

**Prace Naukowe
Wydziałowego Zakładu Informatyki
Politechniki Wrocławskiej**

5

BAZY DANYCH

Redakcja naukowa

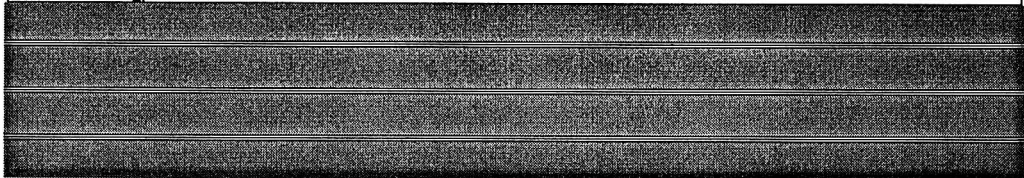
Zygmunt Mazur

**Oficyna Wydawnicza
Politechniki Wrocławskiej**



Prace Naukowe
Wydziałowego Zakładu Informatyki
Politechniki Wrocławskiej

5 BAZY DANYCH



Redakcja naukowa

Zygmunt Mazur



Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2004



WYDZIAŁOWY ZAKŁAD INFORMATYKI

**WYDZIAŁ INFORMATYKI I ZARZĄDZANIA
POLITECHNIKI WROCŁAWSKIEJ**

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

Recenzent

Witold REKUĆ

Opracowanie redakcyjne materiałów

Hanna MAZUR

Projekt okładki

Zygmunt MAZUR

© Copyright by Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2004

OFICyna WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISBN 83-7085-764-7

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 82/2004.

SPIS TREŚCI

<i>Wstęp</i>	5
P. Mazur, <i>Natywne XML-owe bazy danych</i>	7
Z. Mazur, P. Michalec, <i>Temporalno-probabilistyczno-obiektowe bazy danych</i>	21
A. Klosow, D. Muraszko, <i>Alokacja obiektów bazy danych w pamięci stałej z wykorzystaniem mechanizmów grupowania</i>	31
J. Gruber, Z. Mazur, M. Standzoń, <i>System projektowy Sybase PowerDesigner DataArchitect Suite 6.1 oraz środowisko Sybase PowerBuilder 7 Enterprise do modelowania i generowania baz danych</i>	55
D. Podulka, Z. Szpunar, <i>Projektowanie relacyjnych baz danych za pomocą CASE</i>	71

CONTENTS

<i>Introduction</i>	5
P. Mazur, <i>Native XML Databases</i>	7
Z. Mazur, P. Michalec, <i>Temporal Probabilistic Object Databases</i>	21
A. Klosow, D. Muraszko, <i>The Allocation of Database Objects in External Memory with Usage of Clustering Mechanism</i>	31
J. Gruber, Z. Mazur, M. Standzoń, <i>Design Technologies Sybase Powerdesigner DataArchitect Suite 6.1 and Software Tools Sybase PowerBuilder 7 Enterprise for Modeling and Building Database</i>	55
D. Podulka, Z. Szpunar, <i>Relational Database Design Using Case Tools</i>	71

*Problem dobrze ujęty,
to problem w połowie rozwiązany*

• Charles Franklin Kettering

WSTĘP

Pierwszy numer serii „Bazy danych” – *Prace Naukowe Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej* pod redakcją Zygmunta Mazura został wydany w 2000 r. i od tego czasu, systematycznie, co roku ukazuje się kolejny numer. Tematyka piątego już Zeszytu Naukowego „Bazy Danych” jest poświęcona natywnym XML-owym bazom danych, temporalno-probabilistyczno-obiektowym bazom danych oraz projektowaniu baz danych. Podstawową jednostką przechowywania danych w relacyjnych bazach danych jest krotka, w bazach obiektowych – obiekt, natomiast w natywnych bazach danych XML – dokument XML. Język XML (ang. *eXtensible Markup Language* – rozszerzalny język znaczników) jest nowym standardem przyjętym przez *World Wide Web Consortium* (skrót *W3C*) jako język do przesyłania danych w sieci. XML wykorzystywany do opisu baz danych ma wiele zalet w porównaniu ze specyficznymi formatami takimi jak np. format .dbf (dBase) lub .mdb (Access), przede wszystkim jest czytelny i oparty na powszechnym otwartym standardzie. Język XML został utworzony do opisu zawartości i struktury danych (a nie do ich wyglądu).

Artykuły zamieszczone w bieżącym Zeszycie Naukowym dotyczą:

- ❖ natywnych XML-owych baz danych,
- ❖ temporalno-probabilistyczno-obiektowych baz danych,
- ❖ alokacji obiektów bazy danych w pamięci stałej z wykorzystaniem mechanizmów grupowania,
- ❖ systemu projektowego *Sybase PowerDesigner DataArchitect Suite 6.1* oraz środowiska *Sybase PowerBuilder 7 Enterprise* umożliwiającego modelowanie i generowanie baz danych,
- ❖ projektowania relacyjnych baz danych za pomocą *CASE*.

Autor pierwszego artykułu w tym Zeszyście, Paweł Mazur, przedstawia zagadnienia natywnych XML-owych baz danych i problem ich przydatności oraz produkty będące natywnymi XML-owymi bazami danych.

Wiele zastosowań baz danych operuje danymi i wyrażeniami w sytuacjach, gdzie istnieje niepewność, kiedy wystąpi dane zdarzenie i czy w ogóle wystąpi. Zastosowania te w naturalny sposób są reprezentowane przez modele obiektowe. Natomiast czas (zależności temporalne) i niepewność (zależności probabilistyczne) parametryzują różne cechy tego modelu. Zygmunt Mazur i Piotr Michalec przedstawiają w pracy nowy model temporalno-probabilistyczno-obiektowej bazy danych, który pozwala powiązać z każdym zdarzeniem zbiór punktów czasu, a z każdym punktem czasu, prawdopodobieństwo wystąpienia danego zdarzenia

Obecnie coraz częściej w systemach informatycznych są wykorzystywane obiektowe bazy danych. Znaczący wpływ na efektywność przetwarzania danych w tego typu systemach odgrywają mechanizmy grupowania obiektów w pamięci stałej. Aby dokonać właściwego wyboru strategii grupowania obiektów należy posiadać wiedzę na temat struktury obiektów w bazie danych oraz ich zachowania. Problem ten klasyfikuje się do problemów NP-zupełnych i w praktyce często wybór strategii grupowania polega na intuicji administratora bazy danych. Aleksander Klosow i Dariusz Muraszko podjęli próbę wykazania pewnych zależności pomiędzy skuteczną strategią grupowania a wybranymi charakterystykami bazy danych. Rozważania teoretyczne poparli wynikami licznych eksperymentów komputerowych.

Systemy bazodanowe integrowane w sieciach intranetowych i w Internecie mają kluczowe znaczenie praktyczne w nowoczesnych systemach z zasobami informacyjnymi. Autorzy kolejnej pracy, Jacek Gruber, Zygmunt Mazur i Marek Standzoń, przedstawiają system projektowy *Sybase PowerDesigner DataArchitect Suite 6.1* oraz środowisko do modelowania i generowania baz danych – *Sybase PowerBuilder 7 Enterprise*.

Coraz większe wymagania stawiane systemom informatycznym oraz konkurencja rynkowa zmusza producentów oprogramowania nie tylko do podnoszenia jakości tworzonych produktów, ale także do wytwarzania ich w coraz krótszym czasie. Aby sprostać rosnącym wymaganiom niezbędne jest, poza odpowiednią organizacją projektu i właściwym zarządzaniem, stosowanie narzędzi wspomagających proces projektowania czyli narzędzi *CASE* (ang. *Computer Aided/Assisted Software/System Engineering*). Autorzy ostatniej pracy, Dariusz Podulka i Zbigniew Szpunar, omówili programy typu *CASE*, wskazali obszary zastosowań narzędzi *CASE* w projektowaniu relacyjnych baz danych, a także zaprezentowali przykładową aplikację *CASE*, wykonaną w 2002 r. na Wydziale Informatyki i Zarządzania PWr.

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

Zygmunt Mazur

Paweł MAZUR*

NATYWNE XML-OWE BAZY DANYCH

Praca prezentuje natywne bazy danych i omawia aspekt ich przydatności. Okazuje się, że nie zawsze warto skorzystać z natywnej bazy danych pomimo, że aplikacja operuje na danych w formacie XML. Bardzo często lepszym rozwiązaniem jest wykonanie transformacji danych do innego modelu, na przykład relacyjnego. Omówiono także wybrane produkty będące natywnymi bazami danych XML.

1. WPROWADZENIE

Język XML¹ (ang. *eXtensible Markup Language*) należy do grupy języków znacznikowych SGML (ang. *Standard Generalized Markup Language*), co można przetłumaczyć na język polski jako „Standardowy Uogólniony Język Znacznikowy”. Języki znacznikowe służą do opisu przechowywanych danych oraz opisu ich przetwarzania. Tak więc format RTF jest także zapisem znacznikowym.

W XML nazwy znaczników umieszczane są pomiędzy znakami „<” oraz „>”. Każdy znacznik otwierający musi posiadać odpowiadający mu znacznik zamykający (wówczas nazwa musi być poprzedzona znakiem „/”). Dodatkowo, element może posiadać atrybuty, które definiowane są w znaczniku otwierającym oraz elementy-dzieci, definiowane jako elementy znajdujące się w ciele innego elementu. Przedstawia to poniższy przykład:

```
<spis_tresci czego="ksiazka">
  <pozycja nr="1">Tytuł pierwszego rozdziału.</pozycja>
  <pozycja nr="2">Tytuł drugiego rozdziału.</pozycja>
</spis_tresci>
```

W ostatnich latach XML stał się bardzo popularny i chętnie wykorzystywanym formatem przechowywania i przekazywania danych. Wynika to z cech języka XML:

- prostota pod względem złożoności
- niezależność od platformy sprzętowej
- brak potrzeby wykupienia licencji

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

¹ Obecnie najnowsza wersja specyfikacji o numerze 1.0 Second Edition dostępna jest pod adresem <http://www.w3.org/TR/REC-xml>

- ogólnodostępna specyfikacja
- dokumenty XML są plikami tekstowymi
- separacja treści od jej prezentacji
- popularność wykorzystania XML do wielu zastosowań, rozwiązań, systemów oraz fakt mocnego rozwijania XML przez wielu producentów oprogramowania i sprzętu (przyczyna wtórna)

Jak już wspomniano, XML służy m.in. do przechowywania danych. Nasuwa się zatem pytanie: czy można (i czy warto) wykorzystać XML jako podstawowy format baz danych. Przytaczając definicję bazy danych

„Baza danych jest to zbiór danych trwałych, które są wykorzystywane przez system aplikacji danej organizacji (lub przedsiębiorstwa)”²

wydaje się, że nie ma ku temu przeszkód.

XML posiada jednak także wady. Do najpoważniejszych należy rozwlekłość kodu – duża ilość znaczników mocno wpływa na marnotrawienie mocy obliczeniowej na parsowanie dokumentów XML. Do tego dochodzi stałe ułożenie danych względem siebie. Rzeczywistość pokazuje, że bardzo często wymagana jest możliwość zestawienia danych według różnych kryteriów. Wówczas każdorazowe przeszukiwanie dokumentu XML i generowanie nowego, zawierającego wybrane dane w odpowiedniej kolejności, jest bardzo nieefektywnym rozwiązaniem. Widać zatem, i nie można zdecydowanie i bez głębszej analizy stwierdzić, że systemy baz danych oparte na XML będą posiadały równie wiele zalet jak sam XML. Wpływa na to fakt, że SZBD (system zarządzania bazą danych) nie tylko przechowuje dane, ale także je przetwarza.

2. CO TO JEST NATYWNA XML-OWA BAZA DANYCH?

Zgodnie z informacjami zawartymi w [BOURRET XML], określenie „natywna baza danych XML” po raz pierwszy pojawiła się w kampanii marketingowej bazy Tamino firmy Software AG³. Firma ta jest w Polsce znana dzięki dużej akcji promującej Tamino w ramach konkursu na początku 2002 roku.

Natywne bazy danych XML (ang. *Native XML Databases*) służą do przechowywania danych w formacie XML [MADE 2003]. Fundamentalną jednostką składowania danych w relacyjnych bazach danych jest krotka, w bazach obiektowych jest nią obiekt. W natywnej bazie danych XML podstawową jednostką jest dokument XML. Nie mówi się natomiast nic o tym, jak fizycznie te dokumenty są przechowywane.

² C.J. Date *Wprowadzenie do systemów baz danych*.

³ <http://www.softwareag.pl>

wywane. Możliwych rozwiązań jest kilka. Mogą to być zwykłe pliki tekstowe, wykorzystana (w odpowiedni sposób) relacyjna, obiektowa lub hierarchiczna baza danych, bądź opracowany własny format składowania danych. Wspólnym mianownikiem tych pomysłów jest oparcie się na składowaniu tekstowym. Można także składować dokumenty XML w postaci binarnej, poprzez zapisanie binarnego modelu dokumentu. Jednym z takich modeli jest DOM⁴ (ang. *Document Object Model*).

Interfejsy, jakie oferują natywne bazy danych, muszą być oparte na XML. Możliwe rozwiązania to [1, 7] XPath⁵, XQuery⁶, XQL⁷, DOM. Można także napisać API (ang. *Application Program Interface*), np. mocno rozwijane XML:DB API.

Natywne bazy danych XML stanowią ostatnio wielkie hasło reklamowe wielu firm. W następnym punkcie tej pracy pokazane jednak będzie, iż nie zawsze natywna baza danych jest wyborem korzystnym.

Nieco szerszym pojęciem od natywnych baz danych XML są systemy zarządzania zawartością XML. Systemy te zbudowane są na natywnej bazie danych, ale umożliwiają znacznie więcej ([XML:DB] i [MADE 2003]):

- operacje dokumentowe wyższego poziomu (edycja zawartości)
- zarządzanie metadanymi
- kontrolowanie wersji
- komponowanie obiektów z dostępnych obiektów.

3. JAK DOKONAĆ WYBORU SYSTEMU BAZY DANYCH?

Postawione w tytule tego rozdziału pytanie dotyczy problemu wyboru rodzaju systemu bazy danych. Największym powodzeniem cieszą się obecnie relacyjne bazy danych. Sprawdziły się w wielu poważnych zastosowaniach, oferują bardzo duże możliwości, a ich implementacje są wysoce wydajne i odporne na błędy oraz oferują wysoki poziom bezpieczeństwa. Ich możliwości są na tyle duże, że ostatnio oferują nawet obsługę danych w formacie XML. Takie bazy danych przyjęło się określać terminem *XML-enabled*, jako że oferują one tylko wsparcie dla XML. Wsparcie to polega na imporcie i eksporcie dokumentów XML [OGRO 2001], natomiast dane są przechowywane w innej postaci niż XML. Wiąże się z tym kilka konsekwencji, które czasami mogą okazać się na tyle poważne, że nie można zaakceptować takiego rozwiązania.

Wiarygodne źródła [DODDS 2001] podają, że kryteria wyboru „baza danych XML-enabled vs natywna baza danych” nie są łatwe do sprecyzowania i przed podjęciem decyzji trzeba poświęcić chwilę na zastanowienie się, jakie kwestie są kluczowe.

⁴ <http://www.w3.org/DOM>

⁵ <http://www.w3.org/TR/xpath>

⁶ <http://www.w3.org/TR/xquery>

⁷ <http://www.w3.org/TR/NOTE-xml-ql>

Zanim przedstawione zostaną niektóre zagadnienia dotyczące odpowiedzi na tytułowe pytanie, warto jeszcze wspomnieć o ważnym podziale dokumentów XML. Dokumenty XML dzielą się na dwie grupy:

- dano-centryczne (ang. *data-centric*)
Służą do transportu danych pomiędzy aplikacjami, a kolejność elementów najczęściej nie ma większego znaczenia. Powstają w wyniku działania aplikacji, mierników, katalogów on-line. Bardzo często takie dokumenty XML są mało czytelne dla człowieka. Przykładami są różnego rodzaju zamówienia, faktury, cenniki, listy adresowe, dane naukowe, wyniki pomiarów. Charakterystyczną cechą jest duża regularność w budowie takiego dokumentu (stała liczba elementów i atrybutów).
- dokumento-centryczne (ang. *document-centric*)
Budowa takich dokumentów jest znacznie mniej regularna, jednak kolejność elementów jest istotna. Dokumenty te są bardziej czytelne dla człowieka i często służą do przechowywania np. książek, artykułów, listów. Najczęściej pisane są przez człowieka lub powstają w wyniku konwersji dokumentu z innego formatu, ale treść pochodzi od człowieka.

Podział ten, choć czasami w praktyce trudny do wyznaczenia, okazuje się być ważny przy wyborze systemu bazy danych.

Przejdziemy teraz do pokazania jak niektóre czynniki mają wpływ na wybór typu systemu zarządzania bazą danych, przy czym rozważane będą głównie modele relacyjny i natywny.

Dokumenty dano-centryczne

Jeżeli można stwierdzić, że realizowany projekt będzie operować na dokumentach dano-centrycznych, wówczas można z powodzeniem wykorzystać relacyjny model bazy danych. Możliwa jest bowiem konwersja struktury dokumentu XML na strukturę tabel. Przykładem może być sposób przechowywania i przetwarzania wiadomości w serwisie WWW Dolnośląskiego Oddziału PTI⁸:

```
<notatka id="id w BD" data="data w formacie rrrr-mm-dd">
  <tytul>Tytuł notatki</tytul>
  <autor mail="email">Imię i nazwisko</autor>
  <zawartosc>
    Treść notatki, być może ze <b>znacznikami</b>.
  </zawartosc>
</notatka>
```

przechowywana jest w relacyjnej bazie danych w tabeli o następującej strukturze⁹:

⁸ <http://www.pti.wroc.pl>

⁹ W rzeczywistości odpowiadający model relacyjny jest nieco odmienny ze względu na pewnego rodzaju optymalizacje, nie ma to jednak znaczenia dla przytoczonego przykładu.

```
CREATE TABLE `notatki` (
  `IdNotatki` int(11) NOT NULL auto_increment,
  `Data` date NOT NULL default '0000-00-00',
  `ImieNazwisko` varchar(100) NOT NULL default '',
  `Mail` varchar(100) default NULL,
  `Tytul` varchar(250) NOT NULL default '',
  `Tresc` text NOT NULL,
  PRIMARY KEY (`IdNotatki`),
  KEY `Data` (`Data`),
  KEY `IdRedaktora` (`IdRedaktora`)
) TYPE=MyISAM COMMENT='Notatki (newsy) w aktualnosciach'
```

Aby tego rodzaju odwzorowanie było możliwe lub żeby schemat tabel nie powodował konieczności tworzenia bardzo nieefektywnych zapytań, struktura dokumentu XML (opisana przykładowo za pomocą DTD¹⁰ – *Document Type Definition*) nie może być zbyt skomplikowana. Najlepszym przypadkiem są dokumenty o bardzo sztywnej strukturze, gdy wszystkie rekordy mają takie same pola i wszystkie są niepuste. Przykładem może być książka telefoniczna [DODDS 2001].

Istnieje kilka algorytmów tworzenia schematu bazy danych na podstawie dokumentu XML [OGRO 2001]. Najprostszy z nich tworzy osobną tabelę dla każdego elementu i atrybutu (a przy pomocy kluczy obcych wyrażone są ich wzajemne powiązania), a najbardziej wyrafinowane dokonują najpierw uproszczenia DTD. Translator DB→XML lub DB↔XML może być integralną częścią systemu bazy danych (właśnie wówczas mamy do czynienia z bazą XML-enabled) lub może być niezależną aplikacją i stanowi wówczas warstwę pośrednią (ang. *middleware*).

Dokumenty dokumento-centryczne

W przypadku dokumentów dokumento-centrycznych prawdopodobnie najlepszym rozwiązaniem będzie użycie natywnej bazy danych XML, ponieważ struktura takiego dokumentu XML nie jest łatwa do odwzorowania na tabele. Dokumenty te mogą posiadać struktury rekurencyjne (np. rozdział zawiera rozdział, który interpretowany jest jako podrozdział), złożone powiązania pomiędzy elementami (na przykład wiele warunkowych wystąpień elementów dzieci wstawiałoby w modelu relacyjnym wiele atrybutów o wartości NULL, co oczywiście jest sytuacją niepożądaną).

Rozróżnia się także dokumenty o częściowo ustrukturyzowanej budowie (ang. *semi-structured data*). Przykładem takiego dokumentu jest karta zdrowotna. W zależności od przebytych chorób i leczenia pewne fragmenty karty są puste, ale inne są wypełnione¹¹. Pomimo, że nie możemy określić takiego dokumentu jako dokumento-centrycznego, natywne bazy danych zdecydowanie lepiej nadają się do ich przechowywania niż relacyjne [DODDS 2001].

¹⁰ Punkt 2.8 z <http://www.w3.org/TR/REC-xml>

¹¹ Rozpatrując tylko te części wypełnione, można by nawet stwierdzić iż jest to dokument typu dano-centrycznego.

Struktura head-body

Bardzo często dokumenty XML mają strukturę typu *head-body*. W części *head* zawarte są wówczas informacje o dokumencie, a część *body* zawiera właściwy dokument. Praktyka pokazuje, że głównie informacje w części *head* stanowią indeksy i najczęściej to do nich wykonuje się zapytania. Wówczas proponowanym rozwiązaniem jest wykorzystanie relacyjnej bazy danych, w której metadane z części *head* zostaną odwzorowane do tabel (są to zwykle dane o dość regularnej strukturze), a dokument właściwy w całości zapisany w polu typu CLOB (ang. *Character Large Object*). Jeżeli jednak w konkretnym przypadku zapytania najczęściej wykonuje się do części *body*, wówczas zaleca się skorzystać z natywnej bazy danych.

Zapytania

Wiedza o typie zapytań, jakie będą wykonywane na bazie danych ma duży wpływ na podjęcie decyzji o wyborze systemu zarządzania bazą danych. Wszelkie zapytania, które wykonują rekurencyjne przeszukiwania (zwłaszcza wyszukiwanie pełnotekstowe), zdecydowanie efektywniej się wykonują w środowisku natywnym. Umożliwiają to na przykład konstrukcje XPath, których język SQL nie posiada. Natomiast, jeżeli dość często wykonuje się zapytania na prostej strukturze dokumentu XML, wówczas przewaga SQL jest znaczna, gdyż posiada bardziej rozbudowany zestaw operatorów manipulujących danymi [DODDS 2001].

Aplikacje

Bardzo ważnym aspektem są aplikacje, które będą korzystać z bazy danych. Jeżeli aplikacja nie posiada mechanizmów dobrej obsługi XML lub aplikacja świetnie obsługująca XML musi się skomunikować z inną aplikacją, która tego nie robi, wówczas należy liczyć się z dodatkowym narzutem związanym z tłumaczeniem danych pobranych z bazy na inny format niż XML. W takich sytuacjach wybór natywnej bazy danych może okazać się nie najlepszy. Być może lepiej byłoby przechowywać dane w bazie typu XML-enabled i w zależności od aplikacji łączącej się z SZBD (System Zarządzania Bazą Danych) eksportować dane w formacie XML lub nie. Należy pamiętać, że w przypadku błędnie podjętej decyzji, negatywne skutki mogą być bardzo uciążliwe oraz kosztowne w przyszłości. Zbyt łatwowerne przyjęcie stwierdzenia, iż XML jest panaceum na wszystko, może zadecydować o niepowodzeniu projektu. Należy się też spodziewać dalszego rozwoju systemów typu XML-enabled i ich możliwości mogą się wkrótce okazać zniechęcające do używania natywnych baz danych, zwłaszcza, jeżeli już zainwestowano w drogie systemy relacyjne lub obiektowe (na przykład firm Oracle, IBM, Microsoft).

Schematy XML

Czasami mamy do czynienia z dokumentami XML, które są opisane przez kilka schematów (DTD lub XMLSchema), nie posiadają schematu w ogóle lub schematy te są dość niestabilne i często ulegają zmianom. Są to wszystko sytuacje, w których wykorzystanie relacyjnych baz danych jest praktycznie niemożliwe, a przynajmniej bardzo mocno odradzane [DODDS 2001].

Import – eksport

Jeżeli kolejność elementów jest istotna w dokumencie XML (dokumentocentryczny XML), wówczas korzystanie z modelu innego niż natywny jest złym pomysłem, ze względu na to, iż nie ma żadnej gwarancji, że po transformacji XML do tabel, pobieranie danych z tabel i generowanie dokumentu XML zwróci kopię zgodną z oryginałem. Dzieje się tak, gdyż nie ma kontroli kolejności pobierania danych z tabel. Aby wyeliminować tę wadę należałoby efektywnie pamiętać kolejność elementów, przy czym obecnie nie ma takich rozwiązań.

Integracja danych

Jak już częściowo zasugerowano w rozdziale 1 tej pracy, XML stał się standardem wymiany danych pomiędzy aplikacjami. W sytuacji, gdy w ramach danego systemu nie ma aplikacji nieposługujących się XMLem warto skorzystać z usług natywnej bazy danych. Jest to rozwiązanie niejako naturalne, niewymagające kłopotliwej konwersji XML na model relacyjny i odwrotnie. Pobieranie danych z wielu źródeł i łączenie ich w jeden dokument bardzo dobrze realizowane jest w natywnych bazach danych, co podkreśla wielu znawców tematu „XML i bazy danych” [DODDS 2001].

Integralność danych

Dużą przewagą modelu relacyjnego nad natywnymi bazami danych jest integralność danych, który realizuje to poprzez referencje klucza obcego do tabeli, w której przechowywane są dane właściwe. Wówczas zmiana danych w jednym miejscu jest widoczna we wszystkich innych powiązanych. XML w zasadzie nie oferuje tego typu mechanizmów. Jednak nie we wszystkich zastosowaniach integralność danych jest pierwszą potrzebą.

4. DOSTĘPNE PRODUKTY

Do najbardziej znanych systemów zarządzania natywną bazą danych należą [HEIN 2003]:

- projekt Apache Xindice [XINDICE] – OpenSource
- eXist Wolfganga Meiera – OpenSource
- eXtensible Information Server (XIS) firmy Sonic – komercyjny
- Tamino (Software AG) – komercyjny
- Ipedo XML Database (Ipedo) – komercyjny

Są to niewątpliwie najbardziej zaawansowane projekty, nadal rozwijane i pielęgnowane, dlatego warto się przyjrzeć im nieco bliżej.

Projekt Apache Xindice

Xindice jest systemem zarządzania natywną bazą danych rozwijanym na zasadach OpenSource przez The Apache Software Foundation [XINDICE]. W grudniu 2001 roku kod systemu dbXML został przekazany firmie Apache i na tej podstawie powstał Xindice. Celem, jaki przyświecał twórcom Xindice, było stworzenie SZBD całkowicie ukierunkowanego na dokumenty XML.

Xindice jako języka zapytań używa XPath. Do uaktualniania danych służy język XML:DB XUpdate. Dostępne jest także API dla Javy XML:DB API. Dla innych języków programowania przygotowano plugin XML RPC.

Xindice udostępnia także operacje (na przykład wykonywanie zapytań) na kolekcjach dokumentów¹². Twórcy Xindice umożliwili przechowywanie dokumentów o różnych schematach w jednej kolekcji, co daje nowe możliwości i ułatwia realizację niektórych zadań. Kolekcje są indeksowane, co znacznie skraca czas realizacji kwerend.

Wielowątkowy motor bazodanowy Xindice jest bardzo wydajny ze względu na liczne optymalizacje pod kątem obsługi XML [MADE 2003]. Dokumenty są przechowywane w postaci skompresowanej, co stanowi kolejny atut tego systemu.

Wadami Xindice jest brak transakcyjności na poziomie węzłów i dość ograniczone możliwości tworzenia zapytań (brak wsparcia dla XQuery), a także brak narzędzi administracyjnych wykorzystujących interfejs graficzny (GUI) [HEIN 2003].

Więcej informacji o Xindice dostępnych jest w [XINDICE].

eXist

Jest to system całkowicie napisany w Javie przez Wolfganga Meiera, udostępniany na licencji OpenSource. Posiada wiele możliwości uruchomień (standalone, servlet lub jako komponent aplikacji). Wykazuje dobre wyniki, gdy rzadko wykonywane są operacje update (ze względu na brak wsparcia dla XUpdate). Uaktualnienia wykonywane są tylko na poziomie dokumentów a nie kolekcji, na których rozmiar nie ma w eXist ograniczeń¹³. Zaletą jest wydajne realizowanie zapytań. Udało się to

¹² Kolekcję można wyobrazić sobie jako analogię do systemu plików.

¹³ W rzeczywistości to ograniczenie wynosi 2^{31} dokumentów w jednej kolekcji – jest to jednocześnie maksymalna liczba dokumentów w całej bazie danych.

osiągnąć za pomocą zapytań XPath wzbogaconego o wiele własnych rozszerzeń (na przykład własne operatory wyszukiwania pełnotekstowego) oraz wykorzystaniu mechanizmu indeksowania. Niestety nie zaimplementowano jeszcze pełnej obsługi XPath. Inną zaletą jest wsparcie dla XML:DB API. Do wad systemu należy zaliczyć brak obsługi języka XQuery. Planowane jest dodanie wielu nowych cech w kolejnych wersjach systemu, głównie wsparcie dla XUpdate i pełne wsparcie dla XPath. Autor systemu przeprowadził testy (z pozytywnym wynikiem) na platformach Linux (SuSE 7.1, Mandrake 8.2/9.0), Solaris 8 i Windows 2000/ME [EXIST].

eXtensible Information Server (XIS)

Jest to już dojrzały system, rozpowszechniany na komercyjnych warunkach przez firmę Sonic¹⁴, której klientami są największe światowe firmy. XIS charakteryzuje się wysoką wielowątkowością, co bardzo pozytywnie wpływa na wydajność. Operacje (na przykład zapytanie) wykonywane są na poziomie węzłów, a nie całego modelu DOM, dzięki czemu nie blokują całego dokumentu. Nie tylko zwiększa to wydajność, ale także procentuje mniejszym zapotrzebowaniem na pamięć. W XIS zaimplementowano języki XPath i XQuery. Do opisu dokumentów XML używa się DTD lub XMLSchema, ale walidacja dokumentów wykonywana jest tylko przy pierwszym zapisaniu dokumentu w bazie danych (przy uaktualnianiu już nie), co może być postrzegane jako wada. XIS posiada własne API dla języka Java, COM i .NET. Niestety nie ma wsparcia dla XUpdate oraz XML:DB API. System dostępny jest na platformy Windows, Solaris, HP-UX i RedHat Linux, a jego cena wynosi co najmniej \$40,000 (w zależności od wersji) [HEIN 2003].

Tamino XML Server

Tamino jest pierwszym systemem zarządzania natywną bazą danych udostępnianym na warunkach komercyjnych. Firma SoftwareAG posiada 47,5% rynku XMLowych baz danych [HEIN 2003]. Tamino charakteryzuje się wysoką wydajnością, świetnie integruje dane z wielu źródeł (nie tylko XML, ale także PDF, MS OFFICE, audio, video itd.). Obsługuje DOM, JDOM, SAX, XML:DB API. Posiada jednak pewne wady: jako język zapytań obsługuje tylko XPath (ale z własnym rozszerzeniem dla zapytań do wielu dokumentów) oraz w czasie operacji wykonuje blokowanie na poziomie dokumentu (a nie węzłów). Serwer dostępny jest na platformach Windows 2000 i XP, Solaris 8 i 9, AIX 5.1 i 5.2, HP-UX 11.0 i 11i, Linux (RedHat i SuSE), przy czym najbardziej zaawansowane prace dotyczą wersji dla systemów Windows. Cena produktu wynosi \$45,000 za każdy procesor [HEIN 2003].

¹⁴ Firma Sonic przejęła w październiku 2002 r. firmę eXcelon – pierwotnego producenta serwera XIS.

Ipedo XML Database

Jest to SZBD wykorzystujący hierarchiczny model bazodanowy do przechowywania dokumentów XML. Podobnie jak Tamino dobrze współpracuje z innymi systemami zarządzania bazami danych i aplikacjami. Posiada bardzo dobre wsparcie dla transakcji (spełniają właściwości ACID) oraz mechanizm wersjonowania plików (dokumentów XML). Obsługuje zarówno XPath jak i XQuery. Podobnie jak XIS, dopuszcza definicję dokumentów XML za pomocą DTD i XMLSchema, ale tu walidacja wykonywana jest przy każdym zapisie dokumentu (a nie tylko przy pierwszym). Posiada własne API dla Java i COM. Do wad możemy zakwalifikować brak wsparcia dla XUpdate oraz XML:DB API. Podobnie jak poprzednie bazy danych Ipedo dostępna jest dla systemów Windows, Solaria i Linux. Cena wynosi \$29,000 za każdy procesor.

Porównanie niektórych cech opisanych powyżej produktów znajduje się w tabeli 1.

5. PODSUMOWANIE

Przeszukując zasoby Internetu bardzo często można przeczytać bardzo pochlebne artykuły na temat możliwości natywnych XMLowych baz danych. Okazuje się, że rzeczywiście, w niektórych sytuacjach natywna baza danych jest najlepszym rozwiązaniem z możliwych. Problem w tym, że nie zawsze tak jest. Wielokrotnie wybór natywnej bazy danych zamiast innej (na przykład relacyjnej) byłby dużym błędem mogącym „zaprocentować” w przyszłości dużymi stratami finansowymi. Powodów wyciągnięcia takich wniosków jest kilka.

- Możliwość translacji dano-centricznych dokumentów XML do modelu relacyjnego, obiektowego lub hierarchicznego. SZBD oparte na tych modelach oferują bardzo duże możliwości i są bardzo wydajne.
- Coraz lepsza obsługa danych w formacie XML przez „tradycyjne”¹⁵ SZBD, co znacznie zmniejsza przewagę natywnych baz danych jako jedyne rozwiązanie problemu przechowywania dokumentów XML

Finanse. Zakup licencji na przykład SZBD firmy Oracle to spory wydatek dla większości firm. Zarzucenie w pewnym momencie rozpoznanego już środowiska, w którym pracują wszystkie aplikacje firmy, konieczność szkolenia pracowników lub zatrudnienie nowych, przepisanie aplikacji w celu przystosowania ich do korzystania z natywnej bazy danych wiąże się z tak wysokimi kosztami i ryzykiem, że jeśli nie widać wyjątkowej potrzeby podjęcia takiej decyzji, nie powinna ona zostać podjęta.

¹⁵ Poprzez tradycyjne rozumiany jest tu model relacyjny, obiektowy lub hierarchiczny

Tabela 1. Porównanie wybranych natywnych XMLowych baz danych (źródło: [HEIN 2003])

	Tamino XML Server	Ipedo XML Database	eXtensible Information Server (XIS)	eXist	Xindice
XPath support	Yes	Yes	Yes	Yes	Yes
XQuery support	No	Yes	Yes	No	No
Schema support	Schema-less, XML Schema and DTD	Schema-less, XML Schema and DTD	Schema-less, XML Schema and DTD	Schema-less, XML Schema and DTD	Schema-less, XML Schema and DTD
XML:DB API	Yes	No, proprietary	No, proprietary	Yes	Yes
XSL support	Yes	Yes	Yes	No	No
Indexing	Yes	Yes	Yes	Yes	Yes
Output formatting	Yes	Yes	Yes	No	No
OS Support	Windows NT, 2000; Sun Solaris 7 & 8, IBM AIX 4.3.3, HP-UX 11, SuSE Linux	Sun Solaris 7 & 8, Red Hat Linux (version 6 & 7), Windows NT, 2000, XP	Windows(NT 4, 2000, XP), Solaris(version 2.6, 7, 8), HP-UX (version 11), Red Hat Linux (version 7.1)	Linux, Solaris and Windows 2000/ME	Unix and Windows

	Tamino XML Server	Ipedo XML Database	eXtensible Information Server (XIS)	eXist	Xindice
Backend database support	Native XML DB and other data stores (relational, etc)	True native XML DB	Object-oriented and other data stores (relational, etc)	Pure native XML DB, with relational option	Pure native XML DB
Supports custom extension	Yes	No	Yes	No	Yes, via XMLObjects
Versioning capabilities	Yes	Yes	No	No	No
Enterprise-class backup/restore/ad	Yes	Yes	Yes	No	No
Access	HTTP, SOAP, WebDAV, Java API, COM API	SOAP, Java API, COM API	Java and COM-based APIs; also .NET	HTTP, XML-RPC, SOAP and WebDAV	XML-RPC plug-in
Data source adapters	Yes; advanced mapping – leave data in relational but access via Tamino	Yes; extra cost add-on	Yes, import data from any OLE DB or ODBC data source, Xconnect to 100s of data sources	No	No
Concurrency Locking	At document level	Locks at the node level	Locks at the node level; also lets queries access data while node is	At document level	At document level
Validation	Occurs on initial load and during	Occurs on initial load and during	Only occurs on initial load, not	No	No

Prace nad natywnymi bazami danych z pewnością będą kontynuowane ze względu na obszary zastosowań, w których ich wybór jest najkorzystniejszy. Natywnych baz danych nie należy jednak traktować jako rozwiązania optymalnego dla każdego przypadku.

LITERATURA

- [BOURRET XML] Bourret R. XML and Databases,
<http://www.rpbouret.com/xml/XMLAndDatabases.htm>
- [BOURRET PROD] Bourret R. *XML Database Products*,
<http://www.rpbouret.com/xml/XMLDatabaseProds.htm>
- [CARR 2001] Carr D.F. *XML-Native Databases*,
<http://www.internetworld.com/magazine.php?inc=071501/07.15.01technology3.html>
- [DODDS 2001] Dodds L., *XML and Databases? Follow Your Nose*,
<http://www.xml.com/lpt/a/2001/10/24/follow-yr-nose.html>
- [EXIST] eXist Home Page <http://exist.sourceforge.net/>
- [HEIN 2003] Heindl M. J., *Native XML Databases*,
<http://students.depaul.edu/~mheindl/Final%20Report-Mark%20Heindl.html>
- [HOLZ 2001] Holzner Steven, *XML – Vademecum profesjonalisty*, Gliwice, Helion, 2001.
- [IPEDO] Ipedo XML Database
http://www.ipedo.com/html/ipedo_xml_database.html
- [MADE 2003] Madeyski L., *XML w bazach danych*, Prace Naukowe Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej, Bazy Danych, Zeszyt 4, 2003, 83–91.
- [OGRO 2001] Ogrodniczuk M., *XML w bazie danych*, Software 2.0, 18 (84), 28–34.
- [TAMINO] Tamino XML Server Home Page
<http://www.softwareag.com/tamino/>
- [W3C] The World Wide Web Consortium (W3C), <http://www.w3.org>
- [XINDICE] Projekt Apache Xindice, <http://xml.apache.org/xindice>
- [XML:DB] Inicjatywa XML:DB <http://www.xmldb.org>

NATIVE XML DATABASES

The paper presents native XML databases (NXD) and the problem of their usefulness is discussed. It turns out that a NXD is not always the best solution – sometimes it is even a wrong one, also when an application uses XML as a basic data format. Very often it is better to transform XML document into another model of databases, for example into relational database. Some native XML databases are also presented in this paper.

*modele baz danych,
obiektywne bazy danych,
temporalne bazy danych,
probabilistyczne bazy danych*

Zygmunt MAZUR*
Piotr MICHAŁEC*

TEMPORALNO-PROBABILISTYCZNO- OBIEKTOWE BAZY DANYCH

Wiele zastosowań baz danych operuje danymi i wyrażeniami w sytuacjach, gdzie istnieje niepewność, kiedy wystąpi dane zdarzenie. Zastosowania te w naturalny sposób są reprezentowane przez modele obiektowe, które pozwalają na łatwy w zrozumieniu opis systemu. Natomiast czas (zależności temporalne) i niepewność (zależności probabilistyczne) parametryzują różne cechy tego modelu. W pracy zostanie przedstawiony nowy model temporalno-probabilistyczno-obiektowej bazy danych. Ten model danych pozwala powiązać z każdym zdarzeniem zbiór punktów czasu, a z każdym punktem czasu prawdopodobieństwo wystąpienia danego zdarzenia.

1. WPROWADZENIE

Opracowanie paradygmatu obiektowego [Atkinson89, Kim96] spowodowało fundamentalną zmianę w sposobie spojrzenia na dane i procedury na nich operujące. Tradycyjnie dane i procedury były przechowywane oddzielnie: dane i związki między nimi w bazie, natomiast procedury w programie użytkowym. W modelu obiektowym następuje połączenie procedur encji z jej danymi. Połączenie takie generalnie uważa się za krok naprzód w zarządzaniu danymi. Encje stają się niezależnymi jednostkami, które stosunkowo łatwo mogą być wielokrotnie wykorzystywane i przenoszone. Zachowanie encji nie jest już powiązane z określonym programem użytkowym, lecz stanowi część samej encji, tak więc bez względu na to, gdzie użyta jest encja, zachowuje się ona w przewidywalny sposób. Te i inne cechy systemu obiektowego ([Figura96], [Harrington91], [Kim96], [Lausen00]) predysponowane są, w naturalny sposób, do przechowywania informacji dla systemów wspomagających projektowanie, produkcję (CAD, CAE, CASE i CAM), tworzenie oprogramowania i systemów, które je wykorzystują: systemy baz wiedzy (np. systemy doradcze,

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

systemy informacji geograficznej), systemy multimedialne, które operują na obrazach, dźwiękach i dokumentach tekstowych oraz systemy języków programowania.

Wiele z tych zastosowań operuje danymi i wyrażeniami w sytuacjach, gdzie istnieje niepewność, kiedy wystąpi dane zdarzenie. Przykładowym wykorzystaniem probabilistyczno-temporalnych baz danych są np.:

- firma logistyczno-transportowa, gdzie na podstawie danych statystycznych można określić aktualne miejsce, w którym znajduje się przesyłka. System bazy danych używany przez taką firmę musi mieć zdolność przechowywania temporalnych form niepewności, np. „Przesyłka będzie dostarczona określonego dnia z prawdopodobieństwem $1/3$ lub dnia następnego z prawdopodobieństwem $2/3$ ”.
- systemy przewidywania warunków atmosferycznych, gdzie w bazie danych przechowuje się warunki pogodowe dla określonej lokalizacji. W bazie takiej znajdują się dane dotyczące nie tylko czasu przeszłego, ale również prognozy na przyszłość. Wiąże się z tym konieczność przechowywania prawdopodobieństwa wystąpienia określonych warunków atmosferycznych, np. „Prawdopodobieństwo, że będzie jutro padać wynosi $2/5$ ”.
- numeryczne modele finansowe, gdzie operuje się na aktualnych i przewidywanych wartościach akcji spółek giełdowych. Jest wiele programów przewidujących zachowanie akcji: „Spodziewamy się, z pewnością 60% procent, że akcje firmy *PEWNY_ZYSK* stracą na wartości 10% w ciągu najbliższych 2 tygodni”. Przechowując w bazie danych takie prognozy należy mieć możliwość ich prezentowania i manipulowania nimi.

Zastosowania tego typu w naturalny sposób są reprezentowane przez modele obiektowe, które pozwalają na jednoznaczny i czytelny opis systemu. Natomiast czas (zależności temporalne) i niepewność (zależności probabilistyczne) parametryzują różne cechy tego modelu.

2. STAN BADAŃ I POZYCJE LITERATUROWE

Prac naukowych na temat integracji informacji temporalnej i probabilistycznej w bazach danych jest bardzo mało. W pierwszej publikacji na ten temat [Snodgrass98], przedstawiono podejście do niepewności temporalnej w relacyjnych bazach danych oparte na rachunku prawdopodobieństwa. Autorzy tej pionierskiej pracy zrobili ważny pierwszy krok do rozwiązania problemu reprezentacji informacji temporalno-probabilistycznej w bazach danych. Rozszerzyli model danych i język zapytań SQL o niepewność wystąpienia zdarzenia w różnych punktach czasu. Powiązali czas zdarzenia ze zbiorem możliwych chwil ograniczających, kiedy zdarzenie może wystąpić oraz rozkładem prawdopodobieństwa na tym zbiorze. Opisali również rozszerzenie języka zapytań SQL, do uzyskiwania informacji przy

występowaniu niepewności, o konstrukcję definiującą wiarygodność uporządkowania, która jest liczbą całkowitą z przedziału $[1, 100]$, wyznaczającą do jakiego stopnia wynik zapytania SQL powinien zawierać niepewność w odpowiedzi (wartość 1 oznacza, że możliwa jest każda odpowiedź na zapytanie, podczas gdy wartość 100, że tylko odpowiedzi, w których nie ma niepewności). Ponadto Snodgrass i Dyreson wprowadzili konstrukcję definiującą wiarygodność związków, która określa proste modyfikacje rozkładów prawdopodobieństwa w bazowych relacjach przed przetwarzaniem warunków zawartych w zapytaniu SQL. Ten sposób przedstawienia niepewności czasu niesie ze sobą ryzyko zwiększenia złożoności obliczeniowej wykonywania zapytań, wobec czego autorzy przedstawili również wydajne algorytmy przetwarzania zapytań.

Innymi pracami podejmującymi temat rozszerzenia relacyjnego modelu danych o niepewność temporalną są [Dekhtyar01], [Dekhtyar02], w których autorzy wprowadzili składnię relacji temporalno-probabilistycznych i pokazali, jak można je konwertować do łatwych w odbiorze, ale mniej wydajnych form, nazywanych **relacjami opisującymi czas**. Zaprezentowali również **teoretyczną algebrę dla relacji opisujących czas** (ang. *TATA – Theoretical Annotated Temporal Algebra*). TATA jest wygodna do opisu zachowania algebraicznych wyrażeń, ale jest niepraktyczna w użyciu, ponieważ relacje opisujące mogą być nieprawdopodobnie duże. Następnie zaprezentowali **algebrę temporalno-probabilistyczną** (ang. *TPA – Temporal Probabilistic Algebra*) i pokazali, że definicja TPA prowadzi do właściwej implementacji TATA mimo faktu, że operuje na zwiezłych relacjach temporalno-probabilistycznych zamiast na relacjach opisujących czas.

Są również inne prace na temat niepewności temporalnej w bazach danych, które do przedstawienia niepewności wykorzystują inne techniki niż rachunek prawdopodobieństwa. Autorzy prac [Dubois89], [Dutta89] proponują wprowadzenie logiki rozmytej do temporalnej niepewności. Wszystkie te prace wprowadzają zastosowanie niepewności temporalnej w relacyjnym modelu baz danych.

Rozszerzenie modelu obiektowego baz danych jest przedstawione w wielu pracach naukowych [Biazzo02], [Eiter01], [Lausen00], [Macyna00], jednak jedyną pracą, znaną przez nas, wprowadzającą zależności temporalno-probabilistyczne do obiektowej bazy danych jest [Biazzo02]. W pracy tej zaproponowano model danych oraz algebrę dla temporalno-probabilistyczno-obiektowych baz danych. Ten model danych pozwala powiązać z każdym zdarzeniem zbiór możliwych punktów czasu T , a z każdym punktem czasu $t \in T$ przedział prawdopodobieństwa pokazujący, że zdarzenie e wystąpiło w punkcie czasu t . Autorzy wzorując się na [Dekhtyar01], [Eiter01] wprowadzili rozróżnienie pomiędzy jawnymi obiektami w bazie, gdzie zbiory punktów czasu z ich przedziałami prawdopodobieństwa są po prostu wyliczone (jak dla algebry TATA), a niejawnymi obiektami, gdzie zbiory punktów czasu są wyrażone przez punkty ograniczające, a ich przedziały prawdopodobieństwa przez funkcje rozkładu prawdopodobieństwa. Tak więc niejawne obiekty bazy są zwiezłym przedstawieniem jawnych. Zabieg ten pozwolił autorom na wydajną implementację

operacji algebraicznych, podczas gdy ich jawne odpowiedniki ułatwiły zdefiniowanie operacji algebraicznych. W pracy zdefiniowano również operacje algebraiczne: selekcji, selekcji z ograniczeniem, zmiany nazwy, rzutowania, wydobywania, naturalnego złączenia, iloczynu kartezjańskiego, warunkowego złączenia, operacji przecięcia, sumy i różnicy na obu instancjach, jawnych i niejawnych, obiektów bazy. Udowodniono, że każda operacja zdefiniowana na jawnych obiektach bazy jest prawidłowo zaimplementowana na ich niejawnych odpowiednikach.

Wobec tego, iż przeprowadzono dopiero pierwsze próby zdefiniowania temporalno-probabilistycznego modelu danych, celowe jest dalsze prowadzenie prac naukowych w tym temacie ze szczególnym uwzględnieniem rozszerzenia temporalno-probabilistycznego obiektowych baz danych.

3. PODSTAWOWE DEFINICJE DLA ROZSZERZENIA TEMPORALNO-PROBABILISTYCZNEGO

Ponieważ należy założyć, że w temporalno-probabilistycznych bazach danych czas jest określony na dowolnym, ale stałym kalendarzu, ważne są definicje zapisu kalendarza. Następujące definicje są przytoczone za pracą [Kraus96].

3.1. KALENDARZ

Definicja jednostki czasu. Jednostka czasu, oznaczana T , składa się z nazwy i zbioru wartości. Zbiór wartości jest uporządkowany liniowo. Jednostka czasu może być skończona lub nieskończona, w zależności czy jej zbiór wartości jest skończony czy nieskończony. Przykład: jednostki czasu nazwane dzień, miesiąc i rok mogą mieć zbiory wartości czasu odpowiednio: $\{1, \dots, 31\}$, $\{1, \dots, 12\}$, zbiór liczb całkowitych.

Definicja hierarchii liniowej jednostek czasu. Hierarchia liniowa jednostek czasu, oznaczana H , jest skończonym zbiorem różnych jednostek czasu z liniowym uporządkowaniem $\subseteq$ między nimi. Największa jednostka czasu może być skończona lub nieskończona, a pozostałe jednostki czasu w hierarchii muszą być skończone. Przykład hierarchii liniowej jednostek czasu:

$$H_1 = \{\text{dzień, miesiąc, rok}\}, \text{gdzie } \text{dzień} \subseteq \text{miesiąc} \subseteq \text{rok}.$$

Definicja punktu czasu. Załóżmy, że $H = \{T_1, \dots, T_n\}$ jest hierarchią liniową jednostek czasu, gdzie $T_1 \subseteq \dots \subseteq T_n$. Punkt czasu t w H jest n -krotką $(v_1, \dots, v_n)$ taką, że dla każdego i (gdzie $1 \leq i \leq n$), v_i jest wartością czasu w zbiorze wartości T_i .

Przykład: używając liniowej hierarchii H_1 , data: 10 Marca 2003 roku może być przedstawiona jako punkt czasu (10, 3, 2003).

Definicja kalendarza. Kalendarz K składa się z liniowej hierarchii jednostek czasu H i orzeczenia dopuszczalności oznaczonego V . Orzeczenie dopuszczalności jest funkcją logiczną, która dla danego kalendarza wyznacza niepusty zbiór dopuszczalnych punktów czasu, a $V(t)$ jest prawdziwe wtedy i tylko wtedy, gdy t jest dopuszczalnym punktem czasu. Zbiór wszystkich punktów czasu w kalendarzu K , oznaczany S_K , jest zdefiniowany jako

$$S_K = \{t: t \text{ jest punktem czasu w } H \text{ i } V(t) \text{ jest prawdziwe}\}.$$

3.2. FUNKCJE ROZKŁADU PRAWDOPODOBIENSTWA

Kolejne założenia i definicje są związane z zastosowaniem w pracy rachunku prawdopodobieństwa. Na tym etapie konieczne jest przedstawienie definicji funkcji rozkładu prawdopodobieństwa.

Niech $S = \{s_1, \dots, s_n\}$ będzie niepustym zbiorem zdarzeń $s_1, \dots, s_n$. Wówczas funkcja rozkładu prawdopodobieństwa na S jest odwzorowaniem p ,

$$p: S \rightarrow [0,1] \text{ takim, że } \sum_{s \in S} p(s) \leq 1.$$

Można teraz przedstawić definicje kilku funkcji rozkładu:

- Najczęściej używaną funkcją rozkładu jest **rozkład równomierny na S** , oznaczany U_S , który jest funkcją $U_S: S \rightarrow [0,1]$ zdefiniowaną przez

$$U(s_i) = 1/n \text{ dla wszystkich } s_i \in S.$$

- **Rozkład dwumianowy na S** , czyli **rozkład Bernoulli'ego**, dla $0 < p < 1$ oznaczony $B_{S,p}$ jest funkcją $B_{S,p}: S \rightarrow [0,1]$ zdefiniowaną jako

$$B_{S,p}(s_i) = \binom{n}{i} p^i (1-p)^{n-i} \text{ dla wszystkich } s_i \in S.$$

- **Rozkład geometryczny na S** , dla $0 < p < 1$ oznaczony $G_{S,p}$ jest funkcją $G_{S,p}: S \rightarrow [0,1]$ zdefiniowaną przez $G_{S,p}(s_i) = p(1-p)^i$ dla wszystkich $s_i \in S$.
- **Rozkład Poissona na S** , dla $\lambda > 0$ oznaczony $P_{S,\lambda}$ jest funkcją $P_{S,\lambda}: S \rightarrow [0,1]$ zdefiniowaną przez $P_{S,\lambda}(s_i) = e^{-\lambda} \lambda^i / i!$ dla wszystkich $s_i \in S$.

Można przedstawić wiele innych funkcji rozkładu prawdopodobieństwa, nie jest to jednak celem tej pracy.

4. TEMPORALNO-PROBABILISTYCZNO- OBIEKTOWE BAZY DANYCH

Model danych przedstawia logiczną strukturę obiektów ze świata rzeczywistego, nałożonych na nie więzów i związków między tymi obiektami. Model danych, w którym wykorzystano cechy obiektowości, jest obiektowym modelem danych ([Figura96], [Harrington91], [Kim96], [Lausen00]).

Obiektowy model danych jest ważnym punktem początkowym dla badań i rozwoju nowych modeli danych. W tym rozdziale zostanie przedstawiona definicja temporalno-probabilistyczno-obiektowego modelu danych, który jest rozszerzeniem modelu obiektowego.

4.1. TYPY I WARTOŚCI

Przed przystąpieniem do określenia modelu obiektowego należy zdefiniować zbiór typów oraz wartości.

Zbiór typów atomowych jest zdefiniowany następująco: $\Theta = \{\text{integer, string, Boolean, float, itd}\}$ wraz ze standardowymi dziedzinami dla tych typów. Dodatkowo należy uwzględnić temporalny typ atomowy określony jako kalendarz τ , którego dziedzina $\text{dom}(\tau)$ składa się z wszystkich dopuszczalnych w τ punktów czasu S_τ .

Zakładając, że A jest zbiorem nazw atrybutów, a Θ jest skończonym zbiorem typów atomowych, to zbiór wszystkich typów jest zdefiniowany przez indukcję:

1. Każdy typ atomowy z Θ jest typem.
2. Jeśli θ jest typem, wówczas $\{\theta\}$ jest typem, który jest nazywany typem zbiorowym.
3. Jeśli $A_1, \dots, A_K$ są atrybutami wzajemnie niezależnymi z A (niezależność atrybutów oznacza, że wartość każdego atrybutu może być zmieniana niezależnie od innych atrybutów) i $\theta_1, \dots, \theta_K$ są typami, to $[A_1: \theta_1, \dots, A_K: \theta_K]$ jest typem.

Każdy typ atomowy $\theta \in \Theta$ ma standardową dziedzinę $\text{dom}(\theta)$. Niech W będzie zbiorem wszystkich wartości, wówczas W definiujemy przez indukcję następująco:

1. Dla wszystkich typów atomowych $\theta \in \Theta$, każda wartość $w \in \text{dom}(\theta)$ jest wartością typu θ .
2. Jeśli $w_1, \dots, w_k$ są wartościami typu θ , to $\{w_1, \dots, w_k\}$ jest wartością typu $\{\theta\}$.
3. Jeśli $A_1, \dots, A_K$ są atrybutami wzajemnie niezależnymi z A i $w_1, \dots, w_k$ są wartościami typów $\theta_1, \dots, \theta_k$, wówczas $[A_1: w_1, \dots, A_k: w_k]$ jest wartością typu $[A_1: \theta_1, \dots, A_K: \theta_K]$.

W systemach obiektowych baz danych typ opisuje strukturalną część klasy. Klasa zamyka w sobie strukturę i specyficzne zachowanie: w szczególności są to operacje tworzenia i usuwania obiektów. Wartości o takiej samej strukturze charakteryzują się wspólnym typem, a obiekty wspólną klasą.

4.2. OBIEKT TEMPORALNO-PROBABILISTYCZNY

Z punktu widzenia struktury obiektu, obiekt temporalno-probabilistyczny może być przedstawiony za pomocą identyfikatora, przedziału czasu oraz rozkładu prawdopodobieństwa określonego na tym przedziale.

Definicja obiektu temporalno-probabilistycznego.

Obiekt jest zdefiniowany jako szóstka (i, k, t, p, σ, v) , w której:

i jest identyfikatorem obiektu,

k jest klasą obiektu,

t oznacza przedział czasu,

p jest prawdopodobieństwem wystąpienia obiektu,

σ jest funkcją rozkładu prawdopodobieństwa,

v jest wartością typu w danej klasie – stanem obiektu.

Identyfikator obiektu i jednoznacznie wyróżnia obiekt w przestrzeni obiektów, a k określa klasę grupy obiektów. Przedział czasu t przechowuje czas ważności obiektu, natomiast rozkład σ prawdopodobieństwa p w przedziale t pozwala na ustalenie ważności (pewności wystąpienia) obiektu w każdym punkcie czasu z przedziału t . Wartość v oznacza stan obiektu.

Ponieważ i, k, t, p oraz σ również są wartościami, to klasa jest typem. Wobec tego może być użyta jako typ w strukturze innej klasy.

4.3. TEMPORALNO-PROBABILISTYCZNO-OBIEKTOWY MODEL DANYCH

Klasy są ułożone w hierarchię, której podstawą są związki IsA między uczestniczącymi w niej klasami. Jeśli klasa k jest podległa klasie k' , to każdy obiekt k jest również obiektem k' . Tego rodzaju hierarchia klas jest podstawą własności dziedziczenia.

Po wprowadzeniu tych pojęć możemy teraz zdefiniować schemat struktury bazy, który przypisuje typy do klas zgodnie z odpowiednią hierarchią klas:

Schemat temporalno-probabilistyczno-obiektowej bazy danych jest czwórką

$$(K, IsA, \tau, I)$$

gdzie:

K jest skończonym zbiorem klas. Intuicyjnie zbiór ten skojarzony jest z obiektową bazą danych,

IsA jest hierarchicznym uporządkowaniem klas, tzn. jeśli $k IsA k'$, to k jest podklasą k' , a k' jest nadklasą k ,

τ jest odwzorowaniem $\tau(K)$, które przypisuje typ do każdej klasy ze zbioru K ,

I jest zbiorem obiektów, które mają przyporządkowaną klasę bazową ze zbioru K .

Baza obiektów zawiera obiekty, których klasy są uporządkowane w hierarchię, a każdy obiekt jest dołączony do swojej klasy bazowej i na tej podstawie interpretuje *IsA* zgodnie z zamierzoną semantyką.

5. DALSZY ROZWÓJ MODELU

W dalszych pracach nad temporalno-probabilistyczno-obiektowymi bazami danych należy rozwinąć teorię przez zdefiniowanie operacji algebraicznych, dla projektowanego modelu danych. Podstawowym zadaniem algebry jest umożliwienie tworzenia wyrażeń. Wyrażenia te mogą służyć do wielu celów między innymi do uzyskiwania danych, aktualizacji danych itp. Algebra stanowi również wygodną podstawę optymalizacji działań na bazie danych. Zgodnie z [Date 00], należy zdefiniować co najmniej siedem podstawowych operatorów:

- **restrykcja** – daje w wyniku zbiór obiektów spełniających określone warunki,
- **rzut** – operacja ta usuwa/przysłania atrybuty z ich wartościami z obiektów danej klasy,
- **złączenie** – daje w wyniku zbiór obiektów obu klas z kombinacjami takich wartości (tych obiektów), które mają jednakową wartość tego samego atrybutu,
- **suma** – oblicza sumę zbiorów obiektów dwóch klas, oraz sumę wartości związanych z obiektami tych klas,
- **różnica** – zwraca w wyniku zbiór obiektów pierwszej klasy z różnicą wartości związanych z obiektami obu klas,
- **przecięcie** – oblicza część wspólną zbiorów obiektów dwóch klas dla wspólnych wartości związanych z obiektami tych klas,
- **iloczyn kartezjański** – daje w wyniku zbiór obiektów obu klas z wszystkimi kombinacjami ich wartości.

Wszystkie te operacje powinny uwzględniać w działaniu oraz wynikach warunki temporalno-probabilistyczne.

6. PODSUMOWANIE

W pracy przedstawiono zarys nowego modelu temporalno-probabilistyczno-obiektowej bazy danych, który docelowo pozwala na powiązanie z każdym zdarzeniem zbioru punktów czasu, a z każdym punktem czasu prawdopodobieństwo wystąpienia danego zdarzenia. Definicję modelu oparto na podstawowych założeniach i definicjach: zapisu kalendarza oraz funkcji rozkładu prawdopodobieństwa. Przedstawiono

definicje typów oraz wartości. Wprowadzono definicje obiektu temporalno-probabilistycznego oraz schemat struktury bazy temporalno-probabilistyczno-obiektowej. Wskazano kierunek dalszych prac nad modelem.

Przedstawiono również kilka zastosowań baz danych, gdzie są używane modele obiektowe oraz występuje niepewność czasu wystąpienia zdarzenia. Zaprezentowano również obecny stan badań naukowych oraz pozycje literaturowe przedstawiające problematykę temporalno-probabilistycznych baz danych.

LITERATURA

- [Atkinson89] Atkinson M., Banilhom F., De Witt D., Dittrich K., Zdonik S., *The Object-Oriented Database System Manifesto*. Proceedings of the First Object-Oriented and Deductive Database Conference, s. 40–57. Kyoto, 1989.
- [Biazzo02] Biazzo V., Giugno R., Lukasiewicz T., Subrahmanian V., *Temporal Probabilistic Object Bases*. INFSYS Research Report, Wien, 2002.
- [Date00] Date C., *Wprowadzenie do systemów baz danych*. WNT, Warszawa, 2000.
- [Dekhtyar01] Dekhtyar A., Ross R., Subrahmanian V., *Probabilistic Temporal Databases*, part I: *Algebra*. ACM Transactions on Database Systems, vol. 26, s. 41–95, New York, 2001.
- [Dekhtyar02] Dekhtyar A., Ozcan F., Ross R., Subrahmanian V., *Probabilistic Temporal Databases*, part II: *Calculus and Query Processing*. Technical Report – University of Maryland, Washington, 2002.
- [Dubois89] Dubois D., Prade H., *Processing fuzzy temporal knowledge*. IEEE Transactions on Systems, Man, and Cybernetics, s. 729–744, New York, 1989.
- [Dutta89] Dutta S., *Generalized events in temporal databases*. Proceedings of the 5th International Conference on Data Engineering, s. 118–126, Los Angeles, 1989.
- [Eiter01] Eiter T., Lu J., Lukasiewicz T., Subrahmanian V., *Probabilistic Object Bases*. ACM Transactions on Database Systems, vol. 26, s. 264–312, New York, 2001.
- [Figura96] Figura D., *Obiektowe bazy danych*. Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1996.
- [Harrington91] Harrington J., *Obiektowe bazy danych dla każdego*. MIKOM, Warszawa, 1991.
- [Kim96] Kim W., *Wprowadzenie do obiektowych baz danych*. WNT, Warszawa, 1996.
- [Kraus96] Kraus S., Sagiv Y., Subrahmanian V., *Representing and integrating multiple calendars*. Technical Reports of University of Maryland, Washington, 1996.
- [Lausen00] Lausen G., Vossen G., *Obiektowe bazy danych: modele danych i języki*. WNT, Warszawa, 2000.
- [Macyna00] Macyna W., Mazur Z., *Temporalno – obiektowe modele baz danych*. Prace naukowe Wydziałowego Zakładu Informatyki Politechniki Wrocławskiej – Bazy Danych 1, s. 19–36, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 2000.
- [Snodgrass98] Snodgrass R., Dyreson C., „Supporting Valid-Time Indeterminacy”, ACM Transactions on Database Systems, vol. 23, s. 1–57, New York, 1998.
- [Ullman00] Ullman J., Widom J., *Podstawowy wykład z systemów baz danych*. WNT, Warszawa, 2000.

TEMPORAL PROBABILISTIC OBJECT DATABASES

There are many database applications which manipulate data and queries with uncertainty about the occurrence time of an event. These applications are naturally represented by object oriented models which allow to create easy in understanding description of database system. Features of that model are parametrized by time (time dependencies) and uncertainty (probabilistic dependencies). In this paper, it is presented a new temporal-probabilistic-object database model. The data model allows to associate with each event a set of time points, and with each time point, an interval for the probability that event occurred.

*grupowanie obiektów, obiektowe bazy danych,
bazy danych, efektywność przetwarzania danych*

Aleksander KŁOSOW*
Dariusz MURASZKO

ALOKACJA OBIEKTÓW BAZY DANYCH W PAMIĘCI STAŁEJ Z WYKORZYSTANIEM MECHANIZMÓW GRUPOWANIA

Obiektowe bazy danych obecnie są używane w wielu systemach informatycznych. Znaczący wpływ na efektywność przetwarzania danych w tego typu systemach mają mechanizmy grupowania obiektów w pamięci stałej. Aby dokonać właściwego wyboru strategii grupowania obiektów należy posiadać wiedzę na temat struktury obiektów w bazie danych oraz ich zachowania. Problem ten klasyfikuje się do problemów NP-zupełnych i w praktyce często wybór strategii grupowania opiera się na intuicji administratora bazy danych. W artykule podjęto próbę wykazania pewnych zależności pomiędzy skuteczną strategią grupowania a wybranymi charakterystykami bazy danych. Rozważania teoretyczne zostały poparte wynikami licznych eksperymentów komputerowych.

1. WPROWADZENIE

Zalety obiektowych baz danych w zaawansowanych systemach wykorzystujących bazy danych, takich jak CAD/CAM/CASE, systemy informacji multimedialnej, systemy sztucznej inteligencji, są szeroko uznawane i wykorzystywane [Banc92], [Ishi93]. Jednym z kluczowych kryteriów oceny tych systemów jest efektywność przetwarzania danych, w czym szczególną rolę odgrywa zmniejszenie liczby dostępów do danych w pamięci zewnętrznej. Aby sprostać takiemu zadaniu, zaproponowano kilka technik rozmieszczenia obiektów bazy danych w pamięci zewnętrznej. Techniki te są znane jako techniki grupowania (ang. *clustering*) lub gronowania, i polegają na znalezieniu takiej lokalizacji obiektów na stronach pamięci zewnętrznej, aby liczba pobieranych w trakcie operacji stron była minimalna. Główna idea polega na tym, aby obiekty, które są używane razem, zostały rozmieszczone blisko siebie w pamięci zewnętrznej, najlepiej na jednej stronie dysku fizycznego.

* Zakład Podstaw Informatyki i Teleinformatyki, Politechnika Wroclawska, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, klosow@wp.pl

Generalnie, do podstawowych celów grupowania obiektów należy:

- minimalizacja liczbyostępów do pamięci zewnętrznej w trakcie odczytywania i modyfikacji obiektów;
- zmniejszenie pojemności zablokowanej pamięci zewnętrznej, a tym samym zablokowanej liczby obiektów, przy współbieżnym dostępie kilku użytkowników;
- zmniejszenie liczby przesyłanych stron podczas transmisji obiektów w sieci;
- przyspieszenie wyszukiwania danych z wykorzystaniem indeksów.

Należy zaznaczyć, że w bazach danych fizyczne grupowanie danych jest jednym z najważniejszych czynników w osiągnięciu szybkiego dostępu do danych, a więc, efektywności działania całej bazy danych. W obiektowo zorientowanych bazach danych, w szczególności ze względu na bogatszą semantykę związków i wskutek tego bogatsze niż w relacyjnych bazach danych możliwości realizacji samych metod grupowania, zagadnienie to jest bardzo interesujące.

Celem pracy jest zbadanie wpływu grupowania na efektywność dostępu do danych w zależności od wybranych kryteriów grupowania oraz rodzaju transakcji występujących w bazie danych. Wszystkie eksperymenty będą przeprowadzone w oparciu o system zarządzania obiektowymi bazami danych ObjectStore 5.0 z wykorzystaniem własnego oprogramowania w języku Visual C++ 5.0.

2. GRUPOWANIE OBIEKTÓW BAZY DANYCH W PAMIĘCI STAŁEJ

2.1. WYBÓR STRATEGII ORAZ KRYTERIÓW GRUPOWANIA

Problem znalezienia najlepszej strategii grupowania dla danej bazy danych jest problemem NP-zupełnym, można jednak zawsze znaleźć strategie lepsze i gorsze. Istotnym elementem przy wyborze strategii grupowania są kryteria grupowania. Należy przede wszystkim zwrócić uwagę na dwa czynniki: strukturę obiektów i wzorzec dostępu stosowany przez aplikację. Pierwszy czynnik jest dobrze znany, natomiast drugi czynnik można określić na podstawie posiadanych wiadomości o aplikacjach, które w przyszłości będą działały na projektowanym systemie obiektowych baz danych. Lepszym sposobem byłoby gromadzenie informacji w jaki sposób przedstawia się przetwarzanie obiektów podczas funkcjonowania systemu. Na podstawie tego typu informacji możliwe jest wygenerowanie statystykostępów do poszczególnych obiektów, a także statystyk ukazujących dostęp do obiektów, pod

warunkiem, że został odczytany inny obiekt powiązany z tymi obiektami zależnościami. Generowanie tego typu statystyk oraz gromadzenie danych niezbędnych do ich generowania może obniżać wydajność systemu. Należy więc wybrać rozwiązanie pośrednie, najbardziej efektywne.

Podczas wyboru strategii grupowania należy wziąć pod uwagę istnienie obiektów złożonych, zależności hierarchiczne dziedziczenia, agregację obiektów, istnienie metod. Strukturę hierarchii dziedziczenia można przedstawić za pomocą drzewa w przypadku dziedziczenia prostego lub grafu skierowanego bez cykli w przypadku dziedziczenia wielokrotnego. Dla zobrazowania hierarchii agregacji i obiektów złożonych grafy mogą mieć cykle. Operacje na tego typu strukturach polegają na przeglądaniu grafu lub drzewa. Wykonywane jest to poprzez odczyt identyfikatorów obiektów i znajdowanie poprzedników lub następników danego węzła. Dobrze dobrana strategia grupowania obiektów powinna zapewnić, by węzły były umieszczone w porządku liniowym. Można to zrealizować umieszczając węzły – obiekty w taki sposób, aby zaraz za nimi następowały wszystkie węzły potomne. Ta strategia jest efektywna, gdy po odczytaniu danego obiektu potrzebne jest odczytanie wszystkich obiektów potomnych [Klo98].

Wśród metod grupowania obiektów można wyodrębnić dwie najbardziej ogólne: **statyczną** i **dynamiczną**. Pierwsza z nich jest to **metoda statyczna**, oparta na schemacie bazy danych. Schemat bazy danych definiuje związki strukturalne między klasami obiektów, w związku z tym obiekty takich klas znajdują się w pewnej zależności od siebie i najprawdopodobniej mogą być wykorzystywane wspólnie. W pracy [Kim96] proponującej metodę statyczną wyróżniono pięć podstawowych strategii grupowania obiektów w obiektowej bazie danych:

- według przynależności obiektów do klasy. Jednym z wariantów tego rozwiązania jest umieszczenie w każdym gronie tylko jednego obiektu. Ta strategia może zostać wykorzystana w bazach danych, w których stosowane są bardzo duże obiekty, nie powiązane z innymi obiektami, do których z reguły odwołuje się pojedynczo i których kopiowanie z pamięci stałej do pamięci głównej jest z reguły kosztowne.
- według pewnych wspólnych cech obiektów, np. wartości danego atrybutu. Ta strategia ułatwia znajdowanie obiektów o cechach określonych w zapytaniu. W tej strategii w gronie można umieścić obiekty należące do różnych klas.
- według hierarchii agregacji. Ta strategia wykorzystywana jest w przypadku, gdy z danego obiektu będziemy odwoływać się do innych obiektów powiązanych z nim poprzez agregację. W tym przypadku istnieje duże prawdopodobieństwo, że zagregowane obiekty będą przetwarzane razem.
- według hierarchii dziedziczenia. W tej sytuacji grupowane są razem wszystkie obiekty należące do hierarchii dziedziczenia danej klasy. Ta strategia jest efektywna dla zapytań dotyczących obiektów należących do wszystkich klas – potomków danej klasy.

- według hierarchii agregacji i hierarchii dziedziczenia. Strategia polega na znalezieniu kompromisowego rozwiązania łączącego strategie przedstawione w dwóch powyższych punktach. Zależności między obiektami można przedstawić w formie grafu skierowanego, tak, aby początkowym węzłem była pewna klasa, która stanowi początek hierarchii agregacji lub hierarchii dziedziczenia.

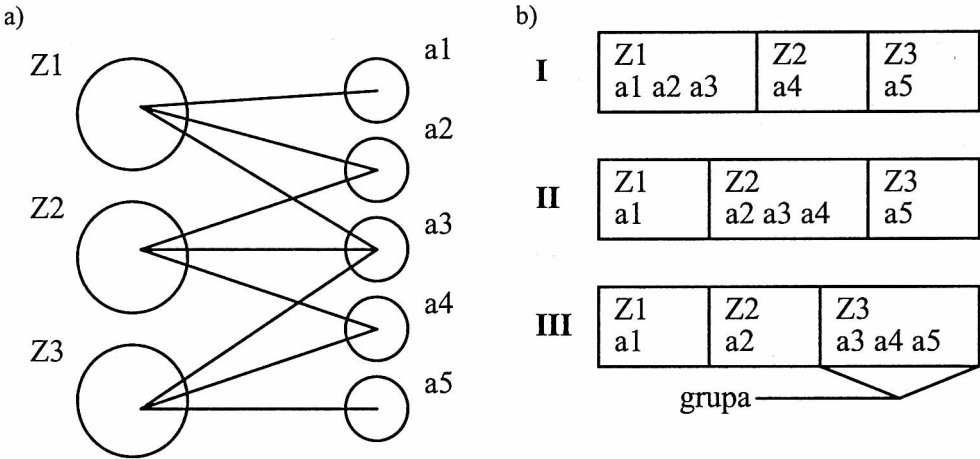
Przedstawione powyżej strategie są strategiami statycznymi, co oznacza, że grona nie mogą być redefiniowane w trakcie działania systemu.

Każda strategia grupowania obiektów w grona powinna umożliwiać wykonanie podstawowych operacji związanych z obiektami, takich jak:

- dodawanie obiektów do już istniejących gron, usuwanie obiektów;
- przenoszenie obiektów z jednych gron do innych;
- ponowną generację gron, w przypadku, gdy zmiany w bazie wpłynęły na kryteria grupowania i niezbędne jest ponowne grupowanie obiektów.

W trakcie doboru strategii grupowania należy uwzględnić fakt, że w różnych fazach projektowania obiektowej bazy danych narzędzia projektowania mogą mieć różne wzorce dostępu do danych. W takim przypadku optymalnym rozwiązaniem byłoby użycie różnych strategii grupowania obiektów w grona dla każdego etapu projektowania. Może się również okazać, że różne aplikacje wykorzystujące określoną bazę danych, a nawet poszczególni użytkownicy wykorzystujący jednocześnie daną aplikację mogą używać różnych wzorców dostępu do tych samych obiektów. Tak więc optymalny wybór strategii grupowania dla konkretnej aplikacji lub grupy użytkowników może w drastyczny sposób obniżyć efektywność innych aplikacji czy też grup użytkowników.

Opisana metoda statyczna podziału obiektów na grona nie zawsze jest wystarczająca. Przede wszystkim nie uwzględnia ona związków logicznych, występujących między obiektami, a nie uwzględnionych w schemacie bazy danych. Związki takie może zdefiniować użytkownik w trakcie eksploatacji bazy danych, np. mogą to być obiekty tej samej klasy mające tę samą wartość danego atrybutu, lub mogą to być obiekty różnych klas stanowiące części jednego urządzenia. Poza tym użycie tylko metody statycznej często prowadzi do powstania pewnej nieokreśloności przy grupowaniu obiektów. Tak, na przykład (rys. 1), kilka obiektów złożonych Z1, Z2, Z3 zawiera wspólne elementy ze zbioru obiektów a1 ... a5. W tej sytuacji trudno jednoznacznie wskazać metodę grupowania tylko ze względu na związek agregacji. W praktyce ewolucja schematu powoduje często unieważnienie istniejącej metody grupowania. Po czwarte, metoda statyczna nie uwzględnia indywidualnego zachowania obiektów. Obiekty tej samej klasy, stanowiące listę, lub składowe obiektu złożonego, mogą różnić się częstotliwością występowania w operacjach, a więc umieszczenie ich w jednym gronie nie zawsze jest efektywne [Fig96].



Rys. 1. Grupowanie obiektów złożonych a) fragment schematu bazy danych,
b) trzy z możliwych metod grupowania obiektów w oparciu o związek agregacji
Fig. 1. Complex objects clustering a) a part of a database schema,
b) three possible methods of aggregation object clustering

W takiej sytuacji pożądane wyniki może dać **metoda dynamiczna**. Pozwala ona wyznaczyć grupę obiektów w oparciu o statystykę operacji w bazie danych. Poprzez rejestrację operacji w bazie danych podczas jej działania uzyskujemy informacje jakie obiekty i ile razy zostały wykorzystane wspólnie w określonym przedziale czasu. Istnieje duże prawdopodobieństwo, że obiekty z większą liczbą wystąpień będą równie często wykorzystane w następnych operacjach. Dobre efekty daje więc grupowanie takich obiektów razem. Grupowanie metodą dynamiczną odbywa się w trakcie działania bazy danych.

2.2. ANALIZA ISTNIEJĄCYCH ROZWIĄZAŃ

Większość systemów obiektowych baz danych oferowanych obecnie na rynku wykorzystuje statyczne strategie grupowania obiektów w grona [Klo98]. Odpowiedzialność za umieszczenie obiektu we właściwym gronie spoczywa na użytkowniku, ponieważ nie zastosowano jeszcze mechanizmu automatycznego umieszczania obiektów w gronach. W większości przypadków nie ma możliwości ponownej reorganizacji gron, to jest stworzenia na nowo gron z istniejących już obiektów.

Przykładowo w systemie ObjectStore obiekty grupowane są bądź w segmentach, które są obiektami o stosunkowo dużych rozmiarach, bądź też w mniejszych obiektach – gronach. W pierwszej kolejności utworzony zostaje segment, a następnie wewnątrz segmentu można utworzyć grona o ustalonych rozmiarach. Zwykle najbardziej

efektywne jest utworzenie gron o rozmiarze odpowiadającym rozmiarowi najmniejszej porcji danych odczytywanej z pamięci masowej w czasie jednej operacji wejścia/wyjścia. Po utworzeniu gron, tworząc obiekt, użytkownik decyduje, w którym gronie ma on zostać umieszczony [WWW2]. W systemie Encore/Observer użytkownik widzi segment będący jednostką do tworzenia gron. Użytkownik ma możliwość umieszczenia obiektu w dowolnym miejscu segmentu, kierując się dowolnymi kryteriami. Podczas odczytu pojedynczego obiektu, system wczytuje z dysku do pamięci jednorazowo cały segment. W systemie GemStone do segmentów mamy dostęp poprzez klasę *Segment* języka OPAL, co umożliwia użytkownikowi kontrolę nad tym, gdzie składowane są obiekty. W systemie Vbase jednostka służąca do przechowywania gron obiektów odpowiada wielkością jednostce transferu danych z dysku twardego do pamięci. Podczas tworzenia nowego obiektu użytkownik może podać obiekt, według którego nastąpi przyporządkowanie nowoutworzonego obiektu do odpowiedniego grona (nowy obiekt zostanie umieszczony w gronie, w którym znajduje się podany obiekt).

Nieco inna sytuacja ma miejsce w systemie O₂, w którym sposób, w jaki grupowane są obiekty zależy od informacji kontrolnej podanej przez administratora bazy. Informacja kontrolna, nazywana również drzewem rozmieszczenia (ang. *placement tree*), określa, w jaki sposób mają być składowane obiekty złożone i ich obiekty składowe.

Jeszcze inne podejście do problemu grupowania zaproponowali twórcy systemu ODE, w którym grona są rozszerzeniem pojęcia typu. Wszystkie obiekty trwałe, czyli te, które mają być składowane w pamięci stałej, będące tego samego typu zostają umieszczone w jednym gronie, które musi zostać uprzednio utworzone przez jawną operację. Dodatkowo użytkownik ma do dyspozycji podgrona, które mogą zostać utworzone wewnątrz grona i służą do grupowania obiektów o podobnych właściwościach (np. o określonej wartości danego atrybutu).

2.3. GRUPOWANIE OBIEKTÓW W SYSTEMIE OBJECTSTORE

Tworząc obiekt w bazie danych ObjectStore można wyszczególnić określony segment, w którym utworzony obiekt będzie przechowywany. Następnie wewnątrz danego segmentu możemy umieścić ten obiekt w konkretnym gronie (ang. *cluster*). Aby użyć grona należy go uprzednio zdefiniować, określając segment, w którym ma on zostać umieszczony. Następnie tworząc obiekt przy pomocy operatora *new()*, zamiast wskaźnika do segmentu używa się wskaźnika do grona.

Każda nowo utworzona baza danych ObjectStore zawiera domyślny segment oraz segment schematu bazy danych, jeżeli ten nie jest przechowywany poza bazą. Jeżeli

używamy podstawowej wersji operatora *new* podając jedynie wskaźnik do bazy danych, a nie do segmentu, wówczas utworzony obiekt umieszczony zostaje w segmencie domyślnym. Ten segment, podobnie jak pozostałe segmenty dostosowuje się by pomieścić nowe dane w nim przechowywane. Jeżeli chcemy stworzyć w bazie danych nowy segment, należy go stworzyć używając funkcji-członka klasy *os_database::create_segment()*.

Grona, podobnie jak segmenty są tworzone jawnie. Grono w bazie danych ObjectStore tworzone jest przy użyciu funkcji-członka klasy *os_segment::create_object_cluster()* w instancji klasy *os_segment*, w której dane grono ma zostać umieszczone.

Przykład 1:

```
os_object_cluster* cluster;
cluster = segment->create_object_cluster(size);
object* an_object = new (cluster, Object::get_os_typespec()) Object;
```

Obiekt *cluster*, czyli grono, jest częścią segmentu, wewnątrz którego mogą być zgrupowane obiekty powiązane zależnościami. Grono może mieć wielkość od rozmiaru pojedynczej strony, aż do 64 KB pamięci.

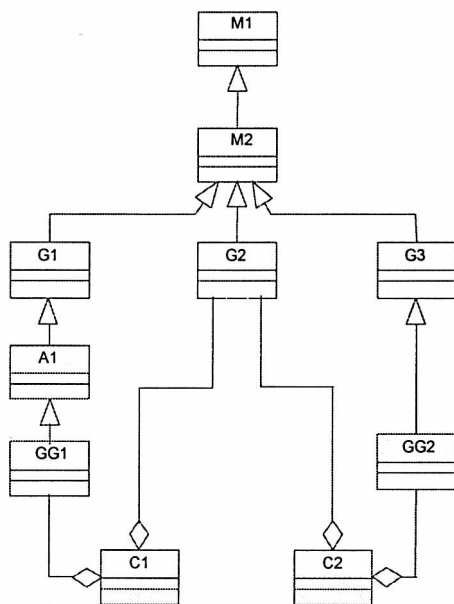
3. EKSPERYMENTY KOMPUTEROWE

3.1. CEL EKSPERYMENTÓW

Celem eksperymentów przeprowadzonych w ramach danej pracy jest pomiar efektywności grupowania obiektów w obiektowej bazie danych. Zaplanowano dwie grupy eksperymentów, z których każda ma na celu ukazanie zmian efektywności operacji wejścia/wyjścia w bazie danych przy zastosowaniu różnych kryteriów grupowania.

Pierwsza grupa eksperymentów ma na celu ukazanie wpływu kryterium grupowania na efektywność odczytu obiektów jednej klasy lub obiektów zagregowanych. Druga grupa eksperymentów ma ukazać wpływ różnych kryteriów grupowania na operacje odczytu i zapisu.

3.2. EKSPERYMENTALNA BAZA DANYCH



Rys. 2. Schemat eksperymentalnej bazy danych

Fig. 2. Prototype database schema

Na potrzeby eksperymentów komputerowych zaprojektowana została testowa baza danych z 10 klas, zawierająca zarówno związki generalizacji, jak i agregacji. Wykorzystano tak prosty schemat bazy danych aby umożliwić łatwiejszą interpretację wyników eksperymentów. Struktura bazy nie jest również w żaden sposób powiązana z celem eksperymentu, aby nie sugerować wyników.

Baza danych przygotowana do przeprowadzenia na niej eksperymentów wypełniona została liczbą obiektów poszczególnych klas przedstawioną w tabeli 1.

Przedstawiony został również rozmiar obiektów poszczególnych klas.

Tabela 1. Populacja testowej bazy danych

Klasa	E_o	S_o [B]	R_o [KB]	(¹) w tabeli uwzględniono minimalny rozmiar instancji klas C1 i C2, w przypadku, gdy nie stworzono żadnych obiektów agregowanych przez klasy C1 i C2. W bazach testowych każdy obiekt klasy C1 agreguje po 250 obiektów klas GG1 i G2, a każdy obiekt klasy C2 po 250 obiektów klas GG2 i G2: E_o – liczba obiektów, S_o – rozmiar obiektu [B], R_o – rozmiar $E_o * S_o$ [KB]
M1	10 000	128	1 250	
M2	10 000	256	2 500	
G1	10 000	384	3 750	
G2	20 000	384	7 500	
G3	10 000	384	3 750	
B1	10 000	512	5 000	
B2	10 000	512	5 000	
C1	40	48 ⁽¹⁾	1,875	
C2	40	48 ⁽¹⁾	1,875	

3.3. OPIS ORAZ ZASADY PRZEPROWADZENIA EKSPERYMENTÓW

Dla każdego eksperymentu powstały trzy bazy danych zawierające dokładnie taką samą liczbę obiektów każdej klasy. Pierwsza baza zawiera obiekty nie zgrupowane, druga baza zawiera obiekty zgrupowane wg kryterium przynależności do klasy, natomiast trzecia baza – obiekty zgrupowane wg agregacji. Łączna liczba obiektów utworzonych przez użytkownika w każdej z baz testowych to 80080. Podczas wypełniania baz obiektami utworzone zostały korzenie do kolekcji zawierających wszystkie obiekty poszczególnych klas. Dodatkowo w bazie zawierającej obiekty zgrupowane wg agregacji utworzone zostały korzenie do każdego obiektu agregującego, tj. do każdego obiektu klas: C1 i C2.

Poszczególne scenariusze eksperymentów wykonane zostały kolejno dla każdej z baz danych z zachowaniem jednakowych warunków. Następnie porównane zostały wyniki uzyskane dla bazy nie zgrupowanej i dla baz zgrupowanych, a także wyniki uzyskane z zależności od przyjętego kryterium grupowania i od rodzaju odczytywanych danych.

Wyniki pomiarów podzielone zostały na dwie grupy: wyniki pomiarów typu „cold-transaction”, wykonywane podczas pierwszego odczytu obiektów z bazy, oraz wyniki pomiarów typu „hot-transaction”, stanowiące średnią pozostałych pomiarów.

Eksperymenty przeprowadzone zostały na komputerze klasy PC w następującej konfiguracji:

Processor:	Intel Celeron 400A
Płyta główna:	MSI BX MASTER
Pamięć RAM:	320 MB
Karta graficzna:	Ge-Force II GTS 32MB
Dysk twardy:	FUJITSU MPF3153AT

Testowe bazy danych tworzone były na czystej, sformatowanej partycji o rozmiarze 1.096.109.248 B. Partycja sformatowana została w systemie FAT32, rozmiar strony na dysku wynosi 4KB. Po przeprowadzeniu eksperymentów na danym egzemplarzu bazy danych był on usuwany i kolejna baza tworzona była na pustej partycji.

Wszystkie eksperymenty przeprowadzone zostały pod systemem MS Windows 98 SE. Aplikacja tworząca bazy danych i realizująca eksperymenty napisana została w systemie ObjectStore Release 5.0, na platformie MS Visual C++ 5.0 Enterprise Edition. Schemat bazy utworzony został przy pomocy programu ObjectStore Database Designer 1.0 Beta 3. Do odczytu zawartości stworzonych baz danych wykorzystano aplikację ObjectStore Inspector 2.5 Evaluation Edition.

Podczas realizacji wszystkich eksperymentów zmierzone zostały czasy odczytu lub aktualizacji obiektów, obejmujące czasy wyszukania i odczytu wartości niezbędnych korzeni oraz czasy realizacji samych odczytów lub aktualizacji. Pominięte zostały natomiast czasy otwierania i zamykania bazy danych, oraz czasy inicjacji transakcji.

Jako kryterium efektywności grupowania obiektów wybrano stosunek rozmiaru odczytywanych obiektów do czasu transakcji:

$$W = \frac{\sum_{i=1}^n (\text{rozmiar_obektu}_i)}{\sum_{i=1}^n (\text{czas_odczytu_obektu}_i)} \quad (1)$$

3.4. EKSPERYMENT NR 1

SKUTECZNOŚĆ GRUPOWANIA OBIEKTÓW PRZY OPERACJACH ODCZYTU I ZAPISU (READ_ONLY/UPDATE)

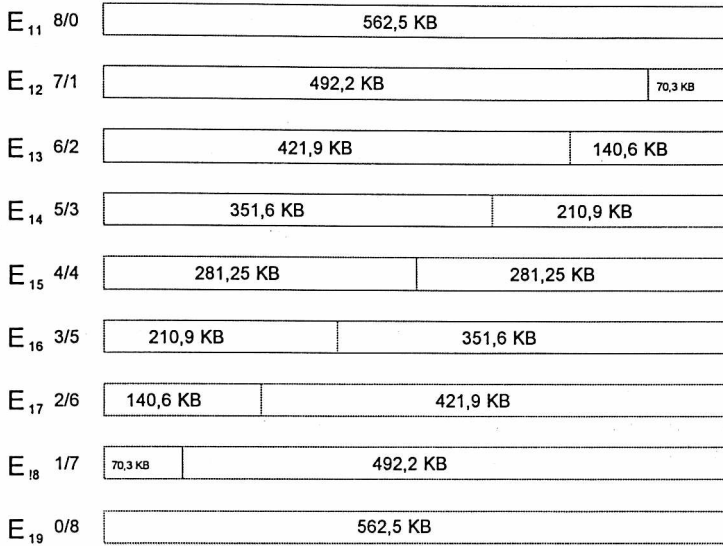
Celem eksperymentu jest ukazanie wpływu operacji aktualizacji w bazie danych na efektywność grupowania obiektów w pamięci stałej.

Hipoteza wstępna: zwiększenie liczby operacji aktualizacji na bazie danych zmniejsza efektywność grupowania obiektów w pamięci stałej.

3.4.1. OPIS SCENARIUSZA

Wykonanie eksperymentu polega na przeprowadzeniu dziewięciu doświadczeń na trzech testowych bazach. Każde z dziewięciu doświadczeń różni się od siebie stosunkiem rozmiaru odczytywanych obiektów (read_only) i rozmiaru obiektów aktualizowanych (update). Zmierzone zostały czasy odczytu i aktualizacji obiektów klas: M1, M2, G1, G2, G3, GG1 i GG2.

Operacja odczytu obiektów polega na pobraniu z dysku twardego do pamięci operacyjnej zadanej liczby obiektów (patrz tabela 2). Realizowana jest w następujących krokach: najpierw odszukany zostaje korzeń wskazujący na kolekcję obiektów danej klasy, następnie odczytana zostaje jego wartość i tworzony jest kursor wskazujący na kolejne obiekty kolekcji.



Rys. 3. Stosunki rozmiaru danych wykorzystanych w eksperymencie nr 1
Fig. 3. Amount of data used in experiment no. 1

Przy wykorzystaniu kursora pobierane są do pamięci operacyjnej kolejne obiekty kolekcji. W przypadku operacji aktualizacji sytuacja wygląda podobnie, z tą jednak różnicą, że po pobraniu kolejnych obiektów do pamięci operacyjnej zmieniane są wartości ich atrybutów. Transakcje odczytu mają atrybut *read_only*, a transakcje aktualizacji – atrybut *update*.

Tabela 2

Typy transakcji dla eksperymentu nr 1 z uwzględnieniem liczby obiektów dla transakcji *read_only*/*update* (doświadczenia E₁–E₉)

Typ	ID _{transakcji}	Klasy	E ₁₁	E ₁₂	E ₁₃	E ₁₄	E ₁₅	E ₁₆	E ₁₇	E ₁₈	E ₁₉
Odczyt obiektów tej samej klasy (<i>read_only</i>)	T ₁₁	M1	500	437	375	313	250	187	125	63	0
	T ₁₂	M2	500	437	375	313	250	187	125	63	0
	T ₁₃	G1	100	87	75	63	50	37	25	13	0
	T ₁₄	G2	200	175	150	125	100	75	50	25	0
	T ₁₅	G3	300	262	225	188	150	112	75	38	0
	T ₁₆	GG1	100	87	75	63	50	37	25	13	0
	T ₁₇	GG2	200	175	150	125	100	75	50	25	0
Aktualizacja obiektów tej samej klasy (<i>update</i>)	T ₂₁	M1	0	63	125	187	250	313	375	437	500
	T ₂₂	M2	0	63	125	187	250	313	375	437	500
	T ₂₃	G1	0	13	25	37	50	63	75	87	100
	T ₂₄	G2	0	25	50	75	100	125	150	175	200
	T ₂₅	G3	0	38	75	112	150	188	225	262	300
	T ₂₆	GG1	0	13	25	37	50	63	75	87	100
	T ₂₇	GG2	0	25	50	75	100	125	150	175	200

Szczegółowe dane na temat ilości obiektów poszczególnych klas odczytywanych lub aktualizowanych w kolejnych doświadczeniach przedstawia tabela 2. Dane na temat rozmiaru obiektów poszczególnych klas odczytywanych lub aktualizowanych w kolejnych doświadczeniach przedstawione zostały w tabeli 3.

Stosunek rozmiaru danych odczytywanych i rozmiaru danych aktualizowanych dla kolejnych doświadczeń zmienia się w sposób ukazany na rysunku 2.

Transakcje przedstawione w tabelach 2 i 3 połączone zostały w scenariusz S_1 przedstawiony niżej:

$$S_1: (T_{11}+T_{12}+T_{13}+T_{14}+T_{15}+T_{16}+T_{17}+T_{21}+T_{22}+T_{23}+T_{24}+T_{25}+T_{26}+T_{27})*10$$

Łączna liczba obiektów odczytanych w ramach eksperymentu to 19 000, a łączny rozmiar obiektów to 5625 KB dla każdej testowej bazy danych.

Tabela 3

Typy transakcji dla eksperymentu nr 1 z uwzględnieniem liczby obiektów dla transakcji read_only/update (doświadczenia E_{11} – E_{19})

Typ	ID _{transakcji}	Klasy	E_{11}	E_{12}	E_{13}	E_{14}	E_{15}	E_{16}	E_{17}	E_{18}	E_{19}
Odczyt obiektów tej samej klasy (read_only)	T_{11}	M1	62,5	54,6	46,8	39,1	31,2	23,3	15,6	7,87	0
	T_{12}	M2	125	109,	93,7	78,2	62,5	46,7	31,2	15,7	0
	T_{13}	G1	37,5	32,6	28,1	23,6	18,7	13,8	9,37	4,87	0
	T_{14}	G2	75	65,6	56,2	46,8	37,5	28,1	18,7	9,37	0
	T_{15}	G3	112,	98,2	84,3	70,5	56,2	42	28,1	14,2	0
	T_{16}	GG1	50	43,5	37,5	31,5	25	18,5	12,5	6,5	0
	T_{17}	GG2	100	87,5	75	62,5	50	37,5	25	12,5	0
Aktualizacja obiektów tej samej klasy (update)	T_{21}	M1	0	7,87	15,6	23,3	31,2	39,1	46,8	54,6	62,5
	T_{22}	M2	0	15,7	31,2	46,7	62,5	78,2	93,7	109,	125
	T_{23}	G1	0	4,87	9,37	13,8	18,7	23,6	28,1	32,6	37,5
	T_{24}	G2	0	9,37	18,7	28,1	37,5	46,8	56,2	65,6	75
	T_{25}	G3	0	14,2	28,1	42	56,2	70,5	84,3	98,2	112,
	T_{26}	GG1	0	6,5	12,5	18,5	25	31,5	37,5	43,5	50
	T_{27}	GG2	0	12,5	25	37,5	50	62,5	75	87,5	100

3.4.2. WYNIKI EKSPERYMENTU

Wyniki przedstawione zostały w postaci wykresu i tabeli wyników dla każdej grupy pomiarów (transakcje „cold-transaction” – wykres nr 1 i tabela nr 4 oraz transakcje „hot-transaction” – wykres nr 2 i tabela nr 5).

Tabela 4

Wyniki pomiaru „cold-transaction” kryterium wydajności grupowania dla eksperymentu nr 1 (stosunek operacji read_only do operacji update).

Warunki dla scenariusza S_1 : $S_1(E_x)$	Kryterium wydajności grupowania [KB/s] dla pomiaru „cold-transaction”		
	Baza 1°	Baza 2°	Baza 3°
$S_1(E_{11})$	13395,349	15157,895	14048,780
$S_1(E_{12})$	507,489	539,831	519,387
$S_1(E_{13})$	267,038	289,884	271,186
$S_1(E_{14})$	194,858	217,441	208,318
$S_1(E_{15})$	126,260	134,958	128,946
$S_1(E_{16})$	114,467	118,935	115,965
$S_1(E_{17})$	103,821	109,132	107,183
$S_1(E_{18})$	90,637	96,128	93,567
$S_1(E_{19})$	79,503	88,903	86,538

Wykres 1

Wyniki pomiaru „cold-transaction” kryterium wydajności grupowania dla eksperymentu nr 1 (stosunek operacji read_only do operacji update)

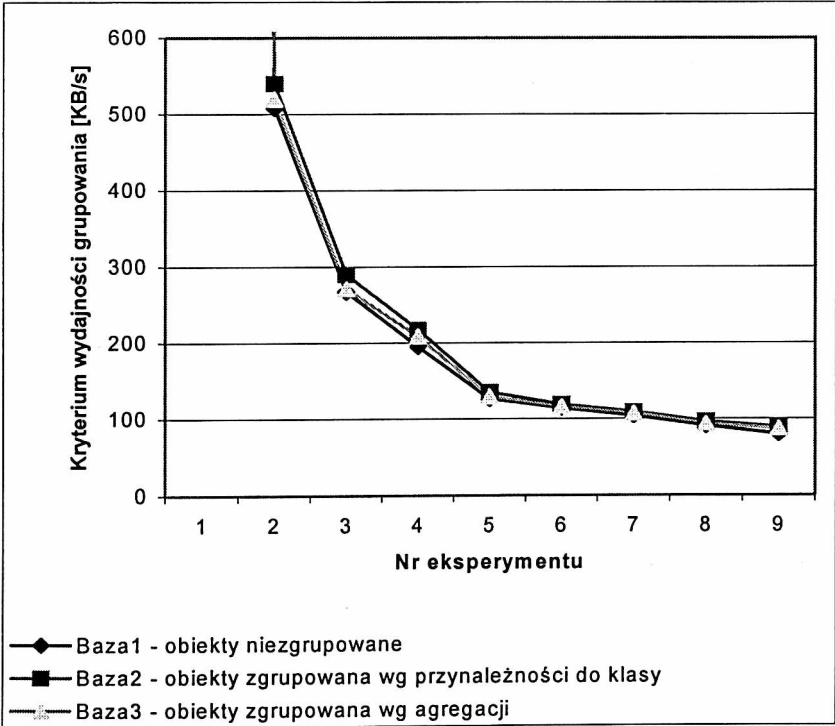


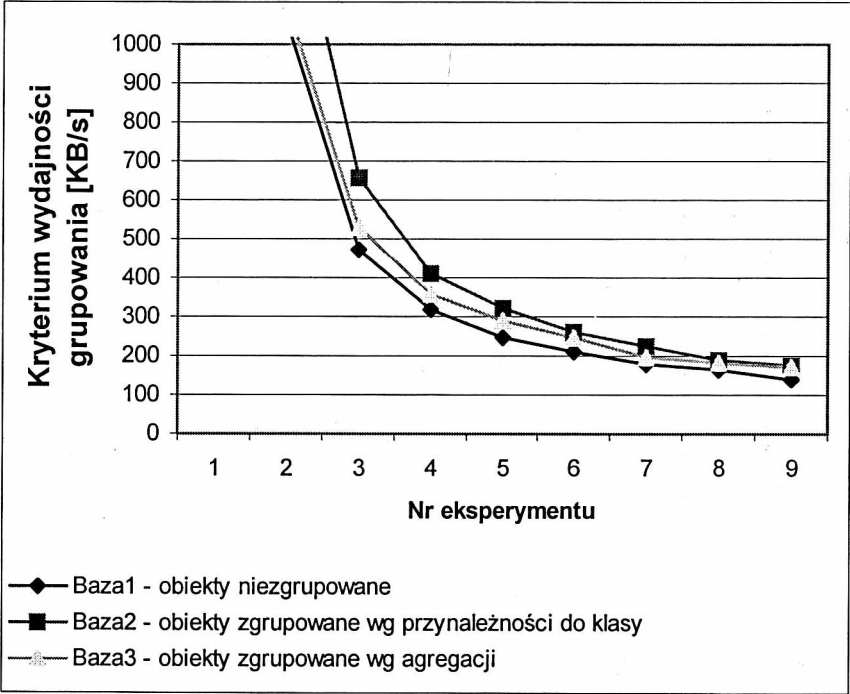
Tabela 5

Wyniki pomiaru „hot-transaction“ kryterium wydajności grupowania dla eksperymentu nr 1 (stosunek operacji read_only do operacji update)

Warunki dla scenariusza S ₁ : S ₁ (E _x)	Kryterium wydajności grupowania [KB/s] dla pomiaru „hot-transaction“		
	Baza 1°	Baza 2°	Baza 3°
S ₁ (E ₁₁)	24685,714	26181,818	22938,053
S ₁ (E ₁₂)	1054,516	1441,602	1095,057
S ₁ (E ₁₃)	470,034	655,373	528,548
S ₁ (E ₁₄)	316,812	411,788	356,044
S ₁ (E ₁₅)	244,759	319,901	289,593
S ₁ (E ₁₆)	212,459	259,291	246,622
S ₁ (E ₁₇)	177,098	224,028	194,748
S ₁ (E ₁₈)	163,595	188,181	180,589
S ₁ (E ₁₉)	139,768	175,944	169,928

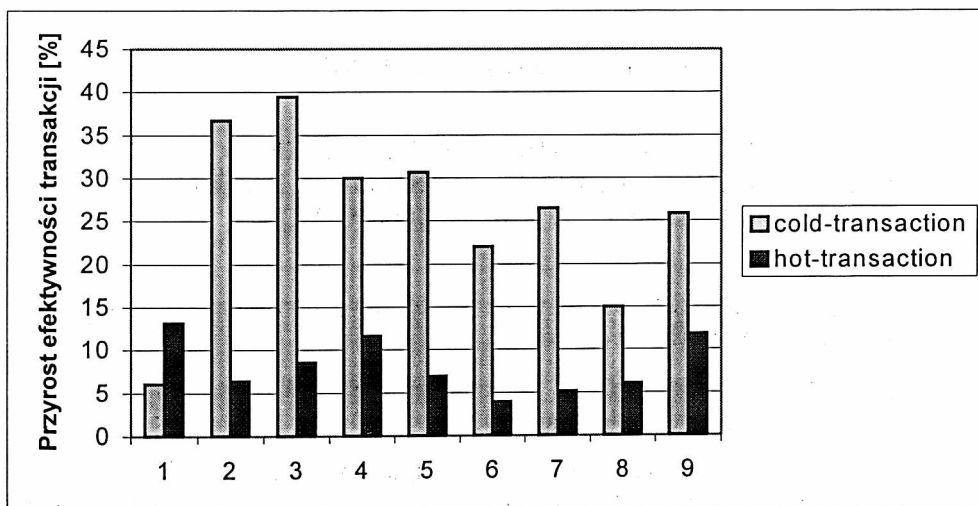
Wykres 2

Wyniki pomiaru „hot-transaction“ kryterium wydajności grupowania dla eksperymentu nr 1 (stosunek operacji read_only do operacji update)

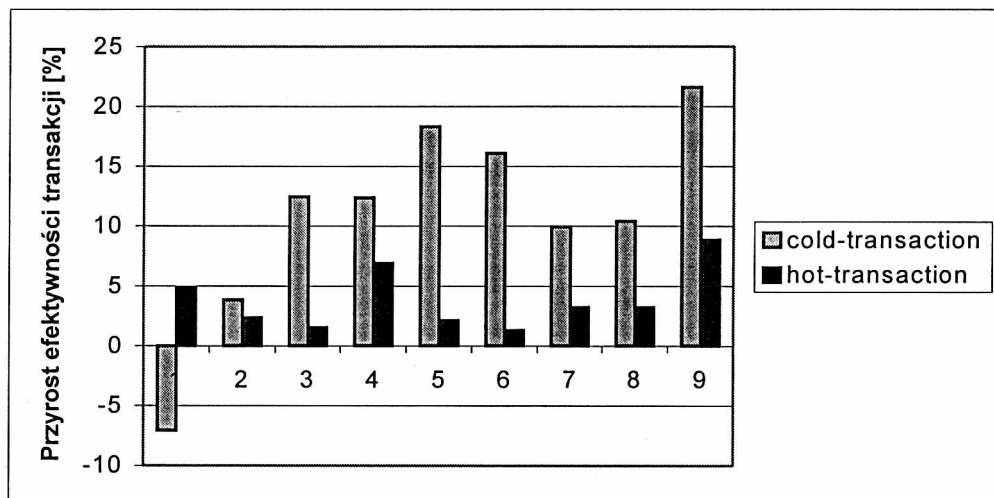


3.4.3. WNIOSKI

Wyniki eksperymentu nr 1 pokazują realny wzrost efektywności operacji na zgrupowanej bazie danych w stosunku do efektywności operacji na bazie nie zgrupowanej. Widoczna jest również przewaga bazy zgrupowanej wg kryterium przynależności do klasy nad bazą zgrupowaną wg agregacji, co zgodne jest z przewidywaniami, ponieważ wszystkie transakcje scenariusza dokonują odczytów i zapisów na kolekcjach obiektów kolejnych klas z pominięciem klas agregujących. Analizując te dwie grupy wyników widać wyraźne zwiększenie efektywności grupowania dla transakcji typu „cold-transaction” w porównaniu z bazą nie zgrupowaną. W przypadku bazy zgrupowanej wg kryterium przynależności obiektów do klasy można zauważyć większy przyrost efektywności dla operacji odczytu niż dla operacji aktualizacji. W przypadku bazy zgrupowanej wg agregacji widać sytuację odwrotną. Zauważyć należy również, że dla doświadczenia pierwszego (tylko operacje odczytu) efektywność transakcji dla bazy zgrupowanej wg agregacji była mniejsza niż dla bazy nie zgrupowanej. Oznacza to, że przy niewłaściwym doborze kryterium grupowania do charakteru przeprowadzanych transakcji zastosowanie grupowania może zmniejszyć efektywność transakcji.



Rys. 4. Przyrost [%] efektywności transakcji dla bazy zgrupowanej wg przynależności obiektów do klasy
 Fig. 4. Increment [%] of transactions efficiency in clustered database (objects belong to the same class)



Rys. 5. Przyrost [%] efektywności transakcji dla bazy zgrupowanej wg agregacji
 Fig. 5. Increment [%] of transactions efficiency
 in clustered database (objects belong to the aggregation)

3.5. EKSPERYMENT NR 2

SKUTECZNOŚĆ GRUPOWANIA OBIEKTÓW PRZY OPERACJACH ODCZYTU OBIEKTÓW DANEJ KLASY I OBIEKTÓW ZAGREGOWANYCH

Celem eksperymentu jest ukazanie, iż wybór kryterium grupowania zależy od rodzaju wykonywanych operacji w bazie danych

Hipoteza wstępna: zmiana rodzaju operacji przy stałym kryterium grupowania pogarsza efektywność grupowania obiektów.

Powyższa hipoteza oznacza, że kryterium grupowania należy dobierać w oparciu o analizę operacji pobierających obiekty z pamięci stałej, a nie tylko w oparciu o analizę związków strukturalnych w schemacie bazy danych.

3.5.1. OPIS SCENARIUSZA

Wykonanie eksperymentu polega na przeprowadzeniu dziewięciu doświadczeń na trzech testowych bazach. Każde z dziewięciu doświadczeń różni się od siebie stosun-

kiem rozmiaru odczytywanych obiektów przynależnych do danej klasy i rozmiaru odczytywanych obiektów zagregowanych. Stosunek rozmiaru odczytywanych obiektów przynależnych do danej klasy i rozmiaru odczytywanych obiektów zagregowanych dla kolejnych doświadczeń zmienia się w sposób przedstawiony na rys. 6.

W ramach eksperymentu odczytane zostaną obiekty następujących klas: G2, GG1, GG2 oraz obiekty agregujące klas: C1 i C2. Wraz z obiektem agregującym odczytane zostają wszystkie zagregowane przez niego obiekty. W przypadku obiektu klasy C1 wraz z nim odczytane zostanie 250 obiektów klasy GG2 i 250 obiektów klasy G2, a w przypadku obiektu klasy C2 – 250 obiektów klasy GG1 i 250 obiektów klasy G2. Odczyt obiektów przynależnych do danej klasy realizowany jest w następujący sposób: najpierw odszukany zostaje korzeń wskazujący na kolekcję obiektów danej klasy, następnie odczytana zostaje jego wartość i tworzony jest kursor wskazujący na kolejne obiekty kolekcji, potem przy wykorzystaniu kursora odczytywane są kolejne obiekty kolekcji. Odczyt obiektów zagregowanych natomiast realizowany jest w następujący sposób: odszukany zostaje korzeń wskazujący na konkretny obiekt agregujący i pobrana zostaje jego wartość, następnie dla każdego z dwóch atrybutów tego obiektu tworzony jest kursor wskazujący na kolejne obiekty kolekcji obiektów zagregowanych i przy jego pomocy są odczytywane wszystkie zagregowane obiekty.

E_{21} 8/0	8750 KB	
E_{22} 7/1	7656,25 KB	1093,75
E_{23} 6/2	6562,5 KB	2187,5 KB
E_{24} 5/3	5468,75 KB	3281,25 KB
E_{25} 4/4	4375 KB	4375 KB
E_{26} 3/5	3281,25 KB	5468,75 KB
E_{27} 2/6	2187,5 KB	6562,5 KB
E_{28} 1/7	1093,75	7656,25 KB
E_{29} 0/8	8750 KB	

Rys. 6. Stosunki rozmiaru danych wykorzystanych w eksperymencie nr 2
Fig. 6. Amount of data used in experiment no. 2

Szczegółowe dane na temat ilości obiektów poszczególnych klas odczytywanych w kolejnych doświadczeniach przedstawia tabela nr 6. Dane na temat rozmiaru obiektów poszczególnych klas odczytywanych w kolejnych doświadczeniach przedstawione zostały w tabeli 7.

Tabela 6

Typy transakcji dla eksperymentu nr 2 z uwzględnieniem liczby obiektów dla transakcji odczytu (read_only) (doświadczenia E₂₁–E₂₉)

Typ transakcji	ID transakcji	Klasy	E ₂₁	E ₂₂	E ₂₃	E ₂₄	E ₂₅	E ₂₆	E ₂₇	E ₂₈	E ₂₉
Odczyt obiektów tej samej klasy	T ₃₁	C1	0	2	5	8	10	13	15	18	20
	T ₃₂	C2	0	3	5	7	10	12	15	17	20
Odczyt obiektów zagregowanych	T ₄₁	G2	10000	8750	7500	6250	5000	3750	2500	1250	0
	T ₄₂	GG1	5000	4375	3750	3125	2500	1875	1250	625	0
	T ₄₃	GG2	5000	4375	3750	3125	2500	1875	1250	625	0

Tabela 7

Typy transakcji dla eksperymentu nr 2 z uwzględnieniem rozmiaru obiektów dla transakcji odczytu (read_only) (doświadczenia E₂₁–E₂₉)

Typ transakcji	ID transakcji	Klasy	E ₂₁	E ₂₂	E ₂₃	E ₂₄	E ₂₅	E ₂₆	E ₂₇	E ₂₈	E ₂₉
Odczyt obiektów tej samej klasy	T ₃₁	C1	0	437,5	1093,75	1750	2187,5	2843,75	3281,25	3937,5	4375
	T ₃₂	C2	0	656,25	1093,75	1531,5	2187,5	2625	3281,25	3718,75	4375
Odczyt obiektów zagregowanych	T ₄₁	G2	3750	3281,25	2812,5	2343,75	1875	1406,25	937,5	468,75	0
	T ₄₂	GG1	2500	2187,5	1875	1562,5	1250	937,5	625	312,5	0
	T ₄₃	GG2	2500	2187,5	1875	1562,5	1250	937,5	625	312,5	0

Transakcje przedstawione w tabelach 6 i 7 połączone zostały w scenariusz S₂:

$$S_2: (T_{31}+T_{32}+ T_{41}+T_{42}+ T_{43})*10$$

Łączna liczba obiektów odczytanych w ramach eksperymentu to 200.000, natomiast łączny rozmiar obiektów to 87 500 KB dla każdej testowej bazy danych.

3.5.2. WYNIKI EKSPERYMENTU

Wyniki przedstawione zostały w postaci wykresu i tabeli wyników dla każdej grupy pomiarów (transakcje „cold-transaction” – wykres 3 i tabela 8 oraz transakcje „hot-transaction” – wykres 4 i tabela 9).

Tabela 8

Wyniki pomiaru „cold-transaction” kryterium wydajności grupowania dla eksperymentu nr 2

Warunki dla scenariusza S_2 : $S_2(E_X)$	Kryterium wydajności grupowania [KB/s] dla pomiaru „cold-transaction”		
	Baza 1°	Baza 2°	Baza 3°
$S_2(E_{21})$	25,436	26,923	25,660
$S_2(E_{22})$	22,906	24,718	24,171
$S_2(E_{23})$	19,316	20,735	19,619
$S_2(E_{24})$	17,570	21,341	20,115
$S_2(E_{25})$	15,880	17,894	17,090
$S_2(E_{26})$	12,324	16,859	17,023
$S_2(E_{27})$	11,146	15,737	16,478
$S_2(E_{28})$	10,658	15,034	16,264
$S_2(E_{29})$	10,011	14,487	15,432

Wykres 3

Wyniki pomiaru „cold-transaction” kryterium wydajności grupowania dla eksperymentu nr 2

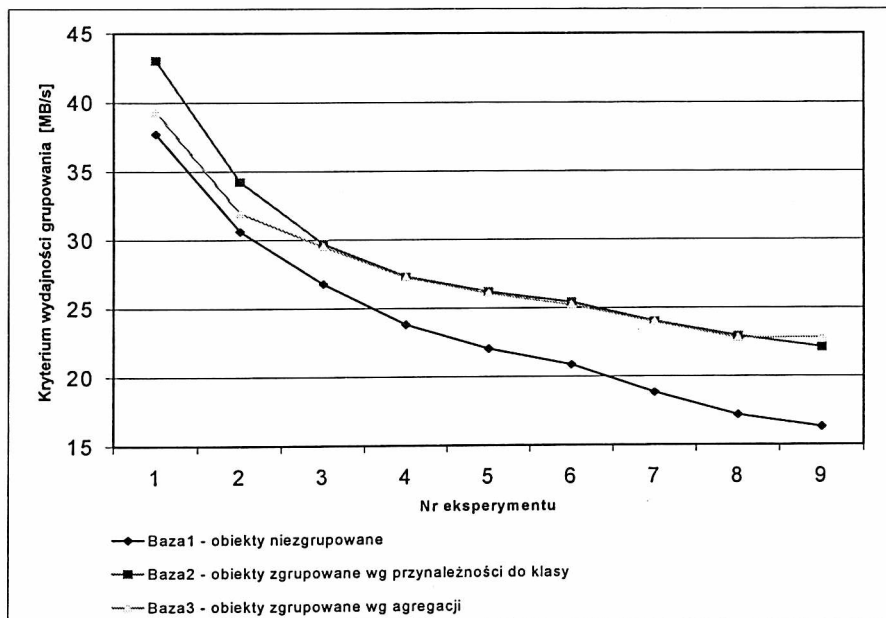


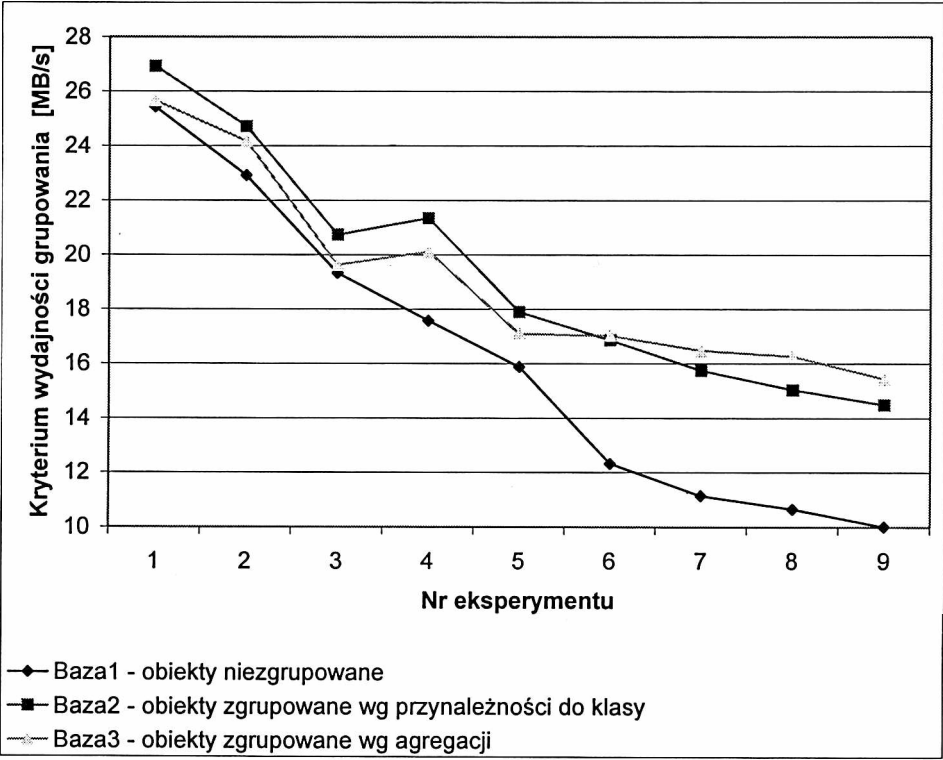
Tabela 9

Wyniki pomiaru „hot-transaction” kryterium wydajności grupowania dla eksperymentu nr 2

Warunki dla scenariusza S ₂ : S ₂ (E _x)	Kryterium wydajności grupowania [KB/s] dla pomiaru „hot-transaction”		
	Baza 1°	Baza 2°	Baza 3°
S ₂ (E ₂₁)	37,716	43,033	39,336
S ₂ (E ₂₂)	30,606	34,239	31,908
S ₂ (E ₂₃)	26,768	29,706	29,494
S ₂ (E ₂₄)	23,806	27,287	27,259
S ₂ (E ₂₅)	22,034	26,189	26,050
S ₂ (E ₂₆)	20,855	25,428	25,216
S ₂ (E ₂₇)	18,831	24,002	23,973
S ₂ (E ₂₈)	17,179	22,932	22,767
S ₂ (E ₂₉)	16,318	22,108	22,826

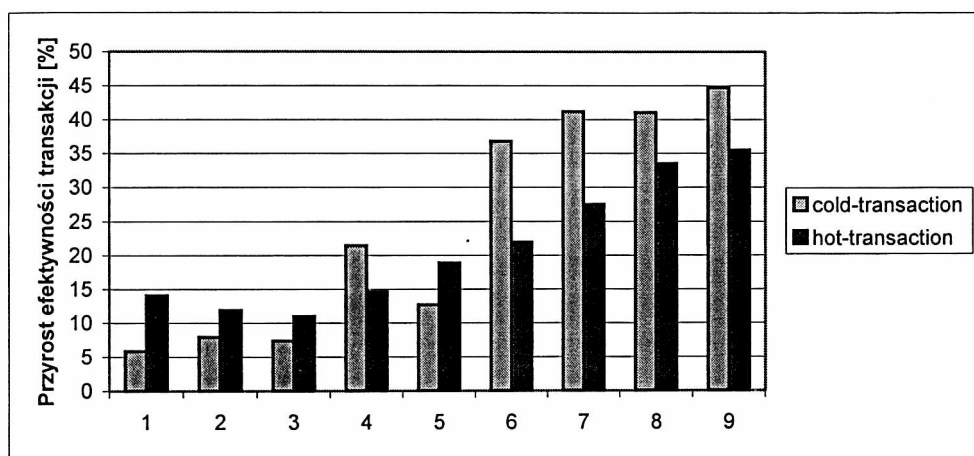
Wykres 4

Wyniki pomiaru „hot-transaction” kryterium wydajności grupowania dla eksperymentu nr 2



3.5.3. WNIOSKI

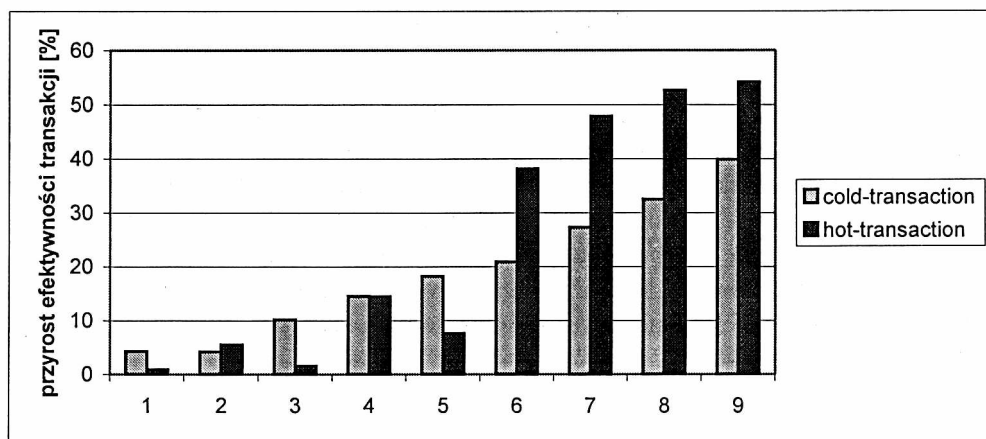
Wyniki pomiarów potwierdzają wcześniej sformułowaną hipotezę, że kryterium grupowania należy dobierać w oparciu o analizę wykonywanych w bazie danych operacji pobierania obiektów. Analizując wyniki pomiarów można zauważyć, że wraz ze wzrostem udziału operacji odczytu obiektów zagregowanych bardziej efektywne są transakcje wykonywane na bazie zawierającej obiekty zgrupowane wg agregacji. Podobnie wygląda sytuacja w bazie zgrupowanej wg kryterium przynależności obiektów do klasy – efektywność rośnie wraz ze wzrostem udziału operacji odczytu obiektów danej klasy.



Rys. 7. Przyrost [%] efektywności transakcji
dla bazy zgrupowanej wg przynależności obiektów do klasy
Fig. 7. Increment [%] of transactions efficiency
in clustered database (objects belong to the same class)

Zauważalne jest, że zarówno dla transakcji typu „cold-transaction”, jak i „hot-transaction” dużo większy procentowy wzrost efektywności następuje dla operacji odczytu obiektów zagregowanych niż obiektów przynależnych do klasy i to zarówno dla bazy zgrupowanej wg kryterium przynależności obiektów do klasy, jak i wg agregacji.

Dla operacji typu „cold-transaction” wzrost efektywności transakcji jest większy w przypadku bazy zgrupowanej wg przynależności do klasy, a dla operacji typu „hot-transaction” wzrost efektywności transakcji jest większy w przypadku bazy zgrupowanej wg przynależności do klasy.



Rys. 8. Przyrost [%] efektywności transakcji dla bazy zgrupowanej wg agregacji
 Fig. 8. Increment [%] of transactions efficiency
 in clustered database (objects belong to the aggregation)

Powyższe prawidłowości potwierdzają fakt, że zastosowanie odpowiedniego kryterium grupowania w znacznym stopniu zwiększa efektywność operacji odczytu danych z bazy. Widzimy również, że grupowanie obiektów wg przynależności do klasy daje lepsze efekty niż grupowanie wg agregacji, ponieważ przy stosunku rozmiaru odczytywanych obiektów zagregowanych do rozmiaru obiektów klasowych 1:1 lepsze wyniki uzyskujemy w bazie zgrupowanej wg przynależności do klasy.

4. PODSUMOWANIE

W pracy zbadano wpływ grupowania obiektów bazy danych w pamięci stałej na czas transakcji, a więc w konsekwencji na efektywność działania systemu z bazą danych. Ze względu na szerokość zagadnienia skupiono się na wybranych jego elementach. Analizie poddano typ kryteriów grupowania oraz rodzaj wykonywanych transakcji. Wykonano testy komputerowe dotyczące dwóch najczęściej stosowanych kryteriów grupowania: według przynależności obiektów do klasy oraz do agregacji, czyli obiektów złożonych. Wykonano również szereg testów w celu zbadania wpływu na skuteczność grupowania zmiany rodzaju transakcji w bazie danych. Zaprojektowano i zaprogramowano 18 różnych eksperymentów z wyróżnieniem dwóch stanów bazy danych: *cold* – kiedy obiekty w transakcji są odczytywane po raz pierwszy i są pobierane w większości z dysku twardego, oraz *hot* – kiedy obiekty w transakcji są pobierane z pamięci podręcznej *cache*. Wyniki eksperymentów na zgrupowanej bazie danych porównano z danymi baz nie zgrupowanych. Otrzymane rezultaty pozwalają na wysunięcie następujących wniosków:

1. Grupowanie obiektów bazy danych w pamięci stałej komputera ma znaczący wpływ na szybkość wykonywania operacji w bazie danych;
2. Zwiększenie liczby operacji aktualizacji na bazie danych zmniejsza efektywność grupowania obiektów w pamięci stałej;
3. Grupowanie według kryterium przynależności obiektów do klas wykazuje większy wzrost efektywności transakcji w porównaniu z innymi kryteriami grupowania;
4. Grupowanie według kryterium przynależności obiektów do klas wykazuje większą stabilność przy operacjach aktualizacji;
5. Zmiana rodzaju operacji w bazie danych zmniejsza efektywność przyjętego schematu grupowania obiektów;
6. Wybór strategii grupowania należy oprzeć na analizie schematu bazy danych jak również danych statystycznych przeprowadzonych transakcji.

Część praktyczna pracy została przeprowadzona w oparciu o komercyjny system zarządzania obiektowymi bazami danych ObjectStore DBMS oraz język MFC Visual C++. Takiego wyboru dokonano ze względu na następujące powody:

- system ObjectStore jest najbardziej wykorzystywanym w praktyce systemem typu OODBMS [www2];
- językiem bazy danych w systemie ObjectStore jest szeroko wykorzystywany standard języka C, co pozwala na stosowanie własnych procedur zarządzania obiektami bazy danych oraz przeprowadzenia pomiarów na potrzebę eksperymentów;
- system ObjectStore 5.0 posiada rozbudowaną bibliotekę funkcji grupowania obiektów bazy danych w pamięci komputerowej.

Ciekawym przedmiotem dalszych badań jest wpływ wielkości grona obiektów na efektywność wykonywania transakcji w bazie danych oraz powtórzenie przeprowadzonych eksperymentów w środowisku sieciowym w oparciu o rozproszoną bazę danych.

Wyniki pracy mogą zostać wykorzystane jak przez projektantów systemów obiektowych baz danych, tak i przez ich administratorów. Potwierdzone empirycznie wnioski mogą ułatwić wyznaczenie strategii grupowania obiektów bazy danych, a w związku z tym przyspieszyć wykonanie transakcji.

BIBLIOGRAFIA

- [Banc92] Bancilhon F., Delobel C., Kanellakis P., *Building an Object-Oriented Database Systems: The Story of O2*, Morgan Kaufmann Pub., 1992.
- [Fig96] Figura D., *Obiektowe bazy danych*. Akademicka Oficyna Wydawnicza PJJ, Warszawa 1996.
- [Ishi93] Ishikawa H., *Object-Oriented Database Systems: Design and Implementation for Advanced Application*, Springer Verlag, 1993.

- [Kim96] Kim W., *Wprowadzenie do obiektowych baz danych*. WNT, Warszawa 1996.
- [Klo98] Klosow A., *Mechanizmy adaptacji w obiektowo zorientowanych bazach danych*. Rozprawa doktorska, Politechnika Wrocławska, Wrocław, 1998.
- [Kuc82] Kucharczyk J., *Algorytmy analizy skupień w języku ALGOL 60*. Państwowe Wydawnictwo Naukowe, Warszawa 1982.
- [Man50] ObjectStore Manual, *ObjectStore C++ Release 5.0 Documentation*.
- [Sub98] Subieta K., *Obiektość w projektowaniu i bazach danych*. Akademicka Oficyna Wydawnicza PJJ, Warszawa 1998.
- [WWW1] <http://www.odi.com>
- [WWW2] <http://www.cetus-links.org>

THE ALLOCATION OF DATABASE OBJECTS IN EXTERNAL MEMORY WITH USAGE OF CLUSTERING MECHANISMS

Nowadays object databases are used in many computer systems. Clustering mechanisms in external memory have a significant influence on the efficiency of data processing in these systems. A knowledge about structure and behaviour of database objects is needed to make a proper choice of clustering strategy. The problem is NP-complete. In practice the choice of clustering strategy depends on experience of database administrator. In this paper a trial of showing some dependencies between effective clustering strategy and selected database characteristic is discuss. Theoretical discussion are supported with results of numerous computer simulations.

*aplikacje wielowarstwowe,
bazy danych, Internet,
Sybase PowerDesigner,
Sybase PowerBuilder*

Jacek GRUBER*
Zygmunt MAZUR*
Marek STANDZON

SYSTEM PROJEKTOWY ***Sybase PowerDesigner DataArchitect Suite 6.1*** **ORAZ ŚRODOWISKO *Sybase PowerBuilder 7 Enterprise*** **DO MODELOWANIA I GENEROWANIA** **BAZ DANYCH**

Systemy bazodanowe integrowane w sieciach intranetowych i w Internecie mają kluczowe znaczenie praktyczne w nowoczesnych systemach z zasobami informacyjnymi. Technologie projektowe i programistyczne do integracji scentralizowanych i rozproszonych baz danych w intranetach, w sieciach korporacyjnych i w Internecie należą obecnie do najnowocześniejszych i najszybciej rozwijających się technologii informatycznych. Celem pracy jest omówienie modelowania i generowania baz danych w środowisku *Sybase PowerDesigner DataArchitect Suite 6.1* oraz w środowisku *Sybase PowerBuilder 7 Enterprise*.

1. WPROWADZENIE

Systemy bazodanowe, biznesowe i systemy zarządzania, w skład których wchodzi bazy danych, aplikacje, systemy zarządzania bazami danych *DBMS* (ang. *Database Management Systems*), ewoluują w kierunku jednolitych korporacyjnych, rozproszonych środowisk systemowych i informacyjnych. Rozwiązania i technologie stosowane do budowy takich systemów uległy istotnym zmianom i obecnie do najbardziej zaawansowanych należą architektury trzywarstwowe oraz technologie aplikacji wielowarstwowych.

Metodyki modelowania i projektowania aplikacji wielowarstwowych stosowane w kilku najbardziej zaawansowanych technologicznie systemach projektowych i sy-

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

stemach *CASE* nowej generacji, dość istotnie różnią się od klasycznych metodyk projektowych i klasycznych systemów *CASE*.

Z pewnością system projektowy *Sybase PowerDesigner* należy obecnie, pod względem jakości i funkcjonalności, do jednych z najlepszych [GruMaz00], [GruMaz01]. Dokonamy analizy możliwości funkcjonalnych tego systemu projektowego, a także narzędzi programistycznych środowiska *Sybase PowerBuilder* do wytwarzania systemów dwuwarstwowych klient-serwer i systemów wielowarstwowych przeznaczonych dla sieciowych środowisk intranetowych i Internetu. Niektóre składniki pakietu *PowerDesigner* – nawet z jego starszych wersji – można również z powodzeniem wykorzystywać przy budowie technologicznie najnowszych aplikacji sieciowych opartych na technologii usług webowych (ang. *Web Services*), ze względu na wysoką jakość tego środowiska projektowego.

2. INTERNETOWE I INTRANETOWE SYSTEMY BAZODANOWE

Do niedawna aplikacje systemów bazodanowych realizowały dostęp do danych jedynie za pomocą technologii klient-serwer. Udostępnianie użytkownikowi danych z bazy umożliwiało oprogramowanie klienta działające na stacjach roboczych.

Bardziej zaawansowanymi technologicznie i funkcjonalnie są architektury wielowarstwowe aplikacji i systemów bazodanowych. Z wielu powodów są one najbardziej przydatne i wygodne w eksploatacji w Internecie i w sieciach intranetowych. Warto jednak zauważyć, że w dalszym ciągu rozwijają się profesjonalne produkcyjne systemy informatyczne w technologii klient-serwer. Przykładem mogą tu być scentralizowane systemy bankowe – znów coraz częściej alokowane na dużych centralnych komputerach typu *main-frame*. Dla systemów tych Web jest tylko medium komunikacyjnym a nie dostarczycielem ich wewnętrznych usług funkcjonalnych.

Szczególnym przypadkiem architektury wielowarstwowej jest architektura trzywarstwowa. Koncepcja architektury wielowarstwowej spowodowała rozwój technologii projektowych i technik programistycznych ukierunkowanych na zintegrowanie systemów bazodanowych ze środowiskiem *WWW*. Technologie te są stale rozwijane i obecnie umożliwiają projektowanie i budowę skomplikowanych scentralizowanych i rozproszonych systemów bazodanowych – zarówno relacyjnych jak i obiektowych – wykorzystujących możliwości funkcjonalne interaktywnych serwerów Webu. Zarówno Internet jak i sieci intranetowe stały się dobrej jakości środowiskiem pracy systemów bazodanowych. Sieciowe systemy bazodanowe w *WWW* pod wieloma względami dorównują systemom baz danych i systemom zarządzania bazami danych działającym w sposób klasyczny w środowiskach operacyjnych i sieciowych. Dobrym przykładem takich systemów są wyszukiwarki sieciowe.

Według [WhiMar03] w aplikacji bazodanowej można wyróżnić siedem istotnych elementów składowych:

1. interfejs użytkownika,
2. kontrola wprowadzanych danych,
3. zadania aplikacji,
4. zasady biznesowe,
5. reguły integralności danych,
6. obsługa danych,
7. przechowywanie danych.

Zaletą architektury, w której tylko element siódmy czyli przechowywanie danych zostanie przeniesione na **serwer plików** natomiast pozostałe elementy będą uruchamiane na komputerze użytkownika, jest umożliwienie dostępu do danych wielu użytkownikom, ale wadą jest niska efektywność pracy (bardzo duży ruch danych w sieci) oraz konieczność odpowiednio dobrych komputerów po stronie klienta (np. coraz większa pamięć operacyjna, która będzie w stanie wykonywać zapytania kierowane do bazy danych przy rosnącej liczbie danych). Architektura ta jest bardzo podobna do rozwiązania opartego na jednym komputerze.

W architekturze dwuwarstwowej na komputerze użytkownika jest obsługiwany element pierwszy i drugi (interfejs użytkownika i kontrola danych) natomiast na **serwerze bazy danych** są wykonywane pozostałe elementy. Takie rozwiązanie zmniejsza liczbę przesyłanych danych w sieci (przesyłane są jedynie zapytania i ich wyniki). Powiększanie się bazy danych jest związane z wymianą tylko serwera na nowszy ponieważ komputery użytkowników służą tylko do wprowadzania danych i wyprowadzania wyników.

W architekturze trzywarstwowej system bazodanowy składa się z **trzech warstw**:

1. interfejsu użytkownika i kontroli danych (element pierwszy i drugi),
2. logiki biznesowej implementującej funkcjonalność systemu i serwera interfejsu (element trzeci i czwarty),
3. bazy danych (element piąty, szósty, siódmy).

Warstwy te są zaalokowane fizycznie na różnych komputerach i pracują w środowisku sieciowym. W takiej architekturze łatwo jest administrować systemem bazodanowym i przeprowadzać jego modyfikacje. W poszczególnych warstwach można to wykonywać niezależnie. Jest to podstawowa zaleta w stosunku do architektury dwuwarstwowej klient-serwer. Dodatkową warstwą, w porównaniu z architekturą dwuwarstwową, jest **serwer WWW**.

Warstwa 1 – interfejs użytkownika. Na stacjach roboczych alokuje się warstwę interfejsu użytkownika realizującą prezentację danych, ich przygotowanie oraz przekazanie warstwie aplikacji w celu wprowadzenia do bazy. Warstwę interfejsu użytkownika można zaimplementować jako stronę *WWW* a jej obsługę powierzyć przeglądarce. Mechanizmy komunikacji interfejsu użytkownika z warstwą aplikacyjną (biznesową) systemu bazodanowego pokrywają się wtedy z komunikacją z serwerem *WWW* za pomocą protokołu *HTTP*.

Warstwa 2 – logiki biznesowej i serwera aplikacji. Implementacja reguł biznesowych jest głównym elementem aplikacji *WWW*. W warstwie tej mogą działać standardowe mechanizmy komunikacji międzywarstwowej, albo standaryzowane mechanizmy z technologii firmowych.

W warstwie serwerów aplikacyjnych alokuje się logikę biznesową z kodem operowania na danych w jednej lub kilku bazach danych lub bazie rozproszonej sieciowo. Dostęp do usług tej bazy jest wielokrotny i możliwy dla wielu klientów i wielu aplikacji współbieżnie albo nawet równolegle – wieloprocesowo a często wielowątkowo. Dąży się do tego, aby logika biznesowa była też łatwo dostępna dla zewnętrznych serwerów i zewnętrznych klientów, co jest istotne przy dokonywaniu tzw. integracji aplikacji. Często w warstwie logiki biznesowej występują również specjalizowane serwery raportów i analiz. Warstwa serwera prezentacyjnego zawiera serwery *WWW*, które obsługują polecenia *HTTP* i generują dynamicznie kod prezentacji do wykonania i wyświetlenia na komputerze klienta w formie stron *HTTP*. W warstwie tej mogą być zaimplementowane i skonfigurowane wymagania na generowanie kodu udostępniającego *GUI* prezentacji na stacjach klienckich, np. na podstawie jego identyfikatora *ID*, roli w systemie bazodanowym lub informacyjnym, przynależności do grupy użytkowników.

Warstwa 3 – bazy danych. Trzecią warstwę bazodanowych aplikacji *WWW* stanowi baza danych łącznie z systemem zarządzania tą bazą. Trzeba zatem dysponować mechanizmami projektowo-implementacyjnymi do realizacji dynamicznego dostępu do danych w bazach oraz administrowania tymi bazami.

3. TECHNOLOGIE I NARZĘDZIA PROGRAMISTYCZNE DO PROTOTYPOWANIA I BUDOWY APLIKACJI WIELOWARSTWOWYCH I SIECIOWO ZORIENTOWANYCH

W celu zastąpienia oprogramowania klientów w rozproszonych lub scentralizowanych aplikacjach bazodanowych przeglądarkami *WWW*, konieczne jest wykorzystanie możliwości funkcjonalnych odpowiednich serwerów oraz zastosowanie oprogramowania scalającego bazy danych z internetową siecią rozległą albo sieciami intranetowymi. Pozwala to uniknąć rozwiązywania licznych problemów związanych z tworzeniem klasycznego oprogramowania klient-serwer oraz osiągnąć wieloplatformowy dostęp do bazy danych. Ponieważ wykorzystuje się przy tym standardowe rozwiązania sieciowe, nieistotne jest, na jakiej platformie sprzętowej i w jakim środowisku operacyjnym działa oprogramowanie aplikacji systemu bazodanowego.

4. SYSTEM PROJEKTOWY

Sybase PowerDesigner DataArchitect Suite 6.1

Analizując możliwości funkcjonalne środowiska projektowego *PowerDesigner* uwzględnimy: rodzaj modelowania, metodologię modelowania (modele) procesów biznesowych i funkcji, możliwość modelowania zachowania (dynamiki), metodologię modelowania danych, możliwości modelowania interfejsu użytkownika, możliwość normalizacji związków w relacjach bazodanowych, możliwości generowania szkieletowych i szczegółowych baz danych, możliwości generowania reguł dla danych, generowania aplikacji łącznie z interfejsem użytkownika oraz wsparcie tworzenia aplikacji wielowarstwowych.

W niniejszej pracy nie zajmujemy się aspektem automatycznego tworzenia dokumentacji projektowanego systemu. Warto jednak z naciskiem zaznaczyć, że tworzona i raportowana dokumentacja jest bardzo precyzyjna i wieloaspektowa.

System *Sybase PowerDesigner* doczekał się już edycji 9.5, dziedzicząc najlepsze cechy i rozwiązania z edycji poprzednich. Uwagę na wersji 6.1 skupiamy dlatego, że wykonaliśmy ćwiczenie projektowe budowania bazodanowej aplikacji internetowej i dysponujemy wynikami tego ćwiczenia, prototypem oraz pełną dokumentacją prototypu i jego modelu, wytworzonych z zastosowaniem tej wersji środowiska projektowego *PowerDesigner*.

Sybase PowerDesigner

Oprogramowanie *Sybase PowerDesigner* ([Powersoft1], [Powersoft2]) zajmuje znaczące miejsce wśród narzędzi CASE, wśród których znajdują się narzędzia nieskomplikowane – jak *EasyCASE* firmy Evergreen, oraz narzędzia bardzo złożone, drogie i o dużych wymaganiach pod względem zasobów – np. *Oracle Designer* firmy Oracle.

Sybase PowerDesigner (omawiamy tu wersję *Suite v.6.1*) stanowi zintegrowany zestaw narzędzi do analizy, projektowania, tworzenia i obsługi złożonych baz danych oraz aplikacji klient-serwer i internetowych. Zawiera mechanizmy wspomagające analizę i specyfikację oraz narzędzia do modelowania danych i procesów, umożliwiające generowanie bazy danych i obiektów aplikacji. Jego modułowa struktura umożliwia elastyczny wybór rozwiązań w zależności od rozmiaru projektów i wymagań użytkowników. Wspiera on pracę grupową i zawiera intuicyjny interfejs użytkownika. Narzędzie to posiada budowę modułową. Możemy w nim wyróżnić następujące składniki:

1. *PowerDesigner ProcessAnalyst v.6.1.0*, służący do analizy i modelowania procesów biznesowych,
2. *PowerDesigner DataArchitect v.6.1.0*, umożliwiający modelowanie danych, generowanie, rewersowanie i dokumentowanie schematów baz dla szerokiej gamy systemów bazodanowych,

3. *PowerDesigner MetaWorks*, wspomagający pracę grupową i zapewniający efektywne zarządzanie obiektami projektu przechowywanymi w centralnym słowniku.

Sybase PowerDesigner ProcessAnalyst

Składnik *ProcessAnalyst* pakietu *Sybase PowerDesigner* jest narzędziem, które umożliwia reprezentowanie procesów w systemie informatycznym. Produktem pracy narzędzia *ProcessAnalyst* jest *PAM* – *ProcessAnalyst Model*. *PAM* oferuje funkcjonalną analizę systemu informatycznego. Służy on do analizy procesów składających się na opisywany system. *PAM* jest uzupełnieniem modelu konceptualnego *CDM* (ang. *Conceptual Data Model*), który omówimy później. *PAM* ukazuje dynamikę istniejącą pomiędzy elementami systemu, natomiast *CDM* opisuje statyczne fakty będące sercem tego systemu.

ProcessAnalyst służy do modelowania przepływów danych pomiędzy poszczególnymi elementami systemu. Spełnia on następujące role: służy do analizy procesów w systemie, reprezentuje te procesy w formie modelu oraz tworzy hierarchiczną strukturę reprezentującą te procesy. *ProcessAnalyst* oferuje cztery metodologie projektowe:

- (1) model funkcjonalny *OMT* (*Object Modeling Technique*) – metodologia domysłna,
- (2) *Yourdon/DeMarco*,
- (3) *Gene & Sarson*,
- (4) *SSADM* (*Structured System Analysis and Design Methodology*).

Wszystkie metodologie zaimplementowane w *ProcessAnalyst* używają tych samych pojęć odnoszących się do przetwarzania danych z graficzną reprezentacją elementów diagramu *DFD* (ang. *Data Flow Diagram*) w notacji *OMT*. Te pojęcia to: proces (ang. *process*), dekompozycja procesu (ang. *process decomposition*), terminator (ang. *external entity*), składnica danych (ang. *data store*), przepływ danych (ang. *data flow*). Pozostałe zaimplementowane metodologie, tj. *Yourdon/DeMarco*, *Gane & Sarson* i *SSADM*, bazują na metodach analizy strukturalnej. Stosuje się w nich diagramy przepływu danych (*DFD*).

Sybase PowerDesigner DataArchitect

Składnik *DataArchitect* pakietu *Sybase PowerDesigner* jest narzędziem do projektowania baz danych, które daje użytkownikowi możliwość projektowania dwupoziomowego, tzn. tworzenia modelu konceptualnego i fizycznego. Używając narzędzia *DataArchitect* mamy dostęp do następujących funkcji:

- (a) modelowanie systemów informacyjnych za pomocą diagramu związków encji, nazywanego modelem konceptualnym *CDM* (ang. *Conceptual Data Model*),

- (b) generowanie odpowiadającego mu modelu fizycznego *PDM* (ang. *Physical Data Model*) dla konkretnego systemu *DBMS* biorąc pod uwagę jego specyfikę,
- (c) dostosowanie modelu *PDM* do wymagań fizycznych i wydajnościowych,
- (d) generowanie skryptu generującego docelową bazę danych,
- (e) generowanie wyzwalaczy integralności referencyjnej, jeżeli są one obsługiwane przez docelową bazę danych,
- (f) dostosowywanie i drukowanie raportów,
- (g) rewersowanie istniejących baz danych i aplikacji,
- (h) definiowanie rozszerzonych atrybutów dla obiektów *PDM*.

Model konceptualny CDM reprezentuje sposób postrzegania bazy danych przez użytkownika. Opisuje koncepcyjne związki pomiędzy różnymi typami informacji, a nie ich fizyczną strukturę. *CDM* jest niezależny od poszczególnych systemów *DBMS*. Model konceptualny bazy danych jest przekształcany do modelu logicznego, który odzwierciedla sposób widzenia bazy danych przez projektanta.

Oba modele (konceptualny i logiczny) posiadają reprezentacje graficzne.

Model fizyczny PDM przystosowuje tworzony projekt do konkretnego *DBMS* i jest kolejnym krokiem do uzyskania fizycznej implementacji. Procedura generowania modelu *PDM* umożliwia przeniesienie całości struktury projektu *CDM* do modelu fizycznego.

Proces projektowania struktury dla nowej bazy za pomocą środowiska *PowerDesigner* rozpoczyna się od poziomu konceptualnego. Obejmuje interakcje pomiędzy obiektami takimi jak dziedziny, atrybuty, encje i relacje, oraz dziedziczenie – szczególną relację, która definiuje encje jako przypadek bardziej ogólny. Model konceptualny stanowi dane wejściowe do wygenerowania modelu *PDM* specyfikującego fizyczną implementację bazy danych oraz weryfikuje i waliduje on projekt.

Inżynieria odwrotna. *PowerDesigner* daje możliwości wygenerowania modelu *PDM* albo poszczególnych obiektów z istniejącej bazy danych. Proces ten nazywamy inżynierią odwrotną (ang. *reverse engineering*).

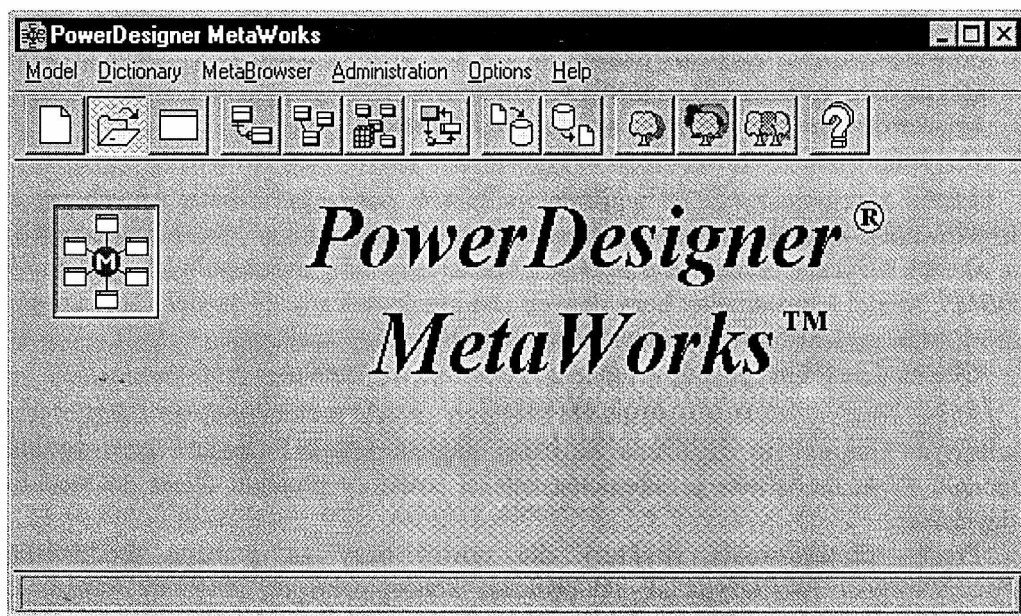
Rewersowaniu mogą podlegać obiekty typów posiadających właścicieli, tzn. tablice, wyzwalacze, perspektywy, synonimy, tabele systemowe, oraz procedury.

Rewersowaniu podlegają również obiekty bazy, które nie posiadają właścicieli. Są to: typy abstrakcyjne, przestrzenie tabel, przestrzenie przechowywania, typy zdefiniowane przez użytkownika, oraz użytkownicy.

PowerDesigner MetaWorks

PowerDesigner MetaWorks (rysunek 1) jest narzędziem zapewniającym koordynację pracy grupowej w złożonych projektach. Zawiera on centralny słownik (repozytorium) i tym samym oferuje scentralizowane zarządzanie modelowaniem

informacji. Umożliwia łatwe przeglądanie modeli procesów, danych, podmodeli, zapewniając spójność definicji obiektów. *MetaWorks* pozwala również administrować prawami dostępu użytkowników oraz blokować modele zapewniając bezpieczeństwo niezbędne dla dużych projektów.



Rys. 1. PowerDesigner MetaWorks

MetaWorks do przechowywania informacji o modelach wykorzystuje własną bazę danych. Dlatego przed składowaniem słownika należy tę bazę najpierw utworzyć, a następnie zdefiniować źródło danych. Źródło danych definiuje połączenie modułu *MetaWorks* z bazą danych. Mimo, że *MetaWorks* może znaleźć istniejące źródło danych, to dla tworzonej bazy danych należy zdefiniować nowe źródło danych. Dopiero wtedy można przystąpić do definiowania tabel słownika, które zapewniają strukturę niezbędną do przechowywania informacji o modelach.

MetaWorks jest programem pracującym w architekturze klient-serwer. Serwer jest centrum aplikacji. Na serwerze składowany jest słownik, który gromadzi i wysyła informacje dotyczące projektów. Informacjami znajdującymi się w słowniku zarządza system zarządzania relacyjną bazą danych (RDBMS). Dostęp do słownika jest realizowany za pomocą klienta poprzez sterowniki ODBC. Klient zgłasza zapytania do serwera aby dokonać konsolidacji (składowanie modelu w słowniku) albo ekstrakcji (pobranie modelu ze słownika). Klient *MetaWorks* może działać na platformach Windows NT/2000, Windows 95/98. Moduły projektowe *PowerDesigner* są instalowane na komputerach klientów, z których mogą się one połączyć ze słownikiem.

Centralny słownik w architekturze klient-serwer ma liczne zalety. Umożliwia pracę grupową, zapewnia integralność danych i spójność danych oraz stwarza możliwość pełnej pracy grupowej.

MetaWorks wprowadza pewną strukturę składającą się z projektów, modeli i podmodeli. Aplikacja jest podzielona na projekty. Projekt jest globalnym szkieletem, na którym można przechowywać model. Projekt może zawierać kilka modeli konceptualnych (*CDM*), modeli fizycznych (*PDM*), modeli *WarehouseArchitect* (*WAM*) oraz modeli *ProcessAnalyst* (*PAM*) wraz z ich podmodelami i to w wielu wersjach. Każdy model zawiera dane związane z konkretną częścią aplikacji i jest często dzielony na podmodele. Kierownik projektu lub jego administrator może przypisać projektantom prawa dostępu do podmodeli. Poszczególne podmodele zawierają część danych w modelu. Projektant może być odpowiedzialny za jeden lub wiele podmodeli.

Słownik *MetaWorks* centralizuje dane projektowe dla wszystkich systemów informatycznych i przez to zapewnia spójność i kompatybilność różnych aplikacji. Rzadko zdarza się aby w jakiejś organizacji aplikacje działały niezależnie. Niezależne operacje w dużej mierze prowadzą do niespójności danych, a to zwykle jest przyczyną załamywania się systemu informatycznego. Słownik *MetaWorks* pomaga zespołowo zarządzać definicjami danych i dziedzinami.

Słownik *MetaWorks* jest narzędziem do zarządzania procesem wytwarzania aplikacji w zespole i pozwala na:

- (1) tworzenie kont i profili dla użytkowników słownika,
- (2) tworzenie projektów, modeli, podmodeli i przypisywanie ich użytkownikom,
- (3) blokowanie i odblokowywanie modeli,
- (4) dzielenie i konsolidację modeli i podmodeli,
- (5) porównywanie definicji obiektów i zarządzanie konfliktami danych,
- (6) zadawanie zapytań do słownika,
- (7) modyfikowanie obiektów w słowniku.

W *MetaWorks* istnieją trzy poziomy interakcji ze słownikiem. Sposób, w jaki użytkownik komunikuje się ze słownikiem, zależy od tego, czy jest on administratorem, kierownikiem projektu czy też projektantem.

MetaWorks posiada zaawansowane mechanizmy zarządzania modyfikacjami w projektach i wersjonowania. Jest też bardzo elastyczny co do odpowiedzialności za konsolidację, ekstrakcję i zarządzanie konfliktami danych. Elastyczność ta wymaga jednak rygorystycznej organizacji procesu projektowania, szczególnie w kwestii zadań i przydzielania odpowiedzialności. Zaleca się stosowanie jednej z dwóch przedstawionych struktur organizacyjnych:

- (a) jeden administrator dokonuje podziału lub konsolidacji modeli oraz rozwiązuje konflikty danych,
- (b) każdy projektant konsoliduje i dzieli modele samodzielnie, natomiast konflikty danych projektanci rozwiązują wspólnie.

Konsolidowane modele, które zawierają dane globalne, składają się zazwyczaj z podmodeli i podprocesów.

Projektanci pracują nad podmodelami, które są ekstrahowane ze słownika. Gdy projektant kończy pracę, zmodyfikowane podmodele są w słowniku konsolidowane. W ten sposób dane globalne są stale uaktualniane najnowszymi modyfikacjami podmodelu.

Istnienie danych globalnych wymaga jednak zarządzania modyfikacjami na nich dokonywanymi. Wymagane jest rozwiązywanie konfliktów modyfikacji tych samych danych. Istnieją dwa sposoby zarządzania modyfikacjami:

- (a) administrator definiuje wspólne dane i zarządza modyfikacjami,
- (b) administrator przypisuje odpowiedzialności dla różnych zbiorów globalnych danych różnym projektantom, zależnie od rozmiaru projektu.

Podczas procesu projektowania administrator albo kierownik projektu może modyfikować model globalny używając słownika.

5. SYSTEM *Sybase PowerBuilder 7 Enterprise*

Skupimy się teraz na opisie możliwości projektowania systemu oraz automatycznego generowania aplikacji w środowisku implementacyjnym *Sybase PowerBuilder 7 Enterprise*, w którym możemy wytwarzać trzywarstwowe aplikacje wewbowe za pomocą następujących składowych tego środowiska:

- (1) *Adaptive Server Anywhere 6.0.1*,
- (2) *PowerBuilder 7.0*,
- (3) serwera Web *Sybase PowerDynamo 3.0*,
- (4) generatora aplikacji *Sybase PowerDesigner AppModeler 6.1.2 for the Web*.

Należy podkreślić, że za pomocą systemu *Sybase PowerBuilder 7 Enterprise* można w sposób zautomatyzowany generować również aplikacje dwuwarstwowe klient-serwer.

Sybase Adaptive Server Anywhere

Sybase Adaptive Server Anywhere to relacyjny, wielodostępny serwer bazodanowy o niewysokich wymaganiach sprzętowych. Jest on przedstawicielem rodziny natywnych korporacyjnych baz danych *Sybase Adaptive Server*, zoptymalizowanym pod kątem efektywności pracy użytkownika. Charakteryzuje go łatwość konfiguracji i zarządzania przy jednoczesnym zapewnieniu wysokiej wydajności i bezpieczeństwa danych. *Adaptive Server Anywhere* jest bazą łatwą do uruchomienia. Instalacja przebiega prawie automatycznie, podobnie jak konfiguracja. Graficzne narzędzie administracyjne, *Sybase Central*, umożliwia zastosowanie techniki przenies i upuść, łatwe zarządzania replikacją, tworzeniem tabel, indeksów i procedur wbudowanych.

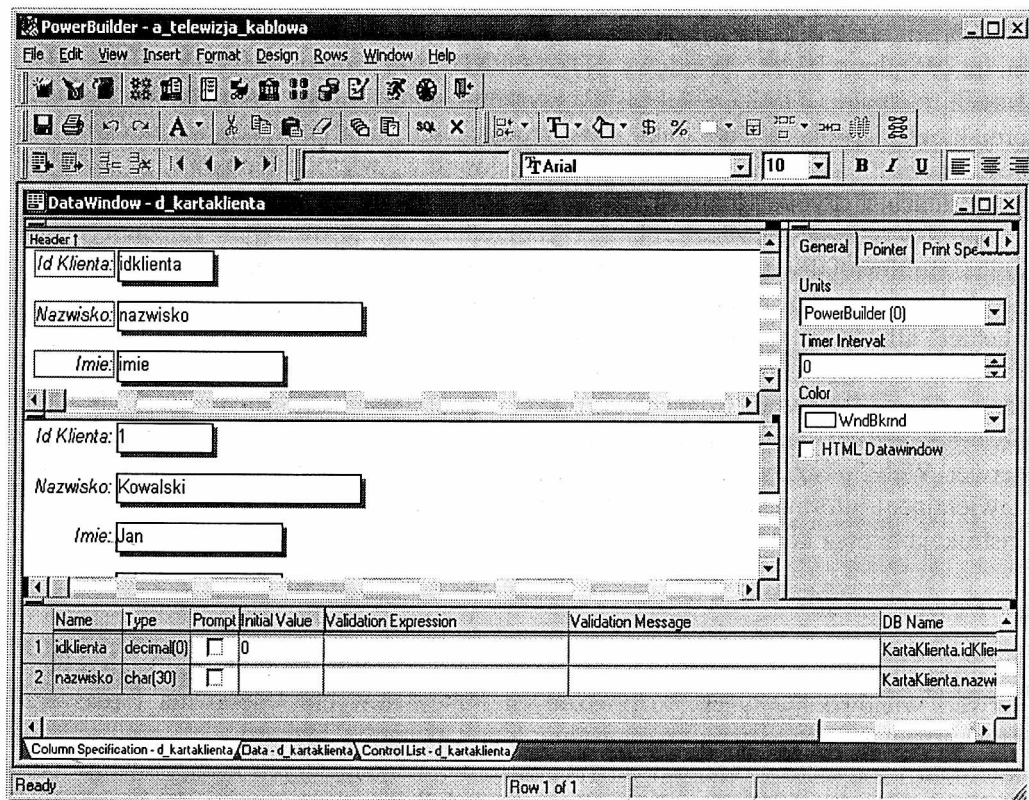
Sybase Adaptive Server Anywhere dostępny jest dla wielu platform systemowych, m. in. Windows 95/98, Windows NT/2000/XP, Sun Solaris, HP-UX, IBM AIX, Novell Netware, OS/2, a także dla systemu Linux. Jego architektura pozwala projektować aplikacje tak, aby bez modyfikacji mogły pracować zarówno na pojedynczym stanowisku, jak i w środowisku wielodostępnym na różnych platformach, z użyciem różnych protokołów sieciowych.

Adaptive Server Anywhere ułatwia tworzenie zaawansowanych aplikacji obsługujących multimedialne typy danych pozwalające na przechowywanie w bazie danych obiektów o dużych rozmiarach (*BLOB*) jak grafika, filmy, dźwięk. Ponadto umożliwia obsługę alfabetów narodowych, łatwe wczytywanie danych zapisanych w innych formatach. Oferuje łatwy i szybki dostęp do danych zarówno przez interfejs *ODBC* (ang. *Open Database Connectivity*) jak i z aplikacji napisanych w C i C++. *Server Anywhere* umożliwia również współpracę z Internetem zapewnioną przez serwer *Web PowerDynamo*. Dzięki temu programiści mogą tworzyć strony *WWW* zawierające informacje z bazy danych w standardzie *HTML* z wbudowanymi instrukcjami oraz kodem w *SQL*.

PowerBuilder

PowerBuilder 7.0 (rysunek 2) to środowisko *RAD* do budowy aplikacji klient-serwer, wielowarstwowych, rozproszonych i internetowych. Usprawnia i rozszerza ono możliwości tworzenia, zarządzania i uruchamiania biznesowych aplikacji dla Internetu i intranetów oraz umożliwia tworzenie standardowych komponentów typu *CORBA*, *COM/DCOM*, *JavaBean*. Posiada dedykowane sterowniki do baz danych i jest dostarczany wraz z narzędziami *PowerDesigner AppModeler* oraz *PowerDesigner AppModeler for the Web*.

PowerBuilder jest efektywnym narzędziem zapewniającym dużą wydajność pracy w środowisku *4GL RAD* podczas tworzenia transakcyjnych aplikacji klient-serwer i aplikacji internetowych dla dużych organizacji. *PowerBuilder* działa na platformach Windows 95/98, Windows NT/2000 (Intel i Alpha), Macintosh, Sun Solaris (Motif), Win OS/2, HP/UX i IBM AIX. Używany język programowania czwartej generacji *PowerScript* jest prosty w użyciu i umożliwia dostęp do wielu wbudowanych funkcji. Można w nim pisać własne funkcje lub korzystać z istniejących, napisanych w C++ lub innym języku, udostępnianych w plikach *DLL*. *PowerBuilder* posiada standardowe cechy narzędzia obiektowego: wielopoziomowe dziedziczenie, hermetyzacja danych i funkcji oraz polimorfizm. Jego kreatory automatycznie tworzą funkcjonalny szkielet komponentu, projekt aplikacji oraz dostosowywalną listę czynności do wykonania. Dołączone zaś biblioteki klas, *PowerBuilder Foundation Class*, oraz przykładowe aplikacje pomagają w szybkim prototypowaniu i konstrukcji oprogramowania.



Rys. 2. Power Builder 7.0, edycja obiektu typu DataWindow

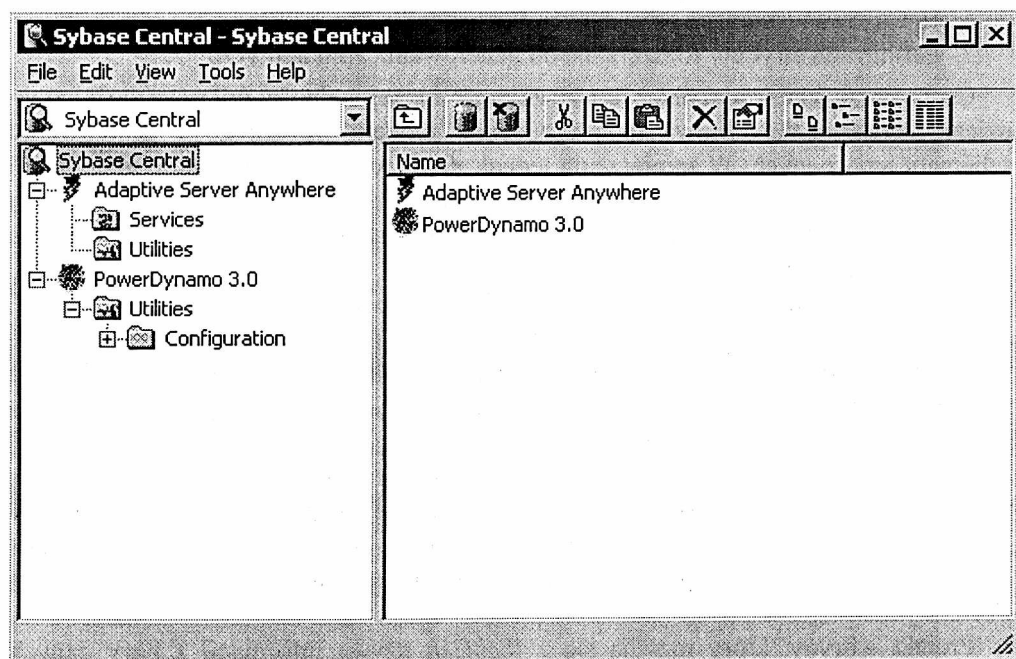
PowerBuilder zapewnia wydajną współpracę z bazami danych. Zawiera on dedykowane sterowniki pozwalające na bezpośredni dostęp do najbardziej rozpowszechnionych systemów relacyjnych baz danych: *DB2*, *DB2/600*, *Informix*, *Microsoft SQL Server*, *Oracle*, *Sybase Adaptive Server Enterprise*, oraz serwera natywnego – *Sybase Adaptive Server Anywhere*. Współpraca z pozostałymi systemami bazodanowymi możliwa jest za pomocą unowocześnionego mechanizmu *ODBC*.

PowerDynamo

PowerDynamo to samodzielny internetowy serwer aplikacji *WWW* udostępniający dynamiczne aplikacje oparte na bazie danych w sieciach intranetowych i w Internecie. Serwer *PowerDynamo* dostępny jest dla platform *Windows 95/98*, *NT/2000* oraz *Sun Solaris*. Uzupełnia on możliwości klasycznych serwerów Web przez dynamiczne wykonywanie szablonów i wysyłanie do klienta odpowiedzi w *HTML* i *JavaScript*. Są to „cienkie” aplikacje (ich funkcje są ograniczone do minimum). Wykorzystanie szablonów zapewnia, że do klienta dociera tylko kod *HTML* dostępny za pomocą

przeglądarki bez konieczności rezygnowania z implementacji logiki biznesowej, która jest tutaj zrealizowana za pomocą wbudowanych instrukcji *SQL*. Szablony *HTML* są przechowywane w bazie danych, co zapewnia ich łatwą replikację. Rozszerzenia języka, takie jak *Java*, *ActiveX*, *C/C++* oraz możliwość wywoływania skompilowanych fragmentów kodu, zwiększają możliwości *HTML*. Dodatkowo, architektura *PowerDynamo* obsługuje *CGI*, *NSAPI* oraz *ISAPI* oferując pełną komunikację z popularnymi serwerami *WWW*. Dla komunikacji z bazami danych *PowerDynamo* obsługuje *ODBC* i *Sybase CT-Lib*.

PowerDynamo zawiera wiele udogodnień ułatwiających i przyspieszających tworzenie aplikacji *WWW*, m.in. przykładową, funkcjonalną aplikację. Kreatory wspierające tworzenie szablonów redukują czas i wysiłek konieczny do stworzenia i uruchomienia aplikacji. *PowerDynamo* zawiera graficzny składacz (ang. *composer*) zapytań *SQL* i kolorujący edytor, a dla bardziej zaawansowanych programistów oferuje współpracę z wieloma innymi środowiskami projektowymi. Pliki oraz istniejące strony *WWW*, stworzone za pomocą innych narzędzi, można łatwo zaimportować, zachować w bazie danych i zarządzać nimi poprzez *PowerDynamo*. Do administracji służy graficzne narzędzie *Sybase Central* (rysunek 3) wykorzystywane w wielu produktach *Sybase*, m. in. w *Adaptive Server Anywhere*.



Rys. 3. Narzędzie *Sybase Central* – administrowanie bazodanowymi serwerami *Sybase Adaptive Server Anywhere* oraz serwerem *WWW Sybase PowerDynamo 3.0*

AppModeler for the Web – Web generator for PowerDynamo

AppModeler for the Web (w skrócie *WebGen*) jest składnikiem pakietu **Sybase PowerDesigner**. W skład pakietu *PowerBuilder* w wersji 7 *Enterprise* wchodzi generator aplikacji *AppModeler 6.1.2* do generowania aplikacji dla następujących środowisk programistycznych: *PowerBuilder 7.0*, *Visual Basic*, *Power++*, *Delphi*, *Web* z zastosowaniem serwera *WWW PowerDynamo 3.0*, generującego strony *ASP (Active Server Pages)*.

AppModeler, narzędzie do projektowania aplikacji klient-serwer i aplikacji internetowych, zapewnia użytkownikowi wszystkie korzyści płynące z fizycznego projektowania danych i implementacji interfejsu. Za jego pomocą można:

1. budować model fizyczny *PDM* (ang. *Physical Data Model*) dla docelowego systemu zarządzania bazą danych z uwzględnieniem specyfiki wybranego docelowego systemu zarządzania bazami,
2. przystosowywać model *PDM* do wymagań fizycznych i wydajnościowych,
3. generować skrypt kreujący bazę danych dla docelowego systemu *DBMS*,
4. generować wyzwalacze integralności referencyjnej jeśli mają one zastosowanie w docelowej bazie danych,
5. przystosowywać i drukować raporty o modelu,
6. rewersować istniejące bazy danych i aplikacje,
7. definiować atrybuty rozszerzone do generowania aplikacji *4GL*.

Cechą narzędzia *AppModeler* jest to, że część funkcjonalności pokrywa się z narzędziem *DataArchitect*. Mowa tutaj o budowie modelu fizycznego i generowaniu schematu bazy danych. W zamierzeniu jednak *AppModeler* jest generatorem aplikacji klienckich. Foldery generatorów aplikacji (*APPGEN*, ang. *Application Generators*) zawierają między innymi podkatalog *DYNTPL* ustawień szablonów generatora *AppModeler for the Web*.

Wszystkie generatory *AppModeler* posiadają podobny interfejs użytkownika. Zasady funkcjonowania generatorów są również podobne, ale istnieją różnice wynikające ze zróżnicowania środowisk docelowych dla aplikacji. W niniejszej pracy opisujemy jedynie działanie generatora *AppModeler for the Web* tworzącego aplikacje w sieciowym środowisku webowym.

AppModeler for the Web jest generatorem aplikacji *WWW* dla sieci *Web*. *WebGen* wykorzystuje dane z *PDM* do stworzenia obiektów z dostarczonych szablonów i wygenerowania aplikacji *WWW*. Projekt witryny internetowej generowany jest bezpośrednio do wskazanego folderu *WWW* lub bazy danych skonfigurowanej dla *PowerDynamo*.

Projekt witryny, który rozwija się z *WebGen* używa informacji z bazy danych o aplikacji. Efektem generowania aplikacji dla *PowerDynamo* są pliki z rozszerzeniem **.STM*, zawierające składnię *HTML*. Gdy uruchamiany jest serwer *PowerDynamo*,

można te pliki przeglądać za pomocą przeglądarki *WWW*. Związki z modelu fizycznego *PDM* generowane są jako odnośniki (ang. *hypertext link*) pomiędzy stronami *WWW*. Lokalizacja sieciowa może być bazą danych *PowerDynamo* albo ścieżką do plików dynamicznych (ang. *Dynamic Files Path*). Każda strona musi posiadać unikatowy adres *HTTP*.

Aby wygenerowany projekt witryny mógł poprawnie działać, należy się najpierw upewnić, czy baza danych istnieje i czy zawiera pola i typy danych zdefiniowane w modelu *PDM*. *WebGen*, tak jak wszystkie inne generatory *AppModeler*, komunikuje się z bazą danych poprzez sterowniki *ODBC*.

6. PODSUMOWANIE

Celem niniejszego artykułu było przedstawienie możliwości projektowania, modelowania i generowania baz danych za pomocą pakietów *Sybase PowerDesigner* i *Sybase PowerBuilder 7 Enterprise*. Omówiono poszczególne składowe tych pakietów (*ProcessAnalyst*, *DataArchitect*, *MetaWorks* oraz *Adaptive Server Anywhere*, *PowerBuilder*, *AppModeler for the Web*), które wspomagają projektowanie i generowanie dwu i trzywarstwowych aplikacji internetowych.

LITERATURA

- [GruMaz01] Gruber J., Mazur Z., *Środowiska projektowe i programistyczne dla zintegrowanych sieciowo systemów bazodanowych*, Pro Dialog – Programowanie i zastosowania komputerów, Nakom, Poznań 2001.
- [GruMaz00] Gruber J., Mazur Z., *Technologie projektowe i programistyczne dla aplikacji wielowarstwowych i zintegrowanych sieciowo systemów bazodanowych*, Prace Naukowe WZI, Bazy danych, Zeszyt 2, Wydawnictwo Politechniki Wrocławskiej, Wrocław 2000.
- [WhiMar03] Whitehorn M., Marklyn B., *Relacyjne bazy danych*, Helion, Gliwice, 2003.
- [Powersoft1] Pliki pomocy oraz dokumentacja firmowa w postaci elektronicznej dołączona do następujących pakietów oprogramowania firmy Powersoft: InfoMaker for Windows, PowerDesigner, PowerBuilder Enterprise, Power++ Enterprise.
- [Powersoft2] Pliki pomocników oraz dokumentacja firmowa w postaci elektronicznej dołączona do następujących pakietów oprogramowania firmy Powersoft (Sybase): PowerDesigner, PowerBuilder Enterprise, Software 2.0, 4 (2000) wraz z dołączonym pakietem Sybase Pack. Źródła internetowe i dokumentacja firmowa.

DESIGN TECHNOLOGIES
SYBASE POWERDESIGNER DATAARCHITECT SUITE 6.1
AND SOFTWARE TOOLS SYBASE POWERBUILDER 7
ENTERPRISE FOR MODELING AND BUILDING DATABASE

Diversity in hardware and software is a fact of life, and network environment is becoming more diverse, not less as computers evolve. In the information systems powerful servers working in teams serve massive databases and surprisingly capable desktop machines provide an increasingly complex array of services, and interact with each other and with bigger systems to compute information in complex information systems. Varied types of computer and network hardware and operating systems are needed to get work done, but the complexity is difficult to deal with. Developing information and database systems by the mean of multi-layer application technologies is making this possible. This paper covers methodologies, techniques and tools that will be fundamental to programmers developing significant database applications in distributed environments. Also, the paper presents some of the typical problems found in software engineering, that can be solved by an approach based on middleware technologies, giving two examples of the technology.

*relacyjne bazy danych,
projektowanie baz danych,
narzędzia CASE*

Dariusz PODULKA*
Zbigniew SZPUNAR**

PROJEKTOWANIE RELACYJNYCH BAZ DANYCH ZA POMOCĄ CASE

Coraz większe wymagania stawiane systemom informatycznym oraz konkurencja rynkowa zmusza producentów oprogramowania nie tylko do podnoszenia jakości tworzonych produktów, ale także do wytwarzania ich w coraz krótszym czasie. Aby sprostać rosnącym wymaganiom niezbędne jest, poza odpowiednią organizacją projektu i właściwym zarządzaniem, stosowanie narzędzi wspomagających proces projektowania – narzędzi CASE (ang. *Computer Aided/Assisted Software/System Engineering*).

Celem tej publikacji jest scharakteryzowanie programów typu CASE, wskazanie obszarów zastosowań narzędzi CASE w projektowaniu relacyjnych baz danych, a także prezentacja przykładowej aplikacji CASE, wykonanej w 2002 roku jako praca dyplomowa inżynierska na Wydziale Informatyki i Zarządzania Politechniki Wrocławskiej.

1. WPROWADZENIE

Narzędzia inżynierii systemów, do jakich należy zaliczyć narzędzia CASE, mają coraz większe znaczenie w procesie produkcji, wdrażania i pielęgnacji oprogramowania. Głównym celem, dla którego są stosowane, jest rozwiązanie problemu złożoności i skali wielkości współcześnie opracowywanych systemów. Ze względu na różnorodność zastosowań nie istnieje jedna, dokładna definicja CASE. Powszechnie termin CASE tłumaczy się jako komputerowo wspomagane tworzenie oprogramowania [Jaszk1997, Wrycza1999, Szejko2002, Barker1996, Yourdon1996].

Określenia CASE używa się także w stosunku do narzędzi wspomagających zarządzanie oraz modelowanie procesów gospodarczych organizacji (co jednak wykracza poza zakres niniejszej pracy).

* 2. rok Uzupełniających Studiów Magisterskich, Wydział Informatyki i Zarządzania Politechniki Wrocławskiej, 50-370 Wrocław, Skwer Idaszewskiego 1, podulka@sun10.ci.pwr.wroc.pl

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

W pierwszej części pracy przedstawiono próbę scharakteryzowania narzędzi CASE, dokonano przeglądu kategorii narzędzi oraz wybranych metodyk wspomaganych komputerowo za pomocą narzędzi CASE. Podjęto próbę wskazania roli narzędzi CASE w projektowaniu baz danych oraz określenia możliwości CASE w tych działaniach.

W drugiej części przedstawiono narzędzie klasy CASE – system informatyczny wspomagający projektowanie relacyjnych baz danych. Zaprezentowano aplikację, która upraszcza i przyspiesza proces budowy bazy danych, wykorzystując graficzne metody modelowania. Opisano architekturę systemu i rolę poszczególnych elementów aplikacji. W nawiązaniu do strukturalnej metodyki projektowania baz danych pokazano wykorzystanie narzędzia w kolejnych etapach procesu projektowania.

2. WSPOMAGANIE PROJEKTOWANIA A NARZĘDZIA CASE

W procesie projektowania można wyróżnić pewne etapy powstawania systemu, zwane łącznie cyklem produkcyjnym lub cyklem życia systemu. Biorąc pod uwagę zakres, w jakim narzędzia CASE obejmują cykl życia systemu, wyróżnia się pakiety częściowe (ang. *workbenches*) oraz środowiska projektowe (ang. *environments*). Pakiety częściowe dzielą się na programy typu *Upper-CASE*, używane w fazach analizy i projektowania oraz *Lower-CASE*, które wspomagają etapy implementacji, testowania i utrzymania systemu. Rozwiązania o charakterze środowisk projektowych, posiadające cechy obu wymienionych grup określane są jako *Cross Life Cycle CASE* lub krócej *Integrated-CASE (I-CASE)*.

Cechą charakterystyczną dla CASE, zarówno częściowych jak i środowiskowych, jest udostępnianie naturalnych form reprezentacji danych – graficznych języków modelowania, które są wykorzystywane zwłaszcza w fazach analizy i projektowania systemu. Metody graficzne pozwalają przedstawić złożony system na odpowiednio wysokim poziomie abstrakcji, dzięki czemu projekt jest bardziej zrozumiały. Równie ważną korzyścią wynikającą z zastosowania wizualnych narzędzi modelowania, jest tzw. „continuum” [Kozacz2002] – łagodne przejście między poziomami modelowania, o różnym stopniu szczegółowości. Aby to umożliwić, narzędzia CASE muszą również zadbać o kontrolę integralności wytwarzanych i przekształcanych produktów prac. Pod tym względem lepiej sprawdzają się środowiska zintegrowane, w których całość procesu twórczego jest pod kontrolą jednego narzędzia. Niestety, I-CASE są uważane za mało elastyczne, często nie adekwatne do potrzeb projektowych lub wręcz narzucające określoną metodologię projektową. Natomiast

w przypadku cząstkowych CASE, problemem może okazać się brak kompatybilności między pakietami różnych producentów. Alternatywą dla wymienionych narzędzi są pakiety otwarte [Wrycza1999] (lub inaczej: uniwersalne), zwane też *Meta-CASE* (*M-CASE*) [Beynon1999]. Programy te posiadają bogaty zasób form i notacji graficznych, nie związanych z żadną konkretną metodologią projektową. Dzięki *M-CASE* możliwe jest opracowanie narzędzi dla własnej metodyki. Wymaga to jednak dużego zaangażowania i doświadczenia od użytkownika, który na bazie ogólnego meta-modelu musi uszczegółowić zbiór symboli graficznych, zasady poprawności albo reguły przekształcania modeli. Dlatego dla typowych zastosowań bardziej uzasadnione jest przyjęcie rozwiązania opartego na sprawdzonej metodologii – wybór dedykowanego CASE.

Wybór odpowiedniej metodyki wraz z narzędziem jest jednym z kluczowych czynników mających wpływ na powodzenie projektu. Co więcej, CASE mogą być przydatne tylko w takim stopniu, w jakim użyteczna jest wspomagana przez nie metodologia [Adamcze1998].

Ze względu na różnorodność projektowanych systemów, w reakcji na ciągły postęp oprogramowania i coraz większe oczekiwania wobec CASE, powstało i rozwinęło się wiele metodyk wytwarzania przeznaczonych dla takich kategorii systemów jak: systemy czasu rzeczywistego, systemy baz danych, hurtownie danych, aplikacje sieciowe i internetowe.

Obecnie dość popularną gałęzią zastosowań CASE jest wspomaganie projektowania baz danych. Pierwsze metodyki wytwórcze dla tej grupy systemów pojawiły się stosunkowo dawno, stąd dziś można zaobserwować zróżnicowanie oferowanych rozwiązań. Początkowo CASE działały w oparciu o strukturalne metody analizy i projektowania (np. STRADIS, IE, SSADM), które porządkowane były w cyklu produkcji zwanym modelem wodospadu (ang. *waterfall*). Nowsze CASE charakteryzują się dużą elastycznością w doborze cyklu tworzenia systemu i stosują strukturalne i (lub) obiektowe metodologie projektowe (np. Coad/Yourdon, OMT, UML). Niektóre pakiety zawierają notacje pochodzące z różnych metodologii, umożliwiając ich wzajemne uzupełnianie się i współdziałanie na różnych etapach cyklu życia.

W ramach projektowania i pielęgnacji systemów baz danych istnieje szereg zadań, których wykonywanie jest wspomagane przez CASE. Na poziomie koncepcyjnym CASE dostarczają środków do modelowania semantycznego. Do najbardziej rozpowszechnionych (w podejściu strukturalnym) należą diagramy związków encji Chena, Bachmana, Shlaer–Mellora, a także standaryzowane notacje Martina i IDEF1X. W narzędziach modelowania obiektowego zwykle dostępne są diagramy klas Coad/Yourdona, Shlaera/Mellora, Rumbaugh, Boocha i objętej oficjalnym standardem notacji UML.

Aby ułatwić opracowanie architektonicznego modelu logicznego, CASE wyposaża się w mechanizmy transformujące model koncepcyjny do logicznego (obiekto-relacyjnego lub relacyjnego). Na poziomie modelowania logicznego CASE pozwalają

zaplanować indeksy, perspektywy, wyzwalacze a także wpisać w schemat dodatkowe ograniczenia.

Wiele uwagi przywiązują producenci CASE do słowników danych (ang. *repositories*), które stanowią mogą obszerną bazę wiedzy. Repozytorium pozwala gromadzić produkty prac i wielokrotnie je wykorzystywać w różnych etapach prac, czy nawet w różnych projektach. Słownik pomaga także utrzymać spójność między modelami. Warto również zaznaczyć, że repozytorium może być wzbogacone o własne mechanizmy wersjonowania.

Gdy poszczególne fazy projektu są realizowane przy pomocy różnych CASE, niezbędne okazać się mogą funkcje importu/eksportu danych. Jednym z wyłansowanych standardów wymiany danych, implementowanym przez CASE, jest CDIF (*CASE Data Interchange Format*).

Wzrastająca rola pracy zespołowej wymaga od narzędzi CASE zapewnienia metod współdzielonego dostępu do zasobów (a ściślej do repozytorium). Stąd obecne są w CASE funkcje do zarządzania użytkownikami, ustalania uprawnień (dla użytkownika lub grupy) do określonych informacji. Dane z repozytorium mogą być udostępniane w sieci. Coraz częściej spotyka się, tworzone przez producentów CASE, tzw. otwarte repozytoria, z którymi programy CASE łączą się za pośrednictwem Internetu.

Dla inżyniera bazy danych najważniejszym mechanizmem może okazać się generator kodu, który na podstawie opracowanego schematu, dla wybranego RDBMS, automatycznie utworzy skrypt SQL. Choć większość RDBMS spełnia założenia standardu SQL, pomiędzy produktami istnieją pewne różnice (np. inne nazwy typów danych binarnych), powodujące konieczność dostosowania kodu. Dla uniezależnienia generatora SQL od częstych zmian specyfikacji RDBMS, narzędzia CASE wyposaża się we własne języki skryptowe, parametryzujące wynikowy kod.

Automatyzacji poddaje się również proces inżynierii odwrotnej, w którym kod źródłowy (SQL) jest przetwarzany w diagram relacji. Proces ten pozwala na migrację danych do nowego środowiska (RDBMS), ułatwia modyfikacje działających systemów oraz umożliwia odtworzenie dokumentacji projektowej.

Zadaniem wspomagany przez CASE jest także raportowanie. Sposób dokumentowania jest uzależniony od przyjętej metodyki. Raporty mogą być generowane na każdym etapie prac. Zapisywane są w wielu różnych formatach danych (np. SXW, RTF, PDF, PS, HTML, XLS). Raportowanie jest ważnym elementem procesu projektowania bazy danych. Wytworzone raporty mogą mieć różnorodne przeznaczenie.

Możliwości i oczekiwania wobec narzędzi CASE są opisane m.in. w [BarlLong 1996, Jaszko1997, Hebbel1997, Frank1997, Roszkow1998, Schmuller2003, Szejko 2002, Grochow2003]. Najbardziej aktualnych informacji o CASE można też poszukiwać na stronach internetowych producentów CASE, których częściowy wykaz został dołączony na końcu publikacji.

3. PREZENTACJA ROZWIĄZANIA

Przedmiotem niniejszej pracy jest zaprezentowanie samodzielnie zaprojektowanego i zaimplementowanego systemu informatycznego wspomagającego projektowanie relacyjnych baz danych. Narzędzie jest przeznaczone do analizy i projektowania struktur relacyjnych baz danych. Posiada wyspecjalizowane funkcje pozwalające zamodelować i opisać schemat bazy danych. Koncentruje się na statycznym aspekcie tworzenia bazy danych. Opiera się na typowym dla narzędzi CASE graficznym języku modelowania oraz na zaczerpniętej z notacji J. Martina [Martin1985] strukturalnej metodologii projektowej. System ten służy pomocą w trzech zasadniczych etapach konstrukcji bazy danych: koncepcyjnym, logicznym i fizycznym. Opisywane narzędzie pomaga m.in. w sposób interaktywny modelować diagramy związków encji, automatycznie weryfikować poprawność modelu i generować schemat relacyjny w formie diagramu graficznego oraz kodu SQL.

Aktualna wersja systemu jest aplikacją jednostanowiskową. Spośród innych systemów tego typu wyróżniają go:

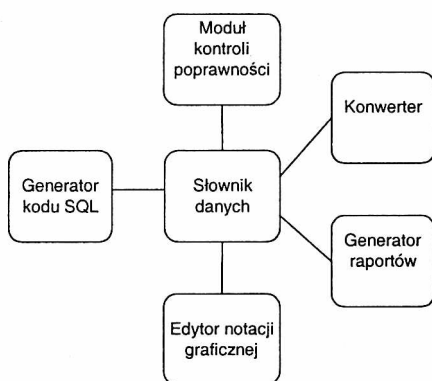
- właściwości notacji semantycznej: rozbudowana charakterystyka elementów modelu, możliwość tworzenia złożonych identyfikatorów encji i definiowania zbioru kluczy kandydujących,
- niezależność od platformy systemowej,
- funkcjonalny, polskojęzyczny interfejs.

Narzędzie posiada walory edukacyjne. Cechami, które nadają mu charakter dydaktyczny, są:

- możliwość zaobserwowania różnic, jakie dzielą model semantyczny od relacyjnego modelu logicznego i dzięki temu poznanie zasad jakie obowiązują podczas transformacji modelu,
- konieczność przeprowadzenia weryfikacji diagramu związków encji, podczas której użytkownik jest zmuszony do usuwania błędów, aby móc wykonać kolejne fazy projektowania bazy danych,
- system pomocy, który dostarcza elementarnej wiedzy z zakresu projektowania baz danych,
- szablony raportów odpowiadające podstawowym wymaganiom dokumentacyjnym w nauczaniu projektowania baz danych.

3.1. ELEMENTY BUDOWY SYSTEMU

Ogólna budowę systemu przedstawia rysunek 1. Widoczny tu schemat jest kompozycją złożoną z sześciu elementów. Centralną częścią systemu jest *słownik danych*. Ze słownikiem są powiązane narzędzia do edycji i przeglądania jego zawartości. W przyjętym rozwiązaniu słownik jest postrzegany jako wewnętrzna, integralna baza danych. Zalety wynikające z tego podejścia to m.in. prostsza instalacja i konserwacja programu, brak ewentualnych problemów komunikacyjnych pomiędzy programem a zewnętrzną bazą danych bezpośredni oraz szybki dostęp do danych przechowywanych w słowniku. Niestety, słownik nie jest dostępny z poziomu innych aplikacji.



Rys. 1. Składowe systemu w kontekście CASE

Nierozłącznym składnikiem systemu jest *edytor diagramów* (*edytor notacji graficznej*). Znajduje on zastosowanie w etapach projektowania konceptualnego i logicznego. Na etapie koncepcyjnym, w oparciu o notację J. Martina, edytor pozwala tworzyć diagramy związków encji (ang. *Entity-Relationship*, *E-R model*, *Entity Relationship Diagram*). W fazie projektowania logicznego edytor służy do wizualizacji wygenerowanego modelu relacyjnego. Modelowanie realizowane jest przez użytkownika w sposób interaktywny. Użytkownik posiada kontrolę nad obiektami diagramu i może nimi w efektywny sposób zarządzać. Pomiedzy graficznym modelem a słownikiem danych zachowana jest integralność.

Za poprawność zbudowanego modelu odpowiedzialny jest w systemie *moduł weryfikacji*. Moduł ten zawiera zestaw metod, za pomocą których wskazany diagram jest sprawdzany pod względem formalnym wraz z detekcją błędów. Zastosowane rozwiązanie umożliwia dostosowanie niektórych kryteriów poprawności do potrzeb dedykowanego RDBMS (ang. *Relational Database Management System*). Za korekcję błędów odpowiedzialny jest użytkownik systemu. W wielu przypadkach czynności kontrolne są przeprowadzane bezpośrednio po wprowadzeniu danych przez

użytkownika. Dzięki temu zasadniczy proces weryfikacji jest prostszy i mniej czasochłonny.

Od wykonywania wielu czasochłonnych i złożonych działań procesu transformacji uwalnia użytkownika konwerter. Jedyne czynności, jakie użytkownik musi dokonać przed rozpoczęciem konwersji, to wybór dedykowanego RDBMS oraz podanie nazwy nowego schematu. Konwerter odpowiada za przekształcenie modelu semantycznego w relacyjny model logiczny. Zasady transformacji są opisane m.in. w [Mazur2002, Beynon2000, Barker1996]. Wśród wielu typowych sytuacji, z jakimi musi poradzić sobie translator, są również, godne podkreślenia, przekształcenia związków rekursywnych i wielokrotnych zależności nadtyp–podtyp. Efektem transformacji, poza utworzeniem i zapisaniem w słowniku danych struktury bazy danych, jest wygenerowanie graficznego modelu.

Reprezentacja graficzna modelu relacyjnego jest dobrym sposobem opisu struktury bazy danych z punktu widzenia użytkownika. Jednak graficzny język dla RDBMS nie jest zrozumiały. Aby móc zaimplementować w RDBMS wykonany model, należy go zapisać w odpowiednim języku programowania. Stąd obecność w systemie *generatora kodu SQL*. Generator kodu przekształca schemat zdefiniowany w języku notacji graficznej na kod programu (skrypt) dla określonego RDBMS. Do tego celu wykorzystuje język SQL DDL (ang. *Structured Query Language Data/Database Definition Language*). Generator posiada opcje umożliwiające określenie elementów, które zostaną dodatkowo dołączone do skryptu. Z generatorem kodu sprzężono prosty edytor tekstu.

Aby ułatwić tworzenie dokumentacji projektowej do systemu dodano *generator raportów*. Zawiera on zestaw gotowych szablonów raportów. Ich źródłem są diagramy związków encji i schematy relacji obecne w słowniku danych. Użytkownik ma możliwość wyboru grup informacji, które mają się znaleźć w raporcie. Pliki raportów są zapisywane w formacie HTML.

3.2. WYKORZYSTANIE MOŻLIWOŚCI CASE W ETAPACH KONSTRUKCJI BAZY DANYCH

Celem tej części pracy jest wskazanie funkcji narzędzia CASE i sposobu ich wykorzystania w poszczególnych etapach tworzenia relacyjnych baz danych.

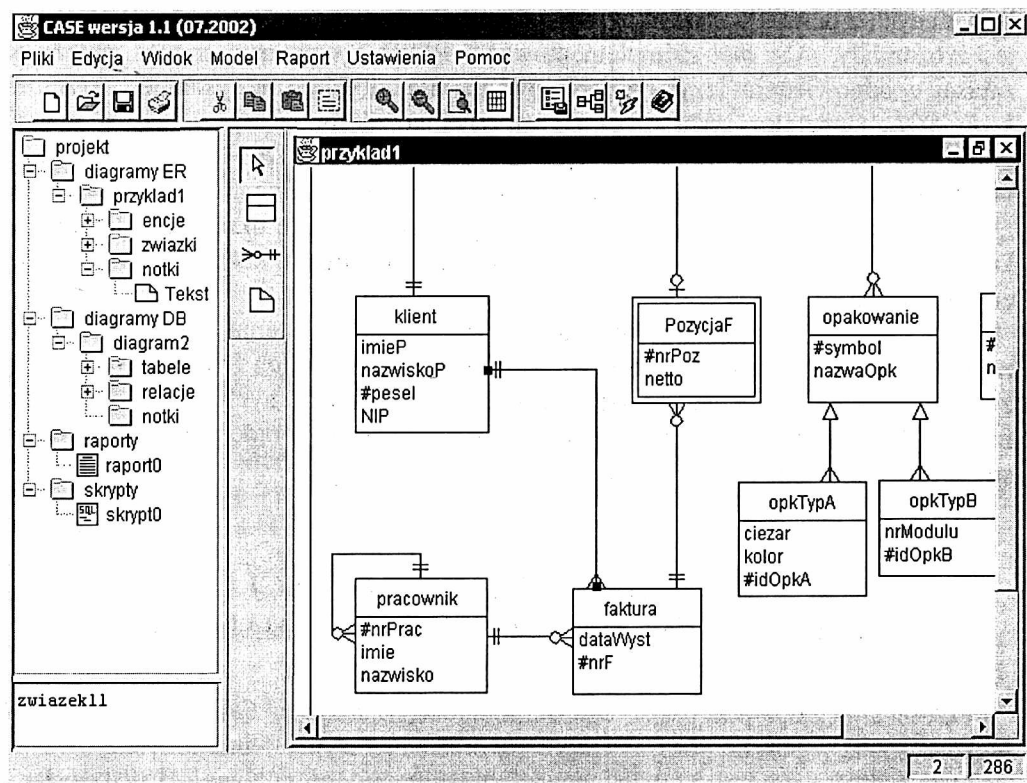
Opis funkcjonalny został wzbogacony o zagadnienia projektowe dla podkreślenia ich roli w procesie projektowania oraz dla pełnego zrozumienia oferowanych przez system usług.

Narzędzie wspiera jedynie wybrane fragmenty procesu wytwórczego, koncentrując się na aspekcie statycznym, dlatego tylko te elementy będą omówione w dalszej części.

3.2.1. MODELOWANIE KONCEPTUALNE

Modelowanie koncepcyjne stanowi pierwszą fazę projektowania bazy danych. Etap ten jest poprzedzony analizą wymagań, która obejmuje rozpoznanie obszaru analizy, zapoznanie z zasadami funkcjonowania organizacji, w której projektowany system ma działać, dokonanie oceny aktualnie wykorzystywanego systemu, prognozowanie przyszłych wymagań nowego systemu i dokonanie całościowej oceny wymagań informacyjnych organizacji.

Modelowanie koncepcyjne, zgodnie z Beynonem [Beynon2000], polega na przekształceniu wcześniej zdefiniowanych wymagań użytkowników w schemat (diagram) zwany modelem semantycznym lub modelem rozszerzonym [Date2000].



Rys. 2. Główne okno aplikacji

Aby rozpocząć budowę modelu, należy w głównym oknie aplikacji (rysunek 2) wybrać opcję „Nowy ERD”, co spowoduje otwarcie nowej, pustej strony. Do jej wypełnienia niezbędne będzie skorzystanie z paska narzędziowego, który zawiera

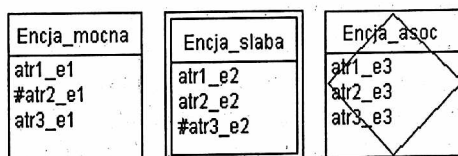
zestaw gotowych symboli graficznych potrzebnych do budowy schematu. Dostępne są ikony encji, związku a także notki. Wybór ikony i kliknięcie w obszarze roboczym diagramu powoduje dodanie obiektu do schematu. System automatycznie przydziela obiektom nazwy, tak, aby każda encja lub związek były rozróżnialne. Unikalność nazw obiektów i ich atrybutów jest jednym z warunków poprawności diagramu. Dwukrotne kliknięcie na wskazany obiekt daje możliwość szczegółowej edycji właściwości tego obiektu.

Własności encji zostały podzielone na trzy grupy: właściwości podstawowe, atrybuty i klucze.

Pierwsza zakładka **Własności** (rysunek 3) pozwala określić nazwę, symbol, typ i opis encji.

Rys. 3. Okno edycji encji

W notacji wyróżnione są następujące typy encji: mocna, słaba i asocjacyjna (rysunek 4).



Rys. 4. Typy encji (od lewej: encja mocna, słaba oraz asocjacyjna)

Wyboru encji mocnej należy dokonać, gdy obiekt ma być identyfikowany niezależnie od innych bytów. Encję słabą stosuje się dla obiektu, którego wystąpienie i możliwość identyfikacji są uwarunkowane istnieniem innej (innych) encji. Encja asocjacyjna uznawana jest za szczególny przypadek encji słabej, która jest identyfikowana wyłącznie za pomocą związków z innymi encjami. Podczas tworzenia encji system domyślnie generuje obiekt mocny.

W drugiej zakładce **Atrybuty** (rysunek 5) znajdują się elementy służące do modelowania atrybutów encji. Za pomocą atrybutów uszczegóławiane jest znaczenie encji. Atrybuty służą do klasyfikacji, określania ilości lub wyrażania stanu encji. Informacja o atrybucie zawiera jego nazwę, typ (przynależność do dziedziny), rozmiar, precyzję, występowanie i unikatowość.

Własności encji

Własności | **Atrybuty** | Klucze

Atrybuty encji

Nazwa Atrybutu	Typ	Arg1	Arg2	Wymagany	Unikatowy
data_zam	Varying Character	brak	brak	☒	☒
nr_zam	Integer	brak	brak	☒	☐

Dodaj Usuń

Ok Zamknij okno

Rys. 5. Edycja atrybutów encji

Nazwa atrybutu podlega pewnym ograniczeniom (np. niepowtarzalny, niezerowy ciąg znaków), których niespełnienie jest sygnalizowane przez system przy tworzeniu i edycji atrybutu lub w czasie kontroli poprawności modelu.

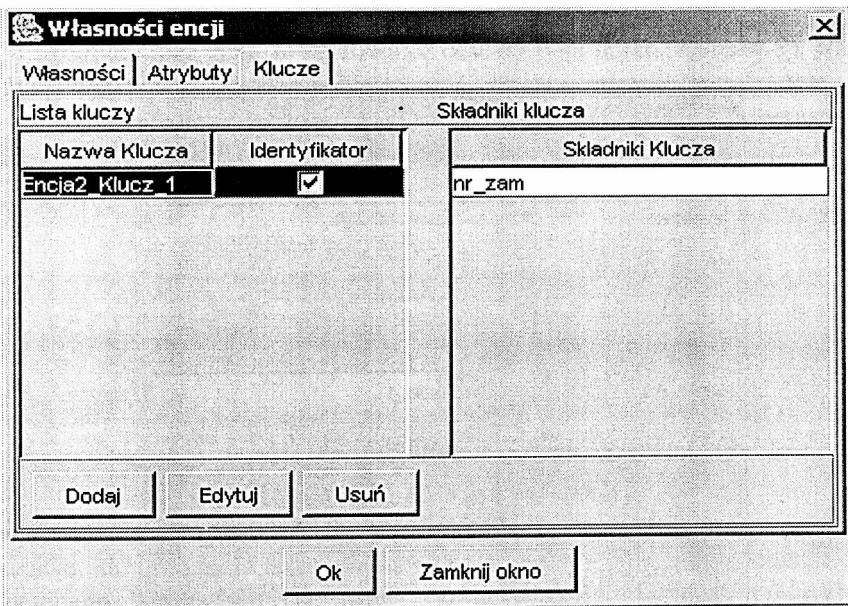
Typ atrybutu oznacza typ wartości danych, jakie ten atrybut będzie przechowywać. W przyjętym rozwiązaniu typ atrybutu należy wybrać z listy. W zależności od dokonanego wyboru, edytowalne lub wyłączone są rozmiar i (lub) precyzja atrybutu. Dostępne w systemie typy atrybutów oraz ich własności przedstawia Tabela 1.

O tym, czy wartość atrybutu jest obowiązkowa, decyduje zaznaczenie pola o nazwie „wymagany”. Natomiast pole „unikatowy” nadaje atrybutowi automatycznie charakter obligatoryjny oraz nakłada ograniczenie przyjmowania nie powtarzających się wartości.

Tabela 1. Typy atrybutów

Typ danych	arg1	arg2	Typ danych	arg1	arg2	Typ danych	arg1	arg2
Short integer	–	–	Time	–	–	Varying character	tak	–
Byte	tak	–	Date & time	–	–	Long varying character	tak	–
Float	tak	–	Timestamp	–	–	Decimal	tak	Tak
Real	tak	–	Binary	tak	–	Char	tak	–
Money	tak	tak	Long binary	tak	–	Varying multibyte	tak	–
Serial	tak	–	Bitmap	tak	–	Date	–	–
Boolean	–	–	Image	tak	–	Number	tak	Tak
Long char	tak	–	Ole	tak	–	Multibyte	tak	–
Text	tak	–	Integer	–	–			

Trzecia zakładka **Klucze** (rysunek 6) umożliwia budowanie kluczy encji. Klucze są podstawą identyfikacji wystąpień encji. Poprawne zdefiniowanie kluczy ma bezpośredni wpływ na zachowanie integralności na poziomie encji.

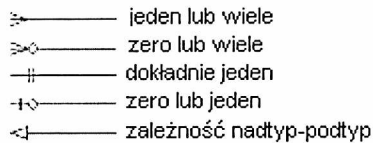


Rys. 6. Edycja kluczy encji

Projektowanie klucza polega na wyselekcjonowaniu atrybutu (klucz prosty) lub większej liczby atrybutów (klucz złożony), które razem będą miały unikatową wartość. Należy też podać nazwę tworzonego klucza. Składnikami klucza mogą być tylko atrybuty o charakterze obligatoryjnym, co jest zapewnione w systemie dzięki ukryciu pozostałych atrybutów. Encja może posiadać wiele kluczy. Spośród nich

wyberany jest jednoznaczny identyfikator, z którego w modelu relacyjnym zostanie utworzony klucz główny. Własny identyfikator musi być obecny we wszystkich encjach z wyjątkiem asocjacyjnych, co wpływa na poprawność diagramu.

Definiując związek (rysunek 7) należy wskazać jedną (związek jednoargumentowy, związek rekursywny) lub dwie encje, których będzie on dotyczyć. Związek wprowadza trwałe, jawne zależności między encjami, których istnienie jest fundamentem do tworzenia reguł integralności referencyjnej.



Rys. 7. Rodzaje związków

Edycja właściwości związku (rysunek 8) obejmuje określenie nazw ról, nazwy powiązania, wybór stopnia i typu uczestnictwa. Stopień uczestnictwa (*liczebność*) wyznacza liczbę instancji encji biorących udział w związku.

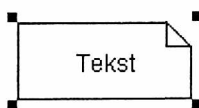
Rys. 8. Okno edycji związku

Zgodnie z przyjętą w systemie notacją, liczebność związku jest jedną z trzech kombinacji: 1:1, 1:Wiele (Wiele:1), Wiele:Wiele. Oddzielnie dla każdej ze stron powiązania ustalany jest typ uczestnictwa, mówiący o tym, czy uczestnictwo w związku jest wymagane czy nie (obowiązkowe czy opcjonalne).

W przypadku związków o stopniu 1:1 istnieje możliwość zaznaczenia kierunku referencji. Jeśli parametr ten nie zostanie określony, to po transformacji do modelu relacyjnego tabele odpowiadające powiązanym encjom będą zawierały wzajemne klucze obce. Wskazanie encji dominującej ogranicza mapowanie klucza jedynie do uczestnika nie wyróżnionego.

Za pomocą związków możliwe jest wprowadzenie zależności nadtyp–podtyp między encjami. Wówczas encja–podtyp dziedziczy wszystkie atrybuty i związki encji–nadtypu. Elementy te są definiowane tylko raz, dla encji–nadtypu, i są wspólne dla wszystkich podtypów. Podtypy posiadają cechy encji słabej. Encja może jednocześnie pełnić rolę nadtypu względem jednej encji i podtypu dla innej encji. Podtypy mogą zachodzić na siebie (zawierać się) lub wzajemnie się wykluczać. Rozróżnienie tych sytuacji jest dokonywane w modelu relacyjnym, za pomocą dodatkowych ograniczeń sformułowanych przez użytkownika.

Na diagramie mogą znaleźć się również obiekty typu notka (rysunek 9). Nie są one formalnie związane z modelem. Notka przechowuje informacje tekstowe, których celem jest uzupełnienie, wyjaśnienie lub skomentowanie zawartości schematu, np. dla pozostałych członków zespołu projektowego.



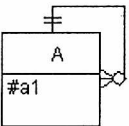
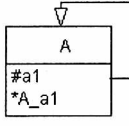
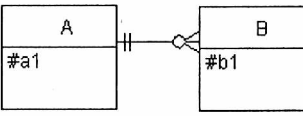
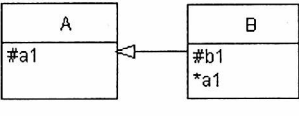
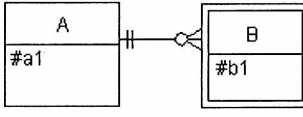
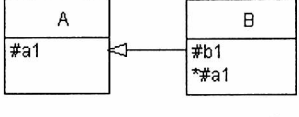
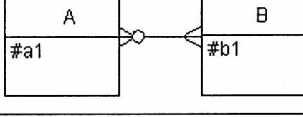
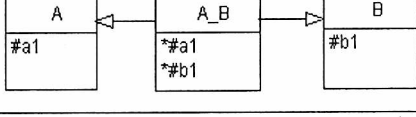
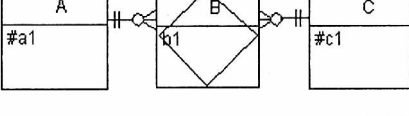
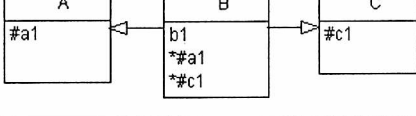
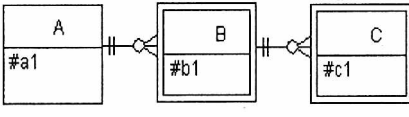
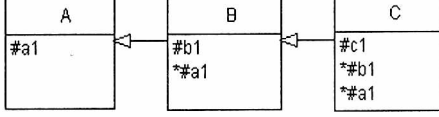
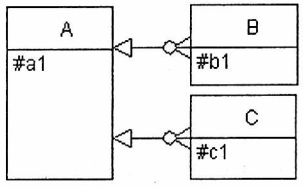
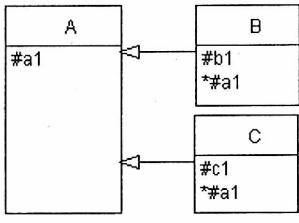
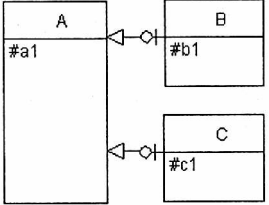
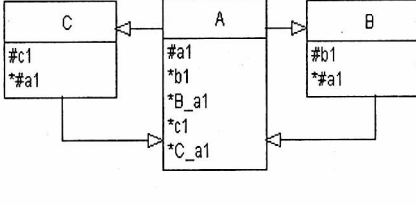
Rys. 9. Notka

3.2.2. PRZEJŚCIE DO RELACYJNEGO MODELU LOGICZNEGO

Przejście do modelu relacyjnego jest środkiem do implementacji modelu koncepcyjnego. W procesie tym, zwanym projektowaniem logicznym danych, model koncepcyjny jest tłumaczony na architektoniczny model relacyjny. Uzyskane w ten sposób struktury danych można bezpośrednio dostosować do wymogów konkretnego systemu zarządzania bazą danych.

Projektowanie logiczne w systemie jest w dużym stopniu zautomatyzowane. Rola użytkownika sprowadza się do wyboru dedykowanego RDBMS i wpisania nazwy nowego schematu. Przed przystąpieniem do transformacji system sprawdza poprawność diagramu. Jeśli wynik kontroli jest pomyślny, to w efekcie utworzony zostaje obraz struktury bazy danych w postaci graficznego diagramu.

Tabela 2. Model koncepcyjny a model relacyjny

Nr	Model koncepcyjny	Model relacyjny
1		
2		
3		
4		
5		
6		
7		
8		

Przykładowe, elementarne schematy koncepcyjne oraz odpowiadające im diagramy relacyjne są zamieszczone w Tabeli 2. Widoczne tu są różnice, jakie zaznaczają się pomiędzy dwoma typami modeli, w tym również sposób mapowania atrybutów na klucze obce i tworzenia kluczy głównych. Symbol „#” w odniesieniu do encji oznacza atrybut identyfikujący encję, natomiast w modelu relacyjnym jest to składnik klucza głównego. Klucze obce (tylko model relacyjny), są poprzedzone symbolem „*”.

Sytuacja pierwsza z Tabeli 2. ilustruje sposób odwzorowania związku rekursywnego 1:0..N w modelu relacyjnym. Jak widać, do tabeli A system dodał klucz obcy A_a1, którego wartości pochodzą z tej samej tabeli, a ich zbiór jest wyznaczony przez wartości klucza głównego ustanowionego na atrybucie a1.

Układ drugi przedstawia związek 1:0..N pomiędzy encjami A i B. W modelu relacyjnym klucz obcy mapowany został do tabeli B, a jego wartości pochodzą z tabeli A. Klucze główne w pierwszej i drugiej sytuacji tworzone są wyłącznie na podstawie identyfikatorów encji.

Model semantyczny w zestawie trzecim składa się z encji mocnej A połączonej związkiem 1:0..N z encją słabą B. W schemacie relacyjnym tabela B ze względu na obecność encji słabej, otrzymuje klucz główny złożony z dwóch atrybutów: a1 i b1 (mimo, że identyfikator encji miał tylko jeden atrybut). Atrybut a1 jest jednocześnie kluczem obcym tabeli A.

Przykład czwarty pokazuje transformację związku 0..N:1..N encji mocnych A i B. W przekształceniu tym system generuje dodatkową tabelę A_B, zwaną tablicą asocjacyjną. W kolejnym kroku mapowane są do niej klucze obce z tabel A oraz B. Klucz główny tabeli A_B powstaje ze złożenia kluczy obcych, czyli z atrybutów a1 i b1. Klucze główne tabel A i B są zbudowane w oparciu o identyfikatory encji A i B.

Piąta sytuacja jest przykładem wykorzystania encji asocjacyjnej. Jest to alternatywny, bardziej zalecany sposób wyrażenia związku Wiele:Wiele, który daje ponadto możliwość zdefiniowania dodatkowych atrybutów asocjacji.

W modelu szóstym widoczny jest rezultat dwustopniowego mapowania klucza obcego. Konstrukcja składa się z encji mocnej A, dwóch encji słabych B i C oraz łączących je związków 1:0..N (A–B) i 1:0..N (B–C). Analiza związku pomiędzy A i B prowadzi do wygenerowania w schemacie relacyjnym klucza obcego w tabeli B, którym jest atrybut a1. Ponieważ encja B jest słaba, więc kluczem głównym tabeli B będą atrybuty a1 i b1. Z powiązania B–C wynika, że klucz główny B trafi jako klucz obcy do tabeli C. Z tego powodu oba atrybuty tabeli B (a1,b1) są mapowane do tabeli C. Ostatecznie kluczem głównym tabeli C będą łącznie atrybuty a1, b1 i c1.

Następne dwa przykłady demonstrowają przekształcanie zależności nadtyp–podtyp. W przyjętym rozwiązaniu podtypy są traktowane jako encje słabe. Transformacja w modelu siódmym przebiega analogicznie do sytuacji na schemacie trzecim. Ostatni, ósmy diagram, prezentuje działanie systemu w przypadku translacji związków o stopniu 1:0..1. System, oddzielnie dla związku A–B oraz A–C, generuje klucze obce, które są mapowane wzajemnie do obu tabel. Tabele C i B otrzymują klucze obce

a1. Klucze główne w tych tabelach nałożone są na atrybuty c1, a1 (w tabeli C) oraz b1, a1 (w tabeli B). Tabela A otrzymuje więc od tabeli B klucz obcy b1, B_a1 natomiast z tabeli C klucz obcy c1, C_a1. Kluczem głównym tabeli A jest a1.

W ramach projektowania logicznego zmianie ulegają również typy atrybutów i niekiedy ich własności. Jest to spowodowane potrzebą dostosowania atrybutów do specyfiki dedykowanego RDBMS. Atrybuty, które stają się częścią klucza głównego, automatycznie otrzymują charakter obligatoryjny.

Wygenerowany schemat relacji można poddawać dalszej edycji, której celem jest uszczegółowienie danych a także sformułowanie dodatkowych ograniczeń (np. realizacja wzajemnego wykluczania). Aby zachować integralność pomiędzy modelami semantycznym i relacyjnym, zakres dozwolonych modyfikacji został zawężony. System nie pozwala dodawać i usuwać tabel, atrybutów ani zmieniać ich nazw. Dostęp do właściwości tabel uzyskiwany jest (podobnie jak w modelu związków encji) po dwukrotnym kliknięciu na wybrany obiekt.

W zakładce **Atrybuty** (rysunek 10) wprowadzić można dla każdego atrybutu wartość domyślną, maskę, nazwę źródła wartości i ograniczenie kolumnowe (zapisane w języku SQL). Gdy ograniczenie ma dotyczyć większej liczby atrybutów (w tym także pochodzących z innych tabel), możliwe jest tworzenie ograniczeń tablicowych.

Nazwa Atrybutu	Typ	Arg1	Agr2	Maska	Wartość p...	Źródło ...	Wym...	Unikat...
dataWyst	date	brak	brak	ddd/mm/rr	bieżąca da...	RDBMS	☐	☐
nrF	integer	brak	brak		1	USER	☒	☒
pesel	varchar2	11	brak			USER	☒	☐
nrPrac	integer	brak	brak			USER	☒	☐

Buttons: Dodaj/Z..., Usun Ogr, ☐ Pokazuj tablicowe

Ok

Rys. 10. Atrybuty tabeli

Zakładka **Klucze** (rysunek 11) udostępnia informacje o kluczach tabeli. Symbole G, O, K oznaczają odpowiednio klucz główny, obcy i kandydujący. Kliknięcie na dowolny klucz powoduje wyświetlenie listy jego składników i nazwy tabeli powiązanej (tylko dla klucza obcego).

Lista kluczy:				Skladniki klucza:
Nazwa klucza	G	O	K	Skladniki Klucza
PK1zamK1	☒	☐	☐	pesel
FK2zamowieni	☐	☒	☐	

Nazwa tabeli powiazanej: klient

Ok

Rys. 11. Klucze tabeli

3.2.3. GENEROWANIE SKRYPTÓW SQL

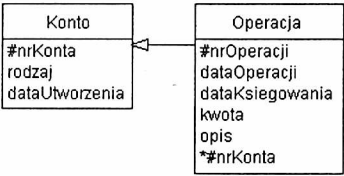
Zapis schematu tabel w języku definicji danych (ang. *Data/Database Definition Language*), zrozumiałym dla RDBMS, jest postrzegany jako element kolejnej fazy projektowania bazy danych, zwanej poziomem fizycznym modelowania. Rezultatem tej fazy są pliki zawierające kod SQL. Skrypty SQL mogą być generowane w kilku odmianach ze względu na zróżnicowanie RDBMS i dla uwzględnienia ich odrębnych cech.

Dostępnym w systemie wariantem kodu SQL jest ANSI-SQL 92 (ANS X3.135). Standard ten jest implementowany przez większość RDBMS oferowanych na rynku, między innymi przez: Oracle 7.x, Sybase SQL Server, Interbase 4.0, PostgreSQL.

Przed wygenerowaniem skryptu istnieje możliwość konfiguracji jego zawartości (rysunek 13).

Ustaleniu podlega wielkość znaków. Do kodu mogą być dołączone opcjonalnie ograniczenia dodatkowe, komentarze i nagłówki. Zawartość utworzonego skryptu (w zależności od dokonanego wyboru) jest wyświetlana we wskazanym edytorze tekstowym. Dzięki temu można sporządzić wydruk, a także zmodyfikować kod w obszarach, których zakres systemu nie obejmuje.

Przykład skryptu i równoważnego diagramu modelu relacyjnego został zaprezentowany na rysunku 12.



```

__*****
— Tytul           : Konto bankowe
— Wersja          : 1.0
— Nazwa RDBMS    : zgodny z ANSI SQL
— Data utworzenia : 2003-03-09
__*****

__*****
—Tabela: Konto
__*****
CREATE TABLE Konto(
    nrKonta varchar2(19) NOT NULL,
    rodzaj integer NOT NULL,
    dataUtworzenia date NOT NULL );

__*****
—Tabela: Operacja
__*****
CREATE TABLE Operacja(
    nrOperacji integer NOT NULL,
    dataOperacji date NOT NULL,
    dataKsiegowania date,
    kwota number(12,2) NOT NULL,
    opis long,
    nrKonta varchar2(19) NOT NULL);

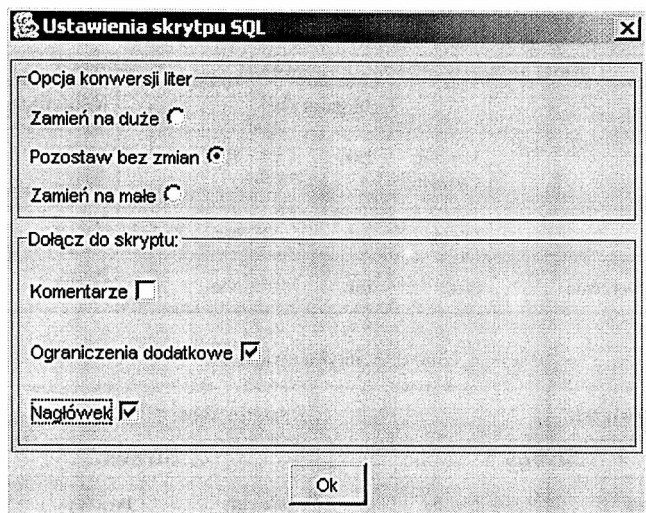
__*****
—Klucz podstawowy: KluczKonta tabeli: Konto
__*****
ALTER TABLE Konto
    ADD CONSTRAINT KluczKonta PRIMARY KEY(nrKonta);

__*****
—Klucz podstawowy: KluczOperacja tabeli: Operacja
__*****
ALTER TABLE Operacja
    ADD CONSTRAINT KluczOperacja PRIMARY
KEY(nrOperacji,nrKonta);

__*****
—Klucz obcy: FK2KluczOperacja tabeli: Operacja w relacji z tabela:
—Konto
__*****
ALTER TABLE Operacja
    ADD CONSTRAINT FK2KluczOperacja FOREIGN
KEY(nrKonta) REFERENCES Konto ( nrKonta);

```

Rys. 12. Skrypt SQL oraz odpowiadający mu model relacyjny w postaci diagramu



Rys. 13. Opcje kodu SQL

3.2.4. RAPORTOWANIE

Raportowanie jest ważnym składnikiem uzupełniającym projekt bazy danych. Wytworzone dokumenty mogą służyć wielu osobom i opisywać różne składowe systemu, dlatego ich zawartość powinna być czytelna i uporządkowana.

Przyjęte w systemie rozwiązanie standaryzuje układ i treść raportów. Raporty tworzone są w formie dokumentów HTML. Każdy typ diagramu posiada odrębny zestaw elementów podlegających dokumentowaniu. Dla diagramu związków encji raport obejmuje wykaz definicji encji wraz z atrybutami i wykaz związków. W przypadku diagramu relacji raport zawiera spis definicji tabel i słownik atrybutów. Elementy te mogą występować w raporcie opcjonalnie.

Przykładowe tabele raportu, utworzone za pomocą CASE, są przedstawione na rysunku 14.

Charakterystyka typów encji

Nazwa encji	Konto
Symbol	KO
Semantyka	Informacje o rachunku rozliczeniowo-oszczędnościowym (koncie) klienta banku
Typ encji	Mocna

Atrybuty encji Konto				
Nazwa	Typ	Charakter obligatoryjny	Unikatowość	Jednoznaczny identyfikator
nrKonta	Varying Character	Tak	Tak	Tak
rodzaj	Integer	Tak	Nie	Nie
dataUtworzenia	Date	Tak	Nie	Nie

Charakterystyka związków

Nazwa związku	Operacje_konta		
Strona 1		Strona 2	
Nazwa roli	Realizuje_dla	Nazwa roli	Posiada
Nazwa encji	Operacja	Nazwa encji	Konto
Stopień związku	Wiele	Stopień związku	Jeden
Typ związku	Opcjonalny	Typ związku	Obowiązkowy

Wykaz tabel modelu relacyjnego

Nazwa tabeli		Operacja						
Atrybut	Dziedzina	Ogr.	Struktura (maska)	Charakter oblig.	Wartości unikalne	Ref.	Wartość domyślna	Źródło wart.
NrOperacji	Integer	KG		Tak	Nie			RDBMS
Data Operacji	Date		dd/mm/rr	Tak	Nie		Bieżąca data	User
Data Księgowania	Date		dd/mm/rr	Nie	Nie			RDBMS
Kwota	Number (12,2)			Tak	Nie		0,00	User
opis	Long			Nie	Nie			User
nrKonta	Varchar2 (19)	KG,KO		Tak	Nie	Konto (nrKonta)		User

Słownik atrybutów

Atrybut	Dziedzina	Przynależność do tabeli
NrKonta	varchar2	Konto, Operacja
rodzaj	Integer	Konto
dataUtworzenia	date	Konto
NrOperacji	Integer	Operacja
dataOperacji	date	Operacja
dataKsiegowania	date	Operacja
kwota	number	Operacja
Opis	long	Operacja

Rys. 14. Przykładowe tabele raportu utworzone za pomocą CASE

4. PODSUMOWANIE

W pracy został przedstawiony system informatyczny wspomagający projektowanie relacyjnych baz danych. Opisano architekturę systemu i rolę poszczególnych elementów aplikacji. Pokazano wykorzystanie narzędzia w kolejnych etapach projektowania bazy danych.

Opis funkcjonalności aplikacji jest także przykładem użycia CASE do realizacji projektu. CASE pozwala na uniknięcie szeregu problemów, których eliminacja (lub redukcja niedogodności z nimi związanych) pozwala użytkownikowi systemu skoncentrować się na kluczowych dla budowy poprawnego projektu czynnościach. Diagramy utworzone za pomocą systemu można w łatwy sposób modyfikować a wyniki prac składować na dysku. Nie ma potrzeby ręcznego poprawiania raportów po dokonaniu zmian na diagramie, gdyż zawsze istnieje możliwość utworzenia nowego, aktualnego raportu. System „czuwa” nad utrzymaniem spójności pomiędzy modelami – w tym przypadku model relacyjny jest zawsze zdeterminowany przez model semantyczny. CASE ułatwia weryfikację poprawności diagramu, wskazując niektóre błędy. Narzędzie udostępnia mechanizmy do efektywnego modelowania, zapewniając przy tym zgodność zbudowanego modelu z przyjętą notacją. Korzystanie z systemu narzucające stosowanie zawartych w nim wzorców, choć nie zawsze adekwatnych do potrzeb, sprzyja standaryzacji tworzonego projektu (np. szablony raportów). Zintegrowanie w narzędziu funkcji charakterystycznych dla wczesnych etapów projektowych (edytor diagramów związków encji) jak również usług

znamiennych dla dalszych faz wytwórczych (edytor diagramów relacji, generator kodu SQL) pozwala zaoszczędzić czas potrzebny na dokonywanie kolejnych transformacji i uniknąć związanych z tym błędów.

Przedstawiony CASE nie jest rozwiązaniem kompleksowym. Oferowana funkcjonalność systemu skupia się wokół problemu modelowania danych i tylko w takim zakresie narzędzie to można uznać za użyteczne.

Na zakończenie warto wspomnieć, że dokonując wyboru programu CASE należy wziąć pod uwagę wiele różnorodnych czynników, w tym przede wszystkim zakres oferowanych usług i ich zgodność z oczekiwaniami, koszt, bezpieczeństwo, niezawodność, serwis, a także standardy, które spełnia metodyka projektowa wbudowana w narzędzie.

LITERATURA

- [Adamcze1998] Adamczewski P., *Zintegrowane systemy informatyczne w praktyce*, Mikom, Warszawa, 1998.
- [Barker1996] Barker R. *CASE*Method, Modelowanie Związków Encji*, WNT, Warszawa, 1996.
- [BarLong1996] Barker R, Longman C. *CASE*Method, Modelowanie funkcji i procesów*, WNT, Warszawa, 1996.
- [Beynon1999] Beynon-Davis P. *Inżynieria systemów informacyjnych*, WNT, 1999, Warszawa.
- [Beynon2000] Beynon-Davis P., *Systemy baz danych*. Warszawa, WNT, 2000.
- [Date2000] Date C. J. *Wprowadzenie do systemów baz danych*, WNT, 2000.
- [Frank1997] Frank M. *The evolution of Client/Server CASE*, 1997
<http://www.dbmsmag.com/>
- [Grochow2003] Grochowski L. *Rozproszone systemy informatyczne*, Dom Wydawniczy ELIPSA, Warszawa, 2003.
- [Hebbel1997] Hebbel F. *Using Object Modeling CASE Tools*, 1997
<http://www.dbmsmag.com/>
- [Jaszk1997] Jaskiewicz A. *Inżynieria oprogramowania*. Helion, Gliwice, 1997.
- [Kozacz2002] *Architektury, komponenty i uparty Polak*. Artykuł Jakuba Chabika, ComputerWorld, 2002. <http://www.computerworld.pl/artykuly/27005.html>
- [Martin1985] Martin J. *Structured Techniques for Computing*, Carma McClure, Prentice-Hall, Englewood Cliffs, New Jersey, 1985.
- [Mazur2002] Mazur H., Mazur Z. *Projektowanie baz danych*, PTI, Wrocław, 2002.
- [Roszkow1998] Roszkowski J. *Analiza i projektowanie strukturalne*, Helion, Gliwice, 1998.
- [Schmuller2003] Schmuller J. *UML Ujednolicony Język Modelowania – wyrażanie związków między klasami w projektowaniu obiektowym*, Helion, Gliwice, 2003.
- [Szejko2002] Szejko S. *Metody wytwarzania oprogramowania*, MIKOM, Warszawa, 2002.
- [Wrycza1999] Wrycza S. *Analiza i projektowanie systemów informatycznych zarządzania. Metodyki, techniki, narzędzia*. WNT, Warszawa, 1999.
- [Yourdon1996] Yourdon E. *Współczesna analiza strukturalna*, WNT, Warszawa, 1996.

Dodatek 1. Wykaz narzędzi i producentów CASE

Nazwa produktu CASE	Informacje o producencie
System Architect	Popkin Software and Systems Inc. www.popkin.com
Oracle Designer	Oracle Corporation, www.oracle.com
Power Designer	Sybase Inc. www.sybase.com
Stp	Aonix www.aonix.com
Case/4/0	Mirco Tool www.mircotool.de
EasyER	Evergreen Software Tools Inc. www.esi.com
ParadigmPlus	Platinum Technology Inc. www.platinum.com
Select SE	Select Software Tools Inc. www.selectbs.com
VisibleAnalyst Workbench, DatabaseEngineer	Visible Systems Corp. www.visible.com
Rational Rose Data Modeler	Rational Software Corp. www.rational.com
4Keeps	A.D. Experts www.adexperts.com
Data Modeler	Iconix SoftwareEngineering Inc. www.iconixsw.com
ER/Studio	Embarcadero Technologies Inc. www.embarcadero.com
Erwin/ERX, ORCompas	Logic Works Inc. www.logicworks.com
SilverRun Enterprise	SilverRun Technologies Inc. www.silverrun.com

RELATIONAL DATABASE DESIGN USING CASE TOOLS

The work characterizes CASE tools and points to the areas of their application in relational database design. A sample CASE application is presented, supporting the structural method of database design. This application has been implemented as an M.Sc. thesis at the Department of Computer Science of the Wrocław University of Technology.

**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 83-7085-764-7