

**Prace Naukowe
Wydziałowego Zakładu Informatyki
Politechniki Wrocławskiej**

4

BAZY DANYCH

Redakcja naukowa

Zygmunt Mazur

**Oficyna Wydawnicza
Politechniki Wrocławskiej**





Prace Naukowe
Wydziałowego Zakładu Informatyki
Politechniki Wrocławskiej

4

BAZY DANYCH

Redakcja naukowa

Zygmunt Mazur



Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2003



WYDZIAŁOWY ZAKŁAD INFORMATYKI

WYDZIAŁ INFORMATYKI I ZARZĄDZANIA
POLITECHNIKI WROCŁAWSKIEJ

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

Recenzent

Aleksander ZGRZYWA

Opracowanie redakcyjne materiałów

Hanna MAZUR

Biblioteka Główna i OINT
Politechniki Wrocławskiej



001758643

Projekt okładki

Zygmunt MAZUR



312456L/1

© Copyright by Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2003

OFICyna WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISBN 83-7085-706-X

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej, Zam. nr 318/2003.

Alu 2003/0490 /x

SPIS TREŚCI

<i>Wstęp</i>	5
Z. Mazur, K. Piechota, <i>Projektowanie schematów multibaz danych</i>	7
D. Król, Ł. Kojs, <i>Wyszukiwanie danych w obiektowej bazie danych</i>	19
H. Mazur, Z. Mazur, <i>Aktywne bazy danych</i>	41
A. Al-Awady, Z. Mazur, <i>Autoryzacja w wielowersyjnych obiektowych bazach danych z uwzględnieniem uwarunkowań czasowych</i>	57
A. Al-Awady, Z. Mazur, <i>Autoryzacja na grafach acyklicznych w obiektowych bazach danych z uwzględnieniem uwarunkowań czasowych</i>	71
L. Madeyski, <i>XML w bazach danych</i>	81
K. Dyczkowska, <i>Technologie udostępniania baz danych w Internecie</i>	91
J. Surma, <i>Metody analityczne w informatycznych systemach zarządzania</i>	109

CONTENTS

<i>Introduction</i>	5
Z. Mazur, K. Piechota, <i>Design of Multidatabase Schemes</i>	7
D. Król, Ł. Kojs, <i>Data Retrieval in Object Database</i>	19
H. Mazur, Z. Mazur, <i>Active Databases</i>	41
A. Al-Awady, Z. Mazur, <i>Authorization on Multiversion Object-Databases with Time Related Aspects</i>	57
A. Al-Awady, Z. Mazur, <i>Authorization Methods for Direct Acyclic Graph in Object-Databases with Time Related Aspects</i>	71
L. Madeyski, <i>XML in Databases</i>	81
K. Dyczkowska, <i>Methods which Are Used to Make Databases Available via the Internet</i>	91
J. Surma, <i>Analytical Methods in Management Information Systems</i>	109

*Projektowanie baz
jest bardziej sztuką niż nauką ścisłą;
wymaga tyle samo intuicji,
co wiedzy teoretycznej i praktycznej*

• Michael J. Hernandez

WSTĘP

Tematyka czwartego Zeszytu Naukowego „Bazy Danych” jest związana z projektowaniem schematu rozproszonej bazy danych, z wyszukiwaniem i autoryzacją w obiektowej bazie danych, z wykorzystaniem języka XML w bazach danych i z technologiami udostępniania baz danych w Internecie.

Przedstawione w tym Zeszycie Naukowym artykuły dotyczą:

- ❖ projektowania schematów multibaz danych,
- ❖ wyszukiwania danych w obiektowej bazie danych na przykładzie systemu obiektowego ObjectStore,
- ❖ autoryzacji w wielowersyjnych obiektowych bazach danych ze szczególnym uwzględnieniem uwarunkowań czasowych,
- ❖ autoryzacji na grafach acyklicznych,
- ❖ aktywnych baz danych,
- ❖ języka XML w bazach danych, który zapewnia przenośność danych,
- ❖ technologii udostępniania baz danych w Internecie,
- ❖ metod analitycznych w informatycznych systemach zarządzania.

W pierwszym artykule zostały przedstawione pewne aspekty projektowania globalnego schematu bazy danych integrującego istniejące już lokalne różnorodne bazy danych. Globalny model danych wprowadza swojego rodzaju „wirtualny” poziom reprezentacji, umożliwia globalnym użytkownikom manipulowanie lokalnymi danymi. Przedstawiono strategię projektowania schematów multibaz danych.

W drugiej pracy przedstawiono metody wyszukiwania danych w obiektowej bazie danych. Zagadnienie zilustrowano na wybranym obiektowym systemie zarządzania bazą danych ObiektStore. Opisano funkcje wyszukiwawcze. Przeprowadzono kilkanaście eksperymentów wyjaśniających mechanizmy zaprezentowanych funkcji wyszukiwawczych.

Kolejna praca zawiera podstawowe pojęcia z zakresu funkcji aktywnych baz danych, omówiono podstawowe problemy występujące w aktywnych bazach danych oraz reguły aktywne ECA.

W następnych pracach przedstawiono podejście autoryzacji do wielowersyjnej obiektowej bazy danych ze szczególnym uwzględnieniem uwarunkowań czasowych. Zostały wyjaśnione pojęcia związane z bezpieczeństwem danych w obiektowych bazach danych mono i wielowersyjnych pracującymi w czasie rzeczywistym. Na przykładach przedstawione zostały uwarunkowania czasowe w nadawaniu i odwołaniu praw dostępu. Zaproponowano nowe podejście do zarządzania autoryzowanym dostępem do baz danych, które umożliwia efektywną, hierarchiczną autoryzację w skierowanych grafach acyklicznych. Podstawową ideą tego podejścia jest rozszerzenie autoryzacji podstawowej o tzw. znaczniki węzłów, które zawierają informacje o umiejscowieniu węzłów objętych autoryzacją w odpowiednich hierarchiach (grafach). Informacja ta jest odpowiednikiem sekwencji autoryzacji intencjonalnych zakładanych od korzenia drzewa (grafu) do odpowiedniego węzła. Istotnie ułatwia to wykrywanie niezgodności autoryzacji żądanej z autoryzacjami ustawionymi. Korzyści wypływające z zastosowania zaproponowanego podejścia są szczególnie widoczne w przypadku baz, których dane są semantycznie powiązane skomplikowanymi hierarchiami i grafami, takich jak obiektowe bazy danych.

W pracy „XML w bazach danych” zaprezentowano szeroki wachlarz produktów i zagadnień związanych z zastosowaniem XML w systemach bazodanowych. Powszechna akceptacja dla XML spowodowała duże zmiany na rynku baz danych. Natywne bazy XML rozszerzają paletę dostępnych narzędzi. Wciąż pojawiają się nowe koncepcje wywierające istotny wpływ na kierunek dalszego rozwoju aplikacji internetowych oraz także systemów bazodanowych opartych na XML.

W kolejnym artykule zaprezentowano technologie udostępniania baz danych poprzez sieć Internet, o których można powiedzieć, że wyznaczają kierunki rozwoju współczesnych systemów integrujących zalety baz danych, szerokie możliwości Internetu oraz potrzeby światowego biznesu.

W ostatnim artykule przedstawiono przegląd zastosowań komputerowej analizy danych w kontekście rozwoju systemów informatycznych zarządzania i hurtowni danych. Omówiono podstawowe metody analityczne i ich praktyczne wykorzystanie w komputerowym wspomaganie decyzji.

Zaprezentowane artykuły są wynikiem prac prowadzonych głównie w Wydziałowym Zakładzie Informatyki na Wydziale Informatyki i Zarządzania Politechniki Wrocławskiej.

Autorzy składają podziękowanie Recenzentowi za uwagi przedstawione w recenzji, które zostały uwzględnione w trakcie przygotowywania ostatecznych wersji artykułów.

Zygmunt Mazur

*homogeniczne bazy danych,
fragmentacja bazy, multibaza danych,
schemat bazy danych*

Zygmunt MAZUR*
Kamila PIECHOTA

PROJEKTOWANIE SCHEMATÓW MULTIBAZ DANYCH

W artykule zostały przedstawione pewne aspekty projektowania globalnego schematu bazy danych integrującego istniejące już lokalne różnorodne bazy danych. Globalny model danych wprowadza swojego rodzaju „wirtualny” poziom reprezentacji, umożliwia globalnym użytkownikom manipulowanie lokalnymi danymi. W pracy są przedstawione strategie projektowania schematów multibaz danych.

1. WPROWADZENIE

Jedną z podstawowych różnic pomiędzy homogenicznymi systemami zarządzania bazami danych (*ang. Database Management System – DBMS*) a systemem zarządzania multibazą danych – MDBMS (*ang. Multidatabase Management System*) jest fakt, że rozproszone bazy danych dla homogenicznych DBMS projektuje się używając metodyki typu *top-down*, zaś dla projektowania multibaz danych stosuje się najczęściej metodyki *bottom-up*. Projektowanie homogenicznych baz danych jest tylko rozszerzeniem projektowania baz scentralizowanych o fazę fragmentacji i alokacji.

Projektowanie multibaz danych stanowi zupełnie inny problem związany z integracją już istniejących lokalnych baz danych i wiąże się z wprowadzeniem globalnego konceptualnego schematu bazy danych. Globalny model danych wprowadza swojego rodzaju „wirtualny” poziom reprezentacji i umożliwia globalnym użytkownikom manipulowanie lokalnymi danymi. Gromadzenie i przetwarzanie informacji pochodzących z różnych źródeł i udostępnianie ich wielu użytkownikom oraz reprezentacja złożonych struktur danych w postaci wielu plików danych prowadzą również do poważnych trudności w utrzymaniu logicznej spójności danych. Globalny użytkownik powinien mieć możliwość sformułowania zapytania bez

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej,
50-370 Wrocław, Wybrzeże Wyspiańskiego 27, mazur@ci.pwr.wroc.pl

potrzeby określania miejsca, gdzie są przechowywane interesujące go dane. W systemach ubogich w ułatwienia dostępu do lokalnych danych, użytkownik musiałby pisać programy zawierające oddzielne transakcje do każdego węzła w odpowiednim lokalnym języku zapytań i otrzymane wyniki łączyć połączyć w swoim programie. Dla użytkowników rozproszonych baz danych najważniejszy jest łatwy i szybki dostęp do danych w różnych węzłach, uzyskiwanie poprawnych odpowiedzi na zapytania oraz niezawodność rozproszonych systemów i zabezpieczenia przed nieupoważnionym dostępem do danych.

2. INTEGRACJA SCHEMATÓW BAZ DANYCH

Prace nad większością istniejących prototypów multibaz danych zostały skoncentrowane przede wszystkim nad badaniami dotyczącymi skonstruowania i usprawnienia mechanizmów pozwalających na sprawne funkcjonowanie MDBMS, takich jak: protokoły transakcyjne czy optymalizatory zapytań. Istnieje niewiele prac zajmujących się integracją schematów lokalnych. Idealnie byłoby gdyby MDBMS potrafiły automatycznie na podstawie lokalnych schematów wygenerować schemat globalny. Czy jednak w ogóle jest możliwe stworzenie narzędzia automatycznie generującego globalny słownik, schemat globalny, czy schematy uczestnictwa z zachowaniem wszystkich reguł integralności? Jak dotąd, nie istnieje nawet powszechnie zaakceptowana metodologia, na podstawie której takie narzędzia można by skonstruować [BellGr94].

Problem integracji schematów w MBD jest bardzo skomplikowanym zagadnieniem. Różnice pomiędzy poszczególnymi lokalnymi schematami dotyczą zarówno semantyki jak i różnic strukturalnych. Lokalne schematy powstawały w sposób niezależny, przy użyciu różnych metodologii jak i realizowały różne filozofie. Główna zasada, którą należy przyjąć przy projektowaniu schematu globalnego mówi, że schemat globalny nie może zawierać więcej informacji niż suma informacji zawartych we wszystkich schematach lokalnych. Z drugiej strony, dzięki dostępowi do globalnego schematu rozproszonej bazy danych możemy uzyskiwać nawet takie informacje, które nie byłyby dostępne w poszczególnych lokalnych bazach danych.

Przykład 1

Dane są schematy trzech baz danych użytkowanych lokalnie w różnych węzłach sieci komputerowej.

węzeł A:

Dział osobowy pewnej firmy

PRACOWNICY(nr-prac, nazwisko, nr-zakł, płeć, zarobek)

węzeł B:

Książka adresowa typu „Panorama firm”, zawierająca REGONy firm i adresy

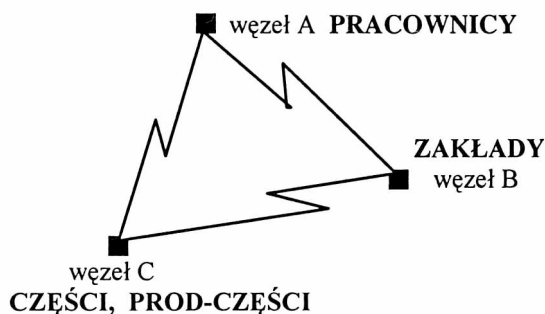
ZAKŁADY(nr-zakł, adres)

węzeł C:

Magazyn części zawierający informacje o częściach i ich dostawcach

CZĘŚCI(nr-części, nazwa, masa, kolor)

PROD-CZĘŚCI(nr-części, nr-zakł)



Globalny użytkownik z dowolnego węzła sieci może uzyskać z rozproszonej bazy danych odpowiedź na zapytanie

„Podaj nazwiska i zarobki wszystkich mężczyzn
produkujących we Wrocławiu zielone części”.

Odpowiedzi na takie zapytanie nie można byłoby uzyskać w żadnym z lokalnych węzłów. Odpowiedzi na takie zapytanie nie uzyska żaden lokalny użytkownik w węźle A, B lub w węźle C.

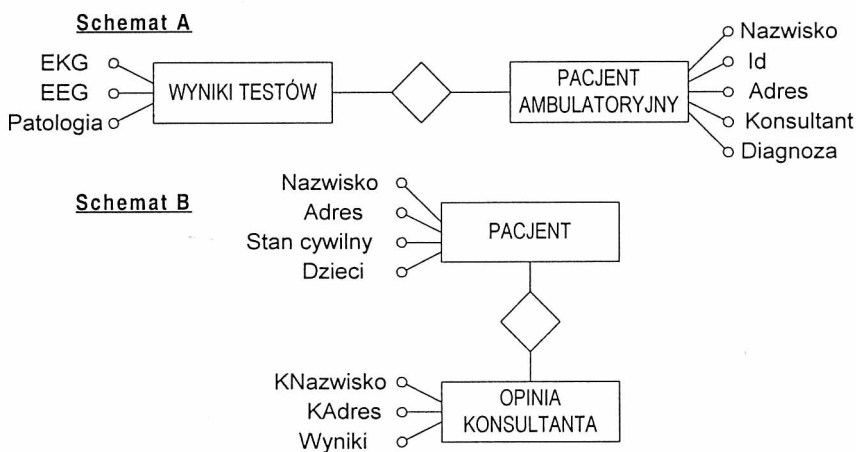
■

Użytkownik rozproszonej bazy danych powinien mieć możliwość sformułowania swojego zapytania bez potrzeby określania miejsca, gdzie tak naprawdę są przechowywane dane, bez znajomości schematów lokalnych baz danych oraz powinien mieć także możliwość korzystania z rozproszonych baz danych w taki sam sposób, jak z pojedynczej, scentralizowanej bazy danych bez konieczności dekomponowania zapytania do lokalnych baz danych opartych najczęściej na różnych modelach danych i oprogramowaniach narzędziowych z różnych firm.

Do podstawowych problemów w integracji lokalnych schematów należy przyjęcie globalnego nazewnictwa. Te same obiekty w różnych schematach mogą mieć różne nazwy i odwrotnie, różne obiekty mogą być identyfikowane tą samą nazwą. Dodatkowo, dane mogą być reprezentowane w poszczególnych schematach na różnych poziomach abstrakcji, np. w jednym ze schematów określony typ danych może być reprezentowany jako atrybut relacji, w innym zaś jako relacja posiadająca własne prawa.

Problemy związane z integracją schematów lokalnych i poszczególne etapy dojścia do schematu globalnego najłatwiej prześledzić na przykładzie. Przytoczony przykład jest bardzo prosty, ale pozwala zobrazować podstawowe problemy dotyczące integracji schematów.

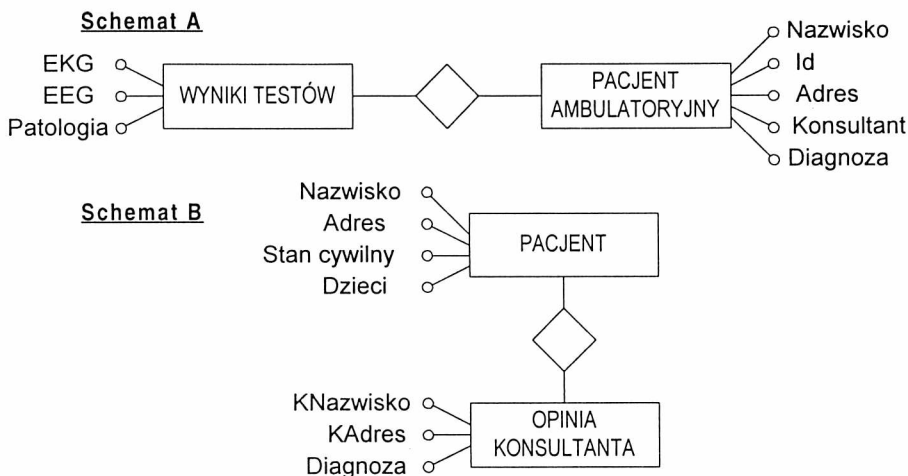
Na rysunku 1 zostały przedstawione dwa schematy lokalne: A i B. Są to modele opisujące dane o pacjentach w dwóch oddziałach pewnego szpitala.



Rys. 1. Schematy lokalne A i B przed integracją
Fig. 1. Local schemes A and B before an integration

Rysunek 1 przedstawia oryginalną postać lokalnych schematów. Kolejne transformacje mają na celu zintegrowanie schematów w pojedynczy, jednolity schemat globalny.

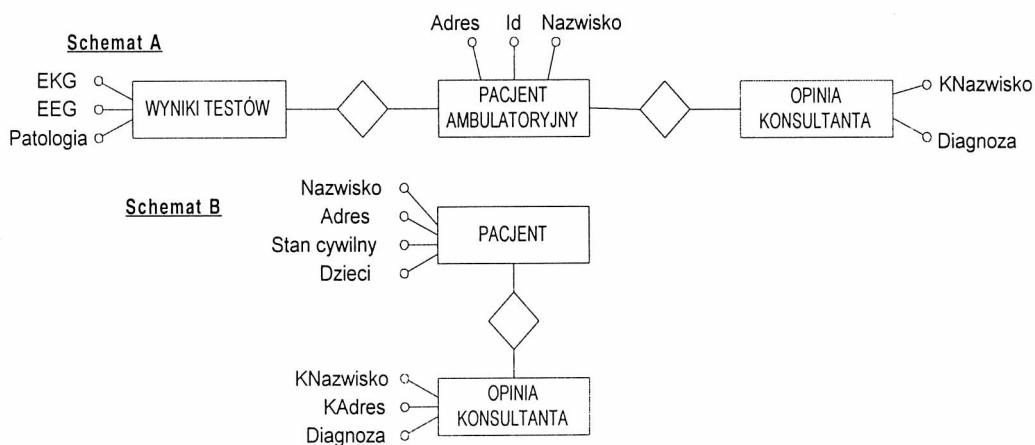
W pierwszym kroku zostały zidentyfikowane atrybuty będące synonimami: w schemacie A – *Diagnoza*, w schemacie B – *Wyniki*. W schemacie globalnym wybrano dla tych atrybutów nazwę *Diagnoza* (rysunek 2). W schemacie A atrybut *Konsultant* jest atrybutem relacji *PACJENT AMBULATORYJNY*, natomiast w schemacie B relacji *OPINIA KONSULTANTA*. Jeżeli zdecydujemy, że bardziej właściwe jest, aby w schemacie globalnym znalazła się relacja *OPINIA KONSULTANTA*, to do schematu A należy taką relację dołączyć, jak to zostało pokazane na rysunku 3.



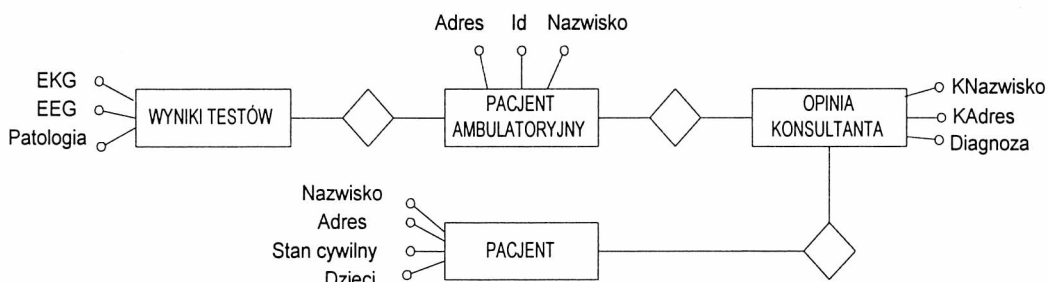
Rys. 2. Wynik pierwszego kroku w integracji schematów
Fig. 2. Step 1 of schemes integration

Na tym etapie można już złączyć oba schematy. Ogniwem łączącym jest relacja *OPINIA KONSULTANTA* (rysunek 4).

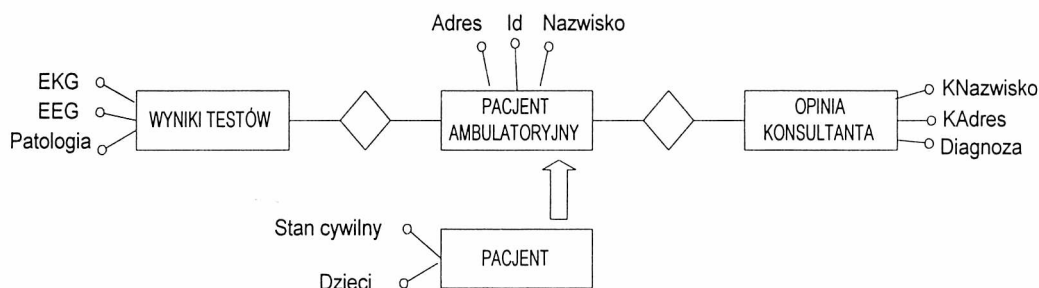
W następnym kroku można zauważyć fakt, że atrybuty relacji *PACJENT* są podzbiorem atrybutów relacji *PACJENT AMBULATORYJNY*. Upewniwszy się, że poszczególne atrybuty posiadają to samo znaczenie, możemy utworzyć związek pomiędzy *PACJENTEM* i *PACJENTEM AMBULATORYJNYM*. Ostateczny zintegrowany model globalnego schematu został przedstawiony na rysunku 5.



Rys. 3. Wynik drugiego kroku w integracji schematów
Fig. 3. Step 2 of schemes integration



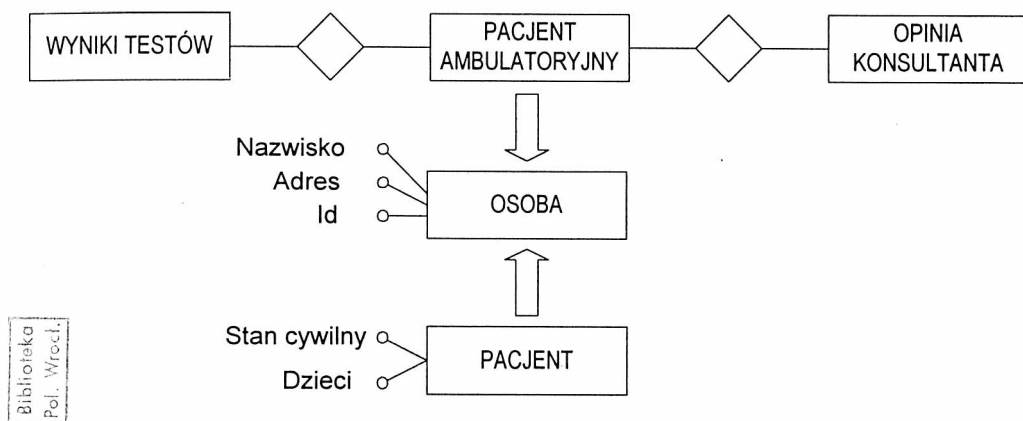
Rys. 4. Wynik trzeciego kroku integracji schematów
Fig. 4. Step 3 of schemes integration



Rys. 5. Zintegrowane schematy bazy danych
Fig. 5. Integrated schemes of database

W czasie kolejnych transformacji należy zwrócić szczególną uwagę na semantykę poszczególnych relacji i atrybutów. Przyjrzyjmy się relacji *PACJENT* w schemacie lokalnym B. Nie wykluczone jest, że pacjentem nie musi być pacjent ambulatoryjny. W takim przypadku uzyskany schemat globalny należy skorygować. Można dodać dodatkową relację *OSOBA*, której atrybutami będzie zbiór wspólnych atrybutów relacji *PACJENT* i *PACJENT AMBULATORYJNY*, jak zostało to pokazane na rysunku 6.

Integracja schematów wymaga zastosowania bogatego semantycznie modelu. Okazuje się, że to zagadnienie jest na tyle skomplikowane, że model relacyjny jest zbyt ubogi semantycznie, aby przedstawić wszystkie aspekty integracji. Integracja wymaga zastosowania związku oznaczającego, że zbiór atrybutów jednej z kategorii jest podzbiorem atrybutów innej (jak przytoczonym powyżej przykładzie związek pomiędzy *PACJENTEM* i *PACJENTEM AMBULATORYJNYM*). Taki rodzaj związku



Rys. 6. Wprowadzenie nowej kategorii OSOBA

Fig. 6. Introduction of a new category OSOBA

między kategoriami nazywany jest *generalizacją*. Bez możliwości używania związku generalizacji, istnieje ryzyko utracenia informacji w procesie scalania schematów lokalnych. Druga znacząca zaleta używania związku generalizacji wiąże się z możliwością minimalizacji częstości używania wartości *null* [DaHw84]. Wracając do wyżej przytoczonego przykładu rozważmy go, pomijając atrybut *Konsultant* kategorii *PACJENT AMBULATORYJNY* w schemacie A i kategorię *OPINIA KONSULTANTA* w schemacie B:

- Kategoria *PACJENT AMBULATORYJNY* w schemacie A składa się z atrybutów: *nazwisko*, *id*, *adres* i *diagnoza*.
- Kategoria *PACJENT* w schemacie B składa się z atrybutów: *nazwisko*, *adres*, *stan cywilny* i *dzieci*.

Z powyższych kategorii można utworzyć w globalnym schemacie jedną ogólną kategorię *PACJENT* składającą się z atrybutów: *nazwisko*, *id*, *adres*, *diagnoza*, *stan cywilny*, *dzieci*.

Pomijając generalizację, relację *PACJENT* można uzyskać wykonując operację zewnętrznego złączenia (*ang. outer joint*) relacji lokalnych *PACJENT* i *PACJENT AMBULATORYJNY*. Wymagałoby to wstawienia wartości *null* dla atrybutów *id* i *diagnoza* w danych o pacjentach pochodzących z lokalizacji A. Podobnie w danych pacjentów z lokalizacji B, atrybuty *stan cywilny* i *dzieci* musiałyby mieć wartość *null*. Nadmierna liczba wartości *null* nie jest korzystna z punktu widzenia użytkownika globalnego.

Model relacyjny nie udostępnia możliwości reprezentacji *agregacji*. Agregacja pozwala na reprezentację związku pomiędzy kategoriami jako obiektu wyższego poziomu. Przykładowo kategoria *SPOTKANIE* mogłaby być reprezentacją związku pomiędzy pacjentem, lekarzem i szpitalem.

3. ALTERNATYWNA STRATEGIA PROJEKTOWANIA GLOBALNEGO SCHEMATU BAZY DANYCH

W przytoczonym wyżej przykładzie projektowania globalnego schematu bazy danych zostało wykorzystane podejście typu *bottom-up*. Początkowo uwaga została skupiona na szczegółach, stopniowo rozszerzana, aż do ogarnięcia całego systemu. W większości metodologii projektowania schematu globalnego stosuje się takie podejście. Wyjątek stanowi metodologia zaproponowana dla systemu MultiStar. Podejście to jest kombinacją analizy typu *top-down* oraz integracji typu *bottom-up*. Dla złożonych schematów integracja schematów lokalnych przy użyciu podejścia typu *bottom-up* jest skomplikowana. Zaproponowano jednak, aby w pierwszej fazie, niezależnie od lokalnych schematów, zaprojektować globalny schemat oparty na analizie wymagań globalnej aplikacji żądającej dostępu do rozproszonych danych. Następnie schemat ten należy traktować jako „cel”, do którego należy dążyć.

Poniżej zostały podane kolejne kroki jakie należy przedsięwziąć stosując się do tej metodologii. Jest to kombinacja podejścia zaproponowanego w systemie MultiStar i podejścia opisanego w poprzednim rozdziale.

1. Analiza wymagań globalnej aplikacji.
2. Konstrukcja globalnego modelu ERD (*ang. Entity – Relationship Diagram*).
3. Specyfikacja modeli ERD dla wszystkich lokalnych baz danych.
4. Identyfikacja synonimicznych nazw w lokalnych schematach. Likwidacja różnic w nazewnictwie.
5. Dopasowanie atrybutów i kategorii. Jeżeli jest to konieczne, konwersja atrybutów na kategorie lub odwrotnie. Każda lokalna kategoria/trybut powinna mieć swoje odzwierciedlenie w globalnych kategoriach/trybutach.
6. Zespolenie zmodyfikowanych lokalnych modeli ERD, poprzez nałożenie ich, w celu uformowania drugiego schematu globalnego. Na tym etapie wszystkie synonimy i homonimy powinny być zidentyfikowane.
7. Identyfikacja związków generalizacji. Wprowadzenie związków generalizacji.
8. Porównanie dwóch uzyskanych schematów globalnych. Jeśli to konieczne, likwidacja różnic.

W procesie dopasowywania lokalnych schematów bardzo ważne jest zbieranie wyczerpujących informacji o przeprowadzonych transformacjach, potrzebnych do konstrukcji schematów pomocniczych. Może okazać się, że np. że dwie bazy danych przechowują tę samą informację, ale robią to na dwa różne sposoby. Okazuje się, że odszukanie wszystkich synonimów w bazach lokalnych, nie polega tylko na zauważeniu, że ten sam atrybut ma dwie różne nazwy.

Przeanalizujmy projektowanie globalnego schematu na następującym przykładzie.

Przykład 2

Rozpatrzmy dwie proste bazy danych, zawierające informacje o odlotach i przylotach samolotów. Jedna z baz znajduje się w lokalizacji X, druga w Y. Dla uproszczenia rozważań obie bazy składają się tylko z jednej relacji:

X (*NrLotu*, *Port*, *Przyl/Odl*, *Data*, *CzasOdlotu*, *CzasPrzylotu*);

Y (*NrLotu*, *Skqd*, *Dokqd*, *Data*, *CzasOdlotu*, *CzasPrzylotu*);

W lokalizacji X atrybut *Przyl/Odl* zawiera informacje, czy samolot rozpoczyna lot od lokalizacji X, czy też lokalizacja X jest punktem docelowym lotu.

Jeżeli *Przyl/Odl* = 'Przyl', to wartość atrybutu *Port* jest portem, z którego lot się rozpoczął, zaś lokalizacja X jest celem lotu.

Jeżeli *Przyl/Odl* = 'Odl', to wartość atrybutu *Port* jest celem lotu, zaś lokalizacja X jest portem macierzystym lotu.

Co do bazy w lokalizacji Y atrybuty *Skqd* i *Dokqd* oznaczają odpowiednio miejsce początku i końca lotu. Przykładową zawartość obu relacji przedstawiają poniższe tabele.

Lokalizacja X

<i>NrLotu</i>	<i>Port</i>	<i>Przyl/Odl</i>	<i>Data</i>	<i>CzasOdlotu</i>	<i>CzasPrzylotu</i>
XY123	A	Odl	01.06.2002	14:10	14:40
PQ781	C	Przyl	04.07.2002	10:50	12:15
AN322	Y	Przyl	05.07.2002	12:15	15:10

Lokalizacja Y

<i>NrLotu</i>	<i>Skqd</i>	<i>Dokqd</i>	<i>Data</i>	<i>CzasOdlotu</i>	<i>CzasPrzylotu</i>
XZ987	Y	W	10.06.2002	12:15	15:00
AS781	C	Y	04.07.2002	15:50	18:00
AN322	Y	X	05.07.2002	12:15	15:10

Jako globalną relację *INFO_LOTY*, zawierającą dane o lotach w obu lokalizacjach, przyjmujemy relację o zbiorze atrybutów takim samym jak relacja w lokalizacji Y.

INFO_LOTY(*NrLotu*, *Skqd*, *Dokqd*, *Data*, *CzasOdlotu*, *CzasPrzylotu*);

Aby utworzyć relację *INFO_LOTY*, nie można użyć operacji UNION na relacjach X i Y. Należy wyspecyfikować reguły, przechowywane w schemacie pomocniczym w lokalizacji X, które pozwalałyby na przekonwertowanie danych z relacji X na postać relacji globalnej.

Poniżej została przedstawiona procedura dokonująca takiej konwersji – procedura przekładu danych tabeli X i Y na dane globalnej relacji INFO_LOTY.

```

begin
  do while not end_of_relation X
    INFO_LOTY.NrLotu = X.NrLotu
    INFO_LOTY.Data = X.Data
    INFO_LOTY.CzasOdlotu = X.CzasOdlotu
    INFO_LOTY.CzasPrzylotu = X.CzasPrzylotu
    If X.Przy/Odl = 'Przyl'
      then begin
        INFO_LOTY.Skąd = X.Port
        INFO_LOTY.Dokąd = 'X'
      end
    else begin
      INFO_LOTY.Skąd = 'X'
      INFO_LOTY.Dokąd = X.Port
    end
  end-if
  next tuple
end-do
INFO_LOTY = INFO_LOTY UNION Y
end

```

■

Jak widać konwersja danych z relacji lokalnych na format relacji globalnej nie jest rzeczą prostą. Inny problem, może wynikać z faktu, że poszczególne bazy przechowują te same dane. Na przykład w sytuacji, gdy między dwoma szpitalami zostanie przewieziony karetką pacjent, informacja o tym zostanie odnotowana w dwóch bazach lokalnych. Przy prostym połączeniu danych z obu baz ta sama informacja pojawi się dwukrotnie.

4. INTEGRACJA DANYCH W HETEROGENICZNYCH BAZACH DANYCH NIE POSIADAJĄCYCH GLOBALNEGO SCHEMATU

W dużych, skomplikowanych środowiskach heterogenicznych baz danych konstrukcja scalającego, globalnego schematu okazuje się być zbyt skomplikowana i kosztowna. Konstruuje się globalny schemat tylko wówczas gdy niezbędna jest ścisła integracja lokalnych baz danych. Istnieją systemy rozproszonych baz danych, które

nie wymagają takiej integracji. Należą do nich MDBMS o swobodnej integracji. Projektowanie logiczne dla takich systemów sprowadza się do projektowania schematów eksportowych. Schematy eksportowe reprezentują tylko pewien widok zdefiniowany tylko na części rozproszonej bazy danych. Rozpatrując metodologię projektowania przedstawioną powyżej, projektowania schematów eksportowych dotyczy pierwszych pięciu faz.

W systemach o jeszcze mniejszym stopniu integracji, gdzie nie stosuje się schematów eksportowych, nie istnieje problem projektowania jakichkolwiek form globalnych widoków. W takich systemach użytkownikowi udostępnia się odpowiedni język, jak MSQL rozwinięty przez Litwina, który umożliwia bieżącą definicję własnych widoków.

5. OBIEKTOWO ZORIENTOWANE PODEJŚCIE W PROJEKTOWANIU GLOBALNEGO SCHEMATU BAZY DANYCH

W ostatnich latach powstało wiele prac zajmujących się wykorzystaniem technik obiektowych w projektowaniu schematu globalnego. Fundamentalną zaletą podejścia obiektowego w modelowaniu danych jest fakt, że dane i procesy operujące na danych (nazywane metodami) są projektowane równocześnie. W konwencjonalnym projektowaniu (strukturalnym) projektowanie struktur danych oraz procesów odbywa się w dużym stopniu niezależnie. Takie podejście odzwierciedla większość współczesnych DBMS, gdzie programy aplikacyjne są używane w sposób niezależny.

Zalety podejścia zorientowanego obiektowo w projektowaniu schematu globalnego można podsumować w następujących punktach:

1. Obiektowy model danych jest bogatszy semantycznie, np. w naturalny sposób pozwala na konstrukcję związków generalizacji i agregacji.
2. Metody mogą zostać wykorzystane do zapewnienia sprawnej integracji danych między poziomem lokalnym i globalnym.
3. Więzy integralności danych zdefiniowane na poziomie lokalnym, mogą być przekazywane w postaci metod do poziomu globalnego.
4. Integracja multimedialnych baz danych staje się możliwa do przeprowadzenia, dzięki możliwości rozszerzenia typów danych. Tak więc, możliwe staje się integrowanie map przechowywanych w postaci obrazów wraz z informacjami szczegółowymi ich dotyczącymi, w postaci tradycyjnej relacyjnej bazy danych.

Obiektowe bazy danych zaczynają cieszyć się coraz większym zainteresowaniem. Technologia ta w ostatnich latach znacząco się rozwinęła. Powstają standardy dotyczące modeli danych, języków zapytań. Obiektowo zorientowane podejście do integracji schematów okazuje się obiecującym rozwiązaniem. Za wadę można uznać

fakt, że konstrukcja „obiektoowo zorientowanych” widoków może wymagać wielokrotnego wykonania operacji złączenia, aby dane przechowywane w postaci relacji można było przełożyć na obiekty.

6. PODSUMOWANIE

W przypadku projektowania multibaz danych pozostaje jeszcze wiele do zrobienia. Ostatnie prace skupiły się głównie nad wyborem modelu danych, który mógłby w lepszym stopniu odzwierciedlić semantykę lokalnych baz danych. Dobrym rozwiązaniem w najbliższej przyszłości wydaje się być zastosowanie do tworzenia schematu integrującego, podejścia obiektoowo zorientowanego, w którym model danych jest znacznie bogatszy semantycznie niż model relacyjny. Należy jednak pamiętać, że proces transformacji danych z jednego modelu na drugi nie jest wykonywany jednokrotnie. Dane są w dalszym ciągu fizycznie przechowywane w postaci tabel. Schemat obiektowy jest tylko swego rodzaju interfejsem dla systemu globalnego. Konieczność przełożenia danych z postaci tabel na obiekty istnieje za każdym razem gdy żądany jest dostęp do nich. Proces ten jest niezwykle kosztowny, gdyż między innymi wymaga wielokrotnego przeprowadzania operacji złączeń relacji. Zastosowanie obiektowego schematu globalnego w multibazach danych wymaga stworzenia dodatkowych mechanizmów optymalizujących proces transformacji modeli.

LITERATURA

- [BellGr94] Bell D., Grimson J., *Distributed Database Systems*, Addison-Wesley Publishing Company, 1994.
- [DaHw84] Dayal U., Hwang H., *View Integration and Generalisation for Database Integration in a Multidatabase*, IEEE Trans.on Software Engineering, 10(6), 1984.
- [MaMa01] Majkowski A., Mazur Z., *Problemy integracji heterogenicznych baz danych*, Prace Naukowe WZI PWr., Zeszyt 2, Wrocław, 2001.

DESIGN OF MULTIDATABASE SCHEMES

The design of DDB still seems to be one of unsolved issue. The present work is a study about knowing methodologies of DDB design and problems, which concern them. The way of distribution data has a significant impact on data processing effectiveness in DDMMS. That way, there will be presented some aspect of distributed data processing in DDBMS.

Dariusz KRÓL*
Łukasz KOJS*

WYSZUKIWANIE DANYCH W OBIEKTOWEJ BAZIE DANYCH

W pracy przedstawiono metody wyszukiwania danych w obiektowej bazie danych. Na ilustrację problemu wybrano obiektowy system zarządzania bazą danych ObjectStore, najbardziej znany na rynku systemów obiektowych. Opisano funkcje wyszukiwawcze: `query()`, `query_pick()` oraz `exists()` zarówno w kontekście zapytań prostych i złożonych oraz pytań standardowych i dwustopniowych. Przeprowadzono 15 eksperymentów, wyjaśniających mechanizmy opisanych funkcji wyszukiwawczych. Uwzględniono indeksowanie za pomocą B-drzewa i funkcji haszującej. Wykorzystanie wyników pracy na etapie realizacji dowolnego systemu obiektowego pozwala na optymalny dobór metod wyszukiwawczych.

1. WPROWADZENIE

Celem pracy jest rozpoznanie i zbadanie sposobów wyszukiwania w obiektowej bazie danych ObjectStore. Zdecydowano się na wybór systemu zarządzania obiektową bazą danych ObjectStore, gdyż jest on przeznaczony do tworzenia dużych, rozproszonych baz danych opartych o technologię obiektową i technologię komponentów. Ponadto system ObjectStore ma obecnie ponad 11-letnią historię i największy, prawie 40% procentowy udział w rynku sprzedanych obiektowych systemów baz danych [5].

W odróżnieniu od baz relacyjnych, obiektowe bazy danych nie posiadają dotychczas zdefiniowanego standardu [1, 7]. Konsekwencją tego są liczne problemy, na które natrafiają projektanci i użytkownicy obiektowych baz danych. Niedogodności te, w przypadku wyszukiwania w obiektowej bazie danych, możemy scharakteryzować jako:

* Politechnika Wrocławska, Zakład Systemów Informacyjnych, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27

- a) problem przetwarzania wyników zapytania,
- b) problem hierarchii dziedziczenia i hierarchii agregacji w zapytaniach,
- c) problemy indeksowania.

Ad a). Rezultatem zapytania w przypadku baz obiektowych może być obiekt lub zbiór obiektów. Takie założenie jest kosztowne i wiele systemów narzuca rozmaite ograniczenia na operacje dotyczące używania takich klas. Typowym ograniczeniem jest założenie, że wszystkie otrzymane w rezultacie atrybuty muszą pochodzić z każdego z wyszukanych obiektów, czyli, że może to być albo sam obiekt albo pojedynczy atrybut. W ten sposób rezultatem wyszukiwania jest zbiór obiektów należących do istniejących już klas. Innym rozwiązaniem jest potraktowanie odpowiedzi jako zbioru obiektów należących do klasy ogólnej, która akceptuje wszystkie obiekty i jej jedynymi metodami są metody służące do drukowania bądź wyświetlania obiektów [2].

Ad b). Wprowadzenie w systemach obiektowych zależności hierarchicznych powoduje rozmaite problemy przy definiowaniu języka zapytań. Zwykle bowiem predykaty składające się na pytanie dotyczą atrybutów znajdujących się na różnych poziomach w hierarchii obiektów, istotne jest wtedy stworzenie łatwych mechanizmów nawigowania w takiej strukturze. Dla obiektowych języków trzeba dokonać rozróżnienia między operacjami *implicit join* – które będą odwoływały się do zależności hierarchicznych agregacji istniejących pomiędzy obiektami złożonymi i *explicit join* – w którym następuje zwykle porównywanie dwóch obiektów (czy to poprzez równość identyfikatorów czy równość wartości). Innym sposobem spojrzenia na operacje *implicit join* i *explicit join* – sposobem prowadzącym do lepszej klasyfikacji zapytań – jest rozróżnienie między zapytaniami z operandami pojedynczymi i wielokrotnymi. Zapytania używające pojedynczych operandów wyszukują zbiór obiektów należących do pojedynczych klas, podczas gdy zapytania z operandami wielokrotnymi reprezentują operacje, które można porównać z relacyjnym *join* i z operacjami na zbiorach. Mówiąc o operacjach złączenia warto jeszcze wspomnieć o jednym problemie. Oprócz zwykłych atrybutów jednowartościowych w systemach obiektowych powszechne jest występowanie atrybutów wielowartościowych modelowanych albo przez nadanie odpowiedniej cechy danemu atrybutowi obiektu, albo też poprzez stosowanie konstruktorów np. *set*. Zapytania powinny również umożliwiać odwoływanie się do takich atrybutów. Inne problemy niesie występowanie w bazach obiektowych zależności hierarchicznych dziedziczenia. Zakresem wyszukiwania pytania może być bowiem tylko pojedyncza klasa albo też klasa wraz z jej podklasami. W większości języków dostępne są obie możliwości [4].

Ad c). Analizując wykorzystywanie w bazach obiektowych indeksów trzeba zauważyć, że istnienie hierarchii agregacji i dziedziczenia powoduje konieczność wprowadzenia indeksów w stosunku do atrybutów zagnieżdżonych. Użycie takich

indeksów ma znaczący wpływ na koszty optymalizacji zapytań, a więc także ma wpływ na sam proces przetwarzania zapytań. Podobnie jak w przypadku baz relacyjnych w przetwarzaniu zapytań mogą być pomocne dane statystyczne takie jak liczba krotek w relacji czy liczba różnych wartości dla każdego atrybutu. Dla baz obiektowych statystyki takie muszą być rozszerzone o takie informacje jak liczba obiektów dla każdej z klas wzdłuż hierarchii dziedziczenia i liczba stron zawierających takie obiekty. Indeksowanie jest ważną metodą przyspieszania wykonania zapytania w obiektowych bazach danych. Jednakże z powodu złożoności obiektowego modelu danych wprowadzono trzy nowe typy indeksów [3]: indeks hierarchii klas, indeks atrybutu zagnieżdżonego oraz indeks dwuwymiarowy. Indeks hierarchii klas jest zakładany na atrybucie klas w hierarchii klas, a indeks atrybutu zagnieżdżonego odnosi się do hierarchii kompozycji klas. Indeks dwuwymiarowy jest indeksem atrybutu zagnieżdżonego powiększonym o indeks hierarchii klas dla każdej klasy z sekwencji klas między klasą indeksowaną a klasą, do której indeksowany atrybut aktualnie należy.

W podejściu obiektowym wyszukiwanie danych dotyczy zarówno języka zapytań oraz systemu nawigacyjnego obiektów [11]. Jedynym systemem nawigacji w ObjectStore jest kursor dla kolekcji obiektów [9]. Natomiast język zapytań jest bardzo rozbudowany i złożony, dlatego też główny nacisk położono na zbadanie jego możliwości i efektywności. W tym celu zbadano eksperymentalnie: sposoby zadawania zapytań prostych i zapytań złożonych, sposoby zarządzania indeksami oraz wpływ indeksowania na efektywność przetwarzania zapytań.

2. ZAPYTANIA W OBIEKTOWEJ BAZIE DANYCH OBJECTSTORE

System ObjectStore jest obiektowym systemem zarządzania bazą. Dane przekazywane pomiędzy pamięcią bazy danych a pamięcią programu są w sposób całkowicie automatyczny, zupełnie nie widoczny dla użytkownika. Programiście dano wygodne narzędzie do budowy złożonych struktur danych w postaci bibliotek sparametryzowanych klas realizujących kolekcje obiektów [10]. Do wyboru są listy, zbiory z powtarzającymi się elementami, czyli wielozbiory i zwykłe zbiory. Ciekawostką jest fakt, że użytkownik może opcjonalnie podać oszacowanie zamierzonych częstości wykonywania podstawowych operacji na kolekcji, na podstawie czego system może automatycznie dobrać najbardziej korzystną reprezentację, np. w postaci B-drzew lub tablic rozproszonych. Kompilator języka C++ w systemie ObjectStore zapewnia możliwość iteracji po tak zdefiniowanych kolekcjach.

Zapytanie w rozszerzonym języku C++ systemu ObjectStore stanowi wyrażenie wiążące jedną lub wiele kolekcji i predykat selekcji. Rezultatem zapytania jest kolekcja lub wskazanie obiektu. Predykat selekcji może być wyrażeniem, więc także innym zapytaniem, dzięki czemu można budować bardziej złożone zapytania. Semantyka zapytania jest następująca: wynikiem jest kolekcja złożona z elementów wejściowej kolekcji, dla których predykat selekcji, czyli wyrażenie zawarte w nawiasach [: :], jest prawdziwy (wartość niezerowa). Dopuszczalne są zapytania zagnieżdżone, w których wynik wewnętrznego zapytania stanowi predykat selekcji lub jego część w zapytaniu zewnętrznym.

Wszystkie zapytania w systemach bazodanowych możemy podzielić na zapytania proste i zapytania złożone. Grupa zapytań prostych ma na celu wyszukanie w bazie danych obiektów odwołując się bezpośrednio tylko do jednej kolekcji obiektów. Pytania złożone są to pytania, które w warunku selekcji łączą w sobie co najmniej dwie kolekcje obiektów.

W ObjectStore istnieją dwie zasadnicze metody zadawania pytań do bazy danych. Są to:

- metoda pytań standardowych funkcją **query()**, **query_pick()** oraz **exists()**,
- metoda pytań dwustopniowych funkcją **query()**, **query_pick()**, **exists()**.

W dalszej części zostaną zaprezentowane dwie powyższe metody zapytań w zależności od złożoności pytania. Najpierw zostaną omówione zapytania proste dla metody pytań standardowych i metody pytań dwustopniowych, a następnie pytania złożone dla obu tych metod z uwzględnieniem optymalizacji zapytań poprzez wprowadzanie indeksowania.

2.1. ZAPYTANIE PROSTE – METODA PYTAŃ STANDARDOWYCH FUNKCJĄ **query()**

W celu wyszukania kolekcji elementów spełniających zadany warunek, należy zastosować funkcję **os_Collection::query()**. Przykład zastosowania tej funkcji przedstawia się w następujący sposób:

```
os_database *db;
os_Set<faktura*> *faktura_set;
...
os_Set<faktura*> &szukane_faktury =faktura_set->query(„faktura*”, „cena > 100”,db);
```

Pytanie 1. Znajdź wszystkie faktury, których cena jest większa niż 100

Warunek selekcji w zapytaniu może także zawierać wartości typu znakowego z tym ograniczeniem, że mogą one być przekazywane za pomocą funkcji **strcmp()** i **strcoll()**.

```
os_database *db;
os_Set<zadanie*> *zadanie_set;
...
os_Set<zadanie*> &szukane_zadanie = zadanie_set-> ("zadanie*", "strcmp(opis_zadania,
\"Banki\")= =0", db);
```

Pytanie 2. Znajdź wszystkie zadania, które w opisie zawierają ciąg Banki

Przykład takiego zapytania przedstawiony jest w postaci pytania 2 i 3. Jak widać warunek selekcji dla pytania 2 zawiera wartość typu znakowego, jednak całość poddawana jest konwersji do wartości całkowitej poprzez funkcję **strcmp()**, która to służy do porównywania wartości atrybutu **opis_zadania** z ciągiem „Banki”.

```
os_database *db;
os_Set<zadanie*> *zadanie_set;
...
szukany_wykonawca = wykonawca_set->query("wykonawca*", "strcmp(imie, \"Agnieszka\") == 0",
db);
```

Pytanie 3. Znajdź wszystkich wykonawców, którzy mają na imię Agnieszka

Pytania proste mogą przyjmować formę bardziej złożoną, kiedy to warunek selekcji odwołuje się do więcej niż jednego atrybutu z danej klasy obiektów. Wówczas warunek selekcji zapytania może zawierać operatory logiczne **AND** (&&), **OR** (||) oraz **NOT** (!). Przykład zastosowania takiego warunku złożonej selekcji dla zapytań prostych przedstawiony jest w postaci pytania 4.

```
os_database *db;
os_Set<faktura*> *faktura_set;
...
os_Set<faktura*> &szukane_faktury = faktura_set->query(„faktura*", „(((cena > 100) && (cena <
1000)) || (cena == 2000)) && strcmp(data, \"21\01\2000\")==0", db);
```

Pytanie 4. Znajdź wszystkie faktury, których cena jest większa niż 100 i mniejsza od 1000 lub równa 2000 z dnia 21 stycznia 2000 roku

2.2. ZAPYTANIE PROSTE – METODA PYTAŃ STANDARDOWYCH FUNKCJĄ `query_pick()`

Funkcja `query_pick()` w odróżnieniu od omówionej wyszukuje pojedyncze obiekty w bazie danych a uściślając, pierwszy obiekt spełniający warunki selekcji zapytania. Zastosowanie tej funkcji przedstawione jest na przykładzie pytania 5 i pytania 6.

```
os_database *db;
os_Set<faktura*> *faktura_set;
...
faktura* szukana_faktura =faktura_set->query_pick(„faktura*”, „cena > 100”, db);
```

Pytanie 5. Znajdź pierwszą fakturę, której cena jest większa niż 100

```
os_database *db;
os_Set<wykonawca*> *wykonawca_set;
...
wykonawca *szukany_wykonawca = wykonawca_set-> query_pick("wykonawca*", "(strcmp(imie,
\"Agnieszka\") == 0) && (strcmp(zakres_obowiazkow, \"Programowanie strukturalne\")==0)", db);
```

Pytanie 6. Znajdź pierwszego wykonawcę, który ma na imię Agnieszka oraz którego zakres obowiązków to „Programowanie strukturalne”

2.3. ZAPYTANIE PROSTE – METODA PYTAŃ STANDARDOWYCH FUNKCJĄ `exists()`

Zadając pytanie egzystencjalne należy posłużyć się funkcją `os_Collection::exists()`, która w odróżnieniu od `query_pick()` i `query()` zwraca niezerową wartość typu `int` w przypadku, gdy dany obiekt znajduje się w kolekcji, oraz wartość 0 w przypadku przeciwnym (patrz pytanie 7).

```
os_database *db;
os_Set<faktura*> *faktura_set;
...
if( faktura_set->exists(„faktura*”, „cena > 100”, db))
print_faktura();
```

Pytanie 7. Czy istnieje faktura, której cena jest większa niż 100?

2.4. ZAPYTANIE PROSTE – METODA PYTAŃ DWUSTOPNIOWYCH FUNKCJĄ `query()`

Podstawową niedogodnością stosowania pytań standardowych jest to, że użytkownik nie ma możliwości przekazywania parametrów do zapytania. Zmniejsza to niewątpliwie ich przydatność w wyszukiwaniu informacji przez użytkownika systemu. Projektant nie zawsze może przewidzieć o co będzie pytał użytkownik.

Aby wyeliminować ograniczenia w definiowaniu zapytań należy zaproponować, aby proces wykonywania zapytania składał się z następujących etapów:

1. Analiza wyrażenia selekcji.
2. Łączenie odwołań do zmiennych i funkcji w zapytaniu.
3. Interpretacja zapytania (wykonanie zapytania ograniczającego).

System ObjectStore umożliwia projektantowi bazy danych i aplikacji zastosowanie takiego schematu wykonania zapytania, nazywanego dalej pytaniem dwustopniowym. W tym celu należy stworzyć obiekt zapytania jako instancję klasy `os_coll_query`. Zapytanie takie będzie analizowane w chwili jego tworzenia. Przetwarzanie pytania obejmuje analizę wyrażenia selekcji, łączenie zmiennych i funkcji oraz wskazanie kolekcji, na której zostanie wykonane zapytanie. W przypadku pytań dwustopniowych pytanie takie jest przetwarzane tylko jeden raz w momencie jego tworzenia, a przy próbie kolejnego użycia jest interpretowane.

Definicję pytania dwustopniowego tworzy się przy użyciu statycznej funkcji: `os_coll_query::create()`. Funkcja ta zwraca wartość typu `const os_coll_query&`. Tworząc zapytanie projektant bazy sytemu bazy danych ma do dyspozycji nie tylko atrybuty, których wartości są typu całkowitego, czy znakowego, ale może wykorzystać również własne zdefiniowane funkcje i metody (patrz pytanie 8, 9 i 10).

```
const os_coll_query &definicja_zapytania_faktur =
os_coll_query::create(„faktura*”, „cena > *(int*) def_min_cena”, db);
```

Pytanie 8. Znajdź wszystkie faktury, których cena jest większa od podanej

```
const os_coll_query &definicja_zapytania_zadania=
os_coll_query::create(„zadanie*”,
„(strcmp(opis_zadania, *(char**) def_opis) ==0), db);
```

Pytanie 9. Znajdź wszystkie zadania, których opis zawiera podany przez użytkownika ciąg znaków

```
const os_coll_query &definicja_zapytania_wykonawcy =
os_coll_query::create(„wykonawca*”,
„(strcmp(imie, *(char**) def_imie) == 0), db);
```

Pytanie 10. Znajdź wszystkich wykonawców o podanym przez użytkownika imieniu

2.5. ZAPYTANIE PROSTE – METODA PYTAŃ DWUSTOPNIOWYCH FUNKCJĄ `query_pick()`

Definicję pytania dwustopniowego dla funkcji `query_pick()` tworzy się przy użyciu statycznej funkcji: `os_coll_query::create_pick()`, np.

```
const os_coll_query &definicja_zapytania_faktur =
os_coll_query::create_pick(„faktura*”, „cena > *(int*) def_min_cena”, db);
int min_cena = 1000;
os_bound_query zapytanie_faktur(
definicja_zapytania_faktur,
(os_keyword_arg(„def_min_cena”, &min_cena)));
faktura* szukana_faktura = faktura_set->query_pick(zapytanie_faktur);
```

Pytanie 11. Znajdź pierwszą fakturę, której cena jest większa od podanej przez użytkownika

2.6. ZAPYTANIE PROSTE – METODA PYTAŃ DWUSTOPNIOWYCH FUNKCJĄ `exists()`

Definicję pytania dwustopniowego dla funkcji `exists()` tworzy się przy użyciu statycznej funkcji: `os_coll_query::create_exists()`, np.

```
const os_coll_query &definicja_zapytania_faktur =
os_coll_query::create_exists(„faktura*”, „cena > *(int*) def_min_cena”, db);
int min_cena = 1000;
os_bound_query zapytanie_faktur(
definicja_zapytania_faktur,
(os_keyword_arg(„def_min_cena”, &min_cena)));
if (faktura_set->exists(zapytanie_faktur)) print_faktura();
```

Pytanie 12. Czy w bazie danych znajduje się faktura, której cena jest większa od podanej przez użytkownika?

2.7. ZAPYTANIE ZŁOŻONE – METODA PYTAŃ STANDARDOWYCH FUNKCJĄ *query()*

Definicja zapytań złożonych w systemie ObjectStore, które w sposób jawny łączą w warunku zapytania obiekty różnych kolekcji, w wyniku których zwracana jest pojedyncza kolekcja obiektów przyjmuje postać:

warunek_kolekcji [: warunek_zapytania :]

gdzie **warunek_kolekcji** jest elementem warunku całkowitego wysokiego poziomu typu **os_Collection**, natomiast **warunek_zapytania** jest kryterium selekcji dla pytania zagnieżdżonego. Pytanie 13 jest przykładem takiego zapytania.

```
os_database *db;
klient_set->query("klient*", "faktura_klienta[: (cena >= 100) :] &&
oferta_klienta[:!strcmp(opis_oferty, \"Bank\"):]", db );
```

Pytanie 13. Znajdź wszystkich klientów, dla których ceny faktur większe są od 100, oraz oferta zawiera opis: Banki

Przykładem pytania zagnieżdżonego jest pytanie 14.

```
os_database *db;
szukane_zadania = zadanie_set->query(„zadanie*”, „podzadanie[:1:]”, db);
```

Pytanie 14. Znajdź wszystkie zadania posiadające podzadania

2.8. ZAPYTANIE ZŁOŻONE – METODA PYTAŃ STANDARDOWYCH FUNKCJĄ *query_pick()*

Na przykładzie funkcji *query_pick()* pokazane zostanie definiowanie zapytań zawierających niejawne złączenia klas. Obrazuje to pytanie 15. Niejawne złączenie w warunku selekcji następuje tutaj pomiędzy obiektami klasy **faktura** a obiektami klasy **klient**.

```
os_database *db;
faktura* szukana_faktura = faktura_set->query_pick( "faktura*", "strcmp(this-> klient_faktury-
>imie, \"Agnieszka\") = 0", db);
```

Pytanie 15. Znajdź pierwszą fakturę, której klient ma na imię Agnieszka

2.9. ZAPYTANIE ZŁOŻONE – METODA PYTAŃ STANDARDOWYCH FUNKCJĄ `exists()`

Warunek selekcji dla zapytań złożonych może być także warunkiem złożonym, w którym poszczególne człony selekcji będą połączone ze sobą za pomocą operatorów logicznych. Przykład takiego zapytania przedstawiony jest w postaci pytania 16.

```
os_database *db;
if( faktura_set->exists( "faktura*", "(cena == 1000) && ( strcmp(this-> klient_faktury->imie
,\"Agnieszka\") == 0) && ( strcmp(data, \"21\201\2000\")==0)", db)) ...
```

Pytanie 16. Czy istnieje w bazie danych faktura, której cena jest równa 1000, a klient faktury ma na imię Agnieszka, z dnia 21 stycznia 2000?

2.10. ZAPYTANIE ZŁOŻONE – METODA PYTAŃ DWUSTOPNIOWYCH FUNKCJĄ `query()`

W przypadku metody pytań dwustopniowych, pytanie 13 przyjmie następującą postać:

```
const os_coll_query &definicja_zapytania_klient=
os_coll_query::create („klient*", "faktura_klienta[: (cena >= *(int*) m_numer1):] &&
oferta_klienta[:!strcmp(opis_oferty, *(char**) m_numer2):]", db);
int numer1 = 1000; char* numer2 = „Banki”;
os_bound_query zapytanie_klient(definicja_zapytania_klient,
(os_keyword_arg(„m_numer1”, &numer1),
(os_keyword_arg(„m_numer2”, &numer2)));
os_Set<klient*> &szukany_klient = klient_set->query(zapytanie_klient);
```

Pytanie 17. Znajdź wszystkich klientów, dla których ceny faktur większe są od 100 oraz oferta zawiera opis: Banki

2.11. ZAPYTANIE ZŁOŻONE – METODA PYTAŃ DWUSTOPNIOWYCH FUNKCJĄ `query_pick()`

Pytanie 15 z rozdziału 2.8 „Znajdź pierwszą fakturę, której klient ma na imię Agnieszka” dla metody pytań dwustopniowych przekształci się w następującą postać:

```
const os_coll_query &definicja_zapytania_faktura=
os_coll_query::create_pick(„faktura*”, „strcmp(this-> klient_faktury->imie , *(char**) m_numer1 )
== 0”, db);
char* numer1 = „Agnieszka”;
os_bound_query zapytanie_faktura(definicja_zapytania_faktura,
(os_keyword_arg(„m_numer1”, &numer1));
faktura *szukana_faktura = faktura_set->query_pick(zapytanie_faktura);
```

Pytanie18. Znajdź pierwszą fakturę, której klient ma na imię Agnieszka

Pytanie 16 jako zapytanie dwustopniowe wygląda następująco:

```
cs_coll_query::create(„faktura*”, „(cena == *(int *) m_numer1) && (strcmp(this->klient_faktury-
>imie, *(char **) m_numer2) ==0) && ( strcmp( data *(char**) m_numer3)==0)”, db);
```

Pytanie 19. Czy istnieje w bazie danych faktura, której cena jest równa podanej kwocie, a klient faktury ma dane imię, z dnia podanego przez użytkownika?

2.12. INDEKS ŚCIEŻKOWY

Podstawową metodą optymalizacji zapytań w obiektowej bazie danych ObjectStore jest dodawanie indeksów do kolekcji obiektów. Do tego celu służy funkcja **os_Collection::add_index()**. Dla przykładowego zapytania możemy zoptymalizować liczbę porównywania wartości w bazie danych, przez co zwiększy się efektywność wyszukiwania.

Ten rodzaj indeksu nazywa się **indeksem ścieżkowym**. Ostatnim argumentem funkcji **add_index()** może być baza danych, segment, klastery obiektów i konfiguracja w jakiej jest przechowywany indeks. Istniejący indeks może zostać usunięty jedynie za pomocą wywołania funkcji **drop_index()**.

System ObjectStore daje również projektantowi możliwość sprawdzenia, czy dana kolekcja obiektów ma już zdefiniowany indeks ścieżkowy. Służy do tego funkcja **has_index()**, która zwraca niezerową wartość typu **os_int32** w przypadku, kiedy kolekcja posiada już określony indeks oraz wartość 0, w przeciwnym przypadku.

Za pomocą funkcji **has_indeks()** możemy sprawdzić, czy kolekcja posiada indeks o wcześniej ustalonej wartości (kluczu).

Definiując indeks można określić jego własności, którymi są:

- porządek przechowywanych wartości,
- zapamiętywanie wartości powtarzających się,
- sygnalizowanie wartości powtarzających się,
- kopiowanie wartości klucza,

- ustalanie wskaźnika na wartość klucza,
- używanie referencji przy wartościowaniu klucza.

Wartość domyślna indeksu przechowywana jest w postaci tablicy haszującej, a indeks ten nazywa się wówczas indeksem nieuporządkowanym. Dla pytań, których wyrażenie zawiera operatory porównań takie jak: $<$, $<=$, $>=$, czy $>$, można wymusić uporządkowanie wartości kluczy indeksów, co znacznie przyspieszy wykonywanie takich zapytań. Taki indeks w postaci B-drzewa nazywa się indeksem uporządkowanym.

2.13. INDEKS TYPU `os_backptr`

Kolejnym sposobem optymalizacji zapytań oferowanym przez system ObjectStore jest definiowanie indeksu typu `os_backptr`. Jest to indeks, którego wartości oraz sposób organizacji mogą być całkowicie kontrolowane przez projektanta systemu bazodanowego. Ponadto dodatkową zaletą tego indeksu jest fakt, że za jego pomocą można nie tylko indeksować wartości atomowe atrybutów obiektów, ale również atrybuty, których dziedziną jest typem złożonym, choćby inną klasą obiektów.

W celu zdefiniowania indeksu typu `os_backptr` należy wprowadzić do definicji klasy obiektu, w którym będziemy chcieli indeksować atrybuty, dodatkowy atrybut typu `os_bacptr`, który musi poprzedzać definicje indeksowanych atrybutów i metod.

Kolejnym krokiem w definiowaniu tego typu indeksu jest dokładne wskazanie, które z wybranych do indeksowania atrybutów będą rzeczywiście indeksowane. Do tego celu służy makro `os_indexable_member()`.

Dostęp do indeksowanego ciała klasy obiektu jest realizowany za pomocą kolejnego makra `os_indexable_body()`, które powoduje, że wszystkie indeksowane wartości będą prawidłowo aktualizowane, a zawarte w definicji wywołania tego makra `os_index()` będzie odpowiedzialne za uaktualnianie indeksów.

Kolejnym etapem w definiowaniu indeksu jest zaprojektowanie dwóch, ewentualnie jednej, funkcji, według których będą rozmieszczane wartości indeksowanych atrybutów.

Ostatnim etapem tworzenia indeksu typu `os_bacptr` jest zdefiniowanie zwykłych indeksów ścieżkowych dla indeksowanych atrybutów w sposób opisany w poprzednim punkcie.

Największą korzyścią stosowania tego indeksu jest fakt, że projektant jest sam w stanie kontrolować wydajność systemu, oraz w miarę potrzeb tak zarządzać indeksowaniem atrybutów, by była ona jak największa. Ponadto, dla specyficznych danych przechowywanych w bazie danych budowa własnych funkcji haszujących dla indeksów jest niejednokrotnie jedynym prawidłowym rozwiązaniem.

3. OPIS EKSPERYMENTÓW

W celu dokładnego zbadania i porównania metod wyszukiwania danych w obiektowej bazie danych ObjectStore zaprojektowano i przeprowadzono 15 eksperymentów. Dla zapewnienia wiarygodności wyników wszystkie eksperymenty realizowano w tym samym środowisku systemowym, tj. Pentium II 233 MHz MMX, 32 MB RAM, Windows 95 OSR PL oraz Development Client/Server 5.1. Liczba obiektów w bazie danych, na której wykonywane były eksperymenty wynosi 84455 obiektów, natomiast liczba klas w schemacie testowej bazy wynosi 14.

Efektywność wyszukiwania we wszystkich eksperymentach rozumiana jest jako najkrótszy czas wyszukiwania kolejnych obiektów w bazie danych. Czas wyszukiwania obiektów mierzony jest przy pomocy standardowych funkcji języka C++, służących do pomiaru czasu systemowego. Za każdym razem mierzony jest czas całkowity wykonania wszystkich zapytań w obrębie danej transakcji, jak również czas wykonania zapytań cząstkowych składających się na całość eksperymentu a wyniki są wypisywane na ekranie monitora. W eksperymentach, w których wykonywane były zapytania metodą pytań ograniczających oraz w przypadku eksperymentów dotyczących indeksowania nie został zmierzony czas przygotowywania pytań a także czas dodawania indeksów do bazy danych. Najważniejszym badanym elementem jest bowiem efektywność wykonywania samych zapytań, nie zaś czas przygotowywania środowiska przez aplikację. Z punktu widzenia użytkownika środowisko bazy danych może być przygotowane wcześniej, nie zaś w momencie zadawania pytań.

Eksperymenty 1–9 składają się z czterech serii zapytań do trzech kolekcji obiektów: **faktura**, **zadanie** oraz **wykonawca**. Dla każdej serii zaprojektowana została odpowiednia aplikacja wyszukująca obiekty w sposób następujący:

- seria 1 – metodą pytań standardowych poprzez wartość atrybutu typu całkowitego; zapytania do kolekcji obiektów **faktura**,
- seria 2 – metodą pytań standardowych poprzez wartość atrybutu typu znakowego; zapytania do kolekcji obiektów **wykonawca**,
- seria 3 – metodą pytań dwustopniowych poprzez wartość atrybutu typu całkowitego; zapytania do kolekcji obiektów **faktura**,
- seria 4 – metodą pytań dwustopniowych poprzez wartość atrybutu typu znakowego; zapytania do kolekcji obiektów **zadanie**.

Wyszukiwanie danych przeprowadzono za pomocą funkcji **query()**, **query_pick()** oraz **exists()** w zależności od rodzaju eksperymentu. Definicje pytań dla serii 1 i 3 to pytania 1, 5, 7, 8, 11 i 12. Dla zapytań serii 2 użyto pytań 3 i 10. Dla serii 3 wykorzystano definicje pytań 2 i 9.

W przypadku eksperymentów, w których wykorzystano optymalizację wyszukiwania, aplikacje testowe wprowadzają do bazy danych indeksy ścieżkowe do

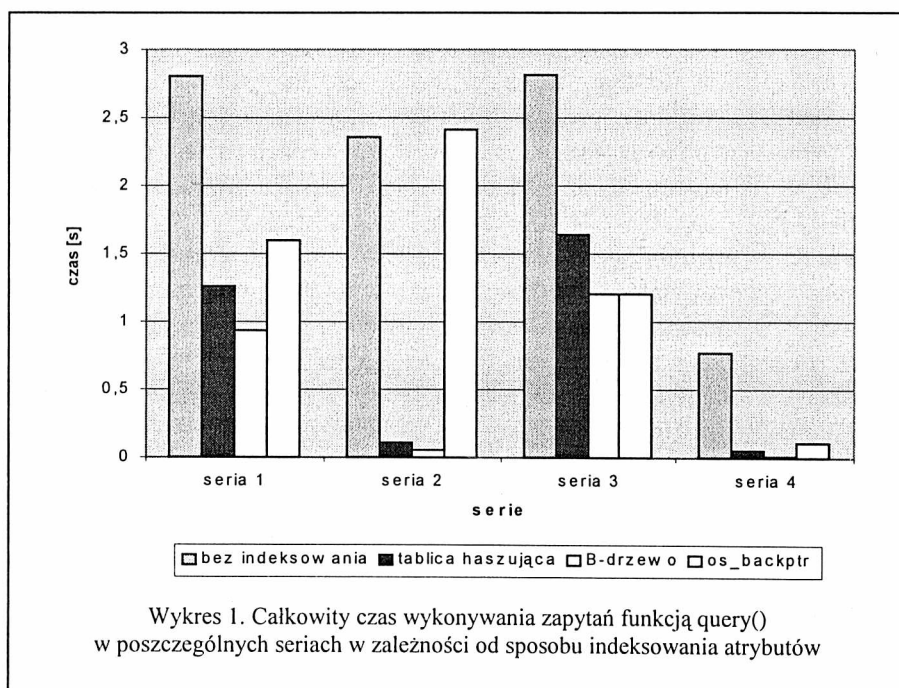
atrybutu **cena** obiektów klasy **faktura**, atrybutu **imie** klasy **wykonawca**, oraz atrybutu **opis_zadania** klasy **zadanie**.

Eksperymenty 10–15 składają się z dwóch serii zapytań do trzech kolekcji obiektów: **faktura**, **klient** oraz **oferta**. Dla każdej serii zaprojektowana została odpowiednia aplikacja wyszukująca obiekty w sposób następujący: seria 1 – metodą pytań standardowych, seria 2 – metodą pytań dwustopniowych.

Z powodu ograniczeń objętościowych nie zamieszczamy wykresów z poszczególnych eksperymentów. Poniżej znajdują się jedynie wykresy zbiorcze dla grup eksperymentów. Pełny opis eksperymentów można znaleźć w pracy [6].

EKSPERYMENTY 1–3

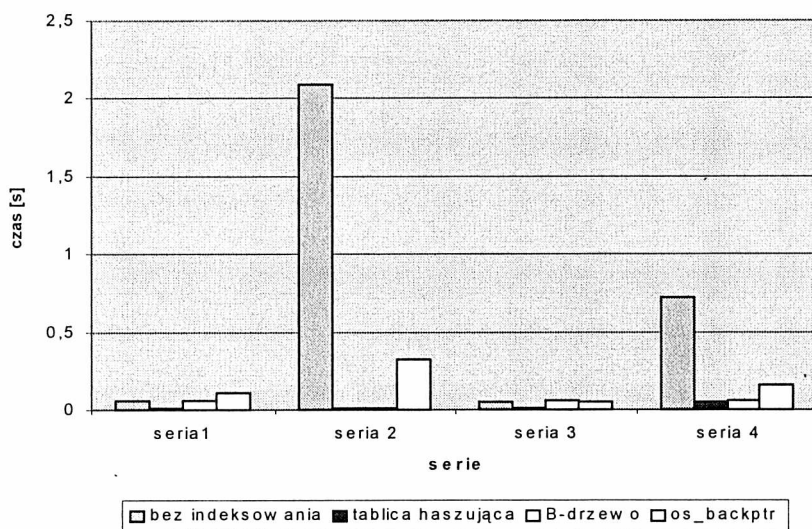
1. Badanie czasu wyszukiwania kolekcji obiektów za pomocą funkcji `query()` metodą pytań standardowych i pytań dwustopniowych.
2. Zbadanie efektywności indeksowania ścieżkowego w postaci tablicy haszującej i B-drzewa za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `query()`.
3. Zbadanie efektywności indeksowania typu `os_backptr` za pomocą metody pytań i metody pytań dwustopniowych funkcją `query()`.



Przeprowadzając analizę wyników z eksperymentów 1, 2 oraz 3 należy zauważyć, że największą efektywność systemu uzyskuje się w przypadku zastosowania indeksowania ścieżkowego w postaci B-drzewa dla każdej z badanych metod we wszystkich seriach. Ponadto indeksowanie typu **os_backptr** może w niektórych przypadkach spowodować nieefektywne działanie systemu, co obrazuje poniższy wykres 1 seria 2. We wszystkich przypadkach indeksowanie to jest najmniej efektywne.

EKSPERYMENTY 4–6

4. Badanie efektywności wyszukiwania pojedynczych obiektów za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `query_pick()`.
5. Zbadanie efektywności indeksowania ścieżkowego w postaci tablicy haszującej i B-drzewa za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `query_pick()`.
6. Badanie efektywności indeksowania typu `os_backptr` za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `query_pick()`.

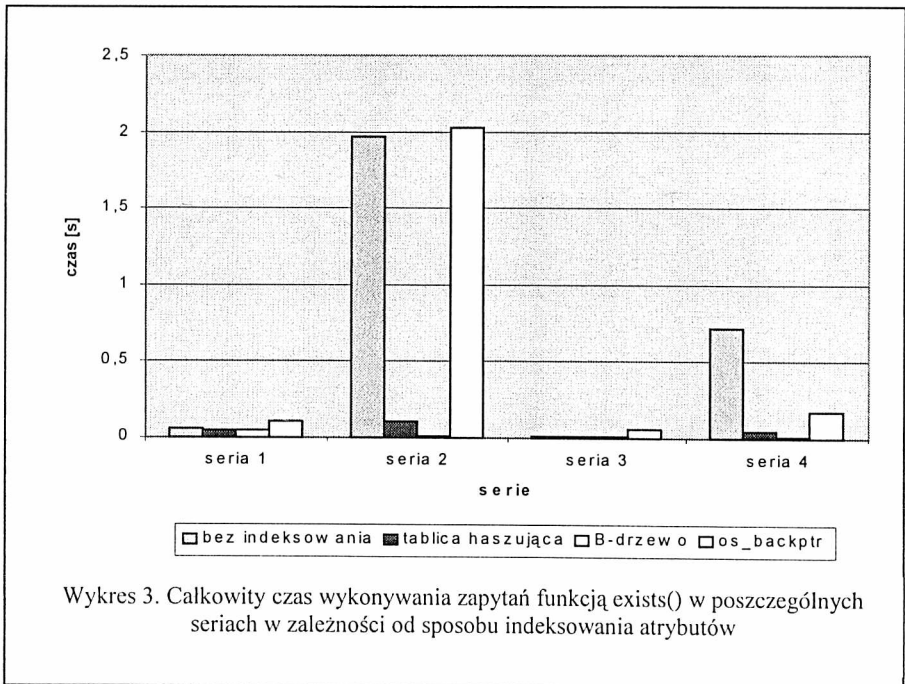


Wykres 2. Całkowity czas wykonywania zapytań funkcją `query_pick()` w poszczególnych seriach w zależności od sposobu indeksowania atrybutów

Wykres 2 jest podsumowaniem wyników otrzymanych w eksperymentach 4, 5 oraz 6. Przedstawiono czasy wykonywania poszczególnych zapytań funkcją `query_pick()` w zależności od sposobu indeksowania atrybutów. W przypadku wyszukiwania za pomocą tej funkcji bardzo nieefektywne staje się zadawanie pytań poprzez wartości całkowite. Wprowadzenie dla serii 1 indeksowania typu `os_backptr` spowodowało pogorszenie efektywności systemu, natomiast najkrótsze czasy wykonywania zapytań otrzymywano przy użyciu indeksowania w postaci tablicy haszującej. Polecaną metodą wyszukiwania jest metoda pytań dwustopniowych poprzez wartości całkowite.

EKSPERYMENTY 7-9

7. Badanie czasu wykonywania zapytań egzystencjalnych za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `exists()`.
8. Badanie czasu wykonywania zapytań egzystencjalnych za pomocą metody pytań dwustopniowych funkcją `exists()` z indeksowaniem ścieżkowym w postaci tablicy haszującej i B-drzewa.
9. Badanie czasu wykonywania zapytań egzystencjalnych za pomocą metody pytań dwustopniowych funkcją `exists()` z indeksowaniem typu `os_backptr`.

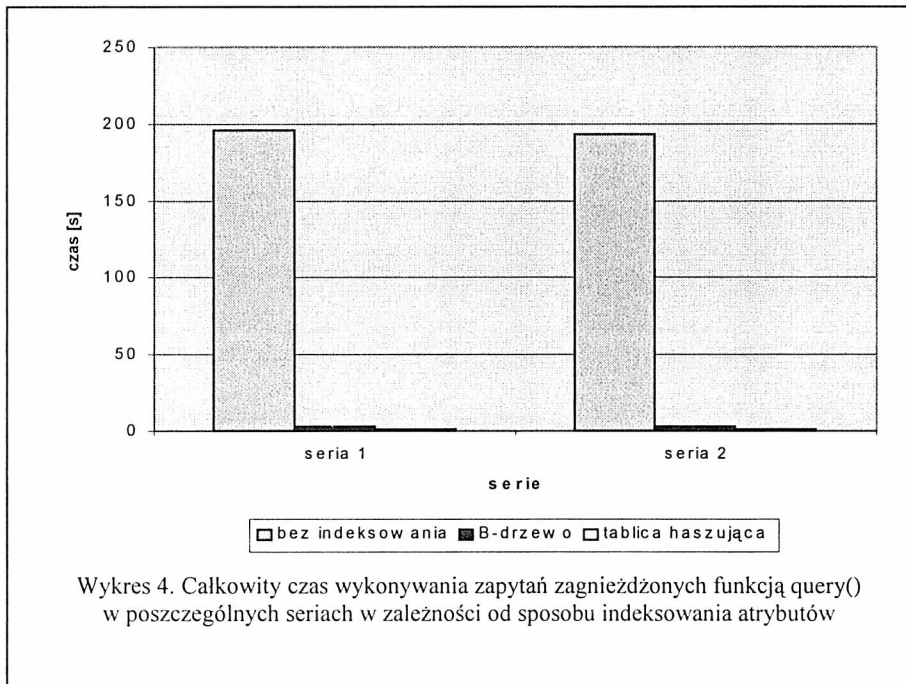


Analiza wyników (patrz wykres 3) potwierdza fakt, że preferowaną metodą wyszukiwania dla funkcji `exists()` jest wyszukiwanie poprzez wartości całkowite metodą pytań dwustopniowych z indeksowaniem ścieżkowym w postaci B-drzewa bądź też w postaci tablicy haszującej, gdyż otrzymane wyniki dla obu sposobów indeksowania różnią się w sposób nieznaczny.

EKSPERYMENTY 10–11

10. Badanie czasu wykonywania zapytań zagnieżdżonych za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `query()`.
11. Badanie czasu wykonywania zapytań zagnieżdżonych za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `query()` z indeksowaniem ścieżkowym w postaci tablicy haszującej i B-drzewa.

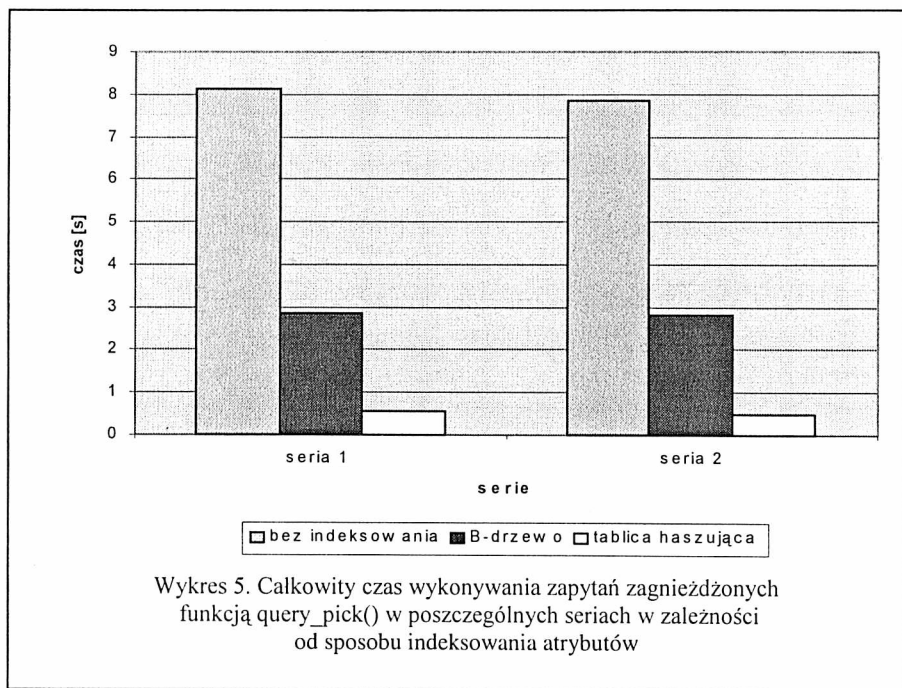
Najefektywniejszą metodą zapytań zagnieżdżonych jest metoda pytań dwustopniowych z indeksowaniem ścieżkowym atrybutów w postaci tablicy haszującej. Dokładne porównanie czasów realizacji zapytań z eksperymentów 10 oraz 11 znajduje się na wykresie 4.



EKSPERYMENTY 12–13

12. Badanie czasu wykonywania zapytań zagnieżdżonych za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `query_pick()`.
13. Badanie czasu wykonywania zapytań zagnieżdżonych za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `query_pick()` z indeksowaniem ścieżkowym w postaci tablicy haszującej i B-drzewa.

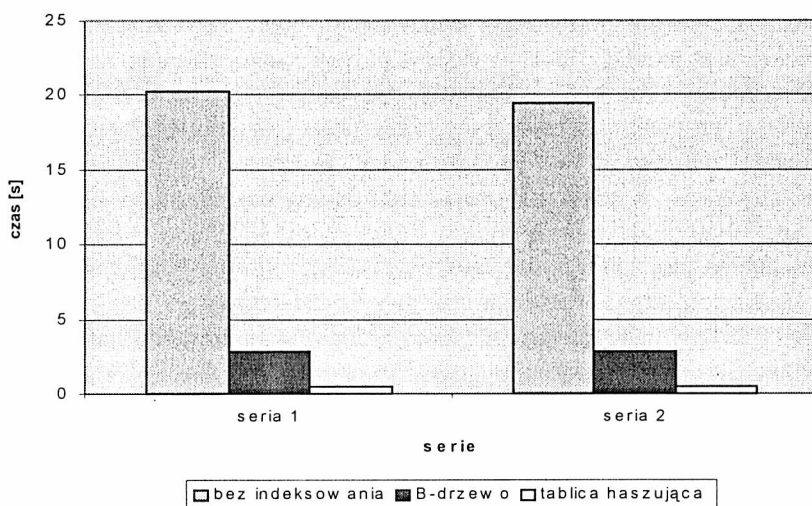
Podobnie jak w przypadku wyszukiwania za pomocą funkcji `query()` dla zapytań zagnieżdżonych preferowaną metodą jest metoda pytań dwustopniowych. W przypadku indeksowania ścieżkowego w postaci B-drzewa wykresy funkcji mają charakterystykę ściśle rosnącą, natomiast w przypadku indeksowania w postaci tablicy haszującej – nie malejącą. Potwierdza to wyniki otrzymane w eksperymencie 11, że w przypadku pytań zagnieżdżonych polecaną metodą będzie metoda pytań dwustopniowych z indeksowaniem ścieżkowym w postaci tablicy haszującej (patrz wykres 5).



EKSPERYMENTY 14–15

14. Badanie czasu wykonywania zapytań zagnieżdżonych za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `exists()`.
15. Badanie czasu wykonywania zapytań zagnieżdżonych za pomocą metody pytań standardowych i metody pytań dwustopniowych funkcją `exists()` z indeksowaniem ścieżkowym w postaci tablicy haszującej i B-drzewa.

Podobnie jak w przypadku wyszukiwania za pomocą funkcji `query_pick()` dla zapytań zagnieżdżonych polecaną metodą będzie metoda pytań dwustopniowych z indeksowaniem ścieżkowym w postaci tablicy haszującej. Na wykresie 6 przedstawiono całkowity czas realizacji zapytań zagnieżdżonych dla funkcji `exists()` w zależności od sposobu indeksowania atrybutów na podstawie eksperymentów 14 i 15.



Wykres 6. Całkowity czas wykonywania zapytań zagnieżdżonych funkcją `exists()` w poszczególnych seriach w zależności od sposobu indeksowania atrybutów

4. PODSUMOWANIE

System ObjectStore oferuje kilka metod pozwalających na sprawne wyszukiwanie informacji. W zależności od potrzeb mogą być to metody bazujące na zapytaniach prostych, które działają na pojedynczych kolekcjach obiektów, lub też zapytania złożone, w których istnieje możliwość łączenia ze sobą dwóch lub większej liczby kolekcji obiektów w warunku selekcji. Warto zaznaczyć, że złączenie to może przebiegać w sposób jawny lub niejawny.

W pracy wyróżniono dwie metody pytań: metoda pytań standardowych i metoda pytań dwustopniowych. Metoda pytań standardowych bardzo dobrze sprawdza się w przypadku wyszukiwania polegającego na budowaniu raportów, zestawień [8], w których warunek selekcji nie zmienia się oraz w przypadku pojedynczych zapytań. Metoda pytań dwustopniowych, która dla pytań pojedynczych jest nieefektywna z powodu dodatkowego czasu potrzebnego na zbudowanie definicji zapytania, dla wielokrotnych zapytań, lub zapytań, w których użytkownicy sami mogą definiować parametry selekcji, jest lepszym rozwiązaniem.

System ObjectStore umożliwia zadawanie pytań trzema funkcjami: **query()**, **query_pick()** oraz **exists()**, które w odpowiedzi zwracają kolejno: kolekcję obiektów, pojedyncze obiekty i wartość niezerową w przypadku, gdy obiekt znajduje się w bazie danych lub 0 w przypadku przeciwnym. Umożliwia także optymalizację zapytań przez wprowadzanie dwóch metod indeksowania. Pierwsza metoda to indeksowanie ścieżkowe atrybutów w postaci B-drzewa lub tablicy haszującej, w której system sam zarządza sposobem przechowywania i rozmieszczenia indeksów. Metoda druga polega na wprowadzeniu indeksu typu **os_backptr**, poprzez który użytkownik ma możliwość zarządzania wartościami i sposobem grupowania indeksów.

Wyniki z przeprowadzonych piętnastu eksperymentów pozwalają stwierdzić, że system ObjectStore jest bardzo wydajny.

W przypadku zapytań prostych (eksperymenty 1–9) stwierdzono, że:

- przy wyszukiwaniu kolekcji obiektów najefektywniejszym sposobem jest indeksowanie ścieżkowe w postaci B-drzewa dla pytań dwustopniowych stosując wartości znakowe,
- przy wyszukiwaniu pojedynczych obiektów najefektywniejsze jest indeksowanie ścieżkowe w postaci tablicy haszującej dla pytań dwustopniowych stosując wartości całkowite,
- dla zapytań egzystencjalnych najefektywniejsze jest indeksowanie ścieżkowe w postaci B-drzewa dla pytań dwustopniowych stosując wartości całkowite lub znakowe.

Ponadto stwierdzono, że najbardziej nieefektywny, ale pozwalający na sprawną pracę z systemem, sposób wyszukiwania dla zapytań prostych to wyszukiwanie bez

indeksowania, gdzie czas realizacji pojedynczego zapytania waha się w granicach 0,05–0,3s.

W przypadku zapytań złożonych, których dotyczyły eksperymenty 10–15, stwierdzono, że jednakowo w przypadku wyszukiwania kolekcji obiektów czy pojedynczych obiektów, jak i dla zapytań egzystencjalnych najefektywniejszym sposobem jest indeksowanie całej ścieżki w postaci tablicy haszującej. Mniej efektywnym sposobem jest wyszukiwanie z indeksem ścieżkowym w postaci B-drzewa. Natomiast, kiedy aplikacje nie wykorzystywały w ogóle indeksowania system stawał się nieefektywny, a czas realizacji pojedynczego zapytania wynosił 25 s.

Problematyka wyszukiwania w obiektowych bazach danych nie została w pracy w pełni wyczerpana. Dalsze badania mogłyby dotyczyć przeprowadzenia zaproponowanych w pracy eksperymentów dla różnych metod grupowania obiektów oraz w innym środowisku bazodanowym np. O2, IRIS, POET, Objectivity lub Persistent i porównania otrzymanych wyników.

Zdobyte doświadczenie przy realizacji pracy pozwoli autorom trafniej dokonywać wyboru odpowiedniego systemu bazy danych w dużych projektach informatycznych.

LITERATURA

- [1] ABITEBOUL S., HULL R., VIANU V., *Foundations of Databases*, Addison-Wesley Publishing Company 1995.
- [2] FIGURA D., *Obiektowe bazy danych*, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1996.
- [3] GOCZYŁA K., *Metody indeksowania w obiektowych bazach danych* (rozprawa habilitacyjna). Zeszyty Naukowe Politechniki Gdańskiej, 1998.
- [4] KIM W., *Wprowadzenie do obiektowych baz danych*, Wydawnictwa Naukowo-Techniczne, Warszawa 1996.
- [5] KŁOSOW A., KRÓL D., *Obiektowe systemy z bazami danych – zastosowania*, Informatyka, nr 2, 2000, str. 16–23.
- [6] KOJS Ł., *Wyszukiwanie w obiektowej bazie danych ObjectStore* (praca magisterska). Politechnika Wrocławska, 2000.
- [7] KRÓL D., NGUYEN 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.
- [8] KRÓL D., MOŻDŻYŃSKI A., *Zestawienia w obiektowych bazach danych*, Informatyka, nr 1, 2000, str. 17 – 20.
- [9] *OBJECTSTORE C++ RELEASE 5.0 DOCUMENTATION*, Burlington, 1995.
- [10] *POTIS*, 1998, <http://www.potis.com.pl>.
- [11] SUBIETA K., *Obiektowość w projektowaniu i bazach danych*, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1998.

DATA RETRIEVAL IN OBJECT DATABASE

Object-oriented query functions are presented. All examples are based on ObjectStore, the most known object-oriented database system. Simple and complex queries for the following functions are described: `query()`, `query_pick()` and `exists()`. 15 experiments are executed. We took into account two indexing methods: hash table and B Tree. The results allow for optimal using the adequate query function.

*aktywna baza danych,
aktywny SZBD, aktywne reguły,
więzy, wyzwacze,
zdarzenie, warunek, akcja*

Hanna MAZUR*

Zygmunt MAZUR*

AKTYWNE BAZY DANYCH

W pracy przedstawiono podstawowe pojęcia z zakresu aktywnych baz danych, omówiono podstawowe problemy występujące w aktywnych bazach danych. Omówiono reguły aktywne ECA (wyzwalacze).

1. WPROWADZENIE

Na początek spróbujemy określić, czym jest **aktywna baza danych** (ang. *active database*). Aktywny system zarządzania bazą danych automatycznie reaguje na zdarzenia. Tradycyjne systemy zarządzania bazą danych są pod tym względem bierne (pasywne) tzn. reagują tylko na żądania użytkowników lub działają automatycznie w określonych odstępach czasowych np. drukują raporty w ostatnim dniu miesiąca. Aktywna baza danych działa nawet wtedy, kiedy użytkownik nie kieruje do niej żadnych żądań. Funkcjonowanie aktywnej bazy danych opiera się na definiowaniu i przetwarzaniu reguł aktywnych (ang. *active rules*) ECA czyli Event – Condition – Action (Zdarzenie – Warunek – Akcja).

Model ECA wykorzystuje trzy rodzaje operacji:

- wystąpienie zdarzenia,
- sprawdzenie warunku,
- wykonanie akcji,

które są wykonywane w dwóch fazach:

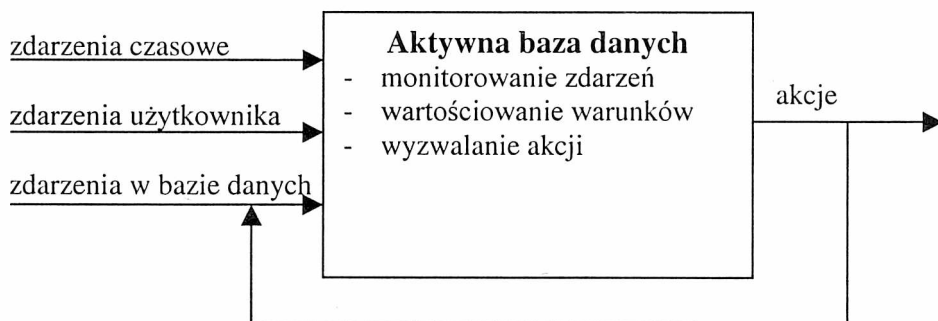
- w chwili wystąpienia zdarzenia,
- w momencie zainicjowania akcji.

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej,
50-370 Wrocław, Skwer Idaszewskiego 1, hmazur@ci.pwr.wroc.pl, mazur@ci.pwr.wroc.pl

System zarządzania bazą danych, który jest przystosowany do rejestrowania zdarzeń i natychmiastowego reagowania na zdarzenia zachodzące w środowisku bazy danych (np. aktualizacja danych) lub w środowisku zewnętrznym (np. upływ czasu) określa się jako **aktywny SZBD** (ang. *active DBMS*). Często za aktywny SZBD uważa się tradycyjny SZBD umożliwiający przechowywanie aktywnych reguł *ECA* w bazie danych. Aktywne SZBD ułatwiają zarówno tworzenie i utrzymanie aplikacji poza tradycyjnym zakresem jak w zakresie tradycyjnym (pasywnym).

2. ARCHITEKTURA AKTYWNEJ BAZY DANYCH

Aktywna baza danych (Rys. 1) odbiera i reaguje na zdarzenia czasowe (np. godz. 24.00 lub ostatni dzień miesiąca), na zdarzenia użytkownika (np. wybór operacji generuj raport) oraz na zdarzenia zachodzące w bazie danych (np. wprowadzeniu w formularzu PIT-5 kwoty określającej wysokość należnej zaliczki, uruchamiana jest akcja drukowania druku PIT i blankietu wpłaty). Monitoruje odbierane zdarzenia, wartościuje warunki i wyzwala określone akcje.



Rys. 1. Architektura aktywnej bazy danych

Moment uaktywnienia bazy danych zależy od zajścia pewnego zdarzenia i spełnienia określonego warunku logicznego.

3. REGUŁY AKTYWNE

Reguła aktywności (ang. *active rules*) *ECA* (ang. *Even – Condition – Action*) jest realizowana następująco: w momencie zajścia zdarzenia *E*, jeśli warunek *C* jest prawdziwy, wykonywana jest akcja *A*.

Reguły aktywne są przechowywane w bazie danych i mogą być współdzielone przez różne programy aplikacyjne. Reguły aktywne stanowią spójny mechanizm umożliwiający z poziomu systemu zarządzania bazą danych wykonywanie zmian

stanu bazy danych, egzekwowanie więzów integralności i praw dostępu, monitorowanie dostępu do danych, śledzenie ewolucji bazy danych oraz otrzymywanie danych pochodnych [KRÓLIKOW 1999].

4. JĘZYKI REGULOWE SZTUCZNEJ INTELIGENCJI A JĘZYKI BAZ DANYCH

Aktywna baza danych według słownika Kazimierza Subiety, to „baza danych zawierająca aktywne reguły” [SUBIETA 1999]. Natomiast reguły aktywne, to „Byty programistyczne (zbudowane z sekwencji instrukcji) umieszczane w bazie danych, których uruchomienie jest powodowane spontanicznie (niezależnie od normalnego przebiegu sterowania programu aplikacyjnego) przez określone zdarzenia zachodzące w bazie danych, np. aktualizację pewnej danej, upływanie pewnego czasu, itp.” [SUBIETA 1999]. Jak już wspomnieliśmy wcześniej reguły aktywne w bazach danych mają postać *ECA*.

Reguły definiowane w sztucznej inteligencji noszą nazwę reguł produkcyjnych i mają postać

wzorzec → akcja.

Reguła jest uaktywniana, jeśli dane mają postać zgodną ze wzorcem (ang. *pattern*), natomiast akcja (ang. *action*) dokonuje modyfikacji pamięci roboczej zgodnie z dopasowywaniem danych. Wzorzec jest określany mianem warunku (ang. *condition*) lub predykatu (ang. *predicate*). Zdarzenia (ang. *events*) uaktywniające reguły są specyfikowane w sposób bezpośredni. Reguła jest uaktywniana tylko wtedy, gdy w pamięci roboczej znajdują się dane pasujące do wzorca (w wyniku wprowadzenia nowych danych lub zmodyfikowania istniejących w pamięci roboczej danych) lub dla reguł zanegowanych, gdy z pamięci są usuwane dane pasujące do wzorca [KROL 1999]. Zdarzeniami uaktywniającymi reguły są więc dopisywanie, modyfikowanie i usuwanie danych. Zdarzenia te są specyfikowane w sposób bezpośredni i jest to zasadnicza różnica pomiędzy językami regułowymi sztucznej inteligencji a językami wykorzystywanymi w systemach baz danych. Języki te różnią się zarówno syntaktyką jak i semantyką oraz wykorzystywaniem mechanizmów wnioskowania.

Reguły aktywne są często nazywane **wyzwalaczami** lub **trygerami** (ang. *triggers*). Określa się je też jako reguły biznesowe (ang. *business rules*). Są one cechą wielu systemów zarządzania bazami danych, relacyjnych, relacyjno-obiektowych i obiektowych.

Język SQL (ang. *Structured Query Language*) wersja SQL3 umożliwia tworzenie elementów aktywnych, czyli wyrażeń lub poleceń, które po zapisaniu w bazie danych są automatycznie wykonywane w odpowiednim momencie.

5. ZDARZENIA ELEMENTARNE

Zbiór zdarzeń elementarnych uaktywniających reguły możemy podzielić na pięć kategorii:

1. **Zdarzenia związane ze stanem obiektu**
 - a. utworzenie obiektu: AFTER CREATE;
 - b. usunięcie obiektu: BEFORE DELETE;
 - c. modyfikacja, odczyt lub dowolny dostęp do obiektu
{BEFORE| AFTER} UPDATE, READ, ACCESS;
2. **Zdarzenia związane z uaktywnieniem metody obiektu**
{BEFORE| AFTER} nazwa_metody;
3. **Zdarzenia związane ze zmianami stanu transakcji**
 - a. rozpoczęcie transakcji: AFTER TBEGIN;
 - b. punkt zatwierdzenia transakcji: AFTER TCOMPLETE;
 - c. zatwierdzenie transakcji: AFTER TCOMMIT;
 - d. wycofanie transakcji: {BEFORE| AFTER} TABORT;
4. **Zdarzenia temporalne**
 - a. określony punkt czasu: AT TIME (timestamp);
 - b. po upływie czasu: AFTER TIME (interval);
 - c. periodycznie co określony czas: EVERY TIME (interval);
5. **Zdarzenia zdefiniowane przez użytkownika:**
{BEFORE| AFTER} nazwa_użytkownika

Poszczególnym, wyżej wymienionym zdarzeniom możemy przyporządkować nazwy:

create (utworzenie obiektu),
delete (usunięcie obiektu),
update (modyfikacja obiektu),
read (odczyt obiektu),
access (dostęp do obiektu),
method_name (uaktywnienie metody),
tbegin (rozpoczęcie transakcji),
tcomplete (przed próbą commit),
tcommit (zakończenie wykonywania kodu transakcji),
tabort (odrzućenie transakcji),
at czas_absolutny (dany punkt na skali czasu),
every przedział_czasu (periodycznie co przedział czasu),
after przedział_czasu (po upływie przedziału czasu).

6. WYRAŻENIA I OPERATORY ZDARZENIOWE

Wystąpienie zdarzenia lub po prostu *zdarzenie* jest parą
(typ zdarzenia, znacznik czasowy).

Znaczniki czasowe umożliwiają globalne uporządkowanie zdarzeń. Historia zdarzeń h danego obiektu jest skończonym zbiorem zdarzeń, w którym żadne dwa zdarzenia nie mogą mieć tego samego znacznika czasowego.

Wyrażenia zdarzeniowe umożliwiają definiowanie zdarzeń elementarnych (predefiniowanych w systemie) oraz zdarzeń złożonych (definiowanych przez użytkowników systemu). Wyrażenie zdarzeniowe E jest odwzorowaniem, dla którego dziedziną i przeciw-dziedziną jest historia zdarzeń: $E: h \rightarrow h$. Wyrażenie zdarzeniowe E określone w historii h wyznacza podzbiór zdarzeń h .

Dla historii h definiuje się wiele operatorów zdarzeniowych, np.:

- $\Phi[h]$ – pusty zbiór zdarzeń elementarnych
- $start[h]$ – pewne nieistniejące zdarzenie poprzedzające wszystkie zdarzenia w h
- $first$ – pierwsze zdarzenie w historii po zdarzeniu $start$
- $\langle n \rangle E$ – n -te wystąpienie zdarzenia E
- $(every \langle n \rangle E)$ – każde kolejne n -te wystąpienie zdarzenia E
- $a[h]$, gdzie a – zdarzenie elementarne – zbiór wszystkich wystąpień zdarzenia a w historii h
- any – zbiór wszystkich zdarzeń elementarnych w historii za wyjątkiem zdarzenia $start$
- $!$ – negacja zdarzenia, $(!E[h]) = h - E[h]$
- $prior(E, F)[h]$ – zbiór wszystkich zdarzeń F , których znaczniki czasowe są większe od jakiegokolwiek zdarzenia E w historii h .

7. WIĘZY I WYZWALACZE

Naturalne jest, że dane wstawiane do bazy danych muszą podlegać kontroli. Kontrola poprawności musi obejmować także, dane modyfikowane i usuwane. Warunki poprawności są często bardzo skomplikowane a ich sprawdzanie jest wielokrotnie powtarzane. Język SQL jest wyposażony w narzędzia pozwalające dołączać więzy integralności do schematu bazy danych. Wyróżnia się więzy integralności kluczy, więzy integralności referencyjnej, więzy nakładane na zakres domen, więzy określające dopuszczalność wartości NULL, więzy dla tabel, rekordów oraz więzy na związki (asercje) między tabelami. **Więzy** są to warunki, które są automatycznie sprawdzane przy próbie wykonania określonej operacji na określonych danych.

Dla zilustrowania pojęcia więzów podajemy kilka przykładów więzów zapisanych w języku SQL:

a) więzy klucza (**PRIMARY KEY**)

```
CREATE TABLE Osoby
    (Nr_Leg CHAR(10) PRIMARY KEY,
    Nazwisko CHAR(30));
```

b) więzy klucza obcego

```
CREATE TABLE Dzieci
    (Nr_Dz CHAR(10) PRIMARY KEY,
    Imię CHAR(30),
    Nr_rodz CHAR(10) REFERENCES Osoby(Nr_Leg));
```

c) więzy NOT NULL (nakładane na atrybuty)

```
CREATE TABLE Dzieci
    (Nr_Dz CHAR(10) PRIMARY KEY,
    Imię CHAR(30),
    Nr_rodz CHAR(10) REFERENCES Osoby(Nr_Leg) NOT NULL);
```

d) więzy CHECK (nakładane na atrybuty)

```
CREATE TABLE Osoby
    (Nr_Leg CHAR(10) PRIMARY KEY,
    Nazwisko CHAR(30),
    Płeć CHAR(1) CHECK( płeć IN ('K', 'M'));
```

e) więzy CHECK (nakładane na dziedziny)

```
CREATE DOMAIN D_Płeć CHAR(1) CHECK( VALUE IN ('K', 'M'));
```

Więzy są wykonywane za każdym razem, gdy ma nastąpić zmiana wartości elementu, którego dotyczy. Ich implementacja polega na uruchomieniu (wyzwoleniu) sprawdzania warunku w chwili zajścia określonego zdarzenia.

Wyzwalacze są to reguły aktywne zapisane w bazie danych, wykonywane po zajściu danego zdarzenia i spełnieniu określonego warunku. W standardzie SQL2 nie ma wyzwalaczy natomiast w SQL3 już są.

Wyzwalacze czyli reguły *ECA* (zdarzenie – warunek – akcja), różnią się od więzów na trzy sposoby [UL_WID 2000]:

1. Wyzwalacze są sprawdzane tylko dla określonych zdarzeń (zazwyczaj dla dopisywania, modyfikowania i usuwania wierszy tabeli). W niektórych implementacjach SQL są też wyzwalacze związane z zakończeniem transakcji.
2. Wyzwalacze testują warunek w chwili zajścia zdarzenia i w zależności od jego spełnienia następuje wykonanie akcji lub nie.
3. Jeśli warunek jest spełniony następuje wykonanie akcji, określonej w wyzwalaczu, która może być jedną operacją lub ciągiem operacji i może nie mieć żadnego związku z wyzwalanym zdarzeniem.

Wyzwalacze tworzy się za pomocą wyrażenia SQL

CREATE TRIGGER.

Nagłówek wyzwalacza zawiera jego nazwę, która powinna być unikatowa w obrębie bazy danych, oraz nazwę tabeli, w której wyzwalacz będzie wywoływany.

7.1. WYZWALACZE W SQL3

Język SQL3 posiada kilka nowych cech w porównaniu z językiem SQL2 m.in. instrukcję **CREATE TRIGGER**, która definiuje wyzwalacz, czyli specyfikuje zdarzenie i akcję [DATE_DAR 2000]. Zdarzeniem jest wykonanie jednej z instrukcji SQL: **INSERT**, **UPDATE** lub **DELETE**. Akcja jest to ciąg instrukcji do wykonania przed (**BEFORE**) lub po (**AFTER**) wystąpieniu zdarzenia. Pierwszą instrukcją w treści wyzwalacza jest na ogół instrukcja warunkowa, której celem jest sprawdzenie określonego warunku. Jeśli warunek jest spełniony w chwili zajścia zdarzenia, to następuje wykonanie instrukcji (SQL). W składni wyzwalacza określa się czy akcja ma być wykonana dokładnie jeden raz dla każdego wystąpienia zdarzenia czy po jednym razie dla każdego wiersza tabeli, którego dotyczy zdarzenie (**FOR EACH ROW**). Jeśli nie ma klauzuli **FOR EACH ROW**, to wyzwalacz jest stosowany tylko raz podczas przetwarzania instrukcji SQL bez względu na to jak wiele razy zaszłoby zdarzenie wchodzące w skład wyzwalacza. W specyfikacji akcji może być również odwołanie do wartości „przed” i „po” w tabeli związanej z danym zdarzeniem.

W instrukcji **CREATE TRIGGER** dla zdarzenia **UPDATE** można określić atrybuty, których ma dotyczyć modyfikacja (klauzula **OF <atrybuty>**). Klauzula ta nie jest dopuszczalna w przypadku zdarzeń **DELETE** i **INSERT**.

Instrukcje SQL występujące w specyfikacji akcji oddziela się średnikami.

Oto składnia wyzwalacza bazy danych:

```
CREATE [OR REPLACE] TRIGGER <azwa_wyzwalacza>
{ BEFORE I AFTER }
[<specyfikacja_instrukcji>]
ON <tabela>
[FOR EACH ROW]
<blok_PL/SQL>
```

gdzie <specyfikacja_instrukcji> jest ciągiem do trzech nazw instrukcji **INSERT**, **DELETE** i **UPDATE** połączonych spójnikiem **OR**. W przypadku **UPDATE** można dodatkowo podać kolumny, aktualizacja których ma powodować uruchomienie wyzwalacza używając składni **UPDATE OF** kolumna,

Kolejność uruchamiania wyzwalaczy jest następująca:

1. Wyzwalacz przed instrukcją;
2. Wyzwalacz przed pierwszym wierszem, na którym operuje instrukcja;
3. Wyzwalacz po pierwszym wierszu, na którym operuje instrukcja;

4.
5. Wyzwalacz przed ostatnim wierszem;
6. Wyzwalacz po ostatnim wierszu;
7. Wyzwalacz po instrukcji.

Aby odróżnić w wyzwalaczach wierszowych stare i nowe wartości w wierszu, używa się następujących oznaczeń odpowiednio na wiersz przed zmianą i na wiersz po zmianie:

OLD – wiersz przed zmianą

NEW – wiersz po zmianie.

Poniższy przykład ilustruje wyzwalacz w języku SQL3 [UL_WID 2000] dla tabeli FilmDyr((Nazwisko, Adres, Cert#, CenaSieci),

który jest aktywowany przy próbie modyfikacji atrybutu CenaSieci i uniemożliwia obniżenie wartości atrybutu CenaSieci.

```
CREATE TRIGGER CenaSieciWyzw
AFTER UPDATE OF CenaSieci ON FilmyDyr
REFERENCING
    OLD AS StaraKrotka
    NEW AS NowaKrotka
WHEN (StaraKrotka.CenaSieci > NowaKrotka.CenaSieci)
    UPDATE FilmyDyr
    SET CenaSieci = StaraKrotka.CenaSieci
    WHERE cert# = NowaKrotka.Cert#
FOR EACH ROW;
```

7.2. WYZWALACZE W DELPHI I INTERBASE

Wyzwalacz w Delphi i InterBase jest podobny do procedury, ale różni się tym, że jest uruchamiany automatycznie w momencie zachodzenia określonych typów zdarzeń. Zdarzenia te mogą dotyczyć całej tabeli lub wiersza danych. Na przykład można zdefiniować wyzwalacz, który jej wywoływany po uaktualnieniu wiersza.

Procedury i wyzwalacze są zwykle tworzone w postaci plików ASCII, które są następnie wykonywane z programu klienckiego ISQL, wewnątrz InterBase. Plik taki zawiera wyrażenia do tworzenia procedury lub wyzwalacza jako część bazy danych. Po zamknięciu transakcji, procedura lub wyzwalacz są natychmiast dostępne do wykorzystania przez aplikacje klienckie.

Procedury i wyzwalacze wykorzystują specjalny język programowania, który jest oparty na SQL. Język zawarty w InterBase jest zoptymalizowany do manipulowania danymi przechowywanymi w tabelach. Posiada pełny dostęp do wyrażeń języka manipulowania danymi (DML), takich jak SELECT, INSERT, UPDATE i DELETE. Udostępnia również rozszerzenia SQL obsługujące zmienne, komentarze, wyrażenia

deklaratywne, sprawdzanie warunków, rozgałęzienia i pętle. Jest językiem zaprojektowanym do wykonywania się wewnątrz bazy danych i operowania na zawartych w niej danych.

Składnia wyzwalacza w InterBase jest następująca [GAWRON 2001]:

```
CREATE TRIGGER <nazwa_wyzwalacza> FOR {tabela | perspektywa}
    {ACTIVE | NOACTIVE}
    {BEFORE | AFTER}
    {DELETE | INSERT | UPDATE}
    [POSITION <numer>]
    AS <treść>
```

Wyzwalacz posiada unikalną nazwę, może być wykonany przed lub po określonym zdarzeniu. Numer jest liczbą z przedziału [0, 32 768] i określa kolejność w jakiej będą wywoływane wyzwalacze, jeśli zdefiniowano ich kilka. Wówczas wyzwalacze są wywoływane o numerach od najmniejszego do największego.

Tworząc wyzwalacz musimy określić czas wywołania wyzwalacza:

BEFORE INSERT	– przed wstawieniem nowego rekordu
BEFORE UPDATE	– przed aktualizacją rekordu
BEFORE DELETE	– przed usunięciem rekordu
AFTER INSERT	– po wstawieniu nowego rekordu
AFTER UPDATE	– po aktualizacji rekordu
AFTER DELETE	– po usunięciu rekordu

Aby utworzyć wyzwalacz powiadamiający aplikację o nowych osobach, można napisać:

```
SET TERM # ;
CREATE TRIGGER NOWA FOR OSOBY
AFTER INSERT
AS
BEGIN
    POST_EVENT „Nowa osoba”
END #
SET TERM ; #
```

Instrukcja **SET TERM** służy do określenia znaku oddzielającego instrukcje (aby nie było konfliktu ze średnikiem kończącym instrukcje SQL).

Instrukcja **POST_EVENT** <nazwa>, służy do wysłania zdarzenia o określonej nazwie (łańcuch znaków lub nazwa zmiennej). Zdarzenie umożliwia wysyłanie komunikatów do aplikacji użytkownika. Aplikacja musi zarejestrować zdarzenia, na które ma reagować i w ten sposób można implementować np. obsługę automatycznego powiadamiania o nowych danych czy o modyfikacjach. Ze zdarzeń korzysta się najczęściej na poziomie wyzwalaczy.

Argumenty za stosowaniem wyzwalaczy w InterBase:

- Pozwalają serwerowi wykonywać złożone operacje na bazie danych bez angażowania oprogramowania klienckiego.
- Mogą być dzielone przez wszystkich klientów mających dostęp do bazy danych. Nie trzeba oprogramowywać tej samej logiki w każdej aplikacji klienckiej; wystarczy ją zaprogramować i przetestować tylko raz na serwerze. Jeśli logika wymaga zmiany, wtedy należy ją zmienić i przetestować tylko raz, a nie wiele razy.
- Zmniejszają ruch sieciowy poprzez odciążenie środowisk aplikacji i przeniesienie przetwarzania na serwer. Jest to szczególnie ważna w przypadku zdalnych użytkowników pracujących z bazą danych przez wolne łącza.
- Zadania te zwykle wykonują się na serwerze o wiele szybciej, ponieważ są one zwykle najsilniejszymi maszynami w sieci.
- Są szczególnie użyteczne do wykonywania złożonych, okresowych czynności, takich jak zamykanie miesiąca lub operacja archiwizowania.
- Wyzwalacze są podstawowym środkiem do utrzymywania spójności bazy danych. Na przykład można je zdefiniować do uruchamiania się przed wprowadzaniem nowego wiersza do bazy danych. Można wówczas przy pomocy wyzwalacza określić, czy wiersz jest kompletny i wewnętrznie spójny, a jeśli nie, uniemożliwić operację. Można także zdefiniować wyzwalacz wykonujący się wtedy, kiedy użytkownik próbuje usunąć wiersz z bazy danych, a tabela, z której wiersz jest usuwany jest nadrzędna w relacji "jeden do wielu". Wyzwalacz mógłby określać, czy istnieją powiązane wiersze w tabelach szczegółowych i usuwać je automatycznie lub blokować operację.
- Wyzwalacze są często wykorzystywane do generowania wartości klucza pierwotnego przy wstawianiu nowego wiersza. Jeśli tabela ma klucz określony za pomocą identyfikatora generowanego, wówczas można wywołać wewnątrz wyzwalacza funkcję `GEN_ID(<nazwa>, <krok>)` w celu utworzenia unikalnego identyfikatora przed wysłaniem nowego wiersza do tabeli.
- Wyzwalacze mogą być również wykorzystywane do zapisywania informacji o statusie tabeli podstawowej do innej tabeli. Na przykład wyzwalacz mógłby być oprogramowany w taki sposób, że byłby uruchamiany zawsze, kiedy zmieni się jakiś wiersz w danej tabeli. Wyzwalacze mogą być także stosowane do rejestrowania zmian w tablicy historii, albo powiadamiania innych aplikacji o wystąpieniu zmiany.
- Wyzwalacze i procedury są integralną częścią mechanizmu "Event Alerter" InterBase. Zdarzenia są normalnie raportowane do menedżera zdarzeń z wnętrza procedury lub wyzwalacza, w oparciu o wyniki zapytania lub testu logicznego.

Wyzwalacze mogą być zdefiniowane do uruchomienia przed operacjami takimi jak: wprowadzanie, uaktualnianie lub usuwanie.

Wyzwalacz z opcją **BEFORE** przy operacji **INSERT** lub **UPDATE** może określać, czy wstawiany lub modyfikowany wiersz jest kompletny i wewnętrznie spójny. Wyzwalacze mogą być również wykorzystywane do wypełniania kolumn, do których użytkownik może nie mieć bezpośredniego dostępu, takich jak unikalny identyfikator klucza pierwotnego tabeli, nazwa użytkownika lub data ostatniego uaktualnienia.

Wyzwalacz z opcją **AFTER** może być wykorzystywany do rejestrowania zmian wiersza. Wyzwalacze mają dostęp do wartości każdego pola wiersza sprzed i po uaktualnieniu. Mogą być również wykorzystywane do uaktualniania informacji w powiązanych tabelach.

Poniższy przykład ilustruje wyzwalacz, który automatycznie umieści unikatową wartość NR_KLIENTA w tabeli KLIENCI, po wygenerowaniu polecenia wstawienia nowego rekordu [JAKUBOW 2001]:

```
SET TERM ^ ;
CREATE TRIGGER SET_NUM_KLIENT FOR KLIENCI
ACTIVE BEFORE INSERT
POSITION 0
AS
BEGIN
    NEW.NR_KLIENTA = GEN_ID(NUM_KLIENT_GEN,1);
END
SET TERM ; ^
```

W kolejnym przykładzie został utworzony wyzwalacz, który sprawdza, czy dane klienta, którego numer widnieje na fakturze są w tabeli KLIENCI [JAKUBOW 1999]:

```
SET TERM ^ ;
CREATE TRIGGER SPR_NUM_KLIENT FOR FAKTURY
ACTIVE BEFORE INSERT
POSITION 0
AS
DECLARE VARIABLE ilosc INTEGER;
BEGIN
    IF (NEW.KLIENT IS NOT NULL) THEN
        BEGIN
            SELECT COUNT (*)
            FROM KLIENCI
            WHERE NR_KLIENTA.KLIENCI = NEW.KLIENT
            INTO :ilosc;
            IF (ilosc =0) THEN
                BEGIN
                    EXCEPTION W1;
                END
            END
        END
    SET TERM ; ^
```

Pierwszą czynnością jaką wykonuje wyzwalacz jest sprawdzenie, czy pole określające numer klienta na fakturze nie jest puste i jeżeli nie jest, to poleceniem SELECT wylicza się liczbę klientów w tabeli KLIENCI o danym numerze. Jeśli nie ma ani jednego klienta o danym numerze (ilosc=0), to wywoływany jest wyjątek.

7.3. WYZWALACZE W SYSTEMIE ORACLE

W systemie ORACLE również można stosować wyzwalacze. Wyzwalacze bazy danych (ang. *database triggers*) są procedurami napisanymi w PL/SQL składowanymi w bazie danych i powiązanych z konkretną relacją. Są wykonywane automatycznie w trakcie wykonywania poleceń SQL na odpowiedniej relacji. Z jedną relacją może być związanych wiele wyzwalaczy ale dany wyzwalacz może być związany wyłącznie z jedną relacją [WRE_WIE 1997].

Podobnie jak w innych systemach wyzwalacze mogą być uruchamiane przed (BEFORE) lub po (AFTER) wykonaniu następującego polecenia języka SQL: INSERT, UPDATE, DELETE.

Gdybyśmy chcieli zdefiniować wyzwalacz, uruchamiany po usunięciu krotki z relacji Osoby, to szkielet wyzwalacza w ORACLU byłby następujący:

```
CREATE OR REPLACE TRIGGER Wyzw1
AFTER DELETE ON Osoby
{ewentualnie część deklarująca stałe, zmienne, kursory, wyjątki lokalne}
BEGIN
    {treść wyzwalacza}
    dbms_output.put_line(„Krotka osoby została usunięta”);
END;
```

W treści wyzwalacza nie wolno używać poleceń DDL (and. *Data Definition Language*) ani poleceń sterujących transakcjami języka SQL takich jak COMMIT, ROLLBACK, SAVEPOINT. Nie wolno również wywoływać podprogramów zawierających te polecenia. W definicji wyzwalacza z opcją FOR EACH ROW opcjonalnie może wystąpić klauzula WHEN określająca dodatkowe warunki uruchomienia wyzwalacza, np. [WRE_WIE 1997]:

```
CREATE OR REPLACE TRIGGER Podwyżka
BEFORE UPDATE ON Osoby
FOR EACH ROW
WHEN (OLD.Pensja < 1000)
BEGIN
    {treść wyzwalacza}
END;
```

W klauzuli **WHEN** można stosować operatory matematyczne porównania (=, <=, >=, !=), logiczne (**AND**, **OR**, **NOT**) oraz operatory SQL (**IS NULL**, **LIKE**, **BETWEEN ... AND**, **IN**), np. **WHEN (OLD.Cena BETWEEN 3.5 AND 5.5)**.

W wersji ORACLE 7.0 z jedną relacją mógł być związany tylko jeden wyzwalacz danego typu czyli łącznie dla jednej relacji można było zdefiniować 12 wyzwalaczy:

BEFORE INSERT
 BEFORE UPDATE
 BEFORE DELETE
 BEFORE INSERT ... FOR EACH ROW
 BEFORE UPDATE ... FOR EACH ROW
 BEFORE DELETE ... FOR EACH ROW
 AFTER INSERT
 AFTER UPDATE
 AFTER DELETE
 AFTER INSERT ... FOR EACH ROW
 AFTER UPDATE ... FOR EACH ROW
 AFTER DELETE ... FOR EACH ROW

W wersji ORACLE 7.3 z daną relacją może być związanych wiele wyzwalaczy tego samego typu.

8. SCHEMATY AKTYWNOŚCI

Pomiędzy kolejnymi fazami wyzwalacza mogą zachodzić relacje:

1. **czasowa**
 - a. **bezpośrednia** (natychmiastowa) – faza druga występuje bezpośrednio po pierwszej
 - b. **opóźniona** – faza druga następuje po fazie pierwszej dopiero po pewnym czasie
2. **transakcyjna**
 - a. **połączona** – zdarzenie i akcja są elementami tej samej transakcji
 - b. **oddzielna** – zdarzenie i akcja należą do różnych transakcji
3. **zależności**
 - a. **zależna** – wykonanie drugiej fazy jest zależne od pomyślnego zakończenia pierwszej fazy
 - b. **niezależna** – wykonanie drugiej fazy nie jest zależne od pomyślnego zakończenia pierwszej fazy

9. WYZWALACZE W LOGICE TEMPORALNEJ

Dzięki logice temporalnej ([HUZAR 2002], [KLIMEK 1999]), uwzględniającej związki czasowe, pojęcie wyzwalacza można znacznie rozszerzyć. Warunkiem może być dowolna formuła temporalna opisująca warunki na przestrzeni sekwencji stanów.

Przykładem wyzwalacza może być:

If φ then action,

gdzie φ jest formułą temporalną, a *action* dowolną reakcją na zdarzenie.

Logika umożliwiająca stosowanie operatorów temporalnych czasu przyszłego określana jest jako logika czasu przyszłego (FTL), logika zawierająca operatory temporalne czasu przeszłego jest logiką temporalną czasu przeszłego (PTL), natomiast logika pozwalająca na stosowanie zarówno operatorów czasu przeszłego jak i przyszłego jest logiką temporalną mieszaną (MTL). Przykładowe formuły temporalne:

$\varphi_1 = \text{sometime_future}(A \wedge B \text{ since } C)$

$\varphi_2 = \text{sometime_future}(A \wedge B)$

Jeśli φ jest formułą typu PTL, to w punkcie rozpatrywania (t) znana jest już jej wartość. Wobec tego wyzwalacz jest już wyzwalany w tym punkcie.

W przypadku formuł typu MTL lub FTL wartość formuły w stanie k nie jest jeszcze znana w tym stanie, wobec tego wyzwalacz może być uaktywniony z pewnym opóźnieniem.

10. PODSUMOWANIE

Aktywny system zarządzania bazą danych automatycznie reaguje na zdarzenia. Tradycyjne systemy zarządzania bazą danych są pod tym względem bierne (pasywne) tzn. reagują tylko na żądania użytkowników lub działają automatycznie w określonych odstępach czasowych np. drukują raporty w ostatnim dniu miesiąca. Aktywna baza danych działa nawet wtedy, kiedy użytkownik nie kieruje do niej żadnych żądań. Funkcjonowanie aktywnej bazy danych opiera się na definiowaniu i przetwarzaniu reguł aktywnych *ECA* czyli Event – Condition – Action (Zdarzenie – Warunek – Akcja). W SQL3 istnieją mechanizmy, które pozwalają na definiowanie i zapisanie w bazie danych aktywnych reguł. Są to **wyzwalacze**, które są wykonywane po zajściu danego zdarzenia i spełnieniu określonego warunku.

LITERATURA

- [DATE_DAR 2000] Date C.J., Darwen Hugh, *SQL. Omówienie standardu języka*, WNT, Warszawa, 2000.
- [GAWROŃ 2001] Gawroński Piotr Neil, *InterBase dla delfinów*, Helion, Gliwice, 2001
- [HUZAR 2002] Huzar Zbigniew, *Elementy logiki dla informatyków*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 2002.
- [JAKUBOW 1999] Jakubowski Arkadiusz, *Delphi 4. Tworzenie sytemów baz danych. Księga eksperta*, Helion, Gliwice, 1999.
- [JAKUBOW 2001] Jakubowski Arkadiusz, *SQL w InterBase dla Windows i Linuksa*, Helion, Gliwice, 2001.
- [KLIMEK 1999] Klimek Radosław, *Wprowadzenie do logiki temporalnej*, Wydawnictwa AGH, Kraków, 1999.
- [KRÓLIKOW 1999] Królikowski Zbyszko, *Zaawansowane systemy baz danych*, ProDialog nr 8, Nakom, Poznań, 1999.
- [SUBIETA 1999] Subieta Kazimierz, *Słownik terminów z zakresu obiektowości*, Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1999.
- [UL_WID 2000] Ullman Jeffrey, Widom Jennifer, *Podstawowy wykład z systemów baz danych*, WNT, Warszawa, 2000.
- [WRE_WIE 1997] Wrembel Robert, Wieczyrzecki Waldemar, *Projektowanie aplikacji bazy danych ORACLE*, Nakom, Poznań, 1997.

ACTIVE DATABASES

This paper presents basic notions of active databases. Basic problems of active databases and active rules (triggers) are discussed.

*autoryzacja,
wielowersyjność,
obiektywne bazy danych,
uwarunkowanie czasowe*

Amin Al-AWADY*

Zygmunt MAZUR*

AUTORYZACJA W WIELOWERSYJNYCH OBIEKTOWYCH BAZACH DANYCH Z UWZGLĘDNIENIEM UWARUNKOWAŃ CZASOWYCH

W artykule przedstawiono podejście autoryzacji do wielowersyjnej obiektowej bazy danych ze szczególnym uwzględnieniem uwarunkowań czasowych. Zostały wyjaśnione pojęcia związane z bezpieczeństwem danych w obiektowych bazach danych mono i wielowersyjnych pracującymi w czasie rzeczywistym. Na przykładach przedstawione zostały uwarunkowania czasowe w nadawaniu i odwołaniu praw dostępu.

1. WPROWADZENIE

W ostatnich latach rozwój systemów baz danych wzbudził zainteresowanie nowymi ich zastosowaniami, przede wszystkim w projektowaniu wspomaganim komputerowo (CAD) i komputerowego wspomagania inżynierii oprogramowania (CASE). W zastosowaniach tych zachodzi konieczność przetwarzania danych multimedialnych (schematy, rysunki, obrazy, animacje, dźwięki, itd.). Najbardziej obiecującą propozycją jest wykorzystanie w modelu bazy danych paradygmatu obiektowego. System zarządzania bazą danych (SZBD) szczególnie w nowych zastosowaniach, musi wspomagać nowe funkcje. Jedną z najważniejszych nowych funkcji SZBD jest zarządzanie wersjami, które istotne jest w systemach baz danych zorientowanych obiektowo. Systemy te wymagają zarządzania jednocześnie różnymi wersjami tego samego obiektu. Tak rozszerzone bazy danych nazywane są wielowersyjnymi obiektowymi bazami danych (ang. *MOODB – Multiversion object-oriented database*) [CeJo 90, RyWi 94, Wic 94a, WiRy 94, OzVa 99].

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Skwer Idaszewskiego 1, mazur@ci.pwr.wroc.pl

Jedną z najważniejszych cech wpływających na efektywność OODB jest autoryzacja dostępu do danych, która jest ważną funkcją, zapewniającą ochronę danych w środowisku wielu użytkowników korzystających wspólnie z dużej bazy danych. Poufne informacje w bazach danych należy chronić przed przypadkowym, bądź umyślnym zniszczeniem, ujawnieniem lub modyfikacją. Możliwości te stanowią niezbędny element dobrego systemu zarządzania bazą danych [SBP 01].

2. KONCEPCJA MODELU AUTORYZACJI

Autoryzację dostępu do danych (ang. *authorization*) można określić jako "fakt posiadania przez jednego użytkownika określonego uprawnienia dostępu do obiektu utworzonego przez drugiego użytkownika" [Kim 96]. Według pracy [Subi 99], autoryzacja to mechanizm lub czynność przypisania użytkownikom praw dostępu do poszczególnych danych, obiektów, usług lub funkcji systemu.

Model autoryzacji z uwzględnieniem uwarunkowań czasowych, przedstawiony w niniejszej pracy zawiera znaczniki węzłów, które zawierają informacje o pozycji autoryzowanego poddrzewa w całej hierarchii obiektów. Proponowany model autoryzacji nazywać będziemy autoryzacją czasową i oznaczamy ją przez AC . W skład tego modelu wchodzi następujące podstawowe elementy:

S – zbiór podmiotów, które będą żądały dostępu do obiektów,

O – zbiór obiektów, do których będzie przyznawane różnego typu prawo dostępu,

A – zbiór typów dostępu (czytanie, modyfikacje, tworzenie, kasowanie, itd.),

Typ – hierarchiczny (h) lub niehierarchiczny (nh),

t – czas udzielania autoryzacji,

ow – okres ważności nadanej autoryzacji.

Rodzaj dostępu do danych może być zatem zdefiniowany jako:

(s, zk, a, typ, t, ow) ,

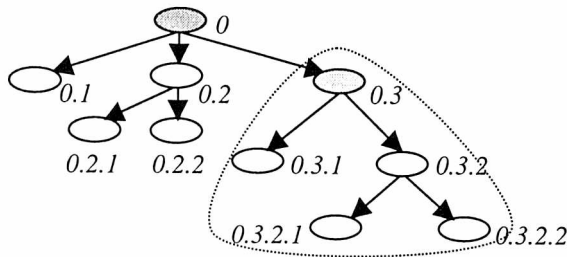
gdzie $s \in S$, $zk \in O$, $a \in A$, zk – znacznik komponenta (węzła), który oznacza sekwencję numerów i konstruowany jest w taki sposób, aby jednocześnie identyfikował ten komponent i wszystkich jego przodków w drzewie wyvodu. Jeżeli węzeł jest n -tym synem jego ojca, którego znacznikiem jest p , to znacznikiem węzła jest $p.n$. Znacznikiem korzenia drzewa wyvodu jest 0 .

Autoryzację AC z uwzględnieniem uwarunkowań czasowych możemy zdefiniować teraz w następujący sposób:

$AC = auth(s, zk, a, typ, t, ow)$,

Predykat $auth$ jest funkcją określającą, czy prawo dostępu ma wartość **true** czy **false**. Jeśli $auth(s, zk, a, typ, t, ow)$ ma wartość **true**, to podmiot lub użytkownik s ma autoryzację o przywileju a na obiekt zk typu typ (hierarchicznie lub nie hierarchicznie) od momentu t na okres ow .

Na rysunku 1 przedstawiono przykładową hierarchię węzłów, którym można nadawać lub odbierać autoryzację.



Rys. 1. Przykładowa hierarchia obiektów autoryzacji

Rozważmy następujący przykład:

$$AC = \text{auth}(s, 0, \text{Read}, h, 10.X.2000, 1_month).$$

Użytkownik s uzyskał autoryzację o pozytywnym przywileju (*Read*) do czytania typu hierarchicznego, nadaną na komponent o znaczniku 0. Autoryzacja jest rozprzestrzeniana na wszystkie komponenty poddrzewa tego węzła (pojęcie niejawnej autoryzacji) w kategorii hierarchii. Czas nadania tej autoryzacji to 10.X.2000 i jest ważna ona przez 1 miesiąc do 9.XI.2000.

Odebranie tego przywileju odbywa się poprzez udzielanie negatywnego przywileju do tego węzła w następujący sposób:

$$AC = \text{auth}(s, 0, -\text{Read}, h, 10.X.2000, 1_month).$$

Autoryzacja ta oznacza odebranie przywileju wcześniej nadanego na węźle o znaczniku 0.

Autoryzację można udzielić na dowolnym węźle w hierarchii obiektów lub je odebrać. Napis: $AC = \text{auth}(s, 0.3, \text{Write}, h, 15.X.2000, 3_months)$, oznacza, że autoryzacja o przywileju (*Write*) do zapisu nadana jest na węźle o znaczniku 0.3 i pociąga za sobą taką samą autoryzację na wszystkich bezpośrednich i pośrednich potomkach tego węzła (tj. 0.3.1, 0.3.2, 0.3.2.1 i 0.3.2.2). Jej ważność wynosi 3 miesiące od daty jej udzielania (15.X.2000). Na rysunku 1 komponenty, na które została udzielona ta autoryzacja, są otoczone przerywaną linią. Przywilej ten (*Write*) implikuje także przywilej (*Read*). Z tego wynika, że ta autoryzacja pozwala na czytanie węzła 0.3 i wszystkich jego komponentów. Można także odebrać nadaną autoryzację na jakiś okres czasu. Autoryzacja:

$$AC = \text{auth}(s, 0.3.2, -\text{Write}, h, 20.X.2000, 1_weak),$$

oznacza odebranie autoryzacji na węźle 0.3.2. i jego komponentów (0.3.2.1 i 0.3.2.2) na okres 1 tygodnia od daty jej cofnięcia (20.X.2000). Po tym okresie w dniu 27.X.2000 autoryzacja uaktywni się i będzie ważna do końca czasu jej wcześniejszego nadania – do dnia 14.I.2001.

3. MODEL WERSJOWANIA OBIEKTU

Pojęcie autoryzacji z uwzględnieniem uwarunkowań czasowych, może być stosowane do różnych modeli wersjowania. Do przedstawienia modelu autoryzacji zastosujemy podejście wersjowania bazy danych przedstawione w pracy [CeJo 90], w którym jednostką spójności jest wersja bazy danych. Wersję bazy danych definiuje się jako zbiór wersji wszystkich obiektów wielowersyjnych przechowywanych w bazie danych tak, że każdy obiekt występuje w niej dokładnie w jednej wersji. Każdej wersji bazy danych nadawany jest unikalny identyfikator w sposób automatyczny w momencie jej utworzenia przez system zarządzania bazą danych, co powoduje, że wszystkie wersje bazy danych są logiczne odizolowane od siebie i zmiany dokonywane w każdej z nich nie są widoczne w pozostałych. Stan wersji bazy danych jest zbiorem wartości zawartych w niej wersji obiektów.

Jednowersyjna baza danych przechowuje reprezentacje pojedynczego stanu modelowanej rzeczywistości. Transakcje definiowane przez użytkownika, przeprowadzają bazy danych z danego stanu spójnego do innego stanu spójnego reprezentującego nowy stan modelowanej rzeczywistości. Nowa reprezentacja zastępuje reprezentację stanu poprzedniego bazy danych. Oznacza to, że gdy użytkownik modyfikuje choć jeden obiekt, to zmienia on reprezentację całej modelowanej rzeczywistości.

W wielowersyjnej bazie danych, użytkownik tworząc nową wersję obiektu tworzy w istocie reprezentację nowego stanu całej modelowanej rzeczywistości. Zatem, jeśli ma być zachowany poprzedni stan bazy danych, to musi on być zachowany jako całość. W modelu z wersjami bazy danych przedstawionym w pracy [CeJo 90] taka pojedyncza reprezentacja stanu rzeczywistości nazywana jest *wersją bazy danych*. W tym modelu baza danych jest zdefiniowana jako zbiór logicznie niezależnych wersji bazy danych. Wersja bazy danych jest parą składającą się z identyfikatora wersji bazy danych (identyfikator jest nadawany przez system zarządzania bazą danych automatyczny w chwili jej utworzenia) i zbioru wersji wszystkich obiektów zawartych w wielowersyjnej bazie danych po jednej wersji na obiekt. Stan wersji bazy danych jest zbiorem wartości zawartych w niej wersji obiektów.

Wersja bazy danych najczęściej różni się jedynie częściowo od wersji przodka, z której została wywiedziona. Oznacza to, że wersje tych samych obiektów zawartych w różnych wersjach bazy danych, mogą mieć te same wartości. W celu uniknięcia tego problemu a także uniknięcia redundancji, wersje obiektów o tych samych wartościach są współdzielone przez różne wersje bazy danych.

Dzięki zastosowaniu tak zwanych *znaczników wersji* rozwiązano problem identyfikacji wersji obiektów, które należą do określonych wersji bazy danych. Znaczniki te są tworzone w sposób umożliwiający również wskazanie przodka każdej wersji bazy danych. Pierwsza wersja bazy danych przedstawiana jako korzeń drzewa,

otrzymuje znacznik 0. Kolejne znaczniki powstają przez dodanie numeru wersji wywodzonej do znacznika wersji, z której jest ona wywodzona.

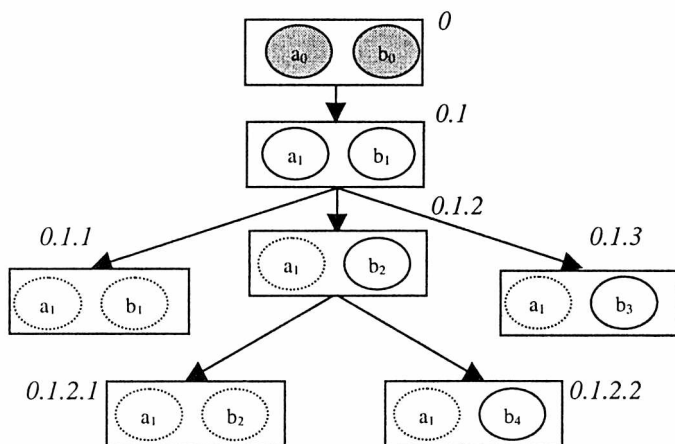
W każdej wersji bazy danych może występować co najwyżej jedna wersja każdego obiektu. Do powiązania wersji obiektu z wersjami bazy danych służy tabela zwana *tabelą asocjacji*. Każdy obiekt wielowersyjny ma własną tabelę asocjacji, w której jeden wiersz tej tabeli odpowiada jednej wersji fizycznej obiektu. Wersja fizyczna jest fizyczną kopią danego obiektu i może być ona współdzielona przez wiele wersji bazy danych. Niektóre wielowersyjne obiekty mogą nie występować w danej wersji bazy danych i dlatego wprowadzono wersję obiektu oznaczaną przez *nil*, traktowaną jako wersję pustą. Takie wersje (*nil*) obiektów wielowersyjnych przechowywane są w wersji bazy danych 0.

Nowo utworzona wersja bazy danych współdzieli wszystkie wersje fizyczne obiektów ze swoim rodzicem. Aby nie zachodziła potrzeba przechowywania w tabeli asocjacji wszystkich znaczników wersji bazy danych, które współdzielią daną wersję obiektu wprowadzono tzw. *regułę wyvodu* w modelu wersji bazy danych.

Reguła wyvodu jest następująca:

„Jeżeli dana wersja bazy danych nie jest jawnie wymieniona w tabeli asocjacji dla danego obiektu, to należy szukać tego obiektu u rodzica tej wersji bazy danych”.

Mechanizm ten, stosuje się rekurencyjnie dla dowolnej liczby przodków w drzewie wyvodu wersji bazy danych, współdzielących poszukiwaną wersję obiektu.



Rys. 2. Drzewo wyvodu wersji bazy danych

Jako przykład rozważmy, wielowersyjną bazę danych składającą się z siedmiu wersji bazy danych. Drzewo wyvodu wersji bazy danych jest przedstawione na rysunku 2. Ukazana na nim wielowersyjna baza danych zawiera wersje 0, 0.1, 0.1.1,

0.1.2, 0.1.3, 0.1.2.1 i 0.1.2.2, a także dwa wielowersyjne obiekty A i B . Obiekt A występuje w dwóch wersjach a_0 (*nil*) w wersji bazy danych 0 i a_1 w wersji bazy danych 0.1 . Tabela asocjacji tego obiektu jest podana w tabeli 1. Wersja a_1 jest współdzielona przez wszystkie wersje bazy danych w drzewie wyводу o korzeniu 0.1 . Obiekt B występuje w pięciu wersjach: b_0 , b_1 , b_2 , b_3 i b_4 w wersjach bazy danych 0 , 0.1 , $0.1.2$, $0.1.3$, i $0.1.2.2$ w kolejności. Tabela asocjacji obiektu B jest podana w tabeli 2. Wersja bazy danych $0.1.1$ współdzieli wersję obiektu b_1 z wersją bazy danych 0.1 , natomiast wersja bazy danych $0.1.2.1$ współdzieli wersję b_2 z bazą danych $0.1.2$.

Jak można zauważyć na przedstawionym przykładzie, znacznik wersji i związana z nim zasada wyводу pozwalają na całkowite wyeliminowanie uaktualnień powiązań wersji obiektów z wersjami bazy danych w przypadku tworzenia nowych wersji bazy danych oraz zapewnienia uniknięcia redundancji wersji obiektów.

Tab. 1. Asocjacja obiektu A

Wersje obiektu A	Znaczniki wersji
a_0	0
a_1	0.1

Tab. 2. Asocjacja obiektu B

Wersje obiektu B	Znaczniki wersji
b_0	0
b_1	0.1
b_2	$0.1.2$
b_3	$0.1.3$
b_4	$0.1.2.2$

Do aktualizacji wartości współdzielonej wersji w bazie danych używany jest następujący algorytm [CeJo 90]:

„dodawany jest nowy wiersz do tabeli asocjacji, wiążący nową wersję obiektu ze znacznikiem wersji bazy danych, w której wykonywane jest uaktualnienie. Poprzednie wystąpienia tego znacznika zostaną usunięte z tabeli asocjacji i zastąpione znacznikami tych wszystkich bezpośrednich dzieci rozpatrywanej wersji bazy danych, które były niejawnie wymienione w tabeli asocjacji”.

Powyższy proces można zilustrować na przykładzie przedstawionym na rysunku 2. W wyniku modyfikacji obiektu A w wersji bazy danych $0.1.2$ otrzymujemy zmodyfikowaną tabelę asocjacji przedstawioną w tabeli 3. Ten sam proces jest stosowany w celu utworzenia i usuwania obiektów w danej wersji bazy danych. W celu utworzenia wielowersyjnego obiektu, należy najpierw dodać jego wersję *nil* do korzenia drzewa wyводу, a następnie obiekt jest uaktualniany w standardowy sposób. W celu usunięcia wersji obiektu wystarczy uaktualnić ją na *nil*.

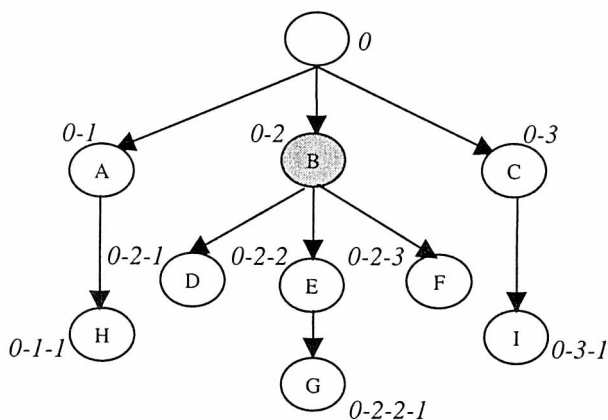
Tab. 3. Asocjacja obiektu A

Wersje obiektu A	Znaczniki wersji
a_0	0
a_2	$0.1.2$
a_1	$0.1, 0.1.2.1, 0.1.2.2$

4. METODA AUTORYZACJI CZASOWEJ NA OBIEKTACH ZŁOŻONYCH

W modelu obiektowym jak wiadomo, jeden obiekt może być częścią składową innego obiektu i obiekt, do którego odwołujemy się z danego obiektu poprzez wartość jego atrybutu może być częścią składową danego obiektu. W tym przypadku mamy do czynienia z obiektami złożonymi (kompozytowymi). Powiązanie kompozycyjne oznacza, że takie powiązanie dwóch obiektów odpowiada relacji *is-part-of* między obiektami. W zależności od modelu danych, hierarchia kompozycji może łączyć wielowersyjne obiekty, wersje obiektu, klasy lub dziedziny atrybutów złożonych tworzących poddrzewa hierarchii dziedziczenia. Przedstawiony model autoryzacji, będzie uwzględniał wielowersyjne obiekty złożone i wersje ich komponentów, które także mogą być złożone.

Model autoryzacji przedstawiony w rozdziale 2 można rozszerzyć do przypadku wielowersyjnych obiektów złożonych. Na rysunku 3 pokazano przykładową hierarchię obiektów złożonych (hierarchia kompozycji).



Rys. 3. Przykład hierarchii kompozycji

Pozycja obiektu w hierarchii może być identyfikowana przez pojedynczy znacznik kompozycji, który jest identyczny z pojęciem znacznika węzła przedstawionego w rozdziale 2. Do rozróżnienia znaczników wersji od znaczników kompozycji w symbolach znaczników zamiast kropek będą używane znaki ‘-’.

Obiekt złożony o znaczniku 0 składa się z obiektów A, B i C. Obiekt A o znaczniku 0-1 ma jeden komponent H o znaczniku 0-1-1, natomiast obiekt B o znaczniku 0-2 składa się z obiektów D, E i F (0-2-1, 0-2-2 i 0-2-3). Obiekt E o znaczniku 0-2-2 ma z kolei komponent G o znaczniku 0-2-2-1. Obiekt C o znaczniku 0-3 ma również jeden komponent I o znaczniku 0-3-1.

Zgodnie z semantyką znaczników kompozycji, poprzez jeden znacznik można znaleźć wszystkie obiekty, które pośrednio lub bezpośrednio są składnikami danego obiektu. Na przykład, obiekt G o znaczniku $0-2-2-1$ jest bezpośrednim komponentem obiektu $0-2-2$ tj. obiektu E i pośrednio komponentem obiektu B o znaczniku $0-2$.

Rozpatrzmy następującą autoryzację nadaną na obiekcie B o znaczniku $0-2$:

$$AC = \text{auth}(s, 0-2, \text{Read}, h, 20.XI.2000, 3_months).$$

Autoryzacja ta, jest nadawana na obiekcie o znaczniku $0-2$ (Rys. 3), jest także rozprzestrzeniana na wszystkie komponenty tego obiektu D , E , F i G (tj. $0-2-1$, $0-2-2$, $0-2-3$ i $0-2-2-1$) i aktualna jest przez podanego okres czasu (tj. 3 miesiące począwszy od 20.XI.2000), jeśli nie zostanie wcześniej cofnięta. Jeżeli autoryzacja w powyższym przykładzie została udzielona jako typ nh (nie hierarchiczny) w sposób następujący:

$$AC = \text{auth}(s, 0-2, \text{Read}, nh, 20.XI.2000, 3_months)$$

- wówczas będzie udzielana tylko na obiekcie o znaczniku $0-2$ i nie będzie rozprzestrzeniana na komponenty tego obiektu.

5. METODA AUTORYZACJI CZASOWEJ NA WERSJACH OBIEKTÓW ZŁOŻONYCH

Metody autoryzacji w tym modelu będą stosowane do dwóch hierarchii:

- hierarchii wyводу wersji bazy danych,
- kompozycji hierarchii (hierarchia obiektów złożonych).

Hierarchia wyводу wersji bazy danych jest konstruowana zgodnie z regułami przedstawionymi w rozdziale 2 (model wersji bazy danych). Do identyfikacji pojedynczego węzła (wersje bazy danych) lub poddrzewa tego węzła używane są znaczniki wersji. Kompozycja hierarchii jest także konstruowana według zasad z rozdziału 3. Identyfikacja pojedynczego węzła (obiekty wielowersyjne) lub poddrzewa jego węzłów odbywa się poprzez użycie znaczników kompozycji. Rodzaj dostępu do węzłów można zdefiniować jako:

$$(s, zwb, zk, a, typk, typb, t, ow), \quad \text{gdzie } s \in S, zk \in O, a \in A,$$

a autoryzację AC_w na wersjach obiektów z uwzględnieniem uwarunkowań czasowych w następujący sposób:

$$AC_w = \text{auth}(s, o, a, zwb, zk, typb, typk, t, ow), \quad \text{gdzie:}$$

$typb$ – hierarchiczny (hb) lub nie hierarchiczny (nhb) w drzewie wersji bazy danych,

$typk$ – hierarchiczny (hk) lub nie hierarchiczny (nhk) w drzewie kompozycji,

zwb – znacznik wersji poddrzewa wersji bazy danych,

zk – znacznik kompozycji obiektu.

W definicji tej można wyróżnić następujące przypadki, w których można udzielać autoryzacji:

- typ hierarchiczny w drzewie wersji bazy danych i hierarchiczny w drzewie kompozycji (hb, hk),
- typ hierarchiczny w drzewie wersji bazy danych i nie hierarchiczny w drzewie kompozycji (hb, nhk),
- typ nie hierarchiczny w drzewie wersji bazy danych i hierarchiczny w drzewie kompozycji (nhb, hk),
- typ nie hierarchiczny w drzewie wersji bazy danych i nie hierarchiczny w drzewie kompozycji (nhb, nhk).

Dla pierwszego przypadku jako przykład rozważmy następującą autoryzację:

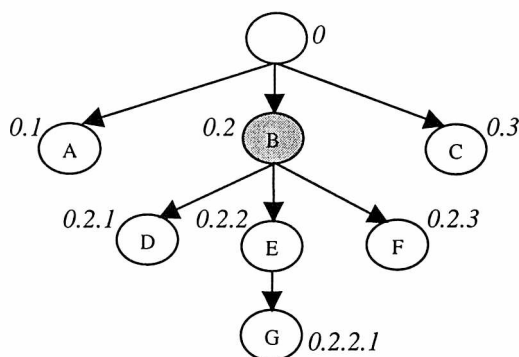
$AC_w = auth(s, 0.2, 0-2, +Read, hb, hk, 15.X.2000, 2_months)$,

zakładając, że dotyczy ona kompozycji hierarchii podanej na rysunku 3. i hierarchii wersji bazy danych podanej na rysunku 4. Autoryzacja ta, przyznawana jest na obiekcie złożonym B (0-2) wraz ze wszystkimi jego komponentami tj. obiekty D , E , F i G w poddrzewie wersji bazy danych o korzeniu 0.2.

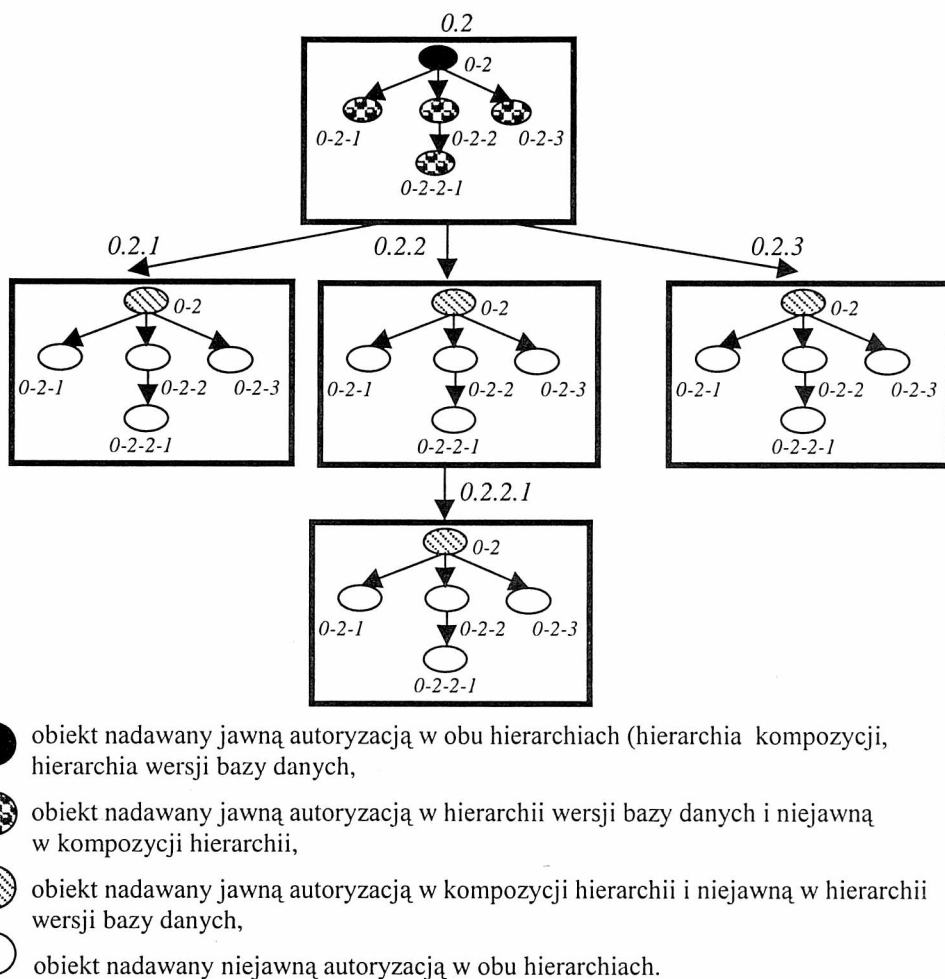
Umiejscowienie tej autoryzacji jest przedstawiane na rysunku 5. Zauważmy, że:

- na obiekcie B (0-2) nadano jawną autoryzację w kompozycji hierarchii, jawną w wersji bazy danych 0.2 i niejawną w wersjach bazy danych 0.2.1, 0.2.2, 0.2.3 i 0.2.2.1, autoryzacja ta jest ważna przez dwa miesiące od daty jej udzielania tj. 15.X.2000.
- na obiektach D (0-2-1), E (0-2-2), F (0-2-3) i G (0-2-2-1), nadano niejawną autoryzację w kompozycji hierarchii, jawną w wersji bazy danych 0.2 oraz niejawną w pozostałych wersjach bazy danych tj. 0.2.1, 0.2.2, 0.2.3 i 0.2.2.1.

Można zauważyć, że w ostatnim przykładzie, autoryzacja została udzielona w obu hierarchiach (hierarchia kompozycji i hierarchia wersji bazy danych).



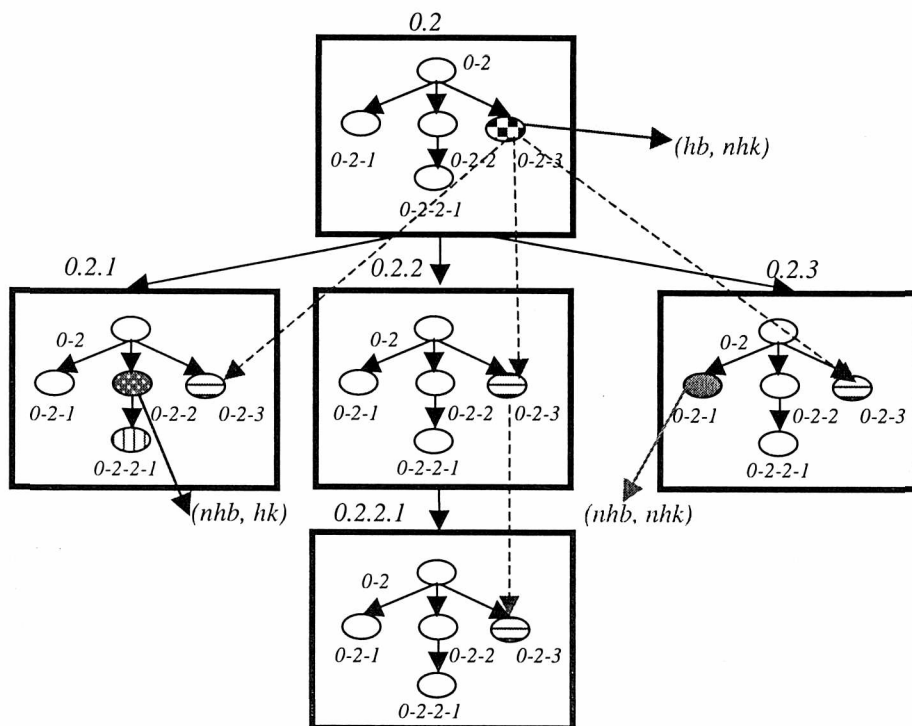
Rys. 4. Przykład hierarchii wywodu wersji bazy danych



Rys. 5. Umieszczenie autoryzacji nadawanej węzłom 0.2 i 0-2 w obu hierarchiach (hierarchia wersji bazy danych i hierarchia kompozycji)

Pozostałe trzy przypadki możemy opisać poprzez następujące trzy autoryzacje:

1. $AC_{w1} = auth(s, 0.2, 0-2-3, Write, hb, nhk, 15.X.2001, 1_month),$
2. $AC_{w2} = auth(s, 0.2.1, 0-2-2, Read, nhb, hk, 15.X.2001, 2_months),$
3. $AC_{w3} = auth(s, 0.2.3, 0-2-1, Read, nhb, nhk, 15.X.2001, 3_months).$



- ☒ obiekt nadawany jawną autoryzacją typu hierarchicznego w hierarchii wersji bazy danych i jawną typu nie hierarchicznego w kompozycji hierarchii,
- ⊖ obiekt nadawany niejawną autoryzacją typu hierarchicznego w hierarchii wersji bazy danych i jawną typu nie hierarchicznego w kompozycji hierarchii,
- ⊙ obiekt nadawany jawną autoryzacją typu nie hierarchicznego w poddrzewie wersji bazy danych i jawną typu hierarchicznego w kompozycji hierarchii,
- ⊖⊖ obiekt nadawany niejawną autoryzacją typu nie hierarchicznego w poddrzewie wersji bazy danych i niejawną typu hierarchicznego w kompozycji hierarchii,
- ⊙⊙ obiekt nadawany jawną autoryzacją typu nie hierarchicznego w poddrzewie wersji bazy danych i jawną typu nie hierarchicznego w kompozycji hierarchii

Rys. 6. Umieszczenie autoryzacji AC_{w1} , AC_{w2} i AC_{w3}

Powyższe autoryzacje należy interpretować w następujący sposób:

1. Autoryzacja $AC_{w1} = auth(s, 0.2, 0-2-3, Write, hb, nhk, 15.X.2001, 1_month)$ jest udzielana na obiekcie 0-2-3 w wersji bazy danych 0.2. Użytkownik będzie miał możliwość modyfikowania obiektu 0-2-3 we wszystkich wersjach bazy danych, które są wywodzone pośrednio lub nie z wersji bazy danych o znaczniku 0.2, bo

została mu udzielona autoryzacja typu hierarchicznego w drzewie wersji bazy danych. Natomiast nie będzie mógł modyfikować składników obiektu 0-2-3, bo nie uzyskał tej autoryzacji w drzewie kompozycji.

2. Autoryzacja $AC_{w_2} = auth(s, 0.2.1, 0-2-2, Read, nhb, hk, 15.X.2001, 2_months)$
nadana jest na obiekcie 0-2-2 i jego składniki (0-2-2-1) w wersji bazy danych 0.2.1.
3. Autoryzacja $AC_{w_3} = auth(s, 0.2.3, 0-2-1, Read, nhb, nhk, 15.X.2001, 3_months)$
nadana jest na obiekcie 0-2-1 w wersji bazy danych 0.2.3.

Zakres tych autoryzacji jest pokazany na rysunku 6.

6. PODSUMOWANIE

W pracy przedstawiono model autoryzacji, w którym można pokazać w jaki sposób można sterować dostępem do bazy danych uwzględniając czas nadawania i odbierania uprawnień. Przedstawione metody autoryzacji prezentowanego modelu są pojęciowo proste i łatwe w implementacji a mimo to wystarczająco silne do operowania złożonymi ograniczeniami dostępu. Metody te także umożliwiają efektywne, hierarchiczne autoryzowanie w wielu hierarchiach.

Korzyści wypływające z zastosowania przedstawionego modelu będą szczególnie widoczne w bazach danych, w których dane są semantycznie powiązane skomplikowanymi hierarchiami, tak jak w wielowersyjnych obiektowych bazach danych a w szczególności w przypadkach baz danych zawierających wiele wersji obiektów złożonych, w których struktura kompozycji jest złożona. Jest to typowy przypadek projektowania baz danych, gdzie obiekty mają wiele komponentów dostępnych w wielu wersjach.

LITERATURA

- [CeJo 90] Cellary W., Jomier G., *Consistency of Versions in Object-Oriented Databases*, Proc. 16 VLDB Conference, Brisbane, Aug. 1990, pp. 432–441.
- [OzVa 99] Ozsu M., T., Valduriez P., *Principles of Distributed Database Systems* (2nd Edition), Prentice Hall, Upper Saddle River, NJ, 1999.
- [RyWi 94] Rykowski J., Wiczerzycki W., *Using versioned object-oriented data in programming languages*, Microprocessing and Microprogramming, 40, 901–904, 1994.
- [SBP01] Stokłosa J., Bilski T., Pankowski T., *Bezpieczeństwo danych w systemach informatycznych*, Wydawnictwo Naukowe PWN, Warszawa, Poznań, 2001.

- [Wiec 94] Wiczerzycki W., *Version management in design databases*, Proc. 9th Int. Conf. On Computer and Information Sciences, Antalya, November 1994.
- [WiRy 94] Wiczerzycki W., Rykowski J., *Version support for CAD/CASE databases*, East-West Database Workshop, Klagenfurt, Austria, September 1994.

AUTHORIZATION ON MULTIVERSION OBJECT-DATABASES WITH TIME RELATED ASPECTS

This paper gives an overview of authorization on multiversion Object-Databases with emphasis on time related aspects of database security. Basic notions related to real-time data security in Object-Databases (one and multiversion). The time aspects of granting and revoking rights are presented by examples.

autoryzacja,
acykliczne grafy,
obiektywne bazy danych,
uwarunkowanie czasowe

Amin Al-AWADY*

Zygmunt MAZUR*

AUTORYZACJA NA GRAFACH ACYKLICZNYCH W OBIEKTOWYCH BAZACH DANYCH Z UWZGLĘDNIENIEM UWARUNKOWAŃ CZASOWYCH

W artykule zaproponowano nowe podejście do zarządzania autoryzowanym dostępem do baz danych, które umożliwia efektywną, hierarchiczną autoryzację w skierowanych grafach acyklicznych. Podstawową ideą tego podejścia jest rozszerzenie autoryzacji podstawowej o tzw. znaczniki węzłów, które zawierają informacje o umiejscowieniu węzłów objętych autoryzacją w odpowiednich hierarchiach węzłów (grafach). Informacja ta jest odpowiednikiem sekwencji autoryzacji intencjonalnych zakładanych od korzenia drzewa (grafu) do odpowiedniego węzła. Istotnie ułatwia to wykrywanie niezgodności autoryzacji żądanej z autoryzacjami ustawionymi. Korzyści wypływające z zastosowania zaproponowanego podejścia są szczególnie widoczne w przypadku baz, których dane są semantycznie powiązane skomplikowanymi hierarchiami i grafami, takich jak obiektywne bazy danych.

1. WPROWADZENIE

W systemach umożliwiających dziedziczenie wielokrotne, klasy tworzą zakorzeniony spójny skierowany graf acykliczny (ang. *DAG – Directed Acyclic Graph*). Hierarchia klas *DAG* jest spójna, co oznacza, że nie ma odizolowanych węzłów. Każdy węzeł jest dostępny od korzenia. Jeśli dwa węzły są połączone, to węzeł niższego poziomu dziedziczy wszystkie własności (tj. atrybuty i metody) węzła wyższego poziomu. Jeśli dany węzeł ma więcej niż jednego bezpośredniego poprzednika, dziedziczy on wszystkie własności tych węzłów.

Kompozycja *DAG* odzwierciedla struktury obiektu złożonego reprezentującego stan rzeczywistości. Obiektem złożonym nazywany jest obiekt zawierający co najmniej jeden atrybut, którego wartością jest wskazanie na inny obiekt. Objekt złożony może się składać z pewnej liczby komponentów (atrybutów), które także

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej,
50-370 Wrocław, Skwer Idaszewskiego 1, mazur@ci.pwr.wroc.pl

mogą być złożone. Jeśli obiekt jest bezpośrednim komponentem dokładnie jednego obiektu złożonego, to odwołanie do niego jest wyłącznym odwołaniem, w przeciwnym razie odwołanie jest wspólne.

W proponowanym w tej pracy modelu autoryzacji zostały zdefiniowane i zastosowane tak zwane znaczniki węzłów autoryzacji zaproponowane w pracy [CeWi 94a], które zawierają informacje o pozycji danego węzła we wszystkich hierarchiach, a także zawierają informacje o umiejscowieniu węzłów objętych autoryzacją w odpowiednich hierarchiach (grafach). To znacznie ułatwia przekazywanie i odwołanie autoryzacji, oraz ułatwia wykrywania niezgodności autoryzacji żądanej z autoryzacjami przydzielonymi. Korzyści podejścia tego modelu autoryzacji stają się szczególnie ważne w przypadku baz danych, których dane są semantycznie powiązane skomplikowanymi hierarchiami i grafami.

2. METODY AUTORYZACJI CZASOWEJ NA GRAFACH ACYKLICZNYCH

W tym artykule przedstawione rozszerzenia metod autoryzacji wymienione w pracy [AlMa 03] o typy tak zwane skierowane acykliczne grafy, w którym jeden węzeł może należeć do więcej, niż jednego bezpośredniego przodka. Skierowany acykliczny graf (*DAG*), który ma różne korzenie jest przekształcany w taki sposób, aby miał tylko jeden korzeń, poprzez dodanie nowego węzła, który łączy się z wszystkimi pierwotnymi korzeniami. Węzeł taki będzie miał znacznik 0 [Wiec 94b].

W celu nadawaniu autoryzacji czasowej na grafach acyklicznych, lepszym sposobem jest określenie najpierw dla każdego węzła mającego bezpośrednio więcej niż jednego poprzednika tylko jeden poprzednik (z którego on się wywodzi). Taki poprzednik jest podstawą dla konstruowania rozpatrywanego węzła i będzie nazywany *głównym przodkiem*. Główny przodek, jest to węzeł którego znacznik po porównywaniu z wszystkimi innymi bezpośrednimi przodkami, ma najmniejszy numer na pierwszej różniącej się pozycji.

Z dwóch bezpośrednich przodków o znacznikach [Wiec 94b]:

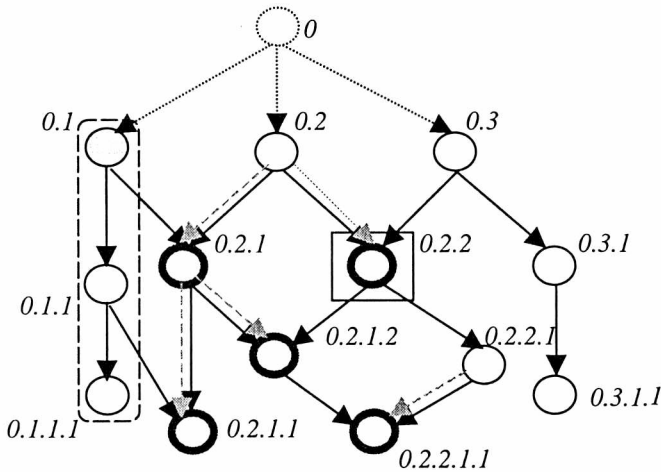
$$zk_1 = a_1, a_2, \dots, a_m \text{ i } zk_2 = b_1, b_2, \dots, b_n,$$

węzeł zk_1 jest głównym przodkiem wtedy i tylko wtedy jeśli

$$\exists (\forall_{i \geq 1} a_i = b_j \wedge a_{i+1} < b_{j+1}).$$

Rysunek 1 prezentujący przykład skierowanego acyklicznego grafu ze znacznikami jego węzłów, zawiera on pięć węzłów mających więcej niż jednego bezpośredniego przodka, są to: 0.2.1, 0.2.2, 0.2.1.1, 0.2.1.2 i 0.2.2.1.1. Na rysunku 1 możemy zauważyć, że węzeł o znaczniku 0.2.2 posiada dwa bezpośrednie przodki: 0.2 i 0.3. Według reguły określania przodka głównego wynika, że węzeł 0.2 jest jego głównym,

ponieważ jego drugi element jest mniejszy niż drugi element znacznika 0.3, a głównymi przodkami węzłów 0.2.1, 0.2.1.1, 0.2.1.2 i 0.2.2.1.1 są odpowiednio węzły 0.2, 0.2.1, 0.2.1 i 0.2.2.1.



Rys. 1. Przykład skierowanego acyklicznego grafu

Jak przedstawiono w [AlMa 03], autoryzacja udzielana na danym węźle pociąga za sobą taką samą autoryzację na każdej składowej tego węzła (należącej do hierarchii tego węzła) w następujący sposób:

$$AC = auth(s, zk, a, typ, t, ow),$$

gdzie zk – znacznik kompozycji składający się z węzłów zawartych w poddrzewie hierarchii, zakorzenionej przez znacznik zk .

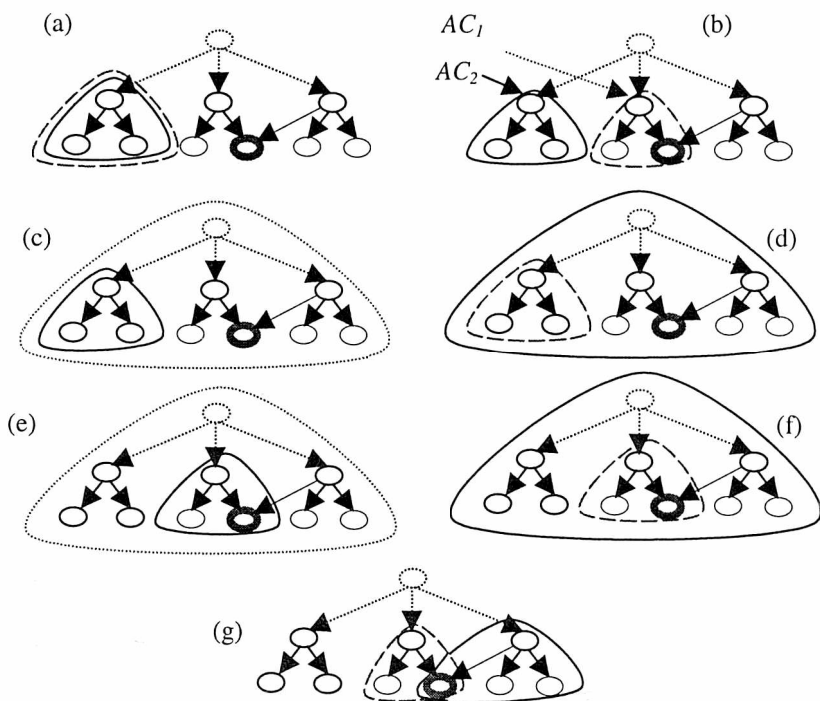
Jeśli węzeł ma więcej niż jednego bezpośredniego przodka, to użytkownik może uzyskać więcej niż jedną niejawną autoryzację na tym węźle. Z rysunku 1 wynika, że jeżeli użytkownik uzyskał uprawnienie do odczytu węzła 0.2 na czas określony, to jednocześnie uzyskuje niejawną autoryzację do odczytu na węźle 0.2.2 na ten sam okres. Gdy następnie użytkownik uzyska uprawnienia odczytu węzła 0.3, to ponownie uzyska niejawną autoryzację na węźle 0.2.2 (węzeł otoczony kwadracikiem).

Negatywne autoryzacje mogą powodować kolizje między autoryzacjami niejawnymi na węzłach, które mają więcej niż jednego bezpośredniego przodka. Jeśli użytkownik uzyska np. negatywną autoryzację odczytu na węźle 0.3, to spowoduje to kolizję z (pozytywną) niejawną autoryzacją, którą użytkownik uzyskał z węzła 0.2. Zgodność dwóch autoryzacji jest ustalona przez następującą regułę (reguła kompatybilności autoryzacji):

Dwie autoryzacje, udzielane temu samemu użytkownikowi są zgodne (kompatybilne), jeśli są pozytywne lub mają różne typy (pozytywny, negatywny) a ich zakres nie ma wspólnego węzła, lub jeśli jedna z nich jest jawną autoryzacją, która

unieważnia niejawną. Natomiast jeśli ich znak jest różny (pozytywny i negatywny), a ich zakres ma wspólny węzeł, a także obie są niejawne, to wtedy są niezgodne.

Wszystkie możliwe powiązania między umiejscowieniami dwóch autoryzacji AC_1 i AC_2 są pokazane na rysunku 2, w którym zakres pierwszej autoryzacji (AC_1) jest otoczony przerywaną linią, natomiast drugiej (AC_2) ciągłą linią.



Rys. 2. Porównywanie dwóch umiejscowień autoryzacji

Na rysunku 2 można zauważyć kilka różnych zakresów dwóch autoryzacji, udzielanych na węzły przedstawionego grafu. Zakresy mogą być:

- I. równe (zakresy dwóch autoryzacji AC_1 i AC_2 są równe wtedy i tylko wtedy, gdy znaczniki ich węzłów są identyczne jak na rysunku 2a),
- II. umiejscowienie AC_1 może zawierać AC_2 (zakres autoryzacji AC_1 obejmuje zakres AC_2 wtedy tylko, gdy znacznik węzła na którym udzielono AC_1 jest korzeniem (poprzednikiem)¹ znacznika węzła AC_2 – węzeł AC_2 jest wywodzony z AC_1 – rys. 2c, 2e),

¹ Znacznik węzła $ns_1 = a_1, a_2, \dots, a_m$ jest poprzednikiem znacznika węzła $ns_2 = b_1, b_2, \dots, b_n$ wtedy gdy $m < n \wedge \forall_{i \leq m} a_i = b_i$

- III. AC_1 może być zawarte w AC_2 (zakres autoryzacji AC_1 jest zawarty w AC_2 wyłącznie wtedy, gdy znacznik węzła AC_2 jest korzeniem znacznika AC_1 – węzeł AC_1 jest wywodzony z AC_2 – rys. 2d, 2f),
- IV. AC_1 może mieć wspólnego zakresu z AC_2 (rys. 2g),
- V. różne (jeśli przypadki I, II, III i IV nie występują – rys. 2b).

Do porównywania znaczników węzłów zdefiniujemy trzy następujące operatory:
 Niech $ns_1 = a_1, a_2, \dots, a_m$ i $ns_2 = b_1, b_2, \dots, b_n$ będą dwoma znacznikami węzła.
 Możemy powiedzieć, że [Wiec 94b]:

znacznik ns_1 jest różny od znacznika ns_2 ($ns_1 \neq ns_2$), wtedy gdy

$$m \neq n \vee (m = n \wedge \exists_{i \leq m} a_i \neq b_i);$$

znacznik ns_1 zawiera znacznik ns_2 ($ns_1 \supset ns_2$), wtedy gdy

$$m \geq n \wedge \forall_{i \leq n} a_i = b_i;$$

znacznik ns_1 zawarty w znaczniku ns_2 ($ns_1 \subset ns_2$), wtedy gdy

$$m \leq n \wedge \forall_{i \leq m} a_i = b_i.$$

Z podanej klasyfikacji umiejscowień autoryzacji i pokazanej na rysunku 2 możemy podsumować zasady zgodności dwóch autoryzacji AC_1 i AC_2 w następujący sposób:

- I. Jeśli autoryzacja AC_1 jest udzielana innemu użytkownikowi niż AC_2 , to we wszystkich przypadkach są one zawsze kompatybilne bez względu na to jakie są rodzaje i typy przywilejów.
- II. W przypadku (b) autoryzacje zawsze są kompatybilne, ponieważ są udzielane na dwóch różnych węzłach, które nie mają wspólnego komponenta.
- III. W przypadku (a) zawsze autoryzacje są zgodne, ponieważ późniejsza autoryzacja unieważnia wcześniejszą kiedy udzielane są jawnie na tym samym węźle.
- IV. W przypadkach (c) i (d) autoryzacje także są zgodne ponieważ późniejsza unieważnia wcześniejszą i ze względu na to że nie mają wspólnego węzła mającego więcej niż jednego przodka.
- V. Przypadki (e), (f) i (g), autoryzacje nie są kompatybilne jeśli ich znak (pozytywny, negatywny) jest różny i jedna z nich pozostaje odrzucona z wspólnego węzła mającego więcej niż jednego przodka ze względu na ich zakres który go obejmuje. Przebieg autoryzacji jest według algorytmu przedstawiony niżej (algorytm autoryzacji na grafie).

Aby rozwiązać problem niezgodności udzielanych autoryzacji zastosujemy następujące metody udzielania autoryzacji na grafach acyklicznych.

W celu określenia udzielenia autoryzacji trzeba najpierw wyznaczyć przodków głównych znaczników, które mają więcej niż jednego poprzednika. Do określenia głównego przodka danego węzła stosujemy wcześniej podane zasady określania

głównego przodka. Ta operacja pozwala wyeliminować problem niezgodności autoryzacji.

Zauważmy, że na rysunku 1, podgraf zakorzeniony przez węzeł o znaczniku 0.1 obejmuje węzły ($0.1.1$, $0.1.1.1$, $0.2.1$, $0.2.1.1$, $0.2.1.2$ i $0.2.2.1.1$), ale nie wszystkie zostały wywodzone z tego węzła.

Pierwsze dwa węzły ($0.1.1$ i $0.1.1.1$) zostały wprowadzone z niego, natomiast cztery ostatnie ($0.2.1$, $0.2.1.1$, $0.2.1.2$ i $0.2.2.1.1$) są wywodzone z innego węzła (z innych węzłów); nasuwa się następujące pytanie: czy te węzły należą do zakresu autoryzacji udzielanej na tym węźle czy też nie?

Aby odpowiedzieć na to pytanie i ustalić umiejscowienie autoryzacji w DAG, trzeba najpierw definiować rozszerzenie znacznika węzła następująco:

Mówimy, że zk_1 jest rozszerzeniem znacznika zk_2 , jeśli $zk_1 > zk_2$.

Wynika z tego, że znacznik węzła $0.2.1$ jest rozszerzeniem następujących znaczników: 0 i 0.2 . Znacznik węzła $0.2.1.1$ jest rozszerzeniem 0 , 0.2 i $0.2.1$, znacznik $0.2.1.2$ jest rozszerzeniem 0 , 0.2 i $0.2.1$, natomiast znacznik $0.2.2.1.1$ jest rozszerzeniem 0 , 0.2 , $0.2.2$ i $0.2.2.1$ (rys. 2).

Tak więc autoryzacja nadawana jest na węźle 0.1 dotyczy tylko węzłów o znacznikach 0.1 , $0.1.1$ i $0.1.1.1$. Węzły te są otoczone przerywaną linią. Zwróćmy uwagę, że zasięg autoryzacji rozważonej nie obejmuje węzłów: $0.2.1$, $0.2.1.1$, $0.2.1.2$ i $0.2.2.1.1$, które należą do podgrafu zakorzenionego przez 0.1 , ponieważ nie są one jego rozszerzeniami.

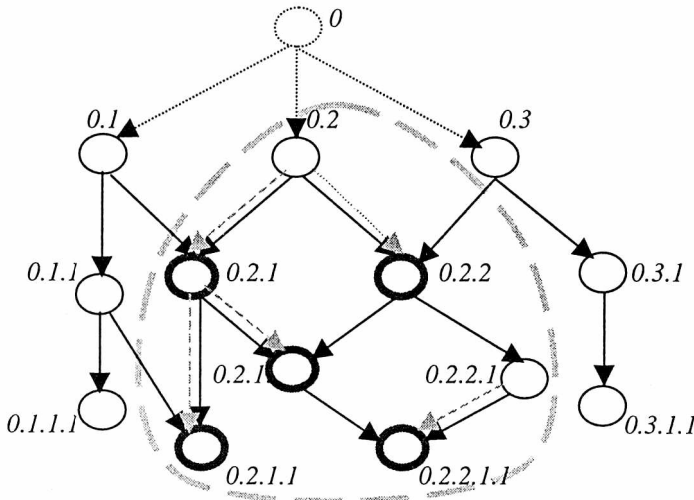
Algorytm autoryzacji na grafie wykonywany jest w następujących etapach:

1. Autoryzacja jest udzielana na korzeń podgrafu.
2. Cały podgraf jest rekurencyjnie przeglądany, w celu znajdowania węzłów mających więcej niż jednego bezpośredniego przodka.
3. Każdy znaleziony taki węzeł (mający więcej niż jednego bezpośredniego przodka), sprawdzany, czy jest udzielana na niego autoryzacja, lub czy jego znacznik jest rozszerzeniem innego autoryzowanego znacznika.
4. Jeśli nie, to autoryzacja zostaje przyjęta od momentu jej udzielania i pozostaje ważna do końca *ow* (okres ważności).
5. W przeciwnym wypadku (węzeł jest autoryzowany, lub jest rozszerzeniem autoryzowanego węzła), sprawdza się czy jest dla tego samego użytkownika.
6. Jeżeli nie, autoryzacja zostaje przyjęta od momentu jej udzielania i pozostaje ważna do końca *ow* (okres ważności).
7. W przeciwnym razie (dla tego samego użytkownika) sprawdza się czy autoryzacje nie zostają w kolizji (czy autoryzacje są zgodne).
8. W przypadku zgodności autoryzacji, zostaje przyjęta od momentu jej udzielania i pozostaje ważna do końca *ow*, natomiast jeśli nie są one zgodne będzie odrzucona.

Do zilustrowania algorytmu autoryzacji z uwzględnieniem uwarunkowań czasowych, opisujemy następujące dwie przykładowe autoryzacje:

- 1) $AC_1 = \text{auth}(s, 0.2, \text{Read}, h, 1.X.2002, 2_months)$,
- 2) $AC_2 = \text{auth}(s, 0.1, \text{Write}, h, 1.X.2002, 2_months)$.

Autoryzacja AC_1 udzielana jest na podgrafie zakorzenionym przez węzeł 0.2 i jest rozpowszechniana na wszystkie węzły w pod hierarchii tego węzła. Zasięgiem tej autoryzacji jest otoczony przerywaną linią (rys. 3). Podgraf tego węzła obejmuje sześć węzłów mających więcej niż jednego bezpośredniego przodka (0.2.1, 0.2.2, 0.2.1.1, 0.2.1.2, 0.2.2.1 i 0.2.2.1.1) i są one rozszerzeniem znacznika zakorzenionego w podgrafie 0.2.



Rys. 3. Przykład autoryzacji AC_1 na węzle skierowanego acyklicznego grafu 0.2

Autoryzacja AC_2 udzielana jest na podgrafie zakorzenionym przez węzeł 0.1. Taka autoryzacja przebiegać będzie następująco: w pierwszym etapie autoryzacja

$$AC_2 = \text{auth}(s, 0.1, \text{Write}, h, 15.X.2000, 2_months),$$

jest rozprzestrzeniana na wszystkie węzły, które są wywodzone z węzła 0.1 (są one rozszerzeniem tego węzła (0.1, 0.1.1 i 0.1.1.1)). Jej zakres otoczony jest przerywaną linią (rys. 4). Następnie szukane są inne węzły, z którymi związany jest ten węzeł (0.1) na niższym poziomie. Po odnalezieniu takich węzłów (0.2.1), taka sama autoryzacja będzie automatycznie implikowana na te węzły (autoryzacje AC_1 i AC_2 są kompatybilne – znak przywileju dostępu jest taki sam).

W tym przypadku autoryzacja

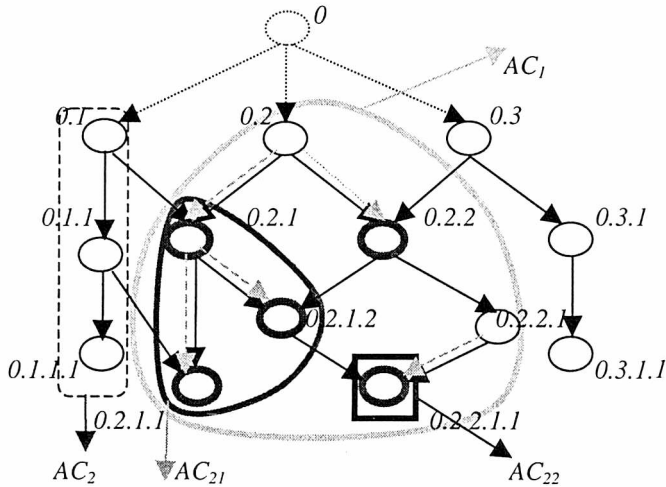
$$AC_2 = \text{auth}(s, 0.1, \text{Write}, h, 1.X.2002, 2_months) \Rightarrow$$

$$AC_{21} = \text{auth}(s, 0.2.1, \text{Write}, h, 5.X.2002, 2_months) \Rightarrow$$

$$AC_{22} = \text{auth}(s, 0.2.2.1.1, \text{Write}, h, 1.X.2002, 2_months),$$

oznacza to, że autoryzacja na węzle 0.1 implikuje taką samą autoryzację na węzle 0.2.1, która z kolei implikuje taką samą na węzle 0.2.1.2 i na węzle 0.2.2.1.1. Zakresy

tych autoryzacji pokazane są na rysunku 4. W przypadku kiedy udzielamy negatywną autoryzację na węźle 0.1 była by rozprzestrzeniana tylko na te węzły, które są rozszerzeniem znacznika tego węzła (tj. $0.1.1$ i $0.1.1.1$), natomiast nie została by przyjęta na węźle $0.2.2$ (według algorytmu).



Rys. 4. Przykład udzielania autoryzacji AC_1 i AC_2 na skierowanego acyklicznego grafu

PODSUMOWANIE

W pracy zaproponowano nowe podejścia do zarządzania autoryzacją w obiektowej bazie danych z uwzględnieniem uwarunkowań czasowych w nadawaniu i odwoływaniu uprawnień. Przedstawiona autoryzacja jest definiowana jako rozszerzenie autoryzacji podstawowej o tzw. znaczniki węzłów, które zawierają informacje o umiejscowieniu węzłów objętych autoryzacją w odpowiednich hierarchiach i grafach. Informacja ta jest odpowiednikiem sekwencji autoryzacji intencjonalnych zakładanych od korzenia drzewa (grafu) do odpowiedniego węzła. Istotnie ułatwia to wykrywanie niezgodności autoryzacji żądanej z autoryzacjami wcześniej ustawionymi.

Korzyści zastosowania zaproponowanego podejścia są szczególnie widoczne w bazach danych, w których dane są semantycznie powiązane skomplikowanymi hierarchiami i grafami, tak jak w obiektowych bazach danych.

LITERATURA

- [AlMa 03] Al-Awady A., Mazur Z., *Autoryzacja w wielowersyjnych obiektowych bazach danych z uwzględnieniem uwarunkowań czasowych*. Bazy Danych, Prace Naukowe Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej, Zeszyt 4, Wrocław, 2003.
- [CeJo 90] Cellary W., Jomier G., *Consistency of Versions in Object-Oriented Databases*, Proc. 16 VLDB Conference, Brisbane, Aug. 1990, pp. 432–441.
- [Kim 96] Kim W., *Wprowadzenie do obiektowych baz danych*, WNT, Warszawa 1996.
- [OzVa 99] Ozsu M., T., Valduriez P., *Principles of Distributed Database Systems* (2nd Edition), Prentice Hall, Upper Saddle River, NJ, 1999.
- [RyWi 94] Rykowski J., Wierczycki W., *Using versioned object-oriented data in programming languages*, Microprocessing and Microprogramming, 40, 901–904, 1994.
- [SBP 01] Stokłosa J., Bilski T., Pankowski T., *Bezpieczeństwo danych w systemach informatycznych*, Wydawnictwo Naukowe PWN, Warszawa – Poznań 2001.
- [Subi 99] Subieta K., *Słownik terminów z zakresu obiektowości*, Akademicka Oficyna Wydawnicza PLJ, Warszawa 1999.
- [Wiec 94a] Wierczycki W., *Version management in design databases*, Proc. 9th Int. Conf. On Computer and Information Sciences, Antalya, November 1994.
- [Wiec 94b] Wierczycki W., *Hierarchical locking without intentional locks*, Poznań 1994.
- [WiRy 94] Wierczycki W., Rykowski J., *Version support for CAD/CASE databases*, East-West Database Workshop, Klagenfurt, Austria, September 1994.

AUTHORIZATION METHODS FOR DIRECT ACYCLIC GRAPH IN OBJECT-DATABASES WITH TIME RELATED ASPECTS

In this paper a new approach to control authorization, which provide simple and efficient hierarchical authority strategy for several hierarchies and/or DAGs (*Direct Acyclic Graphs*) is proposed. The basic of this approach is an extention of classical authorization by node stamps which contain information about the position of the node in all the hierarchies. This information is equivalent to the sequence of intentional authorization from the root to a given node. Substantially simplifies detection of authorization incompatibilities. Advantages of this approach become particularly beneficial i case of database which data are complex hierarchies and DAGs. This a typical case of *Object-Oriented Databases*.

Lech MADEYSKI*

XML W BAZACH DANYCH

Java i XML to dwa bardzo modne i często używane ostatnio słowa – klucze. Trudno sobie jednak wyobrazić poważne aplikacje korporacyjne bez użycia baz danych. Są one integralną częścią zarówno aplikacji zastanych (legacy applications), jak i nowych – tworzonych w architekturze wielowarstwowej – aplikacji internetowych biznesu elektronicznego. Podobnie jak Java zapewnia przenośność tworzonych aplikacji, czyli ich niezależność od platformy sprzętowo-systemowej, tak XML zapewnia przenośność danych. Nic więc dziwnego, że Java – a w szczególności XML – wywiera duży wpływ również na stosunkowo hermetyczną do tej pory dziedzinę baz danych. Można śmiało stwierdzić, że wraz z popularyzacją XML także w bazach danych nastąpiła swego rodzaju rewolucja. Pojawiło się całe spektrum nowych rozwiązań, które zostaną przedstawione w niniejszym artykule. Użyta klasyfikacja bazuje na klasyfikacji stosowanej przez Ronalda Bourreta [1].

1. WSTĘP

Jest wiele czynników decydujących o wyborze i architekturze środowiska bazodanowego. Część z nich ma charakter pomocniczy, inne odgrywają w tym procesie szczególnie istotną rolę. Jeżeli środowisko bazodanowe ma wspierać XML (*eXtensible Markup Language*), to najważniejszym czynnikiem mającym wpływ na wybór bazy danych jest jej przeznaczenie. Dokumenty XML można bowiem podzielić na dwie główne kategorie:

1. Dano-centryczne (*data-centric*).
2. Dokumento-centryczne (*document-centric*).

Dano-centryczne dokumenty XML to takie dokumenty, w których XML jest używany do transportu danych np. zamówienia, notowania giełdowe, dynamicznie generowane katalogi online czy listy adresowe na podstawie znanych, regularnych zbiorów danych. Ich fizyczna struktura – np. kolejność elementów siostrzanych lub sposób przechowywania danych (w atrybutach, elementach) – jest często nieistotny.

Dokumento-centryczne dokumenty XML to np. książki, listy elektroniczne, broszury. Charakteryzują się one nieregularną lub mało regularną strukturą, a ich struktura fizyczna (np. kolejność elementów) jest istotna.

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Wybrzeże Wyspiańskiego 27, madeyski@ci.pwr.wroc.pl.

Zaprezentowany podział dokumentów XML na dwie kategorie jest bardzo pomocny przy doborze odpowiedniej bazy danych, chociaż w praktyce podział ten nie zawsze jest tak klarowny. Może bowiem zdarzyć się, że dokumenty XML dokumento-centryczne zawierają regularnie strukturyzowane dane (np. metadane dokumentu) i na odwrót – dokumenty XML dano-centryczne mogą np. zawierać dane o nieregularnej strukturze.

2. XML W BAZACH DANYCH

Rozwiązania bazodanowe dobiera się w zależności od kategorii dokumentów XML, które mają być obsługiwane.

Do pobierania i składowania danych w dokumentach dano-centrycznych, potrzebna jest baza danych dostosowana do przechowywania tych danych (np. relacyjna lub obiektowa) i oprogramowanie odpowiedzialne za transfer (import i eksport) danych w postaci dokumentów XML. Oprogramowanie to może być wbudowane w bazę i wtedy mówimy, że taka baza oferuje mechanizm translacji BD-XML (*XML-enabled database*) lub może być zewnętrznym tłumaczem BD-XML, czyli produktem klasy *Middleware*.

Do pobierania i składowania danych w dokumentach dokumento-centrycznych, potrzebna jest Nativna Baza XML (*Native XML Database*), czyli baza danych specjalnie zaprojektowana do przechowywania dokumentów XML lub System Zarządzania Zawartością (*Content Management System*), będący swego rodzaju aplikacją zaprojektowaną do zarządzania dokumentami zbudowaną z wykorzystaniem natywnej bazy danych. Natywne Bazy Danych jak i Systemy Zarządzania Zawartością są zaprojektowane, by przechowywać fragmenty zawartości np. rozdziały, elementy słownika, metadane dokumentu. Systemy Zarządzania Zawartością, choć zwykle używają Natywnej Bazy XML, mają dodatkową funkcjonalność (edycja zawartości, kontrola wersji, zarządzanie przepływem dokumentów).

Powyższe ogólne zasady są pomocne, nie są jednak ścisłe. Wiąże się to z tym, że oferowane rozwiązania gwałtownie ewoluują, a granice pomiędzy poszczególnymi grupami produktów mają charakter arbitralny. Na przykład granice pomiędzy tradycyjnymi bazami danych a natywnymi bazami danych XML ulegają zatarciu. Tradycyjne bazy danych powszechnie już oferują wsparcie dla XML, a natywne bazy XML zaczynają wspierać składowanie dokumentów w tradycyjnych (relacyjnych) bazach danych.

3. GRUPY PRODUKTÓW

W związku z pojawieniem się szerokiego spektrum produktów bazodanowych wspierających XML konieczne stało się ich pogrupowanie i określenie jednolitej terminologii. W związku z tym wyodrębniono następujące grupy produktów [1]:

- Translatory DB-XML (*Middleware*).
- Bazy danych wspierające transfer dokumentów XML (*XED – XML Enabled Databases*).
- Natywne bazy XML (*NXD – Native XML Databases*).
- Serwery XML (*XS – XML Servers*) i serwery aplikacji XML (*XAS – XML Application Servers*).
- Systemy zarządzania zawartością (*CMS – Content Management Systems*).
- Motory kwerend XML (*XQE – XML Query Engines*).
- Translatory XML-Obiekty (*XDB – XML Data Binding*).

W dalszej części tego rozdziału powyższe grupy produktów zostaną pokrótce omówione.

3.1. TRANSLATORY DB-XML

Translatory DB-XML (*Middleware*) zapewniają translację pomiędzy dokumentami XML a bazami danych. Są one zwykle wykorzystywane w przypadku aplikacji dano-centricznych. Gdy są połączone z komercyjną bazą danych jako jej integralna część, to o takiej bazie mówi się, że wspiera transfer dokumentów XML (*XML Enabled Database*).

Znane są dwa podejścia wykorzystywane w produktach tej klasy:

- Mapowanie bazujące na szablonie (*template based mapping*) – zwykle ograniczające się do transformacji BD->XML.
- Mapowanie bazujące na modelu (*model based mapping*) – bardziej nadające się do realizacji dwukierunkowej transformacji BD<->XML.

Przykładowymi przedstawicielami tej grupy produktów są dwa komercyjne produkty firmy Oracle: XML SQL Utility for Java i XSQL Servlet oraz XML-DBMS, będący produktem Open Source.

3.1.1. XML SQL UTILITY FOR JAVA I XSQL SERVLET FIRMY ORACLE

Produkt XML SQL Utility for Java wykorzystuje mapowanie bazujące na modelu zapewniając dwukierunkowe mapowanie BD<->XML. XSQL Servlet jest serwiletem używającym XML SQL Utility for Java do transferu danych z BD do XML.

W przypadku, gdy dane są transferowane z BD do XML, użytkownik dostarcza wyrażenie SELECT lub JDBC ResultSet zaś rezultaty są zwracane w postaci dokumentu XML lub DOM (*Document Object Model*). Gdy dane są transferowane z XML do BD, użytkownik dostarcza dokument XML lub DOM.

3.1.2. XML-DBMS

XML-DBMS [6] jest produktem klasy Open Source, wykorzystującym mapowanie bazujące na modelu, zapewniając dwukierunkowe mapowanie BD<->XML. Używa obiektowo-relacyjnego mapowania, które jest opisane poprzez język mapowania bazujący na XML. Użytkownicy mogą definiować, w jaki sposób elementy, atrybuty, PCDATA są mapowane w BD. Język jest dość elastyczny i zapewnia np. zdolność do przechowywania własności (atrybutów) w tabeli klasy lub w oddzielnej tabeli własności czy możliwość wyboru formatu parsowanej daty, czasu itd.

XML-DBMS może także generować relacyjny schemat (w postaci wyrażenia CREATE TABLE) na podstawie dokumentu DTD.

Struktura typowej aplikacji xml-dbms:

1. Stworzenie obiektu map.
2. Przekazanie obiektu map do obiektu domToDBMS lub DBMSToDom.
3. Odbiór i manipulacja rezultatami translacji.

3.2. BAZY WSPIERAJĄCE TRANSFER DOKUMENTÓW XML

Bazy danych wspierające transfer dokumentów XML (*XML Enabled Databases*), to bazy danych (zwykle relacyjne) zawierające rozszerzenie (wykorzystujące mapowanie bazujące na szablonie lub modelu) służące do transferu danych pomiędzy dokumentami XML, a samą bazą danych. Są one zwykle wykorzystywane w przypadku aplikacji dano-centrycznych.

Przykładami produktów, które można zaliczyć do tej grupy są m.in.:

- Oracle 8i, 9i ;
- Informix;
- DB2 XML Extender and DB2 Text Extender;
- Microsoft Access 2002;
- Microsoft SQL Server 2000.

Produkty te oferują bezpośredni dostęp do danych poprzez ODBC/JDBC.

Przykładowo Oracle 9i wprowadza obiektowy typ danych XMLType [4, 5], który przechowuje dokumenty XML jako CLOB. Umożliwia m.in. wykonywanie SQL-owych operacji na zawartości XML-owej. Na dokumentach XML mogą być

realizowane kwerendy (z użyciem XPath), co pozwala użytkownikom nie tylko pobierać dane z tych dokumentów w aplikacjach, ale także budować indeksy poszczególnych elementów i atrybutów w dokumentach.

3.3. NATYWNE BAZY XML

Natywne bazy XML (*Native XML Databases*) definiują logiczny model dokumentu XML zawierający przynajmniej elementy, atrybuty, PCDATA i porządek dokumentów (np. XPath, XML Infoset, modele implikowane przez DOM i SAX). Natywne bazy XML są wykorzystywane zarówno w przypadku aplikacji dano-centrycznych jak i dokumento-centrycznych.

W przypadku natywnych baz XML dokument XML jest fundamentalną jednostką (logicznego) składowania, tak jak w przypadku relacyjnych baz danych, jest nią wiersz tabeli. Nie jest natomiast wymagany żaden konkretny model fizycznego składowania. Natywna baza XML może bazować na relacyjnej, hierarchicznej lub obiektowej bazie danych albo też wykorzystywać własny format składowania np. indeksowane, kompresowane pliki.

Można wyróżnić dwie kategorie natywnych baz XML:

- Ze składowaniem tekstowym. Dokumenty są składowane w formie tekstowej (np. jako BLOB w relacyjnej bazie danych, plik w systemie plików z indeksami dokumentów w XML, własny, zoptymalizowany system składowania z indeksami, wsparciem transakcji itp.).
- Ze składowaniem binarnym. W tym przypadku składowany jest binarny model dokumentu (np. DOM).

W natywnych bazach XML jedynym interfejsem do danych jest interfejs bazujący na XML: XPath, DOM lub API bazujące na XML np. XML:DB API. To ostatnie jest interesującą koncepcją, która znalazła już praktyczne zastosowanie w przypadku natywnej bazy XML Apache Xindice.

XML:DB API zostało stworzone przez XML:DB Initiative [2] w celu ułatwienia tworzenia aplikacji wykorzystujących XML w bazach danych. Funkcjonalnie jest ono tym dla natywnych baz XML, czym ODBC lub JDBC dla relacyjnych baz danych.

XML:DB API bazuje na koncepcji kolekcji, które przechowują zasoby. Zasobem może być dokument XML, binarny BLOB lub jakiś inny typ jeszcze nie zdefiniowany. Kolekcje mogą być organizowane w sposób hierarchiczny, ich architektura przypomina system plików. Kolekcje oferują serwisy pozwalające na realizację kwerend dokumentów XML z wykorzystaniem XPath lub uaktualnianie zasobów w transakcyjnie bezpieczny sposób. Kwerendy mogą dotyczyć nie tylko pojedynczych dokumentów XML, ale także kolekcji.

Nie wszystkie natywne bazy XML wymagają, by z kolekcją był związany schemat. Oznacza to, że w takiej bazie, określanej mianem niezależnej od schematu (*schema-independent*), można przechowywać dowolne dokumenty XML w kolekcjach bez względu na schemat. Właśnie takie rozwiązanie przyjęto w przypadku bazy Apache Xindice [3]. Jest ono bardzo elastyczne i ułatwia życie programistom, ale należy pamiętać o konsekwencjach (integralność danych). Do niektórych zastosowań należy więc wybierać natywną bazę XML wspierającą schematy lub szukać innego rozwiązania problemu przechowywania danych XML.

Xindice zawiera wielowątkowy motor bazodanowy optymalizowany dla danych XML. Dokumenty są przechowywane w formie skompresowanej. Xindice zawiera motor kwerend bazujący na XPath. Kolekcje są indeksowane w celu zwiększenia efektywności realizowanych kwerend. W celu aktualizacji danych Xindice implementuje XML:DB XUpdate. Zawiera również kompletny zestaw narzędzi linii komend i trójwarstwowe API:

- XML:DB API to główne API do tworzenia aplikacji dbXML w Javie. Jest zbudowane na bazie dbXML Corba API.
- dbXML Corba API zapewnia dostęp do dbXML za pomocą innych języków niż Java. Jest zbudowane na bazie Core Server API. W przyszłości będzie ono zastąpione nowym sieciowym API.
- Core Server API jest wewnętrznym API motoru bazodanowego.

3.4. SERWERY XML I SERWERY APLIKACJI XML

Serwery XML (*XML Servers*) są nie tyle bazami danych, co kompletnymi platformami do tworzenia aplikacji eBusiness czy aplikacji internetowych. Wykorzystują dane w postaci dokumentów XML w procesie natywnej komunikacji (wysyłania i odbierania danych) i składowania (persystencji). Zapewniają dostęp do różnych danych np. zastanych baz danych, systemów plików itd. wspierając procesy integracji istniejących systemów. Serwery XML są zwykle wykorzystywane w przypadku aplikacji dano-centricznych.

Serwery aplikacji XML (*XML Application Server*) zwykle bazują na szablonach i używają języków skryptowych lub własnych języków znaczników, w które wbudowane są wyrażenia SQL do odczytywania danych i dynamicznego budowania dokumentów XML.

Produkty z tej grupy wykorzystują zwykle serwery WWW (*Web servers*), które obsługują komunikację poprzez Web i serwują statyczne dokumenty. Serwery aplikacji XML są wykorzystywane zarówno w przypadku aplikacji dano-centricznych, jak i dokumento-centricznych. W tej grupie produktów umieszcza się na przykład Apache Cocoon [7] czy ColdFusion firmy Macromedia (dawniej Allaire) [8].

Linia podziału pomiędzy serwerami XML a serwerami aplikacji XML nie jest klarowna. Zaproponowany przez Bourreta podział może się wydawać dyskusyjny. Wykorzystywanie języków skryptowych lub własnych języków znaczników nie wydaje się być jedynym i głównym wyznacznikiem przynależności produktu do grupy serwerów aplikacji, choć ich popularność jest bezdyskusyjna. Przykładowo na platformie Java serwery zawierające jedynie kontenery EJB nie wykorzystujące języków skryptowych ani własnych języków znaczników są zaliczane do serwerów aplikacji.

3.5. SYSTEMY ZARZĄDZANIA ZAWARTOŚCIĄ

Systemy zarządzania zawartością (*Content Management Systems*) spełniają zwykle rolę fasady natywnych baz XML, które są niejako ukryte przed użytkownikiem.

Systemy zarządzania zawartością są dostosowane do zarządzania dokumentami (*document-centric documents*), a także oferują dodatkową funkcjonalność (np. edycję zawartości, kontrolę wersji, dostęp dla wielu użytkowników). W rezultacie zapewniają składowanie, pobieranie i tworzenie nowych dokumentów z istniejących fragmentów (zawartości).

Systemy zarządzania zawartością różnią się od natywnych baz XML tym, że oferują więcej funkcjonalności związanej z zarządzaniem dokumentami (np. kontrolę wersji).

Większość takich systemów może publikować XML w sieci WWW, ale w odróżnieniu od serwerów aplikacji XML, systemy zarządzania zawartością są głównie zaprojektowane do zarządzania dokumentami (*document-centric documents*) podczas gdy XML'owe serwery aplikacji są zaprojektowane do serwowania zarówno dano-centrycznych jak i dokumento-centrycznych dokumentów w sieci WWW.

Do tej grupy produktów można zakwalifikować komercyjne rozwiązania oferowane przez firmę Documentum [9], jak również projekt Prowler [10] rozwijany na zasadach Open Source. Ten ostatni jest jednym z elementów większej całości – portalu informacyjnego przedsiębiorstwa (*Enterprise Information Portal*).

3.6. MOTORY KWEREND XML

Motory kwerend XML (*XML Query Engines*) są samodzielnymi programami/motorami zapewniającymi tylko i wyłącznie realizację kwerend dokumentów XML. Mogą implementować różne języki – oczywiście najlepiej te, które są standaryzowane przez W3C, a więc XPath i XQuery.

3.7. TRANSLATORY XML-OBIEKTY

Translatory XML-Obiekty (*XML Data Binding*) wiążą dane zawarte w dokumentach XML z obiektami zaprojektowanymi specjalnie dla tych danych. Zaletą wykorzystania w aplikacjach (zwykle dano-centrycznych) rozwiązań z tej grupy jest prosty i bardziej naturalny niż w przypadku DOM, SAX czy JAXP sposób operowania na danych. Operacje na dokumentach XML są bowiem realizowane na związanych z nimi obiektami np. ziarnami Javy (*JavaBeans*). Obiecującymi rozwiązaniami wykorzystującymi wiązanie danych zawartych w dokumentach XML z obiektami Javy są JAXB (*Java Architecture for XML Binding*) [11] firmy Sun i Castor XML [12] organizacji ExoLab Group wspierającej tworzenie oprogramowania na zasadach Open Source.

Produkty należące do tej grupy mogą niekiedy przechowywać i odtwarzać obiekty z bazy danych.

4. PODSUMOWANIE

W artykule skrótkowo zaprezentowano szeroki wachlarz produktów i zagadnień związanych ze wsparciem XML w systemach bazodanowych. Pewne jest, że powszechna akceptacja dla XML spowodowała duże zmiany w segmencie rynku baz danych. Natywne bazy XML i inne produkty bazodanowe wspierające XML nie są oczywiście panaceum na wszelkie bolączki. Rozszerzają one paletę dostępnych narzędzi, a jeśli zostaną odpowiednio użyte, mogą przynieść twórcom systemów wiele korzyści. Autor ma nadzieję, że treści zaprezentowane w tym artykule będą pomocne w dokonaniu świadomego wyboru odpowiednich narzędzi.

Trudno przesądzać, jaki kształt przyjmą systemy bazodanowe w przyszłości. Możliwe są różne scenariusze począwszy od rozwiązań uniwersalnych, które będą ewoluować z obecnych systemów relacyjnych baz danych po rozwiązania kombinowane łączące różne typy repozytoriów. Wciąż również pojawiają się nowe koncepcje (jak chociażby *XML Data Binding*) mogące wywrzeć istotny wpływ na kierunek dalszego rozwoju aplikacji internetowych, a w konsekwencji również systemów bazodanowych wspierających XML.

LITERATURA

- [1] Strona domowa Ronalda Bourreta, <http://www.rpbourret.com>
- [2] XML:DB Initiative, <http://www.xmldb.org>
- [3] Projekt Apache Xindice, <http://xml.apache.org/xindice/>

- [4] Oracle Technet, <http://technet.oracle.com/tech/xml/content.html>
- [5] Oracle, Database Support for XML, http://download-east.oracle.com/otndoc/oracle9i/901_doc/appdev.901/a88894/adx05xml.htm#1012692
- [6] Ronald Burret, XML-DBMS, <http://www.rpbouret.com/xmldbms/index.htm>
- [7] Projekt Apache Cocoon, <http://xml.apache.org/cocoon/index.html>
- [8] Macromedia ColdFusion Server, <http://www.macromedia.com/software/coldfusion/>
- [9] Documentum, <http://www.documentum.com/products/content/index.html>
- [10] Prowler, http://www.infozone-group.org/projects_main.html
- [11] Sun JAXB, <http://java.sun.com/xml/jaxb/index.html>
- [12] ExoLab Group, <http://castor.exolab.org>

XML IN DATABASES

Java and XML are hot topics today. However it is difficult to realize serious corporate portals or Internet applications without databases. Databases are important parts of legacy applications as well as multitiered eBusiness applications. Java supports applications portability through platform independence and XML supports data portability. It is no wonder that Java and especially XML exert an influence on database field. It appeared a wide range of new products which are presented in this article. The terminology is based on the proposed by Ronald Bourret [1].

*baza danych, Internet,
technologia internetowa,
aplikacja internetowa*

Klara DYCZKOWSKA *

TECHNOLOGIE UDOSTĘPNIANIA BAZ DANYCH W INTERECIE

Biorąc pod uwagę, że nie ma bardziej uniwersalnego narzędzia do pobierania informacji niż WWW oraz że nie ma lepszych technik przechowywania informacji niż bazy danych, ich połączenie umożliwia budowanie zadziwiająco użytecznych aplikacji. W niniejszej pracy zaprezentowano technologie udostępniania baz danych poprzez sieć Internet, o których można powiedzieć, że wyznaczają kierunki rozwoju współczesnych systemów, integrujących zalety baz danych, szerokie możliwości Internetu oraz potrzeby światowego biznesu.

1. WPROWADZENIE

Metody statycznego umieszczania informacji tekstowej w serwisach WWW należą do najprostszych. W początkach rozwoju internetowych witryn twórcy stron kopiowali zawartość baz danych i umieszczali je w serwisach jako zwykły tekst. Za każdym razem, gdy należało uaktualnić udostępniane informacje, wymieniana była cała strona na serwerze. Oczywiście było to zajęcie bardzo czasochłonne i niosło ze sobą ryzyko pojawienia się błędów. Szczególnie dużo pracy miały firmy, które świadczyły usługi prezentacji informacji ekonomicznych zmieniających się kilkanaście-kilkadziesiąt razy dziennie. Każda dezaktualizacja takich konwencjonalnych stron wymagała interwencji specjalisty informatyka.

Najprostszym krokiem do poprawy istniejącej sytuacji było oczywiście rozwiązanie, polegające na połączeniu technologii baz danych i stron WWW. Wkrótce zaczęto doskonalić technologie, umożliwiające dynamiczne generowanie zawartości witryn internetowych, na których informacja odzwierciedla stan serwera podłączonej bazy danych w momencie żądania usługi przez klienta. Stosowane obecnie rozwiązania przedstawiono poniżej w podziale na dwie podstawowe grupy:

- metody prezentacji (*ang. client-side*) – stosowane po stronie przeglądarki;
- metody generacji (*ang. server-side*) – pracujące po stronie serwera.

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Skwer Idaszewskiego 1, dyczkowska@ci.pwr.wroc.pl

2. METODY PREZENTACJI (*CLIENT-SIDE*)

Tradycyjnie postrzegana prezentacja polega na takim przedstawieniu wygenerowanej treści, aby była zrozumiała i atrakcyjna dla użytkownika. Aktualne serwisy są jednak nie tylko atrakcyjne wizualnie. Przeglądając strony można korzystać z interakcji – np. wypełniać ankiety, głosować, wybierać treść przygotowywaną specjalnie dla konkretnego użytkownika. Nawet przesuwając myszkę nad obrazkiem można zobaczyć jakiś ciekawy efekt. Generalnie, dzięki oprogramowaniu po stronie klienta, możliwe jest:

- zapewnienie dowolnego sposobu prezentacji informacji;
- zbieranie danych od użytkowników; skrypty i aplety są w stanie kontrolować poprawność wprowadzanych danych i to nie tylko w zakresie podstawowym jak to, czy pole jest puste czy nie, ale także sprawdzają, czy wprowadzona wartość jest właściwa danemu polu;
- dokonywanie obliczeń na podstawie danych wprowadzonych do formularzy;
- wykonanie operacji na komputerze klienta; najlepszym przykładem jest tu mechanizm *cookie*, gdzie skrypt odszukuje plik na komputerze klienta, odczytuje z niego informacje i potrafi je wykorzystać.

Wielką zaletą stosowania rozwiązań programistycznych po stronie klienta jest możliwość zminimalizowania ilości danych przesyłanych przez sieć oraz odciążenie serwera. Niestety programy te często wymagają pewnych zasobów po stronie klienta, więc jeśli nie ma pewności co do środowiska, w jakim będą uruchamiane, to może to stanowić pewien problem.

2.1. JĘZYK JAVASCRIPT

Języki skryptowe, stworzone z myślą o pracy w środowisku przeglądarki, nie pozwalają bezpośrednio na wymianę informacji z bazami danych. Pozwalają one jednak na wstępne przetwarzanie danych – najczęściej kontrolę poprawności po stronie klienta. Jest to istotna sprawa, dzięki której odciąża się serwer i minimalizuje niepotrzebny ruch w sieci. Język skryptowy tym różni się od języka programowania, że program napisany w takim języku nie musi być kompilowany. Wykonywaniem poleceń języków skryptowych zajmuje się odpowiedni program zawierający interpreter komend, np. przeglądarka internetowa. Dzięki skryptom Javascript tworzy się strony, które dynamicznie reagują na zdarzenia po stronie użytkownika i umożliwiają komunikację pomiędzy klientem a serwerem. Przykładowe możliwości, jakie oferuje JavaScript, to [ROMOW 98]:

- wstępna analiza formularzy pod kątem błędów jeszcze przed wysłaniem ich zawartości na serwer wywołujący;
- operowanie na dacie i liczbach;

- dostęp do ustawień przeglądarki i monitora;
- dodawanie właściwości do obiektów tekstowych i graficznych;
- tworzenie dynamicznie dokumentu, generując jego zawartość na podstawie prostych baz danych.

Skrypty JavaScript uważane są za bezpieczne, gdyż nie mogą ingerować w system użytkownika. Dodatkowe poczucie bezpieczeństwa wiąże się z łatwością wglądu w ich źródło, a co za tym idzie, z możliwością analizy i sprawdzenia, co faktycznie robią.

2.2. JAVA APPLETS – PROGRAMY JAVY NA STRONACH WWW

Aplety (ang. *Applets*) są to niewielkie programy narzędziowe zrealizowane w języku Java – nowoczesnym, niezależnym od platformy sprzętowej i systemowej, zorientowanym obiektowo języku programowania. Stworzone są one w ten sposób, że możliwe jest ich wykonywanie jedynie z poziomu innej aplikacji, przykładowo z poziomu przeglądarki. Nie są to więc samodzielne programy. Aplety z reguły osadzone są w dokumentach HTML, dzięki czemu dokumenty hipertekstowe przestają być jedynie "tekstem", a stają się narzędziem, pozwalającym na pełną interakcję z użytkownikiem.

Dzięki apletom Javy można osiągać bardzo skomplikowane efekty, poczynając od poważnych zastosowań bazodanowych, a skończywszy na efektach graficznych. Zastosowanie apletów umożliwia uruchamianie z poziomu przeglądarki interaktywnych prezentacji multimedialnych lub nawet programów obliczeniowych, czy niezbyt skomplikowanych gier. Ich zdarzeniowy charakter pozwala na rejestrowanie ruchów myszki oraz naciśnięć klawiszy. Wielowątkowość pozwala na pisanie bardzo wydajnych i skomplikowanych algorytmów.

Należy podkreślić, iż aplety są całkowicie bezpieczne dla systemu operacyjnego użytkownika. Przeglądarki zdolne do uruchamiania apletów posiadają obiekt, zwany zarządcą (ang. *Security Manager*), który sprawdza, czy aplet nie narusza zasad bezpieczeństwa.

Zaletą Javy jest to, że jest ona niezależna od platformy sprzętowej i systemowej zarówno serwera jak i klienta. Aplety są kompilowane do postaci tzw. b-kodu i interpretowane dopiero w oknie przeglądarki klienta, niezależnie od tego czy pracuje na PC, Apple'u, HP, czy Sun'ie, niezależnie od tego, jaki ma system operacyjny i jaką przeglądarkę (musi tylko implementować maszynę wirtualną Javy JVM) – chociaż efekty mogą się nieco różnić. Niestety, brak spójności pomiędzy rozwiązaniami poszczególnych przeglądarek nie przyniósł tej technice wielkiego sukcesu.

Dodatkowo posiadają one pewną znaczącą wadę: witryny z apletami Javy ładują się znacznie dłużej, niż w przypadku zastosowania innych technologii, zapewniających tę samą funkcjonalność.

3. METODY GENERACJI

Generacja treści serwisu po stronie serwera, przy średnich i dużych przedsięwzięciach, jest dużo bardziej złożona niż sama prezentacja u klienta. Przy projektowaniu takich systemów stosuje się taką samą metodykę, jak przy tworzeniu każdego innego złożonego systemu informacyjnego. Istnieje wiele technologii wspomagających analityków, projektantów i programistów w czasie cyklu tworzenia systemu. Zwłaszcza w części programistycznej rozwój jest imponujący – w środowiskach implementacji systemów webowych istnieje całe bogactwo ofert.

Wybór technologii jest na ogół proporcjonalny do złożoności projektu – im bardziej złożony i docelowo niezawodny musi być system, tym bardziej zaawansowane techniki generacji treści muszą być zastosowane.

3.1. CGI – COMMON GATEWAY INTERFACE

CGI (ang. *Common Gateway Interface*) jest to pierwszy, najstarszy standard wymiany informacji pomiędzy przeglądarką klienta, a programem uruchamianym na serwerze. Wymiana informacji jest realizowana przez zmienne środowiskowe serwera oraz standardowe wejście oraz wyjście programu. Interfejs CGI definiuje sposób, w jaki serwer WWW może komunikować się z dowolnymi programami wykonywalnymi. Skrypty CGI są to programy umieszczone na serwerze i tam wykonywane. Jest to rozwiązanie na tyle standardowe, że do tworzenia skryptu CGI można zastosować właściwie dowolny język programowania: musi on mieć tylko funkcje lub metody pozwalające na dostęp do zmiennych środowiskowych oraz posiadać standardowe wejście i wyjście. Niemniej jednak najpopularniejszymi językami do pisania skryptów CGI jest Perl oraz C/C++.

Ponieważ CGI umożliwia uruchamianie na serwerze całkowicie dowolnych programów, zastosowanie tej technologii praktycznie nie ma ograniczeń. Do klasycznych zastosowań należą: liczniki odwiedzin, wyszukiwarki, książki gości, rozmaite ankiety i głosowania. Za pośrednictwem strony WWW możliwa jest także zmiana hasła konta pocztowego, czy wysyłanie faksów i SMS-ów. Z większym lub mniejszym udziałem skryptów CGI można też obsługiwać sklepy internetowe czy systemy elektronicznej bankowości.

Mechanizm działania

Podstawowa idea polega na pobraniu danych od klienta, przekazaniu ich do aplikacji, przetworzeniu i odesłaniu odpowiedzi do przeglądarki w postaci dokumentu HTML. Proces wymiany informacji między przeglądarką, serwerem, a aplikacją CGI składa się z następujących etapów [COLB 98]:

- Przeglądarka wysyła do serwera żądanie, zawierające URL wskazujący na skrypt .cgi. Z żądaniem przesyłane są dane w postaci mało zrozumiałego ciągu znaków, złożonego z sekwencji znaków alfanumerycznych połączonych znakami '+' i '&'. W ten sposób przeglądarka koduje znaki tak, aby dotarły one do serwera bez uszczerbku. Niestety nakłada to na programistę obowiązek rozszyfrowania tego ciągu.
- Serwer rozpoznaje, że wymagane jest uruchomienie programu wykonywalnego .cgi, sprawdza, czy są odpowiednie uprawnienia do wykonania tego zadania, a następnie powołuje nowy proces i przekazuje zadanie do aplikacji CGI;
- Aplikacja CGI jest uruchamiana i wykonuje przewidziane funkcje, a następnie zwraca wynik do serwera, z zachowaniem konkretnego formatu;
- Serwer przekazuje całą, gotową odpowiedź w postaci pliku HTML do przeglądarki klienta.

Zalety i wady

Zaletą skryptów CGI jest niezależność od przeglądarki klienta, gdyż całe przetwarzanie odbywa się po stronie serwera. Do zalet CGI należy też przenośność kodu. Raz napisany skrypt może być wykonywany na różnych platformach systemowo-sprzętowych, gdyż niemal każdy serwer zawiera jego implementację. Należy jednak wspomnieć, iż różnice między systemami operacyjnymi mogą czasem doprowadzić do pojawienia się pewnych niespójności w działaniu CGI.

Największą słabością CGI jest to, że nie bardzo nadaje się on do zastosowania jako interfejs do baz danych. Skrypt korzystający z bazy danych za każdym razem musi się do niej podłączyć oraz autoryzować. Powoduje to znaczny spadek wydajności serwisu, a także może wpłynąć na spowolnienie pracy bazy danych.

Inną wadą CGI jest mała efektywność działania. Za każdym razem serwer WWW musi powołać nowy proces, w którym uruchomiony zostanie skrypt CGI. Skrypt musi być wtedy wczytany do pamięci, uruchomiony i zostać zamknięty. Przy dużym obciążeniu serwera każdorazowe czytanie kodu CGI zajmuje dużo czasu i w efekcie całość działa dość wolno.

3.2. API WEB SERWERÓW

API (ang. *Application Programming Interface*) jest to metoda przypisana do systemu operacyjnego (lub innej aplikacji), umożliwiająca poprzez zestaw programowanych przerw, wywołań, formatów wymiany informacji, funkcji programowych i elementów języka programowania efektywną komunikację z systemem.

API serwerów WWW pojawiły się w reakcji na podstawowe ograniczenia CGI – efektywność i wydajność – są one uruchamiane w tej samej przestrzeni adresowej i w tym samym procesie, co serwer WWW. Cechą charakterystyczną jest to, że serwer

WWW nie zwalnia bibliotek obsługujących rozszerzenie API zaraz po wykonaniu żądania. Robi to dopiero wtedy, gdy uzna, że nie warto dłużej przechowywać załadowanej biblioteki, gdyż nikt jej nie używa lub kiedy kończy pracę serwis realizowany przez serwer WWW. Dzięki temu, że rozszerzenie jest mocno zintegrowane z serwerem WWW, wszelkie zasoby wykorzystywane podczas pracy serwera mogą pozostawać zainicjowane i być użyte przez kolejne żądania, z pominięciem procesu pozyskiwania ich od systemu. Poprawia to znacznie wydajność rozszerzeń wykonywanych tymi technikami.

Rozszerzenia API są zgodne z trójwarstwowym modelem architektury klient-serwer, co zapewnia przetwarzanie z dala i niezależnie od stacji klienckiej. Chociaż aplikacja API jest uruchamiana na serwerze, nie ogranicza to jednak w żaden sposób jej działania. Może ona wywoływać inne aplikacje w dowolnym miejscu sieci, np. na serwerze aplikacyjnym.

Przykładem jest rozszerzenie programistyczne ISAPI (*Internet Server Application Programming Interface*), zaproponowane przez firmę Microsoft do swojego serwera WWW *Internet Information Server* (IIS). Zostało ono stworzone przy współpracy firm Process Software i Microsoft. Innym rozwiązaniem zawierającym API jest produkt firmy Netscape, a mianowicie NSAPI (*Netscape Server Application Programming Interface*), działający i rozszerzający możliwości serwera WWW *Netscape Enterprise Server*.

Istnieje jeszcze wiele mniej znanych rozwiązań zawierających API. Należą do nich między innymi Apache API oraz *Spyglass Server Application Development Interface*.

Zalety i wady

Standardy rozszerzenia funkcjonalności serwera WWW nadają się doskonale do tworzenia aplikacji udostępniających zasoby bazy danych przez strony WWW. Aplikacje bazodanowe tworzy się podobnie jak w CGI, czyli z reguły dane od użytkownika pobiera się za pomocą formularza HTML. Aplikacja API wywoływana jest w momencie zatwierdzenia i wysłania wypełnionego formularza przez użytkownika. Następnie wykonywana jest operacja na bazie danych i zwracana strona WWW, zawierająca rezultat wykonania transakcji.

Aplikacje API charakteryzują się wysoką efektywnością. Są kompilowane, co sprawia, że działają one bardzo szybko i idealnie nadają się do wykorzystania na witrynach WWW, wymagających dużej wydajności. Są stosunkowo proste do wykonania, a przy tym umożliwiają tworzenie niemal dowolnych rozszerzeń funkcjonalności serwera WWW. Często przy ich pomocy realizuje się obsługę innych języków osadzonych na stronach – jak to ma miejsce choćby w przypadku PHP. Należy jednakże zwrócić też uwagę na pewne wady. Niestarannie stworzone rozszerzenie może obniżyć bezpieczeństwo lub znacznie pogorszyć funkcjonowanie całego serwera WWW.

3.3. PHP – PERSONAL HYPERTEXT PREPROCESSOR

Personal Home Pages (PHP) lub inaczej Hypertext Preprocessor to nowoczesna technologia dynamicznego tworzenia stron WWW. Dobrze sprawdza się w zastosowaniu do stron internetowych ze średnią liczbą odwiedzin, gdzie istotny jest układ graficzny i prędkość działania. Został tak pomyślany, aby można było dzięki niemu wykorzystywać zasoby najróżniejszych baz danych, pozwala na używanie wszelkich standardów HTML, a także standardu XML.

PHP jest językiem skryptowym, wzorowanym na tak popularnych językach jak C, Perl i Java i funkcjonuje jako projekt typu *open source*. Jest narzędziem stworzonym przede wszystkim z przeznaczeniem na platformy linuxowe. Jednak twórcy tej nowoczesnej technologii nie zapomnieli o użytkownikach Windows i z myślą o nich zaprojektowali dystrybucję PHP w wersji do 32-bitowych systemów Microsoftu. Być może ten czynnik zdecydował o dużej popularności a być może to, iż jest on dostępny całkowicie bezpłatnie, w pełnej wersji, także przy komercyjnych zastosowaniach. Faktem jest, że PHP zyskuje coraz większe zainteresowanie wśród internetowych projektantów.

Mechanizm działania

Mechanizm działania PHP jest stosunkowo prosty. Przeglądarka internetowa wysyła do serwera HTTP żądanie. Serwer sprawdza, czy w żądanym dokumencie są jakiegokolwiek skrypty PHP. Obsługa poprawnego skryptu PHP przez serwer HTTP może być zrealizowana w dwóch trybach:

- w trybie CGI – zasada działania skryptu jest identyczna jak zewnętrznego programu uruchamianego przez interfejs CGI; w tym przypadku serwer uruchamia zewnętrzny program interpretera poleceń języka PHP przy obsłudze każdego żądania dostępu; po wykonaniu skryptu proces jest zamykany i wszelkie wykorzystywane w nim zasoby, takie jak np. połączenia z serwerem bazy danych, przestają istnieć;
- w drugim możliwym trybie pracy, polecenia języka PHP wykonywane są przez rezydujący w obszarze pamięci serwera WWW moduł interpretera; w momencie, gdy serwer otrzymuje żądanie, jest ono przekazywane do któregoś z procesów potomnych; różne procesy potomne mogą wykorzystywać to samo połączenie z bazą danych.

Po przetworzeniu skryptu serwer generuje stronę WWW opartą wyłącznie na znacznikach HTML. Wygenerowana strona przesyłana jest do przeglądarki.

Udostępnianie bazy danych

PHP doskonale nadaje się do współpracy praktycznie z każdą bazą danych. Wbudowane polecenia umożliwiają realizowanie podstawowych operacji, takich jak:

- nawiązanie komunikacji z bazą danych;

- wysłanie zapytania do bazy danych;
 - odczytanie otrzymanych informacji;
 - zamknięcie połączenia z bazą danych;
- oraz inne bardzo pomocne, typu: zarządzanie sesją, uwierzytelnianie użytkowników, zarządzanie prawami dostępu, i wiele innych.

Zalety i wady

Olbrzymią zaletą jest łatwość opanowania i możliwość efektywnego stosowania tej technologii bez posiadania głębokiej wiedzy i doświadczenia. PHP zawiera mechanizmy generujące szczegółowe komunikaty o błędach, które są istotną pomocą w efektywnym rozwiązywaniu problemów. Innymi zaletami jest podobieństwo do składni języka C, łatwość dodawania kodu PHP do zwykłego HTML-a, rozszerzalność, możliwość współpracy z bazami danych, itp.

Dla wielu użytkowników ważne jest to, że języka PHP nie dotyczy pojęcie wymagań licencyjnych ani kosztów zakupu narzędzi – całkowite koszty rozwiązania ograniczają się do funkcjonalności samej aplikacji. Jednocześnie jest to produkt, który charakteryzuje się dynamicznym rozwojem i ciągłymi pracami nad udoskonalaniem jego możliwości.

PHP dobrze sprawuje się w sytuacjach wymagających szybkiego stworzenia aplikacji bądź wprowadzania częstych modyfikacji, co jest obecnie wymuszane przez warunki panujące w branży internetowej. Dzięki temu, że kod skryptu jest osadzony w dokumencie HTML, realizacja projektu jest szybka i efektywna. Niestety przy bardziej złożonych projektach ta cecha staje się wadą. Gdy nad serwisem pracuje więcej osób, potrzebny jest podział kompetencji: część z nich zajmuje się tworzeniem wyglądu (ang. *design*) a część programuje elementy funkcjonalne, czyli biznesowe. Rozwiązaniem rozszerzającym zastosowanie tego języka jest zastosowanie szablonów – ale w tym zakresie wciąż nie ma jednoznacznego produktu.

W zastosowaniach biznesowych istotnym problemem jest fakt, że klient zawsze otrzymuje kod źródłowy dostarczanej aplikacji. Ochrona produktu, jako wartości intelektualnej i pewnego kapitału jest więc utrudniona.

3.4. ASP – MICROSOFT ACTIVE SERVER PAGES

Microsoft Active Server Pages to bardzo popularna, nowoczesna i wydajna technologia Microsoftu. Jest ona używana głównie przez serwer *Internet Information Server*, ale zdarzają się również inne rozwiązania, typu *Personal Web Serwer* czy *Apache*.

ASP znajduje zastosowanie zarówno przy tworzeniu prostych, dynamicznych stron WWW, jak i skomplikowanych serwisów, opartych o bazy danych i obsługujących transakcje. Z bardziej zaawansowanych możliwości należy wspomnieć o wsparciu dla technologii ActiveX i COM (*Component Object Model*). W złożonych aplikacjach

szablony prezentacji oddzielone są od skomplikowanej logiki biznesowej. Oparciem jest tutaj architektura systemu Windows: logikę biznesową programuje się przy pomocy obiektów COM i COM+. Innymi słowy, przy rozrastaniu się projektu w ASP pozostawia się już tylko sam wygląd strony i odwołania do obiektów systemowych, które zajmują się całą resztą. Dobrą stroną jest to, że projekt można robić przyrostowo, poczynając od małego w ASP, a skończywszy na dużym w COM'ach.

Środowisko programowania aplikacji ASP składa się z pewnej liczby obiektów, za których pośrednictwem można programowo zarządzać aplikacją WWW. Egzemplarze (instancje) obiektów można wykorzystywać w kodzie ASP w dowolnym miejscu. Obiekty mają swoje właściwości, metody i zdarzenia. Obiekty mogą być tworzone przez programistę bądź też możliwe jest wykorzystanie wbudowanych, np.:

- Application – przechowuje informacje o stanie aplikacji; jest używany do współdzielenia danych pomiędzy kilkoma klientami korzystającymi w tym samym czasie z określonej aplikacji;
- ASPError – zawiera informacje o błędach, które wystąpiły podczas przetwarzania skryptu;
- ObjectContext – służy do potwierdzenia lub przerwania wykonywanej transakcji;
- Request – zawiera wszystkie informacje, które są wysyłane z przeglądarki do serwera; posiada on cztery kolekcje, przechowujące określone dane:
 - Cookies – pobieranie *cookies*;
 - Form – pobieranie danych z formularzy, wysyłających je z wykorzystaniem POST;
 - QueryString – pobieranie danych z formularzy, wysyłających je z wykorzystaniem metody GET;
 - ServerVariable – pobieranie informacji o wartości zmiennych serwera;
- Response – służy do przesyłania danych do przeglądarki użytkownika;
- Server – udostępnia podstawowe narzędzia, w tym metodę CreateObject, służącą do tworzenia komponentów;
- Session – służy do przechowywania informacji o użytkowniku podczas jego pobytu na witrynie WWW; w momencie pojawienia się żądania strony ASP, sprawdza on wartość identyfikatora sesji; jeżeli tego identyfikatora nie ma, serwer uważa żądanie strony za początek wizyty i tworzy nowy, unikatowy identyfikator;

Mechanizm działania

ASP, czyli programowalne strony WWW, łączą w sobie HTML ze specjalnymi skryptami interpretowanymi przez serwer. W czasie gdy przeglądarka otwiera plik .asp, serwer HTTP przekazuje to żądanie (uwzględniając ewentualne parametry) do biblioteki dynamicznej asp.dll w celu przetworzenia. Mechanizm ASP sprawdza, czy w żądanym dokumencie znajduje się kod języka skryptowego.

Po przetworzeniu skryptu do przeglądarki wysyłana jest już jedynie informacja w formie czystej strony HTML, nie wymagająca żadnych dodatków czy plug-inów.

Dzięki temu cały ciężar przetwarzania zlokalizowany jest po stronie serwera. Odciąża to komputer klienta i uniezależnia aplikację od środowiska przeglądarki.

Udostępnianie bazy danych

Do największych zalet ASP należy możliwość współpracy z bazami danych. Wewnątrz skryptów można odczytywać i zapisywać dane do bazy, tworzyć zapytania w SQL czy też programować bardziej skomplikowane operacje. Komfortowym i wydajnym mechanizmem, służącym do współpracy z bazami danych jest dostarczany przez Microsoft obiekt ActiveX Data Object (ADO). Zapewnia on szybki i prosty dostęp do bazy danych przez ODBC przy niewielkich zapotrzebowaniach pamięci. Jest to najważniejszy komponent serwerowy w większości bazodanowych aplikacji WWW. ADO dostarcza obiekty, które umożliwiają podłączenie do bazy danych oraz operowanie na danych.

Zalety i wady

Aplikacje przygotowane w technologii ASP są proste w tworzeniu oraz aktualizacji kodu oraz dają możliwość szybkiego dokonywania zmian w wyglądzie stron WWW, przez co zdobywają coraz więcej zwolenników.

Co do zalet istotnych dla twórców aplikacji bazodanowych warto wspomnieć, iż ASP pozwala na uruchamianie skryptów i komponentów wewnątrz transakcji. Umożliwia to np. jednoczesne manipulowanie wieloma rozproszonymi bazami danych, bez tworzenia jakiegokolwiek zagrożenia dla zgromadzonych w nich danych. Oczywiście w razie niepowodzenia w realizacji choćby jednej z częściowych operacji, składających się na daną transakcję, mechanizm obsługi transakcji sam zatroszczy się o odwołanie wszystkich pozostałych. Nie bez znaczenia jest również brak zależności od języka programowania – ASP potrafi korzystać z wielu języków skryptowych: Visual Basic, Java, Perl bądź REXX.

W przypadku ASP można liczyć na solidne wsparcie ze strony producenta – firmy Microsoft. Objawia się to przede wszystkim rosnącą ilością komponentów, rozszerzających możliwości ASP, pojawianiem się narzędzi wspomagających programowanie w tym języku, itp. Między innymi ASP posiada wbudowany debugger (serwer IIS 4.0 posiada zintegrowany "odpluskwiacz" do stron ASP).

Co do wad tej technologii, to główną jest oczywiście uzależnienie od platformy Windows. Oznacza to bowiem konieczność poniesienia stosunkowo dużych nakładów finansowych na samym wstępie – koszt Internet Information Server. Aby uruchomić skrypty ASP w Linuksie lub Novell Netware należy zakupić dodatkowe oprogramowanie emulujące. Istnieją wprawdzie bezpłatne moduły do serwera Apache, które udostępniają pewną funkcjonalność środowiska ASP, ale trudno mówić o pełnej zgodności kodu źródłowego. Pod tym względem ASP zdecydowanie ustępuje swojemu głównemu wieloplatformowemu konkurentowi – PHP.

Jeśli chodzi o zastosowania komercyjne, to niestety, jak przy wszystkich językach skryptowych, nie istnieje opcja chociażby częściowej kompilacji skryptu w celu przyspieszenia jego działania lub ochrony przed kradzieżą kodu. W tym celu trzeba zakupić stosunkowo drogie oprogramowanie, np. firmy Server Object, pozwalające na zaszyfrowanie gotowej aplikacji i jej odszyfrowywanie "w locie" w momencie pobierania skryptu przez serwer WWW.

3.5. JSP – JAVA SERVER PAGES

JSP (Java Server Pages) jest technologią platformy Java, której twórcą jest Sun Microsystems. Tworzy ona szybki, nowoczesny i wieloplatformowy mechanizm generowania dokumentów WWW, umożliwiający tworzenie przenaszalnych aplikacji internetowych. Pozwala na tworzenie zarówno prostych katalogów produktów jak i aplikacji wyposażonych w zaawansowane funkcje handlu elektronicznego i obsługę dowolnych systemów baz danych. Można powiedzieć, że stanowi kompletne rozwiązanie dla rozbudowanych projektów, a przez zespoły pracujące nad dużymi przedsięwzięciami jest wykorzystywana jako szablon prezentacji dla jeszcze silniejszych technologii: servletów oraz wykorzystujących je serwerów aplikacji.

Strony JSP są to dokumenty tekstowe, w których pomiędzy kodem HTML umieszczane są polecenia w języku Java. Na podobnej zasadzie opiera się większość rozwiązań skryptowych, jednak kod zawarty w stronach JSP nie jest interpretowany przy każdym zapytaniu. Strony JSP są kompilowane przy pierwszym załadowaniu (bądź po zmianie ich treści) oraz przetwarzane automatycznie do postaci servletów. Przy każdym następnym odwołaniu strona JSP działa jak zwykły servlet, dzięki czemu nie obserwuje się znaczących opóźnień związanych z przetwarzaniem.

Poza zwykłym kodem HTML, na stronach JSP mogą być umieszczane trzy główne rodzaje elementów, charakterystycznych dla JSP:

- elementy skryptowe (*ang. Scripting elements*) – umożliwiają wstawienie kodu w języku Java, który ma być częścią generowanego servletu; służą do przetwarzania i tworzenia zawartości dokumentów WWW oraz manipulowania obiektami;
- akcje (*ang. Actions*) – określają komponenty, które mają być użyte na stronie oraz mają wpływ na działanie kontenera JSP;
- dyrektywy (*ang. Directives*) – zawierają ogólne informacje o stronie, które są następnie dostarczane do kontenera JSP.

Strony JSP obsługując żądanie mogą tworzyć i/lub mieć dostęp do obiektów. Możliwe są następujące rodzaje obiektów:

- Explicit objects – obiekty tworzone jawnie przez programistę poprzez akcje bądź użycie kodu skryptowego; mogą to być:
 - obiekty tworzone w kodzie Java, zagnieżdżonym w dokumencie;
 - komponenty JavaBeans wykorzystywane w dokumencie; komponentem JavaBeans, wykorzystywanym przez JSP może być

dowolna skompilowana klasa w języku Java, posiadająca zbiór właściwości i metody do ich odczytu i zmiany;

- Implicit objects – tzw. obiekty ukryte, tworzone przez wewnętrzne mechanizmy JSP, dostępne standardowo w każdym dokumencie JSP; każdy z tych obiektów ma klasę lub interfejs zdefiniowany w Javie lub w pakiecie API JavaServlet; np.:
 - request – reprezentuje żądanie klienta, umożliwia dostęp do parametrów żądania, jego typu (GET, POST), *cookies*, itp.;
 - response – jest skojarzone z odpowiedzią wysyłaną do klienta;
 - out – może być używane do wysyłania odpowiedzi do klienta (rzadko stosowane);
 - session – powiązany z zapytaniem, służy do kontroli sesji;
 - application – obiekt otrzymywany poprzez `getServletConfig().getContext();`
 - config – jest to obiekt `ServletConfig` dla danej strony;
 - pageContext – przechowuje specyficzne cechy serwera;
 - page – nie jest używany w Javie; został on stworzony z myślą o używaniu na stronach JSP innych języków skryptowych niż Java.

JSP pozwala na zdefiniowanie kontekstu, w jakim używa się danego komponentu. Kontekst określa, kiedy dany obiekt ma być tworzony, jaki jest jego zasięg i kiedy można go wyrzucić z pamięci. Są do dyspozycji cztery konteksty: *page*, *request*, *session*, *application*. Korzystanie z obiektów o zasięgu większym niż bieżący dokument pozwala m.in. na oszczędniejsze gospodarowanie zasobami oraz współdzielenie obiektów przez dokumenty obsługujące to samo żądanie.

Mechanizm działania

Strony JSP są przetwarzane w dwóch fazach: *HTTP translation time* i *request processing time*. Faza *HTTP translation time* ma miejsce przy pierwszym załadowaniu strony przez klienta. Następuje wtedy skompilowanie strony JSP do klasy Javy oraz automatyczne przetworzenie do postaci servletu. Przy każdym następnym żądaniu wykonania strony JSP (pod warunkiem, że oryginalny dokument JSP nie uległ zmianie) jest ona przetwarzana w tzw. *fazie request processing time*. Strony JSP są wykonywane w kontenerze JSP (*ang. JSP engine*), zainstalowanym na serwerze WWW lub serwerze aplikacji. Zapytania kierowane przez klienta przechowywane są w obiekcie request, który wysyłany jest do kontenera. Następnie kontener JSP wysyła obiekt request do komponentu po stronie serwera, zdefiniowanego wcześniej w pliku JSP. Komponentem tym może być servlet, JavaBean lub Enterprise JavaBean. Komponent obsługuje żądanie, komunikując się w razie potrzeby z bazą danych, i wysyła z powrotem obiekt response. Obiekt ten jest przetwarzany zgodnie z reprezentacją HTML a następnie całość, w postaci gotowej strony HTML, jest odsyłana do klienta.

Udostępnianie bazy danych

Dostęp do bazy danych z aplikacji JSP jest realizowany za pomocą interfejsu JDBC, z wykorzystaniem sterowników JDBC dostarczanych przez producentów systemów zarządzania bazami danych. Operacje dostępu do bazy danych mogą być realizowane bezpośrednio poprzez interfejs JDBC (dynamiczny SQL) lub w oparciu o standard SQLJ (statyczny SQL).

Zalety i wady

Największą zaletą JSP jest niezależność od platformy sprzętowej i systemowej, jak i możliwość obsługi dowolnego systemu baz danych. Poza tym, korzystne jest to, że JSP opiera się na języku Java, który jest narzędziem mocniejszym od języków skryptowych a jednocześnie łatwym w użyciu.

Dzięki odseparowaniu logiki biznesowej od warstwy prezentacji jest bardzo wygodną technologią do zastosowania w większych projektach, nad którymi pracuje kilka osób i wskazane jest, aby wszyscy pracowali równolegle nad poszczególnymi fragmentami aplikacji. Pozwala na łatwy podział pracy między projektantów (grafików) i programistów języka Java.

Niestety do poprawnego działania JSP wymagane jest odpowiednie oprogramowanie w postaci: systemu operacyjnego, dysponującego wirtualną maszyną Javy, serwera obsługującego JSP (np. Apache Tomcat), interpretera języka Java oraz tzw. *Java Runtime Environment*, który to mechanizm zapewnia działanie interpretera Javy. Pewne znaczenie ma też sprzęt komputerowy, jakiego się używa. Pocieszeniem jest fakt, że całość wymaganego oprogramowania można ściągnąć nieodpłatnie z sieci (dotyczy to tylko niektórych producentów).

3.6. JAVA SERVLETS – PROGRAMY JAVY NA SERWERZE WWW

Servlety, czyli komponenty napisane w języku Java i uruchamiane po stronie serwera są jedną z ciekawszych propozycji, jeśli chodzi o wybór środowiska do tworzenia aplikacji internetowych. Mogą one pracować według dowolnego protokołu transmisji i na dowolnej platformie systemowej i sprzętowej. Są one znakomitą metodą tworzenia internetowych interfejsów użytkownika do istniejących systemów zarządzania przedsiębiorstwem bądź systemów zarządzania bazami danych. Znajdują zastosowanie zwłaszcza w aplikacjach typu e-commerce lub rozbudowanych portalach, wymagających złożonej logiki biznesowej do zarządzania ofertą serwisu, dostosowania potrzeb do poszczególnych użytkowników, obsługi zamówień klientów czy integracji z zewnętrznymi aplikacjami. Środowisko, w którym działają, czyli tzw. kontener, przechowuje pulę servletów gotowych do działania w trybie równoległym,

co zwiększa wydajność i umożliwia synchronizowanie kilku wywołań jednocześnie (korzystanie ze wspólnych danych lub równoległe prowadzenie procesów).

API servletów zawiera komplet niezbędnych obiektów i metod, które udostępniają informacje o bieżącym połączeniu HTTP. API obejmuje też obsługę sesji za pomocą „cookies” wraz z mechanizmem przypisywania danych do konkretnego użytkownika aplikacji.

Mechanizm działania

Servlet może być wywołany w przeglądarce na kilka sposobów:

- poprzez podanie URL servletu i wywołanie go z przekazaniem parametrów metodą GET;
- poprzez stworzenie formularza na stronie HTML i wywołanie servletu z przekazaniem parametrów metodą POST;
- z wykorzystaniem znacznika <SERVLET> na stronie HTML.

Servlet odbiera strumień wejściowych danych, przetwarza je i wysyła wynik z powrotem do klienta, który dokonał wywołania.

Udostępnianie bazy danych

Przetwarzanie baz danych w Javie odbywa się za pośrednictwem JDBC, które stanowi unowocześnioną i niezależną od platformy odmianę interfejsu ODBC (*Open Database Connectivity*), używanego do przetwarzania relacyjnych baz danych.

Zalety i wady

Bardzo użyteczne jest to, że servlet może przechowywać informację stanu tzn. servlet nie musi przy każdym wywołaniu inicjalizować się i jest w stanie utrzymać trwałe połączenie z zasobami danych. Wygląda to tak, że po wywołaniu servlet pozostaje w pamięci i reaguje na kolejne wywołania aż do czasu, gdy administrator przerwie jego działanie, bądź wyłączy serwer WWW.

Servlety zapewniają dużo większe bezpieczeństwo i to zarówno w znaczeniu bezpieczeństwa języka komputerowego (*ang. safety*), jak i kontroli nad dostępnymi operacjami systemowymi (*ang. security*). Menedżer bezpieczeństwa servletów automatycznie zapewnia dostęp do tzw. sandbox – jest to bezpieczne miejsce dla działania servletów. W Java Web Serverze lokalne servlety, zapisane i zainstalowane w katalogu servletów uważa się za „godne zaufania” i udostępnia się im wiele niebezpiecznych funkcji.

Większość zalet servletów wynika z samych właściwości języka Java:

- Java jest językiem wygodnym i narzucającym dobre kanony konstrukcji kodu;
- przenośność języka Java zapewnia niezależność od platformy sprzętowej i systemowej a do tego zunifikowany interfejs JDBC zapewnia dostęp do większości systemów baz danych;

- uniwersalność języka Java umożliwia dostęp do różnorodnych rozwiązań technologicznych (np. INDI, XML);
- bogactwo narzędzi i pakietów Javy pozwala na integrację z najróżniejszymi produktami; między innymi możliwa jest integracja z systemami programowania rozproszonego i monitorami transakcji, np. Enterprise Java Beans (EJB);
- obiektowy język Java doskonale nadaje się do modelowania przy pomocy narzędzi CASE co zapewnia panowanie dużymi projektami;
- przy wykorzystaniu szablonów dokumentów bądź też JSP można doskonale oddzielić warstwę prezentacji (kod HTML) od warstwy logiki (kod Javy);
- wysoki poziom programowania w Javie pozwala na uzyskanie oprogramowania niezawodnego i łatwego do dalszej modyfikacji;
- kod jest używany w postaci skompilowanej, łatwiej zatem zadbać o ochronę praw autorskich.

Przy wszystkich zaletach, jakie niesie z sobą język Java, są też pewne wady. Język Java wymaga wysokiej jakości kodowania, co wiąże się z dużą pracochłonnością. Na rynku aplikacji internetowych szybkość powstawania prototypów i wersji produkcyjnych jest zaś elementem niezmiernie istotnym. Niestety wprowadzanie poprawek w servletach wiąże się z koniecznością rekompilacji kodu, a niekiedy nawet przeładowania całego serwera.

Do tego, mimo stosunkowo krótkiego okresu, jaki upłynął od zdefiniowania Javy, pojawiły się już różne jej implementacje. Możliwa jest sytuacja, że programy napisane nawet w czystej Javie (ang. „*pure Java*”) będą się wykonywały na pewnych implementacjach maszyny wirtualnej Javy, na innych nie będą się wykonywały bądź będą wykonywały się niepoprawnie. To przeczy oczywiście podstawowym założeniom, czyli przenaszalności aplikacji zakodowanych z użyciem Javy.

Jednocześnie servlety nie najlepiej nadają się do pisania prostych aplikacji. Głównie wynika to z tego, że kod HTML jest generowany w aplikacji, co powoduje duży współczynnik kodu aplikacyjnego nad HTML'owym. Gdy ilość kodu HTML przeważa nad kodem Javy, robienie poprawek w treści produkowanej przez servlet jest uciążliwe. I w końcu, są skomplikowane w instalacji: wymagają pakietu JDK, serwera servletów np. Tomcat, sprzężonego z nim serwera WWW, np. Apache'a, obsługi baz przez JDBC. Należy jednak przyznać, że większość serwerów jest fabrycznie przystosowana do obsługi servletów w Javie i nie wymaga dodatkowych zabiegów ze strony administratora.

3.7. SERWERY APLIKACJI

Implementacje serwerów aplikacyjnych są to platformy do projektowania i udostępniania złożonych systemów. Stanowią one propozycję przewidzianą dla bardzo złożonych projektów *Business To Customers* (B2C) oraz *Business To Business* (B2B). Ich zadaniem jest rozwiązywanie problemów związanych z bezpieczeństwem,

skalowalnością, współpracą z web-serwerami, bazami danych i innymi aplikacjami oraz umożliwienie korzystania z zasobów sieci.

Obecnie istnieje około trzydziestu różnych webowych serwerów aplikacyjnych (FEIL 00), z których każdy prezentuje odmienne rozwiązania do obsługi standardów oraz techniki komponentowej. Dostawcy stosują różne podejścia do podstawowych problemów rozproszonego przetwarzania, każdy z produktów charakteryzuje się inną szybkością i łatwością wdrażania oraz zorientowaniem na poszczególne obszary e-biznesu.

4. UWAGI KOŃCOWE

Serwisom internetowym stawia się coraz większe wymagania, m.in. związane z dynamicznym udostępnianiem baz danych. W związku z tym pojawiają się nowe technologie i standardy. Postęp ten stawia projektantów serwisów internetowych przed problemem wyboru najodpowiedniejszej technologii. Jednocześnie zarządcy systemów muszą czuwać nad zapewnieniem właściwego poziomu wydajności i ochrony zasobów udostępnianych w sieci. Na podstawie dokonanego w pracy przeglądu technologii prezentacji i generacji zawartości witryn internetowych, stwierdzono brak możliwości jednoznacznego wskazania najlepszej metody udostępniania baz danych poprzez sieć Internet, jako że przy ocenie i wyborze należy kierować się dokładną specyfikacją wymagań, dotyczących złożoności i skalowalności tworzonej aplikacji, w kontekście ograniczeń m.in. o charakterze sprzętowo-finansowym.

LITERATURA

- [ASP2000] *Working with Active Server Pages* www.linux.itt.pl/asp
- [COLB 98] Colburn R., *CGI*, Helion, 1998.
- [FEIL 00] Feiler J., *Application Servers, Powering the Web-Based Enterprise*, Ac. Pr. 2000.
- [GIRD 00] Girdley M. *Developing Database Applications and Applets with the JDBC Chapter 15*, <http://docs.rinet.ru:8083/JWP>
- [GROCH 00] Grochala M., *Java, aplikacje bazodanowe*, Helion, 2000.
- [HARV 00] Harvey P., ODBC www2.linuxjournal.com/lj-issues/issue67/3395.html
- [HILT 00] Hilton C., *PHP3 Internetowe aplikacje bazodanowe*, Helion, 2000.
- [Java2001] The source for Java <http://java.sun.com>
- [JSP 01] Java Server Pages Specification
java.sun.com/aboutJava/communityprocess/first/jsr053/jsp12.pdf
- [KLIM 98] Klimowicz D., Porębski W., *Język Java: Podstawy, Programowanie, Zastosowania*, Gdańsk, 1998.
- [PHP 01] Portal technologii PHP www.php.net i www.php.pl
- [RATS 2001] Ratschiller T., Gerken T., *PHP4 Aplikacje*, Robomatic, Wrocław, 2001.

- [ROMO 98] Romowicz W., *HTML i JavaScript*, Helion, 1998.
- [SerJSP 00] Servlets & JSP Tutorial: www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
- [SOBCZ 00] Sobczyk Ł., *Budowanie Aplikacji w języku Java dla Internetowej Platformy Oracle (...)*, Software 2.0, 2/2000.
- [SOPA 00] Sopala A., *Pisanie programów Internetowych*, Mikom, Warszawa, 2000.
- [TII2001] Technologie, Internet, Informacje: www.javasoft.pl

METHODS WHICH ARE USED TO MAKE DATABASES AVAILABLE VIA THE INTERNET

This document presents the survey of methods, which are used to access databases and make them available via the Internet, the complex and advanced technologies, which set the trends of the current systems development.

*hurtownia danych,
maszynowe uczenie się,
systemy informatyczne,
systemy wspomagania decyzji*

Jerzy SURMA*

METODY ANALITYCZNE W INFORMATYCZNYCH SYSTEMACH ZARZĄDZANIA

W pracy przedstawiono przegląd zastosowań metod komputerowej analizy danych w kontekście rozwoju systemów informatycznych zarządzania i hurtowni danych. Omówiono podstawowe metody analityczne i ich praktyczne wykorzystanie w komputerowym wspomaganiu decyzji.

1. WPROWADZENIE

Systemy klasy ERP (ang. *Enterprise Resource Planning*) w codziennych transakcjach rejestrują megabajty danych o funkcjonowaniu przedsiębiorstwa. Ocenia się, że obecnie średnio mniej niż 10% zasobów informacyjnych tego typu jest wykorzystywanych do wsparcia podejmowania decyzji strategicznych. Dzieje się to w sytuacji, kiedy jest już dzisiaj oczywiste, że odpowiednio agregowane i analizowane dane pochodzące z systemów ERP mogą być bezcennym fundamentem trafnych decyzji umożliwiających uzyskanie korzystnej pozycji rynkowej. Naturalnie komputerowa analiza danych nie może zakłócić codziennej pracy systemów transakcyjnych, powinna bazować na zintegrowanych i skonsolidowanych danych pochodzących z różnych źródeł oraz powinna mieć dostęp do danych historycznych. Te wymagania implikują potrzebę budowania odpowiedniej architektury informatycznej, która znalazła odzwierciedlenie w systemach hurtowni danych i przetwarzania analitycznego on-line OLAP (ang. *on-line analytical processing*) [INMON 01].

* Wydziałowy Zakład Informatyki, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Skwer Idaszewskiego 1, surma@ci.pwr.wroc.pl

2. ZASTOSOWANIA ANALIZY DANYCH

Komputerowa analiza danych, pomimo wielu trudności natury formalnej, dojrzała do poziomu przynoszącego wymierne wdrożenia. Badania w zakresie systemów wspomagania decyzji mają swoje korzenie w statystyce matematycznej i badaniach operacyjnych. Ostatnio głównym obszarem badań w tym zakresie są tzw. indukcyjne algorytmy maszynowego uczenia się (ang. *machine learning*) [MITCHELL 97] i wywodzące się z tej dziedziny techniki odkrywania wiedzy w bazach danych (ang. *knowledge discovery in databases*), zwane potocznie metodami eksploracji danych (ang. *data mining*) [PIATETSKY 96] (patrz rozdział 3). Sukcesy w tej dziedzinie zaowocowały powstaniem szeregu rozwiązań biznesowych pozwalających na efektywne wsparcie pracy menedżera. Najbardziej spektakularne efekty biznesowe uzyskano dotąd w następujących zastosowaniach [SURMA 03]:

- Analityczny CRM (ang. *analytical customer relationship management*), czyli m.in. analiza preferencji i zachowań klientów w celu generowania trafnych decyzji co do sprzedaży dodatkowej, sugerowania odpowiednich promocji czy zakupu nowych produktów. U podstaw tych analiz leży odpowiednie wykonanie segmentacji rynku i obliczenie tzw. wartości klienta w czasie (ang. *customer lifetime value*).
- Ocena stopnia możliwości odejścia klienta do konkurencji (ang. *churn*), czyli analiza przyczyn dotychczasowych odejść klientów i na tej podstawie określenie prawdopodobieństwa odejść u aktualnych klientów. Jest to powszechnie stosowana technika przez operatorów telekomunikacyjnych w celu odpowiedniego ukierunkowania akcji marketingowych, czy „właściwej” obsługi klientów rokujących szybkie odejście.
- Wyszukiwanie nadużyć (ang. *fraud detection*), powszechnie stosowane w bankowości podejście polegające na analizie przypadków oszustw w transakcjach finansowych, mające na celu określenie wzorców zachowań przestępczych. Poprzez odpowiednie monitorowanie bieżących transakcji i wykorzystanie takich wzorców można zlokalizować aktualne próby nadużyć.
- Badania trendów (ang. *trend analysis*), czyli wykorzystanie danych historycznych dla problemu prognozowania. Podejście często stosowane przez sieci handlowe umożliwiające wykrywanie tzw. sezonowości w sprzedaży, co pozwala na efektywne zarządzanie procesem dystrybucji.

3. METODY ANALIZY DANYCH

Za kulisami spektakularnych sukcesów biznesowych stoją techniki analizy i eksploracji danych. Są to algorytmy przetwarzające duże repozytoria danych w celu pozyskania użytecznych informacji [PIATETSKY 96]. Ogólnie przedstawione wcześniej zastosowania trafiają w jedno ze standardowych zadań:

- Klasyfikacja: przyporządkowanie obiektu do predefiniowanej klasy. Na przykład zaklasyfikowanie wniosku kredytowego jako wniosku niskiego, średniego lub wysokiego ryzyka. Zadanie klasyfikacji może być realizowane z wykorzystaniem drzew decyzyjnych (ang. *decision tree*), sztucznych sieci neuronowych (ang. *neural nets*), czy algorytmów wykorzystujących analizę przykładów (ang. *case-based reasoning*).
- Estymacja: zadanie zbliżone do zadania klasyfikacji przy założeniu, że klasa, do której przyporządkowujemy obiekt ma ciągły, a nie dyskretny zbiór wartości. Tak rozumiana estymacja jest zwykle wykorzystywana dla zadania prognozowania. Prognozowanie jest zwykle realizowane z wykorzystaniem tzw. regresyjnej odmiany drzew decyzyjnych (ang. *regression decision tree*), sztucznych sieci neuronowych i metod statystycznych (analiza szeregów czasowych).
- Wykrywanie związków: wyszukiwanie grup obiektów, które występują razem w określonych kontekstach. Klasycznym przykładem tej klasy zadań jest analiza koszyka zakupów (ang. *market basket analysis*), gdzie szuka się odpowiedzi na pytanie jakie produkty sprzedaje się razem, co umożliwia m.in. na inicjowanie sprzedaży dodatkowej. Wykrywanie związków realizuje się poprzez wykorzystanie algorytmów analiz asocjacji (ang. *association rules analysis*).
- Grupowanie: znajdowanie w zbiorze obiektów podzbiorów (grup) obiektów o podobnych własnościach. Dobrym przykładem zastosowania metod grupowania jest segmentacja rynku w badaniach marketingowych, gdzie można odnaleźć segmenty odpowiadające rzeczywistej sytuacji rynkowej, a nie zakładanej czy przypuszczanej. To zadanie jest realizowane algorytmami grupowania (taksonomii) (ang. *cluster analysis*).

4. ASPEKTY PRAKTYCZNE

Dojrzałość techniczna i biznesowa omówionych metod zdeterminowała rozwój systemów hurtowni danych i gotowych pakietów analitycznych przez czołowych producentów systemów zarządzania baz danych takich jak Oracle, IBM czy Microsoft. Klasycznym przykładem tego trendu jest rozwój systemu hurtowni danych przez firmę SAP (SAP BW) oraz tzw. aplikacji analitycznych (ang. *analytical application*), w ramach których dostępne są następujące techniki eksploracji danych (ang. *data mining*): drzewa decyzyjne, analizy asocjacyjne i algorytmy grupowania. Standardowo wszystkie te algorytmy są zintegrowane z SAP BW, mając efektywny dostęp do zasobów informacyjnych i są dostosowane do następujących rozwiązań biznesowych:

- Analiza wartości klienta w czasie dla potrzeb analitycznego CRM (ang. *CLTV analysis*)

- Identyfikacja grupy docelowej dla potrzeb zarządzania kampaniami marketingowymi (ang. *RFM analysis*)
- Segmentacja rynku dla potrzeb badań marketingowych (ang. *ABC analysis*)
- Planowanie i zarządzanie kampaniami marketingowymi
- Analiza i planowanie procesu sprzedaży

Rozwiązania analityczne są w niektórych przypadkach rozwinięte o specyficzne rozwiązania branżowe. Przykładowo takie specjalizowane moduły analityczne opracowano dla potrzeb przemysłu motoryzacyjnego, przemysłu farmaceutycznego, sektora energetycznego i sieci handlowych sprzedaży detalicznej.

5. PODSUMOWANIE

W „Firmie XXI wieku” intuicja biznesowa pozostanie najprawdopodobniej jednym z kluczowym atutów kadry menedżerskiej. Wsparcie intuicji rzetelną i aktualną informacją oraz odpowiednią analizą pozwoli ją wykorzystać tam gdzie występuje jej rzeczywista potrzeba. Sztuczna inteligencja w próbach symulacji tych subtelnych obszarów ludzkich zachowań przechodzi wyraźny kryzys i nic nie wskazuje na to, aby w tym stuleciu jakaś kolejna wersja systemu klasy ERP wyeliminowała w końcu potrzebę istnienia ludzkiego zespołu zarządzającego.

LITERATURA

- | | |
|----------------|--|
| [INMON 01] | Inmon W. H., <i>Corporate Information Factory</i> , John Wiley & Sons, 2001. |
| [PIATETSKY 96] | Piatetsky-Shapiro G., Fayyad U., Smyth P., and Uthurusamy R., <i>Advances in Knowledge Discovery and Data Mining</i> , AAAI/MIT Press, 1996. |
| [MITCHELL 97] | Mitchell T.M., <i>Machine Learning</i> , McGraw-Hill, 1997. |
| [SURMA 03] | Surma J., <i>Systemy ERP, niewykorzystane źródło wiedzy</i> , Strategie Biznesu, SAP, 2003. |

ANALYTICAL METHODS IN MANAGEMENT INFORMATION SYSTEMS

The work presents the impact of data warehouse and data mining methods on enterprise resource planning systems.





BIBLIOTEKA GŁÓWNA

312456L/1

Wydawnictwa Politechniki Wrocławskiej
są do nabycia w następujących księgarniach:
„Politechnika”

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
budynek A-1 PWr., tel. (0-71) 320-25-34;
„Tech”

plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. (0-71) 320-32-52

Prowadzimy sprzedaż wysyłkową

ISBN 83-7085-706-X