pod uwagę licznosc zmienionych obiektów i tendencje tych zmian decydujemy się na aktualizację bazy danych, i jej zawartości. Najczęściej aktualizacja dotyczy tylko pewnej części zakresu zaobserwowanych zmian. Do tego czasu, jeśli nie wprowadzamy zmian na bieżąco nie możemy korzystać z systemu w optymalny sposób. Na tej podstawie zaproponowano zmiany w podstawowym modelu obiektowym.

Najważniejszą cechą modelu stała się dynamika obiektów reprezentowana przez obiekt wieloklasowy oraz migrację obiektów. Dynamika danych wymaga wprowadzenia nowych pojęć w modelu danych, które uwzględniają specyfikę polegającą na potrzebie ciągłych zmian w strukturze i w zachowaniu obiektów. Wymagania te wynikają bezpośrednio z dynamicznego charakteru rzeczywistego świata.

Przedstawione w pracy teoretyczne rozważania nad wieloklasowością oraz związaną z nią migracją obiektów potwierdziły możliwość natychmiastowego wprowadzania zmian. Tym samym obiekty wieloklasowe spełniają wymagania narzucone przez zmieniające się obiekty.

Model obiektów wieloklasowych jest ogólniejszy niż zaproponowane dotychczas w literaturze obiektowe modele danych. Obiekt będący fundamentalnym pojęciem tego modelu umożliwia wprowadzanie zmian zachodzących w obiektach, a związek obiekt-podobiekt zmian zachodzących między obiektami. Spójność obiektów wynika z ich przynależności do jednego schematu obiektów i spełnianiu zdefiniowanych ograniczeń.

Zaproponowana definicja obiektu, w której podkreśla się dynamiczny charakter obiektu a nie tylko jego strukturę, rozszerza podstawowy obiektowy model danych. Tradycyjna statyczna więź pomiędzy klasami i obiektami została wymieniona na związek dynamiczny obiekt-podobiekt. Pojęcia obiektu i klasy zostały ujednolicone.

Problem zmian został szczegółowo omówiony dla obiektów wieloklasowych. Ograniczenia definiowane w postaci obiektów decydują o populacji i jakości danych w bazie danych. Można więc ujednolicić system migracji obiektów z systemem przynależności obiektów do klas, np. wzajemnym wykluczeniem obiektów lub ich koniecznym współwystępowaniem. Powstały w ten sposób zintegrowany system w pełni oddaje zmienność rzeczywistości. Dzięki tym własnościom model obiektów wieloklasowych umożliwia pełną kontrolę nad zmieniającymi się czy zmienianymi obiektami.

Baza obiektów wieloklasowych umożliwia wprowadzanie zmian na bieżąco, a więc permanentną ewolucję obiektu. Dzięki temu jedyną przyczyną kosztownej modernizacji systemu z bazą danych może być zesterzenie się sprzętu czy oprogramowania. Aktualne wyniki prac nad automatycznym generowaniem interfejsu użytkownika na podstawie specyfikacji danych obiektowych [Eng96], czyli także po wprowadzeniu zmian w obiektach wieloklasowych, już dzisiaj są bardzo optymistyczne. Wykorzystanie możliwości, jakie daje obiekt wieloklasowy w połączeniu z automatycznym generowaniem obiektowych systemów

może zaspokoić zapotrzebowanie użytkowników na konieczność modyfikacji systemów ze względu na dynamiczny charakter danych rzeczywistych.

Najważniejszymi wynikami pracy są:

1. Dokonanie szerokiego przeglądu zagadnień dynamiki danych w obiektowych bazach danych.
2. Opracowanie podstawowych elementów obiektowego modelu danych spełniającego wymagania dynamiki obiektów. Zawiera on nowe pojęcia służące do reprezentacji semantyki procesów dynamicznych zachodzących w rzeczywistym świecie. Podstawą tego modelu jest obiekt wieloklasowy oraz związek obiekt-podobiekt reprezentujący strukturę powiązań między obiektami. Schemat obiektów, który jest rozszerzeniem klasycznej hierarchii klas wynikającej ze schematu bazy danych, obejmuje związek dziedziczenia oraz związek przynależności obiektu do klasy.
3. Zdefiniowanie operatorów migracji obiektów wieloklasowych: dodaj, usuń i przenieś oraz rozszerzonych operatorów dodaj+ i przenieś+ uwzględniających pochodzenie atrybutów.
4. Opracowanie modelu implementacyjnego ograniczeń dla obiektów, w tym opracowanie obiektu-ograniczenia.
5. Zaprojektowanie strukturalnego języka manipulacji danymi MOW, w którym względem klasycznego podejścia rozszerzono syntaktykę i semantykę instrukcji: insert, update i delete.
6. Określenie fizycznej struktury obiektu oraz pliku danych odpowiednich do stosowania w bazie obiektów wieloklasowych.

Uzyskane w pracy wyniki oraz przeprowadzone analizy dotychczasowych osiągnięć pozwalają na sformułowanie następujących wniosków:

1. Dotychczas stosowane struktury obiektów, bazujące na przynależności obiektu do jednej, niezmienniej klasy, w praktyce nie nadają się do implementacji w szybko zmieniającej się rzeczywistości.
2. Unifikacja pojęć obiektu, klasy i ograniczeń jest możliwa. Podejście takie upraszcza język manipulowania danymi w obiektowej bazie danych.
3. Istnieją możliwości realizacji wieloklasowości obiektów, co rozwiązuje problem wielokrotnego dziedziczenia oraz migracji obiektów. Odpowiednią do tego strukturą jest obiekt wieloklasowy. Aby możliwe było stosowanie wieloklasowości należy uwzględnić powiązania z podobiektami jak i nadobiektami.
4. Język manipulowania obiektami wieloklasowymi może być rozszerzeniem standardowego języka SQL, dzięki czemu jest łatwy w użyciu. Jego implementacja w systemie, w miejsce żądań języka bazującego na SQL, nie wymaga wielu zmian. Instrukcje języka zachowały swoją pierwotną funkcję, a ponadto zyskały nowe możliwości.
5. Można oczekiwać zwiększenia efektywności działania bazy danych po zastosowaniu standardowych metod, takich jak: optymalizacja zapytań, indeksowanie obiektów oraz grupowanie danych.

7.1. Kierunki dalszych badań

Wprowadzenie możliwości modyfikacji obiektów “on-line” otwiera nowe kierunki badań w dziedzinie systemów informacyjnych. Bardzo obiecujące wydaje się wykorzystanie pojęcia obiektu wieloklasowego oraz operatorów migracji w systemach agenckich, współbieżnych oraz z autoryzacją dostępu.

W środowisku dynamicznym zmiany obiektów mogą doprowadzić do nieodpowiedniej reprezentacji obiektów-klas, np. niewielka liczba obiektów-klas w stosunku do olbrzymiej liczby obiektów-instancji, może oznaczać nieprawidłowy dobór atrybutów dla podobiektów. Jednym z możliwych rozwiązań tego problemu jest automatyczne badanie schematu obiektów, znajdowanie kandydatów dla obiektów-klas, ich weryfikacja i usuwanie ewentualnych konfliktów. Do tworzenia odpowiedniego schematu obiektów można by użyć metody klasyfikacji Galoisa [Haa98]. Utworzony w ten sposób schemat mógłby być porównany z istniejącym schematem. Skonstruowane w tym celu odpowiednie macierze mogą być podstawą algorytmu porównującego. Prace w tym kierunku, wprawdzie dla klasycznego modelu, są prowadzone na uniwersytecie w Montrealu. Modyfikacja algorytmów uwzględniająca wieloklasowość oraz homogeniczność obiektów i klas jest interesującym tematem dalszych badań.

Przy ocenie przydatności języka MOW nie można nie wspomnieć o indeksowaniu jako jednego z najefektywniejszych mechanizmów optymalizacji zapytań. Problemem indeksowania obiektów zajmuje się wielu autorów [Ber94, Ber98a, Goc98]. Do najczęściej opisywanych metod należą: indeksy wielokrotne, indeksy złączeń, indeksy zagnieżdżone, indeksy ścieżek, pliki sygnatur, indeksy dziedziczenia, H-drzewa i drzewa częściowego porządku. Do tej pory, nigdzie *explicite* nie zaproponowano takiej struktury indeksowania, która zezwalałaby na występowanie obiektów w wielu klasach, czyli utworzenia indeksu dla związku obiekt-podobiekt. Można jednakże pokusić się o stwierdzenie, że modyfikacja wymienionych struktur może zapewnić efektywne wyszukiwanie także dla obiektów wieloklasowych. Problemem, który należałoby rozwiązać jest opracowanie warunków aktualizacji indeksu. Kiedy należy dokonać aktualizacji i w jaki sposób? Może to być kolejnym obszarem badań, zarówno teoretycznym jak i praktycznym, polegającym na przeprowadzeniu eksperymentów.

Badając dynamikę obiektów natrafiamy na problem automatycznej adaptacji zmian w bazie danych, która samoczynnie dostosowuje swoje charakterystyki wewnętrzne do środowiska użytkowego. Adaptacja musi być przeprowadzona w taki sposób, by uzyskać przy tym odpowiednio wysoki poziom efektywności działania. Przykładem takiego mechanizmu jest grupowanie obiektów, czyli sposób rozmieszczenia obiektów na stronach pamięci zewnętrznej [Klo98]. Aby skonstruować taki mechanizm dla obiektów wieloklasowych można by zdefiniować odległość dla związku obiekt-podobiekt a przez to odległość między obiektami wieloklasowymi. W tego typu metodzie badawczej pożądanym uzupełnieniem pracy teoretycznej jest weryfikacja wyników na drodze eksperymentalnej. Ponadto wydaje się, że problem grupowania obiektów wieloklasowych stanowi dobry punkt wyjścia do dalszych badań nad implementacją mechanizmów adaptacyjnych w rzeczywistych bazach danych.

Na zakończenie warto wspomnieć, że wynikami osiągniętymi w rozprawie i kontynuacją prac nad bazą obiektów wieloklasowych są zainteresowani Zakład Mostów Instytutu Inżynierii Lądowej Politechniki Wrocławskiej oraz Polskie Koleje Państwowe, z

którymi od paru lat jest prowadzona intensywna współpraca. Definicja obiektu wieloklasowego może być zaimplementowana w Systemie Zarządzania Mostami Kolejowymi SMOK bazującym na obiektowej bazie danych OB OK [Bie97, Bie01].

Bibliografia

- [Abi95] Abiteboul S., Hull R., Vianu V.: *Foundations of Databases*. Addison-Wesley Publishing Company 1995.
- [Ala89]* Alagic S.: *Object-Oriented Database Programming*. Berlin Heidelberg, Springer Verlag 1989.
- [Alb93] Albano A., Bergamini R., Ghelli G., Orsini R.: *An Object Data Model with Roles*. The Proceedings of the 19th International VLDB Conference, 1993, 39-51.
- [Alb95a] Albano A., Diotallevi M., Ghelli G.: *Extensible Objects for Database Evolution: Language Features and Implementation Issue*. Proceedings of the 5th Intl. Workshop on Database Programming Languages (DBPL), 1995.
- [Alb95b] Albano A., Ghelli G., Orsini R.: *Fibonacci: A programming language for object database*. VLDB Journal 1995, vol. 4, no. 3, 403-439.
- [Alb98]* Albano A., Antognoni G., Ghelli G.: *View Operations on Objects with Roles for a Tatically Typed Database Language: The Type Rules and the Operational Semantics*. Unpublished paper 1998, 1-57.
- [Alh93a] Alhajj R., Arkun M.: *An Object Algebra for Object-Oriented Database Systems*. Data Base 1993, 8, 13-22.
- [Alh93b] Alhajj R., Arkun M.: *Object-oriented query language facilitating construction of new objects*. Information and Software Technology 1993, 35, 519-529.
- [And92]* Andries M., Gemis M., Paredaens J., Thyssens I., Van den Bussche J.: *Concepts for graph-oriented object manipulation*. Advances in Database Technology - EDBT '92. Proceedings of the 3rd International Conference on Extending Database Technology (Vienna 1992). Berlin Heidelberg, Springer Verlag 1992, 21-38.
- [Ang93]* Ang James S. K., Conrath D. W.: *The ITHACA Office Object Model: Modeling and Implementation*. Data Base 1993, 11, 5-14.
- [Atk89] Atkinson M., Bancilhon F., DeWitt D., Dittrich K., Maier D., Zdonik S.: *The Object-Oriented Database System Manifesto (A Political Pamphlet)*. Proceedings First Conference Deductive and Object-Oriented Databases. Kyoto, Japan, 1989.
- [Atw93]* Atwood T.: *The Object DBMS Standard*. Object Magazine 1993, 3, 37-44.
- [Bal90] Balfour A., Britton C.: *InfoExec: object-oriented/semantic database implementation*. Information and Software Technology 1990, 32, 290-296.

- [Ban93]* Bancilhon F.: *Object Database Systems: Functional Architecture. Object Technologies for Advanced Software*. The Proceedings of First JSSST International Symposium, Kanazawa, Japan, November 4-6. Berlin Heidelberg, Springer Verlag 1993, LNCS 742, 163-175.
- [Bar96]* Bardou D., Dony Ch.: *Split Objects: a Disciplined Use of Delegation within Objects*. The Proceedings of Object-Oriented programming Systems, Languages, and Applications (OOPSLA). ACM 1996, 122-137.
- [Bas94] Bassiliades N., Vlahavas I.: *Modeling Constraints with Exceptions in Object-Oriented Databases*. Entity-Relationship Approach - ER '94. The Proceedings of International Conference, ER '94, Manchester, UK December 13-16. Berlin Heidelberg, Springer Verlag 1994, LNCS 881, 189-204.
- [Bee88]* Beech D.: *Intensional Concepts in an Object Database Model*. The Proceedings of Object-Oriented programming Systems, Languages, and Applications (OOPSLA). ACM 1988, 164-175.
- [Bee91]* Beeri C., Milo T.: *A Model for Active Object-Oriented Databases*. The Proceedings of the 17th International Conference on Very Large Data Bases. Barcelona, September 1991, 337-349.
- [Bee93] Beeri C., Kornatzky Y.: *Algebraic optimization of object-oriented query languages*. Theoretical Computer Science 1993, 116, 59-94.
- [Bee94] Beeri C.: *Query Languages for Models with Object-Oriented Features*. Advances in Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1994, 47-71.
- [Ber89]* Bertino E., Kim W.: *Indexing Techniques for Queries on nested Objects*. IEEE Transactions on Knowledge and Data Engineering 1989, June, 196-214.
- [Ber92] Bertino E.: *A View Mechanism for Object-Oriented Databases*. Advances in Database Technology - EDBT '92. Proceedings of the 3rd International Conference on Extending Database Technology (Vienna 1992). Berlin Heidelberg, Springer Verlag 1992, 136-151.
- [Ber94] Bertino E.: *Index Configuration in Object-Oriented Databases*. VLDB Journal 1994, 3, 355-399.
- [Ber97]* Bertino E., Foscoli P.: *On Modeling Cost Functions for Object-Oriented Databases*. IEEE Transactions on Knowledge and Data Engineering 1997, vol. 9, no 3, 500-507.
- [Ber98a] Bertino E., Catania B., Chiesa L.: *Definition and analysis of index organizations for object-oriented database systems*. Information Systems 1998, vol. 23, no 2, 65-108.
- [Ber98b] Bertino E., Guerrini G., Rusca J.: *Object Evolution in Object Databases*. Theoretical Approaches to Dynamic Worlds: from the Frame Problem to the Theory and Practice of Knowledge 1998.
- [Bey92]* Beynon-Davies P.: *Entity models to object models: object-oriented*

- analysis and database design*. Information and Software Technology 1992, 34, 255-262.
- [Bha91] Bhalla N.: *Object-oriented data models: a perspective and comparative review*. Journal of Information Science 1991, 17, 145-160.
- [Bha93]* Bhalla N., Balasundaram S.: *Operations and queries in object-oriented databases supporting complex objects*. Information and Software Technology 1993, 35, 54-62.
- [Bie97] Bień J., Król D., Rawa P., Rewiński S.: *Komputerowa ewidencja obiektów inżynierskich*. Seria: System Zarządzania Mostami Kolejowymi SMOK. Tom I. Warszawa 1997.
- [Bie01] Bień J., Król D., Rawa P., Zwolski J.: *Komputerowe wspomaganie utrzymania obiektów inżynierskich*. Seria: System Zarządzania Mostami Kolejowymi SMOK. Tom V. Wrocław 2001 (w przygotowaniu).
- [Bjo89] Björnerstedt A., Hulten Ch.: *Version Control in an Object-Oriented Architecture*. Object-Oriented Concepts, Databases, and Applications. ACM Press, Addison-Wesley 1989, 451-485.
- [Bla93] Blakeley J.: *OQL[C++]: Extending C++ with an Object Query Capability*. In Modern Database Systems. ACM Press, Addison-Wesley 1993, 69-88.
- [Bra92] Bratsberg S.: *Unified Class Evolution by Object-Oriented Views*. Entity-Relationship Approach - ER '92. Berlin Heidelberg, Springer Verlag 1992, 433--439.
- [Bre89]* Bretl R., Maier D. et al.: *The GemStone Data Management System*. Object-Oriented Concepts, Databases, and Applications. ACM Press, Addison-Wesley 1989, 283-308.
- [Bro91] Brown A. W.: *Object-oriented databases*. Applications in Software Engineering. McGraw-Hill Book Company 1991.
- [Bry94]* Bryant T., Evans A.: *OO oversold*. Information and Software Technology 1994, 36, 35-42.
- [Buc94]* Buchmann A.: *Active Object Systems*. Advances in Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1994, 201-224.
- [Bus92]* Van den Bussche J., Van Gucht D., Andries M., Gyssens M.: *On the Completeness of Object-Creating Query Languages*. The Proceedings 33rd Annual Symposium on Foundations of Computer Science 1992, 372-379.
- [Bus93] Van den Bussche, Heuer A.: *Using SQL with Object-Oriented Databases*. Information Systems 1993, 18, 461-487.
- [Cos94]* Costa J., Sernadas A., Sernadas C.: *Object inheritance beyond subtyping*. Acta Informatica 1994, 31, 5-26.
- [Cro92]* Croft W. B., Turtle H.: *Retrieval of Complex Objects*. Advances in

- Database Technology - EDBT '92. Proceedings of the 3rd International Conference on Extending Database Technology (Vienna 1992). Berlin Heidelberg, Springer Verlag 1992, 217-229.
- [Cro93]* Crowe M. K.: *Object systems over relational databases*. Information and Software Technology 1993, 35, 449-461.
- [Dan92] Daniłowicz Cz.: *Modele systemów wyszukiwania informacji uwzględniające preferencje użytkowników końcowych (rozprawa habilitacyjna)*. Prace Naukowe BG i OINT Politechniki Wrocławskiej 1992, 6 (3).
- [Dar92]* Dar S., Gehani N.H., Jagadish H.V.: *CQL++: A SQL for the Ode Object-Oriented DBMS*. Advances in Database Technology - EDBT '92. Proceedings of the 3rd International Conference on Extending Database Technology (Vienna 1992). Berlin Heidelberg, Springer Verlag 1992, 201-216.
- [Dat00] Date C.J.: *Wprowadzenie do systemów baz danych*. Wydawnictwa Naukowo-Techniczne, Warszawa 2000.
- [Del95] Delobel C., Lecluse Ch., Richard P.: *Databases: From Relational to Object-Oriented Systems*. International Thomson Publishing 1995.
- [Dij93] Dijkstra J.: *On complex objects and versioning in complex environments*. Entity-Relationship Approach - ER '93 The Proceedings of International Conference, ER '93, Arlington, USA, December 15-17. Berlin Heidelberg, Springer Verlag 1993, LNCS 823, 13-23.
- [Dit88]* Dittrich K. R.: *Advances in Object-Oriented Database Systems*. Proceedings of the 2nd International Workshop on Object-Oriented Database Systems (Bad Muenster 1988). Berlin Heidelberg, Springer Verlag 1988.
- [Dit91]* Dittrich K. R.: *Object-Oriented Database Systems: The Notion and the Issues*. On Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1991, 3-10.
- [Dit94]* Dittrich K. R.: *Object-Oriented Data Model Concepts*. Advances in Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1994, 29-45.
- [Dog94] Dogac A., Ozsu T., Biliris A., Sellis T.: *Advances in Object-Oriented Database Systems*. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1994.
- [Eng96] Engelson V., Fritzson D., Fritzson P.: *Automatic Generation of User Interfaces from Data Structure Specifications and Object-Oriented Application Models*. ECOOP '96 - Object-Oriented Programming. Berlin Heidelberg, Springer Verlag 1996, LNCS 1098, 114-141.
- [Fig96] Figura D.: *Obiektowe bazy danych*. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1996.
- [Fis89]* Fishman D.H., Beech D. et al.: *Overview of the Iris DBMS*. Object-

- Oriented Concepts, Databases, and Applications. ACM Press, Addison-Wesley 1989, 219-250.
- [Fow97]* Fowler M.: *Dealing with Roles*. <http://www2.awl.com/cseng/titles/0-201-89542-0/apsupp/roles2-1.html>. 1997, 1-18.
- [Gem93]* Gemis M., Paredaens J.: *An Object-Oriented Pattern Matching Language*. Object Technologies for Advanced Software. The Proceedings of First JSSST International Symposium, Kanazawa, Japan, November 4-6. Berlin Heidelberg, Springer Verlag 1993, LNCS 742, 339-355.
- [Goc98] Goczyła K.: *Metody indeksowania w obiektowych bazach danych (rozprawa habilitacyjna)*. Zeszyty Naukowe Politechniki Gdańskiej 1998, 565.
- [Gor93] Goralwalla I., Ozsu T.: *Temporal Extensions to a Uniform Behavioral Object Model*. Entity-Relationship Approach - ER '93 The Proceedings of International Conference, ER '93, Arlington, USA, December 15-17. Berlin Heidelberg, Springer Verlag 1993, LNCS 823, 110-121.
- [Got96] Gottlob G., Schrefl M., Roeck B.: *Extending Object-Oriented Systems with Roles*. ACM Transactions on Information Systems 1996, 14 (April), 268-296.
- [Gou91] Goutas S., Soupos P., Christodoulakis D.: *Formalization of object-oriented database model with rules*. Information and Software Technology 1991, 33, 741-757.
- [Gri82] van Griethuysen (ed.): *Concepts and terminology for the conceptual schema and the information base, computers and information processing*. ISO/TC97/SC5/WG3 International Organization for Standardization, Geneva 1982.
- [Gru91]* Grumbach S., Vianu V.: *Tractable Query Languages for Complex Object Databases*. The proceedings of the Tenth ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database System, New York 1991, 315-327.
- [Gru96] Gruber M.: *SQL*. Wydawnictwo HELION, Gliwice 1996.
- [Gud97]* Gudes E.: *A uniform indexing scheme for object-oriented databases*. Information Systems 1997, vol. 22, no 4, 199-221.
- [Gue98a]* Guerrini G.: *An Active and Deductive Object-Oriented Data Model (rozprawa doktorska)*. Universita di Genova 1998.
- [Gue98b]* Guerrini G., Berino E., Bal R.: *A Formal Definition of the Chimera Object-Oriented Data Model*. Journal of Intelligent Information Systems 1998, 11, 5-40.
- [Gys94]* Gyssens M., Paredaens J., Van den Busch J., Van Gucht D.: *A Graph-Oriented Database Model*. IEEE Transactions on Knowledge and Data Engineering 1994, August, 572-586.
- [Haa98] Haav H., Matskin M.: *Discovery of Object-Oriented Schema and Schema Conflicts*. W [Lit98], 154-156.

- [Hal92]* Halper M., Geller J., Perl Y.: *"Part" Relations for Object-Oriented Databases*. Entity-Relationship Approach - ER '92. Berlin Heidelberg, Springer Verlag 1992, 406-422.
- [Ham95]* Hamza H., Benadjaoud G.: *An object algebra based on relationships between objects*. The Proceedings of the 10th ISCIS 1995, 305-312.
- [Han99]* Han J., Xie Z., Fu Y.: *Join Index Hierarchy: An Index Structure for Efficient Navigation in Object-Oriented Databases*. IEEE Transactions on Knowledge and Data Engineering 1999, vol. 11, no. 2, 321-337.
- [Her92]* Herzig R., Gogola M.: *Transforming Conceptual Data Models into an Object Model*. Entity-Relationship Approach - ER '92. Berlin Heidelberg, Springer Verlag 1992, 280-298.
- [Her94]* Herzig R., Gogola M.: *A SQL-like Query Calculus for Object-Oriented Database Systems*. Object-Oriented Methodologies and Systems. The Proceedings of International Symposium, ISOOMS '94, Palermo, Italy, September 21-22. Berlin Heidelberg, Springer Verlag 1994, LNCS 858, 20-39.
- [Hor94]* Horng J.-T., Liu B.-J.: *Some Aspects of Operations in an Object-Oriented Data Base Based on Graphs*. Journal Systems Software 1994, 24, 155-179.
- [Hwa94]* Hwang D.: *Function-Based Indexing for Object-Oriented Databases (rozprawa doktorska)*. 1994.
- [Igl84] Iglewski M., Madej J., Matwin S.: *Pascal. Język wzorcowy. Pascal 360*. Wydawnictwa Naukowo-Techniczne, Warszawa 1984.
- [Ish93]* Ishikawa H.: *Object-Oriented Database System. Design and Implementation for Advanced Applications*. Berlin Heidelberg, Springer Verlag 1993.
- [Jac91]* Jackson M. S.: *Tutorial on object-oriented databases*. Information and Software Technology 1991, 33, 4-12.
- [Jod98] Jodłowski A.: *Role w obiektowych bazach danych*. Seminarium IPI PAN. Warszawa 1998.
- [Jun96] Jungclaus R., Saake G., Hartmann T., Sernadas C.: *TROLL – A Language for Object-Oriented Specification of Information Systems*. ACM Transactions on Information Systems 1996, 14 (April), 175-211.
- [Kap98]* Kappel G, Retschitzegger W., Schwinger W.: *A Comparison of Role Mechanisms in Object-Oriented Modeling*. Unpublished paper 1998, 1-8.
- [Kem90]* Kemper A., Kilger C., Moerkotte G.: *Function Materialization in Object Bases*. The proceedings of the 1990 ACM SIGMOD Int'l. Conf. on Management of Data, USA 1990, 364-374.
- [Kho90]* Khoshafian S.: *Insight into object-oriented databases*. Information and Software Technology 1990, 32, 274-289.
- [Kim87]* Kim W., Ballou N., Banerjee J., Chou H., Garza J., Welk D.: *Features of the Orion Object-Oriented Database System*. The Proceedings of the 13th International VLDB Conference 1987, 319-329.

- [Kim89a]* Kim W., Kim K-CH., Dale A.: *Indexing Techniques for Object-Oriented Databases*. Object-Oriented Concepts, Databases, and Applications. ACM Press, Addison-Wesley 1989, 371-394.
- [Kim89b]* Kim W., Lochovsky F.: *Object-Oriented Concepts, Databases, and Applications*. ACM Press, Addison-Wesley 1989.
- [Kim91]* Kim S., Lee S.: *A Formalization of a Query language for Complex Objects*. The proceedings of 21st ACM Computer Science Conference 1991, 136-145.
- [Kim96] Kim W.: *Wprowadzenie do obiektowych baz danych*. Wydawnictwa Naukowo-Techniczne, Warszawa 1996.
- [Klo96] Klosow A.: *Testowanie procedur automatycznej modyfikacji schematu w obiektowej bazie danych*. Raport Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej SPR 45/96, 1996.
- [Klo98] Klosow A.: *Mechanizmy adaptacji w obiektowo zorientowanych bazach danych (rozprawa doktorska)*. Raport Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej PRE 49/98, 1998.
- [Klo00] Klosow A., Król D.: *Obiektowe systemy z bazami danych - zastosowania*. Informatyka 2000, 2, 16-23.
- [Kni98]* Kniesel G.: *Type-safe Delegation for Dynamic Component Adaptation*. ECOOP '98 - Componentt-Oriented Programming. 1998, 1-11.
- [Koł01] Kołaczek G.: *Model mechanizmu sterowania i zarządzania dostępem poprzez role w systemach informatycznych (rozprawa doktorska)*. Raport Instytutu Sterowania i Techniki Systemów Politechniki Wrocławskiej, 2001.
- [Kos93] Koszłajda T.: *Wieloporcyjne i wielowersyjne obiektowe bazy danych (rozprawa doktorska)*. Politechnika Poznańska, Instytut Informatyki, Poznań 1993.
- [Kri93] Krieger D., Anrews T.: *C++ Bindings to an Object Database*. In Modern Database Systems. ACM Press, Addison-Wesley 1993, 89-107.
- [Kro94] Król D.: *Foundations of object model in information systems (in Polish)*. Report of Main Library and Information Center Technical University of Wrocław PRE 158, 1994.
- [Kro95a]* Król D.: *Dynamic bibliographic object in Digital Libraries*. The Proceedings of the International Symposium on Digital Libraries 1995. University of Library and Information, Tsukuba Science City, Japan, August 22-25 1995.
- [Kro95b] Król D.: *Modeling Bibliographic Database Dynamics on the Base of an Object-Oriented Approach*. The Proceedings of the Tenth International Symposium on Computer and Information Sciences 1995. Istanbul Technical University, Ephesus, Turkey, October 30-November 1, 1995.
- [Kro95c] Król D, Bień J., Piasecki M.: *Object concept in multimedia bridge management system for Polish state railway*. Report of Computer

- Center Technical University of Wrocław PRE 5, 1995.
- [Kro95d] Król D., Nguyen N. T.: *A data model for dynamic object-oriented information system*. The Proceedings of the ISAT '95, Bazewicz (ed.), Informatics Library of Universities, Wrocław 1995, 126-131.
- [Kro96] Król D., Nguyen N. T.: *A query language for dynamic object-oriented information system*. The Proceedings of the ISAT '96, Bazewicz (ed.), Informatics Library of Universities, Wrocław 1996.
- [Kro00a] Król D., Możdżyński A.: *Zestawienia w obiektowych bazach danych*. Informatyka 2000, 1, 17-20.
- [Kro00b] Król D., Możdżyński A.: *Summaries in Object Databases*. Intelligent Information Systems IX. Proceedings of the Workshop held in Bystra, Poland 12-16 June 2000, 62-66.
- [Kro00c] Król D., Nguyen N. T., Daniłowicz Cz.: *Aspekty migracji w obiektowej bazie danych*. Materiały konferencyjne Multimedialne i Sieciowe Systemy Informacyjne MISSI '2000, Oficyna Wydawnicza Politechniki Wrocławskiej. Wrocław 2000, 103-110.
- [Kro00d] Król D., Nguyen N. T.: *Dynamic aspects into object-oriented information system*. The Proceedings of the ISAT '2000, Grzech, Wilimowska (ed.), Informatics Library of Universities, Wrocław 2000.
- [Kro01] Król D., Kojas Ł.: *Wyszukiwanie danych w obiektowej bazie danych*. Prace naukowe Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej. Oficyna Wydawnicza Politechniki Wrocławskiej 2001 (w przygotowaniu).
- [Kul93] Kulkarni K.: *Object-orientation and the SQL standard*. Computer Standards & Interfaces 1993, 15, 287-300.
- [Lai94] Lai V., Kuilboer J., Guynes J.: *Temporal Databases: Model Design and Commercialization Prospects*. Data Base 1994, 25 (August), 6-18.
- [Lam97]* Lam F., Chau H., Wang R.: *An Efficient Indexing Scheme for Object with Roles*. Proceedings of the 15th National Conference on Database 1997, 139-153.
- [Lau00]* Lausen G., Vossen G.: *Obiektowe bazy danych*. WNT Warszawa 2000.
- [Lee98]* Lee W. C., Lee D. L.: *Path Dictionary: A New Access Method for Query Processing in Object-Oriented Databases*. IEEE Transactions on Knowledge and Data Engineering 1998, vol. 10, no. 3, 371-387.
- [Li94] Li Q., Dong G.: *A framework for object migration in object-oriented databases*. Data and Knowledge Engineering 1994, 13, 221-242.
- [Li98] Li Q., Lochovsky F.: *ADOME: An Advanced Object Modeling Environment*. IEEE Transactions on Knowledge and Data Engineering 1998, March/April, 255-275.
- [Lie93]* Lieberherr K., Xiao C.: *Formal Foundations for Object-Oriented Data Modeling*. IEEE Transactions on Knowledge and Data Engineering 1993, 5 (June), 462-478.
- [Lin96]* Lin J., Ozsoyoglu Z.: *Processing OODB Queries by O-Algebra*. CIKM

- '96, Rockville, USA 1996, 134-142.
- [Lit98] Litwin W., Morzy T., Vossen G.: *Advanced in Databases and Information Systems*. ADBIS'98. Poznań, Poland, Springer Verlag 1998, LNCS 147.
- [Liu93a]* Liu L.: *An Object-Centered Approach for Manipulating Hierarchically Complex Objects*. *Object Technologies for Advanced Software*. The Proceedings of First JSSST International Symposium, Kanazawa, Japan, November 4-6. Berlin Heidelberg, Springer Verlag 1993, LNCS 742, 194-219.
- [Liu93b]* Liu L.: *A recursive object algebra based on aggregation abstraction for manipulating complex objects*. *Data and Knowledge Engineering* 1993, 11, 21-60.
- [Liu96] Liu L., Meersman R.: *The Building Blocks for Specifying Communication Behavior of Complex Objects: An Activity-Driven Approach*. *ACM Transactions on Database Systems* 1996, 21 (June), 157-207.
- [Lor88]* Lorie R., Schek H.: *On Dynamically Defined Complex Objects and SQL*. *Advances in Object-Oriented Database Systems*. Berlin Heidelberg, Springer Verlag 1988, 323-328.
- [Man91]* Manola F., Dayal U.: *An Overview of PDM: An Object-Oriented Data*. *On Object-Oriented Database Systems*. Berlin Heidelberg, Springer Verlag 1991, 13-27.
- [Man94]* Manola F.: *An Evaluation of Object-Oriented DBMS Developments*. (1994 edition). Report of GTE Laboratories Incorporated, 1994, TR-0263-08-94-165.
- [Mar92]* Mariani J.: *Oggetto: An Object Oriented Database Layered on a Triple Store*. *The Computer Journal* 1992, 35, 108-118.
- [Mar97]* Marshall B., Kennedy J., Barclay P.: *bclasses: A construct and method for modeling co-operative object behavior*. *Information and Software Technology* 1997, 39, 351-361.
- [Maz00]* Mazur Z.: *Bazy danych*. *Prace Naukowe Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej*. Oficyna Wydawnicza Politechniki Wrocławskiej 2000.
- [McL91]* McLeod D.: *Perspective on object databases*. *Information and Software Technology* 1991, 33, 13-21.
- [Men94] Mendelzon A., Milo T., Waller E.: *Object Migration*. The proceedings of the 1994 ACM SIGMOD/PODS 94, USA 1994, 232-242.
- [Mit93] Mitchell G. A.: *Extensible Query processing in an Object-Oriented Database (rozprawa doktorska)*. Brown University 1993.
- [Mit94] Mitchell G., Zdonik S., Dayal U.: *Optimization of Object-Oriented Query Languages: Problems and Approaches*. *Advances in Object-Oriented Database Systems*. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1994, 119-146.

- [Mos93] Moscowitz S.: *ODMG-93. The object database standard for C++*. C++ Report 1993, October, 45-48.
- [Mul00] Muller R. J.: *Bazy danych język UML w modelowaniu danych*. Wydawnictwo MIKOM, Warszawa 2000.
- [Nah91]* Nahouraii E., Petry F.: *Object-Oriented Databases*. IEEE Computer Society Press, 1991.
- [Ngu96] Nguyen N. T., Sekuradzki W.: *Temporalne bazy danych*. Praktyka i teoria informacji naukowej i technicznej 1996, t. 4 nr 2, 11-17.
- [Ngu98] Nguyen N. T., Sekuradzki W.: *Models of evolution of information systems*. 9th International Symposium on System Modelling Control. Zakopane, Poland 1998.
- [Nie89] Niestrasz O.: *A Survey of Object Oriented Concepts*. Object-Oriented Concepts, Databases, and Applications. ACM Press, Addison-Wesley 1989, 3-21.
- [Och92] Ochuodho S. J.: *Object-oriented database support for software project management environments: data-modeling issues*. Information and Software Technology 1992, 34, 283-307.
- [Odb93]* Odberg E.: *Schema Versioning and Class Hierarchy Modifications in Object-Oriented Databases*. Workshop on Supporting the Evolution of Class Definition. OOPSLA '93, Washington D.C., USA, Sep. 26 - Oct. 1 '92.
- [Odb95] Odberg E.: *MultiPerspectives: Object Evolution and Schema Modification Management for Object-Oriented Databases (rozprawa doktorska)*. Department of Computer Science and Telematics, Norwegian Institute of Technology, February 1995.
- [Oli94]* Olivier M., Solms S.: *A Taxonomy for Secure Object-Oriented databases*. ACM Transactions on Database Systems 1994, 19 (March), 3-46.
- [Oxb91]* Oxborrow E., Davy M., Kemp Z., Linington P., Thearle R.: *Object-oriented data management in specialized environments*. Information and Software Technology 1991, 33, 433-442.
- [Ozs94]* Ozsú M.: *Transaction Models and Transaction management in Object-Oriented Database management Systems*. Advances in Object-Oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1994, 147-184.
- [Pan92]* Pankowski T.: *Podstawy baz danych*. Wydawnictwo Naukowe PWN, Warszawa 1992.
- [Pan00] Pankowski T.: *Model bazy danych obiektów częściowo etykietowanych (rozprawa habilitacyjna)*. Rozprawy Politechniki Poznańskiej, 2000, 354.
- [Pap91] Papazoglou M.: *Rules: a methodology for representing multifaceted objects*. The Proc. DEXA 91. Database & Expert Systems Application Conf., Springer Verlag 1991, 7-12.

- [Pap97] Papazoglou M., Kramer B.: *A database model for object dynamics*. The VLDB Journal 1997, 6, 73-96.
- [Par89]* Parsaye K., Chignell M., Khoshafian S., Wong H.: *Intelligent Databases*. John Wiley & Sons 1989.
- [Pat93]* Paton N., Leishman, Embury S., Gray P.: *On using Prolog to implement object-oriented databases*. Information and Software Technology 1993, 35, 301-311.
- [Paw83]* Pawlak Z.: *Systemy informacyjne*. Wydawnictwa Naukowo-Techniczne, Warszawa 1983.
- [Per90] Pernici B.: *Objects with Roles*. IEEE/ACM Conference on Office Information Systems 1990, 11, 205-215.
- [Pet93]* Peters R., Ozsu T.: *Reflection in a Uniform Behavioral Object Model. Entity-Relationship Approach - ER '93* The Proceedings of International Conference, ER '93, Arlington, USA, December 15-17. Berlin Heidelberg, Springer Verlag 1993, LNCS 823, 34-45.
- [Pia96] Piasecki M.: *OB! OK! obiektowy system zarządzania bazą danych*, Informatyka 1996, 8, 1-4.
- [Pir92]* Pirotte A., Delobel C., Gottlob G.: *Advances in Database Technology - EDBT '92*. Proceedings of the 3rd International Conference on Extending Database Technology (Vienna 1992). Berlin Heidelberg, Springer Verlag 1992.
- [Pis93]* Pistor P.: *Objektorientierung in SQL3: Stand und Entwicklungstendenzen*. Informatik-Spektrum 1993, 16, 89-94.
- [Pło00] Płodzień J.: *Optimization Methods in Object Query Languages (rozprawa doktorska)*. The Polish Academy of Science 2000.
- [Pro94] Proper H. A.: *A Theory for Conceptual Modelling of Evolving Application Domains (rozprawa doktorska)*. University of Nijmegen, The Netherlands, April 1994.
- [Rad94] Radeke E., Scholl M.: *Framework for Object Migration in federated Database Systems*. The Proc. Int'l. Conf. on Parallel and Distributed Information Systems (PDIS), Austin 1994.
- [Rad95]* Radeke E., Scholl M.: *Functionality for Object Migration Among Distributed, Heterogeneous, Autonomous DBA*. The Proc. Int'l. WS Research Issues on Data Engineering, Taipei 1995.
- [Rai94]* Raiha L.: *Delegation: Dynamic Specialization*. ACM 1994, 172-179.
- [Ram95]* Ramaswamy S., Kanellakis P.: *OODB Indexing by Class-Division*. The proceedings of the 1995 ACM SIGMOD Int'l. Conf. on Management of Data, New York, USA 1995, 139-150.
- [Ric91] Richardson J., Schwartz P.: *Aspects: extending objects to support multiple, independent roles*. The Proc. 1991 ACM SIGMOD Int'l. Conf. on Management of Data, ACM, New York 1991, 298-307.
- [Rie96a]* Riedel H., Scholl M.: *A Formalization of ODMG Queries*. The Proc. IFIP 1996.

- [Rie96b]* Riedel H., Scholl M.: *The CROQUE-Model: Formalization of the Data Model and Query Language*. Technical Report 23, 1996.
- [Ros93] Rose E., Segev A.: *TOOA: A Temporal Object-Oriented Algebra*. ECOOP '93 - Object-Oriented Programming. Berlin Heidelberg, Springer Verlag 1993, LNCS 707, 297-325.
- [Saa95]* Saad A., Bertino E., Ismail M. A.: *A cost model for queries in object-oriented databases*. Object Oriented Systems 1995, 2, 1-26.
- [Sav93]* Savnik I., Mohoric T., Dolenc T., Novak F.: *Database Model for Design Data*. OOPS Messenger 1993, 4, 26-40.
- [Sch94]* Scholl M., Tresch M.: *Evolution towards, in, and beyond Object Databases*. The Proc.3rd GI Workshop Information Systems and Artificial Intelligence, Hamburg 1994, 64-82.
- [Ska86] Skarra A., Zdonik S.: *The Management of Changing Types in an Object-Oriented Database*. OOPSLA '86 Proceedings. ACM Press 1986, 483-495.
- [Ska89]* Skarra A., Zdonik S.: *Concurrency Control and Object-Oriented Databases*. Object-Oriented Concepts, Databases, and Applications. ACM Press, Addison-Wesley 1989, 395-421.
- [Sla93] Slack J., Unger E.: *Integrity in object-oriented database systems*. Computers & Security 1993, 12, 389-404.
- [Sly91]* Slyngbaek P., Kent W.: *A Data Modeling Methodology for the Design and Implementation of Information Systems*. On Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1991, 198-205.
- [SQL92] American National Standards Institute: *American National Standard for Information System Database Language SQL*. ANSI X3.135-1992, American National Standards Institute 1992.
- [Spo91] Spooner D. L.: *Towards an Object-Oriented Data Model for a Mechanical CAD Database System*. On Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1991, 198-205.
- [Sta94]* Stapor K.: *Wyprowadzenie schematu bazy danych z dynamicznej specyfikacji systemu informatycznego (rozprawa doktorska)*. Instytut Informatyki Politechniki Śląskiej 1994.
- [Sto91]* Stonebraker M.: *Object Management in Postgres using Procedures*. On Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1991, 29-43.
- [Su91] Su J.: *Dynamic Constraints and Object Migration*. The Proceedings of the 17th International VLDB Conference 1991, 233-242.
- [Su93]* Su S. Y. W., Guo M., Lam H.: *Association Algebra: A Mathematical Foundation for Object-Oriented Databases*. IEEE Transactions on Knowledge and Data Engineering 1993, 5, 775-798.
- [Su98] Su S. Y. W., Hyun S., Chen M.: *Temporal Association Algebra: A Mathematical Foundation for Processing Object-Oriented Temporal Databases*. IEEE Transactions on Knowledge and Data Engineering 1998, vol. 10, no. 3, 389-407.

- [Sub98] Subieta K.: *Obiektywność w projektowaniu i bazach danych*. Akademicka Oficyna Wydawnicza PLJ. Warszawa 1998.
- [Sun92]* Sun W., Meng W., Yu C.: *Query Optimization in Distributed Object-Oriented Database Systems*. The Computer Journal 1992, 35, 98-107.
- [Tet91] Teta J.S.: *Model złożonej relacyjnej bazy danych (rozprawa doktorska)*. Raport BG i OINT Politechniki Wrocławskiej PRE 136, 1991.
- [Tho93] Thompson C.: *A reference model for object data management*. Computer Standards & Interfaces 1993, 15, 121-147.
- [Tre92]* Tresch M., Scholl M.: *Meta Object Management and its Application to Database Evolution*. Entity-Relationship Approach - ER '92. Berlin Heidelberg, Springer Verlag 1992, 299-321.
- [Tsi89]* Tsichritzis D., Niestrasz O.: *Directions in Object-Oriented Research*. Object-Oriented Concepts, Databases, and Applications. ACM Press, Addison-Wesley 1989, 523-536.
- [Ull00] Ullman J., Widom J.: *Podstawowy wykład z systemów baz danych*. Wydawnictwa Naukowo-Techniczne. Warszawa 2000.
- [Uni95a] *UniSQL: Database Product*. Application Program Interface. Reference Guide. UniSQL Inc. August 1995.
- [Uni95b] *UniSQL: Database Management System*. User's Manual. UniSQL Inc. August 1995.
- [Urb94a] Urban S., Lai Ch., Saxena S.: *The Design and Translation of ORL: An Object Retrieval Language*. Journal Systems Software 1994, 24, 187-206.
- [Urb94b]* Urban S., Abdellatif T.: *An Object-Oriented Query Language Interface to Relational Databases in a Multidatabase Environment*. Proceedings of the 14th International Conference on Distributed Computing Systems (Poznan, June 21-24 1994). IEEE Computer Society Press 1994, 387-394.
- [Wad95]* Wade S., Braekevelt P.: *IR_Framework: an object oriented framework for developing information retrieval systems*. Program 1995, 29, 15-29.
- [Wan89]* Wand Y.: *A proposal for a Formal Model of Objects*. Object-Oriented Concepts, Databases, and Applications. ACM Press, Addison-Wesley 1989, 537-559.
- [Wan95]* Wang B., Hitchcock P.: *InterSect_DM: a hypertext data model based on OODBMS*. Information and Software Technology 1995, 37, 177-190.
- [Wie90]* Wieringa R. J.: *Algebraic Foundations for Dynamic Conceptual Models (rozprawa doktorska)*. Vrije Universiteit Te Amsterdam 1990.
- [Wie91] Wiederhold G.: *Views, Objects, and Databases*. On Object-Oriented Database Systems. Berlin Heidelberg, Springer Verlag 1991, 29-43.
- [Wie93] Wiczerzycki W.: *Obiektowe bazy danych*. Informatyka 1993, 1, 8-12.
- [Wie94]* Wieringa R. J., Jonge W., Spruit P.: *Roles and dynamic subclasses: a*

- modal logic approach*. Technical report. Vrije Universiteit Te Amsterdam 1994.
- [Wie95] Wieringa R. J., Jonge W., Spruit P.: *Using Dynamic Classes and Role Classes to Model Object Migration*. Theory and Practice of Object Systems 1995, 1, 61-83.
- [Wil90]* Wilkinson K., Lyngbaek P., Hasan W.: *The IRIS Architecture and Implementation*. IEEE Transactions on Knowledge and Data Engineering 1990, vol. 2, no. 1, 63-75.
- [Won97a] Wong R., Chau H. L., Lochovsky F.: *A Data Model and Semantics of Objects with Dynamic Roles*. IEEE Proceedings of 13th Conference of Data Engineering 1997, 402-411.
- [Won97b]* Wong R., Chau H. L., Lochovsky F.: *Dynamic Knowledge Representation in DOOR*. IEEE Proceedings of 13th Conference of Data Engineering 1997, 89-96.
- [Won98]* Wong R., Chau H. L.: *Method Dispatching and Type Safety for Objects with Multiple Roles*. IEEE Proceedings of Conference of Technology of Object-Oriented Languages and Systems 1998, 286-296.
- [Yoo93] Yoo S., Chang H.: *An Object-Oriented Query Model Supporting Views*. *Object Technologies for Advanced Software*. The Proceedings of First JSSST International Symposium, Kanazawa, Japan, November 4-6. Berlin Heidelberg, Springer Verlag 1993, LNCS 742, 251-263.

ZAŁĄCZNIK

Syntaktyka podstawowych instrukcji języka MOW w notacji BNF

$\langle \text{Instrukcja} \rangle ::= \langle \text{Instrukcja konfiguracji} \rangle$
 $\quad | \langle \text{Instrukcja wprowadzania} \rangle | \langle \text{Instrukcja aktualizacji} \rangle | \langle \text{Instrukcja usuwania} \rangle$
 $\quad | \text{count} (\langle \text{Instrukcja wyszukiwania} \rangle) | \langle \text{Instrukcja wyszukiwania} \rangle$
 $\langle \text{Instrukcja konfiguracji} \rangle ::= \text{set migration checked} | \text{set migration unchecked}$
 $\langle \text{Instrukcja wprowadzania} \rangle ::= \text{insert} \langle \text{Obiekt} \rangle | \text{insert} \langle \text{Oid} \rangle \text{ into } \langle \text{Oid} \rangle \{, \langle \text{Oid} \rangle\}$
 $\quad | \text{insert} \langle \text{Oid} \rangle \text{ into } \langle \text{Instrukcja wyszukiwania} \rangle$
 $\langle \text{Obiekt} \rangle ::= (\langle \text{Oid} \rangle, \langle \text{Zbiór podobiektów} \rangle, \langle \text{Zbiór nadobiektów} \rangle, \langle \text{Zbiór atrybutów}$
 $\quad \text{podobiektów} \rangle, \langle \text{Wartość obiektu} \rangle, \langle \text{Zbiór metod} \rangle)$
 $\langle \text{Oid} \rangle ::= \langle \text{Nazwa} \rangle$
 $\langle \text{Zbiór podobiektów} \rangle ::= \langle \text{Oid} \rangle \{, \langle \text{Oid} \rangle\}$
 $\langle \text{Zbiór nadobiektów} \rangle ::= \langle \text{Oid} \rangle \{, \langle \text{Oid} \rangle\}$
 $\langle \text{Zbiór atrybutów podobiektów} \rangle ::= \langle \text{Atrybut} \rangle \{, \langle \text{Atrybut} \rangle\}$
 $\langle \text{Atrybut} \rangle ::= \langle \text{Nazwa} \rangle$
 $\langle \text{Wartość obiektu} \rangle ::= \langle \langle \text{Atrybut} \rangle : \langle \text{Wartość} \rangle \{, \langle \text{Atrybut} \rangle : \langle \text{Wartość} \rangle \} \rangle$
 $\langle \text{Zbiór metod} \rangle ::= \langle \text{Mid} \rangle \{, \langle \text{Mid} \rangle\}$
 $\langle \text{Mid} \rangle ::= \langle \text{Nazwa} \rangle$
 $\langle \text{Instrukcja aktualizacji} \rangle ::= \langle \text{Instrukcja aktualizacji wartości} \rangle | \langle \text{Instrukcja aktualizacji}$
 $\quad \text{atrybutów} \rangle | \langle \text{Instrukcja aktualizacji obiektu} \rangle$
 $\langle \text{Instrukcja aktualizacji wartości} \rangle ::= \text{update} \langle \text{Oid} \rangle \text{ set } \langle \langle \text{Atrybut} \rangle : \langle \text{Wartość} \rangle$
 $\quad \{, \langle \text{Atrybut} \rangle : \langle \text{Wartość} \rangle \} \rangle$
 $\langle \text{Instrukcja aktualizacji atrybutów} \rangle ::= \text{update} \langle \text{Oid} \rangle \text{ del } \langle \text{Atrybut} \rangle \{, \langle \text{Atrybut} \rangle\}$
 $\langle \text{Instrukcja aktualizacji obiektu} \rangle ::= \text{update} \langle \text{Oid} \rangle \text{ from } \langle \text{Oid} \rangle \{, \langle \text{Oid} \rangle\}$
 $\quad \text{into } \langle \text{Oid} \rangle \{, \langle \text{Oid} \rangle\} | \text{update} \langle \text{Oid} \rangle \text{ from } \langle \text{Oid} \rangle \{, \langle \text{Oid} \rangle\} \text{ into } \langle \text{Instrukcja}$
 $\quad \text{wyszukiwania} \rangle | \text{update} \langle \text{Oid} \rangle \text{ from } \langle \text{Instrukcja wyszukiwania} \rangle \text{ into } \langle \text{Oid} \rangle \{,$
 $\quad \langle \text{Oid} \rangle\} | \text{update} \langle \text{Oid} \rangle \text{ from } \langle \text{Instrukcja wyszukiwania} \rangle \text{ into } \langle \text{Instrukcja}$
 $\quad \text{wyszukiwania} \rangle$
 $\langle \text{Instrukcja usuwania} \rangle ::= \text{delete} \langle \text{Oid} \rangle | \text{delete} \langle \text{Oid} \rangle \text{ from } \langle \text{Oid} \rangle \{, \langle \text{Oid} \rangle\}$

| **delete** <Oid> **from** <Instrukcja wyszukiwania>
 <Instrukcja wyszukiwania> ::= <Instrukcja wyszukiwania> <Operator mnogościowy>
 <Instrukcja wyszukiwania> |
select <Klauzula celu> |
select <Klauzula celu> **from** <Klauzula zakresu> |
select <Klauzula celu> **where** <Klauzula kwalifikacji> |
select <Klauzula celu> **from** <Klauzula zakresu> **where** <Klauzula kwalifikacji>
 <Operator mnogościowy> ::= **union** | **difference** | **intersection**
 <Klauzula celu> ::= [**distinct**] <Cel> {, <Cel>}
 <Cel> ::= <Ścieżka> [*]
 <Ścieżka> ::= <Oid>{.<Oid> }.<Atrybut> | Sub(<Oid>).<Atrybut> |
 <Zmienna>{.<Oid>}.<Atrybut>
 <Zmienna> ::= <Nazwa>
 <Klauzula zakresu> ::= <Zmienna> **in** <Oid> {, <Oid>}{, <Zmienna> **in** <Oid> {, <Oid>}}
 <Klauzula kwalifikacji> ::= <Term złożony> | <Klauzula kwalifikacji> **or** <Term złożony>
 <Term złożony> ::= { <Term> | <Term złożony> **and** <Term>
 <Term> ::= [**not**] <Predykat> | (Klauzula kwalifikacji)
 <Predykat> ::= <Porównanie> | <Element zbioru> | <Kwantyfikator> | <Zawieranie>
 | <Równość>
 <Porównanie> ::= <Ścieżka złożona> <Operator porównania> <Ścieżka złożona>
 | <Ścieżka złożona> <Operator> <Stała>
 | <Ścieżka złożona> <Operator> <Zapytanie kwantyfikowane>
 <Ścieżka złożona> ::= <Ścieżka> | <Agregacja> (<Ścieżka>)
 <Agregacja> ::= **avg** | **sum** | **max** | **min**
 <Zapytanie kwantyfikowane> ::= [**all** | **any**] <Instrukcja wyszukiwania>
 <Operator porównania> ::= = | ≠ | < | > | ≤ | ≥
 <Element zbioru> ::= <Ścieżka> **isin** <Zapytanie kwantyfikowane>
 | <Ścieżka> **isin** <Ścieżka>
 <Kwantyfikator> ::= **forall** | **forany** <Zmienna> **in** <Ścieżka> (<Klauzula kwalifikacji>)
 <Zawieranie> ::= <Ścieżka> **contains** <Zapytanie kwantyfikowane>
 <Równość> ::= <Ścieżka> **is** <Ścieżka>