Maria Chałon

SYSTEMY BAZ DANYCH

Wprowadzenie

Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2001

PRZEDMOWA

Nie ulega wątpliwości, że systemy baz danych zajmują bardzo ważne miejsce we współczesnych metodach tworzenia systemów informacyjnych. Właściwie zapisane i przekazane informacje powiększają wspólną wiedzę, dlatego sprawą dużej wagi jest sposób ich zapisywania. Język naturalny, tak wygodny w przypadku swobodnego porozumiewania się ludzi, nie jest dostatecznie precyzyjnym narzędziem zapisu i przechowywania informacji w komputerze. Odpowiednio zebrane i zestrukturalizowane informacje tworzą modele danych będące podstawą tworzenia systemów baz danych. Właśnie to nastawienie na praktyczną stronę zagadnienia odróżnia tę książkę od innych, bardziej teoretycznych podręczników o tej tematyce. Publikacja ta powstała jako rozszerzona wersja treści zajęć prowadzonych dla studentów kierunku Informatyka. Podstawowym celem autorki było łączenie wiedzy teoretycznej leżącej u podstaw systemów baz danych z praktycznym wykorzystaniem opisywanych metod. W związku z tym układ książki jest następujący:

1. Część I stanowi wprowadzenie do II, III i IV części książki i zawiera opis podstawowych cech bazy, krótką historię baz danych oraz definicję podstawowych pojęć, którymi operuje się w kolejnych częściach podręcznika lub które stanowią istotną dla dalszych rozważań informację. W ostatnim rozdziale części pierwszej opisano metodologię projektowania baz danych.

2. W części drugiej po krótkim wprowadzeniu przedstawiono wybrane metody reprezentacji danych, takie jak model relacyjny, model obiektowy oraz dedukcyjny. Część ta zawiera opis przedstawianych modeli.

3. W części trzeciej podano przykłady implementacji wybranych systemów. Dla modelu relacyjnego wprowadzono nieproceduralny język czwartej generacji SQL, ilustrując jego własności licznymi przykładami. Aplikacja o nazwie Ośrodek Wypoczynkowy jest przykładem bazy obiektowej. Jako przykład bazy dedukcyjnej posłużył opis Laboratorium Medycznego.

4. W części czwartej szczególny nacisk położono na architekturę rozproszonych baz danych. Jako przykład narzędzia do realizacji rozproszonych pod względem funkcji i danych baz posłużył system Informix. W rozdziale tym nie mogło zabraknąć również sposobu prezentacji danych w Internecie.

Satysfakcjonujące połączenie wiedzy teoretycznej z praktycznym jej wykorzystaniem jest zawsze niezwykle trudne. W tym wypadku mamy do czynienia z wieloma problemami dotyczącymi baz danych i z potrzebą wybrania tych, które w istotny i znaczący sposób reprezentują aktualną wiedzę na temat baz, a także mają szansę rozwoju i wykorzystania w przyszłości.

Zawarty w książce materiał zainteresuje nie tylko studentów informatyki zajmujących się teorią i praktycznym tworzeniem baz danych w systemach informatycznych, lecz także tych wszystkich informatyków, którzy w swojej pracy zawodowej zajmują się przetwarzaniem danych.

CZĘŚĆ I

Wprowadzenie do problematyki baz danych

Projektowanie bazy danych powinno rozpoczynać się zawsze od analizy informacji, które mają znaleźć się w bazie i powiązań istniejących między nimi. W wyniku wstępnej fazy prac powstaje schemat pojęciowy, stanowi on model informatyczny rozważanego systemu informacji.

C. Delobel, M. Adiba

1. WSTĘP

Rozpowszechniło się błędne przekonanie, że bazy danych i systemy zarządzania bazami danych dotyczą błahej dziedziny zapamiętywania i uzyskiwania danych. Być może sprawił to angielski wyraz *datum* wywodzący się z łaciny i oznaczający dosłownie fakt. Dane jednak nie zawsze odpowiadają faktom. Czasami nie są sprecyzowane lub opisują coś, co się nigdy nie zdarzyło, np. ideę. Okazuje się jednak, że problem nie tkwi w danych i sposobie ich zapisu, ale w ich interpretacji. Interpretacja danych może być zawarta w samych danych lub w programach, które z nich korzystają. Dopiero dane i ich interpretacja tworzą pełny obraz wycinka rzeczywistości, którą chcemy opisać za pomocą komputera. Jest oczywiste, że potrzebujemy na tyle abstrakcyjnej interpretacji, aby tolerowała ona pewne zaburzenia świata rzeczywistego, a jednocześnie była dostatecznie ścisła, dając pojęcie o tym, jak dane są wzajemnie powiązane. Konstrukcję pojęciową zapewniającą taką interpretację nazywamy **modelem danych**. Podstawowe własności opisywanej rzeczywistości należą do dwóch klas: klasy **własności statycznych** i klasy **własności dynamicznych**.

Własności statyczne, zwane schematem bazy danych, są stałe, niezmienne w czasie. Schemat odpowiada temu, co nazywamy zazwyczaj językiem definiowania danych (ang. *Data Definition Language – DDL*). Język ten definiuje dopuszczalne struktury danych w ramach danego modelu. To znaczy, że określa własności, które muszą być prawdziwe dla wszystkich wystąpień bazy danych określonego schematu. Są dwa uzupełniające się sposoby definiowania struktur danych:

1. Dozwolone obiekty i związki specyfikuje się za pomocą ogólnych reguł definiowania dla kategorii, do której należą.

2. Niedozwolone obiekty i związki wyklucza się definiując więzy, czyli nakładając ograniczenia na kategorię.

Własności dynamiczne, zwane stanem bazy danych, są wyrażone zbiorem operacji odpowiadających językowi manipulowania danymi (ang. *Data Manipulation Language – DML*). W zbiorze tym zdefiniowane są dozwolone operacje, które mogą być wykonywane na pewnym wystąpieniu bazy danych D_1, w celu otrzymania wystąpienia D_2. Przykładem tego typu operacji są np. operacje aktualizacji. Aktualizacja zmienia bazę danych z jednego stanu w drugi. Nowy stan jest wprowadzany przez stwierdzenie faktów, które stają się prawdziwe lub przez zaprzeczenie faktów, które przestają być prawdziwe. Do tej klasy operacji należą również takie operacje, które nie powodują zmiany w wystąpieniu, a mimo to są dynamiczne, gdyż powodują zmianę stanu bazy danych, np. zapytania. Zapytanie nie modyfikuje w żaden sposób bazy danych, ale jest używane głównie do sprawdzania czy pewien fakt lub grupa faktów jest speł-

niona w danym stanie bazy danych. Funkcje zapytań mogą być również używane do wyprowadzania danych z ustalonych faktów.

Model danych jest integralną częścią Systemu Zarządzania Bazą Danych (SZBD), czyli takiego systemu, który dostarcza zarówno mechanizmów do definiowania schematów baz danych, jak i operacji, które można stosować do przekształcenia jednej bazy w inną, oczywiście o tym samym schemacie. Dokładnie mówiąc, SZBD spełnia trzy zasadnicze funkcje: zarządza plikami, wyszukuje informacje oraz zarządza całą bazą danych, czyli stanowi jakby powłokę, która otacza bazę danych i za pomocą której dokonują się wszystkie operacje na danych. Sama baza danych jest magazynem danych z nałożoną na niego wewnętrzną strukturą.

Z bazą też są powiązane inne właściwości. Należą do nich:

– *współdzielenie danych*, czyli możliwość korzystania przez wielu użytkowników w tym samym czasie z tych samych danych,

– *integracja danych*, czyli utrzymywanie i administrowanie bazą nie zawierającą niepotrzebnie powtarzających się lub zbędnych danych,

– *integralność danych*, czyli baza danych musi dokładnie odzwierciedlać obszar analizy, którego ma być modelem, co oznacza, że istnieje odpowiedniość między faktami przechowywanymi w bazie a modelem rzeczywistym i każda zmiana występująca w modelu rzeczywistym musi być wprowadzona w bazie opisującej ten model,

– *bezpieczeństwo danych*, czyli ograniczenie dostępu w celu zapewnienia integralności bazy. Problem bezpieczeństwa jest bardzo ważny i zostanie szczegółowo omówiony w dalszych rozdziałach.

– *abstrakcja danych*, czyli możliwość przechowywania pewnych istotnych do określonych potrzeb właściwości obiektów i związków między nimi.

– *niezależność danych*, czyli oddzielenie danych od procesów, które używają tych danych. Celem jest osiągnięcie sytuacji, w której organizacja danych byłaby niewidoczna dla użytkownika i programów użytkowych korzystających z tych danych. Również dąży się do tego, aby żadna zmiana programu aplikacyjnego nie miała wpływu na strukturę danych.

Podane właściwości stanowią na ogół pożądane cechy idealnej bazy danych. W rzeczywistości większość modeli baz danych nie spełnia wszystkich tych cech.

2. KRÓTKA HISTORIA BAZ DANYCH

Pierwsze bazy danych powstały w latach sześćdziesiątych, kiedy to duże firmy komputerowe, takie jak General Electric Company, tworzyły konkretne SZBD. Pojawiły się wtedy pierwsze pakiety danych do użytku ogólnego [18, 20].Tradycyjna baza danych służyła wyłącznie do przechowywania danych. Ich przetwarzanie wykonywane było przez programy użytkowe współpracujące z SZBD. Obsługa tych programów była tak skomplikowana, że wymagała obecności programisty o dużych kwalifikacjach. W latach sześćdziesiątych stworzono sieciowy system zarządzania IDMS [13, 18] działający na dużych komputerach IBM, który przez następne dwadzieścia lat był potentatem wśród SZBD. System ten miał duży wpływ na stworzenie przez grupę Codasyl [12] modelu będącego standardem w dziedzinie sieciowych modeli baz danych. Jednak prawdziwą rewolucję wywołał w roku 1970 naukowiec z firmy IBM dr E.F. Codd swoją pracą *A Relational Model for Large Shared Data Banks* [6], która miała decydujący wpływ na rozwój architektury baz danych. Idea przedstawiona przez Codda stała się podstawą do stworzenia pierwszych relacyjnych baz danych. Swój szybki rozwój i olbrzymią popularność zawdzięczają bazy relacyjne przede wszystkim sposobowi zapisu danych za pomocą tabel zwanych relacjami oraz możliwością bezpośredniego zaimplementowania tego typu SZBD na komputerach osobistych [7, 8, 13]. Duża popularność baz relacyjnych w wielu komercyjnych zastosowaniach nie wyklucza jednak faktu, że bazy te obecnie przestają być narzędziem wystarczającym do opisu wielu problemów. Model relacyjny oferuje bardzo elastyczny sposób reprezentowania faktów i formułowania zapytań o te fakty, ale nie oferuje łatwego sposobu reprezentowania więzów integralności i modyfikacji. Więzy te i funkcje modyfikacji są zazwyczaj implementowane w programach użytkowych działających na zewnątrz bazy relacyjnej. Dąży się do tego, aby baza danych nie stanowiła jedynie miejsca przechowywania faktów i informacji, ale miała wbudowaną „inteligencję". Tak więc zaczęły powstawać dedukcyjne bazy danych [2, 3, 4, 10]. Dedukcyjne bazy danych oferują notację o dużej ekspresji, służącą do reprezentowania faktów, więzów integralności, zapytań i modyfikacji. Obecnie ich wadą jest brak wydajnych implementacji dużych baz faktów. W ostatnich latach pojawiło się zwiększone zapotrzebowanie na nowe typy systemów baz danych opartych na pojęciu obiektowości [4, 15, 17]. Nie tylko powstało wiele komercyjnych obiektowych SZBD, lecz również wielu producentów relacyjnych systemów wyposaża swoje produkty w cechy charakterystyczne dla baz obiektowych. Pomysł zastosowania równoległej architektury komputerów przy problemach zarządzania bazami danych ma również wieloletnią historię [4]. Widać więc, że wielki wpływ na kształt systemów baz danych ma: przetwarzanie wbu-

dowane w bazę danych, obiektowość i równoległość [4]. Istotne miejsce w systemach komputerowych zajmują systemy rozproszone [9], a co za tym idzie pomimo tego, że jest jeszcze miejsce na scentralizowane, duże bazy danych specjalizujące się w realizacji wielkiej liczby transakcji, wiele osób uznało lata dziewięćdziesiąte za erę rozproszonych baz danych. Ostatnio można również zaobserwować gwałtowny wzrost prezentacji różnego typu informacji dzięki dynamicznie rozwijającej się sieci Internet.

3. POJĘCIA PODSTAWOWE

Wprowadzimy obecnie pewne pojęcia, którymi będziemy posługiwać się w dalszej części podręcznika. Należą do nich:

Dana (nazwa, cecha, wartość) – podstawowy składnik informacji, który na ogół stanowi zbiór znaków będących literami, cyframi lub znakami interpunkcyjnymi.

Encja – każdy przedmiot, zjawisko, stan, pojęcie, każdy obiekt, który potrafimy odróżnić od innych obiektów. Encja może być określona przez: <nazwa obiektu, cecha obiektu, wartość obiektu, czas>. Czas jest bardzo niewygodnym czynnikiem przy modelowaniu. Częściej pomija się czas i wprowadza się uporządkowanie zdarzeń według kolejności ich występowania.

Powiązania między danymi jedno-jednoznaczne ($1 \leftrightarrow 1$). Mamy dane dwa zbiory encji A i B. Każdej encji ze zbioru A odpowiada najwyżej jedna encja ze zbioru B i na odwrót (nie wszystkie encje obu zbiorów muszą być w związku).

Powiązania między danymi wielo-jednoznaczne ($n \twoheadleftarrow\!\!\rightarrow 1$) Mamy dane dwa zbiory encji A i B. Każdej encji ze zbioru A odpowiada najwyżej 1 encja ze zbioru B, chociaż 1 encji ze zbioru B może odpowiadać wiele encji ze zbioru A (**jedno--wieloznaczne** ($1 \leftrightarrow\!\!\!\!\rightarrow m$) – symetrycznie do poprzedniego).

Powiązania między danymi wielo-wieloznaczne ($m \twoheadleftarrow\!\!\!\!\rightarrow n$) są najbardziej złożonym sposobem łączenia obiektów danych przy projektowaniu bazy. Każdy element jednego obiektu jest związany z kilkoma elementami innego obiektu i na odwrót.

Klucz – wyróżniony atrybut lub minimalny zbiór wyróżnionych atrybutów.

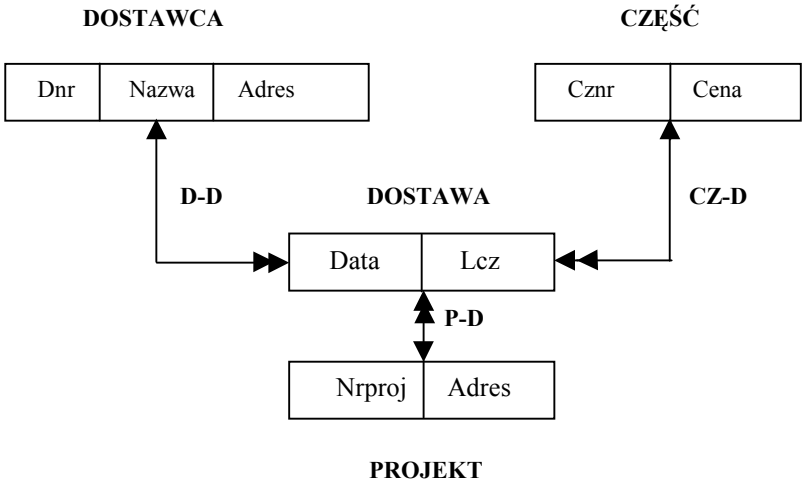
Klucz kandydujący – wyróżniony atrybut lub minimalny zbiór wyróżnionych atrybutów, który w sposób jednoznaczny identyfikuje daną.

Klucz główny – klucz wybierany ze zbioru kluczy kandydujących.

Język Definiowania Danych (DDL) – zbiór reguł wyrażających własności statyczne danych. DDL definiuje dopuszczalne struktury danych (obiekty i zależności między nimi), tworząc schemat bazy danych.

• *Przykład 3.1*

Mamy dany schemat prostej sieciowej bazy [18] **ZAOPATRZENIE** przedstawionej na rys. 3.1.



gdzie: **D-D**, **CZ-D**, **P-D** powiązania jedno-wieloznaczne

Rys. 3.1. Diagram schematu **ZAOPATRZENIE**

Baza składa się z czterech rekordów: **DOSTAWCA, CZĘŚĆ, DOSTAWA, PROJEKT** opisanych za pomocą wyróżnionych cech oraz trzech powiązań (set D-D, set CZ-D i set P-D) pomiędzy rekordami, które zawierają informacje o tym, który z rekordów jest rekordem nadrzędnym (ang. *owner*), a który podrzędnym (ang. *member*). Język DDL dla schematu **ZAOPATRZENIE** ma nastepująca postać:

Data base **ZAOPATRZENIE**
Record **DOSTAWCA**
Dnr *string 2,key*;
Nazwa *string 20*;
Adres *string 20*;
End **DOSTAWCA.**
Record **CZĘŚĆ**
Cznr *string 2,key*;
Cena *real*;
End **CZĘŚĆ.**
Record **PROJEKT**
Nrproj *string 2,key*;
Adres *string 20*;
End **PROJEKT.**
Record **DOSTAWA**
Data *integer*;
Lcz *integer*;

```
    End DOSTAWA.
    Set D-D
        owner DOSTAWCA;
        member DOSTAWA;
    End D-D.
    Set CZ-D
        owner CZĘŚĆ;
        member DOSTAWA;
    End  CZ-D
    Set P-D
        owner PROJEKT;
        member DOSTAWA;
    End P-D.
End ZAOPATRZENIE.
```

Język Manipulowania Danymi (DML) – zbiór operacji określających dynamiczne własności bazy danych, czyli stan bazy danych.

• *Przykład 3.2*

Dla podanego wyżej schematu bazy **ZAOPATRZENIE** mamy za zadanie podać adresy realizacji tych projektów, dla których dostarczono **CZĘŚĆ** o numerze 100 (Cznr = 100). Przykład realizacji zapytania w DML:

```
            FIND FIRST CZĘŚĆ RECORD WHERE (CZĘŚĆ.Cznr=100)
            IF ( STAN=0) GO TO KONIEC
            CZ-D:=CZĘŚĆ
NASTCz: FIND NEXT CZ-D SET
             IF ( STAN=0)  GO TO KONIEC
            P-D:=CZ-D
            FIND OWNER P-D SET
             GET P-D.Adres
            GO TO NASTCz
KONIEC: STOP
```

Podany przykład pokazuje użycie języka proceduralnego, w którym mamy do czynienia z wyszukiwaniem jednostkowym. To samo zadanie rozwiązane za pomocą wyszukiwania grupowego z wykorzystaniem języka RALAN (model bazy relacyjny) pokazuje przykład 3.3.

• *Przykład 3.3*

Przykład realizacji zapytania w języku Ralan [18].

1. Określenie schematu bazy danych za pomocą nagłówków:

DOSTAWCA (Dnr, Nazwa, Adres)
CZĘŚĆ (Cznr, Cena)
PROJEKT (Nrproj, Adres)
DOSTAWA (Dnr, Cznr, Nrproj, Data, Lcz)

2. Tabele relacji:

DOSTAWCA:

Dnr	Nazwa	Adres
10	ETO	WROCŁAW
11	MTO	WROCŁAW
12	ZTO	WARSZAWA
13	WPP	KRAKÓW

DOSTAWA:

Dnr	Cznr	Nrproj	Data	Lcz
10	100	2000	15.09.2000	15
10	150	2000	15.09.2000	20
12	100	2001	20.01.2001	30
11	150	2002	20.09.2001	10

3. Program:

a) *R*1 ← DOSTAWA (Cznr=100)

*R*1:

Dnr	Cznr	Nrproj	Data	Lcz
10	100	2000	15.09.2000	15
12	100	2001	20.01.2001	30

b) *R*2 ← R1 [Dnr]

*R*2:

Dnr
10
12

c) *R*3 ← *R*2 | **DOSTAWCA** (gdzie | – znak łączenia relacji)

*R*3:

Dnr	Nazwa	Adres
10	ETO	WROCŁAW
12	ZTO	WARSZAWA

d) LIST *R*3 [Adres]

Wynik:

Adres
WROCŁAW
WARSZAWA

Język Zarządzania Danymi (DCL) jest to zbiór komend stworzonych do zapewnienia bezpieczeństwa i spójności danych. Można je podzielić na dwie grupy:

– potwierdzanie i odwoływanie transakcji blokowania dostępu do tablic,

– nadawanie i odwoływanie przywilejów dostępu do zasobów bazy danych.

• *Przykład 3.4*

Pokazano nadawanie przywilejów do zasobów bazy **ZAOPATRZENIE** użytkownikowi o nazwie **EWA.**

ogólna postać komend:	
GRANT {uprawnienia,…./ALL}	GRANT **SELECT**
ON {nazwa obiektu}	ON **ZAOPATRZENIE**
TO {nazwa użytkownika …}	TO **EWA**

Transakcja jest to zdarzenie powodujące zmianę stanu bazy danych. Nowy stan jest wprowadzany przez stwierdzenie faktów, które stają się prawdziwe i (lub) przez zaprzeczenie faktów, które przestają być prawdziwe. Każda transakcja powinna mieć właściwości: niepodzielności, spójności, izolacji i trwałości (ang. ***a****tomic,* ***c****onsistency,* ***i****solation,* ***d****urability*, stąd określenie cech transakcji ACID). Niepodzielność transakcji jest rozumiana jako jednoznaczne jej zakończenie: zatwierdzenie lub anulowanie. Spójność to przeprowadzenie systemu z jednego stanu spójnego do innego również spójnego. Izolacja to wykonanie transakcji w sposób nie kolidujący z innymi realizowanymi transakcjami. Trwałość wyniku transakcji polega na zapisywaniu w pamięci stałej systemu wyników, co chroni je przed awarią procesu serwera. Z punktu widzenia aplikacji, transakcja to zbiór kolejnych instrukcji nieproceduralnego języka manipulacji danymi o nazwie SQL (rozdz. 8). Zbiór ten zakończony jest instrukcją COMMIT WORK (zatwierdzenie transakcji) lub ROLLBACK (anulowanie transakcji). W razie awarii systemu automatycznie realizowane jest anulowanie transakcji.

Baza danych – zbiór danych zorganizowanych w pewien logiczny i zestrukturalizowany sposób. Bieżąca struktura zależy od modelu danych przyjętego przy organizowaniu tych danych. Jej wielkość zależna jest od liczby danych i od wzajemnych powiązań między nimi.

Sformatowana baza danych – zbiór danych w postaci skończonego zbioru wzorców (schematów, formatów) służących do wyrażenia pewnych informacji o stanie świata rzeczywistego. Zakres odwzorowanej wiedzy nie powinien być szeroki.

Niesformatowana baza danych – zbiór faktów i reguł tworzących nowe fakty na podstawie istniejących w postaci sieci semantycznych, wyspecjalizowanych języków opisu wiedzy. Zakres odwzorowania wiedzy może być bardzo szeroki.

System Zarządzania Bazą Danych (SZBD) jest zorganizowanym zbiorem narzędzi umożliwiającym dostęp do baz danych i zarządzanie nimi. Dzięki SZBD dostępne są takie operacje, jak: przechowywanie danych, tworzenie i utrzymywanie struktur danych, umożliwienie równoczesnego dostępu wielu użytkownikom, wprowadzanie mechanizmu bezpieczeństwa i prywatności, odzyskiwanie danych i operowanie na przechowywanych danych, wprowadzanie i ładowanie danych, udostępnienie wydajnych mechanizmów indeksowania pozwalających na szybkie znalezienie wybranych danych, zapewnienie spójności różnych rekordów, ochrona przechowywanych danych przed utratą za pomocą kopii bezpieczeństwa i procedur odtwarzania.

Aby spełnić te wymagania, stworzono rożne typy SZBD [13, 18, 19, 20]. Oparto je na następujących modelach:

hierarchicznym – w modelu tym dane przechowywane są w postaci struktury drzewiastej (obecnie jest to system przestarzały);

sieciowym – w modelu tym dane zapisane są w postaci rekordów i powiązań między rekordami, które na równi z danymi są nośnikami informacji (w przykładzie 3.1 powiązanie określone jest słowem *set*). Systemy sieciowe są bardzo szybkie i efektywnie wykorzystują pamięć masową. Pozwalają na tworzenie złożonych struktur danych, są jednak bardzo mało elastyczne i wymagają żmudnego projektowania. Przykładem takiego systemu jest system rezerwacji biletów lotniczych;

relacyjnym – w modelu tym dane mają prawdopodobnie najprostszą strukturę jaką może mieć baza, czyli tabelę. Są łatwe w użyciu i bardzo popularne. Przykładem są systemy: ORACLE, INFORMIX, SYBASE;

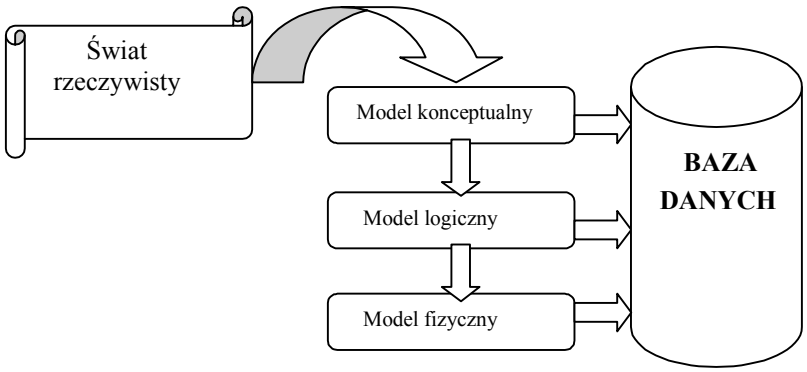
obiektowym – w modelu tym przechowywane i obsługiwane są obiekty takie, jak obrazki, zdjęcia, filmy video. Przykładem może być SZBD ORACLE 8. Baza ta w tabelach przechowuje obiekty.

4. METODOLOGIA PROJEKTOWANIA BAZ DANYCH

4.1. Wprowadzenie

Ogólnie można powiedzieć, że w bazie danych zawarta jest wiedza odnosząca się do pewnego wydzielonego fragmentu świata rzeczywistego. Ustalenie związków pomiędzy danymi w bazie i faktami w świecie rzeczywistym, a więc ustalenie semantyki danych nie odbywa się bezpośrednio. Pomostem umożliwiającym ustalenie tych związków jest model konceptualny [19, 20] bazy danych. Aby dane mogły dostarczać informacji, ich organizacja powinna umożliwiać efektywne przetwarzanie. Istnieją różne sposoby organizowania danych: tablice, listy, formuły itp. Modelując dane, staramy się organizować je tak, aby wiernie odzwierciedlały sytuację rzeczywistą i jednocześnie, aby można je było zapisać w pamięci komputera. Te dwa wymagania nierzadko są sprzeczne. Aby móc określić najlepszą organizację informacji w danym zastosowaniu, musimy poznać jej cechy charakterystyczne. Cechy te pozwolą sformułować ogólne stwierdzenie co do sposobu zorganizowania i przetwarzania danych. Niesprzeczny, formalny zbiór takich stwierdzeń definiuje model danych.

Można wyróżnić trzy zasadnicze etapy konstruowania modelu: model konceptualny, model logiczny i model fizyczny (rys. 4.1).



Rys. 4.1. Modele bazy danych

Model konceptualny jest modelem świata rzeczywistego. Najbardziej znanym modelem tego typu jest model danych związków encji Chena [5]. Modelowanie koncep-

tualne jest etapem poprzedzonym analizą wymagań, czyli wiąże się z uzyskiwaniem od użytkowników początkowego zbioru informacji i wymagań dotyczących przetwarzanych danych. W myśl metody zaproponowanej przez Chena [3, 19] i szeroko stosowanej w teorii i praktyce baz danych do podstawowych faktów rozpatrywanych w świecie rzeczywistym, o których wiedza jest reprezentowana w bazie zaliczamy: występowanie obiektów (ang. *entity*), istnienie pomiędzy tymi obiektami wzajemnych powiązań (ang. *relationship*) oraz posiadanie przez obiekty i powiązania określonych wartości (ang. *value*) atrybutów (ang. *attribute*). Rodzaj wiedzy o świecie rzeczywistym jaka powinna być odwzorowana w bazie danych, a więc zaprojektowanie schematu bazy jest jednym z najtrudniejszych i najważniejszych zadań przy projektowaniu bazy. Od poprawnie zaprojektowanego schematu zależy właściwe działanie całego systemu wykorzystującego bazę. Poprawny, to znaczy spełniający następujące wymagania [11, 19, 20]:

– ścisły związek z faktami świata rzeczywistego, tzn. łatwy sposób ich tworzenia i rozumienia,

– kompletność informacji,

– podatność na zmiany, a więc ewolucyjność schematu,

– stabilność, czyli projektowany schemat powinien uwzględniać przewidywalne zmiany,

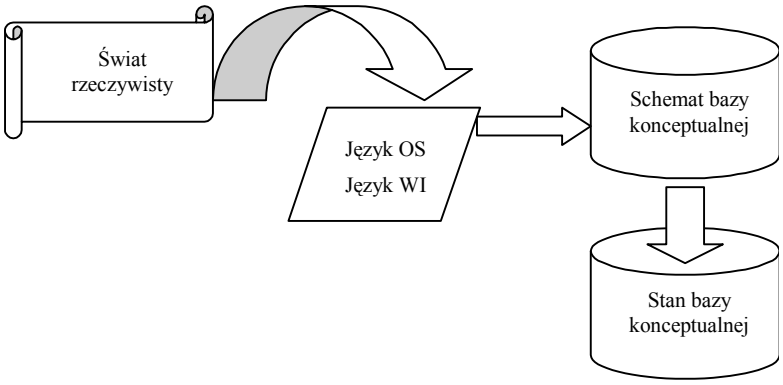
– możliwość tworzenia różnych obrazów danych, czyli różnych logicznych modeli baz danych.

Model logiczny jest to, z punktu widzenia architektury danych, zbiór ogólnych zasad posługiwania się danymi, np. modele klasyczne czy modele semantyczne danych. Proces modelowania logicznego wiąże się z określeniem zawartości bazy danych niezależnie od wymogów konkretnej implementacji fizycznej. Model fizyczny to sposób reprezentacji danych w pamięci komputera wyrażony za pomocą plików, rekordów, struktur dostępu itp. odpowiednich dla określonej konfiguracji sprzętu i oprogramowania.

4.2. Proces projektowania i jego składowe

Proces projektowania obejmuje czynności i zdarzenia występujące między pojawieniem się problemu a powstaniem dokumentacji opisującej rozwiązanie problemu zadawalające z punktu widzenia funkcjonalnego, ekonomicznego i innych wymagań. Ta definicja procesu projektowania podana została przez Kricka [14] w roku 1971 i jest na tyle ogólna, że można ją zastosować w większości przypadków. Oczywiście projektowanie rzadko przebiega według identycznych sieci działań składowych. Ich postać zależy przede wszystkim od klasy projektowanych obiektów, od specyfiki zadania projektowego oraz od wielu różnych cech charakterystycznych dla projektowanych obiektów czy systemów. Inaczej postępujemy na etapie projektowania modelu

konceptualnego, logicznego czy też fizycznego. Można jednak dopatrzyć się pewnych wspólnych cech, to znaczy sekwencji działań podstawowych, do których należą formułowanie problemu, generacja rozwiązań, ocena rozwiązań itp. Oczywiście punktem wyjściowym do sformułowania problemu jest analiza potrzeb i wymagań. W wyniku ogólnej i szczegółowej analizy problemu uzyskujemy pewien zbiór danych. Zbiór ten po „uporządkowaniu" stanowi zadanie projektowe, które zapisane w formie dokumentu według standardów obowiązujących w danym systemie tworzy założenia projektowe. „Uporządkowanie" jest to proces przejścia od faktów w świecie rzeczywistym do danych w bazie danych. Etapy przejścia od świata rzeczywistego do konceptualnej bazy danych przedstawiono na rysunku 4.2.



Rys. 4.2. Tworzenie konceptualnej bazy danych

W etapie pierwszym należy zdefiniować pewien wąski podzbiór języka naturalnego, służący do opisu stanów świata rzeczywistego (w skrócie Język Opisu Stanów – OS) oraz język służący do formułowania więzów integralności (w skrócie Język Więzów Integralności – WI). Celem jest określenie, jakie w ogóle obiekty świata rzeczywistego, jakie powiązania między tymi obiektami i jakie atrybuty (cechy) obiektów są interesujące w danej klasie zastosowań. Przy tak sformułowanym problemie z góry określa się konieczność wykorzystania takich operacji, jak selekcja, klasyfikacja i nazywanie obiektów. Równocześnie związane z tym jest określenie pewnego zbioru warunków zwanych więzami integralności, których spełnienie jest konieczne, aby o danej strukturze można było powiedzieć, że jest ona niesprzeczna ze swoim opisem. Więzy integralności są to wyrażenia, które stwierdzają zachodzenie w świecie rzeczywistym pewnych stałych niezmiennych zależności. Ich źródłem jest znajomość właściwości semantycznych świata rzeczywistego. Więzy integralności są warunkami koniecznymi, ale nie dostatecznymi poprawności bazy danych. Możliwa jest w bazie taka sytuacja, że pomimo spełnienia wszystkich więzów dane w bazie nie odpowiadają żadnemu poprawnemu stanowi świata rzeczywistego. W czasie projektowania bazy danych, przy zastosowaniu konkretnego modelu danych i związanej z nim metody

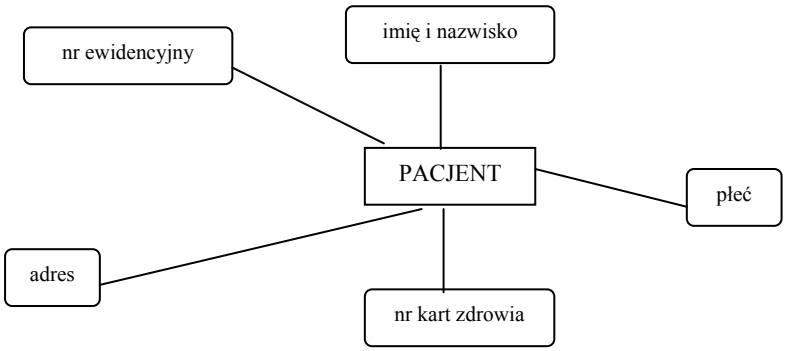
szereg więzów integralności zostaje ujętych w samej strukturze bazy danych. W wielu przypadkach jednak więzy muszą być formułowane w specjalnym języku.

Etap drugi to tworzenie schematu bazy danych. Schemat stanowi zbiór zdań języka opisu stanów prawdziwych względem rozpatrywanego stanu świata rzeczywistego (np. w pewnym przedziale czasu, w którym nie zaszły żadne interesujące nas zmiany) oraz zbiór zdań języka więzów integralności. W każdym z więzów ma swoje odbicie jakieś ograniczenie semantyczne, odzwierciedlające nasze intuicyjne rozumienie pewnej niezmiennej właściwości, jaką ma rozpatrywany przez nas wycinek świata rzeczywistego. Schemat jest stały, niezmienny, każda zmiana to zupełnie nowa baza danych.

W etapie trzecim budowana jest pewna struktura matematyczna, czyli model abstrakcyjny stanu świata rzeczywistego. Model ten na ogół można przedstawić jako strukturę złożoną, w skład której wchodzą zbiory: obiektów, powiązań, wartości, oraz atrybuty i więzy integralności. Manipulowanie w bazie konceptualnej może obejmować operacje wyszukiwania, modyfikowania, dołączania i usuwania. Niektóre z nich, jak np. wyszukiwanie, nie zmieniają stanu bazy i po ich wykonaniu nie występuje konieczność sprawdzania więzów integralności. W przypadku pozostałych może się to okazać konieczne.

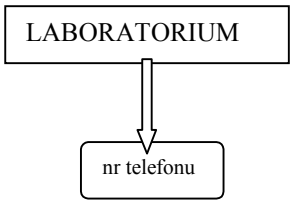
4.3. Opis modelu konceptualnego

Jako przykład modelu konceptualnego posłuży model zwiazków encji Chena. Model ten powstał w wyniku praktycznych doświadczeń przy projektowaniu bazy danych za pomocą dostępnych na rynku SZBD. Zgodnie z propozycją Chena [5] podstawowe fakty rozpatrywane w świecie rzeczywistym są opisywane za pomocą obiektów. Każdy obiekt jest określony poprzez atrybuty mające pewne wartości. Pomiędzy obiektami istnieją powiązania. Obiekty, atrybuty, powiązania i wartości można klasyfikować w zbiory. Podstawą tej klasyfikacji jest posiadanie przez nie pewnej własności określonej dla każdego zbioru [19]. Obiekty reprezentowane są za pomocą wartości atrybutów. Nazwa atrybutu jest na ogół jedną z jego wartości. W celu jednoznacznej identyfikacji obiektu w zbiorze obiektów określamy klucz każdego obiektu. Jest to atrybut lub zbiór atrybutów określony na tym zbiorze, który różnym obiektom w tym zbiorze przyporządkowuje różne wartości. Oznacza to, że wskazanie wartości klucza obiektu jednoznacznie wyznacza obiekt w zbiorze obiektów. Jeden zbiór obiektów może mieć kilka kluczy. Jeżeli jest kilka kluczy, to wybieramy z nich ten, który jest najbardziej sensowny semantycznie i dla którego można przewidzieć dopuszczalny zakres wartości. Nazywamy go kluczem głównym. Wartości, które klucz główny przyporządkowuje obiektom traktujemy jako reprezentacje tych obiektów. Każdy obiekt w modelu Chena nosi nazwę encji. Encje tego samego typu tworzą zbiory encji. Na rysunku 4.3 podano przykład przedstawienia encji PACJENT.



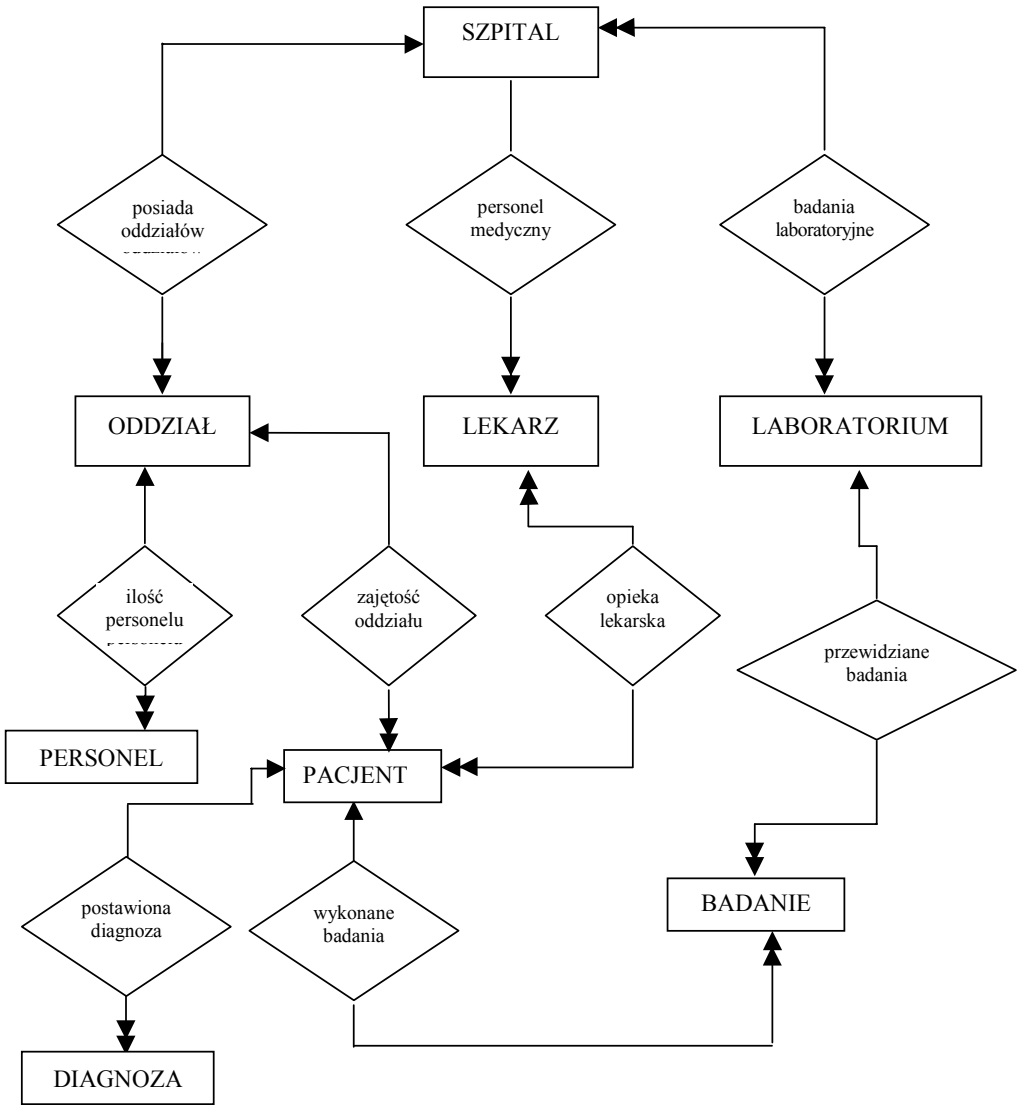
Rys. 4.3. Przedstawienie atrybutów zdefiniowanych na zbiorze encji PACJENT

Atrybuty mogą być jedno- lub wielowartościowe (rys. 4.4).



Rys. 4.4. Atrybut dla encji Laboratorium

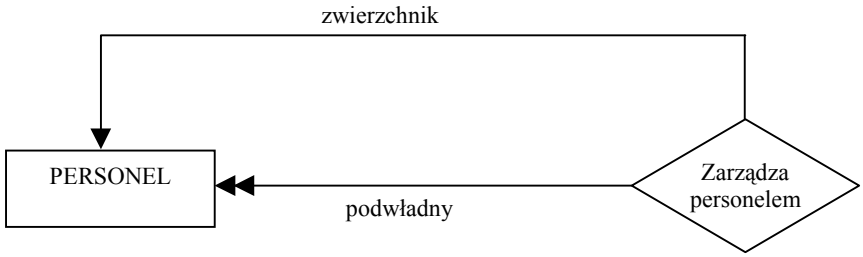
Encje zawierają niewiele informacji. Między zbiorami encji zachodzą pewne związki. Wyróżniamy związki typu 1:1, 1:*n* (*m*:1) i *n*:*m*. Związki na równi z encjami są źródłem informacji. Strukturę bazy danych zorganizowaną zgodnie z modelem związków encji można przedstawić za pomocą diagramu związków encji [19] składającego się z wierzchołków połączonych etykietowanymi krawędziami. Zarówno sposób określenia krawędzi między wierzchołkami jak i przypisane etykiety przekazują istotną informację o własnościach semantycznych odwzorowywanego świata rzeczywistego. Wyróżniamy następujące rodzaje wierzchołków: prostokąty etykietowane nazwami encji oraz romby etykietowane nazwami zbiorów powiązań. Występowanie konkretnego wierzchołka z przypisaną mu etykietą przekazuje informacje, że dany zbiór encji, powiązań lub wartości występuje w opisie świata rzeczywistego. Na rysunku 4.5 [19] pokazano diagram dla medycznej bazy danych. Zbiór takich encji, jak: SZPITAL, ODDZIAŁ, LEKARZ, PERSONEL, LABORATORIUM, PACJENT, BADANIE, DIAGNOZA, reprezentuje na rysunku prostokąt z etykietą. Zbiory encji odpowiadają różnym, ogólnym klasyfikacjom encji. Przynależność do zbioru określa predykat, to znaczy cechy konkretnego wcielenia pewnej encji mogą być testowane w celu ustalenia, czy encja ta należy, czy nie do zbioru encji. Zbiory encji nie muszą



Rys. 4.5. Diagram związków encji

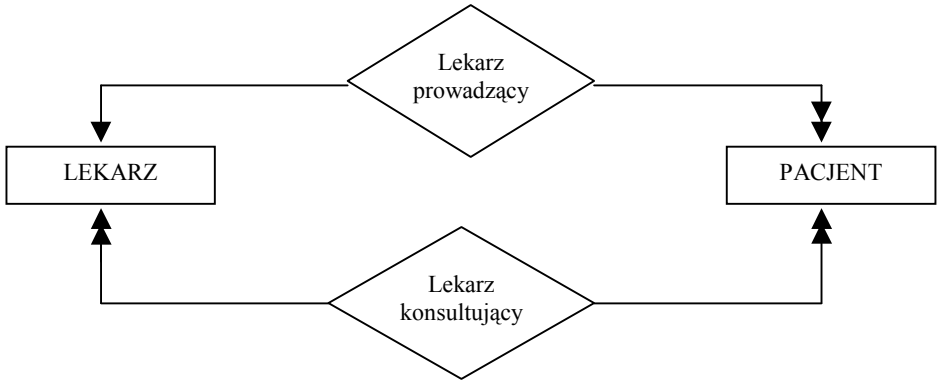
być rozłączne. Konkretna encja może należeć do więcej niż jednego zbioru, np. lekarz może także być pacjentem. Zbiory powiązań są reprezentowane przez romb z etykietą. Zbiory encji należące do zbioru związków są wskazywane przez krawędzie łączące je z opisanym typem związku.

Ta sama encja może pełnić w tym samym związku różne role. Mówimy wtedy o związku rekurencyjnym. Ilustruje to rysunek 4.6.

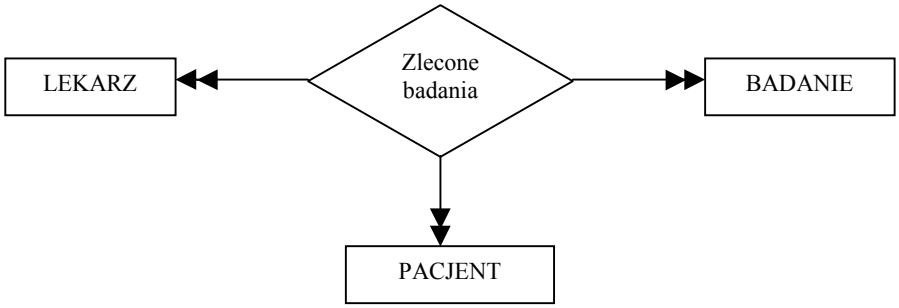


Rys. 4.6. Zbiór związku rekurencyjnego

Może istnieć więcej niż jeden zbiór związku między tymi samymi dwoma zbiorami encji. Przykład podano na rysunku 4.7.



Rys. 4.7. Dwa zbiory związku między tymi samymi zbiorami encji

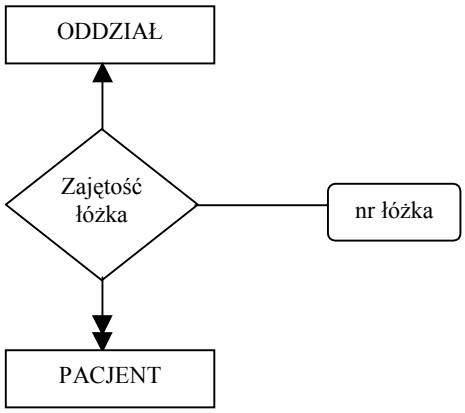


Rys. 4.8. Zbiór związku trzyargumentowego

Zbiór związku może być zbiorem związku *n*-argumentowego (rys. 4.8), to znaczy może dotyczyć *n* zbiorów encji. Diagram przedstawia związek między encjami LEKARZ, PACJENT, BADANIE. Semantyka zbioru wieloargumentowego może być czasem dość trudna do zrozumienia. Na rysunku 4.8 mamy następujące związki:

LEKARZ może zalecić wykonanie różnych BADAŃ kilku PACJENTOM, jedno BADANIE może być zalecane przez kilku LEKARZY różnym PACJENTOM oraz jeden PACJENT może mieć do wykonania kilka BADAŃ zleconych przez różnych LEKARZY.

W modelu związków encji zbiór wartości nazywa się dziedziną. Zbiór wartości może być związany nie tylko ze zbiorem encji, ale również ze zbiorem związku (rys. 4.9). Zbiory wartości przedstawiono na rysunku w postaci prostokątów o zaokrąglonych brzegach.



Rys. 4.9. Atrybut zbioru związku

Zbiory związku także mają klucze zwane kluczami związku. Klucz związku składa się z kluczy encji tych zbiorów encji, które są zawarte w danym zbiorze związku. Oprócz związków występujących w modelu encji istnieją tzw. logiczne ograniczenia zwane więzami. Więzy to pewne własności, które dla danego zbioru encji są prawdziwe lub fałszywe. Inaczej mówiąc, są zachowane lub naruszone. Jeśli wartości danych są zgodne z istniejącą wiedzą o obiekcie, to oczekuje się, że te więzy zostaną zachowane. W modelu danych więzy są szczególnie przydatne wtedy, gdy mają ogólny charakter, to znaczy gdy mogą być definiowane i stosowane w zbiorach obiektów, a nie są ustalone dla konkretnego obiektu. W modelu danych więzy są konieczne ze względu na semantykę i integralność. Semantyka odzwierciedla dokładnie sytuację świata rzeczywistego, a integralność umożliwia SZBD ograniczenie możliwych w danym schemacie stanów bazy do takich, które zachowują więzy. Rozróżniamy dwa podstawowe typy więzów: więzy *wbudowane* [19] oraz więzy *jawne.* Więzy *wbudowane* stanowią bardzo ograniczony mechanizm definiowania więzów, na ogół określają pewne wymagania co do struktury (np. związki w modelu hierarchicznym mogą mieć tylko strukturę drzewa). Czasami zachowanie tych więzów może sprawić, że struktura bazy nie odpowiada wiernemu opisowi świata rzeczywistego. Aby usunąć wady wynikłe z definiowania więzów *wbudowanych*, wprowadzono więzy *jawne*.

Więzy te zapewniają elastyczny mechanizm umożliwiający rozbudowę struktury bazy. Rozgraniczenie między więzami *wbudowanymi* i *jawnymi* jest na ogół płynne i zależy w dużej mierze od struktur modelu danych. Im więcej ograniczeń nakładają struktury modelu danych, tym więcej zawiera on więzów *wbudowanych* i tym mniej trzeba definiować więzów *jawnych.* Definicja więzów *jawnych* może być wyrażona w sposób statyczny lub dynamiczny. Specyfikacja statyczna wyraża reguły mówiące o tym, które ze stanów bazy są poprawne, czyli które mogą wystąpić. Poprawność rozumiana jest w sensie zachowania integralności bazy (rozdz. 8). Specyfikacja dynamiczna dotyczy operacji. Należy tak definiować operacje w bazie, aby w wyniku ich zastosowania nie naruszyć integralności bazy. Jest i trzeci typ więzów, więzy *niejawne*. Można go wyprowadzić ze zdefiniowanych więzów *wbudowanych* i *jawnych.* W hierarchicznej bazie danych każdy obiekt podrzędny ma najwyżej jeden obiekt nadrzędny. Wynika z tego, że każdy obiekt ma tylko jeden obiekt poprzedzający dowolnego typu. Pierwsze zdanie mówi o więzach *wbudowanych*, drugie o *niejawnych*.

Modele związków encji powstałe w procesie praktycznych badań nad projektowaniem bazy danych powinny być przede wszystkim wystarczająco ogólne i bogate semantycznie, ponadto powinny zawierać wszystkie podstawowe cechy systemów komercyjnych tak, aby mogły być łatwo implementowane.

4.4. Przykład realizacji konkretnej bazy

4.4.1. Sprecyzowanie wymagań dotyczących projektowanej bazy danych

Przykładowa baza danych wykonana została w ramach projektu studenckiego (T. Idasiak, T. Kasper). Przeznaczona jest dla sklepu ze sprzętem elektronicznym, ma być pomocą w zarządzaniu transakcjami (sprzedaż, dostawa towaru) poprzez dokumentację kolejnych zakupów i dostaw. Zawiera pełną informację dotyczącą:

- towaru,
- stanu magazynu,
- danych klientów i dostawców,
- wystawianych faktur,
- dokonanych zakupów,
- zrealizowanych dostaw.

Baza realizuje następujące funkcje:

- zapisywanie danych do bazy,
- dopisywanie danych do bazy,
- modyfikowanie danych znajdujących się w bazie,
- usuwanie danych z bazy,
- wyszukiwanie danych zgodnie z zapotrzebowaniem.

Projektując niniejszą bazę wzięto pod uwagę możliwość wyszukiwania danych zgodnie z następującymi zapytaniami:

- wylistuj wszystkich dostawców, klientów lub sprzedawców,
- wylistuj elementy lub podzespoły o określonych atrybutach,
- podaj parametry opisujące dany element (podzespół),
- wylistuj elementy (podzespoły) dostarczone/zakupione przez dostawcę/klienta (np. w określonym dniu, o określonej cenie),
- wylistuj sprzedaże dokonane przez danego klienta, na które nie została jeszcze wystawiona faktura,
- wylistuj sprzedaże (klientów) dokonane po danym dniu powyżej danej ceny,
- wylistuj dostawy zrealizowane przez dostawcę,
- wylistuj dane sprzedawcy, który wystawił daną fakturę, itd.

4.4.2. Model konceptualny bazy danych

Projektowana baza danych zawiera następujące encje:

DOSTAWCY	Identyfikator, Nazwisko, Imię, Adres (Miasto, Kod, Ulica, Tel/Fax);
SPRZEDAWCY	Określone przez takie same pola jak **DOSTAWCA**;
KLIENTA	Określone przez takie same pola jak **DOSTAWCA**;
ELEMENTU	Identyfikator, Wartość, Cena, Ilość_Szt, Producent, Opis;
PODZESPOŁU	Identyfikator, Nazwa, Cena, Gwarancja, Ilość_Szt, Producent, Opis;
FAKTURY	Identyfikator, Dane_Firmy, Data_Wystawienia, Płatność;
SPRZEDAŻY	Nr Sprzedaży, Cena_Sumaryczna, Data_Sprzedaży, Ilość_Sztuk;
DOSTAWY	Nr Dostawy, Data_Dostawy, informacje o dostawcy.

4.4.3. Normalizacja

W celu wyznaczenia pojedynczych tabel i powiązań pomiędzy nimi przeprowadzono normalizację (rozdz. 5, p. 5.6), którą zrealizowano w następujących krokach:

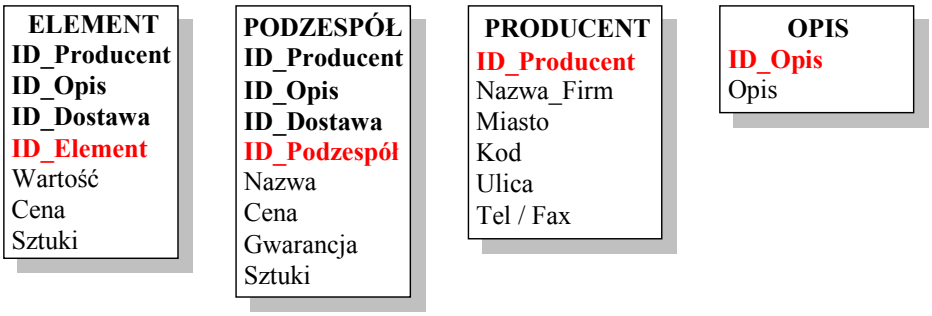
- zebranie zbioru danych,
- przekształcenie nieznormalizowanego zbioru danych w tabele,
- wykorzystanie jednego z algorytmów normalizacji.

Przy każdym kolejnym przekształceniu podzielono strukturę danych na coraz więcej tabel bez straty powiązań zachodzących między danymi (rozdz. 5).

W wyniku normalizacji w projektowanej bazie danych otrzymano następujące tabele:

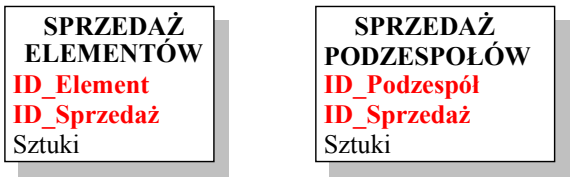
- Tabele danych osobowych:

- Tabele dotyczące towaru:



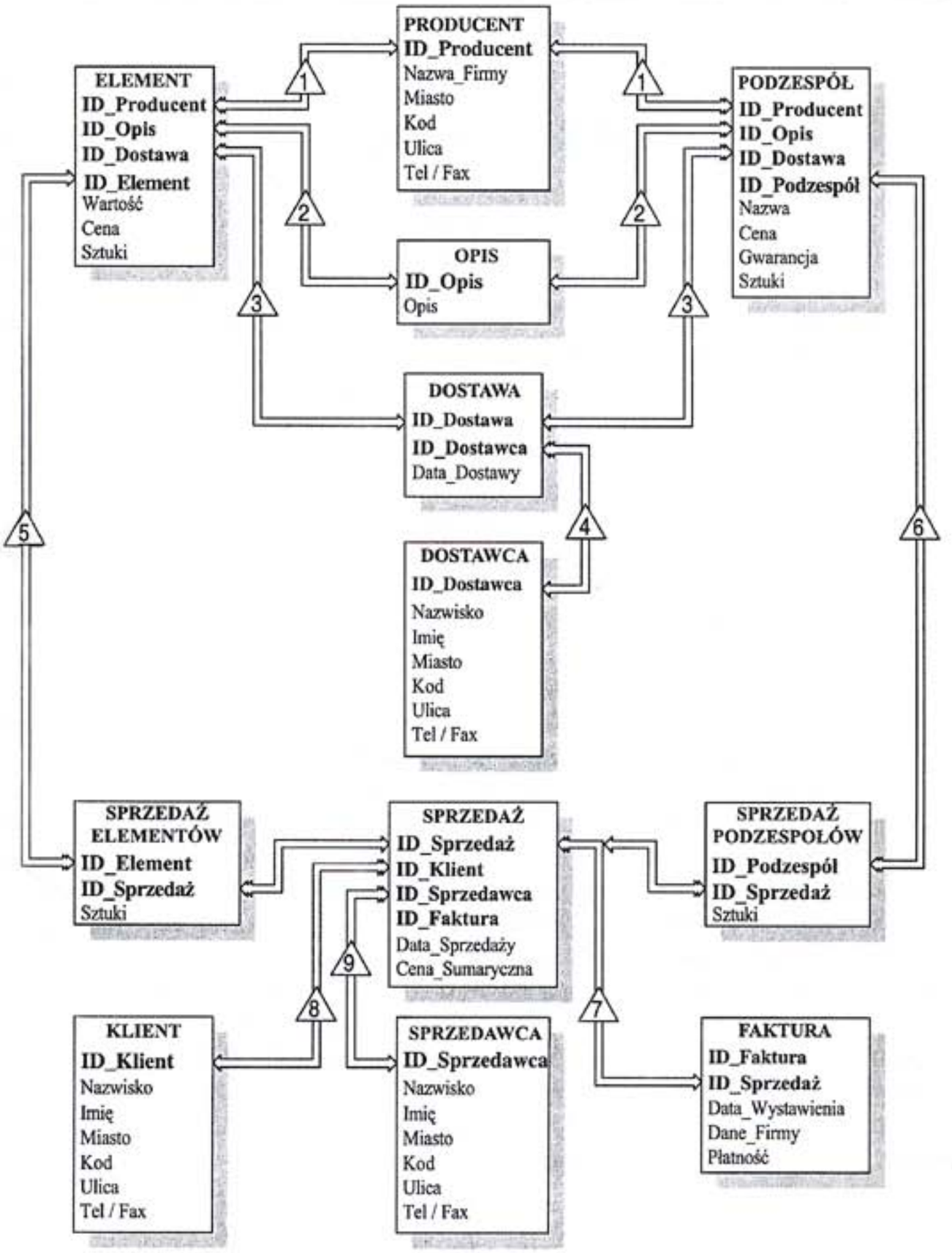
- Tabele dotyczące transakcji:

- Tabele powstałe w celu wyeliminowania powiązań zależności typu wiele do wielu (*n*:*m*):



Łącząc poszczególne tabele z uwzględnieniem relacji między nimi otrzymano diagram encji, w którym zachodzące relacje opisane są w następujący sposób:

BAZA DANYCH OBSŁUGI KLIENTA SKLEPU ELEKTRONICZNEGO



1 – wyprodukował producent, 2 – posiada opis, 3 – dostarczono w dostawie, 4 – dostawę zrealizował, 5 – sprzedano elementy, 6 – sprzedano podzespoły, 7 – wystawiono fakturę, 8 – klient zakupił, 9 – sprzedaż zrealizował

PODSUMOWANIE

Baza konceptualna jest podstawą do tworzenia struktur logicznej bazy danych. Miejsce i rola schematu konceptualnego zależy od sposobu organizacji systemu bazy danych. W przypadku gdy mamy do czynienia z lokalnymi lub rozproszonymi bazami danych, opartymi na jednym modelu, schemat konceptualny ma najczęściej znaczenie pomocnicze i wykorzystywany jest, często nieformalnie, do tworzenia schematu logicznego. Stanowi także punkt odniesienia w przypadku niejednoznaczności, jakie mogą pojawić się przy interpretacji danych przez różnych użytkowników. W wielomodelowych, lokalnych bazach danych schemat konceptualny stanowi integralną część bazy danych. Oprócz niego współistnieją w systemie schematy logiczne utworzone według różnych modeli danych. Wraz z rozwojem sieci komputerowych i związanych z nimi rozproszonych baz danych pojawia się zupełnie nowe znaczenie schematu konceptualnego. Użytkownik korzystając z bazy danych nie odczuwa, że baza ta jest fragmentem systemu rozproszonego. Zawartość wszystkich baz danych tworzących system rozproszony opisana jest w globalnym schemacie konceptualnym, z którego może być wyprodukowanych wiele lokalnych schematów logicznych. Jeżeli teraz użytkownik korzysta z całej rozproszonej bazy, to operuje na poziomie globalnym. SZBD dokonuje koniecznych przekształceń zarówno na poziomie globalnym, jak i na poziomach lokalnych tak, że użytkownik nawet nie wie, z którą z lokalnych baz aktualnie jest związany. Sposób przejścia z poziomu konceptualnego do logicznego zależy przede wszystkim od typu bazy logicznej i metodologii jej projektowania. Już na poziomie logicznym istnieją różne metody projektowania schematów logicznych. W podejściu sieciowym bierze się pod uwagę nie tylko związki logiczne między danymi, ale także przewidywane sposoby i częstość odwoływania się do danych [1, 13]. Wpływa to dodatnio na efektywność systemu, ale jednocześnie zmniejsza niezależność danych. W podejściu relacyjnym bierze się pod uwagę pewne zależności między atrybutami [1, 7, 8] i na tej podstawie tworzy się pewien zbiór danych o pożądanych własnościach. W podejściu tym nie bierze się pod uwagę wydajności systemu, a jedynie zależności między atrybutami, które wyrażają pewne semantyczne własności odwzorowywanego świata rzeczywistego.

BIBLIOGRAFIA

[1] Banachowski L., *Bazy Danych – tworzenie aplikacji*, WNT, Warszawa 1997.

[2] Beynon-Davies P., *Expert Database Systems: a gentle introduction Maidenhead*, MacGraw-Hill 1991.

[3] Beynon-Davies P., Knowlege *Engineering for Information Systems Development; an introduction to information systems engineering*, Maidenhead MacGraw-Hill 1993.

[4] Beynon-Davies P., *Systemy baz danych*, WNT, Warszawa 1998.

[5] Chen P.P.S., *The Entity Relationship Model: towards and unified view of data*. ACM Tran. of Database Systems 1976, 1(1), s.9–36

[6] Codd E.F., *A Relational Model for Large Shared Data Banks Communications of ACM*, June 1970, Vol. 13, No. 6, s. 377–387.

[7] Codd E.F., *Extending the relational Database Model to Capture More Meaning*. ACM Transactions on Database Systems, Dec. 1979, Vol. 4, No. 4, s. 397–434.

[8] Codd E.F., *The Relational Model for Database Management: Version 2.* Reading, Mass. Addison-Wesley 1990.

[9] Coulouris G., Dillimore J., Kindberg T., *Systemy rozproszone. Podstawy i projektowanie*, WNT, Warszawa 1998.

[10] Das S.K. *Deductive Databases and Logic Programming*, Adison-Wesley Publishing Company 1981.

[11] Date C.J., *Wprowadzenie do baz danych*, WNT, Warszawa 1983.

[12] *DBTG: Report of the CODASYL Database Task Group*. ACM, April 1971.

[13] Delobel C., Adiba M., *Relacyjne bazy danych*, WNT, Warszawa 1989.

[14] Krick E.V., *Wprowadzenie do techniki projektowania technicznego*, WNT, Warszawa 1975.

[15] MacVittie D.W., MacVittie L.A., *Programowanie zorientowane obiektowo*, Mikom, Warszawa 1996.

[16] Martin J., *Organizacja baz danych*, PWN, Warszawa 1983.

[17] Myer B., *Object Oriented Software Construction*, New York, Prentice-Hall 1997.

[18] Pankowski T., *Podstawy baz danych*, PWN, Warszawa 1992.

[19] Tsichritzis D.C., Lochovsky F.H., *Modele danych*, WNT, Warszawa 1990.

[20] Ullman J.D., *Systemy baz danych*, WNT, Warszawa 1981.

CZĘŚĆ II

Wybrane metody reprezentacji danych

...prezentowanie użytkownikowi bazy danych w jej obrazie konceptualnym jest niewygodne. Dlatego proponowane są modele danych, których celem jest dostarczenie środków umożliwiających prezentowanie użytkownikowi takiego obrazu bazy danych, który byłby dla niego możliwie naturalny i łatwo zrozumiały... Model danych jest pewnym narzędziem wykorzystywanym do zrozumienia organizacji danych w logicznej bazie danych. Nie opisuje danych w tej bazie, a jedynie wskazuje sposób w jaki dane mogą być organizowane oraz sposób w jaki można organizować do nich dostęp.

T. Pankowski

WPROWADZENIE

Punktem wyjścia do tworzenia logicznego modelu danych jest model konceptualny. Istnieje wiele różnorodnych modeli logicznych danych. Wśród modeli klasycznych na szczególną uwagę zasługuje model relacyjny mający jednolite podstawy teoretyczne. Jego prekursorzy model sieciowy i hierarchiczny są już praktycznie nieużywane. Model relacyjny ze względu na solidne podstawy matematyczne jak i dużą popularność przy budowie komercyjnych SZBD jest tematem rozdziału piątego. Model relacyjny jest klasycznym przykładem sformatowanej bazy danych. Ostatnio wiele cech klasycznych modeli danych, a w szczególności modelu relacyjnego pojawiło się pod nazwą obiektowego modelu danych. Trudno jest jednoznacznie określić, czy systemy obiektowych baz danych oferują lepsze warunki do zarządzania bazami niż modele klasyczne. Bez wątpienia podejście obiektowe nadaje się do pewnych niestandardowych aplikacji, jak systemy informacji geograficznej lub projektowanie wspomagane komputerowo. Pozostaje jednak sprawą nierozstrzygniętą, czy jest rzeczą sensowną użycie obiektowości w standardowych aplikacjach komercyjnych baz danych. Bazy obiektowe są tematem rozdziału szóstego. Wraz z rozwojem takich właściwości systemów baz danych, jak: rozproszenie, zarządzanie obiektami o strukturze hierarchicznej, przechowywanie tekstów, grafiki, obrazów, animacji, dźwięków pojawił się pomysł wykorzystania logiki formalnej w tych systemach. Przy pomocy logiki istnieje możliwość wyrażenia w sposób jednolity takich operacji, jak: definiowanie danych, operowanie danymi i zapewnienie integralności danych. Logika umożliwia również poszerzenie konwencjonalnej bazy danych o narzędzia dedukcyjne. W bazach dedukcyjnych oprócz faktów przechowuje się również reguły. Liczba przechowywanych faktów jest dużo większa od liczby reguł. Język baz dedukcyjnych opiera się na schemacie zdefiniowanym w fazie konceptualnej. Szczególny nacisk położony jest na efektywność systemu. Problem efektywności sprowadza się głównie do efektywnego przeszukiwania bazy i tworzenia dzięki regułom nowych faktów w bazie. Charakterystyczną cechą baz dedukcyjnych jest to, że w wyniku kierowanych do bazy zapytań istnieje możliwość przeglądania wszystkich rozwiązań i analizowanie ich. Dedukcyjne bazy danych są tematem rozdziału siódmego.

5. MODEL RELACYJNY JAKO PRZYKŁAD MODELU KLASYCZNEGO

5.1. Uwagi ogólne

Na rynku komercyjnym dominują relacyjne bazy danych oparte na modelu Codda [13, 14, 15, 16]. Są one podstawą większości takich systemów, jak: Clipper, dBase, Foxpro, Delphi oraz dużych SZBD takich, jak Oracle, Ingress, Progress, Informix.

Podstawowe zalety relacyjnych baz danych to [4, 21, 38]:

– *prostota struktur danych zredukowanych do relacji* – nie wprowadza się żadnych pojęć poziomu fizycznego. Zbiory encji i powiązania między encjami zawsze są reprezentowane przez dwuwymiarowe tablice,

– *nieproceduralne, deklaratywne języki manipulacji danymi* – języki nieproceduralne określają, które informacje mają być wyszukiwane, a nie w jaki sposób,

– *niezależność fizyczna i logiczna danych* – definicja struktur fizycznych i ścieżek jest niezależna od modelu danych,

– *optymalizacja dostępu do bazy danych* – automatyczne generowanie strategii dostępu,

– *oparcie modelu na szczególnie precyzyjnym fundamencie teoretycznym* – teoria zbiorów.

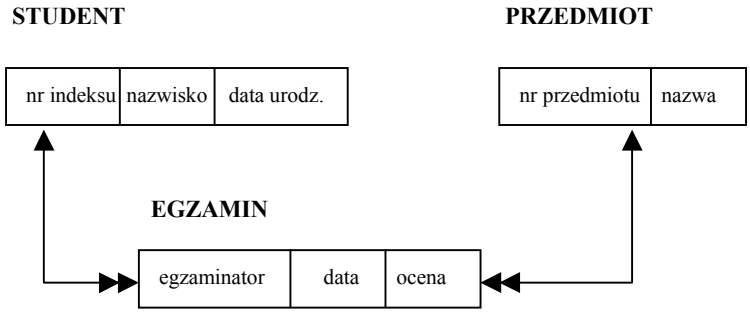
Dzięki tym zaletom bazy relacyjne znalazły zastosowanie w typowych aplikacjach: bankowych, magazynowych, ewidencji personalnej itp.

Trzy podstawowe cechy są charakterystyczne dla modelu relacyjnego:

– struktura danych (jest tylko jedna struktura danych w bazie),

– dostępność operatorów algebry relacji,

– więzy integralności w sposób jawny lub niejawny zapewniające zgodność pamiętanych w relacjach informacji.



Rys. 5.1. Zapis w modelu sieciowym

Jeżeli chodzi o ekspresyjność modelu, czyli zdolność do ujmowania informacji o świecie rzeczywistym, to zapisy zarówno w modelu sieciowym (rys. 5.1) jak i relacyjnym (rys. 5.2) są jednakowo czytelne.

STUDENT (nr indeksu, nazwisko, data urodz.)
PRZEDMIOT (nr przedmiotu, nazwa)
EGZAMIN (nr indeksu, nr przedmiotu, egzaminator, data, ocena)

Rys. 5.2. Zapis w modelu relacyjnym

5.2. Opis modelu

Struktury danych modelu relacyjnego opierają się na pojęciach:

dziedziny – zbioru dopuszczalnych wartości danych,

relacji – tabela dwuwymiarowa będąca podzbiorem iloczynu kartezjańskiego wybranych dziedzin,

krotki – wiersza tabeli dwuwymiarowej,

atrybutu – nazwy kolumny tabeli dwuwymiarowej,

klucza – atrybutu identyfikującego wiersz tabeli.

Każda encja w modelu relacyjnym jest zapisana w postaci KROTKI, czyli wiersza tabeli. Zbiór wierszy, czyli zbiór KROTEK opisanych tymi samymi atrybutami tworzy relację. Relacja, jedyna struktura danych w modelu, jest tabelą, dla której spełniony jest następujący zbiór zasad:

1. Każda relacja w bazie ma jednoznaczną nazwę.
2. Każda kolumna w relacji (jako zbiór) ma jednoznaczną nazwę (atrybut).
3. Wszystkie wartości w kolumnie muszą być tego samego typu (dziedzina).
4. Porządek kolumn w relacji nie jest istotny (elementy zbioru nie są uporządkowane). Ważne jest to, aby kolumny nie powtarzały się w tabeli.
5. Liczba kolumn w relacji to stopień relacji.
6. Każdy wiersz w relacji (KROTKA) musi być różny.
7. Porządek wierszy nie jest istotny.
8. Każde pole leżące na przecięciu wiersza i kolumny powinno być atomowe. Oznacza to, że wartość atrybutu w danej kolumnie powinna być daną elementarną [18]. Tabela zawierająca tylko dane elementarne znajduje się w pierwszej postaci normalnej (1PN). Algorytm tworzenia 1PN opisano w [33].
9. Każda relacja musi mieć klucz główny. Klucz główny to jedna lub więcej kolumn tabeli, w których wartości atrybutów jednoznacznie identyfikują wiersz tabeli.
10. W każdej relacji może istnieć wiele kluczy kandydujących i kluczy obcych. Kluczem obcym w danej tabeli nazywamy klucz główny innej tabeli z nią powiązanej.

5.3. Operacje na relacjach

Definicje podstawowych pojęć relacyjnego modelu danych podano w [33] rozdz. 16, p.16.1. Korzystając z tych pojęć, operacje na relacjach można podzielić na dwie grupy: operacje mnogościowe, które wynikają z faktu, że relacja jest zbiorem oraz operacje relacyjne, które oparte są na rozumieniu relacji jako zbioru krotek. Relacje mnogościowe na zbiorach to: suma, różnica, przekrój, dopełnienie. Operacje na krotkach to: projekcja, łączenie, selekcja i dzielenie.

Operacje mnogościowe można zdefiniować zgodnie z [33] w następujący sposób:

SUMA Relacja $T(U)$ jest sumą relacji $R(U)$ i $S(U)$, co oznaczamy
$T = R \cup S$ wtedy i tylko wtedy, gdy
$T = \{t \in \text{KROTKA}(U) : t \in R \cup t \in S\}$

RÓŻNICA Relacja $T(U)$ jest różnicą relacji $R(U)$ i $S(U)$, co oznaczamy
$T = R - S$ wtedy i tylko wtedy, gdy
$T = \{t \in \text{KROTKA}(U) : t \in R \cap t \notin S\}$

PRZEKRÓJ Relacja T jest przekrojem relacji $R(U)$ i $S(U)$, co oznaczamy
$T = R \cap S$ wtedy i tylko wtedy, gdy
$T = \{t \in \text{KROTKA}(U) : t \in R \cap t \in S\}$

DOPEŁNIENIE Relacja $T(U)$ jest dopełnieniem relacji $R(U)$, co oznaczamy
$T = \overline{R}$ wtedy i tylko wtedy, gdy
$T = \text{KROTKA}(U) - R = \{t \in \text{KROTKA}(U) : t \notin R\}$

Dopełnienie relacji $R(U)$ określone jest tylko wtedy, gdy zbiór KROTKA(U) jest skończony.

Operacje mnogościowe są określone na relacjach jednakowego typu i ich wartościami są również relacje tego samego typu co argumenty. Przytoczone operacje mnogościowe są zawężeniem operacji dobrze znanych z teorii mnogości, gdzie definiowane są one w odniesieniu do dowolnych zbiorów.

Operacje relacyjne definiujemy zgodnie z [33] następująco:

PROJEKCJA

Dana jest relacja $R(U)$ oraz zbiór $X \subseteq U$. Relację $T(X)$ nazywamy projekcją relacji $R(U)$ na zbiór X, co oznaczamy $T = R[X]$ wtedy i tylko wtedy, gdy:

$$T = \{t \in \text{KROTKA}(X) : (\exists r \in R)\ t = r[X]\}$$

• *Przykład 5.1*

Dana jest relacja $R(U)$ stopnia trzeciego zawierająca informację o studentach. U jest zbiorem atrybutów $U = \{$**Nazwisko, Wydział, Rok_studiów**$\}$.

Każdy z atrybutów określony jest przez zbiór wartości:

Nazwisko = {Nowak, Kowal, Owad, Jawor, Olski}, **Wydział** = {Elektronika, Fizyka, Matematyka}, **Rok_studiów** = {1, 2, 3}.

Wykonujemy operacje projekcji relacji *R* na relacje *T*(*X*), gdzie *X* jest zbiorem atrybutów *X* = {**Nazwisko, Rok_studiów**} i na relacje *S*(*Y*), gdzie *Y* = {**Wydział, Rok_studiów**}

R(*U*):

Nazwisko	**Wydział**	**Rok_studiów**
Nowak	Fizyka	1
Kowal	Elektronika	2
Owad	Fizyka	1
Jawor	Matematyka	3
Olski	Fizyka	1

T(*X*):

Nazwisko	**Rok_studiów**
Nowak	1
Kowal	2
Owad	1
Jawor	3
Olski	1

S(*Y*):

Wydział	**Rok_studiów**
Fizyka	1
Elektronika	2
Matematyka	3

W wyniku operacji projekcji uzyskujemy relacje *T*(*X*) i *S*(*Y*) stopnia drugiego. Czasami w wyniku projekcji następuje utrata informacji. Sytuację taką ilustruje przykład 5.2.

• *Przykład 5.2*

Projekcja relacji *R*(*U*) na *T*(*X*), gdzie:

U = {**Nazwisko, Imię, Wydział**}, *X* = {**Nazwisko, Wydział**}

R(*U*):

Nazwisko	**Imię**	**Wydział**
Nowak	Ewa	Fizyka
Nowak	Jan	Fizyka
Kowal	Piotr	Elektronika
Kowal	Adam	Elektronika

T(*X*):

Nazwisko	**Wydział**
Nowak	Fizyka
Kowal	Elektronika

ŁĄCZENIE

Dane są relacje *R*(*X*) i *S*(*Y*) oraz $Z = X \cup Y$. Relacje *T*(*Z*) nazywamy łączeniem tej relacji wtedy i tylko wtedy, gdy:

$$T = \{t \in \text{KROTKA}(Z) : t[X] \in R \cap t[Y] \in S\}$$

Łączenie $T = R \mid S$, inaczej zwane konkatenacją tych krotek, które mają jednakowe wartości w zbiorze atrybutów *X* i *Y*.

• *Przykład 5.3*

Dane są dwie relacje: *T*(*X*) i *S*(*Y*), gdzie *X* = {**Nazwisko, Rok_studiów**} i *Y* = {**Wydział, Rok_studiów**}. Wykonujemy operacje łączenia, w wyniku której uzyskujemy relacje *R*(*U*).

T(*X*):

Nazwisko	**Rok_studiów**
Nowak	1
Kowal	2
Owad	1
Jawor	3
Olski	1

S(*Y*):

Wydział	**Rok_studiów**
Fizyka	1
Elektronika	2
Matematyka	3

R(*U*):

Nazwisko	**Wydział**	**Rok_studiów**
Nowak	Fizyka	1
Kowal	Elektronika	2
Owad	Fizyka	1
Jawor	Matematyka	3
Olski	Fizyka	1

• *Przykład 5.4*

Dane są dwie relacje *A*(*X*) stopnia drugiego i *B*(*Y*) stopnia trzeciego, gdzie: *X* = {**Nr dostawcy, Miasto**}, a *Y* = **{Nr dostawcy, Nr części, data dostawy**}. W wyniku operacji łączenia dostajemy relację *Z*(*U*) stopnia wyższego niż relacje *A*(*X*) i *B*(*Y*).

A(*X*):

Nr_dostawcy	**Miasto**
1020	Kraków
2040	Wrocław
3000	Opole
4000	Opole

B(*Y*):

Nr_dostawcy	**Nr_części**	**Data_dostawy**
1020	100	1999
4000	107	2000
1110	201	1999
2040	207	1998
2020	100	1999

Z(*U*):

Nr_dostawcy	**Miasto**	**Nr_części**	**Data_dostawy**
1020	Kraków	100	1999
2040	Wrocław	207	1998
4000	Opole	107	2000

SELEKCJA

Relacje $T(U)$ nazywamy selekcją relacji $R(U)$ względem warunku selekcji E.

$$T(U) = \{t \in \text{KROTKA}(U) : E(t) = \text{true}\}$$

• *Przykład 5.5*
Dana jest relacja $R(U)$ gdzie $U = \{\mathbf{A, B, C, D}\}$:

R(U):

A	**B**	**C**	**D**
a	*x*	1	3
a	*y*	4	2
c	*x*	3	3
b	*x*	2	1

Przy warunku selekcji $E(t) = C \geq D \cap (A = a \cup A = b)$ w wyniku uzyskamy relacje $S(U)$:

S(*U*):

A	**B**	**C**	**D**
a	*y*	4	2
b	*x*	2	1

DZIELENIE

Dana jest relacja $R(U)$ zbiór $X \subset U$. Relacje $T(U - X)$ nazywamy podzieleniem relacji R przez zbiór X wtedy i tylko wtedy, gdy

$$T = \{t \in \text{KROTKA}(U - X) : (\exists s \in \text{KROTKA}(X)\ )\ t \mid s \in R\}$$

• *Przykład 5.6*

Dana jest relacja $R(U)$, gdzie U = {**Nazwisko, Przedmiot**}. Zbiór wartości, czyli domena (DOM), dla atrybutu **Nazwisko** to DOM(**Nazwisko**) = {Nowak, Kowal, Owad}, a domena, czyli zbiór wartości dla atrybutu **Przedmiot** to DOM(**Przedmiot**) = {Fizyka, Matematyka, Obwody}.

$R(U)$:

Nazwisko	**Przedmiot**
Nowak	Fizyka
Kowal	Matematyka
Owad	Fizyka
Owad	Obwody
Nowak	Matematyka
Nowak	Obwody

Wynikiem podzielenia relacji R(**Nazwisko, Przedmiot**) przez zbiór {**Przedmiot**} jest relacja:

$T(U - X)$:

Nazwisko
Nowak

Jeśli $(s, p) \in R(S, P)$ oznacza, że student s zdał egzamin z przedmiotu p, to zbiór $R/\{s\}$ jest zbiorem tych studentów, którzy zdali egzaminy ze wszystkich przedmiotów.

Operacje mnogościowe i relacyjne tworzą zbiór operacji algebry relacji. Algebra ta jest podstawą do tworzenia języka manipulacji danych w relacyjnej bazie danych. Operacje relacyjne, przede wszystkim projekcja i łączenie, służą do badania zależności między danymi oraz do projektowania schematu logicznego bazy relacyjnej. Aby wprowadzić pojęcie schematu relacyjnego, należy wyjaśnić znaczenie zależności funkcyjnej.

5.4. Schemat relacyjny

We wstępie do części pierwszej określono pojęcie schematu bazy danych. Schemat jest stały, niezmienny dla danej bazy. Zmiana schematu to stworzenie zupełnie nowej bazy. Schemat bazy relacyjny zwany dalej schematem relacyjnym opisuje własności

statyczne bazy relacyjnej. Schemat relacyjny to zbiór encji, pomiędzy którymi są powiązania będące na równi z encjami nośnikami informacji. Aby wyjaśnić pojęcie schematu relacyjnego, podano następujący przykład:

• *Przykład 5.7*

Dana jest relacja: $R(U) = E(I, N, P, O)$, gdzie: I – numer indeksu, N – nazwisko studenta, P – przedmiot egzaminu, O – ocena egzaminu. Aby relacja mogła reprezentować pewną informację, musi mieć cechy, które się nie zmieniają w czasie. Te cechy to: numer indeksu $i \in E(I)$ mający jednoznacznie przyporządkowane nazwisko $n \in E(N)$. Każdy numer to jedno nazwisko, lecz niekoniecznie odwrotnie. Każdej parze $(i, p) \in E(I, P)$ przyporządkowano jednoznacznie ocenę $o \in E(O)$. Podane przykłady powiązań noszą nazwę zależności funkcyjnych między określonymi zbiorami atrybutów.

Schematem relacyjnym nazywamy parę uporządkowaną $\underline{R} = (U, F)$ o zbiorze atrybutów U i zbiorze zależności funkcyjnych F opisanych nad U.

• *Przykład 5.8*

Jeżeli zbiór atrybutów $U = \{I, N, P, O\}$ i zbiór zależności funkcyjnych określimy jako $F = \{I \rightarrow N, IP \rightarrow O\}$, to schemat relacyjny E:

$$\underline{E} = (\{I, N, P, O\}, \{I \rightarrow N, IP \rightarrow O\})$$

W schemacie możemy określić klucz główny, czyli atrybut lub minimalny podzbiór zbioru atrybutów, które mają cechę jednoznacznego, unikalnego identyfikowania krotek w każdej relacji.

• *Przykład 5.9*

Dla schematu relacyjnego $\underline{E}$:

$$\underline{E} = (\{I, N, P, O\}, \{I \rightarrow N, IP \rightarrow O\})$$

Cechę jednoznacznej identyfikacji mają IP, INP, $INPO$. Najmniejszym z nich jest IP. W dalszej części atrybuty będące kluczami będą podkreślane. Pozostałe będziemy nazywać atrybutami niekluczowymi.

5.5. Rozkład schematu relacyjnego

Analiza zależności funkcyjnych między kluczami a pozostałymi atrybutami w schemacie relacyjnym pozwala orzekać o posiadaniu przez schemat pewnych niepożądanych właściwości, czyli defektów. Defekty te mogą być usuwane drogą odpowiedniego procesu rozkładania tych schematów na schematy elementarne. Proces ten nazywa się normalizacją. Rozkład schematów na elementarne może odbywać się:

– bez straty danych,
– bez straty zależności funkcyjnych,
– bez straty danych i zależności równocześnie na składowe niezależne.

Schemat relacji $R(U, F)$ jest rozkładalny bez straty danych na dwa schematy $R[X]$ i $R[Y]$, gdy $X \cup Y = U$ i dla każdej relacji $R = R[X] \mid R[Y]$ (twierdzenie i dowód w [33]).

• *Przykład 5.10*

Relacja E (I, N, P, O) mówi nam o tym, że student o numerze indeksu $i \in I$, nazwisku $n \in N$ zdał egzamin z przedmiotu $p \in P$, na ocenę $o \in O$. Czyli schemat relacyjny ma postać $\underline{E} = (\{I, N, P, O\}, \{I \rightarrow P, IP \rightarrow O\})$

E:

I	**N**	**P**	**O**
80001	Ajacki	Fizyka	5
80001	Ajacki	Matematyka	3
80003	Nowak	Fizyka	4
80004	Kowal	Fizyka	4
80003	Nowak	Matematyka	5
80006	Celer	Fizyka	2
80006	Celer	Matematyka	2

Relacje E można rozłożyć na schematy elementarne (wykonać operacje projekcji) na kilka sposobów.

Sposób 1 (tabele *E*1, *E*2)

*E*1:

I	**N**
80001	Ajacki
80003	Nowak
80004	Kowal
80006	Celer

*E*2:

I	**P**	**O**
80001	Fizyka	5
80001	Matematyka	3
80003	Fizyka	4
80004	Fizyka	4
80003	Matematyka	5
80006	Fizyka	2
80006	Matematyka	2

Sposób 2 (tabele *E*2, *E*3)

*E*2:

I	P	O
80001	Fizyka	5
80001	Matematyka	3
80003	Fizyka	4
80004	Fizyka	4
80003	Matematyka	5
80006	Fizyka	2
80006	Matematyka	2

*E*3:

I	N	P
80001	Ajacki	Fizyka
80001	Ajacki	Matematyka
80003	Nowak	Fizyka
80004	Kowal	Fizyka
80003	Nowak	Matematyka
80006	Celer	Fizyka
80006	Celer	Matematyka

Sposób 3 (tabele *E*1, *E*2, *E*4)

*E*1:

I	N
80001	Ajacki
80003	Nowak
80004	Kowal
80006	Celer

*E*2:

I	P	O
80001	Fizyka	5
80001	Matematyka	3
80003	Fizyka	4
80004	Fizyka	4
80003	Matematyka	5
80006	Fizyka	2
80006	Matematyka	2

*E*4:

I	P
80001	Fizyka
80001	Matematyka
80003	Fizyka
80004	Fizyka
80003	Matematyka
80006	Fizyka
80006	Matematyka

Dla wszystkich przypadków mamy:

$$E1 \mid E2 = E2 \mid E3 = E1 \mid E2 \mid E4$$

A więc wszystkie są poprawne, jednak sposób 1 jest bardziej naturalny i lepiej oddaje semantykę odwzorowania rzeczywistości. Można zauważyć, że rozkłady 1 i 3 dają dwie identyczne tabelki, a dodatkowo przy 3 rozkładzie mamy jeszcze jedną nadmiarową tabelę. Jeżeli interesuje nas informacja o uczęszczaniu na wykłady, to bardziej przydatny jest rozkład sposobem 3.

Rozkład schematu bez straty zależności funkcyjnych ilustruje przykład 5.11:

• *Przykład 5.11*

Mamy dany schemat relacyjny *R*:

$$\underline{R} = (U, F) = (\{S, W, D\}, \{S \rightarrow W, S \rightarrow D, D \rightarrow W, W \rightarrow D\})$$

gdzie:

S – nazwisko studenta, *D* – nazwisko dziekana, *W* – wydział

Relacja *R* ma następujące wartości:

R:

S	**W**	**D**
Nowak	Elektronika	Biernatt
Kowal	Chemia	Wojtas
Makow	Elektronika	Biernatt
Olcki	Chemia	Wojtas

Możemy dokonać rozkładu wg zależności:

1. S → W(S → D)

*R*1:

S	**W**
Nowak	Elektronika
Kowal	Chemia
Makow	Elektronika
Olcki	Chemia

*R*2:

S	**D**
Nowak	Biernatt
Kowal	Wojtas
Makow	Biernatt
Olcki	Wojtas

2. D → W

*R*3:

W	**D**
Elektronika	Biernatt
Chemia	Wojtas

*R*4:

S	**D**
Nowak	Biernatt
Kowal	Wojtas
Makow	Biernatt
Olcki	Wojtas

3. W → D

*R*5:

W	**D**
Elektronika	Biernatt
Chemia	Wojtas

*R*6:

S	**W**
Nowak	Elektronika
Kowal	Chemia
Makow	Elektronika
Olcki	Chemia

Można łatwo stwierdzić, że dla wszystkich rozkładów relacje spełniają warunek

$$R = R1 \mid R2 = R3 \mid R4 = R5 \mid R6$$

Dla schematów można również określić takie same zależności funkcyjne.
$R1 = R6 = (\{S, W\}), \{S \rightarrow W\})$
$R2 = R4 = (\{S, D\}), \{S \rightarrow D\})$
$R3 = R5 = (\{W, D\}), \{W \rightarrow D, D \rightarrow W\})$
W wyniku łączenia schematów mamy:
$R1 \mid R2 = (\{S, W, D\}), \{S \rightarrow W, S \rightarrow D\})$
$R3 \mid R4 = (\{S, W, D\}), \{S \rightarrow D, W \rightarrow D, D \rightarrow W\})$
$R5 \mid R6 = (\{S, W, D\}), \{S \rightarrow W, W \rightarrow D, D \rightarrow W\})$
Czyli $\underline{R} = R3 \mid R4 = R5 \mid R6 \neq R1 \mid R2$

W podanym przykładzie rozkłady były bez straty danych, ale nie wszystkie były bez straty zależności. W rozkładzie $R1 \mid R2$ gubimy zależności $W \rightarrow D$, $D \rightarrow W$. Przy

łączeniu schematów $R3 \mid R4$ zależność $S \rightarrow W$ można wyprowadzić z pozostałych, tak jak $S \rightarrow D$ ze schematów $R5 \mid R6$. Okazuje się również, że rozkłady bez straty zależności nie zawsze są rozkładami bez straty danych [33]. Rozkłady, które zachowują równocześnie dane i zależności zwane są rozkładami na składowe niezależne. Badał je Rissanen [36, 37]. Proces rozkładu schematu relacyjnego na zbiór schematów nazwano procesem dekompozycji lub normalizacji.

5.6. Normalizacja

Termin normalizacja pochodzi od pojęcia postaci normalnej [13, 14, 15, 16], które wprowadził twórca modelu relacyjnego Codd. Pierwsza postać normalna (1PN) określa, że relacja jest tabelą, w której na przecięciu każdej kolumny i każdego wiersza wartość atrybutu jest wartością atomową, tzn. nie zbiorem, ciągiem tylko wartością pojedynczą. Przykładami 1PN są tabele rozpatrywane w rozkładach bez straty danych i bez straty zależności funkcyjnych. Tabele te mają pewne anomalia. Aby wyjaśnić z jakimi anomaliami mamy do czynienia i jak ich należy unikać, posłużymy się przykładem.

• *Przykład 5.12*
Dany jest schemat relacyjny

$$\underline{E} = (\{\underline{I}, N, A, K, \underline{P}, O\}, \{\underline{I} \rightarrow NAK, \underline{IP} \rightarrow O\})$$

gdzie:
I – numer indeksu, N – nazwisko, A – adres studenta, K – kierunek studiów, P – przedmiot, O – ocena, IP – klucz danego schematu.

E:

I	**N**	**A**	**K**	**P**	**O**
77100	Kowal	Wrocław	Elektronika	Matematyka	3
77100	Kowal	Wrocław	Elektronika	Fizyka	4
77101	Nowak	Legnica	Chemia	Matematyka	3
77102	Olcki	Wrocław	Chemia	Matematyka	3
77100	Kowal	Wrocław	Elektronika	Ekonomia	5

Okazuje się, że jeśli zbudujemy schemat relacyjny z tabelą określoną przez zbiór atrybutów (w tym przypadku $U = \{I, N, A, K, P, O\}$), to informacja zawarta w tej tabeli nie zawsze będzie prawdziwa. Nieprawidłowości występujące w tym schemacie to anomalia:

dołączania – w relacji reprezentowane są tylko nazwiska tych studentów, którzy zdali egzamin z danego przedmiotu, inaczej klucz IP nie byłby pełny,

aktualizacji – zmiana adresu pociąga za sobą konieczność przeglądania dużej, często zmiennej liczby krotek; może to doprowadzić przed końcem aktualizacji do sprzecznej informacji w bazie,

usuwania – student 77102 zdał egzamin tylko z jednego przedmiotu, jeśli go unieważnimy, trzeba usunąć całą informację o studencie.

Nieprawidłowości wynikają z tego, że nie wszystkie atrybuty zależne są od całego klucza, niektóre zależne są tylko od jego części, czyli istnieje tzw. niepełna zależność funkcyjna od klucza. Likwidacja tej zależności jest możliwa dzięki określeniu drugiej postaci normalnej (2PN), czyli dzięki rozkładowi schematu za pomocą operacji projekcji.

Tabele relacji są następujące:

*E*1:

I	**N**	**A**	**K**
77100	Kowal	Wrocław	Elektronika
77101	Nowak	Legnica	Chemia
77102	Olcki	Wrocław	Chemia

*E*2:

I	**P**	**O**
77100	Matematyka	3
77100	Fizyka	4
77101	Matematyka	3
77102	Matematyka	3
77100	Ekonomia	5

$\underline{E} = E[I, N, A, K] \mid E\,[I, P, O]$ rozłożono na dwie relacje:

$$\underline{E1} = (\{I, N, A, K\},\ \{I \rightarrow NAK\}) \quad \text{i} \quad \underline{E2} = (\{I, P, O\},\ \{IP \rightarrow O\})$$

Przeprowadzając schemat relacyjny do 2PN trzeba brać pod uwagę następujące fakty:

– schemat jest już w 2PN, jeśli każdy jego klucz jest jednoelementowy,

– przeprowadzanie schematu relacyjnego do 2PN nie jest procesem jednoznacznym, tzn. dla jednego schematu może istnieć wiele równoważnych informacyjnie układów projekcji tego schematu w 2PN,

– spośród zbioru układu równoważnych schematów relacyjnych wybieramy to rozwiązanie, które zawiera najmniej elementów. Nazywamy go układem minimalnym. Układ ten jest o tyle optymalny, o ile prawdą jest, że wraz ze wzrostem liczby relacji wzrasta złożoność układu.

Schemat będący w 2PN może również posiadać nieprawidłowości.

• *Przykład 5.13*

Dany jest schemat relacyjny:

$$\underline{E} = (\{W, A, P, D\}, \{\underline{W} \rightarrow APD, P \rightarrow D\})$$

gdzie:

W – wykonawca stanowi klucz relacji, *A* – adres, *P* – projekt, *D* – data ukończenia projektu.

E:

W	**A**	**P**	**D**
Budrem	Kraków	P1000	99
Elpo	Opole	P1000	99
WRT	Opole	P1001	98
WSK	Łódź	P1002	97

W schemacie określone są pewne funkcje:

– każdy wykonawca *W* ma jednoznacznie określony adres *A*,

– dla każdego wykonawcy *W* jest jednoznacznie określona data *D* ukończenia projektu *P*,

– termin ukończenia projektu *P* jest taki sam dla wszystkich jego wykonawców *W*.

Pomimo że schemat jest w 2PN, posiada następujące nieprawidłowości:

dołączania – nie ma informacji o projekcie *P* i dacie *D* jego zakończenia, jeśli nie jest określony choć jeden jego wykonawca *W*,

aktualizacji – zmiana daty *D* ukończenia projektu *P* pociąga za sobą konieczność przeglądania dużej, często zmiennej liczby krotek (może to doprowadzić przed końcem aktualizacji do sprzecznej informacji w bazie),

usuwania – zaniechanie wykonywania projektu *P* powoduje czasami wykreślenie całej informacji o projekcie.

Wymienione anomalia wynikają z tego, że niektóre atrybuty (*A* i *P*) są zależne tylko od klucza *W*, natomiast *D* od atrybutu nie będącego kluczem. Czyli istnieje tzw. zależność przechodnia.

W wyniku procesu normalizacji schematu relacyjnego $\underline{E}$ uzyskamy następujące schematy elementarne $\underline{E1}$ i $\underline{E2}$:

*E*1:

W	**A**	**P**
Budrem	Kraków	P1000
Elpo	Opole	P1000
WRT	Opole	P1001
WSK	Łódź	P1002

*E*2:

P	**D**
P1000	99
P1001	98
P1002	97

$E = E[W, A, P] \mid E[P, D]$ rozłożono na dwa schematy:

$\underline{E1} = (\{W, A, P\}, \{\underline{W} \rightarrow AP\})$ i $\underline{E2} = (\{P, D\}, \{P \rightarrow D\})$

Przeprowadzając schemat relacyjny do trzeciej postaci normalnej (3PN) trzeba brać pod uwagę następujące fakty:

– przeprowadzanie schematu relacyjnego do 3PN nie jest procesem jednoznacznym, tzn. dla jednego schematu może istnieć wiele równoważnych informacyjnie układów projekcji tego schematu w 3PN,

– spośród zbioru układu równoważnych schematów relacyjnych wybieramy to rozwiązanie, które zawiera najmniej elementów. Nazywamy go układem minimalnym. Układ ten jest o tyle optymalny, o ile prawdą jest, że wraz ze wzrostem liczby relacji wzrasta złożoność układu.

– każdy schemat w 3PN ma tę właściwość, że między atrybutami niekluczowymi nie ma zależności funkcyjnej. Czyli w relacjach wraz ze zmianą wartości atrybutu niekluczowego nie jest związana anomalia. Właściwość ta przysługuje jedynie atrybutom niekluczowym.

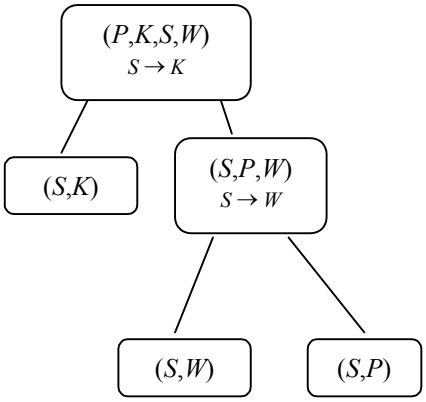
W 3PN istnieją jednak defekty związane z atrybutami kluczowymi. Atrybuty kluczowe mogą być zależne funkcyjnie między sobą, mogą być zależne funkcyjnie od atrybutów niekluczowych, jak również od części pewnego klucza, do którego nie należą. W 3PN może więc istnieć niepełna i przechodnia zależność atrybutów kluczowych. Prowadzi to do powstawania anomalii omawianych poprzednio. Anomalia te wynikają z tego że schemat relacyjny $\underline{E}$ nie jest w postaci normalnej Boyce'a–Codda [15, 33]. Z licznych doświadczeń wynika jednak, że ta postać nie zawsze jest pożądana. Dlatego problem ten musi być rozwiązywany inaczej, na przykład przez zastosowanie algorytmów syntezy schematów relacyjnych [25].

Zależność funkcyjna odzwierciedla pewne semantyczne związki wynikające z odzwierciedlenia rzeczywistości. Uogólnieniem zależności funkcyjnej jest zależność wielowartościowa [22, 23]. Tak jak zależność funkcyjna $X \rightarrow Y$ mówi nam, że każda krotka typu X wyznacza jednoznacznie krotkę typu Y, to zależność wielowartościowa $X \twoheadrightarrow Y$ oznacza, że krotka typu X wyznacza jednoznacznie zbiór krotek typu Y. Zależność wielowartościowa nie ma własności odwrotnej projekcji. Jak istotną rzeczą jest uwzględnienie zależności wielowartościowych przy normalizacji pokazuje przykład analizowany w [33].

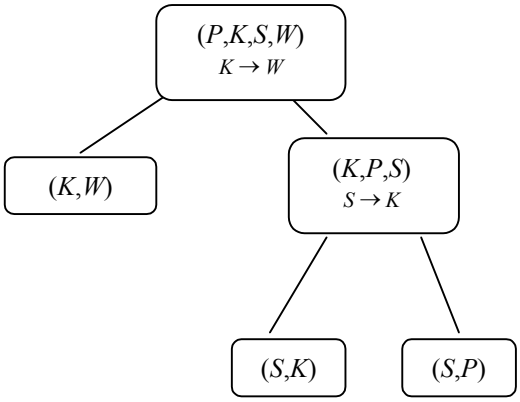
• *Przykład 5.14*
Mamy schemat relacyjny $\underline{S} = (U, F \cup M)$, gdzie:
$U = \{P, K, S, W\}$
P – podręcznik, K – kurs, S – słuchacz kursu, W – wykładowca,
$F = \{S \to K, K \to W, S \to W\}$, co oznacza, że:
jeden słuchacz S uczestniczy w jednym kursie K,
każdy kurs K ma jednego wykładowcę W,
każdy słuchacz S ma jednego wykładowcę W oraz
$M = \{K \twoheadrightarrow P\}$, co oznacza, że każdy kurs ma zalecany zbiór podręczników.

Graficznie można to przedstawić w postaci drzew binarnych:

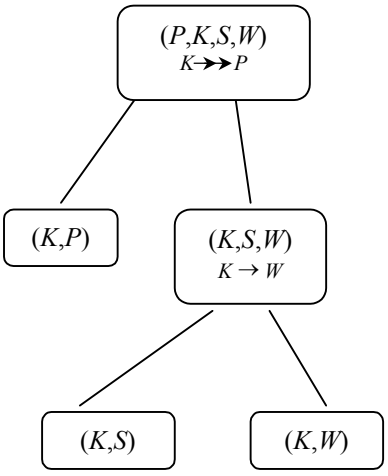
1 sposób: $S \to K$



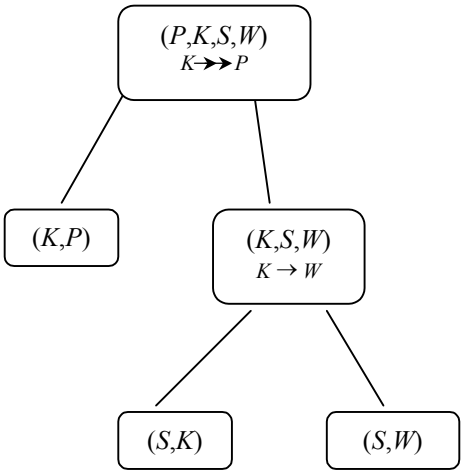
2 sposób: $K \to W$



3 sposób: $K \twoheadrightarrow P$



4 sposób: $K \twoheadrightarrow P$



Przy dekompozycji wg $S \rightarrow K$ lub $K \rightarrow W$ uzyskujemy nienaturalny schemat powiązania słuchacza S z podręcznikiem P, bez określenia kursu K na jaki uczęszcza słuchacz. Bardziej naturalne jest połączenie wg $K \twoheadrightarrow P$ kursu K i słuchacza S (trzeci sposób) niż wykładowcy W i słuchacza S (czwarty sposób). Ogólnie można stwierdzić, że najpierw dekomponuje się schemat według powiązań wielowartościowych, a następnie dopiero pozostałe. Zależność wielowartościowa czasami określana jest jako czwarta postać normalna (4PN).

Jedną z najważniejszych czynności przy projektowaniu bazy danych jest utworzenie schematu. Schemat przesądza o walorach użytkowych bazy, ułatwia zrozumienie działania i manipulowania danymi w bazie oraz zmniejsza możliwość popełnienia błędów. Czasami również ma wpływ na efektywność pracy komputera.

Zadanie tworzenia schematu można sformułować następująco:

Określamy jeden uniwersalny schemat bazy $\underline{S_U} = \{R = (U, F)\}$, czyli traktujemy bazę jako jedną wielką relację, gdzie: U – zbiór wszystkich atrybutów występujących w bazie, F – zbiór wszystkich powiązań funkcyjnych i przekształcamy w wyniku stosowania pewnej procedury w schemat $\underline{S_K} = \{R_i = (U_i, F_i), i = 1, 2, \ldots n\}$ będący zbiorem schematów relacyjnych przy założonych kryteriach. Jako kryteria przyjmuje się zwykle: stopień normalizacji schematów relacyjnych oraz liczbę tych schematów. W literaturze [15, 21, 36, 38] spotykamy wiele algorytmów projektowania schematu bazy danych. Najistotniejsze wśród nich to:

– algorytm dekompozycji [14] i jego rozszerzenia [22],
– algorytm Bernsteina [3],
– algorytm Niekludowej–Calenki [33],
– algorytm Rissanena [36, 37].

Żaden z tych algorytmów nie może być określany mianem najlepszego czy najgorszego. Algorytm dekompozycji i jego rozszerzenia pozwalają co prawda uzyskać schemat bazy w 4PN, jednak nie zachowują zależności funkcyjnych, co w wielu wypadkach stanowi istotną wadę. W innych przypadkach niestosowanie tego algorytmu, który jako jedyny uwzględnia zależności wielowartościowe, również może prowadzić do niewłaściwych rozwiązań. W algorytmie Bernsteina nie są zachowane dane, co poważnie ogranicza jego użyteczność. Wydaje się, że najlepszym jest algorytm Niekludowej–Calenki. Zachowuje zarówno dane jak i zależności oraz daje w wyniku rozkładu minimalną liczbę schematów $\underline{R_i}$. Niestety nie uwzględnia zależności wielowartościowych. Metoda Rissanena daje rozkład zachowujący dane i zależności, jednak liczba wynikowych schematów $\underline{R_i}$ jest maksymalna.

Rozpatrywane algorytmy mają charakter czysto syntaktyczny, ograniczają się do analizy zależności funkcyjnych i wielowartościowych. Zakładają, że zależności między atrybutami są jednoznaczne. Nie biorą pod uwagę analizy semantycznej, która jest tak istotna w projektowaniu bazy. Można je traktować jako etap pośredni przy tworzeniu bazy na podstawie innych modeli ujmujących problemy strukturalizacji i klasyfikacji obiektów.

5.7. Wskazówki praktyczne

Użyteczność bazy danych w dużej mierze zależy od sposobu zdefiniowania tablic danych opisujących obiekty oraz ich wzajemne relacje. Poprawne zestrukturalizowanie baz danych pozwala na szybki i łatwy dostęp do wszystkich potrzebnych informacji poprzez odtwarzanie powiązanych elementów z bazy. Projektowanie bazy należy rozpocząć od określenia przeznaczenia bazy i funkcji jakie ma spełniać, a co za tym idzie od określenia jakiego rodzaju elementy danych potrzebne są do opisu obiektów oraz przewidzieć możliwość rozszerzenia bazy w miarę zwiększania funkcji spełnianych przez tę bazę. A więc należy tworzyć elastyczne struktury danych. Kolejnym krokiem jest ich logiczne zorganizowanie, czyli pogrupowanie elementów związanych z poszczególnymi obiektami i umieszczenie ich w pojedynczych tabelach oraz utworzenie tabel relacji opisujących powiązania pomiędzy elementami danych z różnych tabel. W dobrze zaprojektowanej bazie żaden element danych nie występuje więcej niż jeden raz. Wyjątkiem od tej reguły są elementy danych wykorzystywane jako klucze powiązań. Aby powiązać ze sobą dwie tabele danych, wiersze muszą mieć takie same numery identyfikacyjne; tego typu powtarzanie się danych jest zatem konieczne. Operacją służącą do uporządkowania danych według założonej kolejności jest indeksowanie. Przyspiesza ona wyszukiwanie danych. Również rozmiar tabeli, a przede wszystkim liczba kolumn, ma wpływ na szybkość uzyskiwania informacji, łatwiej bowiem jest przetwarzać dane w mniejszych tabelach niż w większych. Liczba kolumn w tabeli powinna być określona przez naturę elementów danych. Należy próbować grupować pokrewne kolumny związane z tymi samymi obiektami danych w tych samych tabelach danych. W niektórych przypadkach, gdy do opisu obiektu potrzebna jest duża liczba kolumn można je umieścić w różnych tabelach. Ważne jest, aby zachować równowagę pomiędzy liczbą tabel a ich rozmiarami. Istotną sprawą są powiązania pomiędzy elementami danych, a co za tym idzie powiązania pomiędzy tabelami. Wyjątek stanowią relacje typu jeden do jednego między obiektami, kiedy to obiekty te można umieścić w jednej tabeli. W każdym innym przypadku powtarzanie się elementów powoduje marnotrawstwo pamięci oraz zagrożenie dla precyzji informacji w bazie. Typowe bazy danych pozwalają spojrzeć na powiązania między danymi na jeden z trzech sposobów: jako na wspomniane relacje typu 1:1 (jedno-jednoznaczne), 1:*m* lub *n*:1 (jedno-wieloznaczne lub wielo-jednoznaczne) oraz *n*:*m*. (wielo--wieloznaczne). Spośród tych trzech rodzajów relacji, relacje 1:1 stanowią najprostsze powiązanie pomiędzy obiektami danych. Ten typ relacji wiąże jeden obiekt z innym pojedynczym obiektem. Jeżeli do przedstawienia obiektów w pamięci komputera używamy tabel, to obiekty te można umieścić we wspólnej tabeli. W praktyce relacje typu 1:1 między obiektami są rzadkością i zapisywanie ich w jednej tabeli prowadzi do wielokrotnego powtarzania pól, a nawet całych wierszy tabeli. W przypadku powiązań jeden do wielu jedną z kolumn tabeli rezerwuje się do przechowywania da-

nych stanowiących tzw. klucze podstawowe. Klucz podstawowy to minimalny podzbiór zbioru atrybutów jednoznacznie identyfikujący dane w tabeli. Do łączenia z inną tabelą danych przy opisywaniu relacji jeden do wielu niezbędna jest także kolumna lub kolumny stanowiące klucze obce, tzn. odnoszące się do powiązanych tabel. Jeśli relacje między obiektami są wiele-wielu, to trzeba zdefiniować i utworzyć jedną lub wiele tabel relacji, aby te relacje obsłużyć. Tabele relacji wyglądają dokładnie tak samo jak zwyczajne tabele. W rzeczywistości są to tabele danych. Jako takie składają się z pewnej liczby wierszy i kolumn. Istotną różnicą między dwoma rodzajami tabel jest to, że w tabeli relacji przechowywane są elementy danych określające powiązania między dwoma obiektami danych. Każdy wiersz takiej tabeli wiąże podstawowy klucz jednej tabeli z podstawowym kluczem innej. Tabela relacji może nie mieć kolumny własnego klucza.

6. OBIEKTOWY MODEL DANYCH

6.1. Uwagi ogólne

We współczesnej informatyce obiektowość ma wiele różnych znaczeń [4, 11, 26]. Termin ten po raz pierwszy został zastosowany w odniesieniu do grupy języków programowania SIMULA. Wprowadzono pojęcie abstrakcyjnego typu danych jako zintegrowanego pakietu struktur danych i procedur. Następnie zaczęto go używać w odniesieniu do baz danych. Główna różnica między obiektowymi językami programowania a bazami danych jest taka, że obiektowe bazy danych wymagają istnienia trwałych obiektów. W obiektowych językach programowania obiekty istnieją tylko przez krótki czas podczas wykonywania programu. Próbowano różnych sposobów podejścia do problematyki baz danych [19, 27, 34, 40]. Począwszy od wykorzystania możliwości modelu relacyjnego i poszerzenie go o cechy obiektowe do zamiany relacyjnego modelu danych na bogatszy semantycznie model danych zawierający odpowiednio do potrzeb cechy obiektowości. Przykładem tego jest stworzenie modelu nazwanego NF2 zawierającego zagnieżdżone relacje, czyli relacje nie w pierwszej postaci normalnej (1PN). Zostały one zaproponowane jako sposób operowania złożonymi obiektami. Obiekt złożony to obiekt o strukturze hierarchicznej, którego atrybuty mogą mieć wartości zarówno elementarne, jak i złożone. Podobnie jak relacje proste, również relacje zagnieżdżone mogą być definiowane przez podanie listy nazw kolumn. Nazwa kolumny w relacji zagnieżdżonej może odwoływać się do grupy wartości atrybutów, a nie tylko do wartości elementarnej atrybutu, przy czym każda z nich może odwoływać się znowu do grupy wartości.

Większość istniejących systemów relacyjnych dostarcza użytkownikowi tylko ograniczonej liczby typów danych (integer, date, real itp). Są one na ogół wystarczające do standardowych zastosowań [20, 31, 32]. Do aplikacji typu komputerowo wspomagane projektowanie (CAD), czy systemów informacji geograficznej te typy danych nie są wystarczająco bogate. Rozszerzenie zakresu typu danych w zależności od zastosowań jest po prostu niepraktyczne. Lepszym rozwiązaniem wydaje się danie użytkownikowi możliwości nieograniczonego rozszerzania systemu baz danych o określone przez niego i zaimplementowane tak zwane abstrakcyjne typy danych, w skrócie ADT (ang. *Abstract Data Types*). ADT jest to typ obiektu, który określa dziedzinę wartości i zbiór operacji zaprojektowanych do działania na tych wartościach [1, 9].

6.2. Obiekty i klasy

Obiektowa baza danych [29] składa się z obiektów i klas obiektów powiązanych pewną liczbą mechanizmów abstrakcji. Obiekt jest przeniesieniem do języka programowania struktur faktycznie istniejących w rzeczywistym świecie. Obiekt określony jest przez swój stan lub zachowanie. W języku programowania stanem obiektu jest opisujący go zestaw atrybutów, natomiast jego zachowanie definiuje zestaw metod– procedur, które możemy na nim wykonać. Metody pozwalają obiektowi na komunikację z innymi obiektami i są uaktualniane przez komunikaty przekazywane między obiektami. Metody i atrybuty są w obiekcie bezpośrednio powiązane. W bazie obiektowej dwa identyczne rekordy mogą się odwoływać do dwóch różnych obiektów dzięki wprowadzeniu jednoznacznego identyfikatora generowanego przez system. Istnieje możliwość rozróżniania dwóch obiektów o takich samych cechach. Jest to niemożliwe w modelach relacyjnych, gdzie dwie identyczne krotki zawsze wskazują ten sam obiekt. Wszystkie obiekty muszą mieć właściwość hermetyzacji. Jest to proces umieszczania danych i procedur w jednym opakowaniu w ramach zdefiniowanego interfejsu i udostępnienie go z zewnątrz w sposób kontrolowany przez interfejs. Hermetyzacja określona jest niejawnie w definicji obiektu, ponieważ wszystkie operacje na obiektach muszą być wykonywane przez zdefiniowane procedury dołączone do obiektów. Uogólnieniem pojęcia obiektu jest pojęcie klasy. Klasa obiektów jest zgrupowaniem podobnych obiektów. Używamy jej do określania wspólnych dla grupy obiektów atrybutów, metod i związków. Klasy obiektów definiują schemat bazy danych. Obiektowy model danych dostarcza dwóch mechanizmów, które umożliwiają projektantowi bazy konstruowanie struktur lub konstruowanie hierarchii klas obiektów. Te mechanizmy abstrakcji to: uogólnienie i agregacja. Uogólnienie umożliwia deklarowanie pewnych klas obiektów jako podklas innych klas obiektów, np. klasy *Kierownik*, *Sekretarka*, *Technik* mogą być zadeklarowane jako podklasy klasy *Pracownik*. Agregacja jest procesem, dzięki któremu obiekt wyższego poziomu jest używany do grupowania pewnej liczby obiektów niższego poziomu. Na przykład encję Samochód można zbudować ze zbioru części kół, podwozia, silnika itp. Obiektowy model danych musi również dostarczać sposobów na powiązanie obiektów z klasami obiektów. W tym wypadku mówimy, że klasa obiektów klasyfikuje obiekt lub odwrotnie, obiekt jest instancją klasy obiektów.

6.3. Atrybuty

Obiektowe bazy danych definiują wiele różnych typów atrybutów. Można je podzielić na takie, które odnoszą się bezpośrednio do obiektu lub do całej klasy obiektów. Atrybuty obiektu opisują stan konkretnego obiektu. Wyróżniamy atrybuty proste, jednowartościowe oraz atrybuty złożone, wielowartościowe. Dziedziną atrybutów

prostych są typy danych znane ze strukturalnych języków programowania, np. integer, real, string i jako takie nie odbiegają od atrybutów relacyjnych baz danych. Do atrybutów prostych zaliczamy również atrybuty referencyjne, to znaczy takie, które definiują powiązania między obiektami. Wartością takiego atrybutu jest identyfikator obiektu, którego dotyczy referencja. Atrybuty referencyjne odpowiadają kluczom obcym w bazie relacyjnej. Atrybuty wielowartościowe składają się z wielu atrybutów prostych tworzących takie struktury, jak: lista, kolekcja, tabela. Tego typu atrybuty zwiększają elastyczność budowania złożonych struktur danych. Również definicja obiektu, który występuje w definicji innego obiektu i tworzy obiekt kompozytowy może być atrybutem wielowartościowym. Występują również atrybuty wyliczane. Wartość takiego atrybutu nie jest przechowywana w obiekcie, lecz jest wyliczana w momencie odwoływania się do niego. Atrybut taki jest po prostu wywołaniem określonej metody obliczającej jego wartość na podstawie stanu innych atrybutów.

6.4. Metody

Pełna definicja metody danej klasy składa się z deklaracji i kodu. Deklaracja opisuje działanie metody, określa dokładnie nazwę metody, nazwy i rodzaj argumentów i jeśli metoda zwraca jakiś wynik, to jego rodzaj, czyli klasę. Sprawdzanie poprawności podawanych w metodzie argumentów odbywać się musi na poziomie kompilacji. Kod metody to lista instrukcji w języku programowania (np. w języku C++) wykonująca operacje na podanych argumentach lub na argumentach klas. Nowy obiekt można tworzyć albo przez wykonanie operacji *new*, albo stosowanie obiektów prototypowych. Jeżeli mamy do czynienia ze środowiskiem mało zmieniającym się, to do tworzenia obiektów należy wykorzystać metodę *new.* Tę samą metodę *new* można wywoływać w celu utworzenia wielu obiektów tej samej klasy. Metoda *new* jest metodą danej klasy, nie będzie ona natomiast metodą obiektu. W nowo utworzonym obiekcie nie będą już przechowywane informacje zawarte w definicji klasy, to jest wartości atrybutów wspólnych dla wszystkich obiektów, czy implementacje metod. W przypadku środowiska szybko zmieniającego się do tworzenia obiektów należy wykorzystać obiekty prototypowe. Nowy obiekt powstaje z już istniejącego obiektu poprzez modyfikację jego argumentów i metod. Aby zdefiniować treść metody, trzeba użyć standardowego języka programowania. Komunikaty muszą zawierać nazwę metody, identyfikator obiektu i odpowiednie parametry metody.

6.5. Dziedziczenie i tożsamość obiektów

Podstawową cechą programowania obiektowego i obiektowych baz danych jest mechanizm dziedziczenia. Pozwala on na tworzenie nowych klas zwanych podklasami z klas już istniejących. Proces dziedziczenia jest związany z pojęciem hierarchii

uogólnienia. Istnieją dwa główne typy dziedziczenia: dziedziczenie struktury i dziedziczenie zachowania. Przy dziedziczeniu struktury mówimy, że podklasa dziedziczy atrybuty swojej nadklasy. Przy założeniu, że na przykład klasa Pracownik ma atrybuty: *nazwisko*, *imię*, *adres*, *pensja* wówczas podklasy: *Kierownik*, *Sekretarka*, *Technik* mają takie same atrybuty. Czyli struktura danych w podklasie jest przejmowana z klas zwanych nadklasami. Dołączane są także własne struktury podklasy. Przy dziedziczeniu zachowania podklasa dziedziczy metody swojej nadklasy. Gdy klasa Pracownik ma określoną metodę *OBLICZAJ PŁACĘ*, to podklasa Kierownik też ma określoną tę metodę. Jeżeli obiektowa baza realizuje dziedziczenie pojedyncze, to klasa może być podklasą tylko jednej nadklasy. Jeśli jedna klasa ma wiele podklas, mówimy wtedy o dziedziczeniu wielokrotnym. Przy dziedziczeniu wielokrotnym klasa może dziedziczyć atrybuty i metody od więcej niż jednej nadklasy. Jedną z zalet dziedziczenia jest to, że kod metod jest wykorzystywany przez wiele obiektów często należących do różnych klas. Najczęściej używane metody mogą być zdefiniowane dla nadklasy, a w razie potrzeby redefiniowane w poszczególnych podklasach. Dziedziczenie upraszcza w znaczny sposób tworzenie nowych klas. Dziedziczone struktury danych i metody mogą być przenoszone w sposób bezpośredni lub modyfikowane podczas przenoszenia z nadklasy do podklasy (np. zmiana nazwy atrybutu). W metodach dostarczających mechanizmu dziedziczenia wielokrotnego zachodzą różne konflikty związane z dziedziczeniem atrybutów i metod. Przykładem może tu być przypadek, gdy w nadklasach danej klasy są atrybuty czy metody o takiej samej nazwie, a różniące się semantyką czy wartościami. Poważne problemy z dziedziczeniem występują w przypadku zapytań do bazy. W języku zapytań mogą być używane również metody. Jeżeli w podklasie jakaś metoda została redefiniowana, to przy ogólnym odwołaniu do tej metody wynik może być błędny. Aby tego uniknąć, należy zachować zgodność z atrybutami i metodami zdefiniowanymi w nadklasie. Mechanizm istnienia różnych funkcji pod tą sama nazwą nazywa się polimorfizmem. Przy wykonywaniu operacji na bazie wygodne jest użycie tej samej nazwy metody dla różnych rodzajów działań. Odbywa się to poprzez redefiniowanie metody zdefiniowanej na wyższym poziomie w hierarchii dziedziczenia. Metoda, pod której nazwą kryje się wiele implementacji nosi nazwę metody przeciążonej. Tożsamość obiektu jest to cecha, która pozwala odróżnić go od pozostałych obiektów istniejących w systemie. Ma to szczególne znaczenie podczas operacji wyszukiwania i kasowania. Dokładna identyfikacja jest korzystna, jeśli się założy, że wartościami atrybutów obiektu mogą być również obiekty. Problem identyfikacji w systemach obiektowych dotyczy zarówno obiektów, jak i klas obiektów. Identyfikację klas można zapewnić narzucając unikalność nazwy. Obiekt ma unikalny identyfikator, który służy zarówno do identyfikacji obiektu jak i do wprowadzania powiązań między obiektami, jeśli wartością atrybutu obiektu pewnej klasy jest inny obiekt. Identyfikator jest odpowiednikiem klucza w bazie relacyjnej. Zaletami stosowania identyfikatora obiektu w stosunku do klucza jest to, że klucz jest unikalny zwykle w obrębie relacji, natomiast identyfikator w całej bazie. Identyfikato-

ry są implementowane przez system, więc programista nie musi się obawiać o odpowiedni ich dobór. Pojęcie identyfikatora wprowadza pojęcie równości i identyczności obiektów. Dwa obiekty są identyczne, gdy mają ten sam identyfikator. Dwa obiekty są równe, jeśli wartości wszystkich atrybutów są rekursywnie równe. Prawdziwe jest twierdzenie, że dwa obiekty identyczne są równe, natomiast odwrotne jest fałszywe.

6.6. Enkapsulacja

Enkapsulacja jest to rozdzielenie deklaracji i implementacji metod, dzięki czemu uzyskuje się ochronę danych przed niepowołanymi użytkownikami oraz zapewnia się prywatność zawartości metod. Enkapsulacja umożliwia strukturalizację programu poprzez jego podział na niezależne moduły. Pełna enkapsulacja zapewnia, że funkcjonowanie i wewnętrzna budowa obiektu nie są widoczne dla pozostałych obiektów. Dostęp do tego obiektu odbywa się poprzez metody tego obiektu wykonujące pewne operacje na danych zawartych w tym obiekcie. Wywołanie metod obiektu ma charakter ogólny, natomiast struktura danych obiektu i operacje na nich wykonywane mają charakter lokalny. Pojęcie enkapsulacji wywodzące się z obiektowych języków programowania nawiązuje do abstrakcyjnych typów danych. Można wyróżnić osobno miejsce deklaracji, gdzie umieszcza się wszystkie operacje jakie mogą być wykonywane na danym obiekcie, od miejsca definicji, w którym zawarte są wszystkie definicje danych opisujących stan obiektu oraz definicje metod opisujących operacje na danych. Użytkownik ma dostęp tylko do części deklaracyjnej. W przypadku baz danych problem trochę się komplikuje. Powstaje niejasność, czy dane powinny znajdować się w części deklaracji, przez co będą widoczne i dostępne dla użytkownika, czy też mają zostać ukryte w części definicyjnej. W bazach obiektowych dane są umieszczone w obiekcie. Obiekt w części definicji ma atrybuty i metody. Zarówno dane jak i operacje na nich zawarte są w tej samej bazie danych. Enkapsulacja gwarantuje, że użytkownik ma dostęp do danych tylko za pomocą metod. Model enkapsulacji zapewnia niezależność logiczną danych. Można zmienić zestaw atrybutów opisujących dany obiekt oraz implementacje metod, nie zmieniając programów odwołujących się do tych danych, a deklaracja w tym wypadku nie ulegnie zmianie. Zachowanie pełnej niewidoczności danych nie zawsze jest korzystne. Przy zapytaniach do bazy danych wskazany byłby bezpośredni dostęp do atrybutów obiektu. Systemy obiektowe oferują taki dostęp poprzez zdefiniowanie operacji czytania i modyfikacji. Tego typu rozwiązanie ma dwie zalety: operacje te są napisane w sposób efektywny, a użytkownik zostaje zwolniony z obowiązku pisania wielu metod czytających i modyfikujących atrybuty.

6.7. Utrzymanie poprawności bazy

Zachowanie poprawności danych i zależności miedzy nimi jest ważnym problemem w przypadku rozbudowanych baz danych. Mogą się pojawiać dwa rodzaje nieprawidłowości: brak zgodności wartości atrybutów z narzuconymi wcześniej warunkami lub niepoprawne powiązania między atrybutami i obiektami w bazie danych. Należy więc dążyć do:

– zachowania poprawności danych jednostkowych, czyli utrzymania poprawności wartości atrybutów w obiekcie. Przykładem jest wymaganie, aby wartość atrybutu była niepusta,

– zachowania właściwych relacji między atrybutami, na przykład ojciec nie może być młodszy od syna,

– zachowania spójności związków pomiędzy różnymi obiektami, na przykład zapewnienie, aby przy kasowaniu obiektów nie pozostały inne ze wskazaniami na skasowany obiekt.

Utrzymanie poprawności bazy realizuje się poprzez definiowanie więzów integralności, czyli określenie warunków, jakie muszą spełniać wartości atrybutów wszystkich obiektów należących do danej klasy. Po każdej modyfikacji atrybutów obiektu więzy są testowane. Jeśli dana operacja narusza więzy, generowany jest błąd, a operacja zostaje anulowana. Również korzysta się tu z procedur wywoływanych zdarzeniami zachodzącymi w bazie. Takim zdarzeniem może być usunięcie, modyfikacja lub wstawienie danych. Procedury te, określane jako wyzwalacze, opisano w p. 8.6.

6.8. Mechanizmy ochrony dostępu do danych

W obiektowych bazach danych podstawową jednostką ochrony jest obiekt. Budując model ochrony trzeba pamiętać, aby był on spójny z modelem danych, uwzględniał takie właściwości baz danych, jak dziedziczenie czy istnienie obiektów kompozytowych. Mocno rozbudowany system zabezpieczeń może mieć znaczny wpływ na efektywność pracy bazy. Mechanizm ochrony dostępu dla baz można na przykład podzielić na: zbiór obiektów O, którym będą przyznawane różnego typu prawa dostępu, zbiór podmiotów P, które będą żądały dostępu do obiektów oraz zbiór dostępów D – czytanie, kasowanie, modyfikacja. Wówczas tryb dostępu można zdefiniować jako funkcję:

$$F(p, o, d), \text{ gdzie } o \in O, p \in P, d \in D \text{ a } f: P \times O \times D \rightarrow (\text{Prawda, Fałsz}),$$

która przyjmuje wartości Prawda, gdy podmiot p może uzyskać dostęp d do obiektu o, a Fałsz, gdy go nie ma. Na podstawie informacji o możliwości dostępu do jakiegoś obiektu można określić zasady i wyprowadzić reguły dostępu do obiektów, na przy-

kład: *Kierownik* ma dostęp do wszystkich danych, do których ma dostęp jego *Podwładny* (dziedzina *P*) lub użytkownik, który może modyfikować dany obiekt, może go również czytać (dziedzina *D*).

6.9. Uwagi końcowe

Obiektowe bazy danych są potężnym narzędziem w rękach programistów. W odróżnieniu od relacyjnego systemu [9, 11] dostarczają pełnego wsparcia dla obiektowego modelu programowania użytego w językach podobnych do C++ i Java. Ten model programowania jest intuicyjny, dobry do modelowania relacji i wygodny dla dużych projektów. Obiektowa baza danych łączy ze sobą semantykę obiektowego języka programowania z zarządzaniem danymi i zapytaniami. Pozwala to na zarządzanie dużą liczbą danych i modelowanie relacji pomiędzy nimi. Obecnie obiektowe techniki są widziane jako modna forma tworzenia oprogramowania. Należy jednak zdać sobie sprawę z tego, że modelowanie obiektowe jest dobrą metodą do tworzenia bardzo złożonych struktur systemów programistycznych. Te same struktury, które nam pozwalają wyrazić semantykę relacji pomiędzy obiektami mogą być użyte do tworzenia programu o postaci modularnej, który jest lepszy strukturalnie i łatwiejszy do zrozumienia. Obiektowe programy pozwalają programiście łączyć kod z danymi w jednej strukturze zwanej obiektem. Definicja tego obiektu nazwana jest klasą. Klasy i obiekty dla małych programów są proste i przejrzyste, ale dopiero wykorzystanie ich w dużych i złożonych projektach pozwala nam właściwie docenić podejście obiektowe. Również zarządzanie relacjami między komponentami w dużych złożonych systemach stanowi poważny problem. Obiekty bardzo dobrze nadają się do naturalnego modelowania relacji, co jest ważnym powodem przy wykorzystaniu ich w dużych projektach. Siła i moc obiektowych baz danych leży w tym, że obiekty ściśle odwzorowują istotę rozwiązywanego problemu. Logiczne powiązania między obiektami mogą być jasno zobrazowane, jeśli się użyje do tego celu dziedziczenia i polimorfizmu. Obiektowa baza danych zapewnia: długookresowe zapamiętywanie, dużą pojemność składowania danych, zapytania bazujące na wartościach, współdzielenie obiektów między programami, niezależne od urządzenia formaty, precyzyjną obsługę błędów w bazie. Ponadto obiektowa baza danych pozwala na rozwiązywanie problemu referencji w programie, relacji jeden do wielu, zapytań bazowanych na wartościach wraz z sortowaniem wyników, inteligentnego zarządzania obiektami. Bazy obiektowe są dobrze przystosowane do wielkiej liczby danych i szybkiej realizacji zapytań opartych na wartościach. Przykład realizacji systemu opartego na modelu obiektowym podano w rozdz. 9 części III.

7. DEDUKCYJNE BAZY DANYCH

7.1. Uwagi ogólne

W dedukcyjnych bazach danych do opisu danych używa się języka logiki pierwszego rzędu, natomiast operowanie na danych sprowadza się do wartościowania formuł logicznych [17, 39]. Języki logiki charakteryzują się zrozumiałą semantyką, są narzędziami umożliwiającymi jednorodne opisywanie faktów i reguł rządzących modelowaną rzeczywistością, więzów integralności oraz zapytań kierowanych do bazy. Teoria języków logiki ułatwia rozwiązywanie wielu problemów związanych z efektywnym zarządzaniem bazą. Pojedyncze reguły logiczne ze względu na wysoki stopień ekspresji mogą zastąpić całe zbiory jawnie definiowanych faktów, co z jednej strony upraszcza proces modelowania encji, z drugiej zwiększa czytelność i ułatwia zrozumienie wybranego sposobu opisu rzeczywistości [31, 39]. Pierwsze badania nad zastosowaniem logiki do systemów baz danych prowadzone były w latach 60. W tym samym czasie pojawiły się prace Kuhnsa (1967), Levina i Marona (1967) oraz Greena i Raphaela (1968). We wszystkich tych pracach pokazywano możliwości zastosowania języka logiki do zadawania zapytań oraz wykorzystanie zasady rezolucji [4] do wyszukiwania informacji. Systemy te można określić jako typu „pytanie – odpowiedź" działające na niewielkiej liczbie faktów. Pierwsze prace teoretyczne na temat ograniczeń języka logiki do formułowania zapytań prowadzone były przez Di Paola w 1969. Od tego czasu rozwijane są koncepcje programowania logicznego. Obecne prace [2, 6, 7, 8, 10] nad udoskonaleniem tego typu systemów polegają na integracji reguł z systemami zarządzania bazami. Prowadzone są intensywne badania, na ogół w ramach dużych projektów badawczych, których celem jest budowa efektywnych systemów dedukcyjnych. Należałoby tu jeszcze powiedzieć, że błędem, pomimo wielu podobieństw, jest utożsamianie baz dedukcyjnych z systemami eksperckimi [12, 30, 41]. Systemy eksperckie mogą odpowiadać systemom reprezentacji wiedzy, podczas gdy systemy zarządzania bazami wiedzy stanowią odpowiednik baz dedukcyjnych.

7.2. Model dedukcyjnej bazy danych

U podstaw każdego systemu bazy danych leży model danych, który jest zbiorem pewnych abstrakcyjnych pojęć umożliwiającym opis fragmentu modelowanej rzeczywistości.

Dedukcyjną bazę danych w postaci ogólnej można zdefiniować jako parę:

$$\Theta := (K, L)$$

gdzie: K to skończony zbiór klauzul, L to język logiki pierwszego rzędu.

Klauzula bazy danych jest to formuła logiczna następującej postaci:

$$A_1 \vee A_2 \vee A_3 \vee ..... \vee A_m \Leftarrow L_1 \wedge L_2 \wedge L_3 \wedge ..... \wedge L_n$$

gdzie: $m \geq 1$, $n \geq 0$, A_i jest atomem, L_j jest literałem.

Część klauzuli występująca po lewej stronie zwana jest głową klauzuli, natomiast część występująca po prawej stronie ciałem klauzuli.

W związku z tym nasuwa się określenie takich pojęć jak reguła i fakt.

Reguła – to klauzula z niepustym ciałem,

fakt – to klauzula pozbawiona ciała.

W dedukcyjnej bazie danych można wyróżnić:

– predykaty bazowe,

– predykaty wirtualne: proste i zmaterializowane,

– predykaty zewnętrzne.

Predykaty bazowe określają fakty. Mają one znacznie szerszy zestaw typów wartości danych prostych dla atrybutów niż jest to przyjęte w bazie relacyjnej. Relacyjna baza danych [24] stanowi szczególny przypadek bazy dedukcyjnej jako baza zawierająca same fakty. W bazie dedukcyjnej istnieje dodatkowo możliwość definiowania atrybutów zagregowanych. Reguły będące wystąpieniami predykatów wirtualnych umożliwiają wywodzenie nowych faktów (predykatów wirtualnych prostych) na podstawie istniejących w bazie predykatów bazowych i otrzymanych w wyniku wcześniejszego zastosowania reguł innych predykatów wirtualnych. W szczególnych sytuacjach predykaty wirtualne mogą być materializowane, to znaczy, że wywiedzione z nich fakty mogą zostać w sposób jawny zapamiętane w bazie danych tak jakby były wystąpieniami odpowiedniego predykatu bazowego. Materializacja predykatów zwiększa efektywność systemu. Na ogół materializuje się predykaty, do których często odwołują się zapytania. Również materializuje się te, które wyznaczają liczne zbiory wirtualnych faktów, aby wielokrotnie nie stosować tych samych reguł do otrzymania tej samej informacji. Materializacja faktów wymaga jednak stosowania dodatkowo pewnych dynamicznie uaktualnianych mechanizmów, dla zachowania integralności bazy, łącznie ze zmianą predykatów bazowych. Reguły mają określony tzw. język regułowy zawierający składnię i mechanizm wnioskowania. Język ten wywodzi się ze sztucznej inteligencji. W zależności od rodzaju bazy istnieje wiele różniących się pod względem semantyki i syntaktyki języków.

Trzecim typem predykatów są predykaty zewnętrzne zdefiniowane poza systemem. Predykaty te mają za zadanie między innymi:

– sprawdzenie, czy między argumentami zachodzą określone związki, np. równości większości itp.,

– realizację operacji arytmetycznych na argumentach predykatów,
– konwersję typów argumentów,
– wymianę informacji z operatorem.

Predykaty te nie mogą odwoływać się do systemu bazy danych, ani odczytywać, ani uaktualniać danych.

7.3. Języki dedukcyjnych baz danych

Języki dedukcyjnych baz danych składają się z dwóch komponentów: deklaratywnego i proceduralnego. Komponent deklaratywny jest odpowiednikiem opracowanego przez firmę Codasyl [18] języka definiowania danych (ang. Data Definition Language). Umożliwia on definiowanie reguł i ograniczeń integralnościowych przechowywanych w bazie. Deklaratywność języka polega na tym, że użytkownik nie jest świadom sposobu wartościowania reguł i ograniczania podczas współbieżnego wykonywania transakcji, jak również nie ma możliwości wymuszania określonej procedury ich wartościowania. Transakcja jest logiczną jednostką pracy, mogącą się składać z jednej bądź kilku elementarnych akcji, takich jak wywołanie instrukcji typu [28] INSERT, UPDATE czy DELETE mogących zmienić stan bazy danych. Transakcja, jak podano w rozdziale 3, ma następujące własności:

– atomowość oznaczającą, że jest jednostką niepodzielną, to znaczy, że wszystkie jej akcje elementarne są jednocześnie potwierdzane albo odwoływane,
– trwałość gwarantującą, że w przypadku pomyślnego zakończenia transakcji wszystkie wprowadzone przez nią zmiany do bazy nie zostaną później utracone nawet w razie awarii systemu,
– izolację gwarantującą wykonanie transakcji w sposób bezkolizyjny,
– spójność – odwzorowującą taki stan bazy, w którym spełnione są wszystkie zdefiniowane w niej więzy integralnościowe w inny również spójny stan. Odwoływanie transakcji nie powoduje utraty spójności bazy danych.

Komponent proceduralny jest odpowiednikiem języka manipulowania danymi (ang. Data Manipulation Language). Umożliwia przygotowanie aplikacji bazy oraz predykatów zewnętrznych.

W zależności od rodzaju bazy istnieje wiele języków regułowych. Ich korzenie wywodzą się ze sztucznej inteligencji. Tradycyjne systemy działają na podstawie reguły dedukcyjnej zwanej regułą odrywania, którą zapisuje się następująco:

$$\frac{(\alpha \Rightarrow \beta), \alpha}{\beta}$$

Reguła ta oznacza, że jeżeli z przesłanki α wynika przesłanka β oraz α jest prawdziwe, to przyjmujemy, że fakt β jest również prawdziwy. Regułę można uznać za odpowiednik reguły wnioskowania zwanej w logice arystotelesowskiej jako *modus ponen-*

do ponens. Bardzo często przyjmuje się, że wystąpienie faktu już świadczy o jego prawdziwości. Ogólnie reguła ma postać: wzorzec ⇒ akcja. Reguła jest uaktywniana wtedy, gdy wzorzec (warunek, predykat) odpowiada danym znajdującym się w pamięci roboczej systemu, natomiast akcja dokonuje ich modyfikacji. Reguły w systemach baz danych przyjmują następująca postać:

ON zdarzenie
IF warunek
THEN akcja.

Reguła uaktywniana jest tylko wtedy, gdy wystąpi określone zdarzenie. Warunek reguły zdefiniowany w treści reguły jest sprawdzany na podstawie danych. Jeśli warunek jest spełniony, to podejmowana jest określona akcja. Ten sposób reprezentacji znany jest jako reguły ECA (ang. *Even Condition Action*). Szczegóły i stopień złożoności specyfikacji zdarzeń, warunków i akcji różnią się w znacznym stopniu w poszczególnych systemach. Niektóre systemy są wyposażone w mechanizmy porządkowania reguł, których celem jest określenie jaka reguła powinna być wykonana w sytuacji, gdy wiele reguł zostało uaktywnionych. Związki pomiędzy poszczególnymi elementami reguł występujące w językach regułowych są bardzo zróżnicowane. W większości systemów reguły są uaktywniane przez operacje wprowadzania, usuwania i aktualizacji danych (INSERT, DELETE i UPDATE) zawartych w określonej relacji. Do każdej wprowadzanej, usuwanej lub aktualizowanej krotki może nastąpić odwołanie w części warunkowej i akcji reguły. Większość języków regułowych stwarza możliwość uaktywniania reguł na podstawie operacji dostępu do danych. Innym typem uaktywniającym reguły są zdarzenia czasowe, które uaktywniają reguły w określonym czasie lub przedziale czasu. We wszystkich językach regułowych część warunkowa określa warunek lub zapytanie dotyczące informacji przechowywanej w bazie. Warunek może być pominięty w definicji reguły. W wielu językach regułowych warunki mogą odnosić się do modyfikowanych danych lub stanu bazy przed modyfikacją Akcja reguły określa operacje jakie mają być wykonane, jeśli część warunkowa reguły jest spełniona. Akcje mogą być sekwencjami operacji wyszukiwania i modyfikacji danych. Akcja może odnosić się do danych, których modyfikacja spowodowała, że reguła została uaktywniona. Na przykład reguła:

ON APPEND TO *badanie*
DO DELETE *badanie*
WHERE *badanie.nazwa = nowa_nazwa*

jest uaktywniana zawsze, gdy do tabeli zawierającej rodzaj badań dopisywane są nowe badania. Akcja tej reguły polega na usunięciu istniejącego badania, jeśli nazwa nowego badania jest taka sama.

Jednym z istotnych problemów zarządzania regułami jest kwestia wyboru reguły w sytuacji, gdy w wyniku wystąpienia określonego zdarzenia zostanie uaktywnionych wiele reguł. W tym przypadku istnieje wiele strategii postępowania. Do znanych należą strategie: świeżości, blokowania, specyficzności i przypadkowości. Strategia świe-

żości polega na określeniu reguły, która najpóźniej została dołączona do reguł. Zadaniem strategii blokowania jest eliminacja tych reguł, które wcześniej w procesie wnioskowania były wykorzystane. Strategia specyficzności opiera swoje działanie na uwzględnieniu dużej liczby różnych przesłanek w regułach. Są preferowane reguły mające większą liczbę przesłanek, z których jest wybierana przesłanka o mniejszej liczbie zmiennych. Wszystkie te strategie spełniają w programie funkcje pewnego rodzaju filtrów, które maja ograniczyć liczbę reguł tak, aby wybrać jedną. Jeśli w wyniku zastosowania wymienionych strategii istnieje ciągle więcej niż jedna reguła do uaktywnienia, to stosuje się strategię przypadkowości, która wybiera regułę w sposób przypadkowy.

7.4. Metody wnioskowania

Jeśli chodzi o techniki wnioskowania, to wyróżniamy wnioskowanie progresywne, regresywne i mieszane. Osobną grupę stanowią techniki wnioskowania wykorzystujące wiedzę niepewną, wśród których szczególna rolę odgrywa wnioskowanie rozmyte.

Idea wnioskowania progresywnego jest niezwykle prosta. Na podstawie dostępnych reguł i faktów należy generować nowe fakty tak długo, aż wśród wygenerowanych faktów znajdzie się postawiona hipoteza. Podstawową cechą tego sposobu wnioskowania jest zwiększanie się bazy faktów, co czasami staje się zjawiskiem niepożądanym. Przyspiesza to co prawda proces sprawdzania postawionej hipotezy, jednak zajmuje niepotrzebnie pamięć komputera.

Algorytm wnioskowania progresywnego.

BW: Wiedza początkowa/Fakty
REPEAT
1. Określ zbiór **C** reguł w bazie wiedzy **BW**, dla których są spełnione przesłanki.
2. Ze zbioru **C** wybierz regułę **R** na podstawie strategii sterowania wnioskowaniem.
3. **BW**: = Wynik uaktywnienia reguły **R** działający na **BW** plus **BW**.

UNTIL Osiągnięto cel lub nie można zastosować więcej reguł.

Ilustrację opisanego algorytmu dla bazy wiedzy zawierającej cztery reguły oraz trzy fakty można znaleźć w pracy [35].

Wnioskowanie regresywne przebiega w odwrotną stronę niż wnioskowanie progresywne. Ogólnie polega ono na wykazaniu prawdziwości hipotezy głównej na podstawie prawdziwości przesłanek. Kiedy nie wiadomo, czy jakaś przesłanka jest prawdziwa, wtedy traktowana jest jako nowa hipoteza i należy ją wykazać. Jeśli w wyniku takiego postępowania zostanie wreszcie znaleziona reguła, której wszystkie przesłanki są prawdziwe, to konkluzja tej reguły jest prawdziwa. Na podstawie tej konkluzji do-

wodzi się następną regułę, której przesłanka nie była poprzednio znana. Postawiona hipoteza jest prawdziwa, jeśli wszystkie postawione przesłanki dadzą się wykazać.

Algorytm wnioskowania regresywnego.

BW: Wiedza początkowa/Fakty
REPEAT

1. Określ zbiór **C** reguł w bazie wiedzy **BW**, których konkluzje dają się są zunifikować.
2. Ze zbioru **C** wybierz regułę **R** na podstawie strategii sterowania wnioskowaniem.
3. Jeśli przesłanka reguły **R** nie znajduje się w bazie wiedzy **BW**, dokonaj wnioskowania regresywnego dla przesłanki reguły **R** (aby ja udowodnić).

UNTIL Hipoteza zostanie wykazana lub nie można zastosować więcej reguł.

Ilustrację opisanego algorytmu dla bazy wiedzy można znaleźć w pracy [35].

Zasadniczą cechą, która odróżnia wnioskowanie regresywne od progresywnego jest mniejsza liczba generowanych nowych faktów oraz niemożność równoczesnego dowodzenia kilku hipotez. Ogólnie w typowych zastosowaniach wnioskowanie regresywne jest efektywniejsze, bardziej rozpowszechnione oraz czas oczekiwania na osiągniecie rozwiązania postawionej hipotezy jest w wielu przypadkach dużo krótszy niż przy wnioskowaniu progresywnym.

Kompromis między wnioskowaniem progresywnym i regresywnym stanowi wnioskowanie mieszane, dzięki czemu jest pozbawione niektórych wad opisywanych metod. Strategia tego wnioskowania opiera się na wykorzystaniu ogólnych reguł, zwanych metaregułami, stanowiących metawiedzę, na podstawie której program zarządzający dokonuje odpowiedniego przełączania między poszczególnymi rodzajami wnioskowania. W zależności od sytuacji system może automatycznie dobrać odpowiedni sposób wnioskowania. Przy przechodzeniu z jednego wnioskowania na drugie za hipotezę główną zawsze wybiera się tę, którą postawił użytkownik. Dzięki temu na każdym etapie wnioskowania istnieje możliwość udzielania odpowiedzi na postawioną hipotezę. We wnioskowaniu mieszanym system musi mieć wprowadzony oprócz bazy wiedzy również zbiór zawierający metareguły. W działaniu systemu można wyróżnić jakby dwie maszyny wnioskujące: progresywną i regresywną. Wiedza zapisana w metaregułach może preferować jeden z rodzajów wnioskowania. Na początek zakłada się, że priorytet ma wnioskowanie regresywne. W związku z tym stosuje się go tak długo, dopóki da się wykorzystać wszystkie reguły związane z wybranym wnioskowaniem. Po każdym cyklu wnioskowania są sprawdzane warunki zapisane w metaregułach. Jeśli nie da się dalej stosować wnioskowania regresywnego, to system jest przełączany na wnioskowanie progresywne. Po wyprowadzeniu kolejnych faktów sprawdza się, czy system uzyskał odpowiedź na postawioną hipotezę.

Jeśli tak, to wynik stanowi rozwiązanie. W przeciwnym razie proces jest kontynuowany aż do osiągnięcia celu lub gdy zostaną wyczerpane wszystkie możliwości jego wykazania.

Niżej podamy przykład wnioskowania mieszanego.

• *Przykład 7.1*

Baza reguł podzielona jest na dwie części zawierające 8 reguł. Część bazy reguł związana z wnioskowaniem regresywnym zawiera reguły:

R1 **F** and **H** $\Rightarrow$ **K**
R2 **E** and **A** $\Rightarrow$ **K**
R3 **E** and **B** $\Rightarrow$ **H**

Część bazy związana z wnioskowaniem progresywnym zawiera reguły:

R4 **A** and **G** $\Rightarrow$ **B**
R5 **B** and **D** $\Rightarrow$ **H**
R6 **G** and **D** $\Rightarrow$ **E**
R7 **A** and **B** $\Rightarrow$ **D**
R8 **A** and **C** $\Rightarrow$ **G**

Założono, że należy wykazać hipotezę **K**. Prawdziwe są fakty **A** i **C**, a priorytet ma wnioskowanie regresywne. O wyborze reguły decyduje kolejność umieszczenia na liście. Tak więc jako pierwsza zostanie użyta reguła **R1**, której część warunkowa zawiera dwa nieznane fakty **F** i **H**. Należy wykazać prawdziwość tych faktów. Ponieważ w konkluzji reguły **R3** znajduje się fakt **H**, więc w drugim kroku zostanie wybrana ta właśnie reguła. Ponieważ w części warunkowej tej reguły znajdują się dwa nowe fakty **E** oraz **B**, więc do wykazania mamy w sumie trzy fakty **F, E, B.** Dla rozważanej bazy wiedzy nie można zastosować więcej reguł przy wnioskowaniu regresywnym, wobec czego system zostaje przełączony na wnioskowanie progresywne. Wobec znanych faktów **A** i **C** będą kolejno uaktywniane następujące reguły **R8, R4, R7, R6, R5,** przy czym kolejno zostaną wprowadzone nowe fakty **G, B, D, E, H** do bazy faktów. Ponieważ zostały uaktywnione wszystkie fakty, w tej części przechodzimy do wnioskowania regresywnego. Z faktów **F, E, B**, które poprzednio przy wnioskowaniu regresywnym należało wykazać, tylko fakt **F** nie został wykazany. Oznacza to, że postawionej hipotezy **K** nie da się wyprowadzić z reguł **R1** i **R3**. Wobec tego należy rozważyć inne reguły, których części warunkowe nie były jeszcze badane. Regułą taką jest reguła **R2.** System sprawdza, czy fakty **E** oraz **A** znajdują się w bazie, jeśli tak, to reguła **R2** zostanie uaktywniona i fakt **K** zostaje uznany za prawdziwy, co kończy proces wnioskowania.

Można zauważyć, że proces wnioskowania przeprowadzony był w 10 krokach. Gdyby została przeprowadzona analiza wnioskowania mieszanego z priorytetem pro-

gresywnym, proces wnioskowania trwałby 7 kroków. Nie należy jednak wyciągać wniosku, że nadanie priorytetu wnioskowaniu progresywnemu przyspiesza uzyskanie wyników. Ogólnie o tym, który sposób wnioskowania jest efektywniejszy świadczy: struktura bazy wiedzy, kolejność umieszczania reguł w bazie oraz zastosowane strategie sterowania wnioskowaniem tworzące metawiedzę o rozwiązaniu danego problemu.

7.5. Podstawowe funkcje systemu zarządzania dedukcyjną baza danych

Wszelkie operacje prowadzone na faktach i regułach zorganizowane są w postaci zbiorów elementarnych jednostek zwanych transakcjami. Każda transakcja musi być wykonana w całości lub w całości wycofana. W przypadku pomyślnego wykonania transakcji wszystkie wprowadzone przez nią zmiany do bazy muszą być zachowane. Transakcja nie ma prawa naruszać spójności bazy, to znaczy, że po wykonaniu transakcji muszą być zachowane wszystkie wcześniej zdefiniowane więzy integralności. System zarządzania bazą dedukcyjną powinien charakteryzować się własnościami, które zapewnią bezpieczny i efektywny dostęp do bazy wielu użytkownikom. W związku z tym musi realizować następujące funkcje:

– ochrony spójności i trwałości danych,
– zarządzania współbieżnością transakcji,
– autoryzacji dostępu do danych,
– fizycznego zarządzania danymi,
– weryfikacji więzów integralności,
– optymalizacji wykonywania operacji, a w szczególności procesu wnioskowania.

Część z tych funkcji implementowana jest już w bazach relacyjnych, inne, jak na przykład weryfikacja czy optymalizacja, wymagają nowych rozwiązań w stosunku do możliwości oferowanych przez bazy relacyjne.

7.6. Realizacja zapytań i ich optymalizacja

Ponieważ większość ludzkiej wiedzy nagromadzona jest w postaci tekstu, więc tekst jest najważniejszym komponentem technologii inteligentnych baz danych. Problem polega na powiązaniu wiedzy tekstu podstawowego z reprezentacją wiedzy, którą można wykorzystać do wnioskowania. Chodzi tu nie tylko o natychmiastowe wyszukiwanie informacji, ale o stworzenie procesów przekształcających tekst w reprezentację wiedzy, która mogłaby być zrozumiała i wykorzystywana przez systemy wnioskujące. To wszystko wiąże się nierozerwalnie z technikami reprezentacji, gromadzenia, zarządzania i dostępu do wiedzy. Najbardziej popularnymi metodami reprezentacji wiedzy i jej strukturalizacji są: sieci semantyczne, tabele decyzyjne

i drzewa decyzyjne [35].W dedukcyjnych bazach informacje jakie uzyskuje użytkownik są wynikiem procesów wnioskujących. Algorytmy wartościowania zapytań kierowanych do bazy powinny charakteryzować się: poprawnością, kompletnością i skończonością uzyskanych odpowiedzi na zadane pytania. Istnieje wiele rodzajów tego typu algorytmów, w zależności od na przykład kolejności stosowania reguł podczas procesu wnioskowania, czy liczby faktów, które są pobierane z predykatów bazowych w jednym cyklu procesu wnioskowania. Każda z grup algorytmów wymaga stosowania dodatkowych mechanizmów optymalizacji przeprowadzanych współbieżnie z samą metodą. Zaproponowane metody optymalizacji można podzielić na: syntaktyczne i semantyczne. Metody syntaktyczne opierają się na precyzyjnej analizie postaci zapytania oraz reguł wnioskowania, których należy użyć w celu uzyskania odpowiedzi. Przykładowo, wynikiem tej analizy może być zmiana kolejności wartościowania podzapytań lub zmiana kolejności użycia alternatywnych reguł tego samego predykatu. Metody semantyczne opierają się na dodatkowej wiedzy, która zwykle nie jest jawnie przechowywana w bazie. Warto tu wspomnieć o więzach integralności, które w stosunku do baz relacyjnych zostały znacznie rozszerzone. W bazach dedukcyjnych więzy są definiowane za pomocą reguł logicznych. Transakcja modyfikująca fakty lub reguły dedukcyjnej bazy może zostać zatwierdzona wyłącznie wtedy, gdy w wyniku swego działania nie naruszyła zdefiniowanych w bazie więzów. Więzy bardzo ułatwiają precyzyjne modelowanie rzeczywistości, mogą jednak przyczynić się do istotnego zmniejszenia efektywności systemu. Wynika to z konieczności odczytu i udowodnienia bardzo dużej liczby faktów przed podjęciem decyzji o zatwierdzeniu lub odrzuceniu transakcji. Więzy można podzielić na natychmiastowe i opóźnione. Więzy natychmiastowe muszą być spełnione po wykonaniu każdej elementarnej operacji transakcji. Więzy opóźnione muszą być spełnione po zakończeniu ostatniej elementarnej operacji składającej się na transakcję i mogą być przejściowo naruszone w trakcie realizacji transakcji. Można też wyróżnić więzy statyczne i dynamiczne. Więzy statyczne dotyczą pojedynczego stanu modelowanej rzeczywistości. Więzy dynamiczne ograniczają możliwe zmiany stanów. Zarówno więzy statyczne jak i dynamiczne mogą być podzielone ze względu na liczbę faktów, które muszą zostać przeanalizowane w trakcie ich weryfikacji. Wyróżniamy tutaj więzy zagregowane i niezagregowane [2, 6].

7.7. Uwagi końcowe

Budowa systemów opartych na modelu dedukcyjnym nie jest prosta i wymaga ścisłego współdziałania całej grupy osób: ekspertów, inżynierów wiedzy, programistów. Dodatkowym utrudnieniem jest fakt, że ta dziedzina nie posiada własnej metodologii. Obecnie stosowane techniki wydają się dostępne do prostych zastosowań, natomiast do bardziej skomplikowanych ich budowa i wdrażanie stają się trudne praktycznie na

każdym etapie implementacji. Do najbardziej skomplikowanych problemów, z którymi boryka się metodologia tworzenia systemów w dedukcyjnych bazach należą:

– problem adekwatnej reprezentacji wiedzy,
– rozumowanie w przypadkach wiedzy niepewnej,
– problemy z gromadzeniem wiedzy,
– interfejs pomiędzy systemem a użytkownikiem.

Za wąskie gardło przy tworzeniu systemów dedukcyjnej bazy danych uważa się ekstrahowanie i gromadzenie wiedzy. Ekspertyza nie zawsze jest łatwo dostępna. Różni eksperci często diametralnie różnią się między sobą podejściem do tego samego problemu. Pomimo pewnego postępu, w eksperymentach z automatyzacją procesu ekstrahowania wiedzy bezpośrednie jej gromadzenie nadal nie jest praktycznie możliwe. Baza wiedzy budowana jest na podstawie wiedzy pozyskanej od ekspertów przez inżyniera wiedzy. Jeżeli ta wiedza okaże się niepełna, niepewna, zróżnicowana zależnie od punktu widzenia eksperta lub nieprawidłowa, to te same błędy pojawią się w bazie wiedzy systemu. Błędy semantyczne często są też wynikiem przekłamań spowodowanych przez inżyniera wiedzy na skutek nieporozumienia lub błędnej interpretacji informacji uzyskanej od specjalisty. Błędy semantyczne i syntaktyczne mogą powstać na skutek użycia niewłaściwej formy reguł wnioskowania lub wybrania niewłaściwej formy interpretacji wiedzy. W obecnym stadium technologia systemów inteligentnych nie potrafi sobie jeszcze efektywnie radzić z reprezentacją wiedzy różnego typu w ramach jednego systemu. Częstym źródłem błędów jest też wadliwie zaprojektowany moduł interfejsu między użytkownikiem a systemem, który często dostarcza użytkownikowi niekompletnych, niejasnych lub wręcz mylnych informacji. Obecne systemy eksperckie nie są w stanie samodzielnie weryfikować swojej poprawności ani uściślać bazy wiedzy. Obie te czynności muszą być inicjowane z zewnątrz i wykonywane przez inżynierów wiedzy przy współpracy z samymi ekspertami.

Przykład systemu opartego na modelu dedukcyjnym podano w rozdz. 10 części III.

PODSUMOWANIE

Część druga zawiera opis trzech reprezentatywnych modeli danych: relacyjnego, obiektowego i dedukcyjnego. W latach osiemdziesiątych zapowiadano [4], że systemy mające za podstawę model obiektowy znacznie się różniący od modelu relacyjnego, prześcigną na początku lat dziewięćdziesiątych systemy oparte na modelu relacyjnym danych. Obecnie mamy rok 2001, a bazy relacyjne nadal są podstawą większości systemów komercyjnych. Bazy relacyjne wymagają, aby wszystkie dane były przedstawiane jako serie dwuwymiarowych tabel. Systemy oparte na modelu relacyjnym były rzeczywiście wielkim wydarzeniem w latach osiemdziesiątych, lecz programiści bardzo szybko przekonali się, że „życie nie jest serią dwuwymiarowych tabel". Wzrost złożoności współczesnych programów i wzrost użycia dynamicznych modeli danych ukazał ograniczenia relacyjnych baz danych. Wielu producentów systemów relacyjnych zaczyna oferować modele obiektowe [ORACLE 8] i twierdzi, że nowe wersje języka SQL mają cechy obiektowości. Również bazy dedukcyjne mają poszerzony zakres typów atrybutów w stosunku do baz relacyjnych. Dzięki wyrażeniu więzów integralności za pomocą reguł logiki pierwszego rzędu możliwe jest bardzo precyzyjne modelowanie powiązań między encjami i ograniczeń występujących w modelowanej rzeczywistości. Nie ma ograniczeń w modyfikowaniu reguł i więzów integralności. Bazy dedukcyjne mają nowe funkcje, na przykład funkcje warunkowe, uaktualniania hipotetycznego czy też rekurencyjne predykaty, które istotnie zwiększają zakres zastosowań.

BIBLIOGRAFIA

[1] Bancilhon F., Delobel C., Kanellakis P., *Building an Object – Oriented Database System: the story of O2*, Morgan Kaufmann 1992.

[2] Bayer P., Lefebvre A., Vieille L., *Architecture and Design of the EKS Deductive Database System*, VLDB Journal on Prototypes of Deductive Database Systems 1993.

[3] Bernstein P.A., *Synthesizing Third Normal Form Relations for Functional Dependencies*, ACM Transactions on Database Systems 1976, Vol. 1, No. 4 s. 277–298.

[4] Beynon-Davies P., *Systemy baz danych*, WNT, Warszawa 1998.

[5] Beynon-Davies P., *Information Systems Development; an introduction to information systems engineering*, Macmillan Press 1993.

[6] Beynon-DaviesP., Expert *Database Systems: a gentle introduction*, Maidenhead, MacGraw-Hill 1991.

[7] Beynon-Davies P., *Knowlege Engineering for Information Systems Development; an introduction to information systems engineering*, Maidenhead, MacGraw-Hill 1993.

[8] Beynon-Davies P., Tudhope D., Taylor C., Jones C.B., *A Semantic Data-base Approach to Knowlege – Based Hypermedia Systems*, Information and Software Technology, June 1994, 36(6), s. 323–329.

[9] Blaha M.R., Premerlani W.J., Rumbaught I.E., *Relational Database Design Using an Object-Oriented Methodology CACM*, 31(4), s. 414–427.

[10] Bratko I., *Prolog Programming for Artificial Intelligence*, 2nd Ed. Reading, Mass. Addison-Wesley 1990.

[11] Cattell R.G.G., *The Object Database Standard ODMG-93 Reease 1.1*, Morgan Kaufmann 1994.

[12] Chwiałkowska E., *Sztuczna inteligencja w systemach ekspertowych*, Zakład Nauczania Informatyki Micom, Warszawa 1991.

[13] Codd E.F., *A Relational Model for Large Shared Data Banks Communications of ACM*, June 1970, Vol. 13, No. 6, s. 377–387.

[14] Codd E.F., *Further Normalization of the Date Base Relational Model*, Englewood Cliffs: Prentice-Hall 1972, s. 33–64.

[15] Codd E.F., *The Relational Model for Database Management: Version 2*. Reading, Mass. Addison-Wesley 1990.

[16] Codd E.F., *Extending the relational Database Model to Capture More Meaning*, ACM Transactions on Database Systems, Dec. 1979 Vol. 4, No. 4, s. 397–434.

[17] Das S.K. *Deductive Databases and Logic Programming*, Adison-Wesley Publishing Company 1981.

[18] DBTG: *Report of the CODASYL Database Task Group*, ACM, April.

[19] Date C., *Introduction to Datebase Systems 5th Ed*, Addison-Wesley 1990.

[20] Delobel C., Lecluse C., Richard P., *Database from relational to object-oriented systems International Thompson Publishing*, 1995.

[21] Delobel C., Adiba M., *Relacyjne bazy danych*, WNT, Warszawa 1989.

[22] Fagin R., *Multi-Valued Dependencies and a New Normal Form for Relational Database*, ACM Tran. of Database Systems 1977, 2(1).

[23] Fagin R., *Normal Form and Relational Database Operarators ACM SIGMOD*, Int. Symposium on the Management of Data, 1979, s. 153–160.

[24] Gardarin G., Valduriez P., *Relational Databases and Knowledge Bases*, Reading Mass. Addison--Wesley 1990.
[25] Kent W., *A Simple Guide to Five Normal Forms in Relational Database Theory*, CACM 1983, 26(2).
[26] MacVittie D.W., MacVittie L.A., *Programowanie zorientowane obiektowo*, Mikom, Warszawa 1996.
[27] Martin J., *Organizacja baz danych*, PWN, Warszawa 1983.
[28] Miller T., Powell D., *Księga eksperta*, Helion, Gliwice 1998.
[29] Myer B., *Object Oriented Software Construction*, New York, Prentice-Hall 1997.
[30] Mulawka J.J., *Systemy ekspertowe*, WNT, Warszawa 1996.
[31] Oszu M.T., Valduriez P., *Principles of Distributed Database Systems*, Englewood-Cliffs, New York, Prentice Hall 1991.
[32] Oszu M.T., Valduriez P., *Distributed Database Systems: where are we now?* Database Programming and Design, March 1992.
[33] Pankowski T., *Podstawy baz danych*, PWN, Warszawa 1992.
[34] Prabhu C.S.R., *Semantic Database Systems: a Functional introduction*, London, Sangam 1992.
[35] Prace: Kuśmierski W., *Dedukcyjna baza danych modelująca laboratorium medyczne*, dyplom pod kierunkiem M. Chałon, ICT PWr.,Wrocław 1997.
Feluś T., Mrówczyński M., *Obiektowo zorientowana baza danych wspomagająca zarządzanie*, dyplom pod kierunkiem M. Chałon, ICT, PWr., Wrocław 1999.
[36] Rissanen J., *Independent Components of Relations*, ACM Transactions on Database Systems 1977, Vol. 2, No. 4, s. 317–325.
[37] Rissanen J., *Theory of relations for Database – a Tutorial Survey*, Proc. MFCS. Lecture Notes in Computer Science, Berlin–Heidelberg 1978, Vol. 64, s. 537–551.
[38] Tsichritzis D.C., Lochovsky F.H., *Modele danych*, WNT, Warszawa 1990.
[39] Tsur S., Naqvi S., *A Logical Language for Data and Knowledge Bases*, Computer Science Press, New York 1989.
[40] Teorey T.J., *Database Modelling and Design: the Fundamental Principles 2nd Ed*. San Mateo, Calif. Morgan Kaufmann 1994.
[41] Winston P.H., *Artificial intelligence*, Reading, Mass. Addison-Wesley 1984.

CZĘŚĆ III

Stosowanie modeli danych

Jakość modelu danych jest wyznaczona jedynie jego użytecznością przy rozważaniu, organizacji i użyciu danych. Tak jak w większości sytuacji konkretny problem wyznacza środki potrzebne do jego rozwiązania. Ponieważ różne modele danych dostarczają różnych narzędzi do ich modelowania, więc użyteczność modeli zależy od problemów, do których są stosowane.

D. Tsichritzis, F. Lochovsky

WPROWADZENIE

Jednym z istotnych zagadnień przy wyborze modelu danych, a co za tym idzie SZBD przez projektanta, a potem przez użytkownika jest jego złożoność. Uważa się, że im mniej złożony jest model, czyli czym większą prostotą się charakteryzuje, tym łatwiej użytkownikowi zrozumieć go i poprawnie używać. Argument prostoty jest często podnoszony przez zwolenników relacyjnych baz danych. Można wyróżnić dwa rodzaje złożoności: złożoność struktury i złożoność więzów. W obu przypadkach relacyjne modele danych są mniej złożone niż obiektowe czy dedukcyjne. Jednakże brak złożoności może być wadą wtedy, gdy brakuje mechanizmów wspomagających użytkownika w interpretacji danych. Innym zagadnieniem przy wyborze modelu danych jest dopasowanie struktury danych do możliwości ich modelowania. Jeżeli dane są w sposób naturalny hierarchiczne, to trudno jest przedstawić je w postaci tabeli dwuwymiarowej. Na wybór modelu danych do konkretnej aplikacji może mieć wpływ rodzaj języka manipulacji danych. Dąży się do stworzenia jednego uniwersalnego języka, jak np. SQL, z możliwością jego modyfikowania w zależności od potrzeb modelu. Ciekawym zagadnieniem jest wybór między językiem naturalnym a sztucznym. Badania wykazały [4, 5, 19], że użytkownik korzystający z języka naturalnego często formułuje nieracjonalne żądania do bazy ze względu na zbyt dużą swobodę przy zadawaniu pytań. Języki sztuczne zazwyczaj narzucają użytkownikowi sposób zadawania pytań.

Na podstawie omówionych kryteriów można wybrać odpowiednie modele danych dla każdej modelowanej dziedziny. W rozdziałach ósmym, dziewiątym i dziesiątym podano przykłady konkretnych systemów zbudowanych na modelach relacyjnym, obiektowym i dedukcyjnym.

8. NIEPROCEDURALNY JĘZYK CZWARTEJ GENERACJI

8.1. Uwagi ogólne

Języki służące do wyszukiwania danych dzielimy na: proceduralne i nieproceduralne. W językach proceduralnych użytkownik opisuje procedurę wyszukiwania i uzyskiwania danych na pewnym poziomie szczegółowości. Wyróżniamy *wyszukiwanie jednostkowe*, którego przykładem może być sieciowa baza danych, oraz *wyszukiwanie grupowe*, którego przykładem jest definicja ciągu operacji algebry relacji. W językach nieproceduralnych użytkownik podaje warunki, jakie powinien spełniać żądany przez niego wynikowy zbiór danych. Wyrażenia nieproceduralne muszą być przetłumaczone w systemie na ciąg wyrażeń języka proceduralnego. W podejściu nieproceduralnym wykorzystuje się jeden z języków rachunku relacji: rachunek krotek lub rachunek domen. Relacyjny rachunek na krotkach stał się podstawą języka SQL [7], natomiast relacyjny rachunek na domenach podstawą interfejsów QBE (zapytanie przez przykład). Pierwowzorem SQL był język SEQUEL [5, 6, 15, 19, 21]. Definicja składni standardu języka relacyjnych baz danych SQL po raz pierwszy została opublikowana w 1986 roku w oparciu o dwa dialekty SQL IBM i Oracle [8]. Jej ulepszona wersja SQL1 powstała rok później. W 1989 roku [9] opublikowano wersję SQL89 zawierającą głównie poprawę integralności bazy. Dopiero w 1992 wydano pełną specyfikację rozszerzonej wersji pod nazwą SQL2 [10]. Ta różnorodność wersji sprawia, że nie ma jednolitego standardu SQL, jest wiele dialektów tego języka. Ponadto stale powstają nowe wersje SQL.

Instrukcje języka SQL dzieli się z reguły na cztery grupy zwane również językami. Są to:

– instrukcje definiowania danych,
– instrukcje manipulowania danymi,
– zapytania,
– instrukcje zarządzania danymi.

8.2. Instrukcje definiowania danych

Instrukcje te służą do tworzenia, modyfikowania i usuwania obiektów tworzących bazę danych. Obiektami tymi są tabele, perspektywy, synonimy, indeksy, schematy, katalogi. W skład języka wchodzą między innymi takie instrukcje, jak: CREATE

obiekt powodująca utworzenie obiektu danego typu, ALTER *obiekt* wykorzystywana do modyfikowania tablic, DROP *obiekt* służąca do usuwania z bazy obiektu określonego typu. Ponadto również integralność danych włączona jest do instrukcji definiowania danych. Ze względu na dużą wagę problemu oraz rodzaj komend, których używa się przy określaniu różnych typów integralności zagadnienie to zostanie omówione w osobnym rozdziale.

Aby utworzyć obiekt, na przykład tabelę, użytkownik określa następujące składniki:
– nazwę tabeli,
– nazwę każdej kolumny w tabeli,
– typ danych,
– szerokość każdej kolumny,
– opcjonalny parametr (NULL).

Typy danych określają pewne właściwości dotyczące dopuszczalnych wartości danych w kolumnie. Każda wartość w kolumnie musi być tego samego typu. Standard SQL [10] (ISO 1992) definiuje około 15 typów danych podzielonych na grupy: typy napisowe, typy liczbowe, typy daty i godziny, przedziały między datami itp. Jako parametr opcjonalny stosuje się słowo kluczowe NOT NULL określające, że podczas wprowadzania nowego rekordu do tablicy wartość pola odpowiadającego kolumnie zadeklarowanej jako NOT NULL musi być ustalona lub słowo NULL, gdy wartość ta może być nieustalona. Jeżeli parametr opcjonalny nie jest określony, to przyjmuje się, że kolumna jest typu NULL. Każda kolumna może być również zdefiniowana jako UNIQUE, czyli jednoznaczna. Klauzula ta zabrania wprowadzania do kolumn powtarzających się wartości. Kombinację NOT NULL i UNIQUE można użyć do zdefiniowania cech klucza głównego.

• *Przykład 8.1*

Tworzymy tabelę o nazwie Pracownicy, która ma cztery kolumny a kluczem głównym jest NrPrac.

```
CREATE TABLE Pracownicy;
(NrPrac Number (5);
NazwiskoPrac Varchar (15);
NazwaWydziału Varchar (20);
Pensja Decimal (7,2);
PRIMARY KEY (NrPrac))
```

• *Przykład 8.2*

Dla tabeli z poprzedniego przykładu zdefiniowano cechy klucza głównego.

```
CREATE TABLE Pracownicy;
(NrPrac Number (5) NOT NULL UNIQUE;
NazwiskoPrac Varchar (15);
```

```
NazwaWydziału Varchar (20);
Pensja Decimal (7,2);
PRIMARY KEY (NrPrac))
```

Do definicji kolumny możemy dodać klauzulę DEFAULT <*wartość*> określającą wartość, którą system automatycznie wpisuje do kolumny w przypadku jeśli użytkownik wprowadzi niepełną informację.

- *Przykład 8.3*

Do kolumny NrWydziału w tabeli dodajemy specyfikację DEFAULT4, wskazującą, że domyślnym numerem Wydziału jest 4.

```
CREATE TABLE Pracownicy;
(NrPrac Number (5) NOT NULL UNIQUE;
NazwiskoPrac Varchar (15);
NazwaWydziału Varchar (20);
NrWydziału Smallint DEFAULT 4;
Pensja Decimal (7,2);
PRIMARY KEY (NrPrac))
```

Używając komendy CREATE tworzymy również perspektywy. Perspektywa jest logiczną strukturą, umożliwiającą dostęp do podzbioru kolumn i wierszy danej tabeli lub grupy tabel. Tworzona jest ze względów bezpieczeństwa i dla wygody. Jej organizacja jest taka sama jak tabeli, nie ma jednak własnych danych, a więc nie zajmuje dodatkowej pamięci. Pozwala na utajnianie pewnych danych zawartych w tabelach i upraszcza w wielu przypadkach postać zapytania kierowanego do bazy.

- *Przykład 8.4*

Tworzymy perspektywę Dochody z tabeli Pracownicy.

```
CREATE VIEW Dochody;
AS
SELECT NrPrac, NazwiskoPrac, Pensja;
FROM Pracownicy
```

Czasami z pewnych względów nazwa tabeli czy perspektywy jest niewygodna w użyciu, np. zbyt długa, wtedy można utworzyć synonim, a następnie użyć go zamiast nazwy tabeli czy perspektywy.

- *Przykład 8.5*

Tworzenie synonimu PU dla tabeli Pracownicy Uczelni.

CREATE SYNONIM PU;
FOR Pracownicy Uczelni

Aby przyspieszyć wykonywanie zapytań dotyczących tabel zawierających setki rekordów, tworzymy na podzbiorze ich kolumn indeksy.

- *Przykład 8.6*
  Tworzymy zbiór indeksowy WP.

CREATE INDEX Ind WP;
On Warunki Pracy

Jeżeli grupa kolumn ma tworzyć klucz główny tabeli, to znaczy, że w tabeli może być dokładnie jeden rekord o tej samej kombinacji wartości pól odpowiadających kolumnom klucza głównego. Przy tworzeniu indeksu należy wykorzystać słowo kluczowe UNIQUE np. CREATE UNIQUE INDEX.
Za pomocą instrukcji CREATE mogą być tworzone schematy. Schemat jest nadrzędny w stosunku do tabel i perspektyw, które stanowią jego części. Nazwa tabeli musi być jednoznaczna w danym schemacie, ale ta sama nazwa może wystąpić w wielu schematach. Aby nie było konfliktów z nazwami, trzeba kwalifikować nazwę schematu lub jawnie poprzedzać nazwę tabeli nazwą schematu.

- *Przykład 8.7*
  Tworzenie schematu Uczelnia.

CREATE SCHEMAT Uczelnia;
CREATE TABLE Pracownicy

Zmiana schematu może odbywać się za pomocą instrukcji SET SCHEMAT. Pojęciem szerszym od schematu jest katalog. Katalog jest nazwaną grupą schematów. Instrukcja tworzenia katalogu zależna jest od implementacji. Nazwy schematów w katalogu muszą być jednoznaczne.
Kolejna instrukcja CREATE DOMAIN służy do definicji dziedziny, czyli zbioru poprawnych wartości. Dziedzina jest określona w schemacie i jest identyfikowana przez swoja nazwę. Głównym celem użycia dziedziny jest ograniczenie zbioru poprawnych wartości, które mogą być przechowywane w kolumnie.

- *Przykład 8.8*
  Definicja dziedziny, która określa typ danych i klauzulę wartości domyślnej.

```
CREATE DOMAIN NrSprz;
AS INTEGER;
DEFAULT 9999;
CHECK (VALUE > 1000)
```

Niezależnie od jakiejkolwiek tabeli lub dziedziny mogą być tworzone i nazwane więzy zwane asercjami.

- *Przykład 8.9*
  Tworzenie więzów.

```
CREATE ASSERTION NrPracCheck;
CHECK (NrPrac BETWEEN 100 AND 10999)
```

Sprawdzanie tych więzów odbywa się niezależnie od jakiejkolwiek tabeli i może być wykonywane z opóźnieniem (DEFERRABLE) lub natychmiast (NOT DEFERRABLE). Początkowy sposób sprawdzania więzów może być określany jako INITIALLY DEFERRED lub INITIALLY IMMEDIATE. Za pomocą instrukcji SET CONSTRAINTS, która określa, czy dla listy nazwanych więzów należy wykonać sprawdzanie opóźnione czy natychmiastowe, można zmienić w czasie sesji tryb sprawdzania.

Obiekty utworzone za pomocą instrukcji CREATE można usuwać z bazy, stosując instrukcję DROP o składni DROP *typ obiektu*, *nazwa obiektu*, gdzie typem obiektu jest odpowiednio tabela, perspektywa, synonim czy indeks.

- *Przykład 8.10*
  Usuwanie tabeli Pracownicy.

```
DROP TABLE Pracownicy
```

Istnieje również możliwość modyfikacji struktury bazy danych, to znaczy zmiana definicji kolumn istniejących w bazie, dodanie lub usunięcie kolumny w bazie poprzez użycie polecenia ALTER.

- *Przykład 8.11*
  Dodanie kolumny w tabeli.

```
ALTER TABLE Pracownicy;
ADD COLUMN Nazwisko_profesora
```

Definicję kolumny zmienia się wprowadzając do instrukcji ALTER TABLE zdanie MODIFY.

• *Przykład 8.12*

Jeśli kolumna NrPrac ma pełnić funkcję klucza głównego tabeli Pracownicy, to wartości pól odpowiadające tej kolumnie muszą być określone.

```
ALTER TABLE Pracownicy;
MODIFY (NrPrac NUMER (5) NOT NULL)
```

8.3. Instrukcje manipulowania danymi

Instrukcje języka manipulowania danymi służą do wprowadzania nowych rekordów do tabel, do modyfikowania zawartości jednego lub większej liczby wierszy tabeli oraz do ich usuwania z tabeli. Przez te polecenia reprezentowana jest dynamika bazy. Najprostszą postacią instrukcji INSERT jest:

```
INSERT INTO nazwa tabeli;
VALUES (lista wartości pól)
```

Instrukcja o takiej składni służy do wprowadzania wartości do wszystkich pól rekordu. Oczywiście wartości poszczególnych pól muszą być uszeregowane w takim porządku, jak zadeklarowane są pola przy tworzeniu tabeli.

• *Przykład 8.13*

Jeśli chcemy wpisać wartości w jakimś innym porządku, to należy to zaznaczyć:

```
INSERT INTO Pracownicy;
VALUES (167,”Kowalski”,”Elektronika” 2000)
lub
INSERT INTO Pracownicy;
VALUES (111,”Ryt”,”Informatyka”, 2100)
```

Specjalna wersja polecenia INSERT umożliwia dodanie wielu wierszy do tabeli. Zwykle jest używana, aby umieścić wyniki jakiegoś zapytania w podanej tabeli.

• *Przykład 8.14*

Jeśli chcemy umieścić tabelę Pracowników pracujących na wydziale Elektroniki, to robimy to używając instrukcji SELECT.

```
CREATE TABLE Pracownicy Elektroniki;
(NrPrac Number (5);
NazwiskoPrac Varchar (15);
Pensja Decimal (7,2));
```

INSERT INTO Pracownicy Elektroniki (NrPrac, NazwiskoPrac, Pensja);
SELECT NrPrac, NazwiskoPrac, Pensja;
FROM Pracownicy;
WHERE NazwaWydziału = ”Elektronika”

Zmianę wartości pól jednego lub wielu wierszy tabeli można zrealizować używając instrukcji UPDATE. Składnia tej instrukcji jest następująca:

UPDATE *nazwa tabeli*
SET *zmiany do wykonania w klauzuli SET*
WHERE *warunek dla którego są zmiany*

- *Przykład 8.15*
  Zmiana wartości pola tabeli Pracownicy.

UPDATE Pracownicy;
SET Pensja = Pensja + 2000;
WHERE DataZatrudnienia < ”01.01.1996”

Wiersze z tabeli usuwa się za pomocą instrukcji DELETE o postaci:
DELETE FROM *nazwa tabeli*
WHERE *wyrażenie logiczne*

- *Przykład 8.16*
  Usuwanie wierszy z tabeli.

DELETE FROM Pracownicy;
WHERE NazwaWydziału = ”Elektronika”

Jeśli nieokreślony jest warunek WHERE, to wszystkie rekordy z tabeli są usuwane.

8.4. Zapytania

Zapytania służą do uzyskiwania informacji z tabel tworzących bazę. Wszystkie zapytania zaczynają się słowem kluczowym SELECT. Słowo to stanowi połączenie operatorów projekcji, konkatenacji i selekcji. Do prostego wyszukiwania używa się kombinacji klauzul SELECT FROM WHERE. Kombinacja ta zwana jest blokiem kwalifikacyjnym.

SELECT *atrybuty*
FROM *nazwa tabeli*
WHERE *warunki*

W celu zilustrowania zapytań wykorzystano konkretną bazę danych, której aplikacja znajduje się w opisie systemu FoxPro [12].

• *Przykład 8.17*

Dany jest diagram bazy danych pokazany na rys. 8.1 [12]. Baza składa się z sześciu zbiorów: Klient (Customer), Faktury (Invoices), Detal (Detail), Biura (Offices), Sprzedawca (Salesman), Części (Parts) określonych poprzez swoje atrybuty:

Customer = {**cno**, company, contact, address, city, state, zip, phone, **ono**, ytdpurch, lat, long},
Invoices = {**ino**, **cno**, idate, itotal, **salesman**},
Detail = {**ino**, line, qty, **pno**, price, ltotal},
Offices = {**ono**, itdsales, zmin, zmax, address, city, state, zip, phone},
Salesman = {**salesman**, **ono**, name, ytdsales, address, city, state, zip, phone, notes},
Parts = {**pno**, descript, onhand, onorder, price, cost, ytdunits, ytdsales}.

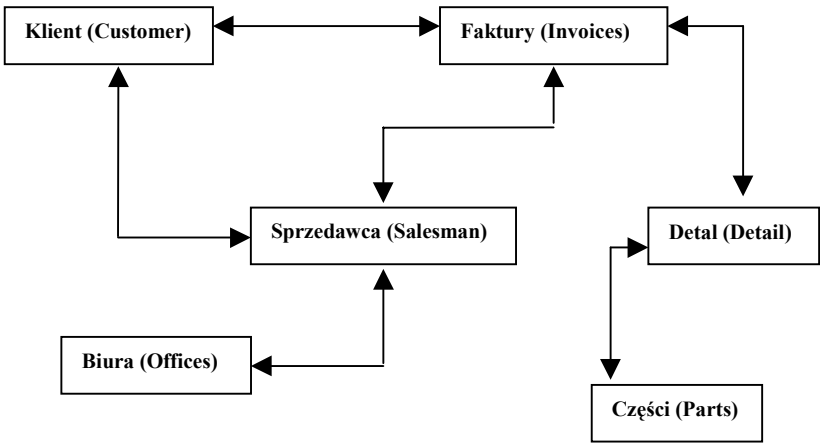
Wylistuj nazwy wszystkich spółek ze zbioru Klienci, które w swojej nazwie zawierają słowo ”Computer”.

```
SELECT company;
FROM Customer;
WHERE company LIKE ”%Computer%”
```

Operator LIKE ma zastosowanie w sytuacjach, gdy użytkownik chce wyznaczyć te rekordy, w których wartość określonego pola tekstowego spełnia pewien wzór, na przykład zaczyna się od określonej litery. Do definiowania wzorów wykorzystuje się specjalnie przydzielony znak, na przykład %. To samo zadanie można rozwiązać na dwa różne sposoby:

```
SELECT company;
FROM Customer;
WHERE ”Computer” & company
albo
SELECT company;
FROM Customer;
WHERE AT (”Computer”, company) > 0
```

Operator BETWEEN i AND umożliwia wyznaczenie wszystkich wierszy w tabeli, dla których wartość określonego pola należy do pewnego przedziału.



Rys. 8.1. Diagram przykładowej bazy danych

• *Przykład 8.18*

Ze zbioru Detale wybierz wszystkie detale, których cena zawarta jest w określonym przedziale cenowym.

```
SELECT Ino, price;
FROM Detail;
WHERE price BETWEEN 3000 AND 4000
```

Za pomocą operatora IN wyznacza się wszystkie rekordy, dla których wartość pewnego pola należy do określonego zbioru.

• *Przykład 8.19*

Ze zbioru Sprzedawca wybierz wszystkich sprzedawców zamieszkałych we Wrocławiu lub Warszawie.

```
SELECT name, city;
FROM Salesman;
WHERE city IN (”Wroclaw”,”Warszawa”)
```

Ponieważ relacja nie ma jawnego uporządkowania wierszy, można to zrobić stosując przetwarzanie relacji. Aby uzyskać posortowaną wyjściową listę, do instrukcji SELECT dodajemy klauzulę ORDER BY z odpowiednim słowem kluczowym (porządek malejący lub rosnący). W celu podsumowania wartości w tabeli używamy klauzuli GROUP BY. Klauzula GROUP BY może mieć swoją własną klauzulę ograniczającą HAVING.

• *Przykład 8.20*

Wylistuj oddziały firmy oraz sumaryczną wartość sprzedaży każdego oddziału. Uporządkuj wydruk według wartości sprzedaży od największego do najmniejszego.

```
SELECT Offices.city, SUM (Invoices.itotal);
FROM Offices, Invoices, Salesman;
WHERE Invoices.salesman = Salesman.salesman;
AND Salesman.ono = Offices.ono;
GRUP BY Offices.ono;
ORDER BY 2 DESCENDING
```

W przykładzie tym operacja SELECT działająca na trzech powiązanych przez wspólne kolumny tabelach (zaznaczono to w warunku WHERE) pozwala wybrać, podsumować i uporządkować w porządku malejącym informacje. Inny przykład pokazuje działanie klauzuli ograniczającej HAVING.

• *Przykład 8.21*

Wybierz te części, dla których suma wartości atrybutu qty jest większa od 50.

```
SELECT Detail.pno, Parts.descript, SUM(qty), SUM (qty*Detail.price);
FROM Detail, Parts;
WHERE Detail.pno = Parts.pno;
GROUP BY Detail.pno;
HAVING SUM (qty) > 50
```

W tym przykładzie łączymy dwie tabele, w warunku podajemy nazwy kolumn, według których nastąpiło połączenie oraz listujemy wartości kolumn wybranych w instrukcji SELECT zgodnie z klauzulą ograniczającą HAVING.

• *Przykład 8.22*

Wylistuj stany położone między 40 a 45 stopniem szerokości geograficznej, w których mieszkają klienci danej firmy.

```
SELECT state;
FROM Customer;
GROUP BY state;
HAVING 40 <= min (lat) AND max (lat) <= 45
```

To samo zadanie można wykonać inaczej, korzystając z możliwości zagnieżdżania zapytań w instrukcjach SELECT.

```
SELECT DISTINCT state;
FROM Customer;
WHERE state  NOT IN;
(SELECT state;
FROM Customer;
WHERE lat < 40 OR lat > 45)
```

Taki typ zapytań wprowadza redundancje informacji. SQL realizuje najpierw zapytanie umieszczone w nawiasach, tzw. podzapytanie. Uzyskany wynik jest porównywany z wynikiem zwracanym przez zewnętrzne zapytanie. Podzapytanie jest instrukcją SELECT zawartą w zdaniu wchodzącym w skład innej instrukcji SELECT.

- *Przykład 8.23*

  Znajdź klienta, który dokonał największego zakupu. Podaj nazwę firmy, nazwisko sprzedawcy oraz wartość zakupu.

```
SELECT Salesman.name, Customer.company, Invoices.ino, Invoices.idate, Invoices.
itotal;
FROM Salesman, Invoices, Customer;
WHERE Salesman.salesman = Invoices.salesman;
AND Invoices.cno = Customer.cno;
AND Invoices.itotal = (SELECT MAX(itotal) FROM Invoices)
```

Inne przykłady zawierające zagnieżdżoną instrukcję SELECT:

- *Przykład 8.24*

  Wylistuj stany, w których klienci nie dokonali żadnych zakupów.

```
SELECT DISTINCT state FROM Customer;
WHERE state NOT IN;
(SELECT Customer.state;
FROM Customer, Invoices;
WHERE Invoices.cno = Customer.cno)
```

- *Przykład 8.25*

  Podaj osobę, która więcej zarabia niż pracownik o nazwisku Jazz.

```
SELECT NrPrac, NazwiskoPrac;
FROM Pracownicy;
WHERE Pensja >;
(SELECT Pensja;
```

```
FROM Pracownicy;
WHERE NazwiskoPrac = ”Jazz”)
```

Wykonanie podzapytania może być powtarzane. Wtedy nazywa się ono podzapytaniem skorelowanym. W takim wypadku otrzymujemy ciąg wartości do porównywania z wynikami najbardziej zewnętrznego zapytania. Konieczne jest istnienie kopii właściwej tabeli.

- *Przykład 8.26*

W przykładzie tym mamy wypisać nazwiska wszystkich pracowników, ich pensje i nazwy wydziałów dla tych pracowników, którzy zarabiają więcej niż wynosi średnia pensja pracownika ich wydziału.

```
SELECT NazwiskoPrac, NazwaWydziału, Pensja;
FROM Pracownicy L;
WHERE Pensja >;
(SELECT AVG (Pensja);
FROM Pracownicy;
WHERE L NazwaWydziału = NazwaWydziału)
```

Tworzymy kopię tabeli Pracownicy o nazwie Pracownicy L. Jedna tabela służy do policzenia średniej pensji (funkcja AVG), druga jest podstawą do porównania wykonanego dla każdego pracownika. Możemy zatem nadawać nazwę alternatywną, zwaną aliasem, kolumnie lub tabeli w ramach kontekstu zapytania. Podobne zadanie z zastosowaniem dwukrotnym tej samej tablicy Sprzedawca podano niżej.

- *Przykład 8.27*

  Porównaj średnie roczne zarobki Sprzedawców.

```
SELECT a.salesman, a.name, a.ytdsales, AVG (b.ytdsales);
FROM Salesman a, Salesman b;
WHERE a.ytdsales < b.ytdsales;
GROUP BY a.salesman
```

Tworzenie kopii jest równoznaczne z łączeniem tablicy samej ze sobą. Trzeba to robić bardzo ostrożnie przede wszystkim w przypadku łączenia tablic o dużej liczbie wierszy, jak również wtedy, gdy w wyniku chcemy uzyskać pewne kombinacje danych, jak pokazano to w podanym dalej przykładzie.

• *Przykład 8.28*

Ze zbioru Części mamy wylistować pary: numer części i jej opis, które zafakturowano temu samemu klientowi. Ponieważ nie istnieje w bazie bezpośrednie powiązanie pomiędzy zbiorami Faktury i Części, aby uzyskać potrzebne informacje należy wziąć pod uwagę zbiór Detale, co zaznaczone jest powiązaniem w warunku WHERE.

```
SELECT a1.pno, a1.descript, a2.pno, a2.descript;
FROM Parts.a1, Parts.a2, Invioces.b1, Invoices.b2, Detail.c1, Detail.c2;
WHERE b1.ino = c1.ino AND c1.pno = a1.pno;
        AND b2.ino = c2.ino AND c2.pno = a2.pno;
        AND b1.cno = b2.cno;
        AND a1.pno < a2.pno
```

Warunek umieszczony w ostatniej linijce przykładu zabezpiecza przed uzyskaniem każdej pary z tabeli głównej i jej kopii dwukrotnie.
Istnieje możliwość połączenia wyników dwóch zgodnych zapytań poprzez użycie operatora sumy UNION, który odpowiada operatorowi sumy algebry relacyjnej.

• *Przykład 8.29*

Wybierz wszystkich klientów zamieszkałych we Wrocławiu i w Poznaniu.

```
SELECT Cno, Campany;
FROM Customer;
WHERE City=”Wrocław”;
UNION;
SELECT Cno, Campany;
FROM Customer;
WHERE City=”Poznań”
```

8.5. Instrukcje zarządzania danymi

Komendy języka zarządzania danymi są wykorzystywane do zapewnienia kontroli danych i funkcji SZBD. Kontrola ta jest czynnością, która wiąże się z przyznawaniem praw dostępu do danych i do narzędzi operowania danymi. Jedną z metod kontroli danych jest określenie praw dostępu przy użyciu perspektywy. Tworzenie perspektywy, która jest tabelą wirtualną, polega na wybraniu tylko niektórych informacji z jednej lub wielu tabel.

• *Przykład 8.30*

Tworzymy perspektywę S1 z tabeli Wykładowcy.

```
CREATE VIEW S1 AS;
SELECT NazwiskoStud, Płeć, KodKursu;
FROM Wykładowcy
```

Jeśli komenda SELECT ma postać SELECT*, to oznacza to, że z tabeli o nazwie Wykładowcy wybieramy wszystkie kolumny.

Do przekazywania uprawnień dostępu służy instrukcja GRANT o postaci:

GRANT *typ uprawnienia*
ON *nazwa tabeli, perspektywy*
TO *nazwa użytkownika*

Typ uprawnienia dotyczy rodzaju operacji wykonywanych na danej tabeli. Na przykład użytkownik ma prawo wykonywać jedną z operacji: INSERT, UPDATE, DELETE, SELECT lub wszystkie, wówczas tryb uprawnienia ALL.

Przekazane uprawnienia można odwołać instrukcją REVOKE.

REVOKE *typ uprawnienia*
ON *nazwa tabeli, perspektywy*
TO *nazwa użytkownika*

- *Przykład 8.31*

Chcemy dać panu o nazwisku A. Nowak prawo oglądania i modyfikowania rekordów Pracowników na wydziale Elektronika, jednocześnie nie dając mu praw usuwania czy wstawiania rekordów.

```
CREATE VIEW Nowak AS;
SELECT*;
FROM Pracownicy;
WHERE NazwaWydziału =;
(SELECT NazwaWydziału;
FROM Pracownicy;
WHERE NazwiskoPrac = ”Nowak A”)

GRANT SELECT, UPDATE
ON Nowak
TO Anowak
```

- *Przykład 8.32*

Aby sam nie mógł zmieniać swojej pensji, modyfikujemy perspektywę.

```
CREATE VIEW Nowak AS;
SELECT*;
FROM Pracownicy;
```

```
WHERE NazwaWydziału =;
(SELECT NazwaWydziału;
FROM Pracownicy;
WHERE NazwiskoPrac = ”Nowak A”;
AND NazwiskoPrac <> ”Nowak A”)
```

Jeżeli użytkownikowi przekaże się określone uprawnienia do danego obiektu, to otrzymuje on dostęp do tego obiektu jedynie w ramach tych uprawnień. Istnieją również klasy użytkowników. Użytkownik należący do klasy CONNECT może oglądać dane innych użytkowników, może wykonywać zadania operowania danymi i tworzyć perspektywy. Uprawnienia RESOURCE umożliwiają użytkownikowi tworzenie indeksów i tabel baz danych oraz przyznawanie praw dostępu do tych tabel i indeksów innym użytkownikom. Uprawnienia administratora bazy dostają tylko niektórzy użytkownicy.

- *Przykład 8.33*

  Nowy użytkownik Nowak mający hasło **hallo** ma określone uprawnienia:

```
GRANT CONNECT, RESOURCE;
TO Nowak;
IDENTIFIED BY hallo
lub odwołane:
REVOKE CONNECT;
FROM Nowak
```

8.6. Integralność bazy danych

Główną cechą nowoczesnych systemów informacyjnych jest proces zapewnienia integralności [3]. Integralność mówi nam o tym, czy zawartość bazy danych odzwierciedla opisywaną rzeczywistość, a więc czy dany stan bazy jest dopuszczalny, czy nie. Integralność jest związana z procesem zmian zachodzących w bazie spowodowanych zarówno przez zdarzenia zewnętrzne jak i wewnętrzne. Zdarzenia, które wywołują zmianę stanu są w terminologii baz danych nazwane transakcjami. Transakcja powoduje przejście jednego stanu w drugi. Nowy stan jest wprowadzony przez stwierdzenie faktów, które stają się prawdziwe lub zaprzeczenie tych, które przestają być prawdziwe. Integralność jest realizowana przez tak zwane logiczne ograniczenia zwane więzami. Definicja więzów może być wyrażona w sposób statyczny, czyli sprawdza się, czy wykonana transakcja nie zmienia stanu bazy w stan niepoprawny lub w sposób dynamiczny, czyli mamy do czynienia z więzami przejść. Więzy przejść są ograniczeniami nałożonymi na przejścia bazy z jednego stanu w drugi. Można wyróżnić integralność: encji, referencyjną, dziedziny.

Integralność encji realizowana jest przez dodanie specyfikacji klucza głównego. Jest to reguła, która mówi, że każda tabela musi mieć klucz główny i że kolumna lub kolumny wybrane jako klucz główny powinny być jednoznaczne i nie zawierać wartości NULL. Bezpośrednią konsekwencją tej reguły jest to, że nie mogą powtarzać się wiersze.

• *Przykład 8.34*
Dodanie klucza głównego do definicji tabeli.

```
CREATE TABLE Pracownicy;
(NrPrac Number (5), NOT NULL UNIQUE;
NazwiskoPrac Varchar(15);
Status Varchar (15);
NazwaWydziału Varchar (20);
Pensja Decimal (7,2);
PRIMARY KEY (NrPrac))

CREATE TABLE Jednostki;
(NazwaJednostki Char (15) NOT NULL UNIQUE;
Poziom Smallint;
KodKursu Char (3);
NrPrac Number (5);
PRIMARY KEY (NazwaJednostki))
```

Integralność referencyjną definiujemy przez specyfikację klucza obcego. Reguła ta mówi, że każda wartość klucza obcego może znajdować się w jednym z dwóch stanów. Normalnie wartość klucza obcego odwołuje się do wartości klucza głównego tabeli w bazie danych lub ma wartość NULL (czyli żadnych powiązań). Utrzymanie integralności referencyjnej nie ogranicza się do określania wartości NULL. Obejmuje również określenie więzów propagacji. Więzy te określają, co powinno się stać z powiązaną tabelą, gdy modyfikujemy wiersz lub wiersze w tabeli docelowej. Możemy wyróżnić podejście ostrożne, ufne i wyważone. W pierwszym przypadku, ostrożnego usuwania, (RESTRICTED) zabraniamy usuwać wiersz z tabeli głównej (np. Pracownicy), dopóki nie będą usunięte wszystkie wiersze tabeli podrzędnej (np. Jednostki). W przypadku drugim, ufnym, istnieje usuwanie kaskadowe (CASCADES), czyli usuwanie wszystkich wierszy powiązanych z głównym (Pracownicy). W przypadku trzecim, wyważonym (NULLIFILES), kiedy usuwamy wiersz główny (Pracownicy), do tablicy Jednostki wstawiamy NULL.

• *Przykład 8.35*
Integralność referencyjna dla tabeli Jednostki:

```
CREATE TABLE Jednostki;
(NazwaJednostki Char (15);
Poziom Smallint;
KodKursu Char (3);
NrPrac Number (5);
PRIMARY KEY (NazwaJednostki);
FOREIGN KEY (NrPrac IDENTIFIES Pracownicy);
ON DELETE SET NULL;
ON UPDATE CASCADE)
```

Tabela Jednostki może mieć też taką postać:

```
CREATE TABLE Jednostki;
(NazwaJednostki Char (15);
Poziom Smallint;
KodKursu Char (3);
NrPrac Number (5);
PRIMARY KEY (NazwaJednostki);
FOREIGN KEY (NrPrac IDENTIFIES Pracownicy);
DELETE RESTRICTED;
ON UPDATE CASCADE)

CREATE TABLE Pracownicy;
(NrPrac Number (5);
NazwiskoPrac Varchar (15);
Status Varchar (15);
NazwaWydziału Varchar (20);
Pensja Decimal (7,2);
PRIMARY KEY (NrPrac))
```

NrPrac ma być ustawiony na NULL, jeżeli powiązany rekord Pracownicy jest usuwany. Jeśli dokonamy jakiejkolwiek zmiany w NrPrac w rekordzie Pracownicy, to zmiana ta powinna zostać odzwierciedlona w powiązanych rekordach Jednostki.

Specjalnym rodzajem procedur zapewniającym integralność referencji przez sprawdzenie relacji logicznych między tabelami są wyzwalacze. Główną zaletą wyzwalaczy jest możliwość ich automatycznego wywoływania. Wyzwalacze uruchamiane są niezależnie od tego, czy akcja została wywołana przez aplikację klienta, czy modyfikacja danych została wymuszona przez serwer bazy danych. Można wyróżnić trzy typy wyzwalaczy, każdy skojarzony z typem modyfikacji danych:

– *update trigger* – akcja wyzwalacza uruchamiana jest przed lub po modyfikacji pola tabeli,

– *insert trigger* – akcja wyzwalacza uruchamiana jest przed lub po wstawieniu nowego wiersza do tabeli,

– *delete trigger* – akcja wyzwalacza uruchamiana jest przed lub po skasowaniu wiersza w tabeli.

Specjalna składnia wyzwalacza kontroluje jakie działanie uruchamia wyzwalacz na wyznaczonej kolumnie. Ogólną postać podano niżej.

IF UPDATE *nazwa kolumny*
BEGIN
wyrażenie w SQL
END
lub
IF UPDATE *nazwa kolumny* AND UPDATE *nazwa kolumny*
BEGIN
wyrażenie w SQL
END

Wyzwalacz jest uruchamiany natychmiast po modyfikacji danych. Na ogół SQL traktuje każde wywołanie wyzwalacza jako transakcję, która może być cofnięta, gdy wystąpi błąd. Służy do tego np. komenda ROLLBACK TRIGGER, gdy chcemy cofnąć modyfikacje danych wykonaną przez wyzwalacz. Wyzwalacze mogą też zmieniać kaskadowo dane w połączonych tabelach.

• *Przykład 8.36*

Mamy dwie tabele. Tabela o nazwie Klient składa się z kolumn: *klient.nr*, *nazwa*, *adres* oraz tabela o nazwie Polisa zawiera kolumny *polisa.nr*, *ubezpieczony.nr*, *sprzedawca.nr*. W tabeli Klient kluczem głównym jest *klient.nr*, a w tabeli Polisa kluczem głównym jest *polisa.nr*. Obydwie tabele powiązane są ze sobą przez kolumny *klient.nr* = *ubezpieczony.nr*. Pole *ubezpieczony.nr* ma właściwość klucza obcego w tabeli Polisa. Wyzwalacz kasowania ustawiony na kolumnie *klient.nr* w tabeli Klient może spowodować akcję kasowania rekordów w tabeli Polisa.

```
ALTER TRIGGER DELETE Klient;
On Klient FOR DELETE;
AS
BEGIN
DELETE FROM Polisa;
WHERE Polisa.ubezpieczony.nr = Klient.klient.nr;
END
```

Wyzwalacze mogą zastępować warunki integralności. Nie dopuszczają lub cofają zmiany, które naruszają zasady integralności danych. Wyzwalacze mogą być uruchamiane, kiedy użytkownik wprowadza do tabeli klucz obcy, który nie ma odpowiednika w kluczu głównym innej tabeli. Wyzwalacze mogą zapewniać zasady integralności bardziej złożone niż zdefiniowane reguły integralności lub sprawdzanie wprowadzonych danych. Odmiennie od nich wyzwalacze mogą odnosić się zarówno do kolumn jak i obiektów bazy danych. Na ogół każdy wyzwalacz jest uruchamiany tylko raz w zapytaniu. Jeżeli akcja wyzwalacza polega na modyfikacji wielu wierszy tabeli, wyzwalacz może zbadać dane wielu wierszy i wykonać stosowną akcję. Modyfikacja wielu wierszy jest zwykle ważna w przypadku obliczania sum w kolumnach.

- *Przykład 8.37*

Policzyć przychód w każdym wierszu tabeli Księga Przychodu po wykonaniu wstawienia wiersza.

```
ALTER TRIGGER policz_przychod;
ON Księga Przychodu FOR INSERT;
AS;
BEGIN;
     UPDATE Księga Przychodu SET;
      Przychód = składka * Prowizja /100;
END
```

W opisanych przypadkach wyzwalacze rozpatrywały wyrażenia modyfikacji danych jako całość. Jeżeli jeden z wierszy nie jest akceptowany, to modyfikacja kończy się niepowodzeniem i cała transakcja jest anulowana. Aby tego uniknąć, konstruuje się wyzwalacze wraz z instrukcjami warunkowymi.

- *Przykład 8.38*

Konstrukcja wyzwalacza z instrukcjami warunkowymi.

```
ALTER TRIGGER sprawdz_zarobki;
ON Pracownik FOR UPDATE;
AS
BEGIN
   IF ((Wydział_nr <> 12) and (Zarobek > 3000));
     BEGIN;
       UPDATE Pracownik SET;
       Zarobek = 3000;
     END;
END
```

Wyzwalacze mogą zagnieżdżać w sobie inne wyzwalacze. Każdy wyzwalacz może uruchamiać inny. Liczba poziomów zagnieżdżenia zależy od systemu. Typowym zastosowaniem gniazda wyzwalaczy jest funkcja, która zapisuje kopie wierszy modyfikowanych przez inny wyzwalacz. Wyzwalacze mogą wykonywać proste analizy oraz porównania przed i po wykonaniu modyfikacji danych oraz wykonywać akcje zależne od wyniku porównania.

W przypadku integralności dziedziny podajemy odpowiedni typ danych dla kolumny lub odpowiedni zakres danych.

• *Przykład 8.39*

Używamy klauzuli CHECK do wymuszenia poprawnej modyfikacji.

```
CREATE TABLE Jednostki;
(NazwaModułu Char (15);
Poziom Smallint;
KodKursu Char (3);
NrPrac Number (5);
PRIMARY KEY (NazwaJednostki);
FOREIGN KEY (NrPrac IDENTIFIES Pracownicy);
ON DELETE SET NULL;
ON UPDATE CASCADE;
CHECK (Poziom IN 1, 2, 3))        wartość wstawiana do POZIOM była
                                        w określonym zbiorze
CREATE TABLE Pracownicy;
(NrPrac Number (5);
NazwiskoPrac Varchar (15);
Status Varchar (15);
NazwaWydziału Varchar (20);
Pensja Decimal (7,2);
PRIMARY KEY (NrPrac);
CHECK (NrPrac BETWEEN 100 AND 10999))
```

8.7. Uwagi końcowe

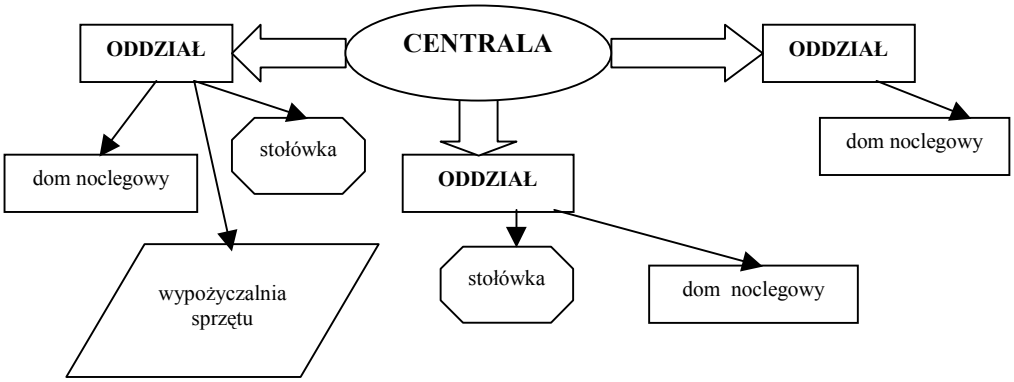
Wraz z rozwojem SZBD język SQL z języka zapytań przekształcił się w język baz danych. Prosta konstrukcja tego języka zawarta w tzw. bloku kwalifikacyjnym składającym się z instrukcji SELECT….FROM….WHERE w sposób niezwykle przejrzysty umożliwiła konstruowanie programów w SQL. Obecne prace nad tworzeniem standardu SQL skupiają się na dwóch zagadnieniach: wzbogaceniu konstrukcji rela-

cyjnych i wprowadzeniu obiektowości. Jeżeli chodzi o pierwsze zagadnienie, to wiele relacyjnych SZBD realizuje już aktywne reguły i procedury wyzwalania, czyli wyzwalacze (trigger). Element czasowy zawarty w procedurze CREATE TRIGGER określa moment aktywacji wyzwalacza. Wydaje się, że głównym celem zmodyfikowanej wersji SQL będzie wprowadzenie obiektowości. Oczekuje się nowych typów danych określanych przez użytkownika oraz uwzględnienie takich cech, jak hermetyzacja, tożsamość czy dziedziczenie.

9. SYSTEM OBIEKTOWO ZORIENTOWANY

9.1. Uwagi ogólne

Projektowanie systemów wspomagających zarządzanie przedsiębiorstwem jest od lat tematem rozważań analityków, projektantów i informatyków. Celem tego rozdziału jest omówienie podejścia do projektowania systemów informatycznych, opartych na modelu obiektowym, wspomagających zarządzanie przedsiębiorstwem na przykładzie biura turystycznego, które prowadzi różnego rodzaju usługi [18]. W tym celu należało zapoznać się ze specyfiką pracy biura turystycznego, z potrzebami i oczekiwaniami względem przyszłego systemu informatycznego, sposobami świadczenia usług. Następnym krokiem jest stworzenie dokumentacji będącej podstawą tworzenia późniejszych struktur informatycznych. Schemat organizacyjny biura przedstawiono na rys. 9.1.

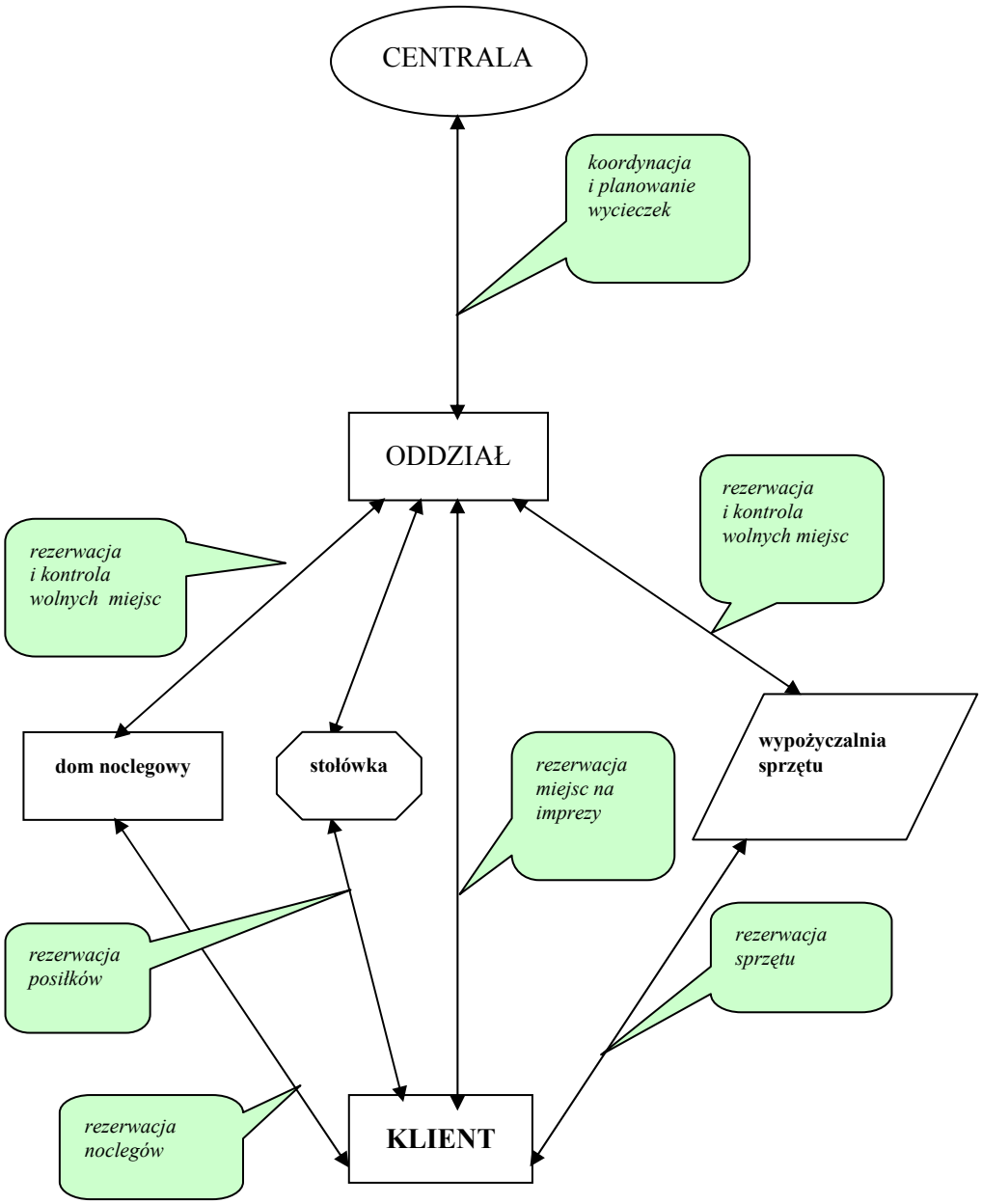


Rys. 9.1. Schemat organizacyjny biura turystycznego

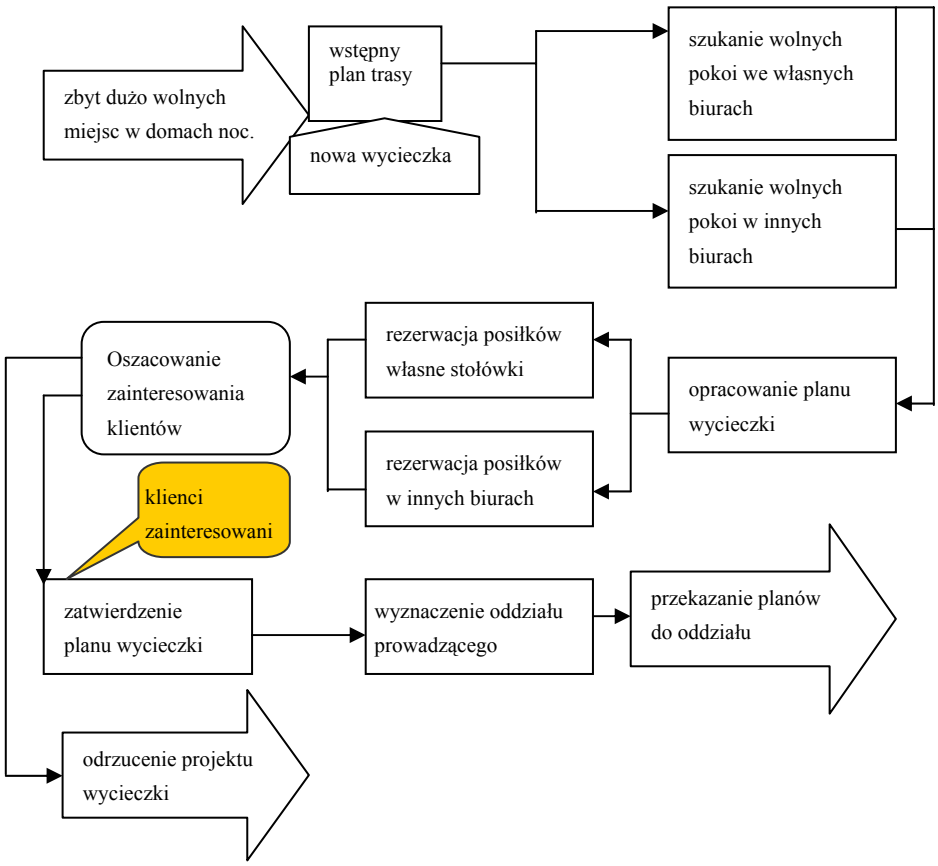
Diagram kontekstowy organizacji, który ukazuje jak wymieniane są w biurze zadania i informacje przedstawiono na rys. 9.2.

Przykładowy proces zachodzący w biurze pokazano na rys. 9.3.

Schemat ten pozwala się zorientować, jakie operacje należy wykonać, aby dany proces zakończył się jednym z możliwych zdarzeń. Każda operacja może być albo kolejnym procesem lub zestawem wywołań metod obiektów i operacji na ich atrybutach, albo działaniem człowieka. Procesy przedstawione na rysunku są czytelne dla człowieka, jednak, aby mógł posługiwać się nimi komputer muszą zostać zapisane w postaci pseudojęzyka.



Rys. 9.2. Diagram kontekstowy organizacji



Rys. 9.3. Przykładowy proces zachodzący w biurze

9.2. Język opisu procesów

Definicja procesu rozpoczyna się słowem kluczowym **Process**, po którym następuje nazwa procesu. Wykonanie procesu rozpoczyna się kiedy zajdą zdarzenia umieszczone po słowie kluczowym **Start on.** Nazwy poszczególnych zdarzeń są umieszczone w apostrofach ^ i oddzielone od siebie przecinkami. Każdy proces składa się z wielu wywołań podprocesów. Wywołanie podprocesu ma postać: słowo kluczowe **Run,** po którym następuje nazwa procesu. Możliwe jest również wywołanie kilku procesów równolegle, z tym że rozpoczęcie kolejnego procesu może mieć miejsce po zakończeniu wszystkich procesów równoległych. Takie wywołanie ma postać: słowo kluczowe **Run,** po którym następuje nawias { }. W nawiasie należy umieścić nazwy poszczególnych procesów rozdzielone przecinkami. Można także wywoływać procesy

zwracające wartość. Takie wywołanie ma postać: słowo kluczowe **Run test**, po którym następuje nazwa procesu. Po tej komendzie powstaje seria poleceń postaci: słowo kluczowe **On,** po którym następuje nazwa wartości zwróconej przez poprzedni proces, a następnie komenda, która ma być wykonana dla tej wartości. Można również wprowadzić słowo kluczowe **Else**. Polecenia znajdujące się po tym słowie wykonywane są dla wartości, które nie zostały wcześniej wymienione po słowach kluczowych **On.** Komendę **Run test** kończy słowo kluczowe **End test**.

Każde polecenie można zastąpić sekwencją instrukcji za pomocą słów kluczowych **Begin... End**, między którymi może wystąpić dowolna ilość komend. Aby w procesie istniała możliwość cofania się do wykonanych wcześniej poleceń, wprowadzono do zaznaczenia jakiegoś miejsca w kodzie słowo kluczowe **Label,** po którym następuje identyfikator zapisany dużymi literami. Powrót do tego miejsca następuje po napotkaniu słowa kluczowego **Back to** z danym identyfikatorem. Słowo kluczowe **Throw** służy do wywoływania zdarzeń. Wykonanie kodu procesu kończy się po napotkaniu słowa kluczowego **!End**. Dopuszczalne jest użycie kilku instrukcji kończących w jednym procesie. Niżej podano opis w postaci pseudojęzyka przykładowych procesów.

• *Przykład 9.1*

Opis w pseudojęzyku procesu planowania wycieczki dla rysunku 9.3.

```
Process PLANOWANIE WYCIECZKI OBJAZDOWEJ
Start on  ^Zbyt dużo miejsc w budynkach^  or  ^Wymyślono nową wycieczkę^
Begin
      Run ^Wstępne zaplanowanie trasy^
      Run
      {
       ^Szukanie wolnych pokoi we własnych budynkach^
          ^Szukanie wolnych pokoi w innych biurach^
     }
      Run ^Opracowanie planu wycieczki^
      Run
      {
         ^Rezerwacja posiłków we własnych stołówkach^
         ^Rezerwacja posiłków w innych biurach^
      }
         Run test ^Oszacowanie zainteresowania klientów^
          On ^Klienci zainteresowani^
            Begin
                 Run ^Zatwierdzenie planu wycieczki^
                 Run ^Wyznaczenie Oddziału Prowadzącego^
                 Throw ^Przekazanie planu do oddziałów^
            !End
          EndOn
          Else
```

```
            Begin
                Throw ^Odrzucenie planu wycieczki^
            !End
            End test
End
End
```

- *Przykład 9.2*

Opis w pseudojęzyku procesu realizacji wycieczki.

```
Process  REALIZACJA WYCIECZKI OBJAZDOWEJ W ODDZIALE
Start on ^Wpłynął projekt nowej wycieczki^
Begin
      Run ^Wprowadzenie wycieczki do oferty^
      Run ^Wyznaczenie odpowiedzialnego pracownika^
      Label REALIZACJA
      Run ^Rezerwacja środków transportu^
      Run ^Wyznaczenie pilotów^
      Run ^Opracowanie i realizacja strategii reklamowej^
         Run test ^Oczekiwanie na zgłoszenia klientów^
          On ^Zbyt mało klientów^
           Run test ^Analiza przyczyn braku zainteresowania^
           On ^Przyczyna braku klientów możliwa do poprawienia^
              Begin
                 Run ^Korekta oferty^
                 Back to REALIZACJA
               End
           On ^Przyczyna braku klientów niemożliwa do poprawienia^
              Begin
                 Run ^Przedstawienie zgłoszonym klientom innych propozycji^
                  Run ^Anulowanie wycieczki^
                 Throw ^Zgłoszenie faktu anulowania wycieczki do centrali^
              !End
           End On
           End On
         On ^Wystarczająca ilość klientów^
           Begin
            Run test ^Sprawdzenie czy pozostały wolne miejsca^
             On ^Tak^
              Run ^Oferta last minute^
             End On
             Run ^Zatwierdzenie wyjazdu^
             Throw ^Wyjazd wycieczki^
            End test
           !End
         End On
         End test
         End On
         End test
End
End
```

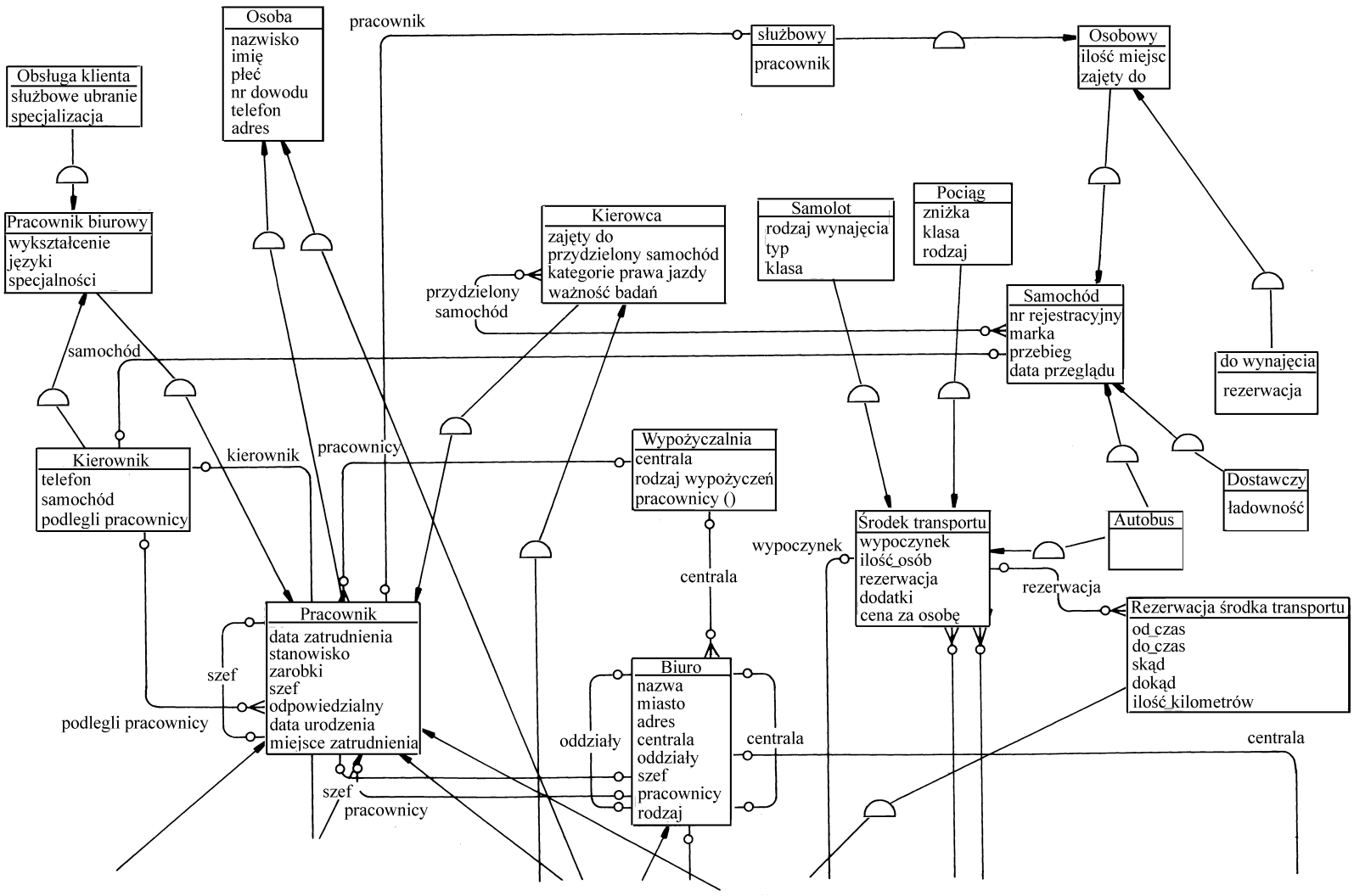
• *Przykład 9.3*

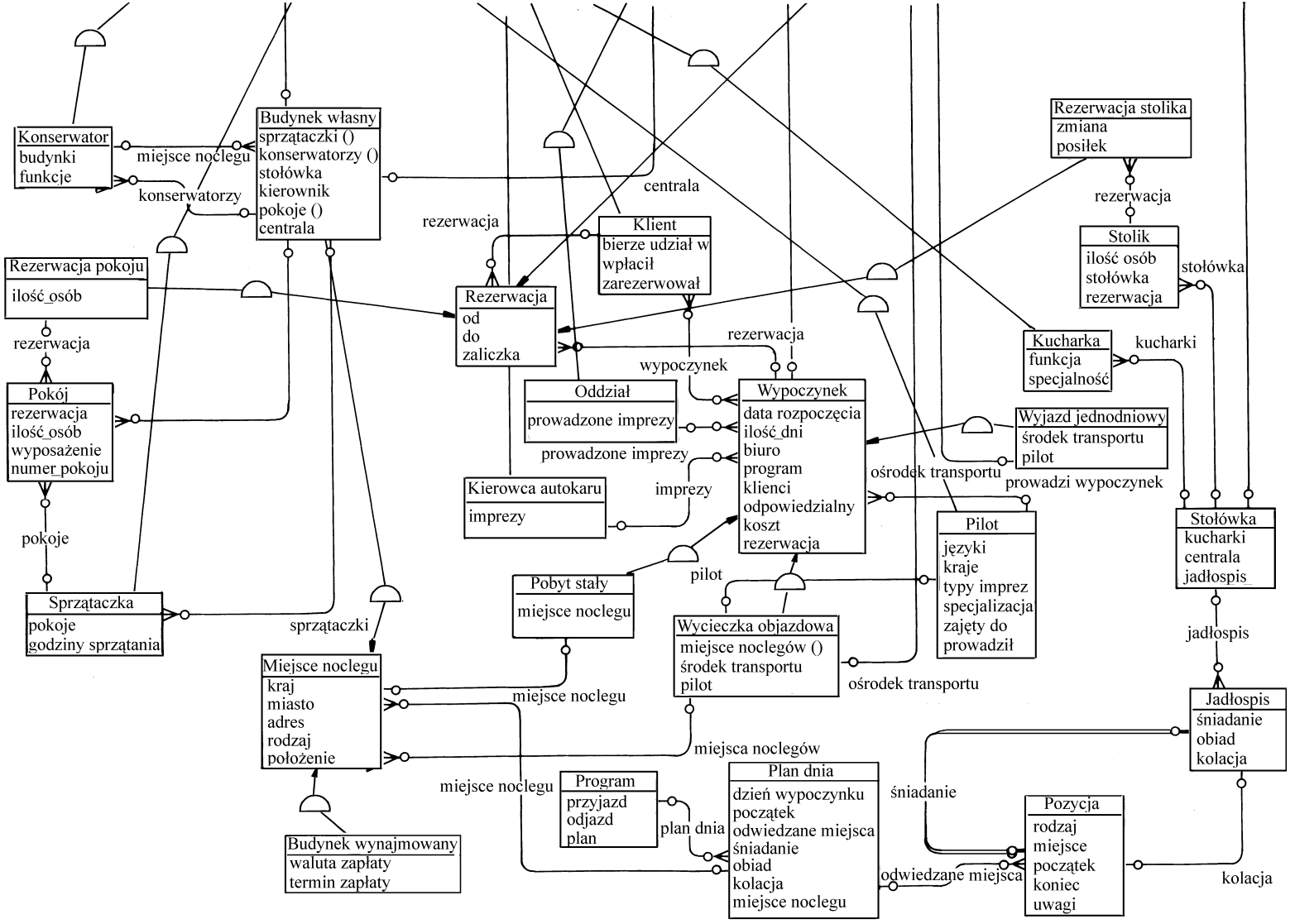
Opis w pseudojęzyku rezerwacji miejsc w domu noclegowym własnym.

```
Process REZERWACJA MIEJSC DOM NOCLEGOWY WŁASNY
Start on ^Rezerwacja z oddziału^, ^Bezpośrednia rezerwacja^
Begin
      Run Test ^Sprawdzenie czy są wolne miejsca^
       On ^Znaleziono^
        Begin
         Label REALIZACJA
         Run ^Rezerwacja miejsca^
         Throw ^Informacja o dokonaniu rezerwacji^
        !End
       End On
       On ^Brak^
        Begin
         Label ODMOWA
         Run ^Odmowa rezerwacji^
         Throw ^Informacja o braku miejsc^
        !End
       End On
       On ^Znaleziono^
        Run test ^Konsultacja z klientem aktualnej rezerwacji^
         On ^Rezerwacja aktualna^
         Back to ODMOWA
         On ^Rezerwacja nieaktualna^
          Begin
           Run ^Anulowanie rezerwacji aktualnego klienta^
           Back to  REZERWACJA
          End
         End On
         End On
        End test
       End On
      End test
End
End
```

9.3. Schemat bazy danych

Posługując się posiadanymi informacjami o przedsiębiorstwie można stworzyć schemat bazy danych. Na schemacie pokazanym na rys. 9.4, uwzględniono wszystkie obiekty wraz z nazwami ich atrybutów, powiązaniami między obiektami, oraz uwzględniono dziedziczenie. Jako model dziedziczenia przyjęto dziedziczenie wielobazowe, ponieważ oferuje ono większą elastyczność przy tworzeniu obiektów. Relacje dziedziczenia są zaznaczone na rysunku liniami zakończonymi strzałkami, gdzie grot strzałki wskazuje obiekt rodzica w tym związku.


Osoba
nazwisko
imię
płeć
nr dowodu
telefon
adres
pracownik
służbowy
pracownik
Osobowy
ilość miejsc
zajęty do
Obsługa klienta
służbowe ubranie
specjalizacja
Pracownik biurowy
wykształcenie
języki
specjalności
Kierowca
zajęty do
przydzielony samochód
kategorie prawa jazdy
ważność badań
przydzielony samochód
Samolot
rodzaj wynajęcia
typ
klasa
Pociąg
zniżka
klasa
rodzaj
Samochód
nr rejestracyjny
marka
przebieg
data przeglądu
samochód
do wynajęcia
rezerwacja
Kierownik
telefon
samochód
podlegli pracownicy
kierownik
pracownicy
Wypożyczalnia
centrala
rodzaj wypożyczeń
pracownicy ()
Dostawczy
ładowność
Środek transportu
wypoczynek
ilość osób
rezerwacja
dodatki
cena za osobę
wypoczynek
Autobus
centrala
rezerwacja
Rezerwacja środka transportu
od czas
do czas
skąd
dokąd
ilość kilometrów
Pracownik
data zatrudnienia
stanowisko
zarobki
szef
odpowiedzialny
data urodzenia
miejsce zatrudnienia
szef
podlegli pracownicy
Biuro
nazwa
miasto
adres
centrala
oddziały
szef
pracownicy
rodzaj
oddziały
centrala
centrala
szef
pracownicy


Konserwator
budynki
funkcje
miejsce noclegu
konserwatorzy
Budynek własny
sprzątaczki ()
konserwatorzy ()
stołówka
kierownik
pokoje ()
centrala
centrala
rezerwacja
Klient
bierze udział w
wpłacił
zarezerwował
Rezerwacja stolika
zmiana
posiłek
rezerwacja
Stolik
ilość osób
stołówka
rezerwacja
stołówka
Rezerwacja pokoju
ilość_osób
rezerwacja
Rezerwacja
od
do
zaliczka
rezerwacja
wypoczynek
Kucharka
funkcja
specjalność
kucharki
Pokój
rezerwacja
ilość_osób
wyposażenie
numer_pokoju
pokoje
Oddział
prowadzone imprezy
prowadzone imprezy
Wypoczynek
data rozpoczęcia
ilość_dni
biuro
program
klienci
odpowiedzialny
koszt
rezerwacja
Wyjazd jednodniowy
środek transportu
pilot
ośrodek transportu
prowadzi wypoczynek
Kierowca autokaru
imprezy
imprezy
Pilot
języki
kraje
typy imprez
specjalizacja
zajęty do
prowadził
Stołówka
kucharki
centrala
jadłospis
Pobyt stały
miejsce noclegu
pilot
Wycieczka objazdowa
miejsce noclegów ()
środek transportu
pilot
ośrodek transportu
jadłospis
Sprzątaczka
pokoje
godziny sprzątania
sprzątaczki
Miejsce noclegu
kraj
miasto
adres
rodzaj
położenie
miejsce noclegu
miejsca noclegów
Jadłospis
śniadanie
obiad
kolacja
miejsce noclegu
Program
przyjazd
odjazd
plan
plan dnia
Plan dnia
dzień wypoczynku
początek
odwiedzane miejsca
śniadanie
obiad
kolacja
miejsce noclegu
śniadanie
Pozycja
rodzaj
miejsce
początek
koniec
uwagi
odwiedzane miejsca
kolacja
Budynek wynajmowany
waluta zapłaty
termin zapłaty

Rys. 9.4. Schemat bazy danych biura turystycznego uwzględniający dziedziczenie

9.4. Język opisu klas

Definicja klasy składa się z nazwy klasy, nazw klas, po których ta klasa dziedziczy, oraz atrybutów i metod danej klasy. Nazwa klasy, pisana dużą literą poprzedzona jest słowem kluczowym **class**. W nowej linii po słowie **inherit** znajduje się lista klas, po której definiowana klasa ma dziedziczyć. Poszczególne klasy muszą być oddzielone przecinkami. Atrybuty klasy pojawiają się po słowie kluczowym **properties**. Lista atrybutów zakończona jest słowem **end**. Każdy z atrybutów powinien się znaleźć w nowej linii. Linia jest zbudowana z nazwy atrybutu, znaku „ : ” oraz typu atrybutu. Można używać następujących typów atrybutów: integer, real, string, date, time, datetime, inne klasy. Jeśli atrybutem jest nazwa innej klasy, to wartością tego atrybutu może być zarówno obiekt tej klasy, jak i dowolnej klasy dziedziczącej po tej klasie. W opisie przyjęto następującą konwencję: jeśli po nazwie atrybutu pojawi się znak (*), to oznacza to, że mamy do czynienia z listą atrybutów danego typu. Więzy integralności podajemy po słowie **constraints.** Mają one postać warunków jakie muszą spełniać wybrane pola. Jeśli wartość warunku wynosi **false**, to wartość atrybutu nie może zostać zaakceptowana. Lista procedur wywołanych zdarzeniami zachodzącymi w bazie umieszczamy za słowem kluczowym **triggers.** Każdy wiersz w opisie klasy zakończony jest znakiem „ ; ”.

Szablon definicji klasy ma następująca postać:

```
class NAZWA KLASY
inherit NAZWA KLASY BAZOWEJ
properties
  nazwa atrybutu: typ atrybutu;
  nazwa atrybutu (*): typ atrybutu;
constrains
  nazwa atrybutu > 0;
  nazwa atrybutu1 > nazwa atrybutu2;
triggers
  if nazwa atrybutu not NULL then delete disabled;
end;
```

Jako przykładową definicję klasy podano klasę bazową ŚRODEK TRANSPORTU. Ponieważ klasa ta nie dziedziczy po innych klasach, więc po słowie kluczowym **inherit** nie występuje nic. ŚRODEK TRANSPORTU posiada atrybuty: *wypoczynek* wskazujący na klasę WYPOCZYNEK, *ilość osób* – atrybut prosty typu integer, *rezerwacja* wskazujący na klasę REZERWACJA, *dodatki* – lista atrybutów prostych typu string oraz *cenę za osobę* – atrybut prosty typu real. Zdefiniowane więzy integralności sprawiają, że atrybuty *cena za osobę* i *ilość osób* muszą mieć wartość większą od zera. Zdefiniowana procedura typu **trigger** sprawia, że danego środka transportu nie da się usunąć, jeśli odwołuje się on do obiektu typu REZERWACJA lub WYPOCZYNEK.

• *Przykład 9.4*
Opis klasy ŚRODEK TRANSPORTU:

```
class ŚRODEK TRANSPORTU
inherit
properties
  wypoczynek: WYPOCZYNEK;
  ilość osób: integer;
  rezerwacja: REZERWACJA ŚRODKA TRANSPORTU;
  dodatki (*): string;
  cena za osobę: real;
constrains
  cena za osobę > 0;
  ilość osób > 0;
triggers
  if wypoczynek, rezerwacja not NULL then delete disabled;
methods
  New();
  Dispose();
  OdczytAtrybutów(): (*)^;
  ZapisAtrybutów (wypoczynek, ilość osób, rezerwacja, dodatki, cena za osobę);
  Ilość dni (wypoczynek): integer;
end;
```

Jako przykład klasy dziedziczącej po innej klasie przedstawiono klasę SAMOLOT. Klasa ta dziedziczy wszystkie atrybuty po klasie bazowej ŚRODEK TRANSPORTU, ponadto posiada własne atrybuty charakteryzujące wyłącznie ją: *rodzaj wynajęcia*, *typ*, *klasa*.

• *Przykład 9.5*
Opis klasy SAMOLOT:

```
class SAMOLOT
inherit ŚRODEK TRANSPORTU
properties
  rodzaj wynajęcia: string;
  typ: string;
  klasa: string;
methods
  New();
  Dispose();
  OdczytAtrybutów() : (*)^;
  ZapisAtrybutów (rodzaj wynajęcia, typ, klasa);
end;
```

• *Przykład 9.6*
Opis klasy PRACOWNIK:

```
class PRACOWNIK
inherit OSOBA
properties
  data_ zatrudnienia: date;
  stanowisko: string;
  zarobki: real;
  szef:  PRACOWNIK;
  odpowiedzialny (*):  WYPOCZYNEK, BIURO, SAMOCHÓD;
  data _urodzenia:  date;
  miejsce_ zatrudnienia:  string;
constrains
   miejsce_ zatrudnienia not NULL;
   data_ zatrudnienia  not NUL;
   adres not NULL;
triggers
   if  odpowiedzialny not NULL then delete disabled;
methods
   New();
   Dispose();
   OdczytAtrybutów(): (*)^;
   ZapisAtrybutów (data_zatrudnienia, stanowisko, zarobki, szef, odpowiedzialny, data_urodzenia, miej-
    sce_zatrudnienia);
   Nazwisko_szefa: string;
   Imię_szefa: string;
   ZaCoOdpowiedzialny: string;
end;
```

- *Przykład 9.7*

  Opis klasy OSOBA:

```
class  OSOBA
inherit
properties
   nazwisko:  string;
   imię:  string;
   płeć:  boolean;
   nr_dowodu:  string
   adres:  string
   telefon:  string;
constrains
nazwisko  not NULL;
methods
   New();
   Dispose();
   OdczytAtrybutów(): (*)^;
   ZapisAtrybutów (nazwisko, imię, płeć, nr_dowodu, adres, telefon);
end;
```

W podobny sposób można opisać wszystkie klasy występujące w systemie.

9.5. Język opisu metod

Każda klasa zawiera metody operujące na wartościach atrybutów tej klasy. Wszystkie metody zawarte są w słowie kluczowym **methods**. Nazwa metody rozpoczyna się od dużej litery i charakteryzuje jej działanie. W każdej klasie zdefiniowano dwie podstawowe metody służące do odczytu i zapisu atrybutów danej klasy. Metoda *OdczytAtrybutów();* wykorzystywana jest do pobierania wszystkich, lub tylko niektórych, wartości atrybutów obiektu. Brak parametrów przy wywoływaniu tej metody powoduje zwrot wartości wszystkich atrybutów (także NULL) danego obiektu. W tym przypadku zwracany jest wskaźnik do listy wskaźników dla określonych wartości atrybutów. Zapisywane jest to w następujący sposób: *OdczytAtrybutów (): (*)^;* Jeżeli interesują nas tylko określone atrybuty, to należy jako parametry podać ich nazwy, a jako wynik działania metody otrzymamy wskaźnik do listy wskaźników zawierającej tylko pożądane atrybuty (*OdczytAtrybutów (zarobki, szef): (*)^ ;*). Aby zapisać wartości atrybutów do obiektów, należy użyć metody *ZapisAtrybutów( );*. W obiekcie zapisywane są tylko te wartości atrybutów, rozdzielone przecinkami, które zostały podane w linii parametrów. Możliwe jest zatem następujące wywołanie metody zapisującej atrybuty: *ZapisAtrybutów (data_zatrudnienia, stanowisko, zarobki, ..., data_urodzenia);*. Wartości zwracane przez te reguły to na ogół wartości proste typu: integer, real, string, date, ale także wskaźniki do list zawierających wartości proste. Zapisuje się to w następujący sposób: *ProgramWypoczynku: (*) string;*. Również obiekt może być wartością zwracaną przez metodę *ProgramWypoczynku: (wypoczynek): PROGRAM;*.W każdej klasie zdefiniowane są dwie metody służące do tworzenia i kasowania obiektów. Do tworzenia nowego obiektu wykorzystuje się metodę *New();*, natomiast do zwalniania zasobów zajętych przez dany obiekt metodę *Dispose();*.

9.6. Język opisu praw dostępu

Definicja praw dostępu danej klasy do pozostałych klas rozpoczyna się słowem kluczowym **rights,** a kończy na słowie kluczowym **end.** Po słowie **rights** podawana jest nazwa klasy, dla której zdefiniowane zostały prawa dostępu, następnie wypisywane są wszystkie obiekty ze wszystkimi ich atrybutami, a obok podawane są prawa dostępu. Przykładowo wybrano trzy prawa dostępu: R (read) – klasa ma prawo tylko do odczytu danego obiektu lub atrybutu, M (modify) – klasa ma prawo modyfikacji oraz odczytu danego obiektu lub atrybutu oraz D (delete) – klasa ma prawo kasowania, modyfikacji i odczytu danego obiektu lub atrybutu. Przyjęcie takiej kaskadowej koncepcji praw ułatwia w znaczny sposób definiowanie i zapis tego w systemie bazy danych. Prawa dostępu podawane są po dwukropku w nawiasach, a wiersz definicji zakończony jest średnikiem. Jeśli w nawiasie nie pojawi się żadna litera, to dana klasa

nie ma żadnych praw dostępu do obiektu lub atrybutu. Niżej podano przykład określenia praw dostępu dla klasy: OBSŁUGA KLIENTA.

• *Przykład 9.8*
Nadawanie praw dostępu:

```
rights          OBSŁUGA KLIENTA

KIEROWNIK: (R);
      Telefon: (R);
      Samochód: (R);
      Podlegli pracownicy: (R);
PRACOWNIK BIUROWY: (R );
      wykształcenie: (R);
      języki: (R);
      specjalności: (R);
KLIENT: (D);
      bierze udział w: (M);
      wpłacił: (M);
      zarezerwował: (M);
OSOBA: (D);
      nazwisko: (M);
      imię: (M);
      płeć: (M);
      nr dowodu: (M);
      telefon: (M);
      adres: (M);
PRACOWNIK …
.
.
end
```

9.7. Zapytania w bazie

Podstawowym elementem każdego systemu baz jest proste i efektywne formułowanie zapytań. Język zapytań powinien charakteryzować się prostotą zapisu złożonych operacji, być efektywny, to znaczy zapytania powinny być optymalizowane pod względem szybkości wyszukiwania informacji, a ponadto powinien być niezależny od aplikacji. Spełnienie tych wymagań w przypadku baz obiektowych jest z wielu powodów trudne. Duża złożoność struktur danych, brak jednolitego modelu oraz sprzeczność z zasadą enkapsulacji (wartości danych nie powinny być bezpośrednio dostępne dla użytkownika) to podstawowe przeszkody. W przypadku baz obiektowych można wyróżnić dwa podstawowe sposoby dostępu do obiektów. Pierwszy z nich to wykorzystanie języka zapytań podobnego w swej konstrukcji do języka relacyjnych baz danych, np. SQL, drugi sposób bazuje na bezpośrednim dostępie do obiektów poprzez

ich identyfikatory. Wykorzystanie identyfikatorów pozwala na poruszanie się po bazie, a metody obiektów na uzyskanie dostępu do danych w obiekcie. Oba sposoby mogą być używane zamiennie. Można, przykładowo, wykorzystując język zapytań otrzymać zbiór wyników spełniających dane warunki, a następnie korzystając z technik nawigacyjnych przeglądać rezultat zapytania. Rezultatem zapytania może być obiekt lub zbiór obiektów klasy, która musi być zdefiniowana w zapytaniu. Zabezpieczeniem w tym przypadku może być założenie, że wszystkie otrzymane atrybuty muszą pochodzić z każdego z wyszukanych obiektów, czyli że może to być albo sam obiekt, albo pojedynczy atrybut. Sposób ten gwarantuje, że wynikiem wyszukiwania będzie zbiór obiektów należących do istniejących już klas. W zapytaniach do baz danych można również wykorzystać metody. Wykorzystanie metod może w znaczący sposób wpłynąć na ułatwienie wyszukiwania informacji w bazie. Rezultat metody może służyć do sformułowania warunków zapytania lub może być zwykłym atrybutem spełniającym określone warunki. Jako przykład zadajemy pytanie: *Znajdź wszystkich kierowców autokaru, dla których impreza (wypoczynek) będzie trwała trzy dni*. W tym przypadku wykorzystamy metodę IlośćDni (impreza) obiektu KIEROWCA AUTOKARU, która jako rezultat swojego działania zwraca wartość równą długości trwania imprezy w dniach.

W dziedzinie prac badawczych nad językiem zapytań dla obiektowych baz kładzie się duży nacisk na zgodność ze standardem języka SQL. Obiektowe języki wzorowane na SQL dysponują podstawowymi możliwościami tego języka takimi, jak: mechanizmy sortowania danych, operacje grupowania danych, funkcje arytmetyczne itp. Formułowanie zapytania w języku obiektowym np. OQL (ang. *Object Query Language*), jest bardzo podobne do formułowanych w SQL. OQL jest oparty na rachunku dziedzin. Zapytanie jest wyrażeniem, zwykle złożonym, o określonym typie. Język udostępnia szereg operatorów o różnej liczbie argumentów, które mogą być także wyrażeniami. Konstrukcja **SELECT-FROM-WHERE** jest pewnym operatorem. Dlatego OQL umożliwia zagnieżdżanie nie tylko na poziomie frazy **WHERE**, ale również przy frazach **SELECT** i **FROM**. **SELECT** określa zbiór atrybutów, który ma być rezultatem zapytania, **FROM** określa zakres zapytania, a **WHERE** określa warunki, jakie muszą spełniać wyszukiwane obiekty. Atrybuty mogą być definiowane w sposób zagnieżdżony. W tym wypadku konieczne jest definiowanie ścieżek dostępu.

- *Przykład 9.9*

Znajdź wszystkie wycieczki objazdowe, które trwały cztery dni, których pilot jest specjalistą od architektury i których klienci są mężczyznami.

SELECT x FROM x IN WYPOCZYNEK **WHERE** x.ilość_dni=4 **AND** x.pilot.specjalizacja = ”Architektura” **AND** x.klient.płeć = TRUE;

- *Przykład 9.10*

Znajdź wszystkich pracowników oraz ich przełożonych. Przełożeni muszą zarabiać mniej niż 1000 i musi być spełniony dodatkowy warunek, że urodzili się po 1975 roku.

SELECT DISTINCT STRUCT (n:x.nazwisko, p: **(SELECT** y **FROM** y **IN** x.szef **WHERE** y.zarobki < 1000) **AND** y.data_urodzenia > ”1975-01-01”) **FROM** x **IN** PRACOWNIK;

- *Przykład 9.11*

Znajdź nazwiska i stanowiska tych pracowników, którzy mają na imię Tomasz lub Marcin oraz zostali zatrudnieni po 1 kwietnia 1999 roku.

SELECT STRUCT (n:x.nazwisko, s:x.stanowisko) **FROM** x **IN (SELECT** y **FROM** y **IN** PRACOWNIK **WHERE** y.data_zatrudnienienia > ”1999-04-01”) **WHERE** (x.imię=”Tomasz” **OR** x.imię=”Marcin”);

- *Przykład 9.12*

Znajdź wszystkich pracowników biurowych, którzy są jednocześnie szefami.

SELECT PRACOWNIK_BIUROWY **WHERE EACH** szef = ”TRUE”; lub
FOR ALL x **IN** PRACOWNIK BIUROWY: x.szef = ”TRUE”;

- *Przykład 9.13*

Znajdź pracowników biurowych, z których co najmniej jeden jest szefem.

SELECT PRACOWNIK_BIUROWY **WHERE EXSISTS** szef = ”TRUE”;

Jeśli mamy do czynienia z atrybutami wielowartościowymi to w zapytaniu należy użyć konstruktora SET.

- *Przykład 9.14*

Znajdź pracowników biurowych, którzy rozpoczynają urlopy: 2.07.99, 1.02.00, 4.02.00.

SELECT PRACOWNIK_BIUROWY **FROM x IN** WYPOCZYNEK **WHERE** data_rozpoczęcia **IN SET** (”1999-07-02”, ”2000-02-01”, ”2000-02-04”)

Przy wyszukiwaniu danych należy wziąć pod uwagę, czy zapytanie dotyczy tylko jednej wyspecjalizowanej klasy. Najprostszym sposobem jest wykonanie operacji sumowania zbiorów.

- *Przykład 9.15*

Znajdź wszystkich pracowników zarabiających więcej niż 1000.

(SELECT x **FROM** x **IN** PRACOWNIK **WHERE** zarobki > 1000);
UNION
(SELECT x **FROM** x **IN** KIEROWCA **WHERE** zarobki > 1000);
UNION
(SELECT x **FROM** x **IN** KIEROWCA_AUTOBUSU **WHERE** zarobki > 1000);
UNION
……..
……..

dla wszystkich obiektów dziedziczących po obiekcie PRACOWNIK.

Znacznym uproszczeniem powyższego zapisu jest wstawienie obok nazwy klasy operatora *, który oznacza, że w zapytaniu będzie uwzględniona cała hierarchia tej klasy.

- *Przykład 9.16*

Znajdź wszystkich pracowników zarabiających więcej niż 1000.

SELECT PRACOWNIK * **WHERE** zarobki > 1000;

Częstym przypadkiem w definicji klas jest fakt, że nowo utworzona klasa jest definiowana przez odwołanie się do siebie samej. W języku zapytań powinna powstać możliwość zadawania pytań dotyczących zależności cyklicznych. Rozważymy atrybut *szef* klasy PRACOWNIK. Dziedziną tego atrybutu jest ta sama klasa.

- *Przykład 9.17*

Znajdź wszystkich pracowników, którzy są podwładnymi pracownika o nazwisku Nowak.

SELECT x **FROM** x **IN** PRACOWNIK, y **IN** PRACOWNIK **WHERE** y = x.szef **AND EXISTS** (**SELECT** y **FROM** y **IN** y.szef **WHERE** y.nazwisko = ”Nowak”);

Inaczej ten przykład można zapisać w prostszy sposób, używając klauzuli **RECURSES**.

SELECT PRACOWNIK (**RECURSES** szef) **WHERE** nazwisko = ”Nowak”

RESURSES szef oznacza, że jak tylko zostanie znaleziony obiekt klasy PRACOWNIK z atrybutem nazwisko = ”Nowak”, musi być rekursywnie zwrócona wartość atrybutu szef dla tego obiektu.

Zapytania z wielokrotnymi obiektami pozwalają na rozstrzygnięcie, czy przedmiotem zapytania ma być cała hierarchia klas, czy tylko pojedyncza klasa oraz rozwiązują problem porównywania atrybutów wielowartościowych.

- *Przykład 9.18*

Znajdź wszystkich pracowników biurowych, którzy mają to samo nazwisko co szef pilotów.

SELECT x **FROM** x **IN** PRACOWNIK_BIUROWY, p **IN** PILOT **WHERE** x.nazwisko = p.szef. nazwisko

Obiektowy język OQL jest zasadniczo językiem wyszukiwania danych, nie posiada SQL-owych komend **INSERT** i **UPDATE**, choć umożliwia wykonywanie metod obiektów, które mogą wykonywać operacje na danych. W OQL brak jest wartości **NULL**, pojęcia tablic oraz perspektyw. Jest jednak możliwość definiowania więzów integralności, wyzwalaczy i nadawania przywilejów.

9.8. Uwagi końcowe

Trudności z jakimi spotykają się pracownicy różnego typu przedsiębiorstw czy biur to zapanowanie nad ogromnym stosem dokumentów oraz konieczność trzymania się ściśle wytyczonego schematu, który jest niefunkcjonalny, ponieważ na przykład rezygnacja jednego klienta w dowolnym momencie realizacji usługi sprawia ogromny zamęt w pozostałych rezerwacjach. Sprawą niezmiernie ważną jest również scalenie i ujednolicenie formatu dokumentów istniejących w przedsiębiorstwie. Zwykle używa się kilku niezależnie pracujących programów. Jedne dotyczą systemu rezerwacji, inne usprawniają księgowość. Należy szukać więc rozwiązania jednolitego, które synchronicznie wspomagałoby pracę na wielu płaszczyznach. Od takiego produktu oczekuje się pogodzenia działalności przedsiębiorstwa jako „operatora" wycieczek, czyli firmy sprzedającej swoją ofertę rozbudowanego systemu rezerwacji i firmy dbającej o własne sprawne biuro obsługi klienta. Możliwości wykorzystania komputerów zarówno w sieciach lokalnych przedsiębiorstwa, jak i dostęp do ogólnoświatowej sieci Internet stworzyło wielkie możliwości rozwoju tego typu firm.

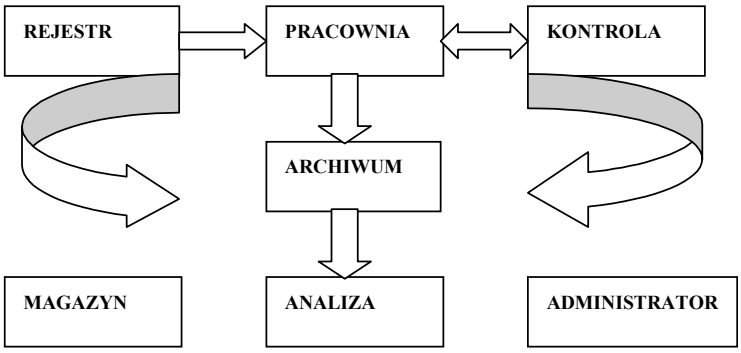
10. SYSTEM DEDUKCYJNY

10.1. Uwagi ogólne

W celu zilustrowania teorii dedukcyjnych baz danych przedstawiono system bazy danych obsługujący laboratorium medyczne [18]. Jak powszechnie wiadomo, medycyna jest dziedziną bardzo szybko rozwijającą się, a wiedza medyczna ma bardzo niski stopień formalizacji. Wykorzystanie sformatowanego modelu danych takiego jakim jest model relacyjny nie oddawałoby istoty problemu. Dlatego też model dedukcyjny wydaje się być odpowiedni do ilustracji laboratorium medycznego. Przykładowy model struktury bazy składa się z siedmiu modułów:

1. REJESTRU – kartoteka pacjentów, rejestracja zleceń, katalog badań;
2. PRACOWNI – weryfikacja zleceń, książka zleceń, wyniki i metody;
3. KONTROLI – materiały kontrolne, próby kontrolne, dokumentacja;
4. MAGAZYNU – kartoteka materiałowa, przychody, rozchody, zestawienia, rozliczenia;
5. ANALIZY – wyznaczanie norm, weryfikacja hipotez, prezentacja;
6. ADMINISTRATORA – konfiguracja, sterowanie, zabezpieczenia, kopie, użytkownicy;
7. ARCHIWUM – archiwizacja, wyniki zbiorcze, zestawienia, rozliczenia;

Schemat blokowy przykładowej bazy danych przedstawia rys. 10.1.



Rys. 10.1. Schemat przykładowej bazy danych Laboratorium

Wszystkie informacje przetwarzane w tych modułach są umieszczone w tabelach. Jest wiele tabel, np. *PACJENT* (id.pacjenta, nazwisko, imię, data urodzenia, ..., kategoria pacjenta), *ODDZIAŁ* (id.oddziału, kod, nazwa), *BADANIA* (id.badania, id.pa-

cjenta, id.oddziału, id.kategorii badania, ..., archiwum) *WYNIKI* (id,wyniku, id.nazwy, id.grupy, wynik liczba, jednostki, ..., data wykonania badania) i inne, gdzie *PACJENT, WYNIKI, BADANIA* to nazwy encji, a w nawiasach podano atrybuty opisujące daną encję. Pomiędzy tabelami istnieją powiązania typu jeden do wielu, które na równi z wartościami atrybutów umieszczonymi w tabelach są nośnikami informacji.

10.2. Pozyskiwanie wiedzy

Tworzenie i pozyskiwanie wiedzy jest najtrudniejszym elementem budowy systemów dedukcyjnych baz danych [1, 3, 18]. Wchodzą tu w grę między innymi takie czynniki, jak: trudności z usystematyzowaniem wiedzy, jej obszerność, możliwość popełnienia pomyłek itp. W podanym przykładzie wiedzę uzyskano drogą konsultacji z ludźmi pracującymi w laboratorium medycznym, ekspertami w tej dziedzinie. Analiza i proces wnioskujący zostały przeprowadzone na podstawie badań tarczycy [14, 16, 17]. Do analizy potrzebne były:

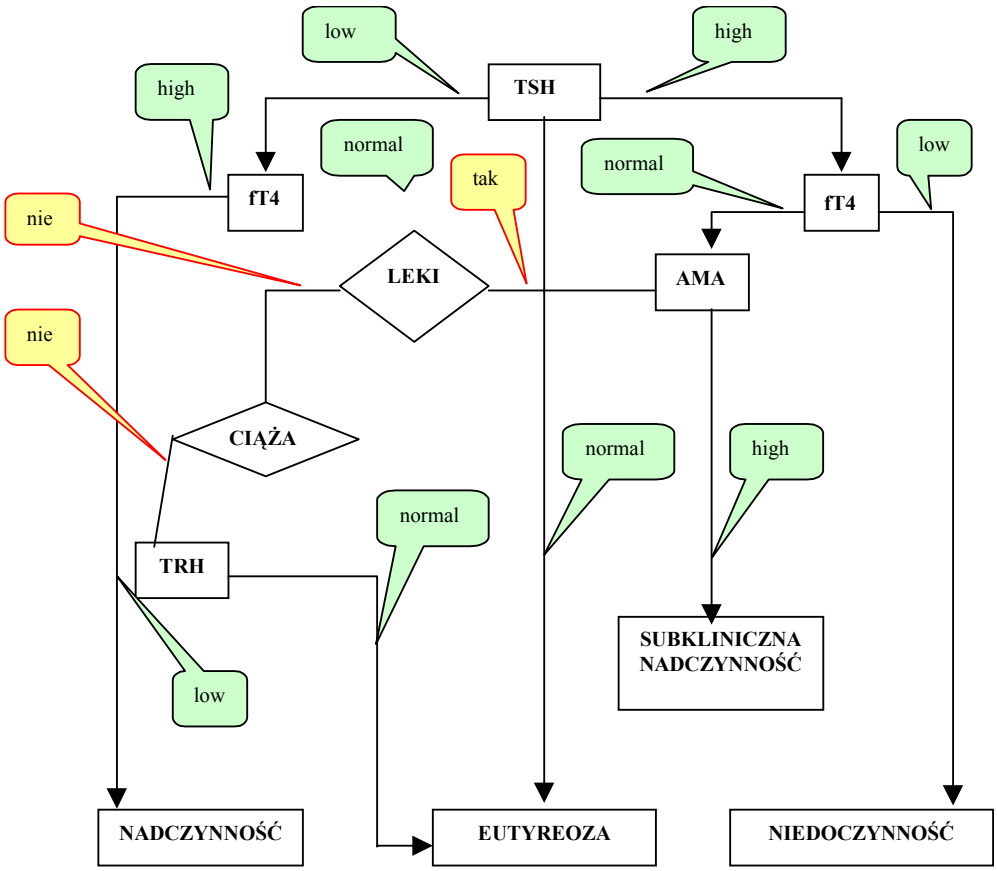
– fakty bazowe zawierające: informacje o pacjentach oraz zestaw badań, np:
- wyniki i normy badania TSH,
- wyniki i normy badania fT4,
- wyniki i normy badania AMA,
- wyniki i normy badania TRH,
- płeć pacjenta;

– fakty wirtualne**,** które uzyskuje się w czasie trwania analizy i są odpowiedziami na postawione przez system wnioskujący pytania:
- czy pacjent jest w trakcie brania konkretnych leków, które mają wpływ na wyniki poszczególnych badań,
- w przypadku gdy pacjent jest kobietą, system może poprosić o informację czy pacjentka nie jest w ciąży,
- czy wyniki podanych wyżej badań mieszczą się w normach, czy też przekraczają dolną lub górną granicę.

Graf przedstawiony na rys. 10.2 pokazuje poszczególne etapy wnioskowania. Widać na nim, że możliwe są do sformułowania cztery hipotezy, konkretne stany chorobowe tarczycy. W przypadku, gdy wynik badania TSH będzie poniżej dolnej granicy normy, a wynik badania fT4 będzie w normie, użytkownik będzie proszony o udzielenie odpowiedzi na pytanie „czy analizowany pacjent bierze jakieś leki”. Możliwe jest zadawanie dwóch rodzajów pytań:

– pytanie sformułowane w taki sposób, że możliwa jest odpowiedź tak lub nie;

– pytanie o większej liczbie odpowiedzi, w którym zbiór możliwych odpowiedzi musi zostać wcześniej określony. Przypadek drugi jest również trudniejszy w analizie dla programisty. Należy zatem tak formułować pytanie, aby zbiór odpowiedzi był możliwie jak najmniejszy.

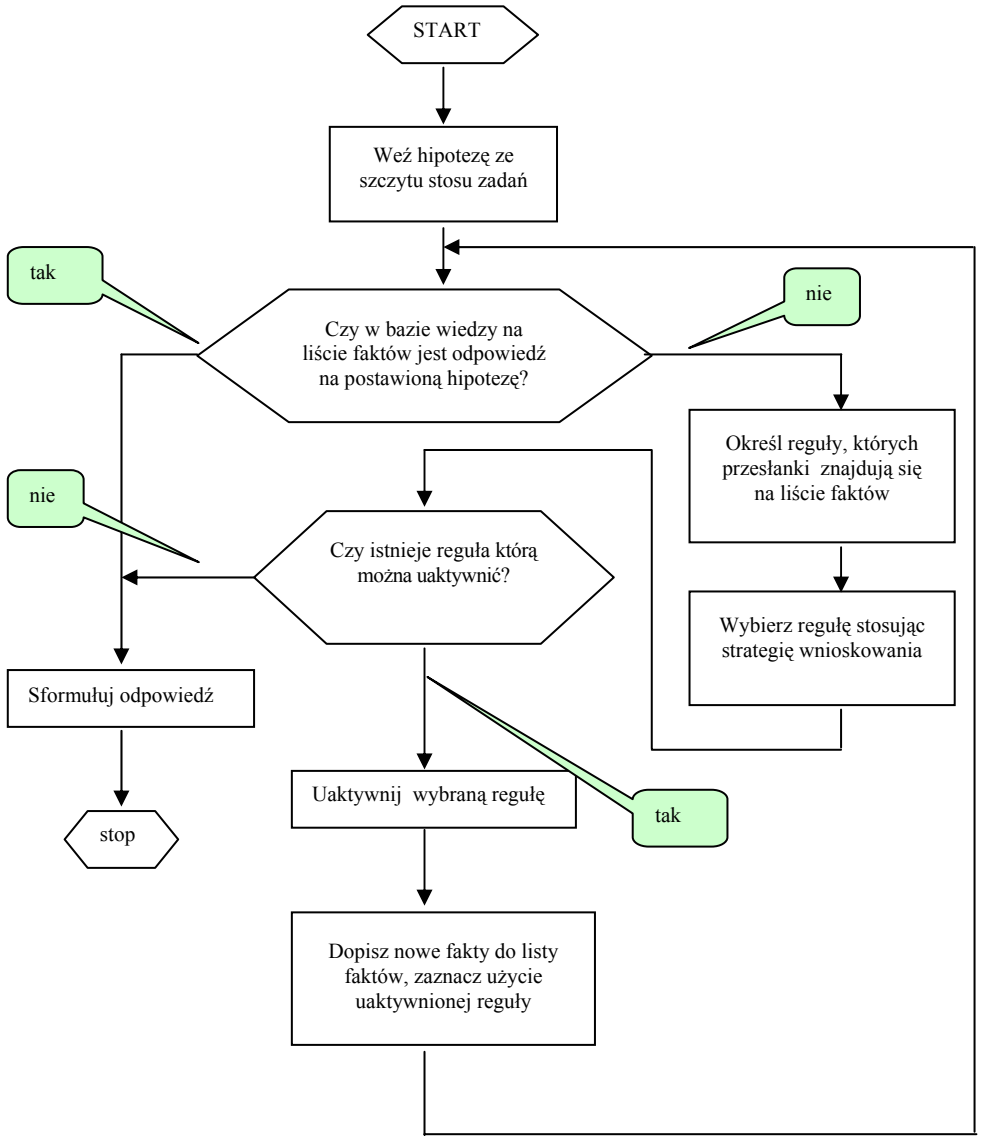


Rys. 10.2. Przykładowy graf do prowadzenia procesu wnioskującego

HIPOTEZY	NADCZYNNOŚĆ TARCZYCY	EUTYREOZA	SUBKLINICZNA NIEDOCZYNNOŚĆ	NIEDOCZYNNOŚĆ TARCZYCY
FAKTY:				
TSH-LOW	+ +	+ + +		
TSH-NORMAL		+		
TSH-HIGH		+	+	+
fT4-LOW				+
fT4-NORMAL	+	+ + + +	+	
fT4-HIGH	+			
TRH-LOW	+			
TRH-NORMAL		+		
AMA-NORMAL		+		
AMA-HIGH			+	
Czy pacjent bierze leki?	N	N N T		
Czy pacjentka jest w ciąży?	N	N T		

Rys. 10.3. Przykład tabeli decyzyjnej

Ze względu na prostotę, czytelność oraz zrozumiałość dla nieprofesjonalistów do pozyskania i usystematyzowania wiedzy została użyta tabela decyzyjna (rys. 10.3). Znak + oznacza, że dany fakt jest prawdziwy.



Rys. 10.4. Algorytm maszyny wnioskującej progresywnie

Kolumny tej tabeli zawierają hipotezy, a wiersze fakty związane z wynikami konkretnych badań laboratoryjnych oraz pytania zadawane przez system podczas procesu wnioskującego. Daje to przejrzysty obraz danej dziedziny. W tabeli znak „+" odpowiada zachodzeniu konkretnych faktów. Na podstawie tej tabeli decyzyjnej zbudowana jest baza reguł i przeprowadzone wnioskowanie progresywne (rys. 10.4). Wnioskowanie to polega na wybraniu jednej z możliwych hipotez, a następnie generowanie za pomocą zdefiniowanych reguł nowych faktów aż do momentu, kiedy dana hipoteza zostanie udowodniona, albo zbiór reguł zostanie wyczerpany.

Kolejne kroki algorytmu maszyny wnioskującej progresywnie można opisać w następujący sposób. W procesie wnioskowania korzysta się z pewnego obszaru pamięci roboczej programu, gdzie przechowuje się wyniki tak zwanego stosu zadań. Rozwiązanie problemu zaczyna się od umieszczenia hipotezy na szczycie stosu zadań (w naszym przykładzie subkliniczna niedoczynność tarczycy), następnie system przegląda listę faktów w bazie wiedzy i sprawdza, czy nie ma tam odpowiedzi na postawioną hipotezę.

Jeżeli jest, to następuje sformułowanie odpowiedzi i zakończenie procesu wnioskowania. Jeżeli w bazie nie ma odpowiedzi na wybraną hipotezę, zostaje określony zbiór reguł, których przesłanki znajdują się na liście faktów. Następnie, stosując strategię sterowania wnioskowaniem, wybiera się odpowiednie reguły. Kolejnym krokiem jest sprawdzanie, czy wybraną regułę można uaktywnić. Jeżeli nie, to proces jest zatrzymany. Oznacza to, że na podstawie zgromadzonych reguł i faktów nie można udowodnić wybranej hipotezy. W przeciwnym razie, po uaktywnieniu reguły, konkluzja jej jest wprowadzana na listę faktów oraz jest odnotowane, że wybrana reguła została użyta w bieżącym procesie wnioskującym – strategia blokowania. Po wprowadzeniu nowego faktu na listę system sprawdza ponownie, czy na liście faktów znajduje się wybrany cel czyli hipoteza. Proces jest powtarzany tak długo, aż zostanie osiągnięty cel lub zostaną wyczerpane wszystkie możliwości jego wykazania.

10.3. Baza wiedzy

Na bazę wiedzy składa się baza faktów i baza reguł. Gromadzenie faktów rozpoczyna się podczas tworzenia całego systemu. Zbierane są szczegółowe dane o pacjencie oraz o badaniach, jakie są zalecane. W bazie faktów mogą występować trzy typy faktów:

– fakty bazowe gromadzone podczas tworzenia systemu,
– fakty wnioskowane wprowadzone jako konkluzje uaktualnionych reguł,
– fakty podane przez użytkownika w czasie procesu wnioskowania.

Różna jest żywotność tych trzech rodzajów faktów. O ile fakty bazowe utrzymują swoją żywotność w czasie całej sesji trwania wnioskowania, o tyle fakty podane przez użytkownika są ważne do końca cyklu wnioskowania, następnie można je zmienić, odpowiadając inaczej na pytania zadawane przez maszynę wnioskującą. Oczywiście

tylko nowy zestaw przechodzi do następnego cyklu wnioskowania, a wszystkie inne tracą ważność i zostają usunięte z bazy. Inaczej przedstawia się sprawa z bazą reguł. W przedstawionym modelu regułą może być:

a) predykat rekurencyjny:

badanie (badanie, badanie cząstkowe, wynik), gdzie *badanie cząstkowe (badanie, wynik)*–predykat *badanie* opisuje badanie zawarte w bazie oraz łączy badania złożone z jego komponentami, które mogą występować samodzielnie i wielokrotnie.

b) predykat wirtualny:

wynik (*norma, badanie*) ⇐
prawidłowa wartość
AND *wynik* > od wartości
AND *wynik* < od wartości

Reguła ta wyznacza badania, dla których *wynik* badania jest poprawny. Ciało reguły definiującej wirtualny predykat *wynik* odwołuje się wyłącznie do predykatu bazowego *prawidłowa wartość* (dane te są określane poprzez normy konkretnego badania). Ogólnie w ciałach reguł można odwoływać się do wielu predykatów zarówno bazowych, jak i wirtualnych, stosując dodatkowo kwantyfikatory ogólne i (lub) szczegółowe (w języku SQL *some* lub *any* i *exists*). Zbiór reguł w przykładzie został sformułowany na podstawie tabeli decyzyjnej. Reguły w takiej postaci w literaturze anglojęzycznej określane są jako *production rules*.

IF *warunek*_1
AND *warunek*_2
.....
AND *warunek_n*
THEN *konkluzja*

Mechanizm wnioskowania dla tego typu reguł próbuje ustalić prawdziwość poszczególnych warunków danej reguły. Proces ustalania prawdziwości warunków powoduje uruchamianie procedur rozpatrujących inne reguły, których konkluzjami są warunki procedury pierwotnej. Pozytywne rozpatrzenie wszystkich warunków powiązanych operacją koniunkcji pozwala na udowodnienie konkluzji. Zaletą reguł jest to, że odpowiadają ludzkiej intuicji i w prosty sposób reprezentują wiedzę heurystyczną oraz dają się łatwo modyfikować. Można zmieniać reguły, nie zwracając uwagi na powiązania tej reguły z innymi. Oczywiście nie można pozwolić na całkowitą dowolność w strukturze bazy reguł. Zapętlenia i reguły sprzeczne mogą powodować nieprzewidziane działanie systemu. Aby ułatwić wprowadzanie reguł, opracowano specjalny format reguł z wykorzystaniem struktury plików typu *ini*. Ogólny format reguł ma postać.

[RULE_1] *nazwa reguły w bazie*,
TAKE=FALSE *parametr reguły* określający, czy reguła została wykorzystana w procesie wnioskowania; po uaktywnieniu jakiejś reguły wartość tego *parametru*

zmieniana jest z FALSE na TRUE. Po zakończeniu procesu wnioskowania *parametr* TAKE we wszystkich regułach ustawiana jest na wartość początkową FALSE.
COUNT=n *faktyczna liczba warunków* jakie muszą zostać spełnione, aby konkluzja reguły została wprowadzona do bazy faktów.
W1: WARUNEK_1 *konkretne warunki*
W2: WARUNEK_2
….
….
….
Wn: WARUNEK_n
K: KONKLUZJA *parametr określa konkluzję reguły*

Edycji reguł można dokonywać w dowolnym edytorze tekstowym. Ważne jest, aby format wprowadzanej reguły był dokładnie taki jak zostało to przewidziane.

10.4. Badanie poprawności bazy wiedzy

Jakość bazy wiedzy w istotnym stopniu decyduje o właściwym działaniu systemu. Poprawność bazy możemy badać zarówno w trakcie tworzenia reguł, jak i w trakcie działania systemu. Podstawowe problemy występujące w bazach wiedzy to:
– nadmierna liczba warunków,
– sprzeczność reguł,
– zapętlenie się reguł,
– pochłanianie reguł,
– wielokrotne odwoływanie się do jednego atrybutu,
– spójność.

Z **nadmierną liczbą warunków** mamy do czynienia wówczas, gdy dwie reguły mają niepotrzebne warunki i obie są pochłaniane przez trzecią regułę. Przykładem mogą być następujące reguły:
(AMA,N) **AND** (fT4,N) **AND** (TSH,H) ⇒ EUTYREOZA
(AMA,L) **AND** (fT4,N) **AND** (TSH,H) ⇒ EUTYREOZA
przy czym atrybut AMA może przyjmować wartości N – normal, L – low. Obie te reguły są pochłaniane przez regułę:
(fT4,N) **AND** (TSH,H) ⇒ EUTYREOZA

Dwie reguły **są sprzeczne** wówczas, gdy ich części warunkowe są spełnione równocześnie lub nie są spełnione we wszystkich możliwych sytuacjach i ich części konkluzyjne są różne dla przynajmniej jednej sytuacji. Dwie reguły są sprzeczne:
(AMA,N) **AND** (fT4,N) **AND** (TSH,H) ⇒ EUTYREOZA
(AMA,N) **AND** (fT4,N) **AND** (TSH,H) ⇒ SUB.NIEDOCZYNNOŚĆ

Zestaw reguł tworzy pętlę, jeżeli uaktywnienie tych reguł jest cykliczne. Aby wykryć zapętlenie reguł dla badanej bazy, buduje się specjalną tabelę zwaną tabelą zależności. W tabeli takiej przedstawia się zależności istniejące między regułami oraz między regułami a celem do wykazania.

Jedna reguła jest **pochłaniana** przez inną regułę wówczas, gdy część warunkowa pierwszej reguły jest spełniona, jest również spełniona część warunkowa drugiej reguły i części konkluzyjne obu reguł są identyczne. Na przykład reguła:

(AMA,N) **AND** (fT4,N) **AND** (TSH,N) ⇒ EUTYREOZA

jest pochłaniana przez regułę

(TSH,N) ⇒ EUTYREOZA

Pochłaniające reguły dają podobny efekt jak reguły z nadmierną liczbą warunków.

Wielokrotne odwoływanie się do jednego atrybutu zachodzi wówczas, gdy w części warunkowej występuje kilka członów zawierających ten sam atrybut. W przypadku gdy odwołania są identyczne, powtarzające warunki są zbędne i wydłużają niepotrzebnie czas wnioskowania. Natomiast w przypadku gdy nie są identyczne, reguła nigdy nie zostanie uaktywniona, ponieważ atrybut nie może jednocześnie przyjmować kilku wartości, np.:

(TSH,N) **AND** (TSH,H) ⇒ EUTYREOZA

Wielokrotne odwoływanie się do jednego atrybutu w części wynikowej jest błędem logicznym, stwarzającym domniemanie wystąpienia błędnego powiązania atrybutu wynikowego z jego wartością.

Testowanie **spójności** polega na wykrywaniu reguł zbędnych, sprzecznych, pochłaniających, reguł z niepotrzebnymi warunkami oraz reguł zapętlonych. Nadmiarowość bazy wiedzy występuje wówczas, gdy pojawią się reguły zbyteczne. Dwie reguły są nadmiarowe, jeśli obie ich części warunkowe są równocześnie spełnione we wszystkich możliwych sytuacjach oraz ich części konkluzyjne są identyczne, np.:

(AMA,N) ⇒ EUTYREOZA

(AMA,L) ⇒ EUTYREOZA

gdzie N i L przyjmują odpowiednie wartości [3].

Mimo że redundancja powoduje nadmiarowość bazy wiedzy, proces wnioskowania odbywa się normalnie, chociaż jest wymagane przeszukiwanie większej liczby reguł.

10.5. Uwagi końcowe

Przedstawiony przykład bazy dedukcyjnej jest częścią większej całości zaprojektowanej pod nazwą Laboratorium medyczne [18]. Cały system podzielony jest na dwie części. Pierwszą z nich jest relacyjna baza danych. Jest to kompletny model laboratorium medycznego, opracowany na podstawie prowadzonej w laboratorium dokumentacji, dotyczącej sposobów rejestracji pacjentów i badań oraz wyznaczania wy-

ników. Dla poprawienia warunków pracy zostały stworzone mechanizmy wymuszania na użytkowniku wprowadzania poprawnych formatów danych, wyłączania pewnych funkcji w zależności od wykonywanych operacji, rozbudowany system obsługi błędów. Dodatkowo funkcje przeglądania i edycji konkretnych informacji zostały pomyślane tak, aby ich wywołanie było możliwe na kilka sposobów w zależności od potrzeb użytkownika. Część druga to część dedukcyjna, której zadaniem jest analiza zgromadzonych danych oraz wyciąganie konkretnych hipotez dotyczących stanu zdrowia wybranego pacjenta. W części tej znajdują się procedury i funkcje przetwarzające bazę wiedzy. Jest to baza dotycząca stanów chorobowych tarczycy.

Wymagania sprzętowo-programowe podyktowane są platformą systemu operacyjnego, pod którą pisana jest baza. W przypadku dużych baz wiedzy ważnym czynnikiem warunkującym sprawne działanie systemu jest stosunkowo szybki dysk twardy, ponieważ istnieje bardzo często potrzeba odczytu pliku z regułami w trakcie procesu wnioskowania.

PODSUMOWANIE

W części III pokazano możliwości wykorzystania przedstawionych wcześniej modeli danych w konkretnych zastosowaniach. Ze względu na niezwykłą uniwersalność i popularność modelu relacyjnego nie przedstawiono konkretnej implementacji tego modelu [20], a jedynie umiejętność wykorzystywania języka zapytań SQL. Niestety, model relacyjny, tak wygodny przy oprogramowywaniu różnego rodzaju sformatowanych baz danych (np. hurtownie, magazyny itp.), nie sprawdza się w przypadku niesformatowanych baz danych. Tutaj przydatne są dedukcyjne i obiektowe bazy danych [2]. Bazy te, oprócz standardowych zakresów typów atrybutów, mają poszerzony zakres predykatów bazowych, konstruktory umożliwiające agregację atrybutów. Dzięki wyrażaniu więzów integralności za pomocą reguł logiki pierwszego rzędu (bazy dedukcyjne) możliwe jest bardzo precyzyjne modelowanie powiązań między encjami i ograniczeń występujących w modelowanej rzeczywistości. Bazy te nie mają ograniczeń w modyfikowaniu reguł, więzów i integralności. Mają wiele nowych niezwykle przydatnych funkcji, na przykład warunkowe uaktualniania, uaktualniania hipotetyczne, czy też rekurencyjne predykaty, które istotnie zwiększają zakres zastosowań.

BIBLIOGRAFIA

[1] Angielski S., Jakubowski Z., *Biochemia Kliniczna*, Perseusz, Sopot 1996.
[2] Austin D., *Poznaj Oracle 8*, MIKOM, Warszawa 1999.
[3] Beynon-Davies P., *Systemy baz danych*, WNT, Warszawa 1998.
[4] Date C.J., *Twelve Rules for a Distributed Database*, Comp. World, June 8, 1987.
[5] Date C.J., *Wprowadzenie do baz danych*, WNT, Warszawa 1983.
[6] Delobel C., Adiba M., *Relacyjne bazy danych*, WNT, Warszawa 1989.
[7] Gruber M., *SQL Helion*, Gliwice 1996.
[8] International Standards Organisation Database Language SQL.ISO/IEC 9075, 1987.
[9] International Standards Organisation Database Language SQL with integrity Enhancment ISO/IEC 9075, 1989.
[10] ISO, Basic Reference Model of Open Distributed Processing, Part 1: Overview and Guide to Use. ISO/IEC JTC1/SC212/WG7 CD 10746-1, International Standards Organization 1992.
[11] Lustig D., *Język SQL dla relacyjnej bazy danych Oracle*, KSW Technimex, Wrocław 1992.
[12] Microsoft Corporation, Introduction to Programming Microsoft FoxPro 1993 (SQL Quiz).
[13] Miller T., Powell D., *SQL Księga eksperta*, Helion, Gliwice 1998.
[14] Muraszkiewicz M., Rybiński H., *Bazy Danych*, Akademicka Oficyna Wydawnicza RM, Warszawa 1996.
[15] Pankowski T., *Podstawy baz danych*, PWN, Warszawa 1992.
[16] Pawelski S., Maj S., *Normy i diagnostyka chorób wewnętrznych*, PZW, Warszawa 1993.
[17] Pawelski S., Maj S., *Normy i interpretacja badań laboratoryjnych*, PZW, Warszawa 1993.
[18] Prace: Kuśmierski W., *Dedukcyjna Baza danych modelująca laboratorium medyczne*, dyplom pod kierunkiem M. Chałon, ICT PWr.,Wrocław 1997.
Feluś T., Mrówczyński M., *Obiektowo zorientowana baza danych wspomagająca zarządzanie*, dyplom pod kierunkiem M. Chałon, ICT PWr., Wrocław 1999.
[19] Tsichritzis D.C., Lochovsky F.H., *Modele danych*, WNT, Warszawa 1990.
[20] Thurrott P., Brent G., Bogdarian R., Tendon S., *Arkana Delphi*, RM 1998.
[21] Ullman J.D., *Systemy baz danych*, WNT, Warszawa 1981.

CZĘŚĆ IV

Rozproszone bazy danych

Wiele osób uznało lata dziewięćdziesiąte za erę rozproszonych baz danych. Rozproszone bazy danych są bardziej skomplikowane niż scentralizowane z powodu dodatkowego czynnika komunikacji sieciowej. Mimo rozlicznych trudności związanych z rozproszeniem wiele organizacji rozpoczęło konstrukcje systemów rozproszonych baz danych.

P. Beynon-Davies

WPROWADZENIE

Systemy rozproszone zdają się wyznaczać naturalny kierunek rozwoju współczesnej informatyki. Mówiąc o systemach rozproszonych z reguły ma się na myśli system jako całość, z infrastrukturą sprzętową, sieciową, systemami operacyjnymi i oprogramowaniem użytkowym. Obecnie dość dobrze znane są już zagadnienia sieci komputerowych i rozproszonych systemów operacyjnych [18, 19]. Powstaje jednak potrzeba badania systemów rozproszonych pod kątem zarządzania danymi oraz aktualizacji i synchronizacji wykonywanych na nich działań. Do najważniejszych zalicza się tu problemy synchronizacji, spójności i rekonstruowania danych, tolerowania uszkodzeń, skalowania i przezroczystość tych systemów. Wiele osób również nie wyobraża sobie prowadzenia jakiejkolwiek działalności bez takich usług, jak: poczta elektroniczna, transmisja danych, czy korzystanie ze skarbnicy danych *www*. Dlatego też te problemy są wyzwaniem dla współczesnej nauki.

11. ARCHITEKTURA ROZPROSZONYCH SYSTEMÓW BAZ DANYCH

11.1. Uwagi ogólne

Coraz ważniejsze miejsce w istniejących systemach komputerowych zajmują systemy rozproszone [3, 6, 20]. System rozproszony (ang. *distributed system*) jest zbiorem samodzielnych komputerów wraz z urządzeniami peryferyjnymi, połączonych za pomocą sieci, na których zainstalowane jest rozproszone oprogramowanie systemowe. Użytkownicy takiego systemu powinni go odbierać jako jedno zintegrowane środowisko. Fakt rozproszenia powinien być tu niezauważalny. Do zainteresowania się systemami rozproszonymi przyczynił się między innymi szybki rozwój sieci rozległych, asynchronicznych sieci ATM i upowszechnienie łączy satelitarnych, co pozwoliło na budowę systemów o niespotykanym dotychczas zasięgu. Pojawiły się projekty tworzenia ogólnokrajowych infostrad, czyli sieci o dużej przepustowości i zasięgu, których trzon stanowią rozproszone systemy danych. Również ciągłe dążenie do efektywnego wykorzystania systemów w przedsiębiorstwach wymusiło powstawanie nowej klasy programów użytkowych wielkiej skali. Są to tzw. systemy komputeryzacji przedsięwzięć (ang. *enterprise computing systems*), w których największy nacisk kładzie się na wysoki stopień integracji i niezawodności, czyli na zachowanie spójności systemów niezależnie od miejsca i sposobu dostępu, oraz szybkie wykrywanie i usuwanie pojawiających się w nich awarii. Systemy takie muszą być dostępne dla użytkowników pracujących na różnym sprzęcie, których mogą dzielić znaczne odległości.

Potencjalna gama zastosowania systemów rozproszonych jest bardzo szeroka. Duże koncerny posiadające swoje oddziały w wielu krajach, giełdy, banki, linie lotnicze opierają swoje działanie na posiadaniu rozproszonego systemu komputerowego. Badania nad systemami rozproszonymi prowadzone są od wielu lat między innymi w Queen Mary and Westfield College, University of London, University of Berkeley, Cambridge University, Newcastle University. Badania niezależne prowadzone są przez wiele firm komputerowych, w tym Sun Microsystems, Xerox PARC, Informix, które implementują mechanizmy systemów rozproszonych w swoich produktach. Pierwsze prace badawcze nad systemami rozproszonymi pojawiły się na początku lat siedemdziesiątych. Wówczas to został opracowany przez Xerox PARC projekt serwera plików oraz opracowany przez Cambridge system oparty na puli procesorów. Prawdopodobnie pierwszy komercyjny system rozproszony powstał w 1980 roku w firmie Apollo Computers. W swej największej konfiguracji łączył on 1800 stacji

roboczych. Współczesne systemy rozproszone [1, 10, 21], to znaczy systemy otwarte, skalowane, tolerujące uszkodzenia, pojawiły się w połowie lat osiemdziesiątych wraz z systemami UNIX BSD 4.2+ opracowanym przez Sun Microsystem oraz Mach opracowanym w Carnegie-Mellon University, będącymi jądrem systemu operacyjnego dla systemów rozproszonych. Powstanie wydajnych systemów sieciowych oraz rozproszonych systemów operacyjnych dało możliwość budowania rozproszonych systemów baz danych. Podstawę tych systemów stanowią specjalizowane serwery baz danych, mające możliwość wymiany danych z innymi serwerami.

11.2. Podstawowe własności rozproszonego systemu baz danych

Rozproszony system baz danych to zbiór lokalnych baz danych stanowiących całość w sensie jednego modelu danych i koordynacji wykonywanych transakcji. W zależności od przedmiotu rozproszenia wyróżniamy rozproszenie pod względem funkcji oraz danych [3, 10]. Mówiąc o rozproszeniu funkcji mamy na myśli separację funkcjonalną między danymi i mechanizmami zarządzania nimi, czyli serwerem a aplikacjami użytkowników, które pobierają wymagane dane poprzez wysyłanie do serwera zapytań. Model obrazujący separację to znany model klient–serwer. W przypadku rozproszenia danych mamy do czynienia z ich rozlokowaniem na różnych serwerach, często oddalonych od siebie. Na ogół jednak ma miejsce zarówno rozproszenie funkcji, jak i danych. O użyteczności rozproszonych baz danych decydują następujące cechy: współdzielenie zasobów, otwartość, współbieżność, skalowalność, przezroczystość, tolerowanie uszkodzeń.

Współdzielenie zasobów jest pojęciem bardzo szerokim. Dotyczy zarówno zasobów sprzętowych, jak i oprogramowania. W przypadku baz danych najbardziej istotna jest możliwość współdzielenia danych. Pozwala ona bowiem na współdzielenie narzędzi pracy, takich jak programy komputerowe, i danych opracowanych przez poszczególnych użytkowników systemu. Umożliwia to tworzenie systemów wspomagania pracy zespołowej CSCW (ang. *Computer Supported Cooperative Working*), gdzie chyba najbardziej uwydatnia się współzależność użytkowania wspólnych danych w różnych stacjach roboczych. Systemy CSCW znajdują zastosowanie w pracach grup projektowych i systemach zdalnego nauczania. Aby z danego zasobu mogło korzystać wielu użytkowników, musi zostać zorganizowany odpowiedni sposób dostępu do niego. System rozproszony możemy zatem przedstawić jako zbiór zarządców zasobów i zbiór korzystających z nich aplikacji. W oparciu o takie abstrakcyjne przedstawienie systemu buduje się dwa modele. Pierwszy z nich to najbardziej obecnie znany i stosowany model klient–serwer, drugi to model oparty na obiektach (nie mylić z obiektowymi bazami danych) [14]. Jest on uważany za bardzo obiecujący, lecz pozostaje wciąż w fazie eksperymentów.

Otwartość systemu określa możliwość dokonania jego rozbudowy o nowe zbiory danych dzielonych. Rozbudowa ta nie wymaga rekonfigurowania systemu czy przeprojektowania istniejących już zbiorów, ani modyfikacji aplikacji klientów. Dobrym przykładem implementacji otwartości systemu jest *Informix Universal Server* (IUS) [2, 7, 12]. Ciekawym mechanizmem zawartym w tym produkcie jest możliwość wprowadzania własnych typów danych oraz definiowanie dla nich funkcji i operatorów. Są to tak zwane moduły *DataBlade*, które są dołączane do serwera na etapie jego konfigurowania i stają się jego integralną częścią [9]. Są na tym samym poziomie co typy wbudowane i mogą być wykorzystywane we wszystkich umieszczonych na serwerze bazach danych [4].

Współbieżność w systemach rozproszonych pojawia się jako naturalna konsekwencja jednoczesnej pracy wielu użytkowników. Równocześnie mamy tu do czynienia ze zjawiskiem równoległości. Procesy użytkowników działają jednocześnie w różnych systemach komputerowych. Największy i najpoważniejszy problem dotyczy synchronizacji dostępu do danych współdzielonych. Jest on zasadniczo rozwiązywany za pomocą kolejkowania dostępów dzielonych, lecz mechanizm kolejkowania nie rozwiązuje problemu aktualności danych. Bardzo ciekawy sposób realizacji współbieżności w systemach zarządzania bazami danych jest zawarty w architekturze *Informixa.* Do realizacji obsługi zadań pochodzących od wielu klientów serwer tworzy automatycznie tzw. *wirtualne procesory* [2].

Zdolność systemu do zmiany rozmiarów bez konieczności modyfikacji systemu wyjściowego określa się jako skalowalność systemu. Problemy związane ze skalowaniem pojawiają się głównie w oszacowaniu zakresu wielkości przetwarzanych danych i sposobie ich adresacji. Przeszacowanie prowadzi bowiem do marnowania pamięci oraz do zwiększenia nakładów systemowych do przetwarzania danych. Niedoszacowanie może prowadzić do unieruchomienia całego systemu i potrzeby modyfikacji wielu tabel. Problem skalowalności od lat jest tematem wielu badań. Próbuje się różnych rozwiązań polegających na stosowaniu danych zwielokrotnionych, użyciu pamięci notatnikowej (podręcznej), zwielokrotnienie serwerów przez ich replikacje itp.

Przezroczystość systemu to ukrywanie przed użytkownikami odrębności składowych systemu i przedstawianie go jako integralną całość. *Przezroczystość dostępu* oznacza, że dostęp do danych, zarówno lokalnych, jak i odległych, jest możliwy za pomocą tych samych działań. *Przezroczystość położenia* umożliwia dostęp do danych bez znajomości ich faktycznej lokalizacji. *Przezroczystość współbieżności* pozwala wielu procesom na niezakłócone operowanie na tych samych obiektach informacji. *Przezroczystość zwielokrotniania* umożliwia użycie wielu kopii obiektów danych w celu zwiększenia niezawodności i wydajności systemu bez wiedzy użytkowników o istniejących obiektach i o tym, czy pracują na oryginale, czy na kopii. *Przezroczystość awarii* ukrywa przed użytkownikami pojawiające się uszkodzenia w systemie i umożliwia programom użytkowym kończenie zadań. *Przezroczystość przemieszczania* określa zdolność przemieszczania obiektów informacji wewnątrz systemu bez

wpływu na działanie użytkowników. *Przezroczystość wydajności* pozwala na rekonfigurowanie systemu w celu poprawienia jego wydajności przez zmianę rozłożenia obciążenia systemu. *Przezroczystość skalowania* pozwala na zmianę skali systemu bez zmiany jego struktury i algorytmów użytkowych.

Szczególnie istotnym zagadnieniem rozproszonych systemów jest zapewnienie spójności danych w przypadku wystąpienia awarii systemu. Aby to uzyskać, stosuje się metodę redundancji sprzętowej. Używa się nadmiarowych elementów systemu lub odtwarzania programowego, czyli stosuje programy usuwające skutki uszkodzeń. Z punktu widzenia danych, redundancja sprzętowa służy do utrzymywania dodatkowej kopii danych na różnych nośnikach. Oprócz obsługi zwierciadlanej (ang. *disc mirroring*) dysków stosuje się replikację danych. Wiąże się to jednak z uruchomieniem dodatkowego serwera baz danych.

11.3. Cele projektowe dla baz rozproszonych

W rozproszonych systemach baz danych strategiczne znaczenie ma spójność, niezawodność i bezpieczeństwo. Bez zapewnienia dostatecznego ich poziomu, rozproszone bazy stają się praktycznie nieużyteczne. Jednym z najważniejszych elementów projektowych jest wprowadzenie do systemu baz danych jednolitego sposobu nazywania współdzielonych zasobów. Chcąc zyskać dostęp do zasobu należy podać jego identyfikator. Zwykle dla wygody użytkowników, aby ułatwić dostęp, wprowadza się nazwy niosące jakieś znaczenie (np. *www.gazeta.com*). Jednakże programy odwołujące się bezpośrednio do zasobów (np. przeglądarki stron *www* w Internecie) stosują inny, często numeryczny identyfikator. W przypadku Internetu podawany jest czteroczęściowy adres komputera macierzystego (ang. *host identifier*), który przykładowo może mieć postać 192.123.123.123 oraz numer portu (ang. *port number*), będący liczbą całkowitą określającą konkretny port komunikacyjny. Z powodu istnienia dwóch sposobów identyfikowania zasobów, innego dla użytkownika i innego dla aplikacji, należy zapewnić mechanizm tłumaczenia nazwy. Najczęściej stosuje się do tego bazy danych, w których przechowuje się nazwy wraz z odpowiadającymi im adresami komunikacyjnymi. Często się zdarza, że tłumaczenie odbywa się wielopoziomowo. Po każdym kroku otrzymywany jest adres niższego rzędu. Taka sytuacja ma miejsce np. w przypadku nazw hierarchicznych. Tłumaczenie nazw trwa aż do otrzymania nazw elementarnych. Ustalenie zależności między nazwą i zasobem, na który nazwa wskazuje, nazywa się wiązaniem. Zwykle nazwy są wiązane z atrybutami, będącymi adresami nazywanych obiektów. Utrzymanie i zarządzanie bazami danych wiązań pomiędzy nazwami i atrybutami zasobów określa się jako usługi nazewnicze. W Internecie usługi te realizowane są przy wykorzystaniu specjalizowanych serwerów DNS (ang. *Domain Name System*) GNS (ang. *Global Name System*) oraz X500 (standard usług katalogowych) zawierających bazy danych pozwalające na tłumaczenie nazw. Potrze-

ba kontekstowego tłumaczenia nazw jest bardzo istotna. Odwołanie do konkretnej tabeli w bazie danych ma sens tylko wtedy, gdy zostanie podana nazwa bazy, a często także serwera. Inaczej mówiąc, ta sama nazwa w zależności od użytego kontekstu może określać inny zasób współdzielony. Usługi nazewnicze zarządzają bazami danych w każdym z kontekstów oraz starają się zapewnić jak najbardziej aktualny stan nazw i ich odwzorowań na inną przestrzeń nazw. Utrzymanie spójności tych baz ma zasadnicze znaczenie dla komunikacji w rozproszonym systemie. Projektowanie systemów komunikacji daje podstawy do tworzenia otwartych systemów rozproszonych. Dwa podstawowe systemy komunikacji to klient–serwer i rozsyłanie grupowe. Komunikacja typu klient–serwer jest ukierunkowana na dostarczanie usług. Zakłada przesyłanie dwóch komunikatów: zamówienia i odpowiedzi. W przypadku rozsyłania grupowego komunikat jest przesyłany do określonej grupy procesów.

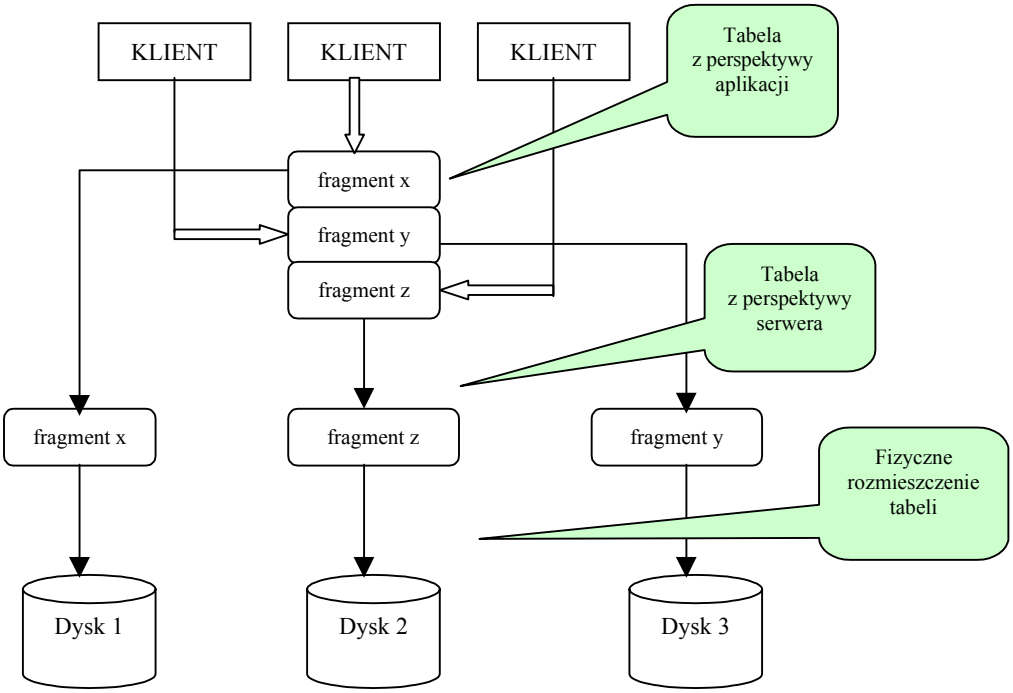
W rozproszonych systemach baz danych fundamentalną formę oprogramowania stanowią serwery baz danych. Od ich jakości zależy potencjalna użyteczność budowanych baz danych. Idea serwera pozwala na tworzenie systemów otwartych, gdyż do posiadanej bazy danych można dodawać nowe aplikacje bez konieczności modyfikowania istniejących aplikacji. Dodatkowym wymogiem dla rozproszonych baz danych jest potrzeba swobodnej wymiany danych pomiędzy wieloma serwerami baz danych. Serwery powinny być również wyposażone w system replikacji danych, pozwalający na prowadzenie lustrzanych serwerów baz danych.

11.4. Modele danych dzielonych i transakcji rozproszonych

11.4.1. Fragmentacja i replikacja danych

W rozproszonych systemach baz danych występuje rozłożenie danych poprzez ich fragmentację lub replikację do różnych konfiguracji sprzętowych i programistycznych umieszczonych w różnych węzłach sieci, zwykle różnych geograficznie miejsc. Rozproszenie pozwala przede wszystkim na odwzorowanie struktur organizacji, dla której system został zaprojektowany, umożliwia większą kontrolę nad danymi w miejscu ich wprowadzania i użytkowania. Rozproszenie uzyskane przez replikację zwiększa niezawodność systemu, pozwala zapewnić ciągłą pracę w przypadku wystąpienia awarii jednego z serwerów, poprawia dostępność systemu i pozwala na odciążenie łączy w przypadku znacznie oddalonych od siebie węzłów sieci. Dobrze zaprojektowana fragmentacja może znacznie podnieść wydajność systemu. Obok niewątpliwych zalet rozproszone bazy mają cały szereg wad. Katalog systemowy takiej bazy musi zawierać dodatkowo informacje o rozmieszczeniu fragmentów bazy oraz o sposobie replikacji. Znacznie bardziej jest skomplikowane sterowanie współbieżnością aktualizacji danych, testowanie systemu rozproszonego i odszukanie przyczyn ewentualnych nieprawidłowości. Ideę fragmentacji ilustruje rys. 11.1. Ten sposób podziału danych po-

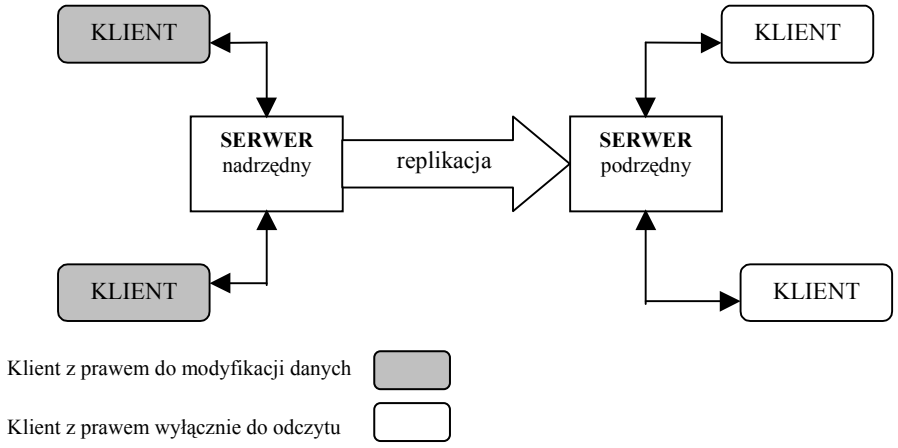
zwala na skrócenie czasu odpowiedzi poprzez możliwość wykonywania operacji wyszukiwania, aktualizacji i innych jako równoległych procesów, osobno dla każdego fragmentu tabeli.



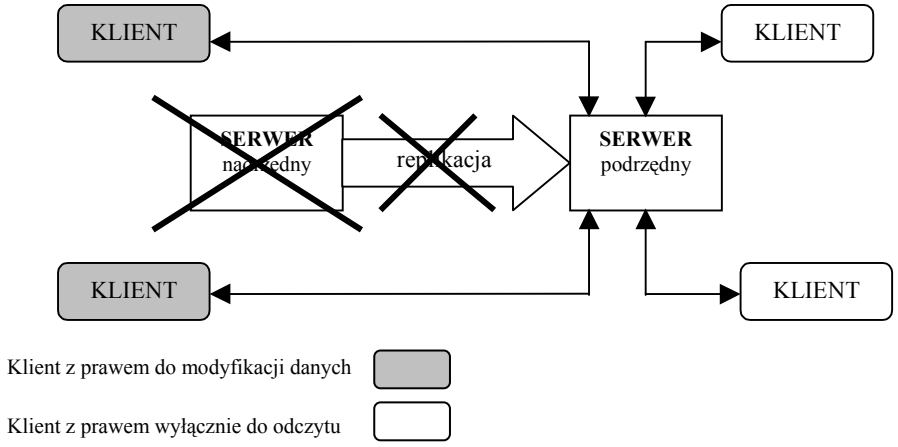
Rys. 11.1. Idea fragmentacji danych w IUS

W przypadku dużych, często używanych tabel można zmniejszyć współzawodnictwo o dostęp do danych rozmieszczając je w kilku osobnych fragmentach usytuowanych na różnych nośnikach danych. Gdy jeden z fragmentów staje się niedostępny, np. na skutek awarii dysku, użytkownicy mogą nadal swobodnie korzystać z danych zawartych w pozostałych, dostępnych fragmentach tabeli. Poprzez fragmentację tabeli można wykonać jej kopię zapasową oraz odzyskiwać z niej dane w sposób równoległy, skracając w ten sposób czas realizacji operacji. Można wyróżnić fragmentację poziomą i pionową. Fragmentacja pozioma to podzbiór zbioru wierszy tabeli wyselekcjonowanych według określonych wartości klucza, odpowiadająca operacji SELECT. Odtworzenie stanu pierwotnego, sprzed fragmentacji, wykonuje się korzystając z operacji UNION. Fragmentacja pionowa stanowi podzbiór zbioru kolumn tabeli, stanowiący określony zbiór cech opisywanego obiektu bazy. Powstaje dzięki operacji rzutowania PROJECT, a stan pierwotny uzyskuje się poprzez operację złączania JOIN kolumn w oparciu o wartości klucza głównego.

W ogólnym ujęciu replikacja danych to proces reprezentowania obiektów bazy w więcej niż jednym miejscu. Może zatem służyć do tworzenia lokalnych kopii w celu zwiększenia bezpieczeństwa systemu lub opracowania raportów bez dostępu do wszystkich danych zawartych w bazie. Replikacji może zostać poddana jedynie część bazy danych. Schemat systemu z replikacją danych pokazano na rys. 11.2.



Rys. 11.2. Schemat systemu z replikacją danych



Rys. 11.3. Przełączanie klientów po awarii serwera nadrzędnego

Mechanizm replikacji danych został zaimplementowany w IUS głównie w celu zapewnienia mechanizmów przetwarzania tolerującego uszkodzenia. Wymagane jest tu zastosowanie serwera nadrzędnego i podrzędnego (ang. *primary, secondary server*). Wszystkie operacje wykonywane na danych zgromadzonych w serwerze nadrzędnym są replikowane i automatycznie odświeżane na serwerze podrzędnym. Replikacja od-

bywa się tylko w jednym kierunku – od serwera nadrzędnego do podrzędnego. Narzuca to pewien wymóg, że klienci korzystający z serwera podrzędnego mogą wykonywać operacje wyłącznie w trybie odczytu. W przypadku awarii serwera nadrzędnego następuje przełączenie klientów do serwera podrzędnego (rys. 11.3) w sposób automatyczny lub ręczny, zależnie od ustawienia pewnych parametrów w pliku konfiguracyjnym.

Korzyścią wynikającą z zastosowania replikacji jest, obok zabezpieczenia przed utratą danych, odciążenie łączy i serwera nadrzędnego od obsługi klientów z prawem dostępu wyłącznie do odczytu. Odciążenie łączy jest szczególnie widoczne wtedy, gdy grupa klientów łączy się do serwera z dużej odległości. Tego typu sytuacje spotyka się w Internecie. W celu uniknięcia częstych, odległych połączeń stosuje się repliki popularnych serwerów (tzw. serwery lustrzane) zlokalizowane po stronie odległych klientów.

11.4.2. Hurtownie danych

Tradycyjne systemy baz danych służą do bieżącego rejestrowania stanu obiektów, ich zmian oraz pozwalają na szybkie dostarczanie informacji. Są to tzw. operacyjne bazy danych. Przykładem takiej bazy może być rozproszony system baz danych służący do rejestrowania wielu czynników pogodowych na danym obszarze, jak: ciśnienie atmosferyczne, wielkość opadów, temperatura powietrza, siła wiatru itp. W każdym węźle systemu zbierane są dane dotyczące wyłącznie pewnego obszaru. Dzięki zastosowaniu rozproszonego sytemu możliwe jest odtworzenie stanu pogody na danym terenie z dowolnego węzła systemu.

Oprócz operacyjnych systemów coraz większe znaczenie zaczynają mieć analityczne systemy baz danych tworzone w postaci hurtowni danych. Celem ich jest określenie pewnych trendów i wzorców zachowań określonych obiektów danych. W oparciu o te systemy budowane są systemy wspomagające podejmowanie decyzji. Operują one na danych historycznych uzyskiwanych z operacyjnych baz danych, pozwalają na aproksymację zachowań obiektów danych zawartych w bazie. Można zaproponować budowę systemu analitycznego do prognozowania zachowań czynników pogodowych, opracowania modeli klimatu na obszarze działania systemu, czy szacowania stanu rzeki na podstawie danych o stanie i zmianach pogody na określonym obszarze. Jak widać z tego przykładu, do tworzenia hurtowni danych wyjątkowo użyteczne są systemy rozproszone. Gromadzą one w sposób lokalny dane zgodnie z miejscem ich użytkowania. Do prowadzenia natomiast globalnej analityki rejestrowanych procesów dane mogą być pobierane ze wszystkich lokalnych węzłów systemu. Aby hurtownia spełniała kryteria analitycznej bazy, powinna wykazywać następujące własności:

– *integrację* w sensie jednolitego sposobu kodowania informacji, stosowania jednolitych formatów pól, identyfikacji obiektów,

– *orientację tematyczną* – informacje o obiektach jednego typu mogą pochodzić z różnych źródeł,

– *nieulotność* – dane raz wprowadzone do hurtowni nie ulegają modyfikacji, są tylko odczytywane,

– *orientację czasową* – dane w hurtowni mają charakter historyczny i musi być im nadany identyfikator czasowy.

Informacje do hurtowni pochodzą z lokalnych serwerów i zwykle są poddawane agregacji w celu wstępnego wyliczenia pewnych miar wykorzystywanych w późniejszych analizach. Składowane w hurtowni informacje są zwykle poddane wcześniejszej obróbce, np. przez wyliczenie średnich, i dopiero one zapisywane są do bazy analitycznej. To wstępne przetwarzanie danych pozwala przyspieszyć czas wykonywania analiz. Tworzenie analitycznych systemów baz danych jest często jednym z priorytetowych czynników uzasadniających budowanie rozproszonych systemów baz danych.

11.4.3. Transakcje

W celu zapewnienia spójności danych dzielonych i rozproszonych podczas operowania na nich należy stosować operacje niepodzielne, czyli transakcje. Aby mogły istnieć operacje niepodzielne, serwer musi mieć mechanizmy wzajemnego wykluczania. Zwykle są to zamki (ang. *mutex*) i blokady (ang. *lock*). Pojawia się zatem problem synchronizacji i harmonogramowania zamówień klientów. Mogą powstać takie sytuacje, w których operacja zapoczątkowana przez jednego klienta nie może być zakończona do czasu wykonania operacji przez innego klienta. Taka sytuacja niesie ze sobą niebezpieczeństwo powstania zakleszczeń (ang. *deadlock*). Mówiąc o operacjach niepodzielnych w kontekście systemów baz danych mamy na myśli sytuacje, w których skutek wykonania dowolnej operacji jest niezależny od operacji wykonywanych współbieżnie. Transakcje mogą posiadać własne zbiory transakcji, co pozwala na hierarchizację wykonywanych operacji. Takie transakcje określa się jako zagnieżdżone. Pozwalają one na dodatkową współbieżność, gdyż operacje na tym samym poziomie zagnieżdżenia mogą być wykonywane jednocześnie. Wykonywanie operacji na danych wielodostępnych wymaga zagwarantowania właściwego sterowania współbieżnością. Dopuszcza się współbieżnie wykonywane transakcje, pod warunkiem, że skutki ich wykonania będą takie same jak gdyby zostały wykonane po kolei. Takie transakcje określa się jako równoważne szeregowo. Mechanizmy pozwalające na osiągnięcie równoważności szeregowej, to: blokowanie, optymistyczne sterowanie współbieżnością, zastosowanie znaczników czasu. W razie blokowania na każdy obiekt zostaje założona blokada przez pierwszą sięgającą do niego transakcję. Dopóki blokada nie zostanie usunięta, żadna inna transakcja nie ma możliwości modyfikowania zablokowanych danych. Mechanizm blokowania jest bardzo złożony [16].

Przy optymistycznym sterowaniu współbieżnością zakłada się, że transakcje są rozdzielone, to znaczy nie występują żadne konflikty pomiędzy nimi. Takie założenie

jest często prawdziwe i znacznie przyspiesza cały proces realizacji zamówień aplikacji klientów. Gdy jednak dochodzi do konfliktu pomiędzy transakcjami, należy zaniechać ich wykonywania i ponowić próbę po pewnym czasie [16].

Mechanizm użycia znaczników czasu polega na nadawaniu transakcjom etykiet z niepowtarzalnymi znacznikami. Zawartość etykiet oznacza pozycję transakcji w ciągu czasowym, co pozwala na całkowite uporządkowanie zamówień pochodzących od transakcji w kolejności zgłoszeń.

Transakcje realizowane na wielu serwerach bazy danych jednocześnie określane są jako transakcje rozproszone. Do ich wykonania potrzebne są wielofazowe protokoły zatwierdzające. Jednym z podstawowych zadań protokołów zatwierdzających jest, by w razie wystąpienia błędów podczas realizacji transakcji rozproszonej zagwarantować spójność danych. Aby to było możliwe, protokół musi mieć odpowiedni mechanizm obsługi błędów.

11.5. Rekonstruowanie danych

W rozproszonych systemach baz danych szczególnie dużą uwagę należy zwrócić na poprawność wykonywanych operacji w celu zagwarantowania spójności danych. Jest to zadanie bardzo trudne, ponieważ w przypadku systemów rozproszonych zwielokrotnieniu ulegają punkty potencjalnych miejsc awarii. Należy więc zapewnić odpowiedni mechanizm gwarantujący odtworzenie stanu obiektów danych w razie wystąpienia awarii serwera. Najpowszechniejszą metodą stosowaną do rekonstruowania przebiegu transakcji jest wykorzystanie pliku rejestru zwanego plikiem rekonstrukcji. Jest to metoda dobra, ale możliwa do zastosowania jedynie w systemach, w których nie wymaga się natychmiastowej odpowiedzi, gdyż metoda ta jest stosunkowo wolna. W celu implementacji tworzony jest zarządca rekonstrukcji, który powinien być odporny na uszkodzenia własnego pliku rekonstrukcji. Odporność realizuje się przez zwielokrotnienie pliku rekonstrukcji na wielu trwałych nośnikach danych. Aby umożliwić przebieg rekonstrukcji, każdy serwer baz danych musi rejestrować historię zmian modyfikowanych obiektów danych. W tym celu tworzy listę zamiarów, dla wszystkich aktualnie działających transakcji, na których zawarte są nazwy i kolejne wartości obiektów danych modyfikowanych przez transakcje. Lista ta jest wykorzystana przez serwer po zakończeniu transakcji w celu identyfikacji modyfikowanych obiektów danych oraz usunięcia tymczasowych wersji danych. W pliku rekonstrukcji dodatkowo przechowywana jest informacja o stanie każdej transakcji oraz o skojarzeniu odpowiedniego obiektu danych z odpowiednią transakcją. Sposób operowania na pliku jest różny w różnych serwerach baz danych. Każdy obiekt danych ma jednoznaczny identyfikator, dzięki czemu kolejne wersje obiektu danych w plikach rekonstrukcji są kojarzone z danymi w serwerze. Po wystąpieniu awarii i wznowieniu prawidłowego funkcjonowania, serwer wykrywa potrzebę uruchomienia mechanizmu

rekonstrukcji. W celu optymalizacji czasu procesu rekonstrukcji trzeba okresowo reorganizować plik rejestru. Informacją potrzebną do odtworzenia stanu obiektów danych jest kopia wszystkich zatwierdzonych wersji obiektów danych. W praktyce tworzy się nowy plik rejestru, w którym umieszczone są kopie wszystkich zatwierdzonych obiektów danych oraz informacje o stanach i listy zamiarów transakcji jeszcze nie zakończonych. Taka operacja pozwala na znaczne zmniejszenie zapotrzebowania na pamięć i w istotny sposób wpływa na skrócenie czasu wykonywania rekonstrukcji bazy danych.

11.6. Bezpieczeństwo w rozproszonych systemach baz danych

Podczas konstruowania rozproszonych systemów baz danych napotyka się na sprzeczność, gdyż z jednej strony należy dążyć do największej otwartości tych systemów, z drugiej otwartość pociąga za sobą większe prawdopodobieństwo nielegalnego dostępu do systemu. Wśród zagrożeń na jakie jest wystawiony system występują:

przecieki – uzyskanie poufnej informacji przez nieupoważnionych do tego odbiorców,

nadużycia – zmiana wartości obiektów przez osoby lub aplikacje do tego nieupoważnione,

kradzież zasobów – użytkowanie zasobów bez uprawnień,

wandalizm – zaburzenie prawidłowego funkcjonowania systemu bez jakichkolwiek korzyści dla sprawcy.

Najtrudniejsze do wykrycia, a zarazem najgroźniejsze dla systemu są przecieki i nadużycia. Wandalizm jest powszechnie spotykany w Internecie, gdzie często dochodzi do podmienienia stron *www* na inne. W systemach rozproszonych komputery są połączone w sieć i wyposażone w systemy operacyjne posiadające standardowy interfejs komunikacyjny. Pozwala to na utworzenie kanałów komunikacyjnych. Naruszenie bezpieczeństwa polega na uzyskaniu dostępu do takiego kanału lub przejęciu go od osoby uprawnionej. W systemach rozproszonych jest to o tyle łatwiejsze, że w celu zapewnienia większej niezawodności niektóre kanały komunikacyjne są zwielokrotnione. Wśród metod przejmowania kanałów komunikacyjnych wyróżnia się:

podsłuch – odbieranie kopii komunikatów bez upoważnienia,

kamuflaż – podszywanie się pod innego użytkownika,

fałszowanie komunikatów – zmiana treści komunikatów,

powtarzanie starych komunikatów – wysyłanie komunikatów w późniejszym czasie.

Większość ataków na system rozproszony pochodzi od legalnych użytkowników. Ataki mogą polegać na prostym mechanizmie usiłowania odgadywania hasła użytkownika aż do bardziej wyrafinowanych form, jak wirusy komputerowe, robaki sieciowe, czy metoda konia trojańskiego. Wirusy komputerowe znane ze środowiska komputerów osobistych w systemach rozproszonych mają znacznie szersze pole działania i stanowią potencjalnie większe zagrożenie. Przy pomocy komunikacji sieciowej

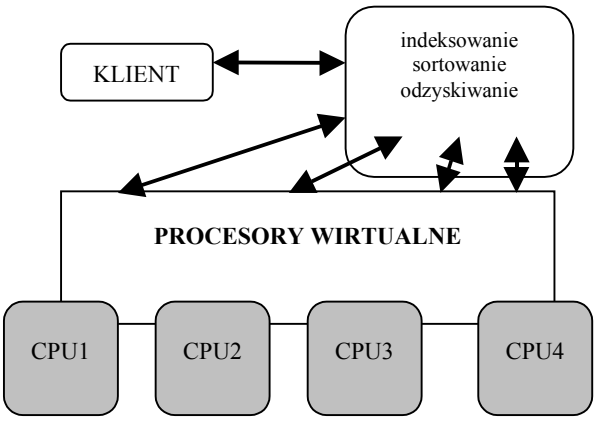
ulegają samoreprodukcji i powielaniu na kolejne części systemu rozproszonego. Robak ma możliwości zdalnego uruchamiania procesów w systemie. Jest on zwykle jednym z narzędzi administratora systemu, ale wykorzystany w niewłaściwy sposób przez nielegalnego użytkownika stanowi doskonałe narzędzie ataku. Metoda konia trojańskiego polega na wprowadzeniu programu-pułapki, który pozornie wykonuje pożyteczne czynności, lecz faktycznie służy nielegalnemu przechwytywaniu informacji.

Wśród mechanizmów obrony przed niepowołanym dostępem wyróżnione zostały: kryptografia, uwierzytelnienie i kontrola dostępu. Zastosowanie kryptografii w celu szyfrowania danych pozwala na utajnienie poufnych informacji w fizycznych kanałach komunikacji lub przy składowaniu w pamięci trwałej, gdzie są narażone na podsłuch czy fałszowanie. Kryptografia pozwala również na wspomaganie mechanizmu uwierzytelnienia, gdyż zaszyfrowana informacja według określonego klucza może zostać odszyfrowana tylko przez odbiorcę znającego odpowiedni klucz. Gdy odbiorca otrzyma informację, której nie jest w stanie we właściwy sposób odczytać, zachodzi podejrzenie, że nadawca nie był upoważniony do jej wysyłania. Dzięki kryptografii wprowadzono cyfrowe podpisy, którymi oznacza się np. obiekty danych. Istnienie podpisu cyfrowego pozwala określić, na zasadzie zbliżonej do sumy kontrolnej, czy do odbiorcy dociera oryginalna, czy też zmieniona postać informacji. Uwierzytelnienie jest natomiast podstawą metody identyfikacji serwerów i klientów. Kontrola dostępu oznacza udzielenie dostępu do wszelkich zasobów wyłącznie tym grupom użytkowników, które mają przyznane prawa na określonym poziomie hierarchii. Większość mechanizmów zabezpieczenia systemów rozproszonych musi być implementowana na poziomie sprzętu lub systemu operacyjnego. Jednakże sam serwer baz danych musi również zapewniać możliwość ograniczenia dostępu do zasobów. W przypadku IUS na poziomie języka SQL można nadawać i odbierać prawa dostępu (instrukcje *GRANT i REVOKE*) do baz danych i pojedynczych tabel. W konfiguracji serwera określa się użytkowników, którzy będą mieli dostęp do baz danych na danym serwerze. Dane są przechowywane przez serwer w specjalnie sformatowanym zabezpieczonym obszarze dysku (ang. *sbspace*), który nie jest standardowym systemem plików, co ogranicza możliwość niepowołanego odczytania tego obszaru. Dodatkowo serwer zapewnia metody śledzenia wszystkich akcji użytkowników, zapisując kiedy i przez kogo zostały podjęte i na którym obiekcie danych. Innym ze sposobów zabezpieczenia jest oprogramowanie separujące system lub jego fragmenty rozgraniczające pewne zasoby systemowe jak i sprzętowe. Stosuje się zapory ogniowe (ang. *firewall*) i serwery pośredniczące (ang. *proxy server*).

Aby skutecznie chronić dane i cały system, należy określić politykę bezpieczeństwa oraz mechanizmy bezpieczeństwa pozwalające na implementację przyjętej polityki.

11.7. Architektura IUS

Jako przykład narzędzia do realizacji rozproszonych systemów baz danych posłuży serwer IUS [2, 4, 9, 12, 13]. Architektura IUS jest określana jako obiektowo-relacyjna. Jej podstawę stanowią dwie właściwości: rozszerzalność i dynamiczna skalowalność. Przez rozszerzalność rozumiana jest tu możliwość dodawania do serwera nowych elementów, jak własne typy danych, funkcje operujące na nich (w tym istnieje możliwość definiowania własnych operatorów logicznych dla nowych typów danych) oraz metod dostępu do danych. Rozszerzenia tego typu w postaci tzw. modułów DataBlade są dołączane do serwera. Charakterystycznym przykładem stosowania modułów DataBlade jest wprowadzenie typu danych dźwiękowych, video i zdefiniowanie funkcji operujących na nich. Dynamicznie skalowalna architektura (ang. *Dynamic Scalable Architecture* – DSA) oznacza zdolność do skalowania zasobów w zależności od zapotrzebowania zgłaszanego przez aplikacje. Jest to możliwe szczególnie przy wykorzystaniu symetrycznych wieloprocesorowych systemów komputerowych (ang. *Symmetric Multiprocessing Computer Systems* – SMPs). Możliwa jest wówczas równoległa praca wielu procesorów dla pojedynczego zadania klienta. Podstawę dynamicznie skalowanej architektury IUS stanowią procesory wirtualne (ang. *virtual processor*). Dają one możliwość dzielenia przetwarzania oraz oszczędność pamięci i zasobów systemu. Jedną z podstawowych korzyści płynących z wykorzystania procesorów wirtualnych jest możliwość przetwarzania równoległego. W niektórych działaniach (rys. 11.4) procesory wirtualne z klasy CPU (jednostki centralnej) pozwalają na utworzenia wielu



Rys. 11.4. Przetwarzanie równoległe dla pojedynczego klienta z wykorzystaniem procesorów wirtualnych

wątków działających równolegle dla pojedynczego klienta. Jest to możliwe w operacji indeksowania, sortowania, odzyskiwania, przeszukiwania, łączenia, agregacji i gru-

powania. Zastosowania przetwarzania równoległego w tych operacjach powoduje skrócenie czasu realizacji poszczególnych zadań, a co za tym idzie skrócenie czasu odpowiedzi na pytania użytkownika. Dzięki zastosowaniu procesorów wirtualnych serwer potrafi obsługiwać w sposób równoległy operacje na tabelach poddanych fragmentacji. Do operacji na każdym fragmencie można utworzyć osobny procesor wirtualny.

Procesory wirtualne wykorzystują pamięć dzieloną przy operowaniu na danych wspólnie wykorzystywanych oraz do szybkiej komunikacji między sobą. Do harmonogramowania przetwarzania jednocześnie realizowanych wątków IUS stosuje kolejki. Kontrola współbieżności jest realizowana z wykorzystaniem zamków i blokad. Do zapewnienia spójności danych zapisywanych w pamięci stałej i dzielonej serwer stosuje znaczniki czasu. Oprócz nich wykorzystuje się również mechanizm sekcji krytycznej oraz punktów kontrolnych.

11.8. Uwagi końcowe

Miarą jakości i przydatności projektowanych systemów rozproszonych baz danych są ich podstawowe własności: współdzielenie zasobów, otwartość, współbieżność, skalowalność, przezroczystość i tolerowanie uszkodzeń. Od doboru rozwiązań technicznych związanych z projektowaniem rozproszonych systemów baz danych zależy stopień osiągnięcia podstawowych własności tych systemów. Pojawiają się tu dodatkowe aspekty funkcjonalności systemu, możliwości jego rekonfigurowania, oraz jakości usług rozumianych jako wydajność systemu, niezawodność, dostępność oraz bezpieczeństwo. Możliwość zapewnienia tych aspektów, które są najbardziej dostrzegalne przez użytkownika końcowego systemu, zależy w głównej mierze od doboru odpowiednich rozwiązań projektowych i zwykle decyduje o ocenie systemu. W systemach rozproszonych ważnym kryterium projektowym jest komunikacja w systemie. Jest ona środkiem pozwalającym na połączenie poszczególnych elementów systemu. Projektowanie schematów komunikacyjnych, jak klient–serwer czy rozsyłanie grupowe, daje podstawy do tworzenia otwartych systemów rozproszonych. W tego typu systemach istotna jest strukturalizacja oprogramowania, która uwidacznia się podczas projektowania serwerów baz danych. Serwery te muszą pozwolić na łatwe dodawanie nowych zasobów danych współdzielonych oraz dzielenie danych i utrzymywanie spójności. Użytkownik musi mieć całkowite zaufanie do prawdziwości i aktualności otrzymywanych odpowiedzi. Jeszcze jedną zaletą tworzenia rozproszonych baz danych jest opracowanie na podstawie danych zgromadzonych w różnych węzłach sieci analitycznych baz danych organizowanych w tzw. hurtownie danych.

Omawiając rozproszone bazy danych wielokrotnie wspominano o dynamicznie rozwijającej się sieci Internet. Niezwykła popularność tego narzędzia sprawiła, że prezentacja danych w Internecie jest tematem następnego rozdziału.

12. PREZENTACJA DANYCH W INTERNECIE

12.1. Uwagi ogólne

Ostatnio można zaobserwować gwałtowny wzrost prezentacji różnego typu informacji, a to głównie za sprawą Internetu [11, 16, 17]. Możliwość prezentacji różnego typu materiałów na graficznych stronach *www* przyczyniła się właśnie do tak błyskawicznego wzrostu zainteresowania integracją baz danych z serwerami *www*. Ma to podstawowe znaczenie przy budowie szerokiej gamy usług internetowych i intranetowych (sieć informatyczna w przedsiębiorstwach). Można podać wiele praktycznych przykładów zastosowania integracji baz danych z możliwościami *www* począwszy od najprostszych zasobów archiwalnych umożliwiających wyszukiwanie tekstowe, prezentację tabel z danymi tekstowymi i liczbowymi poprzez systemy ogłoszeń, systemy śledzenia zamówień, aż po elektroniczne multimedialne katalogi czy nawet sklepy internetowe. Firmom rzucono wyzwanie, gdyż korzyści jakie przynosi posiadanie zintegrowanej bazy danych z serwerem *www* są niewątpliwe. Jednak osoby podejmujące decyzję, czy zastosować nowości programistyczne we własnej firmie czeka jeszcze trudne zadanie znalezienia miejsca dla nowego oprogramowania w istniejącej strukturze informatycznej. Integracja bazy danych z serwerami *www* obejmuje takie zagadnienia, jak: szeroko rozumiana architektura systemu, wydajność, aspekty konfiguracji i administracji, łatwość i intuicyjność wykorzystania oraz bezpieczeństwo danych i transmisji.

Internet ma największy i praktycznie nieograniczony zasięg. Użytkownik może w nim korzystać z wielu usług. Bardzo prężnie rozwija się usługa, dająca możliwość zaprezentowania swojej graficznej wizytówki na stronach *www*. Umożliwia ona w łatwy i przyjazny dla użytkownika sposób zaprezentowanie swoich danych, jak również korzystanie z zasobów udostępnianych na serwerach całego świata. Ogromną zaletą jest niezależność od stosowanej platformy sprzętowej i systemowej.

www stwarza możliwość prezentacji swoich danych oraz odczytywania informacji udostępnianych przez ogromną liczbę jej użytkowników. Aby przedstawić swoje dane, musimy stworzyć własny dokument i umieścić go w serwerze *www*. Odczytywanie odbywa się przy pomocy specjalnych programów nawigacyjnych zwanych przeglądarkami. Ich zadaniem jest nawiązanie komunikacji z serwerem i odpowiednia interpretacja znajdujących się tam dokumentów. Producenci wzbogacają przeglądarki o nowe możliwości korzystania z innych usług internetowych: poczty elektronicznej, protokołów transmisji plików, list dyskusyjnych.

Na potrzeby prezentacji dokumentów w sieci Internet powstał język programowania HTML (ang. *HyperText Markup Language*). Dokument HTML jest zwykłym plikiem tekstowym, w którym znajdują się polecenia HTML. Oznacza to, że w dokumencie jest opisywana struktura a nie wygląd, który zależy od używanego przez klienta programu przetwarzającego, tzw. przeglądarki. Wynika z tego, że dokument taki można utworzyć za pomocą najprostszego edytora tekstów, ręcznie dodając znaczniki. Jednak na rynku pojawiło się już wiele specjalizowanych edytorów, które wydatnie ułatwiają konstruowanie dokumentu, wspomagając wprowadzenie poleceń.

Internet miał początkowo służyć udostępnianiu wielu danych w formie ściśle sprecyzowanej przez standardy HTML. Świat jednak poszedł gwałtownie do przodu i użytkownicy tej rozległej sieci zaczęli dodawać coraz to nowsze i bardziej pomysłowe rozwiązania. Każdy właściciel własnej witryny chce, aby prezentowany przez niego graficzny serwis był atrakcyjny i użyteczny. Gwałtownie wzrasta też zapotrzebowanie na większą ilość informacji, publikowanych za pośrednictwem sieci Internet. Kolejnym wielkim krokiem była możliwość przedstawienia zawartości baz danych bezpośrednio na stronach *www*. Nastąpiła zmiana charakteru, od czysto statycznych stron do dynamicznie aktualizowanych wiadomościami zawartymi w bazach danych. Standard HTML nie uległ wielkim przeobrażeniom. Modyfikowano go tylko w kierunku wykorzystania skryptów i apletów Javy, które wykorzystując odpowiednie sterowniki do systemu zarządzania bazą danych potrafią wybrane informacje umieścić na witrynach *www*.

12.2. Serwery *www*

Jeżeli chcemy udostępnić innym użytkownikom jakieś dane, niezależnie od tego czy będą to dokumenty w formacie HTML, czy informacje zawarte w bazach danych, inne pliki, teksty, obrazki czy animacje, możemy skorzystać z usługi serwera *www*. Jest oprogramowanie, które sprawia, że osoba korzystająca z tej usługi może skorzystać z udostępnionych zasobów, nie wnikając w szczegóły dotyczące ich lokalizacji i metod dostępu do nich. Jedyną istotną kwestią jest uwzględnienie ich w zasobach przez administratora. Można stwierdzić, że serwer *www* jest przezroczysty, to znaczy, że bez względu na to skąd się łączymy nie musimy dokładnie wiedzieć gdzie znajdują się dane. Użytkownik pragnąc skorzystać z zasobów znajdujących się na serwerze generuje żądanie przesłania danych. Serwer odbiera zgłoszenie przeglądarki, przesyła żądany plik i kończy połączenie. Istotną zaletą jest fakt, że pragnąc skorzystać z jakiegoś dokumentu w formacie HTML osobno obsługiwane jest zgłoszenie dotyczące pliku tekstowego, a osobno poszczególnych plików z grafiką, dla których otwierane są niezależne połączenia między przeglądarką i serwerem. Przed transmisją danych do przeglądarki zostaje wysłany znacznik typu danych. Znacznik w języku HTML to polecenie będące specjalnym ciągiem znaków objętym nawiasami ostrymi. Po identy-

fikacji przeglądarka wie, jak wczytać interpreter typu danych. Jeżeli chcemy wysłać do przeglądarki dokument zawierający grafiki w formatach JPG i GIF (dwa podstawowe formaty bitowych plików graficznych), to serwer wysyła znaczniki, na podstawie których przeglądarka wczytuje i uruchamia odpowiednie interpretery formatu.

Aby wybrać odpowiedni serwer *www*, należy zbadać potrzeby i oczekiwania oraz zapoznać się z istniejącą strukturą informatyczną. Ważne jest oszacowanie ruchu, ilości odwołań, rozmiaru i zakresu ośrodka udostępniającego witryny. Poza tym o wyborze konkretnego rozwiązania powinny decydować osoby konfigurujące i modyfikujące zawartość *www* i osoby zarządzające siecią. Bardzo ważną cechą serwera jest połączenie z bazami danych. Jeśli ściąga się dane z systemów zarządzania bazą danych, to trzeba brać pod uwagę typy połączeń serwera *www* z zewnętrzną bazą danych. Najbardziej popularną metodą jest protokół ODBC (ang. *Open Database Conectivity*), jednak niektóre firmy proponują własne rozwiązania. Na przykład produkt Web Side Professional zawiera narzędzie do projektowania aplikacji umożliwiające zakodowanie komend dostępu do danych na stronach HTML. Gdy taka strona zostanie wywołana, serwer *www* wykonuje te komendy, ściąga dane i wkleja je na stronę.

12.3. Dane na stronach *www*

12.3.1. Wprowadzenie

Twórcy dzisiejszego oprogramowania prześcigają się w tworzeniu i dodawaniu nowych narzędzi czy obiektów dla uatrakcyjnienia serwisu *www*. Ostatnio kluczową sprawą jest udostępnienie informacji bezpośrednio z bazy danych, jak również możliwość ingerencji bezpośrednio w strukturę bazy za pomocą jednej z popularnych przeglądarek. Projektanci baz danych, serwerów *www* znaleźli kilka sposobów na dynamiczne tworzenie serwisu prezentującego interesujące bazy danych. Zasady projektowania interfejsu do baz danych są proste. Najnowsze standardy języka HTML umożliwiają umieszczenie w dokumencie elementów aktywnych, dzięki którym użytkownik za pośrednictwem przeglądarki stron *www* może oprócz przeglądania własnoręcznie wprowadzić dane lub wykonać zapytania do bazy danych. Informację zwrotną otrzymuje on dzięki specjalnym mechanizmom. Podstawowy mechanizm współpracy między przeglądarką, serwerem *www* a serwerem bazy danych można zawrzeć w czterech krokach:

1. Poprzez przeglądarkę generowane jest zapytanie klienta.
2. Serwer *www* przekazuje zapytanie do bazy danych.
3. Baza danych przetwarza zapytanie i udziela odpowiedzi.
4. Odpowiedź przetworzona zostaje na HTML i przekazana do przeglądarki.

Taka specyfika współpracy przeglądarki i bazy danych nie jest korzystna dlatego, że nie ma trwałego połączenia między przeglądarką a serwerem. Nie możemy w tym

wypadku mówić o bezpośredniej transakcji wykonywanej na bazie danych. Teraz informacje o stanie transakcji są przekazywane poprzez dynamicznie generowane odnośniki (ang. *links*) lub też wykorzystuje się właściwości dynamiczne tworzonych formularzy. Dostępny jest również specjalny mechanizm umożliwiający przekazywanie odpowiednich wartości podczas komunikacji między przeglądarką a serwerem.

Metody statycznego udostępniania informacji tekstowej na internetowych witrynach *www* należą do najprostszych. Początkowo kreatorzy stron kopiowali zawartość baz danych i umieszczali je na stronach jako zwykły tekst. Gdy po pewnym czasie należało dokonać aktualizacji „webmaster", opiekujący się serwisem wymieniał całą stronę na serwerze. Było to zajęcie bardzo czasochłonne. Firmy świadczące usługę prezentacji informacji ekonomicznych czasami kilkanaście razy dziennie zmieniały zawartość swoich serwerów. Z pomocą przyszło rozwiązanie dynamicznie aktualizowanych stron, na których informacja odzwierciedla stan serwera podłączonej bazy danych. Jest ona na bieżąco modyfikowana, a wyświetlane informacje oddają stan momentu żądania usługi przez klienta.

12.3.2. Metody dynamicznego udostępniania danych w *www*

Użytkownicy zaczęli stawiać coraz większe wymagania dotyczące serwerów *www*. Bardzo ważną kwestią było utrzymywanie aktualności danych. Tak więc zaczęto tworzyć specjalne dodatki do standardowego języka HTML, umożliwiające wykonywanie różnych poleceń systemowych, tworzenie interaktywnej grafiki, czy bezpośredniego odwoływania się do baz danych. Powstało wiele standardów i nowych technologii wykorzystywania specjalnych funkcji, jak: przetwarzanie skryptów, wykorzystywanie języków wysokiego poziomu itp. Pojawienie się nowych technologii postawiło projektantów oprogramowania oraz projektantów serwisów internetowych i intranetowych przed problemem wyboru tej najwłaściwszej. Przed zarządcami systemów stanęły problemy zapewnienia właściwego poziomu wydajności i ochrony zasobów znajdujących się w sieci.

API (ang. *Application Programing Interface*) to zestaw funkcji stworzonych dla serwera *www*. Obecnie na rynku liczą się dwie specyfikacje: dla serwerów Netscape NSAPI oraz ISAPI dla oprogramowania Microsoft. Funkcje API pełnią przeróżne zadania na serwerze i umożliwiają obsługę odwołań do jego zasobów, do zasobów baz danych itp. Funkcje te tworzy się od podstaw, aby zapewnić jak największą wydajność. Tworzenie aplikacji w API wymaga specjalistycznych technik programistycznych, jak wielowątkowość, synchronizacja procesów, bezpośrednie programowanie protokołu i obsługa błędów. Wielu producentów dostarcza gotowe rozwiązania w postaci bibliotek *.dll. Metoda API zwana jest metodą niskiego poziomu.

Do metod wysokiego poziomu należy metoda SSI (ang. *Serwer Side Includes*). Jest to chyba najstarsza metoda dynamicznego udostępniania informacji z baz danych na stronach *www*. Polega ona na zapisywaniu komend SSI bezpośrednio w dokumencie

HTML. Pliki z takimi komendami były odpowiednio identyfikowane rozszerzeniem *.ssi. Umożliwiało to serwerowi *www* wcześniejsze przetworzenie takiego dokumentu, a potem zrealizowanie usługi. Gdy serwer *www* nie posiadał odpowiednich mechanizmów, dodatkowe komendy były ignorowane, a dokument trafiał do klienta w wersji nieprzetworzonej.

Microsoft od początku powstania mocno reklamował swoje rozwiązanie Active X jako najlepsze narzędzie do projektowania aplikacji internetowych. Początek wszystkiemu dał standard DDE (ang. *Dynamic Data Exchange*) zaprojektowany w celu udostępnienia aplikacjom możliwości wymiany danych. Kolejnym krokiem było OLE (ang. *Object Linking and Embedding*), umożliwiające traktowanie dokumentów utworzonych przez jedną aplikację jako zasobnika dla obiektu utworzonego przez inną, np. dokument Word może zawierać tabelę utworzoną w Excel. Zagnieżdżenie to może polegać na włączeniu całego obiektu lub tylko jego łącznika. Kiedy wzrosło zainteresowanie Internetem i Intranetem, technologia OLE Documents przekształciła się w Active X Documents. Pierwotna koncepcja Active X usystematyzowała się i została poszerzona o dodatkowe rozwiązania:

– szkielet serwera Active X (ang. *Internet Serwer API*) opisujący, w jaki sposób serwer *www* powinien komunikować się z aplikacjami wspierającymi, które miedzy innymi pozwalają filtrować i modyfikować dokumenty,

– sterowniki internetowe Active X zapewniające usługi przeglądarek internetowych innych klientów oraz transfer plików w formie modułów, co pozwala łatwo włączać je do aplikacji,

– Active Scripts sterowniki Active X wykorzystujące Java Script, Visual Basic Script, C/C++ oraz inne języki opisowe,

– Code Security Services zapewniające integralność po stronie klienta w odniesieniu do obiektów Active X sprowadzanych ze źródeł nie mających statusu bezpieczeństwa.

Interfejs CGI (ang. *Common Gateway Interface*) każdorazowo podczas zgłoszenia użytkownika powoduje uruchomienie na serwerze odpowiedniego programu, który połączy się z bazą danych, przekaże jej zapytanie, a wynik, który otrzyma przetworzy na format HTML i odeśle do serwera. Zaletą tego rozwiązania jest jego prostota, niestety na jego niekorzyść świadczą słaba integracja z serwerem i niska wydajność. Ta ostatnia cecha wynika z konieczności uruchamiania odrębnego procesu dla każdego zgłoszenia przychodzącego z sieci. Fakt, iż program CGI jest każdorazowo niezależnym wykonywalnym procesem sprawia, że informacja o aktualnym stanie serwera jest dosyć ograniczona. Nie zaleca się stosowania go w przypadku wykonywania skomplikowanych wieloetapowych transakcji. Powstał już zresztą nowy standard Fast CGI. Przy zastosowaniu Fast CGI serwer *www* uruchamiający na żądanie proces obsługi danych z zewnętrznego źródła utrzymuje go w gotowości do obsługi następnych zgłoszeń, dzięki czemu nie ma problemów z uruchamianymi każdorazowo nowymi procesami.

ASP (ang. *Active Serwer Pages*) jest to koncepcja budowania dynamicznych stron *www* opracowana przez firmę Microsoft. Pozwala ona w tekstowych plikach HTML umieszczać wstawki kodu lub całe programy napisane w Visual Basic Script czy Java Script. Bardzo użyteczna jest możliwość poszerzania tworzonych dokumentów o obiekty baz danych i obiekty zapewniające zapamiętanie transakcji klienta. Przygotowana witryna *www* jest w rzeczywistości wzorcem strony, w którym przed wysłaniem do klienta zostaną umieszczone odpowiednio sformatowane dane pochodzące z systemu współpracujących baz danych. Rozwiązania ASP umożliwiają szybkie połączenie z bazą, bardzo szybkie tworzenie profesjonalnych aplikacji *www*, zarówno internetowych jaki intranetowych. Oprócz podstawowych operacji wykonywanych na bazach, takich jak otwieranie i zamykanie połączeń, wykonywanie zadań SQL, dostępne są także bardziej zaawansowane mechanizmy. Należy do nich obsługa transakcji na kilku połączeniach z danym klientem lub korzystanie z procedur wbudowanych.

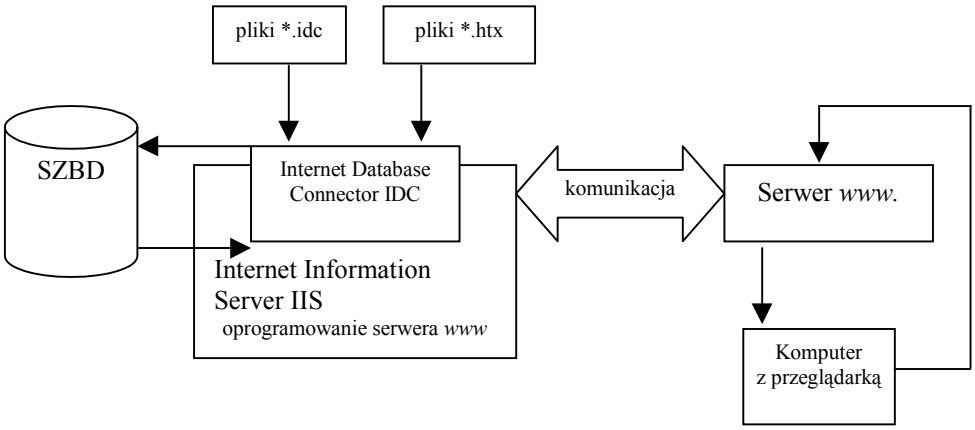
Bardzo dużo ciekawych rozwiązań przy łączeniu aplikacji baz danych ze stronami *www* daje możliwość zastosowania języka Java. Aplikacja Javy może działać niezależnie od serwera. Nie korzysta ona z API serwera. Użytkownicy, wykorzystując przeglądarkę przystosowaną do interpretacji Javy lokalnie przetwarzają aplety zawarte w dokumentach HTML na serwerze. Aplety są plikami niewielkimi więc ich transfer nie obciąża sieci. Wykorzystując aplety Javy do podpinania baz danych łatwo jest zauważyć dużą elastyczność tego języka. Na przykład podczas tworzenia formularza jaki chcemy wypełnić danymi, które następnie zapiszą się do bazy danych na serwerze, wystarczy wykorzystać stworzony do tego celu aplet. Nie ma w tym wypadku konieczności tworzenia dynamicznej strony HTML. Takie napisane przez nas aplety mogą działać zarówno po stronie klienta, czyli przeglądarki, jak również po stronie serwera. Do komunikacji z bazą danych stosuje się specjalny protokół JDBC. W odróżnieniu od produktów Microsoft daje on większą niezależność od platformy sprzętowej i systemowej. Biblioteka JDBC API jest to zbiór klas, które umożliwiają połączenia z bazami danych, wykonują instrukcje SQL i przetwarzają wyniki.

Jeszcze jedno rozwiązanie to narzędzia wizualne. Jest to grupa gotowych programów, które umożliwiają graficzne zaprojektowanie wyglądu stron *www*, schematów baz danych, formatów raportów czy formularzy, na podstawie których jest generowany następnie odpowiedni kod aplikacji klienta czy serwera. Narzędzia wizualne są bardzo praktyczne. Ich projektanci zadbali o to, aby końcowy użytkownik nie miał problemów z różnymi formatami danych oraz z bardziej złożonymi schematami danych, jak na przykład złączenia tabel.

12.3.3. Metody dostępu do baz danych

Narzędzia do połączeń na stronach *www* firma Microsoft oferuje użytkownikom pracującym na platformie Windows NT i korzystającym z bazy danych SQL Serwer i serwera *www*. Oprogramowanie to tworzy jedną całość SQL Internet Connector. Zasadniczą częścią tych aplikacji jest Web Assistant, narzędzie ułatwiające tworzenie dynamicznych stron w HTML. Jest bardzo pomocne podczas pobierania specyficznych danych z serwera bazy danych i używania ich do tworzenia strony w HTML. Dane można uzyskać kilkoma sposobami. Możemy wybierać kolumny z poszczególnych tabel, stosować swobodna formę wyrażeń SQL lub używać zapamiętane procedury dostępu do bazy danych. Używanie takich kryteriów jak data, czas lub fakt modyfikacji danych, umożliwia określenie, kiedy należy uaktualniać strony *www* informacjami z bazy danych. Zaletą takiego rozwiązania jest to, że użytkownicy nie zatrudniają bazy danych w czasie rzeczywistym, lecz uzyskują dostęp do statycznych stron uaktualnianych dynamicznie co pewien czas. Niewątpliwą wadą jest brak możliwości tworzenia dynamicznych stron ad hoc. W tym wypadku stosuje się techniki wykorzystujące ASP i kontrolki Active X.

Inną bardzo popularną metodą udostępniania baz danych na witrynach *www* jest wykorzystanie produktu IDC (ang. *Internet Database Connector*).



Rys. 12.1. Sposób korzystania z IDC

Jest to rozszerzenie Internet Information Serwer wykorzystujące API tego serwera. IDC jest częścią, a konkretnie jedną z bibliotek *.dll (httpodbc.dll) ładowanych do serwera *www*. Właśnie dlatego wszystkie funkcje zawarte w IDC mogą korzystać z zasobów serwera *www*. Aby uzyskać możliwość komunikacji z bazą danych, należy zainstalować odpowiedni sterownik ODBC. IDC wykorzystuje dwa rodzaje plików: *.idc – zawierają informacje o sposobie połączenia z bazą, o instrukcjach SQL

i wszystkich informacjach niezbędnych do wykonania operacji na bazie oraz pliki *.htx – zawierajace informacje w jaki sposób formatować dane uzyskane po wykonaniu plików *.idc.

Firma Oracle oferuje oprogramowanie Web Request Broker będące częścią pakietu Web Server dostarczonego w Oracle 7 Serwer 7.3. Jej produkty obsługują przetwarzanie transakcji on-line i pozwalają na utrzymanie trwałej sesji miedzy przeglądarkami, serwerami *www* i serwerami baz danych. Rozwiązanie to pomija mechanizmy CGI, co powoduje uzyskanie większej kontroli nad procesami generowanymi przez aplikację, obsługę zarządzania jednocześnie wieloma połączeniami do bazy, co z kolei znacznie poprawia wydajność. Dla programistów dostarczany jest WRB API służący do projektowania pakietów aplikacyjnych. Dodatkowo firma opracowała narzędzie projektowe Developer 2000 i Designer 2000. Ich przeznaczeniem jest projektowanie aplikacji internetowych dla zaawansowanych projektantów.

Produktem firmy Informix jest Live Wire Pro, który stanowi część oferty Suite Spot pakietu Online Workgroup Serwer. Produkty Informix udostępniają w ramach tych pakietów oprogramowanie firmy Netscape. Informix dostarcza kilku opcji pozwalających zamieszczać dane z baz na stronach *www*. Najprostszy Web Interface Kit ustanawia proste połączenie *www* – baza danych wykorzystując skrypty CGI. Kolejnym produktem jest Web Connectivity Framework pozwalający ośrodkom *www* utrzymywać informacje o stanie sesji w sposób analogiczny do aplikacji baz danych. Bardzo interesująca jest propozycja Web Data Blade. Jest to środowisko projektowania modułów i aplikacji baz danych oparte na *www*. Stosuje się tutaj technologię obiektową w *www*. Scala ona wszystkie funkcje dostępu do danych i indeksacji. Umożliwia to pamiętanie takich złożonych typów danych jak video, głos, obrazy łącznie z tekstem i innymi typami danych multimedialnych.

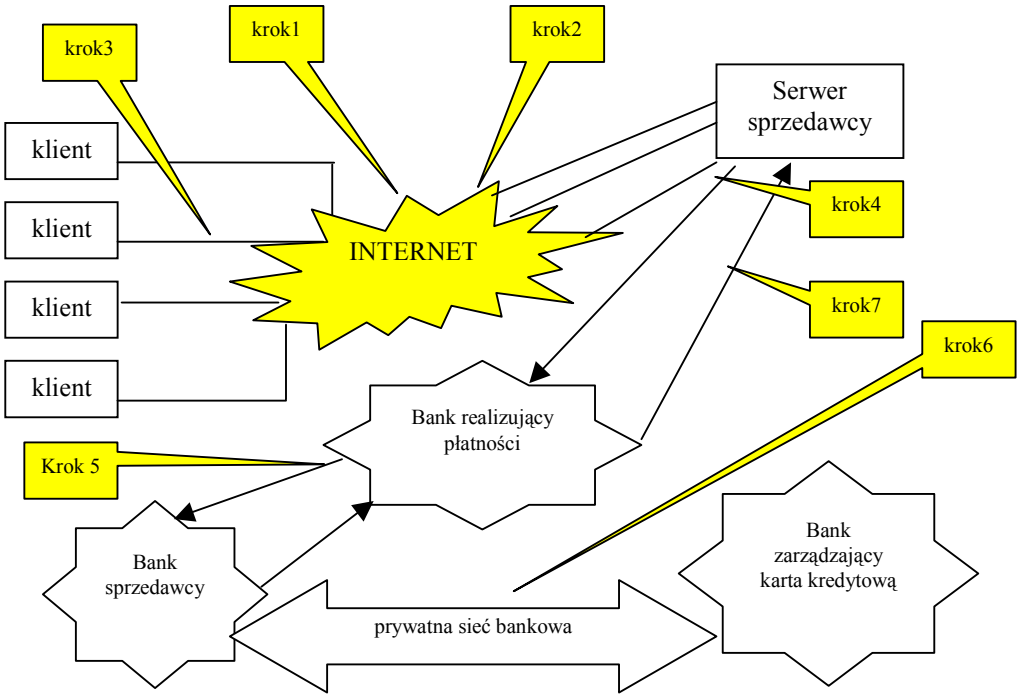
Firma IBM dołączyła możliwości zarządzania złożonymi typami danych do swojego produktu DB2. Dotychczas używano NetDate, który umożliwiał przekształcanie statycznych stron HTML na dynamiczne aplikacje za pomocą zestawu makr. Wykorzystano tu technikę skryptów CGI. NetDate współpracuje z bazą DB2 w sposób bezpośredni, a z bazami innych producentów za pośrednictwem sterownika ODBC.

W Sybase programy scalające pracę przeglądarek, serwerów *www* i baz danych to Web SQL i Object Connect. Pozwalają one na wpisywanie instrukcji takich jak wyrażenia SQL i skrypty Perl. Umożliwiają również generowanie stron mających indywidualizowaną zawartość opartą na szablonach użytkownika i jego preferencjach.

12.4. Serwery handlu elektronicznego

Najbardziej dziś zaawansowanymi aplikacjami wykorzystującymi opisywane serwery *www*, bazy danych, metody współpracy między nimi i udostępnienie informacji dynamicznie na stronach *www* są systemy handlu elektronicznego. Zasadniczą częścią

jest w nich serwer handlowy. Opiera się on na serwerach *www* poszerzonych o skrypty back end, obsługujące prezentację katalogów towarów i procesy sprzedaży. Oczywiście potrzebna jest odpowiednia baza danych SQL. Może ona pracować na serwerze handlu, ale bardzo często umieszcza się ją osobno, poprawiając bezpieczeństwo rozwiązania. Wiele serwerów handlu elektronicznego obsługuje sprzedaż anonimową i personalizowaną. W pierwszym przypadku klient identyfikowany jest w momencie składania zamówienia, w drugim rejestruje się jednorazowo w bazie klientów, a potem identyfikuje się go podczas wejścia na serwer. Istotną sprawą jest elektroniczny koszyk sklepowy. W istocie jest to również baza danych pozycji wybranych przez klienta. Ważne jest należyte skonfigurowanie takiego koszyka, a zależy ono od wielu czynników. Proces płatności w handlu elektronicznym można opisać w ośmiu krokach (rys. 12.2):



Rys. 12.2. Płatności w handlu elektronicznym

Krok 1: nabywca odwiedzając sklep on-line wybiera interesujące go propozycje.

Krok 2: serwer wysyła do klienta formularz z opisami produktu, ceną, formą płatności, numerem transakcji.

Krok 3: klient wybiera formę zapłaty. Specjalne oprogramowanie klienta uruchamia aplikację płatniczą. Dane zostają zaszyfrowane i odesłane na serwer.

Krok 4: zaszyfrowane dane klienta serwer odsyła do specjalnego banku realizującego płatności.

Krok 5: serwer realizujący płatności wysyła dane klienta do banku wskazanego przez sprzedawcę.

Krok 6: bank sprzedawcy potwierdza ważność karty kredytowej (komunikacja z instytucją zarządzającą kartami) i informuje o tym fakcie bank realizujący płatności.

Krok 7: bank realizujący płatności przekazuje otrzymane dane do serwera sprzedawcy.

Krok 8: serwer sprzedawcy finalizuje transakcję i powiadamia nabywcę.

Istotne są oczywiście sprawy ochrony. Jeżeli serwer handlowy zamierza prowadzić transakcje finansowe, to bardzo ważną sprawą jest utrzymanie poufności transakcji (wykorzystanie protokołów SSL lub Secure HTML). Oczywiście powinien on być dodatkowo zabezpieczony. Na rynku dominują trzy wiodące rozwiązania: Net Commerce firmy IBM, Site Server firmy Microsoft oraz Domino Merchant Lotusa.

Polityka bezpieczeństwa jest sprawą, której obecnie poświęca się coraz więcej uwagi [15]. Firmy dbają o tajność własnej informacji. Problem ten jest o wiele szerszy niż tylko obrona przed internetowymi hackerami. Z sondaży wynika, że najwięcej szkody czynią niekompetentni i niezdyscyplinowani pracownicy. Pracowników należy włączać do odpowiednio zaprojektowanego systemu bezpieczeństwa poprzez nadawanie im praw dostępu stosownie do zajmowanych pozycji. Ważne jest, aby zaplanować odpowiednio strategię bezpieczeństwa. Plany te powinny uwzględniać: propozycje fizycznego zabezpieczenia zasobów firmy, opis realizacji metod kontroli dostępu do systemu i jego zasobów, opis metod monitorowania systemu, reakcji na wykrycie zagrożenia lub próby włamania, dokładną implementację procedur naprawiających szkody wywołane przez niepowołanych użytkowników, czy wreszcie opis tworzenia zabezpieczeń i kopii zapasowych.

12.5. Uwagi końcowe

Możliwość wykorzystania bogatych zasobów wiedzy z najróżnorodniejszych dziedzin znajdujących się w sieci Internet [11] pozwala na stworzenie aplikacji prostych, łatwych w obsłudze i funkcjonalnych. Aplikacje takie dysponują następującymi mechanizmami tak charakterystycznymi dla tego typu oprogramowania:

– odpowiednim połączeniem z bazą danych,
– narzędziami dla analiz i raportów,
– odpowiednią prezentacją danych (koszyk internetowy),
– rejestracją użytkowników,
– prostą rozszerzalnością oferty.

Oczywiście profesjonalne oprogramowanie pozwala tworzyć własne serwery handlu elektronicznego dysponujące wieloma dodatkowymi narzędziami i usługami. Zapewniają one oprócz charakterystycznych opcji związanych ze sprzedażą usług czy towarów także ochronę połączeń i aspekty bezpieczeństwa przechowywanych danych. Ważne jest, aby zapewnić w należyty sposób ochronę danych serwera.

PODSUMOWANIE

W części IV poruszono dwie istotne sprawy dla dalszego rozwoju baz danych, a mianowicie problematykę rozproszonych baz danych i możliwości sieci Internet w dziedzinie baz danych. Jeżeli chodzi o projektowanie baz rozproszonych, to w pewnym stopniu możemy traktować to jako pewien sposób projektowania bazy na poziomie fizycznym. Pierwszym etapem projektowania jest utworzenie modelu scentralizowanej bazy, a następnie podjęcie decyzji, jak podzielić i zreplikować utworzony schemat logiczny. Problem polega na tym, aby umieścić dane blisko miejsca gdzie są używane oraz zmniejszyć ilość danych przesyłanych w sieci. Fragmentacja i replikacja danych jest jednym z najważniejszych zadań strategii rozproszenia [20, 22]. Z rozproszonymi bazami danych ściśle związana jest możliwość wykorzystania sieci Internet w dziedzinie baz danych. W obu przypadkach powinna istnieć możliwość korzystania z takich użytecznych cech, jak:

– możliwość umieszczenia serwera aplikacji na dowolnym systemie w sieci,

– możliwość automatycznego, równomiernego rozłożenia obciążenia między kilka serwerów realizujących usługi tego samego typu,

– możliwość działania wieloaplikacyjnych serwerów na jednej lub wielu maszynach,

– zatajenie prawdziwej lokalizacji serwera,

– możliwość wykorzystania niejednorodnych, heterogenicznych baz danych,

– prostota dodawania nowych i modyfikacji istniejących części systemu.

Architektura tego typu istnieje na rynku od lat. Jest ona realizowana w dwojaki sposób. Pierwszy sposób to utrzymanie tendencji dotychczas panującej reprezentowanej przez takie firmy jak Microsoft i Intel. Ich wizja sieci to niezależne stacje robocze PC wyposażone w twarde dyski, na których rezyduje system. Drugi sposób to koncepcja zastąpienia komputerów PC komputerami sieciowymi. Nazywa się je *odchudzonymi klientami*. Urządzenia te przechowują wszystkie dane w swojej pamięci RAM, a podstawowymi wymaganiami jakie muszą spełniać to zawierać środowisko wykonawcze Javy, obsługiwać aplikacje niezależne i rezydujące w sieci oraz wykazywać się niezależnością od rodzaju sieci. Do grupy tej należą Netscape, Oracle, Sun Microsystem.

ZAKOŃCZENIE

Dziedzina baz danych ulega stałym rozszerzeniom w celu obsłużenia coraz bardziej i bardziej złożonych obszarów zastosowań. W ten sposób systemy baz danych zajęły znaczące miejsce w konstrukcji systemów informacyjnych. Przykładem są takie systemy informacyjne jak hipermedia [5] oraz geograficzne systemy informacyjne [8]. Tradycyjne systemy baz danych były projektowane z myślą o przechowywaniu dużej ilości strukturalnych danych. Struktura umożliwiała specyfikację formalnych zapytań i wyszukiwanie informacji. W przypadku systemów hipermedialnych mamy do czynienia z przechowywaniem małych ilości różnych rodzajów mediów połączonych siecią asocjacyjnych powiązań. W związku z brakiem wewnętrznej struktury układanie zapytań w tych systemach jest trudne do zrealizowania. Również geograficzne systemy informacyjne różnią się od tradycyjnych baz danych. Mamy tu do czynienia z modelowaniem i analizą danych przestrzennych. Do zarządzania przestrzenią danych nadaje się zarówno opisany w części II model obiektowy jak i dedukcyjny.

Ogólnie uważa się [1, 22], że nowe kierunki rozwoju baz danych, które prawdopodobnie odegrają istotną rolę w przyszłości to:

– równoległość, czyli zastosowanie technologii komputerów równoległych do zarządzania bazami danych; metoda ta, stosowana od lat, udowodniła swoją skuteczność polepszając wydajność,

– inteligencja, czyli sztuczna inteligencja i jej implementacja w kontekście baz danych; szczególnie chodzi tu o bazy dedukcyjne,

– złożoność, czyli problem użycia baz danych w złożonych sytuacjach aplikacyjnych.

BIBLIOGRAFIA

[1] Beynon-Davies P., *Systemy baz danych*, WNT, Warszawa 1998.

[2] Berry B., Chase D., Delson B., Francis J., O Neill P., Sherwood J., *Informix – Universal Serwer*. Administrations Guide. Published by Informatix Press 1997.

[3] Coulouris G., Dillimore J., Kindberg T., *Systemy rozproszone. Podstawy i projektowanie*, WNT, Warszawa 1998.

[4] Daniell B., Leland J., Maneval D., *Informix – Universal Server*. DataBlade API. Programmers Manual. Published by Informix Press 1997.

[5] Darrel R.R., Tampa F.W.M., *Hypertext and the Oxford English Dictionary*, CACM, July, 1988, 31(7), s. 871–879.

[6] Date C., *Introduction to Datebase Systems 5th Ed*. Addison-Wesley 1990.

[7] Garms J., *Windows NT 4.0 Serwer*. Kompendium Robomatic, Wrocław 1997.

[8] Gatrell A.C., *Concepts of Space and Geographical Data*, 1999.

[9] Halilovic I., Gales Ch., *DataBlade Module Development Overview*, Published by Informix Press 1997.

[10] Hall C.L., *Techniczne podstawy systemów klient–serwer*, WNT, Warszawa 1996.

[11] Informacje zawarte na witrynach www. 1999/2000.

[12] Kline M., Litzell Ch., Schackell J., Landshoff D., Bostrom K., Nowacki B., Howard D., *Blade Manager*. Published by Informix Press 1997.

[13] McCarthy D.R., Dayal U., *The Architecture of an Active Data Base*, Management Systems Proc.SIGMOD 1989.

[14] Myer B., *Object Oriented Software Construction*, New York, Prentice-Hall 1997.

[15] Net World: Polityka bezpieczeństwa i strategie jej realizacji 2/98,
Serwery handlu elektronicznego 2/98,
Narzędzia realizacji polityki bezpieczeństwa 3/1998.

[16] Prace: Jankowski S., *Obiektowo-zorientowany system zarządzania pracą przedsiębiorstwa*, dyplom pod kierunkiem M. Chałon ICT PWr., Wrocław 1998.
Bielecki K., *Architektura rozproszonych systemów baz danych*, dyplom pod kierunkiem M. Chałon ICT PWr., Wrocław 1999.

[17] Rose J., Marshall T., *The Little Black Book: Mail Bonding with OSI Directory Serv*ices. Englewood Cliffs, NJ, Prentice-Hall 1992.

[18] Rochkind M., *Programowanie w systemie Unix dla zaawansowanych*, WNT, Warszawa 1993.

[19] Stevens W., *Programowanie zastosowań sieciowych w systemie Unix*, WNT, Warszawa 1996.

[20] Teorey T.J., *Database Modelling and Design: the Fundamental Principles*. 2nd Ed. San Mateo, Calif. Morgan Kaufmann 1994.

[21] *The Advanced Network System Architecture (ANSA)*, Reference Manual, Castle Hill, Cambridge England, Architecture Project Management, 1989.

[22] Ullman J.D., Widom J., *Podstawowy wykład z systemów baz danych*, WNT, 2000.

Spis rzeczy