Iwona Poźniak-Koszałka

Relacyjne bazy danych w środowisku Sybase

Modelowanie, projektowanie, aplikacje

Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2004

Recenzenci

Włodzimierz STANISŁAWSKI
Aleksander ZGRZYWA

Opracowanie redakcyjne
Aleksandra WAWRZYNKOWSKA

Korekta
Hanna JUREK

Projekt okładki
Aleksandra POŹNIAK

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISBN 83-7085-790-6

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr. 384/2004.

Spis rzeczy

Przedmowa

Trudno dzisiaj znaleźć dziedzinę życia, czy sferę działania człowieka, która nie wspierałaby się na coraz bardziej wyrafinowanych technikach i środkach informatyki. Dane, bazy danych, systemy bazodanowe – są to obecnie pojęcia egzystujące nie tylko w świecie informatyki i związanym z nim wąskim gronem specjalistów. W dobie powszechnej informatyzacji większość „zwykłych śmiertelników" przynajmniej otarła się o te pojęcia i niemal każdy podkłada pod nie swoje własne treści, kierując się intuicją. Wszak mówi się dzisiaj o „społeczeństwach informacyjnych", czy więc można powiedzieć, że dane to informacje, a informacje to wiedza i wreszcie, czy wiedza to mądrość? Otóż z całą pewnością nie jest to tak oczywiste i proste. Dobrze jest mieć zgromadzone informacje – im więcej, tym (teoretycznie) lepiej, ale równie ważny jest problem umiejętnego korzystania ze zgromadzonych zasobów, czyli interpretacji danych. W dzisiejszych czasach możliwości gromadzenia danych są praktycznie nieograniczone; pozwalają na to techniki i narzędzia informatyczne, często nawet z większym potencjałem niż potrzeby użytkowników. Wydaje się jednak, że sedno sprawy leży nie w technikach i technologii, ale w zrozumieniu istoty gromadzenia i korzystania z informacji. Dlatego też, chcąc sprostać wymaganiom i wyzwaniom współczesności, proces poznawania drogi od danych do wiedzy najlepiej rozpocząć *ad ovo* – od zrozumienia podstawy, czyli fundamentu, jakim jest baza danych, na tym fundamencie bowiem lokowane są rozwiązania umożliwiające funkcjonowanie w dzisiejszym świecie – zarówno w sektorze gospodarczym, jak i administracyjnym, bankowym czy medialnym. Można, ocierając się wprawdzie o banał i zarzut patetyczności, niemniej jednak nie mijając się z prawdą, stwierdzić, że bazy danych są „sercem" systemów informatycznych wspierających planowanie, podejmowanie decyzji, zarządzanie zasobami, jak również wszechobecnego Internetu. Z tych powodów celem niniejszego podręcznika, który powstał na podstawie konspektów do wykładów „Bazy danych" oraz „Systemy informatyczne", prowadzonych dla kilku specjalności na Wydziale Elektroniki Politechniki Wrocławskiej, jest przybliżenie zagadnień związanych z bazami danych, a ściślej relacyjnymi bazami danych, faktycznym liderem na rynku rozwiązań komercyjnych.

Książka składa się dziesięciu rozdziałów. W pierwszym, stanowiącym wprowadzenie do zagadnienia baz danych, zaprezentowano teorię leżącą u podstaw relacyj-

nych baz danych oraz wprowadzono podstawowe pojęcia i definicje dotyczące zarówno baz danych, jak i systemów zarządzania bazami danych. Rozdział drugi zawiera charakterystykę środowiska i technologii Sybase, do którego odnoszą się treści kolejnych rozdziałów. Rozdziały trzeci i czwarty poświęcone są językowi komunikacji z bazami danych, czyli językowi SQL. W rozdziałach tych przedstawiono zarówno standardy języka SQL, jak i dialekt Sybase. Kolejne rozdziały (rozdz. 5, 6, 7) dotyczą zagadnień związanych z projektowaniem zarówno baz danych, jak i aplikacji bazodanowych. Na przykładach praktycznych przedstawiono metody i narzędzia realizacji poszczególnych faz cyklu życia systemów informatycznych. Omawiane przykłady zostały zrealizowane za pomocą narzędzi CASE oferowanych przez firmę Sybase. Rozdział ósmy poświęcony jest zagadnieniu normalizacji relacji – zarówno w kontekście metodyki projektowania schematów baz danych, jak i walidacji modelu. Rozdział dziewiąty dotyczy projektowania warstwy fizycznej bazy danych oraz podejmowania decyzji optymalizacyjnych, zapewniających odpowiednią efektywność systemu. W ostatnim rozdziale omówiono metodykę konstruowania aplikacji za pomocą PowerBuildera – narzędzia typu RAD, również pochodzącego z rodziny produktów Sybase.

Myślą przewodnią, jaka towarzyszyła mi przy pisaniu tej książki, była stara, klasyczna maksyma *Praeceptor bonus... Omni modo fiat nobis amicus nec officium in docendo spectet sed affectum* (*Dobry nauczyciel niechaj wszystkimi sposobami stara się być przyjacielem i w nauczaniu niech nie widzi obowiązku, lecz uczucie*, Quintilianus Institutio Oratoria II, I, 1–15), dlatego też mam nadzieję, że adresaci podręcznika, jakimi są przede wszystkim studenci informatyki, ocenią go pozytywnie.

Szczególne wyrazy podziękowania chciałabym złożyć Profesorowi Andrzejowi Kasprzakowi za inspirację, skuteczną mobilizację i wsparcie w chwilach zwątpienia.

Dziękuję również mojej rodzinie za to, że wiarą i zaufaniem wspierała mnie podczas pracy, a mojej córce Aleksandrze specjalnie dziękuję za pracę nad projektem graficznym okładki, który, mam nadzieję, docenią czytelnicy.

Wrocław, 2004

Iwona Poźniak-Koszałka

1 Wprowadzenie

Zagadnienia rozważane w niniejszej książce są związane z administrowaniem, zarządzaniem i projektowaniem relacyjnych baz danych, które są obecnie faktycznym standardem stosowanym w systemach informatycznych, oraz projektowaniem i implementacją aplikacji baz danych. Aplikacje korzystające z baz danych są obecnie podstawą systemów wspomagających działanie właściwie wszystkich sektorów rynku – począwszy od sektora administracji państwowej, sektora finansowego, poprzez sektor telekomunikacyjny, sektor mediów, czy edukacyjny. Firmy produkujące oprogramowanie wprowadzają na rynek nowe narzędzia i technologie, mające nie tylko ułatwić zarządzanie coraz większą ilością danych, ale i przyspieszyć prace związane z projektowaniem i implementacją systemów baz danych (narzędzia CASE (*Computer Aided Software Engineering*), narzędzia RAD (*Rapid Application Development*)). Do czołowych producentów zintegrowanych zestawów produktów do projektowania i udostępniania danych zalicza się firmy: Oracle Corporation, IBM, Microsoft, Sybase.

Dla zilustrowania praktycznych aspektów związanych z dziedziną relacyjnych baz danych, tzn. drogi od modelu danych do aplikacji, wykorzystano technologię i narzędzia firmy Sybase (SQL Anywhere Studio, PowerDesigner, PowerBuilder), zresztą nieprzypadkowo. Firma jest stosunkowo mocno osadzona na polskim rynku informatycznym, jej produkty znalazły zastosowanie m.in. w administracji państwowej i samorządowej (Polska Administracja Celna), w bankowości (Narodowy Bank Polski), w oświacie, na rynku mediów (spółka wydawnicza Agora), czy w telekomunikacji (firma Polkomtel – operator sieci cyfrowej telefonii komórkowej Plus GSM) i – zdaniem tych i innych użytkowników – jest to efektywne środowisko programistyczne, charakteryzujące się łatwością konfiguracji i zarządzania, gwarantujące bezpieczeństwo danych, a także, co nie jest bez znaczenia, przystępne cenowo.

1.1. Teoria relacyjnych baz danych

W niniejszym opracowaniu *baza danych* jest rozumiana jako kolekcja danych, składowana w określonym miejscu i w określony sposób za pomocą technik kompute-

rowych. Dane umieszczone w takim „magazynie" dotyczą konkretnego wycinka rzeczywistości, są więc ze sobą związane obszarem tematycznym. Inaczej mówiąc, czy patrząc z innej strony, każda baza danych jest zbiorem określonych struktur danych; w przypadku baz relacyjnych mamy do czynienia tylko z jedną strukturą danych – jest to *relacja*, czyli omawiany „magazyn" zapełniony zbiorem relacji. Każdy model danych, a więc i model relacyjny może być pojmowany jako złożenie trzech komponentów:

- *Struktury*, czyli zbioru reguł określających sposób, w jaki baza danych może być konstruowana.
- *Części operacyjnej*, czyli zbioru definicji operacji dozwolonych na danych.
- *Ograniczeń*, czyli zbioru reguł integralności utrzymujących bazy danych w stanie spójnym, zgodnym z rzeczywistością.

Pojęcie relacji jest pojęciem matematycznym, często więc wprowadzając teorię relacyjnych baz danych, odwołujemy się do pojęć z dziedziny teorii zbiorów i logiki predykatów. Poniżej, po krótkim rysie historycznym, wyjaśnione zostaną terminy matematyczne oraz ich przełożenie na terminy stosowane w obszarze baz danych.

Za twórcę relacyjnego modelu danych (jako *model danych* rozumie się zintegrowany zbiór pojęć opisujących dane, zależności między nimi, ograniczenia nakładane na organizację danych oraz operacje wykonywane na danych) uważany jest E.F. Codd, który w 1970 roku opublikował artykuł pt. *A relational model of data for large shared data banks*. Artykuł ten stał się punktem zwrotnym w dziedzinie baz danych. Główne zalety w stosunku do poprzednich modeli (hierarchicznego i sieciowego) wynikające z podejścia Codda można krótko ująć w trzech punktach:

1. Duży stopień niezależności danych i aplikacji, tzn. aplikacje korzystające z danych nie wymagają zmian związanych z modyfikacją warstwy fizycznej bazy danych (organizacja plików, kolejność rekordów, ścieżki dostępu).
2. Precyzyjne reguły dotyczące rozwiązywania problemów powtórzeń (redundancji) danych; w swoim artykule Codd wprowadził pojęcie *normalizacji* relacji (omawiane szczegółowo w rozdziale 8).
3. Precyzyjny język zapytań bazujący na rachunku relacyjnym.

Zainteresowanie teorią zaprezentowaną przez Codda było na tyle duże, że już od roku 1970 w trzech ośrodkach informatycznych rozpoczęły się prace nad budową prototypów baz danych opartych na modelu relacyjnym. Pierwszy z projektów (System R) realizowany był w siedzibie IBM w Kalifornii, drugi (INGRES) na Uniwersytecie Berkeley, również w Kalifornii, i trzeci w siedzibie IBM w Wielkiej Brytanii. Projekty zostały ukończone w ciągu około sześciu lat, natomiast komercyjne rozwiązania pojawiły się na rynku informatycznym nieco później – na przełomie lat 70. I 80., ugruntowując swoją pozycję wraz z nadejściem ery komputerów osobistych.

Terminologia i definicje

Dla pełnego zobrazowania pojęcia *relacja* przypomnijmy zarys teorii matematycznej związanej z omawianym zakresem problemowym [9].

Załóżmy, że mamy dwa zbiory D_1 i D_2, gdzie $D_1 = \{20, 40\}$ i $D_2 = \{10, 30, 50\}$. Iloczyn kartezjański takich dwóch zbiorów oznaczony jako $D_1 \times D_2$, jest zbiorem wszystkich uporządkowanych par takich, że pierwszy element należy do D_1, a drugi element należy do D_2, inaczej mówiąc – są to wszystkie możliwe kombinacje elementów ze zbioru D_1 i D_2. W naszym przypadku otrzymamy:

$$D_1 \times D_2 = \{(20, 10), (20, 30), (20, 50), (40, 10), (40, 30), (40, 50)\}.$$

Każdy podzbiór iloczynu kartezjańskiego jest relacją. Przykładowy podzbiór:

$$R = \{(20, 10), (40, 10)\}.$$

Można określić, jakie uporządkowane pary mają pojawić się w relacji poprzez sprecyzowanie warunków ich wyboru. W podanym przykładzie łatwo można zaobserwować, że relacja R zawiera takie pary, w których drugi element równa się 10, czyli możemy zapisać R następująco:

$$R = \{(x, y) \mid x \in D_1, y \in D_2, y = 10\}.$$

W odniesieniu do tych samych zbiorów można zdefiniować inne relacje, na przykład relację S taką, że pierwszy element jest zawsze dwukrotnością drugiego, co zapiszemy następująco:

$$S = \{(x, y) \mid x \in D_1, y \in D_2, x = 2y\}.$$

W naszym przykładzie podany warunek spełnia tylko jedna para $S = \{(20, 10)\}$.

Podana notacja może być łatwo rozszerzona dla większej liczby zbiorów. Przykładowo, dla trzech zbiorów: iloczyn kartezjański $D_1 \times D_2 \times D_3$ jest zbiorem uporządkowanych trójek, takich, że pierwszy element pochodzi ze zbioru D_1, drugi z D_2, a trzeci z D_3. Niech $D_1 = \{10, 30\}$, $D_2 = \{20, 40\}$, $D_3 = \{50, 60\}$, wtedy

$$\begin{aligned} D_1 \times D_2 \times D_3 = \{&(10, 20, 50), (10, 20, 60), (10, 40, 50) (10, 40, 60),\\ &(30, 20, 50), (30, 20, 60), (30, 40, 50), (30, 40, 60)\}. \end{aligned}$$

Każdy podzbiór tych uporządkowanych trójek jest relacją.

Uogólniając tę definicję dla n zbiorów, możemy zapisać:

$$D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$$

lub krócej

$$\bigtimes_{i=1}^{n} D_i .$$

Każdy podzbiór n-tek (inaczej krotek) z takiego iloczynu kartezjańskiego jest relacją na n zbiorach. Należy zauważyć, że definiując relację w taki sposób, możemy specyfikować **zbiory** lub **dziedziny**, z których wybierane są wartości.

Korzystając z wprowadzonych powyżej pojęć, możemy zdefiniować *schemat relacji*.

Niech A_1, A_2, ..., A_n oznaczają atrybuty z dziedzinami D_1, D_2, ..., D_n, wtedy zbiór $\{A_1{:}D_1, A_2{:}D_2, ..., An : Dn\}$ nazywamy schematem relacji. Relacja R opisana schematem S jest więc zbiorem krotek (n-tek), wyznaczonych przez atrybuty z korespondującymi z nimi dziedzinami, co można zapisać formalnie:

$$(A_1 : d_1, A_2 : d_2, \ldots, A_n : d_n), \text{ gdzie } d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n.$$

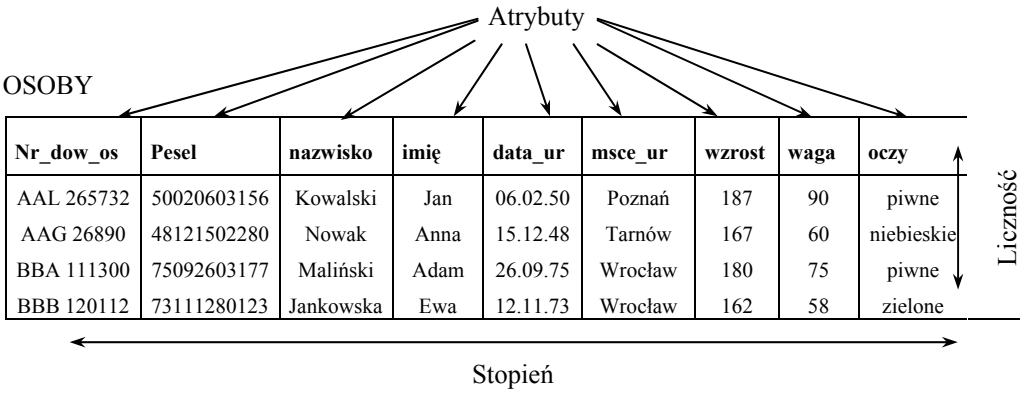
Każda relacja charakteryzuje się *stopniem*, czyli liczbą atrybutów, np. relacja jednoatrybutowa to relacja unarna, relacja dwuatrybutowa to relacja binarna itd., oraz *licznością*, czyli liczbą krotek, jakie zawiera.

Wracając do zastosowania teorii matematycznej w obszarze baz danych należy przypomnieć, że relacyjna baza danych to baza, w której dane mają strukturę relacji, inaczej mówiąc – jest to zbiór relacji. Każda z relacji przechowuje informacje o określonym obiekcie ze świata rzeczywistego. **Schemat bazy danych** jest to więc zbiór schematów relacji, z których każda ma unikalną nazwę. Jeżeli R_1, R_2, ..., R_n są zbiorem schematów relacji, to schemat bazy R zapisujemy następująco:

$$R = \{R_1, R_2, ..., R_n\}.$$

Schemat bazy danych opisuje więc całą bazę danych.

Przyjmuje się, że *tabela*, czyli płaska powierzchnia podzielona na wiersze i kolumny jest reprezentacją konstrukcji matematycznej, jaką jest relacja. Stosując taką reprezentację przyjmujemy, że *wiersz* tabeli odpowiada krotce relacji, *kolumna* tabeli odpowiada atrybutowi relacji. Należy pamiętać, że każdy z atrybutów relacji ma określoną *dziedzinę*, czyli dopuszczalny zbiór wartości, co oznacza, że w każdej kolumnie tabeli oczekiwane są wartości tego samego typu, określone przez dziedzinę. Przejdźmy do ilustracji praktycznych. Załóżmy, że w bazie danych będą przechowywane informacje dotyczące zbioru osób; niech osoby będą opisane poprzez atrybuty: numer dowodu osobistego, Pesel, NIP, nazwisko, imię, data urodzenia, miejsce urodzenia, wzrost, waga, kolor oczu. Reprezentacją relacji OSOBY będzie więc tabela przedstawiona na rys. 1.1.



Rys. 1.1. Przykładowa postać relacji OSOBY

Każdy z wierszy tabeli OSOBY reprezentuje konkretną osobę, każda osoba opisana jest poprzez podanie wartości atrybutów właściwych dla danej osoby. Patrząc na taki zbiór informacji, należy odpowiedzieć sobie na pytanie, z jakich zbiorów wartości mogą pochodzić poszczególne atrybuty (czy np. waga może być określona z dokładnością do 1 kg, czy dokładniej). Wartości, które występują w poszczególnych wierszach zależą od tego, jak zostały określone dziedziny dla poszczególnych atrybutów (rys. 1.2).

Atrybut	**Nazwa dziedziny**	**Znaczenie**	**Definicja dziedziny**
Nr_dow_os	Numery dowodów osobistych	Zbiór wszystkich dopuszczalnych numerów dowodów osobistych	character: size 10
Pesel	Numery Pesel	Zbiór wszystkich dopuszczalnych numerów Pesel	character: size 11
nazwisko	Zbiór nazwisk	Zbiór kombinacji znaków mogących stanowić nazwiska	character: size 80
imię	Zbiór imion	Zbiór imion dopuszczalnych w Polsce	character: size 20
data_ur	Daty urodzenia	Zbiór dopuszczalnych dat urodzenia	date: format dd-mm-rr
msce_ur	Miejsce urodzenia	Zbiór nazw miast	character: size 30
wzrost	Wzrost	Zbiór wartości określających wzrost	smallinteger: zakres 150–220
waga	Waga	Zbiór wartości określających wagę	smallinteger: zakres 50–120
oczy	Kolor oczu	Zbiór możliwych kolorów oczu	character: size 10

Rys. 1.2. Przykładowe dziedziny atrybutów dla relacji OSOBY

Uważna analiza danych pozwala stwierdzić, że relacja OSOBY zawiera dane dotyczące osób dorosłych, legitymujących się dowodem osobistym oraz numerem Pesel, stąd przy określeniu dziedziny atrybutów waga i wzrost wprowadzone zostały ograniczenia górne i dolne, z definicji dziedzin tych atrybutów wynika również, że dane dotyczące wagi i wzrostu muszą być liczbami całkowitymi.

Tabela z rys. 1.1 jest reprezentacją relacji, ponieważ spełnia warunki nakładane na relacje:

- Każda relacja (tabela) w bazie danych musi mieć jednoznaczną nazwę (zbiory matematyczne muszą być jednoznacznie nazywane).
- Każdy atrybut (kolumna) musi mieć jednoznaczną nazwę w ramach jednej relacji (każda kolumna jest zbiorem, musi więc być jednoznacznie nazwana).
- Wszystkie wartości w kolumnie muszą być tego samego typu (muszą pochodzić z zadeklarowanej dziedziny).

• Każde pole leżące na przecięciu wiersza i kolumny (komórka tabeli) musi zawierać wartość elementarną, nie mogą pojawiać się w komórkach tabeli wektory, macierze czy zbiory wartości.

• Każda krotka relacji (każdy wiersz tabeli) musi być unikalna, nie są dozwolone powtórzenia krotek (w zbiorach matematycznych elementy nie są powtarzalne).

• Nie jest istotna kolejność krotek (wierszy w tabeli) w relacji.

• Nie jest istotna kolejność atrybutów (kolumn) w relacji. Schemat relacji zawierający listę atrybutów jest również zbiorem matematycznym, a elementy zbioru nie są uporządkowane.

Jedna z podanych wyżej własności określa, że krotki w relacji (wiersze w tabeli) muszą być unikalne – dlatego też musi istnieć atrybut (kolumna) lub kompozycja atrybutów, które w sposób jednoznaczny identyfikują każdy z wierszy. Taki atrybut lub kompozycja atrybutów nosi nazwę *klucza głównego* (w przypadku kombinacji atrybutów mówimy o kluczu złożonym). W relacjach może istnieć kilka atrybutów mogących pełnić rolę identyfikatora, są to tzw. *klucze kandydujące*, spośród których wybierany jest klucz główny. Każdy z kluczy kandydujących, a tym samym klucz główny charakteryzuje się dwiema cechami:

– musi być jednoznaczny (wartości w kolumnie lub kompozycje wartości w przypadku kluczy złożonych) nie mogą się powtarzać,

– nie może zawierać wartości pustych (wartości *null*).

Dla przykładowej relacji z rys. 1.1 cechy takie mają dwie kolumny: kolumna Nr_dow_os i kolumna Pesel. Obie są więc kluczami kandydującymi i spośród nich należy wybrać klucz główny. Która więc z tych kolumn powinna zostać wybrana jako klucz główny?

Przypomnijmy raz jeszcze, że baza danych jest modelem konkretnego wycinka rzeczywistości, a relacja z rys. 1.1 stanowi przykład ilustracyjny, w którym atrybuty opisujące osoby są dość przypadkowe; nic nie zostało powiedziane, na jakie potrzeby gromadzone są dane w relacji OSOBY. W rzeczywistości każda osoba pełnoletnia posiadająca dowód osobisty i numer Pesel może być związana z różnymi „światami" – może studiować, może być pacjentem, podatnikiem, pracownikiem. W każdej z tych rzeczywistości będzie potrzebny trochę inny zbiór atrybutów do opisu osób. Jeżeli będziemy opisywać osobę, która jest studentem – czyli będziemy projektować uczelnianą bazę danych – z całą pewnością atrybuty takie jak wzrost, waga czy kolor oczu nie będą dla tej rzeczywistości istotne, musimy natomiast uwzględnić taki atrybut jak numer indeksu, który w dodatku w rzeczywistości uczelnianej identyfikuje jednoznacznie studenta – jest więc kluczem kandydującym obok numeru Pesel czy numeru dowodu osobistego. Jeżeli baza danych dotyczyłaby systemu podatkowego, to również część atrybutów z rys. 1.1 byłaby zbędna. Dla urzędów podatkowych, tak jak dla uczelni, nie są interesujące cechy fizyczne podatnika, ale z całą pewnością musiałby pojawić się w relacji opisującej podatnika numer identyfikacji podatkowej (NIP), pełniący rolę klucza kandydującego. W obu przypadkach z zestawu kluczy kandydują-

cych powinno wybrać się taki, który jest najbardziej związany tematycznie z daną rzeczywistością: w przypadku uczelni – numer indeksu, w przypadku systemu podatkowego – numer identyfikacji podatkowej (NIP).

Ilustracją rzeczywistości związanej z systemem podatkowym mogą być tabele: tabela przechowująca dane dotyczące podatnika oraz tabela przechowująca dane dotyczące urzędów skarbowych. Ponieważ każdy z podatników przynależy do określonego urzędu, w tabeli PODATNICY istnieje kolumna przechowująca wskaźniki do wierszy w tabeli URZĘDY SKARBOWE. Kolumna ta nosi nazwę *klucza obcego*. Poprzez klucz obcy realizowany jest sposób łączenia danych z różnych tabel. *Klucz obcy* jest kolumną, lub grupą kolumn tabeli, w której występują wartości z tej samej dziedziny jak w kluczu głównym tabeli związanej.

PODATNICY

NIP	**Nr_dow_os**	**Pesel**	**nazwisko**	**imię**	**data_ur**	**msce_ur**	**Nr_urzędu**
896-106-80-61	AAL 265732	50020603156	Kowalski	Jan	06.02.50	Poznań	1/wroc
894-115-97-90	AAG 26890	48121502280	Nowak	Anna	15.12.48	Tarnów	1/wroc
754-000-80-09	BBA 111300	75092603177	Maliński	Adam	26.09.75	Wrocław	2/wroc
567-609-75-22	BBB 120112	73111280123	Jankowska	Ewa	12.11.73	Wrocław	3/wroc
767-888-65-01	BBA-111350	68050402380	Adamiak	Piotr	04.05.68	Wrocław	4/wroc

klucz główny klucz obcy

URZĘDY SKARBOWE

Nr_urzędu	**nazwa_urzędu**	**kod_pocztowy**	**ulica**
1/wroc	Fabryczna	35-142	Ostrowskiego
2/wroc	Krzyki	34-290	Skarbowców
3/wroc	Śródmieście	32-100	Zapolskiej
4/wroc	Psie Pole	31-291	Zielona

klucz główny

Rys. 1.3. Powiązanie danych podatników z danymi urzędów skarbowych poprzez mechanizm klucza obcego

Przykład podany na rys. 1.3 może stanowić fragment schematu bazy danych, który zapisany za pomocą często stosowanej notacji nawiasowej (nazwa relacji, w nawiasach nazwy atrybutów związane z daną relacją, z podkreślonym kluczem głównym) ma postać:

PODATNICY (NIP, Nr_dow_os, Pesel, nazwisko, imię, data_ur, msce_ur, nr_rzędu)
URZĘDY SKARBOWE (Nr_urzędu, nazwa_urzędu, kod_pocztowy, ulica)

Integralność relacyjna

W poprzedniej części rozdziału przedstawiona została składowa strukturalna relacyjnego modelu danych. Przejdźmy teraz do składowej związanej z integralnością danych, istotną składową każdego modelu danych [5, 8, 9]. Integralność danych określona jest zbiorem reguł (więzów), pozwalających utrzymywać bazę danych w stanie spójnym, czyli wiernie odzwierciedlającym rzeczywistość. W odniesieniu do relacyjnego modelu danych *reguły integralności* (*więzy integralności*) można podzielić na następujące kategorie:

- integralność dziedziny,
- integralność encji,
- integralność referencyjna,
- dodatkowe więzy integralności.

Integralność dziedziny wynika z definicji atrybutów relacji – przy definiowaniu atrybutu należy określić jego nazwę i zakres dopuszczalnych wartości, czyli dziedzinę atrybutu. Żaden z atrybutów nie może przyjmować wartości spoza swojej dziedziny. Przykładowo, jeżeli dziedziną atrybutu kod_pocztowy jest zbiór kodów pocztowych obowiązujący w Polsce, to nie może pojawić się w tym polu wartość inna niż z książki kodowej.

Przed omówieniem pozostałych więzów integralności należy, w celu lepszego ich zrozumienia, omówić specjalną wartość występującą w systemach relacyjnych – wartość *null*. Wartość ta oznacza „nieznany" lub „brak informacji", nie jest równoznaczna ani z zerem, ani ze spacją. Wprowadzenie wartości *null* do systemu relacyjnego spowodowało jednak pewne komplikacje w operowaniu danymi (przetwarzanie zapytań do bazy danych opierające się na rachunku predykatów), ponieważ logika dwuwartościowa (prawda, fałsz) została zastąpiona logiką trójwartościową (prawda, fałsz, nie wiem). Problem ten podzielił specjalistów z zakresu baz danych na dwa obozy: jedni, łącznie z twórcą teorii relacyjnych baz danych, Coddem, dopuszczają występowanie wartości *null*, druga grupa natomiast, z równie znaną postacią jaką jest Date, uważa, że nie jest to konieczne.

Integralność encji odnosi się do wymogów nakładanych na atrybut będący kluczem głównym. Reguła ta określa, że każda z relacji musi posiadać atrybut identyfikujący (klucz główny), prosty lub złożony, wartości klucza głównego muszą być jednoznaczne i nie mogą zawierać wartości *null*. Jeśli klucz główny jest kluczem złożonym, to żadna z jego składowych nie może zawierać wartości *null*.

Integralność referencyjna związana jest z ograniczeniami wartości klucza obcego. W kluczu obcym mogą wystąpić dwa rodzaje wartości:

- wartość z dziedziny klucza głównego w przypadku, gdy istnieje związek pomiędzy danymi w tabelach,
- wartość *null*, jeżeli nie ma takiego związku lub stwierdzamy, że związek jest nieznany.

Jako przykład ilustracyjny rozważmy sytuację wyboru specjalności przez studentów, czyli związku studentów ze specjalnościami. W rzeczywistości na wszystkich

uczelniach studenci przyjmowani są na wydziały i kierunki w ramach wydziału, po pewnym czasie (zależy to od regulacji statutowych poszczególnych uczelni) studenci związują się z wybranymi przez siebie specjalnościami. Należy więc założyć taką sytuację, że część studentów wydziału jest związana ze specjalnością, część zaś studentów lat młodszych nie. Fakt ten musi być uwzględniony poprzez dopuszczenie możliwości pojawienia się w kluczu obcym wartości *null*. W bazie relacyjnej model przedstawionej powyżej rzeczywistości w obrębie jednego wydziału, w dużym uproszczeniu (przykład ma jedynie zilustrować działanie więzów referencyjnych) wygląda następująco:

STUDENCI (nr_indeksu, imię, nazwisko, imię ojca, data_ur, miejsce_ur, id_specjalności)
SPECJALNOŚCI (id_specjalności, nazwa, skrót_nazwy, opiekun)

SPECJALNOŚCI

Id_specjalności	**nazwa**	**opiekun**
ISK	Systemy i sieci komputerowe	Prof. XXX
IMT	Systemy informatyki w medycynie i technice	Prof. YYY
ASI	Systemy informatyczne w automatyce	Prof. ZZZ

↑ klucz główny

STUDENCI klucz obcy ↓

Nr_indeksu	**imię**	**nazwisko**	**Imię_ojca**	**data_ur**	**miejsce_ur**	**Id_specjalności**
100112	Jan	Marecki	Józef	1980	Wrocław	ISK
110230	Anna	Kowalska	Adam	1982	Wrocław	*null*
100560	Piotr	Nawrocki	Benedykt	1981	Konin	IMT
100999	Ewelina	Kamińska	Krzysztof	1981	Kraków	ASI
111780	Kamil	Nowak	Jan	1982	Ziębice	*null*

Rys. 1.4. Ilustracja działania więzów integralności referencyjnej

Reguły dotyczące więzów nie tylko określają, czy w kluczu obcym mogą pojawić się wartości *null*, czy też muszą wystąpić wartości z dziedziny klucza głównego. Reguły te muszą precyzować, co będzie się działo w przypadku usuwania czy modyfikacji danych w tabelach, które są ze sobą powiązane. W omawianym przykładzie należy określić, co stanie się z wierszami zawierającymi dane studentów, jeżeli jakaś specjalność zostanie zlikwidowana. Dysponujemy trzema możliwościami:

• Usuwanie ograniczone (*restricted*) – podejście restrykcyjne, które oznacza, że nie można zlikwidować specjalności, z którą związani są studenci. W rzeczywistości uczelnianej oznacza to, że specjalność może zostać zamknięta, jeżeli wszyscy studenci staną się absolwentami i ich dane można będzie usunąć z tabeli STUDENCI. Ogólnie mówiąc, nie można kasować wierszy z tabeli, jeżeli w innej tabeli występują wiersze powiązane.

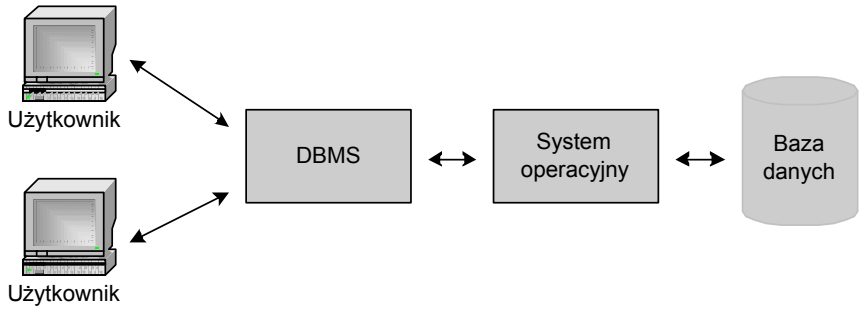
• Usuwanie kaskadowe (*cascaded*) – w podejściu kaskadowym, po usunięciu wierszy z jednej tabeli, wszystkie wiersze powiązane zostają automatycznie usunięte. W naszym przykładzie oznaczałoby to niewyobrażalną sytuację, w której wraz ze zlikwidowaniem specjalności studenci z nią związani zostaliby skreśleni.

• Reguła wstaw *null* (*set null*) – po usunięciu wierszy z jednej tabeli, w powiązanych wierszach w kolumnie klucza obcego ustawiona zostanie wartość *null*. W podanym przykładzie taka zasada wydaje się najbardziej wyważona; studenci mogą poczekać na propozycje władz wydziału dotyczące ponownego wyboru specjalności.

Dodatkowe więzy integralności. Oprócz omówionych więzów integralności (więzy te noszą nazwę podstawowych) istnieją mechanizmy umożliwiające definiowanie dodatkowych ograniczeń, których zadaniem będzie również utrzymywanie bazy danych w stanie spójnym, czyli reguł zapewniających zgodność modelu z rzeczywistością. Definicję oraz sposób implementacji takich reguł omówiono szczegółowo w rozdziale 4.

1.2. System Zarządzania Bazą Danych

Pojęcie *System Zarządzania Bazą Danych* – SZBD (*Database Management System* – DBMS, SZBD) w niniejszym podręczniku odnosić się będzie do systemów zarządzania relacyjnymi bazami danych. Najogólniej mówiąc, DBMS jest to specjalizowane oprogramowanie do obsługi baz danych, czyli oprogramowanie zapewniające użytkownikom możliwości definiowania, zapisywania, wyszukiwania i aktualizacji danych zawartych w bazie.



Rys. 1.5. Ogólna idea przetwarzania w systemach z bazą danych

Pierwsze komercyjne systemy zarządzania relacyjnymi bazami danych pojawiły się pod koniec lat siedemdziesiątych i we wczesnych latach osiemdziesiątych, najbardziej znane były DB2 i SQL/DS firmy IBM oraz Oracle firmy Oracle Co.

Obecnie rynek oferuje dziesiątki produktów DBMS, począwszy od produktów mniejszych, przeznaczonych na komputery klasy PC, przykładowo: Access i FoxPro firmy Microsoft, Paradox firmy Corel, InterBase czy BDE firmy Borland, do dużych, przeznaczonych dla komputerów klasy mainframe, przykładowo: Oracle, DB2, Informix, Sybase, Microsoft SQL Server i inne. Standardowe funkcje realizowane przez Systemy Zarządzania Bazą Danych można podzielić na trzy grupy:

1. Funkcje umożliwiające użytkownikom definiowanie bazy danych, to znaczy specyfikowanie typów, struktury danych i ograniczeń nakładanych na dane. Realizacja tych funkcji (zapamiętanie specyfikacji w bazie danych) odbywa się poprzez język DDL (*Data Definition Languague*), który jest komponentem języka SQL (*Structured Query Languague*) będącego obecnie formalnie i de facto standardem w relacyjnych systemach baz danych. Język SQL zostanie omówiony w rozdziałach 3 i 4.

2. Funkcje umożliwiające użytkownikom wstawianie, aktualizację, kasowanie i pobieranie danych z bazy, które realizowane są poprzez komponent DML (*Data Manipulation Languague*) języka SQL.

3. Funkcje umożliwiające sterowanie dostępem do bazy danych, realizowane przez:

a) system zabezpieczeń, który uniemożliwia nieautoryzowanym użytkownikom korzystanie z zasobów bazy danych,

b) system implementowania i kontroli więzów integralności, zapewniający spójność pamiętanych danych,

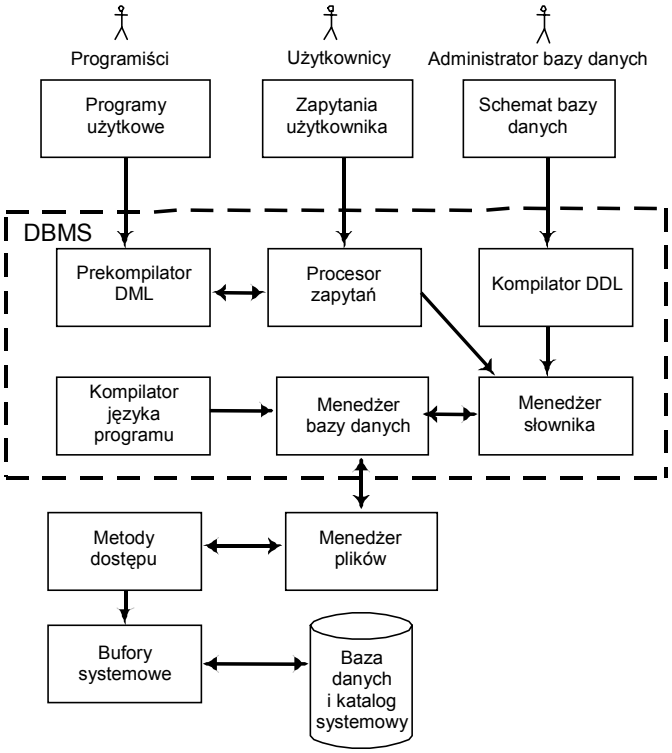
c) system sterowania współbieżną pracą, który zapewnia możliwość korzystania z zasobów bazy danych wielu użytkownikom,

d) system naprawczy, który odtwarza bazę danych w przypadku awarii sprzętu lub oprogramowania,

e) katalog systemowy (meta-dane) dostępny dla użytkowników, który zawiera opisy danych zawartych w bazie, opisy powiązań między danymi, ograniczenia nakładane na dane, autoryzacje użytkowników, czy wreszcie dane statystyczne, takie jak np. ilość i częstotliwość dostępów do bazy danych.

Systemy Zarządzania Bazami Danych są w istocie komponentami softwarowymi wysokiej złożoności, współpracującymi ze sobą w celu realizacji wymienionych wyżej funkcji. Pomimo tego, że na rynku informatycznym istnieje wielu producentów takich systemów i z całą pewnością ich produkty się różnią, można wydzielić reprezentatywną grupę modułów oraz określić relacje między nimi, przyjmując, że są to standardowe moduły typowych DBMS [9, 10, 24]. Patrząc w taki sposób na DBMS przyjmuje się, że każdy z modułów jest przypisany do wykonania określonej operacji. Należy pamiętać również o tym, że DBMS współpracuje z systemem operacyjnym, który wspomaga wykonanie niektórych funkcji, a co za tym idzie – muszą istnieć in-

terfejsy między Systemem Zarządzania Bazą Danych a systemem operacyjnym. Przykładem może być sytuacja związana z ulokowaniem bazy danych i katalogu systemowego. Zazwyczaj baza danych i katalog systemowy przechowywane są na komputerowym nośniku pamięci, a nie w pamięci wewnętrznej. Dostępem do urządzeń we-wy steruje system operacyjny. Moduł wyższego poziomu DBMS zarządzający dostępem do informacji przechowywanych w bazie używa do tego celu funkcji zarządzania danymi, na niższym poziomie realizowanych przez system operacyjny (system zarządzania plikami – menedżer plików). Na rysunku 1.6 przedstawiono główne moduły Systemów Zarządzania Bazami Danych.



Rys. 1.6. Główne moduły Systemu Zarządzania Bazą Danych

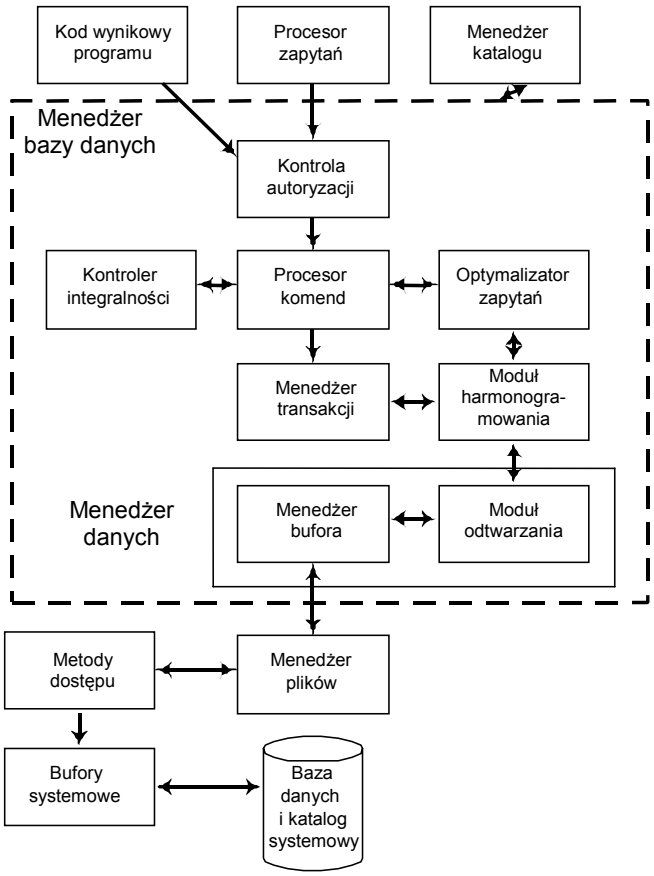
Znaczenie głównych modułów:

- *Procesor zapytań*: obsługuje zapytania interaktywne skierowane do bazy danych, wyrażone w języku operowania danymi (np. SQL). Dokonuje analizy składniowej i semantycznej zapytania.
- *Prekompilator* DML: obsługuje aplikacje i programy użytkownika, w których są osadzone polecenia DML. Polecenia DML zostają przetłumaczone na kod wynikowy

realizujący dostęp do bazy danych, pozostałe części programów obsługuje kompilator konwencjonalny. Obie części są następnie wiązane razem.

- *Kompilator* DDL: konwertuje zdania DDL na zbiór tabel zawierających metadane zapamiętywane w katalogu systemowym poprzez moduł menedżera katalogu.
- *Menedżer bazy danych*: obsługuje zarówno aplikacje i programy użytkowników, jak i zapytania zadawane w trybie interaktywnym. Menedżer bazy danych sprawdza zgodność zapytań ze schematami bazy danych oraz określa, które rekordy danych powinny zostać pobrane dla spełnienia wymogów zapytania.
- *Menedżer katalogu*: zarządza dostępem do katalogu systemowego i jego obsługą.

Każdy z wymienionych modułów rzadko kiedy jest jednorodną całością, raczej charakteryzuje się również budową modułową, składa się z mniejszych komponentów. Standardowe komponenty modułu menedżera bazy danych pokazano na rys. 1.7 [1].



Rys. 1.7. Główne komponenty modułu menedżera bazy danych

Znaczenie komponentów wchodzących w skład modułu menedżera bazy danych:

- *Kontroler autoryzacji*: sprawdza autoryzację użytkownika.
- *Procesor komend*: sprawdza ponownie uprawnienia użytkownika do wykonywanej operacji oraz rodzaj wykonywanej operacji. W zależności od rodzaju operacji sterowanie przekazywane jest do kontrolera integralności, optymalizatora zapytań lub menedżera transakcji.
- *Kontroler integralności*: sprawdza, czy operacje, które zmieniają dane w bazie danych są zgodne z więzami integralności.
- *Optymalizator zapytań*: wyznacza optymalną strategię realizacji zapytań.
- *Menedżer transakcji*: obsługuje transakcje do bazy danych.
- *Moduł harmonogramowania*: steruje pracą współbieżną, inaczej mówiąc zarządza kolejnością wykonywania transakcji.
- *Moduł odtwarzania*: zabezpiecza stan spójny bazy danych w przypadku wystąpienia błędów przetwarzania.
- *Menedżer bufora*: odpowiada za transfer danych pomiędzy pamięcią główną komputera a pamięcią zewnętrzną (dyski, taśmy). Moduł odtwarzania i menedżer bufora często są traktowane łącznie jako *menedżer danych*.

Opierając się na krótkiej charakterystyce budowy oraz funkcji, jakie realizują systemy zarządzania relacyjnymi bazami danych, a dokładniej jądra takich systemów, można odnieść wrażenie, że rozwiązane zostaną wszystkie problemy dotyczące przetwarzania danych, jeżeli tylko użytkownik będzie miał możliwość zaimplementowania w swoim komputerze (lub komputerach) odpowiedniego środowiska bazodanowego. Niestety, jak wszystkie doskonałe i nowoczesne technologie, nawet te najbardziej zaawansowane, tak i technologie oraz narzędzia związane z bazami danych mają oprócz niewątpliwych zalet także wady. Spróbujmy więc krótko podsumować, jakie są główne zalety, a jakie wady przetwarzania danych lokowanych w bazach danych i zarządzanych przez DBMSy.

Zalety:

- Uniknięcie redundancji danych – przechowywanie danych w bazach danych z definicji zakłada unikanie powtórzeń danych; informacje związane z jednym obiektem z założenia pamiętane są tylko raz.
- Uniezależnienie danych od aplikacji – jest to niewątpliwie jedna z ważniejszych zalet systemów baz danych. Zmiana struktury danych nie pociąga za sobą konieczności poprawiania pracujących dotychczas programów.
- Zapewnienie spójności danych – poprzez eliminację redundancji danych oraz umożliwienie implementacji reguł integralności, których przestrzeganie jest kontrolowane przez DBMS, ryzyko anomalii związanych z aktualizacją, dopisywaniem nowych danych czy kasowaniem danych redukuje się praktycznie do zera.
- Współdzielenie danych – z danych przechowywanych w bazie korzysta wiele aplikacji i wielu użytkowników (oczywiście zgodnie z uprawnieniami).

• Zwiększenie bezpieczeństwa danych – systemy zarządzania bazami danych wyposażone są zarówno w mechanizmy zabezpieczeń przed dostępem do bazy danych przez nieautoryzowanych użytkowników, jak i w mechanizmy umożliwiające odtworzenie zawartości bazy po błędach aplikacji działających na bazie danych lub awariach sprzętu.

• Standaryzacja – korzystanie z bazy danych wymusza konieczność stosowania się do określonych standardów przez wszystkich użytkowników. Dotyczy to zarówno formatów danych, jak i nazewnictwa, standardów dokumentów lub procedur aktualizacyjnych.

• Czynnik ekonomiczny – trudno w sposób dokładny oszacować zyski finansowe, osiągane dzięki nakładom poniesionym na inwestycje informatyczne, ale można wyobrazić sobie sytuację w instytucji składającej się z kilku wydziałów, z których każdy używa swojego sposobu przechowywania danych i ich przetwarzania. Nie jest to dobry sposób z wielu względów: utrudniony jest dostęp do całości informacji, każdy z wydziałów ponosi koszty utrzymania zbiorów danych, kosztowna jest wymiana informacji, brak jest gwarancji zachowania standardów. Wydaje się, nawet bez dokładnego szacowania, że lepszym rozwiązaniem jest komasacja budżetu dla zainwestowania w system bazodanowy satysfakcjonujący całość instytucji, z jednym źródłem danych i zbiorem aplikacji korzystających z tego źródła.

Wady:

• Złożoność – technologie i narzędzia, zwłaszcza związane z dużymi bazami danych, stają się coraz bardziej skomplikowane i trudne do opanowania; łatwo wyobrazić sobie, że bez dobrego zrozumienia zasad funkcjonowania czy administrowania systemem łatwo o błędne decyzje, których konsekwencje będą odczuwalne przez całe organizacje.

• Rozmiar – złożoność i szeroki zakres funkcjonalny powodują, że DBMS jest sam w sobie ogromną częścią softwaru, zajmuje megabajty pamięci i wymaga dużych zasobów dla efektywnej pracy.

• Koszty finansowe – zależą oczywiście od wymagań i potrzeb użytkownika i mogą się znacząco różnić. Przykładowo, jeżeli koszt DBMS dla pojedynczego użytkownika na komputerze osobistym jest niewielki, to koszt dużych systemów bazodanowych dla wielu użytkowników może sięgać setek tysięcy złotych, do czego dochodzą jeszcze koszty utrzymania systemu, które są proporcjonalne do ceny zakupu.

• Dodatkowe nakłady na sprzęt – często ze względu na rozmiar DBMS dotychczasowy sprzęt może okazać się niewystarczający, zwłaszcza jeżeli chodzi o zasoby pamięci. Nierzadko okazuje się, że w celu zwiększenia efektywności przetwarzania należy przydzielić komputer o dobrych parametrach dla DBMS.

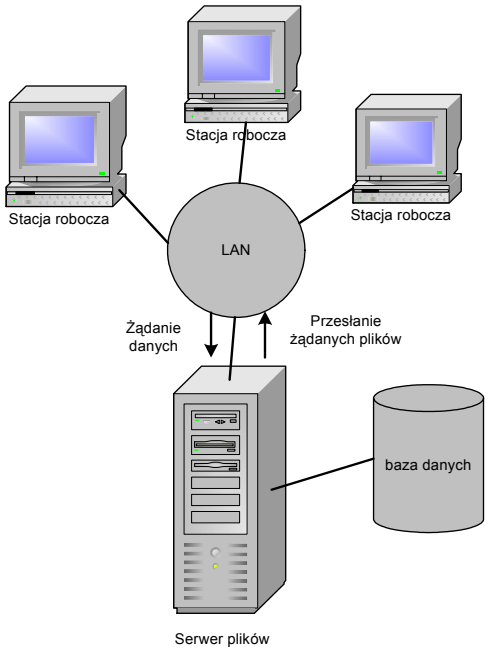
• Efektywność przetwarzania – to dość drażliwa sprawa, należy jednak mieć świadomość, że z definicji baza danych i zarządzający nią system przeznaczone są do obsługi wielu aplikacji, może się więc zdarzyć, że niektóre z nich będą pracowały wydajniej, niektóre mniej wydajnie.

1.3. Architektura systemów baz danych

Wraz z rozwojem sieci komputerowych, zwłaszcza sieci lokalnych LAN, i zwiększeniem możliwości operacyjnych komputerów osobistych znacznie wzrosło zainteresowanie rozwiązaniami umożliwiającymi korzystanie z bazy danych przez wielu użytkowników. Pojawiły się architektury systemów baz danych umożliwiające taki sposób pracy [14, 15, 23].

Architektura jednowarstwowa – serwer plików

W architekturze jednowarstwowej można mówić o rozproszonym przetwarzaniu z wykorzystaniem sieci LAN. Serwer plików zarządza zasobami danych wymaganymi przez programy i aplikacje. Zarówno *System Zarządzania Bazą Danych* (DBMS), jak i aplikacje są posadowione na każdej stacji roboczej. Programy i aplikacje poprzez DBMS zgłaszają do serwera plików zapotrzebowanie na dane (rys. 1.8).

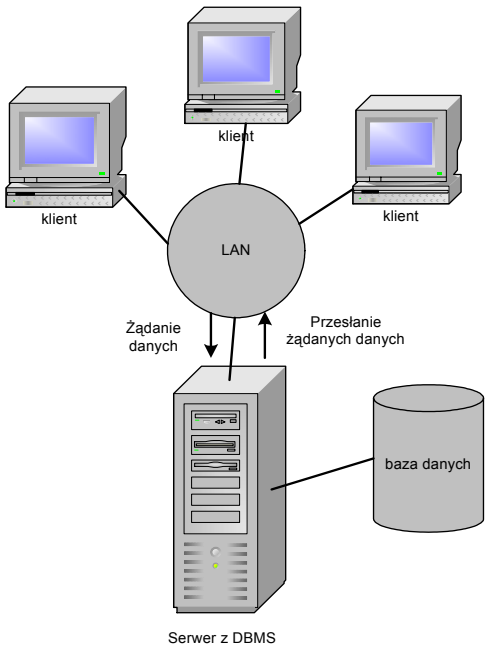


Rys. 1.8. Architektura jednowarstwowa z serwerem plików

Serwer plików w zasadzie umożliwia współdzielenie obszaru dyskowego. Główną wadą takiej architektury jest duże obciążenie sieci, a co za tym idzie problemy z działaniem programów i aplikacji. Następną niedogodnością jest konieczność utrzymywania kopii DBMS na każdej ze stacji roboczych i wreszcie problemy ze sterowaniem w czasie pracą wielu użytkowników.

Architektura dwuwarstwowa klient–serwer

Rozwiązanie omówione poprzednio należy do rozwiązań starszych, ze względu na swoje wady zostało w zasadzie wyparte przez bardzo dzisiaj powszechną architekturę klient–serwer. Termin architektura typu klient–serwer wywodzi się od sposobu interakcji komponentów softwarowych z systemem. Ogólnie mówiąc, *klient* jest procesem, który potrzebuje pewnych zasobów, a proces *serwera* tych zasobów dostarcza, realizując zgłoszone zapotrzebowanie. Mamy więc do czynienia z dwoma warstwami oprogramowania: warstwą serwera i warstwą klienta. Nie ma wymagań formalnych co do lokalizacji obu procesów, teoretycznie mogą one znajdować się na jednym komputerze, w praktyce jednak serwer umieszczany jest na innym komputerze niż procesy klienta, z którymi komunikuje się poprzez LAN. Ideę architektury klient–serwer obrazuje rys. 1.9.



Rys. 1.9. Dwuwarstwowa architektura klient–serwer

W kontekście środowiska baz danych proces klienta, działając na komputerze, na którym działa również aplikacja, zarządza interfejsem użytkownika i logiką aplikacji. Proces serwera natomiast kontroluje autoryzację użytkownika, zabezpiecza stan spójny bazy, wykonuje zapytania do bazy danych i aktualizacje. Ponadto serwer steruje pracą współbieżną oraz mechanizmami odtwarzania bazy danych.

Tabela 1.1 zawiera zestawienie funkcji realizowanych przez stronę klienta i stronę serwera.

Tabela 1.1. Główne funkcje realizowane przez proces klienta i proces serwera

Klient	Serwer
Logika prezentacji – obsługa interfejsu użytkownika	Akceptowanie i realizacja żądań klienta do bazy danych
Akceptowanie i kontrola syntaktyki zgłoszeń użytkownika	Kontrola autoryzacji
Przetwarzanie logiki aplikacji	Kontrola więzów integralności
Generowanie zapytań do bazy danych i transmisja do serwera	Wykonywanie zapytań i transmisja odpowiedzi do klienta
Przekazywanie odpowiedzi do użytkownika	Obsługa katalogu systemowego
	Sterowanie pracą współbieżną
	Sterowanie odtwarzaniem stanu bazy danych

Architektura klient–serwer zdominowała sposób przetwarzania w obszarze baz danych ze względu na wiele zalet, m.in.:

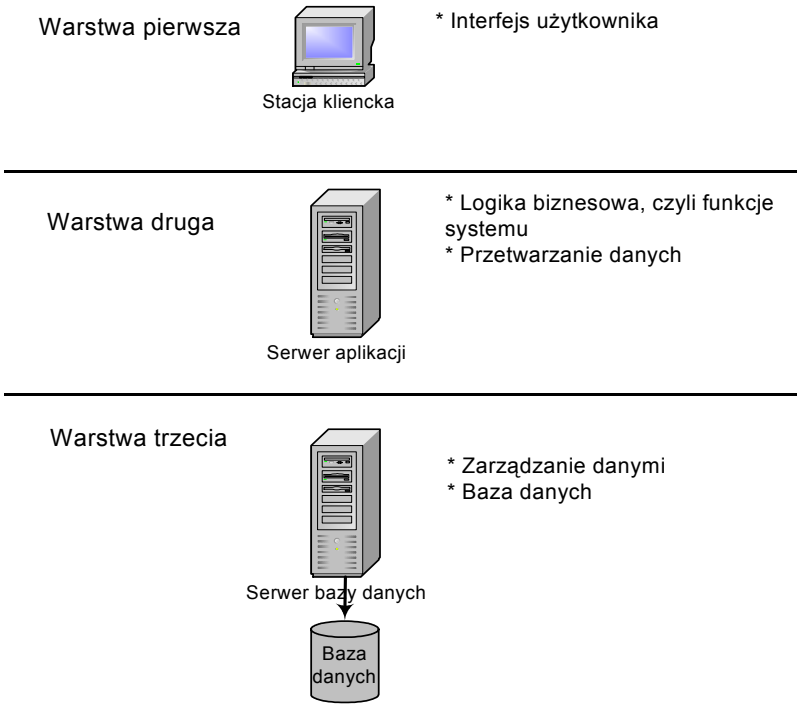
- szeroki dostęp do istniejących baz danych,
- poprawienie współczynników przetwarzania – chociażby przez fakt rezydowania na różnych komputerach serwera i klienta oraz możliwość zrównoleglenia pracy aplikacji,
- zredukowanie w pewnym zakresie kosztów sprzętu – komputerem o dobrych parametrach musi być maszyna, na której posadowiony jest serwer, w odniesieniu do stacji klienckich wymagania nie są aż tak wysokie,
- zmniejszenie kosztów komunikacji w sieci – część przetwarzania odbywa się po stronie klienta, serwer przesyła z bazy tylko dane wynikowe,
- zwiększenie spójności i bezpieczeństwa bazy danych – serwer kontroluje więzy integralności, a więc są one obowiązujące dla wszystkich pracujących aplikacji.

Na koniec należy zwrócić uwagę, że często architektura klient–serwer podawana jest jako przykład rozproszonego systemu baz danych. Nie jest to stwierdzenie słuszne, architektura klient–serwer sama w sobie nie konstytuuje bowiem rozproszonego systemu zarządzania bazą danych (rozproszonego DBMS). Jeżeli można mówić przy tej okazji o rozproszeniu, to raczej o rozproszeniu funkcjonalnym: część funkcji realizowana jest przez stronę klienta, część przez stronę serwera. Rozproszone systemy baz danych, czyli bazy, w których występuje fizyczne rozproszenie danych często do odległych nawet geograficznie serwerów są zarządzane rozproszonymi systemami zarządzania, które mogą być realizowane w architekturze klient–serwer, ale są to w swojej idei i w założeniu inne systemy.

Architektura trójwarstwowa klient–serwer

Do połowy lat dziewięćdziesiątych dwuwarstwowa architektura klient–serwer była dominującą na rynku baz danych. Wraz z rozwojem technik internetowych okazała się jednak niewystarczająca – pojawiły się rozwiązania bardziej zaawansowane technolo-

gicznie i funkcjonalnie. Ogólnie te nowe, dynamicznie rozwijające się architektury, noszące nazwę wielowarstwowych, umożliwiają integrację systemów baz danych ze środowiskiem WWW (*World Wide Web*). Przykładem architektury wielowarstwowej jest tzw. architektura trójwarstwowa. Jej ideę obrazuje rys. 1.10.



Rys. 1.10. Trójwarstwowa architektura klient–serwer

Zgodnie z ideą pokazaną na rys. 1.10, architektura trójwarstwowa wymaga rozdzielenia trzech warstw funkcjonalnych na trzy warstwy logiczne. Nie oznacza to wcale, że potrzeba trzech oddzielnych komputerów, aby taką architekturę zaimplementować. Istotne jest to, że każda z warstw jest oddzielnym „bytem", każda z nich działa jako oddzielna aplikacja lub proces. W takim podejściu logika biznesowa jest oddzielona od warstwy prezentacji i dostępu do danych. Bardziej precyzyjną specyfikację, dotyczącą zarówno modelu, jak i technologii można znaleźć w odpowiednich pracach [17, 21, 25].

W przypadku, gdy w pierwszej warstwie, czyli warstwie klienta lokowany jest jedynie interfejs użytkownika, który realizuje prezentację danych i przekazywanie danych do warstwy aplikacji, mówi się o tzw. „chudym kliencie" ze względu na małą ilość funkcji, jaką w tej technologii realizuje strona klienta. Nie jest to jedyne możliwe rozwiązanie – bardziej złożone interfejsy (realizujące zarówno interfejs użytkownika,

jak i część logiki aplikacji) można budować za pomocą dodatkowych skryptów lub gotowych komponentów instalowanych na stacji roboczej klienta; mamy wtedy do czynienia z „grubym klientem". Rozwiązania takie są uważane dzisiaj za bardziej tradycyjne, ze względu na mniejszą efektywność. W praktyce w rozwiązaniu z „cienkim klientem" interfejs użytkownika jest stroną WWW, którą obsługuje przeglądarka internetowa. Komunikacja interfejsu użytkownika z drugą warstwą, czyli serwerem aplikacji, odbywa się poprzez protokół HTTP lub nowsze technologie, takie jak standard ISAPI, ASP, JDBC lub *cookies*.

Warstwa druga, w której implementowane są reguły biznesowe, realizuje funkcje systemu oraz przetwarzanie danych, kontaktuje się z warstwą klienta i serwerem lub serwerami bazy danych poprzez sieć lokalną LAN (*Local Area Network*) lub rozległą WAN (*Wide Area Network*). Warstwę tę tworzy zestaw obiektów wielokrotnego użytku, nazywanych często obiektami biznesowymi. Obecnie istnieje spory wybór technologii i produktów umożliwiających fizyczną realizację warstwy drugiej, na przykład Serwer ASP (*Active Server Pages*), JSP (*Java Serwer Pages*), PHP (*Hypertext Preprocesor*), NDO (*NetWare Data Object*).

Warstwa trzecia omawianej architektury jest odpowiedzialna za fizyczne przetwarzanie i magazynowanie informacji. Najczęściej stanowią ją serwery baz danych. Jeśli bazy danych są udostępniane w Internecie, czyli z serwisu bazodanowego może korzystać duża i losowa liczba klientów, to bardzo ważna jest skalowalność serwera bazy danych, rozumiana jako możliwość zwiększenia wydajności aplikacji wraz z rosnącą liczbą użytkowników [4, 21].

Omawiając architekturę trójwarstwową i jej związek ze środowiskiem internetowym, nie można pominąć kluczowego elementu internetowych aplikacji bazodanowych, jakim jest serwer WWW. Stanowi on spoiwo na poziomie komunikacyjnym omawianych trzech warstw: prezentacji, aplikacji i danych. Do najpopularniejszych serwerów należą: Apache HTTP Serwer, Microsoft Internet Explorer (IIS), Netscape Navigator, AOL Server, Xitami.

Architektura trójwarstwowa znalazła bardzo istotne zastosowania praktyczne w systemach; przykładem mogą być chociażby systemy klasy OLTP (*On Line Transaction Processing; systemy bieżącego przetwarzania transakcji*), w których w warstwie środkowej zaalokowany jest serwer aplikacji wraz z modułem monitorowania transakcji (*Transaction Processing Monitor*).

Podsumowując ogólną prezentację architektury trójwarstwowej, należy podkreślić, że bazy danych stały się nieodłączną częścią środowiska internetowego, powszechne stają się rozwiązania tzw. e-biznesowe, integrujące bazy danych i technologie internetowe wokół infrastruktury biznesowej, czyli marketingu, reklamy oraz obsługi klientów w sferze realizacji zamówień, obsługi transakcji handlowych, świadczenia usług, np. bankowych.

1.4. Składowe systemów baz danych

Analizując dowolne środowisko baz danych, niezależnie od architektury czy konkretnego *Systemu Zarządzania Bazą Danych*, można wyróżnić pięć zasadniczych komponentów tworzących *systemy z bazą danych*: sprzęt, oprogramowanie, dane, procedury i ludzie.

Sprzęt

Jest rzeczą oczywistą, że każdy DBMS i każda aplikacja wymaga zasobów sprzętowych. Zakres wymagań sprzętowych zależy oczywiście od potrzeb użytkownika, konkretnych rozwiązań czy wreszcie wymagań samego DBMS. W konkretnych sytuacjach wystarczający okazać się może pojedynczy komputer osobisty, ale w przypadku większych systemów może być potrzebny komputer klasy mainframe lub komputery połączone siecią.

Oprogramowanie

Składowymi oprogramowania są zarówno same *Systemy Zarządzania Bazą Danych*, jak i aplikacje i programy użytkowe, systemy operacyjne oraz oprogramowanie sieciowe. Najczęściej programy i aplikacje bazodanowe są pisane w językach trzeciej generacji (3GL), takich jak C, C++, Java, Pascal oraz w językach czwartej generacji (4GL), takich jak SQL osadzony w kodach języków trzeciej generacji. W dzisiejszych rozwiązaniach *Systemy Zarządzania Bazami Danych* są wyposażane w zestawy narzędzi RAD (*Rapid Aplication Development*), umożliwiających szybkie tworzenie aplikacji bazodanowych.

Dane

Przez pojęcie danych należy rozumieć zarówno dane operacyjne, jak i metadane, czyli „dane o danych”. Strukturę danych określa schemat bazy danych.

Procedury

Procedury precyzują zasady projektowania i użytkowania bazy danych, mogą dotyczyć przykładowo:

- sposobu logowania się do konkretnego DBMS,
- instrukcji użytkowania konkretnego DBMS lub konkretnej aplikacji,
- tworzenia kopii bezpieczeństwa bazy danych,
- instrukcji obsługi błędów,
- instrukcji strojenia bazy danych w celu zwiększenia wydajności.

Ludzie

Omawiając udział i związki ludzi ze środowiskiem baz danych, należy przyjrzeć się przede wszystkim rolom, jakie są przypisane poszczególnym grupom ludzkim. Patrząc przez pryzmat pełnionych ról, można wyróżnić cztery kategorie partycypujące

w środowisku bazodanowym: administratorzy baz danych, projektanci baz danych, projektanci i programiści aplikacji bazodanowych oraz użytkownicy końcowi.

- Administratorzy baz danych (*Database Administrator* – DBA) są odpowiedzialni za zarządzanie zasobami bazy danych z ukierunkowaniem na stronę techniczną, w tym monitorowanie użycia danych i strojenie systemu, zapewnienie bezpieczeństwa danych, zabezpieczenie przestrzegania standardów danych, tworzenie grup użytkowników i przydział haseł, szkolenie użytkowników w zakresie pracy z bazą danych, testowanie systemu, wykonywanie kopii zapasowych bazy danych oraz odtwarzanie danych w przypadkach awarii, aktualizowanie wersji systemu.
- Projektanci baz danych są związani z procesem projektowania bazy danych zarówno logicznym, jak i fizycznym. W projekcie logicznym na podstawie analizy obszaru, dla którego projektowana jest baza danych, ustala się, jakie dane będą przechowywane w bazie, jakie są powiązania między danymi, jakie obowiązują ograniczenia odnośnie danych. W projekcie fizycznym zapadają decyzje, jak fizycznie implementować projekt logiczny, tzn. jakie wybrać środowisko, aby w pełni zrealizować założenia logiczne. Inaczej mówiąc, w części logicznej projektu określa się, „co" należy zrobić, a w części fizycznej podaje się, „jak" to zrobić. Z tego sformułowania widać, że potrzebne są nieco inne umiejętności do realizacji projektu logicznego i do projektu fizycznego, często więc przy dużych projektach inna grupa ludzi zajmuje się projektowaniem logicznym, a inna fizycznym. Metodologia projektowania baz danych zostanie szerzej omówiona w kolejnych rozdziałach.
- Projektanci i programiści aplikacji związani są z konstruowaniem programów aplikacyjnych umożliwiających działania na bazie danych. Zwykle programiści związani ze środowiskiem baz danych na podstawie specyfikacji wymagań ustalonej przez analityków systemu konstruują aplikacje w językach trzeciej lub czwartej generacji, umożliwiające przetwarzanie danych zawartych w bazie, tzn. ich aktualizację, kasowanie, dopisywanie czy prezentację w określonym układzie.
- Użytkownicy końcowi – wydawać by się mogło, że jest to określenie samodefiniujące, ale należy zwrócić uwagę na to, że użytkownicy systemów informatycznych to osoby o różnym poziomie wiedzy. Jedna kategoria to tacy użytkownicy, którzy nie mają żadnej wiedzy informatycznej. Korzystają z zasobów bazy poprzez aplikacje realizujące określone funkcje, zazwyczaj z prostym intuicyjnym interfejsem. Druga kategoria to użytkownicy „świadomi", znający środowisko baz danych i możliwości *Systemu Zarządzania Bazą Danych*. Użytkownicy ci sami mogą operować na bazie danych, czy nawet projektować i budować aplikacje bazodanowe na swój własny użytek.

Środowisko Sybase SQL Anywhere Studio

Pakiet Sybase SQL Anywhere Studio jest zintegrowanym zestawem produktów do zarządzania danymi oraz synchronizacji danych w obrębie organizacji, firmy czy przedsiębiorstwa, umożliwiającym szybkie opracowywanie i wdrażanie aplikacji [16,23]. W skład pakietu wchodzą:

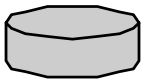
- Adaptive Serwer Anywhere (ASA) – 32-bitowy, w pełni relacyjny, wielodostępny serwer bazodanowy. Może on działać zarówno na serwerze firmy, jak i na komputerach przenośnych, czy też palmtopach. Obsługuje komonenty Javy wbudowane w bazę danych. Serwer ASA jest łatwy do uruchomienia i konfigurowania, działa pod wieloma systemami operacyjnymi: Windows 95/98, Windows NT, Windows CE, Novell NetWare, SunSolaris czy Linux oraz protokołami sieciowymi TCP/IP, SPX, NetBIOS.
- Sybase Central – graficzne narzędzie umożliwiające administrowanie bazą danych w zakresie modyfikowania bazy danych i obiektów bazy danych.
- Sterowniki ODBC i JDBC.
- Sybase SQL Remote – mechanizm do dwukierunkowej replikacji, wykorzystywany przez użytkowników komputerów przenośnych oraz w systemach rozproszonych bez stałego połączenia sieciowego. Mechanizmy replikacji umożliwiają replikację do i z takich źródeł jak: Oracle, Informix, DB2, IMS, VSAM.
- PowerDynamo – serwer aplikacji internetowych do prezentacji danych na stronach WWW.
- Infomaker – narzędzie do raportowania i analizy danych.

Pakiet SQL Anywhere Studio ma dość mocną pozycję na rynku produktów z wbudowaną bazą danych, gdyż zapewnia dobrą wydajność i skalowalność, przy niskich kosztach utrzymania i administracji. Dodatkową zaletą tych produktów jest to, że bezproblemowo współpracują z innymi rozwiązaniami firmy Sybase, np. pakietem PowerDesigner, który jest zestawem narzędzi CASE do projektowania baz danych oraz narzędziem typu RAD do budowy aplikacji, czyli pakietem PowerBuilder. Środowiska projektowe i programistyczne zostaną omówione szczegółowo w rozdziałach 6, 7 i 10.

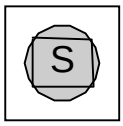
2.1. Architektura serwera ASA

Serwer ASA jest Systemem Zarządzania Relacyjnymi Bazami Danych, czyli bazami, w których dane są zorganizowane w tabelach. Może on pracować w dwóch wersjach: jako personalny serwer bazodanowy lub jako serwer sieciowy [23, 26]. W wersji sieciowej oprócz realizacji wszystkich funkcji serwera personalnego zapewnia komunikację stacji klienckich z serwerem poprzez sieć. Komponenty systemu:

Baza danych – w środowisku ASA jest plikiem o określonej nazwie z rozszerzeniem *.db*. Do pakietu dołączona jest przykładowa baza danych o nazwie *asademo.db*. Na rysunkach ilustrujących architekturę systemu baza danych jest oznaczana jako:

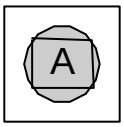


Serwer bazy danych – obsługuje i zarządza dostępem do bazy danych. Każda aplikacja komunikuje się z bazą poprzez serwer. Serwer jest oznaczany jako:



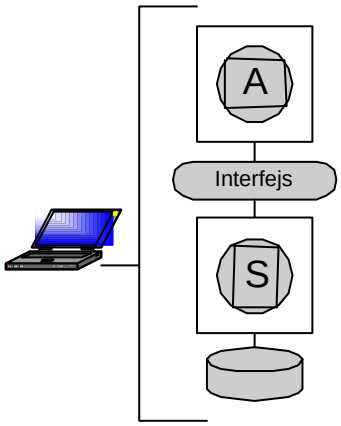
Interfejs programowy – aplikacje komunikują się z serwerem poprzez interfejs. W środowisku ASA udostępnione są interfejsy: ODBC (*Open Database Connectivity*), JDBC (*Java Database Connectivity*), OLE DB (*Database Object Linking and Embedding*), Sybase Open Client, Embedded SQL (*Osadzony SQL*). Interfejs jest oznaczany jako:

Aplikacja klienta – komunikuje się z bazą danych poprzez jeden z interfejsów. Narzędzia typu RAD, takie jak np. PowerBuilder, mają wbudowane własne metody komunikacji z serwerem, dlatego – konstruując aplikację za pomocą tych narzędzi, wykorzystuje się metody sprzężone z odpowiednim narzędziem. Oznaczenie aplikacji klienta:



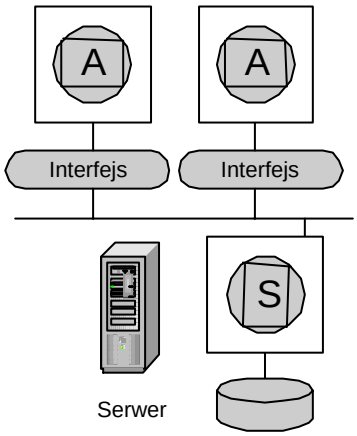
W przypadku, gdy baza danych, aplikacja i serwer posadowione są na jednym komputerze, mamy do czynienia z tzw. aplikacją wolno stojącą (*standalone)*, gdzie

baza danych jest częścią aplikacji, często określaną jako baza wbudowana (*embedded database*). Architektura takiego rozwiązania jest przedstawiona na rys. 2.1.



Rys. 2.1. Architektura systemu jednokomputerowego

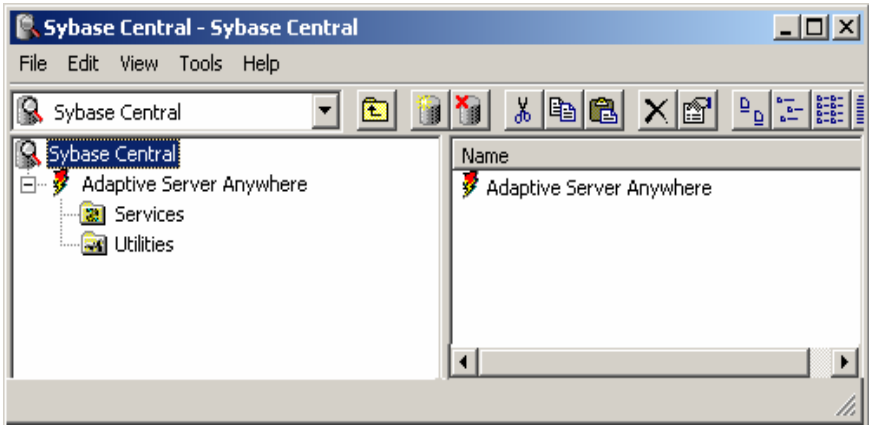
Środowisko ASA obsługuje również instalacje wieloużytkownikowe, gdzie aplikacje pracują na różnych komputerach, łącząc się poprzez sieć z serwerem pracującym na oddzielnej maszynie. W tym rozwiązaniu biblioteki interfejsu są ulokowane na każdym komputerze klienckim. Architekturę tę obrazuje rys. 2.2.



Rys. 2.2. Architektura systemu z wieloma użytkownikami

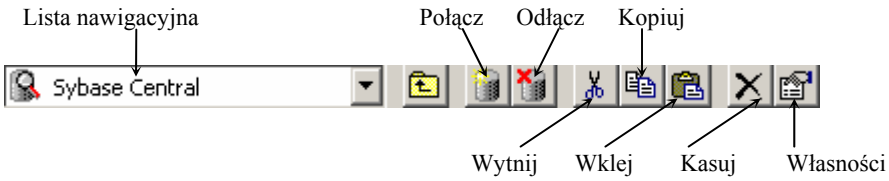
2.2. Administrowanie systemem – Sybase Central

Aplikacja Sybase Central, wchodząca w skład środowiska SQL Anywhere Studio, jest narzędziem graficznym, umożliwiającym zakładanie bazy danych, konstruowanie i modyfikowanie obiektów bazy danych oraz administrowanie serwerem bazodanowym i bazą danych. Główne okno aplikacji przedstawiono na rys. 2.3.



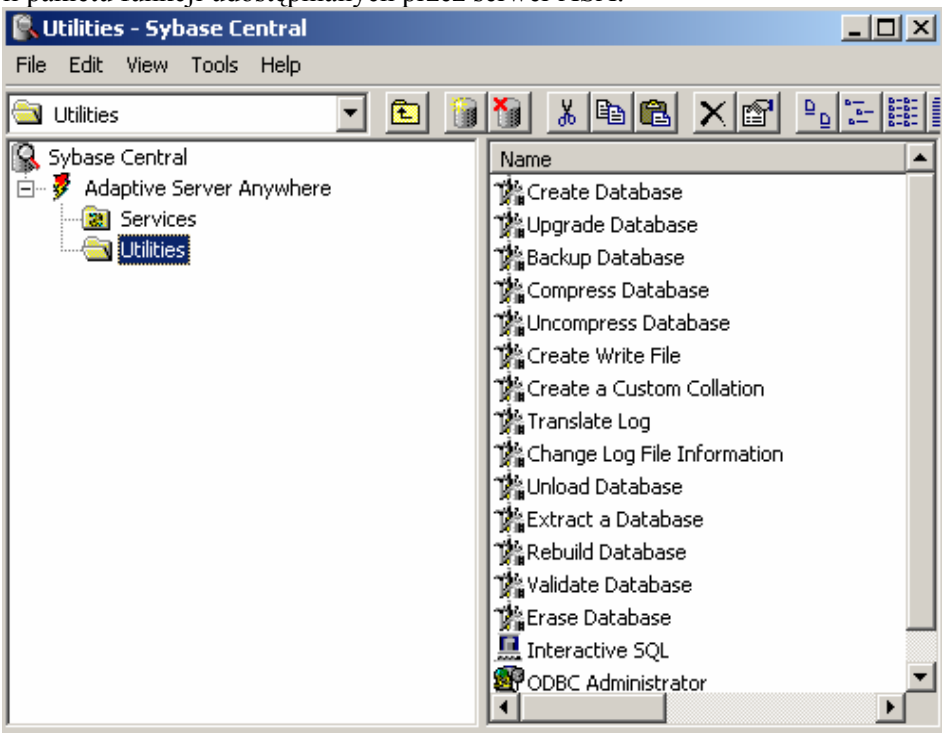
Rys. 2.3. Główne okno aplikacji Sybase Central

Okno jest podzielone na dwa panele. W lewym panelu wyświetlane są nazwy folderów związanych z obiektami bazy danych lub nazwy pakietów usług udostępnianych przez serwer (aplikacje realizujące określone funkcje), układ wyświetlania jest drzewiasty, hierarchiczny. Sybase Central jest korzeniem drzewa. Na niższym poziomie znajduje się Adaptive Server Anywhere i nazwa pakietu aplikacji (*Utilities*). W panelu prawym wyświetlana jest zawartość pakietu lub schemat bazy danych w zależności od wyboru użytkownika. Belka główna oprócz możliwości menu rozwijalnego ma przyciski graficzne umożliwiające wykonanie niektórych standardowych komend. Opis przycisków belki głównej podano na rys. 2.4.



Rys. 2.4. Znaczenie przycisków graficznych belki głównej Sybase Central

Rysunek 2.5 przedstawia rozwinięcie prawego panelu po wybraniu opcji *Utilities*, czyli pakietu funkcji udostępnianych przez serwer ASA.

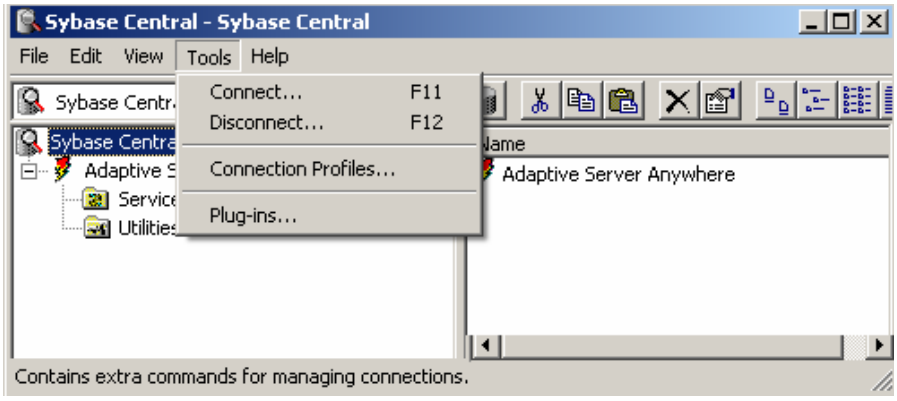


Rys. 2.5. Widok okna Sybase Central z zestawem funkcji

Uruchomienie poszczególnych aplikacji, np. aplikacji realizującej funkcję tworzenia nowej bazy danych (*Create Database*), wykonania kopii bazy danych (*Backup Database*), konfigurowania połączenia z bazą danych (*ODBC Administrator*) czy uruchomienia aplikacji umożliwiającej interaktywną pracę z bazą danych (*Interactive SQL*) odbywa się poprzez kliknięcie myszką na wybraną aplikację i działanie zgodne z menu aplikacji.

Administrowanie konkretną bazą danych można rozpocząć dopiero po nawiązaniu z nią połączenia. W środowisku ASA każda nowo tworzona baza danych jest identyfikowana poprzez ustawienia domyślne: identyfikatora użytkownika jako **DBA** oraz hasła dostępu **SQL**. Oczywiście, w celu zapewnienia odpowiedniego poziomu bezpieczeństwa danych administrator bazy danych powinien dokonać zmiany nazw użytkowników upoważnionych do pracy z bazą danych oraz haseł dostępu. W środowisku Adaptive Server Anywhere znajduje się przykładowa, kompletna baza danych *asademo.db*, będąca zbiorem określonych obiektów, takich jak: tabele wypełnione danymi i odpowiednio ze sobą powiązane, perspektywy, procedury, indeksy, triggery. Znaczenie obiektów nie będących tabelami zostanie szczegółowo omówione w dalszych

rozdziałach (rozdz. 4, 9). Przykłady podane poniżej mają na celu zaprezentowanie możliwości administrowania dowolną bazą danych z poziomu Sybase Central w zakresie operowania na schemacie bazy danych, tworzenia kopii zapasowych, zarządzania pracą użytkowników z wykorzystaniem bazy *asademo.db*. Pierwszym krokiem jest nawiązanie połączenia serwera z bazą danych, które odbywa się poprzez mechanizm ODBC. Z belki Sybase Central należy rozwinąć zakładkę *Tools* i opcję *Connect* (rys. 2.6).



Rys. 2.6. Nawiązywanie połączenia z bazą danych

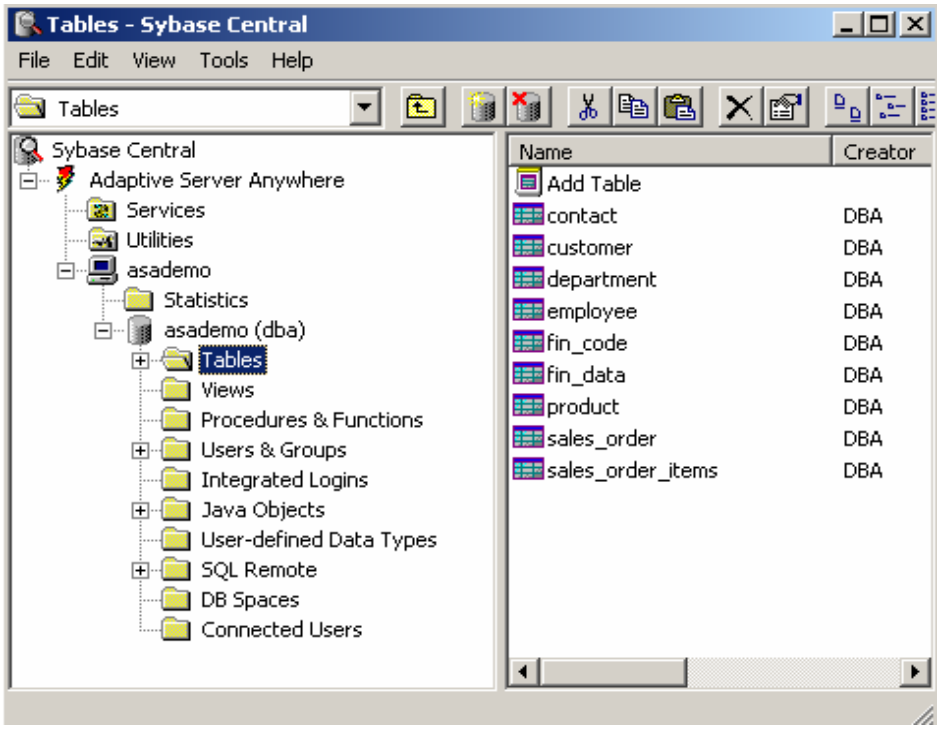
Po wybraniu opcji *Connect* pojawia się okno dialogowe do wyboru odpowiedniego ODBC, które umożliwia połączenie bazy danych z serwerem. Przy łączeniu się z przykładową bazą *asademo.db* obowiązuje nazwa użytkownika *DBA* i hasło *SQL*. Ponieważ działania dotyczą środowiska przykładowego, interfejs ODBC o nazwie *ASA* 7.0 *Sample* jest gotowy do wykorzystania, w przypadku nowo utworzonej bazy taki interfejs należy skonfigurować, wykorzystując funkcję *ODBC Adminstrator* z prawego panelu.

Po przyłączeniu przykładowej bazy danych panel lewy okna głównego Sybase Central ukazuje foldery z obiektami bazy *asademo.db*, które można rozwijać. Standardowe foldery to:

- folder tabel (*Tables*) zawierający tabele wchodzące w skład bazy danych,
- folder perspektyw (*Views*) zawierający tablice wirtualne,
- folder procedur i funkcji (*Procedures & Functions*) zawierający moduły procedur i funkcji związanych z bazą danych,
- folder użytkowników (*Users & Groups*) zawierający nazwy użytkowników i ich zakresy uprawnień do operowania na bazie danych,
- folder obiektów Java (*Java Objects*),
- folder dziedzin (*Domains*) zawierający narzędzia do tworzenia niestandardowych typów danych,
- folder przestrzeni bazy danych (*DB Spaces*) umożliwiający konstruowanie więcej niż jednego pliku *.db* dla jednej bazy danych,

• folder serwerów zdalnych (*Remote Servers*) umożliwiający lokalizację serwerów.

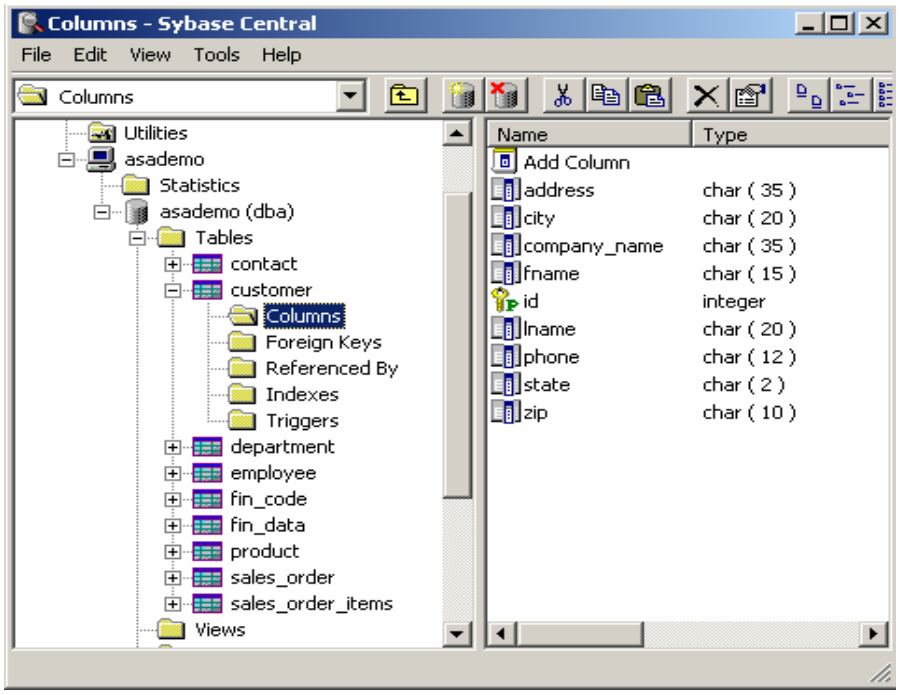
Po rozwinięciu wybranego foldera jego zawartość zostaje wyświetlona w panelu prawym. Rysunek 2.7 ilustruje zawartość okna głównego po przyłączeniu bazy *asademo.db* i rozwinięciu folderu tabel.



Rys. 2.7. Okno główne Sybase Central z przyłączoną przykładową bazą *asademo.db*

W panelu prawym widoczne są nazwy tabel, które wchodzą w skład bazy *asademo.db*. Powyżej listy nazw figuruje nazwa funkcji, która jest udostępniona z tego poziomu, czyli „Dodaj Tabelę" (*Add Table*).

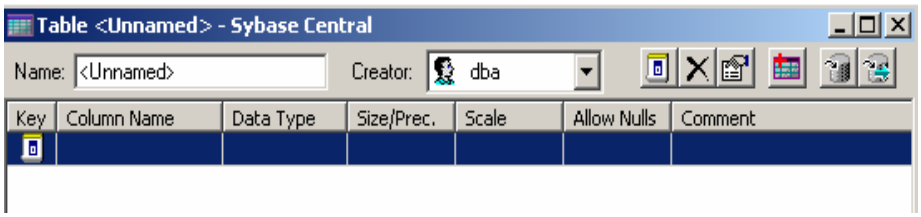
Z każdą tabelą związany jest jej opis również w formie folderów, co oznacza, że dla każdej tabeli istnieje folder kolumn, przechowujący nazwy kolumn wraz z dziedzinami, folder kluczy obcych, więzów integralności podstawowych i dodatkowych (*triggers*) oraz indeksów. Rozwinięcie drzewa folderów następuje w panelu lewym, natomiast wyświetlenie zawartości wybranego folderu w panelu prawym. Rysunek 2.8 obrazuje rozwinięte drzewo folderów dla tabeli o nazwie *customer* w lewym panelu oraz wyświetloną zawartość folderu kolumn w panelu prawym wraz z funkcją udostępnioną na tym poziomie, czyli funkcją „Dodaj Kolumnę" (*Add Column*).



Rys. 2.8. Okno Sybase Central z rozwiniętym folderem kolumn dla tabeli *customer*

Funkcje administrowania bazą danych związane są z operowaniem na obiektach bazy danych (rys. 2.7 i 2.8). Ponieważ pakiet Sybase Central jest narzędziem graficznym, sposób uruchamiania i realizacji poszczególnych funkcji zachowuje standardy środowiska graficznego. Dla zobrazowania sposobu działania pakietu prześledźmy możliwości konstruowania nowych tabel, czyli zmianę struktury bazy danych z wykorzystaniem trybu graficznego udostępnianego przez Sybase Central [23]. W celu założenia nowej tabeli w obszarze bazy danych należy:

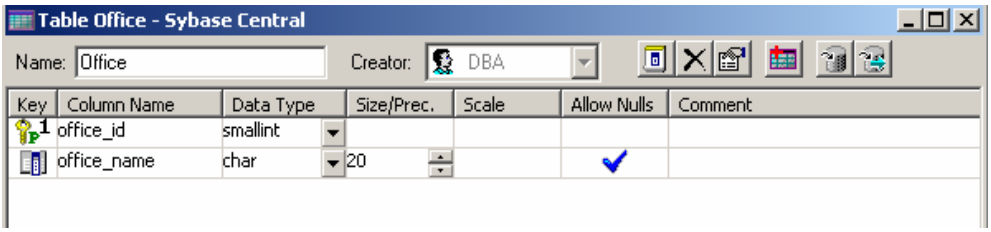
- W lewym panelu otworzyć folder tabel.
- W prawym panelu poprzez podwójne kliknięcie myszką wywołać funkcję *ADD Table*, co powoduje wywołanie graficznego edytora tabel, prezentowanego na rysunku 2.9.



Rys. 2.9. Okno Sybase Central po wywołaniu edytora tabel

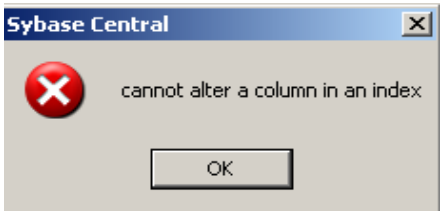
- Nadać tabeli nazwę, którą wpisuje się w polu *Name*.
- Zdefiniować kolumny występujące w dodawanej tabeli poprzez podanie kolejno, wiersz po wierszu, nazwy kolumny (należy pamiętać, że środowisko Sybase nie pozwala na stosowanie spacji w nazwach) typu danych występujących w danej kolumnie (czyli dziedziny). Definiowanie dziedziny jest wspomagane przez pakiet – odbywa się przez wybór typu danych z listy rozwijalnej, na której znajdują się typy danych akceptowane przez serwer ASA. W przypadku niektórych typów danych, np. znakowych, wymagane jest podanie rozmiaru łańcucha, w przypadku liczb dziesiętnych wymagane jest podanie precyzji i skali liczby. Kolejnym krokiem jest określenie, czy w danej kolumnie dozwolone są wartości *null*, czy też kolumna musi być wypełniana danymi. Po zdefiniowaniu kolumny można opatrzyć ją komentarzem, wykorzystując pole *Comment*, co nie jest obowiązkowe; komentarz nie jest zawartością żadnej tabeli, stanowi jedynie ułatwienie w analizie schematu tabeli. Jeśli schemat tabeli jest dokładnie przemyślany, to powinno być z góry wiadomo, która kolumna (lub kolumny) będą pełniły rolę klucza głównego. Kolumny te można zaznaczyć w trybie graficznym na poziomie definicji.

Dla celów ilustracyjnych został zdefiniowany schemat tabeli o nazwie *Office,* z dwiema kolumnami*: office_id*, *office_name*, gdzie kolumna *office_id* pełni rolę klucza głównego tabeli (rys. 2.10).



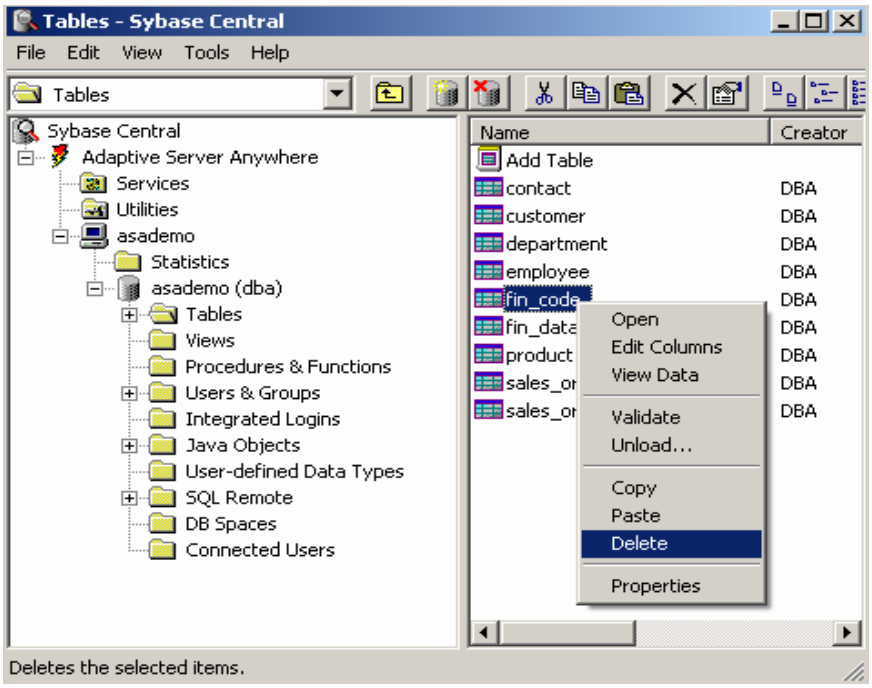
Rys. 2.10. Graficzne definiowanie tabel za pomocą Sybase Central

Oczywiście każdą z tabel bazy danych można ponownie wywołać do edycji poprzez wybranie nazwy tabeli w lewym panelu i uruchomieniu opcji *Edit Columns* w zakładce *File*. Edycja tabeli polega na dodawaniu lub kasowaniu kolumn, zmianie nazw kolumn. Należy jednak zwrócić uwagę na to, że kolumny, które pełnią rolę kluczy głównych i kluczy obcych nie mogą być kasowane, ze względu na rolę, jaką pełnią w bazie danych. W przypadku próby kasowania takich kolumn DBMS wysyła komunikat o błędzie (rys. 2. 11).



Rys. 2.11. Komunikat błędu po próbie kasowania kolumny klucza głównego

W zakres działań związanych z administrowaniem bazy danych wchodzi również kasowanie całych tabel bazy danych. W tym przypadku należy w prawym panelu zaznaczyć tabelę przewidzianą do skasowania i użyć prawego przycisku myszki. Po pojawieniu się menu rozwijalnego wybrać opcję kasowania (*delete*), jak na rys. 2.12.

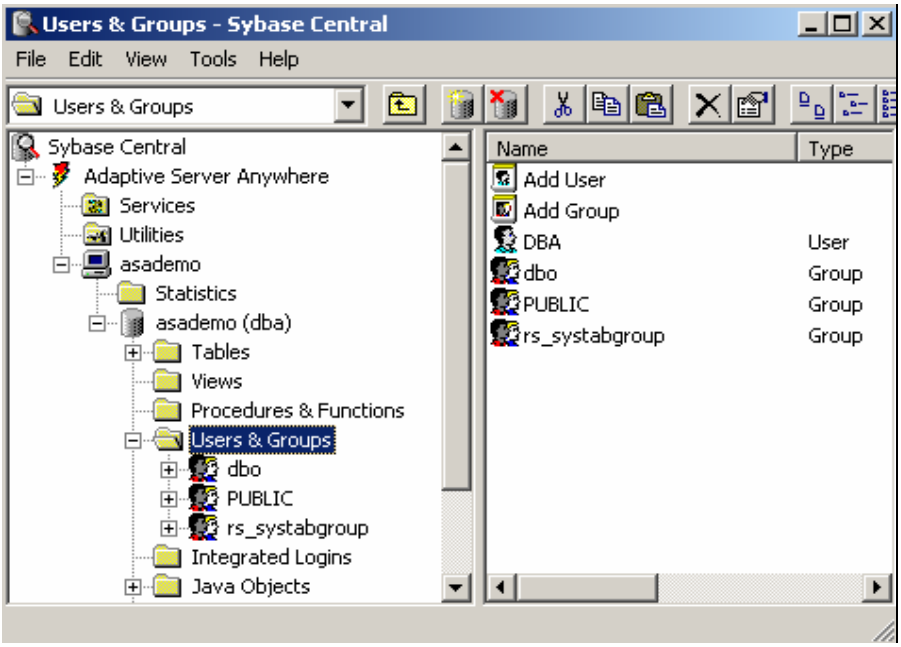


Rys. 2.12. Kasowanie tabeli z bazy danych

Z poziomu narzędzia Sybase Central można administrować wszystkimi obiektami bazy danych, przy czym sposób postępowania jest taki, jak przedstawiono powyżej. Ponieważ inne obiekty bazy danych, takie jak perspektywy (*view*), indeksy (*indexes*), czy procedury pamiętane (*stored procedures*) będą omawiane w kolejnych rozdziałach, metody ich konstruowania czy modyfikacji zarówno za pomocą narzędzia graficznego, jakim jest Sybase Central, jak i metod programowych pojawią się w późniejszych rozdziałach. W niniejszym rozdziale przedstawiono sposób zarządza-

nia pracą z bazą danych użytkowników i grup użytkowników – jest to jedno z ważniejszych zadań administratora bazy danych. Jest rzeczą oczywistą, że z jednej bazy danych, zwłaszcza bazy w architekturze klient–serwer udostępnianej poprzez sieć, korzysta wielu użytkowników. Nie zawsze jednak istnieje potrzeba, aby każdy użytkownik korzystał z całych zasobów i w pełnym zakresie działań. Zadaniem administratora jest przydzielanie odpowiednich uprawnień do korzystania z określonych obiektów bazy danych oraz zakresu dostępnych operacji. Realizacja tego zadania poprzez Sybase Central w odniesieniu do konkretnej bazy danych przebiega w sposób następujący:

- Należy przyłączyć odpowiednią bazę danych.
- W lewym panelu wybrać folder *Users&Groups*. Widok okna głównego przedstawiono na rys. 2.13.



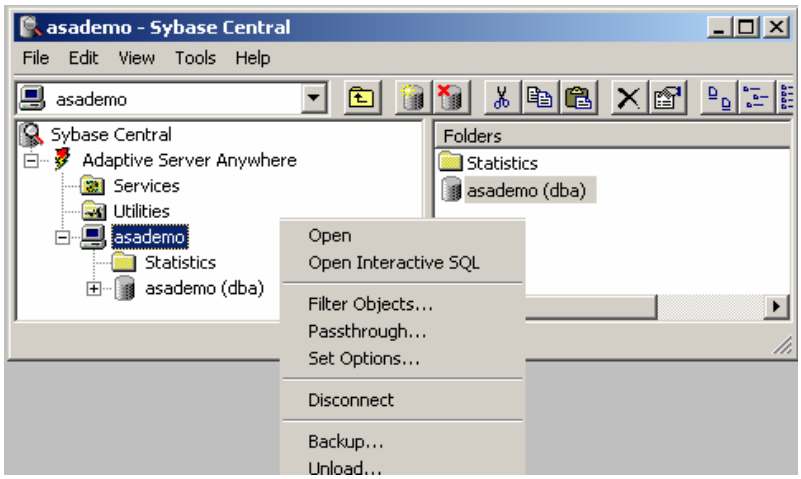
Rys. 2.13. Okno główne Sybase Central dla zarządzania pracą użytkowników bazy danych

- W zależności od tego, czy tworzona ma być grupa użytkowników, czy użytkownik pojedynczy wybieramy odpowiednią funkcję (*Add User*) lub (*Add Group*).
- Uruchamiany zostaje kreator, pracujący w trybie okienkowym, umożliwiający utworzenie grup użytkowników lub użytkowników pojedynczych. Każda grupa lub użytkownik musi mieć swoją nazwę i hasło dostępu do bazy danych, jak również określone prawa dostępu (prawa administratora DBA, czyli pełne uprawnienia, lub też prawa do korzystania z zasobów bazy w charakterze użytkownika). Po poprawnie

wykonanej sekwencji czynności w panelu lewym pojawia się nazwa utworzonej grupy użytkowników, natomiast nazwa pojedynczego użytkownika będzie figurować w panelu prawym.

• Dodawanie użytkowników do grup odbywa się poprzez wkopiowanie nazwy użytkownika z panelu prawego do nazwy grupy w panelu lewym z użyciem menu rozwijalnego, wywoływanego poprzez prawy klawisz myszki i wybór odpowiednio opcji *copy* i *paste*.

Administrator bazy danych w zakresie swoich obowiązków wykonuje również kopie zapasowe bazy danych (rys. 2.14). Istnieją dwa sposoby wykonania z poziomu Sybase Central kopii bazy danych. Sposób pierwszy wiąże się z zakończeniem pracy z przyłączoną bazą danych, czyli w przykładzie bazą *asademo.db*. W przypadku bazy przyłączonej należy zaznaczyć kliknięciem prawym klawiszem myszki bazę, której kopia zapasowa będzie tworzona, co powoduje pojawienie się menu rozwijalnego, z którego należy wybrać opcję *Backup Database* i dalej postępować zgodnie ze wskazówkami aplikacji.



Rys. 2.14. Wykonywanie kopii zapasowej bazy przyłączonej

Drugi sposób polega na rozwinięciu foldera *Utilities* i wywołaniu aplikacji *Backup Database* w prawym panelu. Po uruchomieniu aplikacji w dialogowym trybie okienkowym administrator wskazuje, jaka baza będzie kopiowana i do jakiego katalogu.

Praktyczne wprowadzenie do języka SQL

Język SQL (*Structured Query Language*) – strukturalny, nieproceduralny język zapytań został początkowo zaprojektowany jako język, w którym można było jedynie realizować zapytania do baz danych. Obecnie jednak jest czymś więcej niż tylko językiem zapytań, jest to de facto standardowy język baz danych [13, 16].

Koncepcja języka pojawiła się razem z koncepcją relacyjnych baz danych w laboratorium badawczym IBM w San Jose, gdzie w latach siedemdziesiątych pracował E.F. Codd i gdzie powstał prototypowy system relacyjny – System R. System R używał języka SEQUEL (*Structured English Query Language*). Pomimo wielu publikacji na temat Systemu R, jakie ukazały się w latach siedemdziesiątych, IBM nie zdyskontował wyników badawczych w obszarze zastosowań komercyjnych. Pierwsze implementacje systemów relacyjnych z zastosowaniem języka SQL wykonała korporacja ORACLE. Nieco później pojawiły się rozwiązania firmy INGRES z językiem QUEL, który wprawdzie był bardziej „strukturalny" niż SQL, ale mniej zbliżony do języka angielskiego. Stopniowo na rynku zaczęły pojawiać się rozwiązania komercyjne innych producentów, m.in. firmy IBM. W roku 1982 organizacja ANSI (*American National Standards Institute*) oraz nieco później ISO (*International Standards Organisation*) rozpoczęły prace nad standardem języka relacyjnych baz danych. W roku 1987 opublikowany został dokument przedstawiający definicję standardu SQL, znanego jako SQL1. Standard został jednak bardzo mocno skrytykowany przez znaczących specjalistów z dziedziny baz danych, takich jak E.F. Codd czy C. Date. Główne zarzuty dotyczyły pominięcia tak istotnych zagadnień, jak integralność referencyjna czy sposób konstruowania zapytań – w opublikowanym standardzie to samo zapytanie mogło być konstruowane na wiele różnych sposobów. W odpowiedzi na tę krytykę powstały uzupełnienia wydawane w formie dodatków; pełna specyfikacja standardu znanego jako SQL2 ukazała się w roku 1992. W standardzie tym wyróżnione są dwa podzbiory: minimalny, czyli SQL1 z uwzględnieniem cech integralności oraz poziom pośredni związany z przetwarzaniem transakcyjnym. Kolejne rozszerzenia standardu dotyczące cech obiektowych w relacyjnych bazach danych zostały opublikowane w roku 1999 jako SQL3.

Przedstawienie rysu historycznego miało na celu uzmysłowienie, że w chwili obecnej istnieją trzy standardy SQLa, co samo w sobie jest zaprzeczeniem idei standa-

ryzacji W związku z taką sytuacją pojawiło się pojęcie *dialektu* SQL, czyli sposobu i zakresu implementacji standardu w konkretnym rozwiązaniu komercyjnym. Najczęściej w systemach komercyjnych zaimplementowane są podstawy sprecyzowane w SQL1, rozszerzane dość dowolnie o własne konstrukcje, nie związane z żadnym ze standardów.

3.1. Pojęcia podstawowe

Podstawą teoretyczną języka SQL jest algebra relacyjna i rachunek relacyjny, prezentowane mniej lub bardziej formalnie w podręcznikach z obszaru relacyjnych baz danych, przykładowo [9, 20]. W rozdziale niniejszym omówiono praktyczne możliwości języka SQL (bez wprowadzania formalizmów matematycznych), a ściślej dialektu środowiska *Sybase* [23], w zakresie najistotniejszych dla użytkownika funkcji, czyli:

- zakładania bazy danych i obiektów bazy danych,
- wykonywania operacji wstawiania danych do relacji, modyfikowania i kasowania danych,
- wykonywania zapytań do bazy danych, czyli wyszukiwania danych w określonych konfiguracjach.

Język SQL jest językiem relatywnie prostym, w którym użytkownik specyfikuje przy użyciu komend o składni zbliżonej do prostych zdań w języku angielskim „co" a nie „jak" wykonać w odniesieniu do bazy danych. Według standardów składa się z dwóch głównych komponentów:

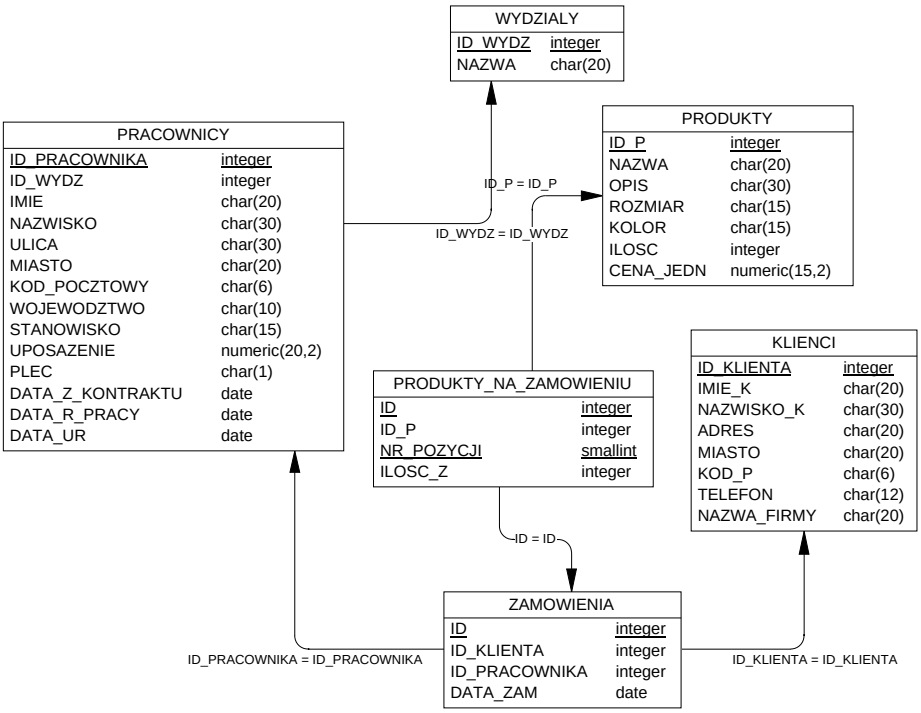
- Języka definicji danych – DDL (*Data Definition Language*), umożliwiającego definiowanie struktury bazy danych oraz sterowanie dostępem do danych.
- Języka manipulacji danymi – DML (*Data Manipulation Language*), umożliwiającego wyszukiwanie i aktualizowanie danych.

W tym rozdziale, wychodząc z założenia, że języka SQL najlepiej uczyć się na przykładach praktycznych, przedstawiono interaktywny sposób pracy z bazą danych, z wykorzystaniem aplikacji (edytora SQL) *Interactive SQL* (*ISQL*), udostępnianej z poziomu pakietu *Sybase Central*. Wszystkie komendy języka SQL odnosić się będą do przykładowej bazy danych *Fitness2.db*. Baza danych *Fitness2.db* opracowana została na potrzeby małej hurtowni odzieży sportowej w zakresie obsługi zamówień. W tablicach bazy danych przechowywane są informacje dotyczące struktury organizacyjnej hurtowni, pracowników, klientów oraz zamówień. Opis wycinka rzeczywistości, dla którego zaprojektowana została baza danych, można scharakteryzować następująco: hurtownia „Fitness" dostarcza na zamówienia swoich klientów określone ilości różnego asortymentu odzieży sportowej. Dane klientów są przechowywane w bazie. Zamówienie, składające się z jednej lub kilku pozycji, jest realizowane przez

konkretnego pracownika z działu sprzedaży. Informacje o dostępnym asortymencie produktów są również przechowywane w bazie danych. Hurtownia składa się z kilku działów organizacyjnych związanych z konkretnym rodzajem pracy, z każdym z działów związana jest określona grupa pracowników. Dane dotyczące pracowników i wydziałów są również przechowywane w bazie danych, w odpowiednich tabelach. W skład bazy *Fitness2.db* wchodzą następujące tabele:

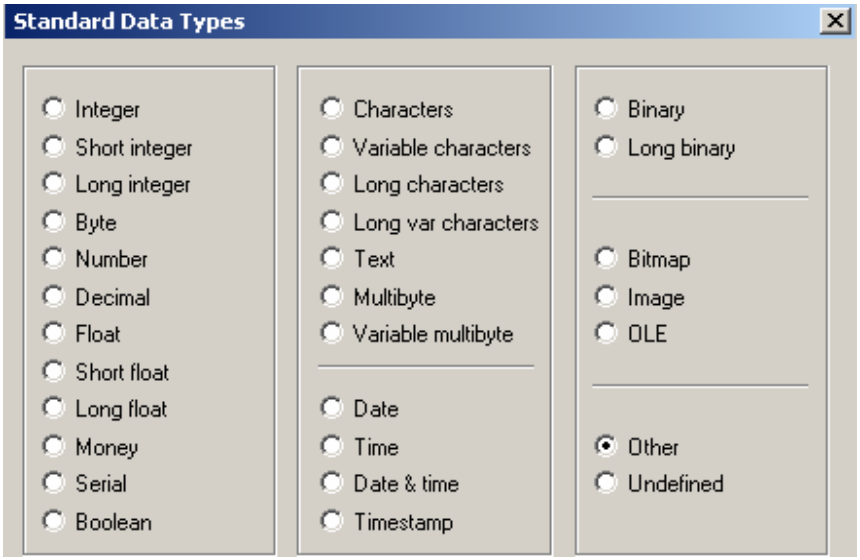
PRACOWNICY (ID_pracownika, Id_wydz, Imię, Nazwisko, Ulica, Miasto, Kod_pocztowy, Województwo, Stanowisko, Uposażenie, Płeć, Data_r_pracy, Data_z_pracy, Data_ur)
WYDZIAŁY (Id_Wydz, Nazwa_wydz)
KLIENCI (Id_klienta, Imię_k, Nazwisko_k, Adres, Miasto, Kod_p, Telefon, Nazwa_firmy)
PRODUKTY (Id_p, Nazwa, Opis, Rozmiar, Kolor, Ilość, Cena_jedn)
ZAMÓWIENIA (Id, Id_klienta, Id_pracownika, Data_zam)
PRODUKTY_NA_ZAMÓWIENIU(Id, Nr_pozycji, Id_p, Ilosc_z)

Tabele bazy danych *Fitness2.db* połączone są ze sobą przez mechanizm klucza obcego, przedstawiony w rozdziale 1. Graficzny obraz schematu bazy danych przedstawiono na rys. 3.1.



Rys. 3.1. Graficzna reprezentacja schematu bazy danych *Fitness2.db*

Schemat graficzny pokazany na rys. 3.1 oprócz obiektów bazy danych, takich jak tabele, kolumny i powiązania pokazuje również typy danych określone dla każdej kolumny tabel. Standardowe typy danych dopuszczalne w środowisku *Sybase* [23] podano na rys. 3.2. Wybór typu danych dla określonej kolumny dokonywany jest oczywiście w momencie definiowania schematu bazy danych – decyzje o tym, jakiego typu dane będą przechowywane w kolumnie podejmuje projektant bazy danych, po dokonaniu analizy środowiska użytkownika, dla którego projektowana jest baza danych.



Rys. 3.2. Standardowe typy danych w środowisku *Sybase*

Przed prezentacją zdań i klauzul w języku SQL, który wprawdzie nie ma ściśle zdefiniowanego formatu pisania komend, warto zapoznać się z kilkoma zasadami poprawiającymi czytelność zapisu oraz notacją przyjętą w zapisie składni ogólnej poszczególnych zdań SQL:

- Każda klauzula zdania SQL powinna zaczynać się w nowej linii.
- W przypadku pisania kilku zdań SQL w dialekcie *Sybase* oddziela się je terminatorem, którym jest znak średnika.
- Wszystkie dane nienumeryczne (znakowe, daty) muszą być ujmowane w zdaniach SQL w apostrofy, tzn. ‘.
- W języku SQL nie ma znaczenia wielkość liter używana do pisania zarówno poleceń, jak i nazw obiektów bazy danych, czytelniejszy jest jednak zapis, gdy słowa stanowiące składnię polecenia pisane są z dużych liter.

Notacja dla ogólnych definicji zdań SQL:

- Litery duże reprezentują słowa składni.

- Litery pisane kursywą reprezentują słowa definiowane przez użytkownika.
- Kreska pionowa oznacza alternatywny wybór, np. | y | z | x |.
- Nawiasy klamrowe oznaczają element wymagany, np. {V}.
- Nawiasy kwadratowe oznaczają element opcjonalny, np. [s].

3.2. Język Manipulacji Danymi

W podrozdziale tym zostaną omówione podstawowe zdania z grupy DML (*Data Manipulation Language*), czyli:

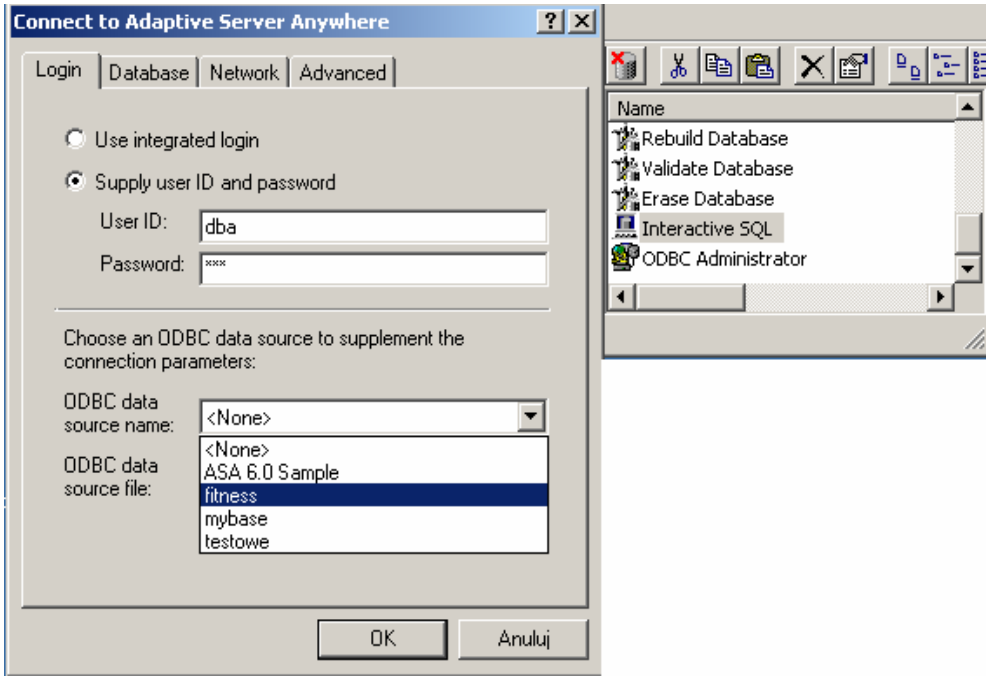
SELECT – wyszukiwanie danych,
INSERT – wstawianie danych do tabel,
UPDATE – aktualizacja danych,
DELETE – kasowanie danych.

Składnia ogólna zdania SELECT

SELECT	[DISTICT | ALL] {*nazwy kolumn* [AS *nowe nazwy*]]}
FROM	*Nazwa tablicy* [*alias*]
[WHERE	*Warunek*]
[GROUP BY	*Lista kolumn*] [HAVING *warunek*]
[ORDER BY	*Lista kolumn*]

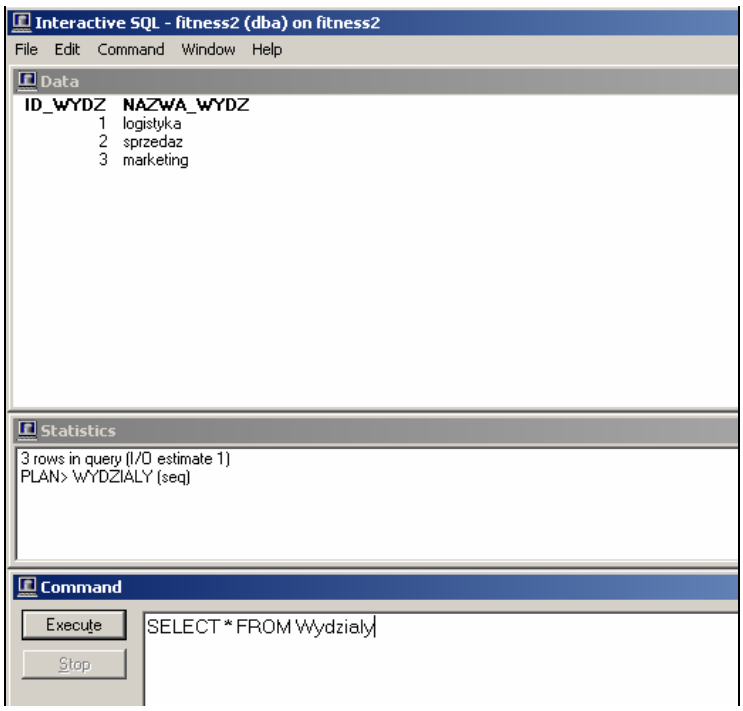
Analizując składnię polecenia wyszukiwania należy zauważyć, że obowiązkowe są jedynie dwie składowe zdania: SELECT (wybierz wiersze z tabeli, wyszukaj wiersze) i FROM (skąd, z jakiej tabeli), pozostałe klauzule są opcjonalne. Klauzula WHERE filtruje wiersze według określonego warunku wyboru, GROUP BY tworzy grupy wierszy z tą samą wartością danych w kolumnie, HAVING podobnie jak WHERE filtruje dane według sprecyzowanego warunku wyboru (tę klauzulę stosuje się tylko do grup wierszy), ORDER BY określa sposób uporządkowania danych wynikowych. Należy pamiętać, że operacja selekcji jest operacją działającą na tabeli i w wyniku jej działania otrzymuje się również tabelę. Warianty użycia zdania SELECT zostaną zilustrowane przykładami, odnoszącymi się do przedstawionej wcześniej bazy danych *Fitness2.db* z wykorzystaniem aplikacji *ISQL*.

Po wywołaniu edytora *ISQL* z poziomu *Sybase Central* należy połączyć się z wybraną bazą danych poprzez okno dialogowe, wykorzystując skonfigurowane wcześniej łącze ODBC (rys. 3.3). Przy połączeniu z bazą danych logujemy się jako użytkownik o nazwie *dba* z hasłem *sql*.



Rys. 3.3. Łączenie z bazą danych *Fitness2.db*

Po prawidłowym połączeniu z bazą danych aplikacja *ISQL* udostępnia ekran do pracy (rys. 3.4). Ekran podzielony jest na trzy części: w części dolnej można wpisywać polecenia SQL i uruchamiać je klawiszem *Execute*, wyniki wyświetlane są w części górnej ekranu, natomiast systemowe dane statystyczne (np. czas wykonania polecenia) wyświetlane są w części środkowej ekranu. Podczas sesji można wykorzystywać standardowe funkcje aplikacji dostępne poprzez menu belki głównej, takie jak zapamiętywanie przebiegu sesji w pliku z rozszerzeniem **.sql* (menu *File*, opcja *Save*), uruchamianie plików z rozszerzeniem **.sql* (menu *File*, opcja *Open*), standardowe możliwości edycyjne (menu *Edit*), ponowne wywołanie wykonywanego uprzednio zdania SQL (menu *Command*, opcja *Recall*). Poprzez menu *Command* można również dokonać przyłączenia i odłączenia bazy danych (opcje *Connect* i *Disconnect*). Schemat bazy danych (nazwy tabel oraz kolumn w tabelach) udostępnia funkcja *Insert* w menu *Edit*. Funkcja ta jest bardzo pomocna w przypadku pracy interaktywnej, gdyż umożliwia wkopiowywanie do zdań SQL pisanych w oknie poleceń nazw tabel i kolumn. W przypadku korzystania z tej funkcji nazwy tabel zawsze są poprzedzone przedrostkiem „DBA", przykładowo; tabela KLIENCI będzie kopiowana w formacie „DBA".KLIENCI.



Rys. 3.4. Okno edytora *ISQL*

Rysunek 3.4 ilustruje również użycie polecenia SQL umożliwiającego wybór wszystkich wierszy z tabeli. W powyższym przykładzie znak „*” interpretowany jest jako polecenie wyboru pełnego zestawu kolumn z tablicy *Wydziały*.

3.2.1. Układanie zapytań

Wybór określonych kolumn z tabeli (projekcja)

Przykładowo – użytkownik potrzebuje listę alfabetyczną pracowników w układzie: nazwisko, imię, data urodzenia. Listę można utworzyć, uruchamiając wykonanie zdania SQL:

SELECT *Nazwisko, Imie, Data_ur*
FROM *Pracownicy*
ORDER BY *Nazwisko*

W wyniku wykonania takiego polecenia utworzona zostanie lista wszystkich pracowników w kolejności nazwisk od A do Z. Zastosowanie klauzuli ORDER BY powoduje sortowanie danych wynikowych w kierunku rosnącym (opcja ASC – *ascen-*

ding), jeżeli wymagany jest kierunek malejący, należy po nazwie kolumny, według której odbywa się sortowanie, wpisać DESC (*descending – malejąco*).

Wynik działania polecenia SQL:

Nazwisko	Imię	Data_ur
Adamska	Ewa	1963-10-27
Bielska	Janina	1958-07-15
Bielski	Leszek	1955-12-04
Burski	Adam	1975-04-10
Frankowski	Jerzy	1973-10-23
Gawron	Anna	1976-03-03
Kowalski	Piotr	1973-11-06
Mirski	Tadeusz	1975-08-12
Nawrot	Kamila	1950-02-07
Nowak	Jan	1965-12-23
Pakulski	Damian	1974-06-14
Porada	Maria	1971-05-09
Wirski	Jakub	1969-05-13
Wyzga	Anatol	1959-11-09

Działanie klauzuli DISTINCT

Zastosowanie klauzuli DISTINCT w zdaniu SQL powoduje eliminowanie wartości powtarzających się. Działanie tej klauzuli zilustrowano w tabeli PRODUKTY, w której przechowywane są dane dotyczące asortymentu produktów oferowanych przez hurtownię „Fitness". Polecenie wybierające całą zawartość tabeli PRODUKTY:

SELECT * FROM *PRODUKTY*

ID_P	Nazwa	Opis	Rozmiar	Kolor	Ilość	Cena_jedn
10	t-shirt	damski	M	czarny	200	10.00
11	t-shirt	damski	L	czerwony	200	10.00
12	t-shirt	męski	XL	czarny	150	15.00
13	bluza	polar	Bez wym	szary	250	20.00
14	bluza	bawełna	Bez wym	szary	250	20.00
15	dres	damski	S	granatowy	80	45.00
16	dres	męski	L	czarny	50	50.00
17	szorty	damskie	S	Czarny	120	12.50

Lista nazw produktów utworzona poprzez zastosowanie zdania SELECT bez klauzuli DISTINCT i z klauzulą DISTINCT:

SELECT *Nazwa*
FROM *PRODUKTY*

SELECT DISTINCT *Nazwa*
FROM *PRODUKTY*

Wynik działania zdania SELECT bez klauzuli DISTINCT:

Nazwa
t-shirt
t-shirt
t-shirt
bluza
bluza
dres
dres
szorty

Wynik działania zdania SELECT z klauzulą DISTINCT:

Nazwa
t-shirt
bluza
dres
szorty

Zapytania z zastosowaniem wyliczeń

W treści zapytania do bazy danych mogą być umieszczane obliczenia, na przykład, jeżeli w tabeli PRACOWNICY w jednej z kolumn przechowywane są dane dotyczące uposażenia miesięcznego każdego z pracowników, a potrzebna jest wielkość rocznych dochodów pracowników, to można zastosować konstrukcję:

SELECT *Id_Pracownika, Nazwisko, Imię, Uposażenie*12*
FROM *PRACOWNICY*

Wynik działania takiego zdania SQL (ostatnia kolumna, której nazwa tworzona jest przez system poprzez nazwę własną oraz wykonane wyliczenie, zawiera wartości wyliczone; należy pamiętać, że rzeczywista zawartość kolumny *Uposażenie* nie zmienia się, dalej zapisane są w niej zarobki miesięczne):

Id_pracownika	Nazwisko	Imię	Uposażenie*12
100	Bielski	Leszek	36000.00
120	Wyzga	Anatol	33600.00
101	Adamska	Ewa	30000.00
112	Bielska	Janina	30000.00
103	Kowalski	Piotr	21600.00
122	Burski	Adam	24000.00
126	Frankowski	Jerzy	24000.00
125	Gawron	Anna	18000.00
106	Mirski	Tadeusz	24000.00
104	Wirski	Jakub	25200.00
117	Nawrot	Kamila	27600.00
115	Porada	Maria	24000.00
116	Nowak	Jan	30000.00
109	Pakulski	Damian	20400.00

W zdaniach SQL można umieszczać wyliczenia polegające na dodawaniu, odejmowaniu, mnożeniu i dzieleniu – oczywiście w odniesieniu do kolumn, które zawierają wartości liczbowe. W wyliczeniach może wystąpić więcej niż jedna kolumna; przykładowo, w tabeli PRODUKTY pamiętane są w kolumnie *Cena_jedn* ceny jednostkowe poszczególnych pozycji oraz w kolumnie *Ilość* – ilość produktu na stanie. Po pomnożeniu obu kolumn przez siebie otrzymamy wartość całkowitą całego zapasu każdego z produktów:

SELECT *Id_P, nazwa, ilosc*cena_jedn*
FROM *PRODUKTY*

ID_P	Nazwa	Ilość * Cena_jedn
10	t-shirt	2000.00
11	t-shirt	2000.00
12	t-shirt	2250.00
13	bluza	5000.00
14	bluza	5000.00
15	dres	3600.00
16	dres	2500.00
17	szorty	1500.00

W przypadkach układania zapytań z wyliczeniami często wykorzystuje się możliwość stosowania tzw. nazw zastępczych (aliasów) dla kolumn zawierających wyniki wyliczeń. Możliwość nadania nazwy aliasowej zapewnia klauzula AS:

SELECT *Id_P*, *nazwa, ilosc*cena_jedn* AS *Wartość_towaru*
FROM *PRODUKTY*

ID_P	Nazwa	Wartość_towaru
10	t-shirt	2000.00
11	t-shirt	2000.00
12	t-shirt	2250.00
13	bluza	5000.00
14	bluza	5000.00
15	dres	3600.00
16	dres	2500.00
17	szorty	1500.00

Podane przykłady konstrukcji zdań SQL dotyczyły sytuacji, gdy z tabel wybierane były wszystkie wiersze. Nie zawsze jednak zachodzi taka potrzeba. Klauzula WHERE umożliwia wyszukiwanie wierszy, które spełniają określone kryteria. Do konstruowania kryteriów wyszukiwania używa się:

- Operatorów porównania
  = równy, > większy,
  < > nie równy, <= mniejszy równy,
  < mniejszy, >= większy równy.
- Operatorów logicznych
  AND, OR, NOT, w odniesieniu do których obowiązują następujące reguły:
  – wyrażenia w kryterium wyboru są przeliczane od strony lewej do prawej,
  – jeżeli w kryterium wyboru znajdują się podwyrażenia w nawiasach, są one wyliczane pierwsze,
  – warunki z operatorami NOT są wyliczane przed warunkami z operatorami AND i OR,
  – warunki z operatorami AND są wyliczane przed OR.
- Operatorów SQL
  BETWEEN AND, IN/NOT IN, LIKE/NOT LIKE, IS NULL/IS NOT NULL.

Zapytania z zastosowaniem operatorów porównania

Generowanie listy pracowników, których uposażenie miesięczne jest mniejsze niż 2000,00 zł.

SELECT *Id_pracownika, Nazwisko, Imię, Uposażenie*
FROM *PRACOWNICY*
WHERE *Uposażenie* < *2000*

Id_pracownika	Nazwisko	Imię	Uposażenie
103	Kowalski	Piotr	1800.00
125	Gawron	Anna	1500.00
109	Pakulski	Damian	1700.00

Zapytania z zastosowaniem operatorów logicznych

Za pomocą operatorów logicznych konstruowane są złożone kryteria wyszukiwania. Przykładowo, lista produktów w rozmiarze „s" lub „xl":

SELECT *
FROM *PRODUKTY*
WHERE *Rozmiar* = 's' OR *Rozmiar* = 'xl'

ID_P	Nazwa	Opis	Rozmiar	Kolor	Ilość	Cena_jedn
12	t-shirt	męski	XL	czarny	150	15.00
15	dres	damski	S	granatowy	80	45.00
17	szorty	damskie	S	czarny	120	12.50

Zapytania z zastosowaniem operatorów SQL

• Zastosowanie operatora BETWEEN/NOT BETWEEN
Lista produktów, których cena zawarta jest między 10 zł a 15 zł

SELECT *Id_p, Nazwa, Opis, Ilość, Cena_jedn*
FROM *PRODUKTY*
WHERE *Cena_jedn* BETWEEN *10* AND *15*

ID_P	Nazwa	Opis	Ilość	Cena_jedn
10	t-shirt	damski	200	10.00
11	t-shirt	damski	200	10.00
12	t-shirt	męski	150	15.00
17	szorty	damskie	120	12.50

Użycie operatora SQL BETWEEN AND jest równoważne z zastosowaniem dwóch operatorów porównania:

SELECT *Id_p, Nazwa, Opis, Ilość, Cena_jedn*
FROM *PRODUKTY*
WHERE *Cena_jedn* >=*10* AND *Cena_jedn* <= *15*

W podanym przykładzie warto zwrócić uwagę na przewagę stosowania operatorów SQL nad operatorami matematycznymi – składnia zdania SELECT jest zdecydowanie prostsza w pierwszym przypadku.

• Zastosowanie operatora IN/NOT IN
Lista pracowników pracujących na stanowiskach kierowniczych lub samodzielnych

SELECT *Nazwisko, Imię, Stanowisko*
FROM *PRACOWNICY*
WHERE *Stanowisko* IN (‘*kierownik*’, ‘*specjalista*’, ‘*menedżer*’)

Nazwisko	Imię	Stanowisko
Bielski	Leszek	kierownik
Wyzga	Anatol	menedżer
Adamska	Ewa	specjalista
Bielska	Janina	specjalista
Nowak	Jan	menedżer

Podobnie w tym przypadku konstrukcja z operatorem IN może być zastąpiona przez:

SELECT *Nazwisko, Imię, Stanowisko*
FROM *PRACOWNICY*
WHERE *Stanowisko* = ‘kierownik’
OR *Stanowisko* = ‘menedżer’
OR *Stanowisko* = ‘specjalista’

• Kryterium wyboru z zastosowaniem wzorca wyszukiwania – operator LIKE/NOT LIKE

Znakiem używanym do tworzenia wzorca wyszukiwania jest „%", znak ten reprezentuje dowolną sekwencję znaków w łańcuchu, natomiast podkreślenie, czyli „_" oznacza dowolny pojedynczy znak w łańcuchu. Przykłady wzorców:

'B%' – oznacza, że szukany jest łańcuch znaków rozpoczynający się na literę B,

'B___' – oznacza, że szukany jest łańcuch o długości 4 znaków, rozpoczynający się na literę B,

'%B' – oznacza, że szukany jest łańcuch znaków o długości przynajmniej = 1, z literą B na końcu,

'%B% – oznacza, że szukany jest łańcuch znaków zawierający na dowolnej pozycji literę B.

Lista nazwisk, imion i dat urodzenia pracowników, których nazwiska zaczynają się na literę B:

SELECT *Nazwisko, Imię, Data_ur*
FROM *PRACOWNICY*
WHERE *Nazwisko*
LIKE '*B%*'

Nazwisko	**Imię**	**Data_ur**
Bielska	Janina	1958-07-15
Bielski	Leszek	1955-12-04
Burski	Adam	1975-04-10

Wszystkie wymienione operatory SQL (BETWEEN, IN, LIKE) mogą być użyte w składni zdania z zaprzeczeniem NOT; gdyby w poleceniu prezentowanym powyżej użyte zostało zaprzeczenie NOT, oznaczałoby to wybór pracowników, których nazwiska nie zaczynają się na literę B:

SELECT *Nazwisko, Imię, Data_ur*
WHERE *Nazwisko*
FROM *PRACOWNICY*
NOT LIKE '*B%*'

3.2.2. Funkcje agregujące (agregaty)

Funkcje agregujące operują na pojedynczych kolumnach i w wyniku działania zwracają pojedyncze wartości. Wywołania funkcji występują tylko w zdaniu SELECT. W standardzie SQL zdefiniowanych jest pięć funkcji agregujących:

COUNT – zwraca liczbę wartości w wyspecyfikowanej kolumnie,

SUM – zwraca sumę wartości w wyspecyfikowanej kolumnie,

AVG – zwraca wartość średnią z wyspecyfikowanej kolumny,

MIN – zwraca wartość najmniejszą występującą w wyspecyfikowanej kolumnie,
MAX – zwraca wartość największą występującą w wyspecyfikowanej kolumnie.

Funkcje SUM i AVG operują na kolumnach zawierających wyłącznie wartości liczbowe, funkcje MIN i MAX na kolumnach zawierających wartości liczbowe i daty. Każda z funkcji agregujących, z wyjątkiem COUNT(*), eliminuje wartości NULL i działa na pozostałych wartościach. Jeśli zachodzi potrzeba wyeliminowania duplikatów wartości, to przed wywołaniem funkcji należy zastosować klauzulę DISTINCT. Funkcja COUNT(*) jest specyficzną postacią funkcji COUNT – zlicza wszystkie wiersze w tabeli. Umiejętne zastosowanie funkcji agregujących umożliwia tworzenie zestawień statystycznych na podstawie danych zapisanych w bazie. Przykłady zastosowań funkcji agregujących:

- Liczba klientów hurtowni; w tym przypadku jest to liczba wierszy w tabeli KLIENCI, ponieważ każdy z wierszy reprezentuje konkretnego klienta hurtowni.

SELECT COUNT (*) AS *Liczba_ Klientów*
FROM *KLIENCI*

Liczba_Klientów
15

- Liczba klientów „aktywnych” w lutym 2003 roku; należy zauważyć, że klienci aktywni to tacy klienci, którzy składali zamówienie. Dane odnośnie do składanych zamówień znajdują się w tabeli ZAMÓWIENIA (Id, Id_klienta, Id_pracownika, Data_zam), w której konkretne zamówienie jest skojarzone z klientem składającym. Ponieważ każdy z klientów hurtowni może składać dowolną liczbę zamówień, a chcemy ustalić, ilu klientów było faktycznie aktywnych, a nie ile razy złożyli zamówienie, w celu wyeliminowania powtórzeń należy zastosować klauzulę DISTINCT.

SELECT COUNT (DISTINCT *Id_klienta*) AS *Aktywni*
FROM ZAMÓWIENIA
WHERE *Data_zam* BETWEEN ‘*03-02-01*’ AND ’*03-02-28*’

Aktywni
4

- Liczba osób pracujących na stanowisku specjalisty oraz łączna kwota ich zarobków

SELECT COUNT(*Id_pracownika*) AS *l_specjalistów*, SUM(*uposażenie*) AS *suma_zarobków*
FROM *PRACOWNICY*
WHERE *Stanowisko* = ‘*specjalista*’

l_specjalistów	**suma_zarobków**
2	5000.00

• Zestawienie zarobków minimalnych, maksymalnych i średniej płacy

SELECT MIN(*Uposażenie*) AS *płaca*_najniższa, MAX(Uposażenie) AS płaca_najwyższa, AVG(Uposażenie) AS *średnie_zarobki*
FROM *PRACOWNICY*

płaca_najniższa	**płaca_najwyższa**	**średnie_zarobki**
1500.00	3000.00	2192.86

Funkcje agregujące mogą być stosowane nie tylko, jak obrazują powyższe przykłady, do operowania na wszystkich wartościach w kolumnie. Poprzez zastosowanie klauzuli GROUP BY można spowodować operowanie na danych pogrupowanych wokół jednej wartości. Wszystkie podsumowania podawane w wyniku będą dotyczyły grup. Przykładowo, wszyscy pracownicy hurtowni związani są z wydziałami, na których pracują. Jeżeli potrzebne byłoby zestawienie dotyczące płac (najniższa płaca, najwyższa płaca, średnie zarobki) w rozbiciu na poszczególne wydziały, należy użyć polecenia SQL z funkcjami sumarycznymi i klauzulą GROUP BY:

SELECT *Id_wydz*, COUNT(*Id_pracownika*) AS *L_prac*, MIN(*Uposażenie*) AS *Płaca_najniższa*, MAX(*Uposażenie*) AS *Płaca_Najwyższa*, AVG(*Uposażenie*) AS *średnie_zarobki*
FROM *PRACOWNICY*
GROUP BY *Id_wydz*

Id_wydz	**L_prac**	**płaca_najniższa**	**płaca_najwyższa**	**średnie_zarobki**
1	3	1800.00	2500.00	2100.00
2	5	1700.00	2500.00	2120.00
3	3	2000.00	3000.00	2333.33
4	2	2500.00	2800.00	2650.00
5	1	1500.00	1500.00	1500.00

Uwaga: wszystkie kolumny podane w zdaniu SELECT muszą wystąpić w klauzuli GROUP BY, chyba że występują jako argumenty funkcji sumarycznych.

Z konstrukcją zapytań wykorzystujących funkcje agregujące wiąże się klauzula HAVING, syntaktycznie zbliżona do klauzuli WHERE. Podobnie jak klauzula WHERE pozwala sformułować kryterium wyboru dla pojedynczych wierszy w tabeli, klauzula HAVING umożliwia sformułowanie kryterium wyboru dla grup wierszy. Jeżeli w zestawieniu opisanym w przykładzie powyżej należałoby wyselekcjonować wydziały o liczbie pracowników większej niż 2, to w poleceniu SQL musi pojawić się klauzula HAVING:

SELECT *Id_wydz*, COUNT(*Id_pracownika*) AS *L_prac*, MIN(*Uposażenie*) AS *Płaca_najniższa*, MAX(*Uposażenie*) AS *Płaca_Najwyższa*, AVG(*Uposażenie*) AS *średnie_zarobki*
FROM *PRACOWNICY*
GROUP BY *Id_wydz*
HAVING COUNT(*Id_pracownika*) >*2*

Id_wydz	L_prac	płaca_najniższa	płaca_najwyższa	średnie_zarobki
1	3	1800.00	2500.00	2100.00
2	5	1700.00	2500.00	2120.00
3	3	2000.00	3000.00	2333.33

3.2.3. Podzapytania – zapytania zagnieżdżone

Z zapytaniami zagnieżdżonymi mamy do czynienia w sytuacji, gdy w zdaniu SELECT (zewnętrznym) osadzone jest inne zdanie SELECT (wewnętrzne), którego wynik determinuje kryterium wyboru zdania zewnętrznego. Bardzo często konstrukcje takie występują łącznie z funkcjami agregującymi. Przykładowo, potrzebna jest lista pracowników, którzy zarabiają mniej niż wynosi średnia płaca w firmie. Aby taka lista powstała, najpierw należy przy użyciu funkcji agregującej wyznaczyć, ile wynosi średnia płaca, a następnie wyszukać pracowników, których uposażenie jest mniejsze niż średnia zarobków:

SELECT *Nazwisko, Imię, Uposażenie*
FROM *PRACOWNICY*
WHERE *Uposażenie* < (SELECT AVG(*Uposażenie*) FROM *PRACOWNICY*)

Id_pracownika	Nazwisko	Imie	Uposażenie
103	Kowalski	Piotr	1800.00
122	Burski	Adam	2000.00
126	Frankowski	Jerzy	2000.00
125	Gawron	Anna	1500.00
106	Mirski	Tadeusz	2000.00
104	Wirski	Jakub	2100.00
115	Porada	Maria	2000.00
109	Pakulski	Damian	1700.00

Inny przykład, którego celem jest pokazanie możliwości konstruowania dość zaawansowanych zestawień statystycznych przy użyciu funkcji agregujących i podzapytań, dotyczy wykonania listy pracowników, których zarobki są większe niż średnia zarobków w firmie oraz podania na liście, o ile są większe.

SELECT *Nazwisko, Imię, Stanowisko, Uposażenie* – (SELECT AVG(*Uposażenie*)
FROM *PRACOWNICY*) AS *Ponad średnią*
FROM PRACOWNICY
WHERE *Uposażenie* > (SELECT AVG(*Uposażenie*) FROM *PRACOWNICY*)

Nazwisko	Imie	Stanowisko	Ponad_średnią
Bielski	Leszek	kierownik	807.14
Wyzga	Anatol	menedżer	607.14
Adamska	Ewa	specjalista	307.14
Bielska	Janina	specjalista	307.14
Nowak	Jan	menedżer	307.14
Nawrot	Kamila	st. referent	107.14

Podane przykłady obrazują tak zwane podzapytania **skalarne**, czyli zwracające pojedynczą wartość. Jeśli podzapytanie zwraca tylko jeden wiersz zawierający więcej niż jedną wartość, to mówimy o podzapytaniach **wierszowych**, natomiast podzapytania zwracające dowolną liczbę wierszy nazywane są **tablicowymi**.

Przykładem zastosowania podzapytania zwracającego wiele wartości może być zestawienie nazw i numerów telefonów tych firm, które kiedykolwiek złożyły zamówienie do hurtowni:

```
SELECT nazwa_firmy, telefon FROM KLIENCI
WHERE Id_klienta IN (SELECT Id_klienta FROM ZAMOWIENIA)
```

Data

nazwa_firmy	telefon
Sportex	35-41-441
Sylwia	45-30-440
M&M	48-24-548
Karo	21-24-548
Kross	34-67-894
Anna	18-67-334
Gardena	11-23-345
Latisana	35-50-345

Często kryterium wyboru podzapytania odwołuje się do wartości z tabeli lub tabel występujących w zdaniu zapytania głównego, co określane jest jako odniesienie zewnętrzne, natomiast tak odwołujące się zapytanie nosi nazwę **podzapytania skorelowanego**. Przykładem może być zapytanie podające zestawienie numerów i dat zamówień wraz z przyporządkowaną nazwą firmy, która zamówienie złożyła.

```
SELECT Id, data_zam, (SELECT nazwa_firmy FROM KLIENCI WHERE
KLIENCI.id_klienta = ZAMOWIENIA.id_klienta)
FROM ZAMOWIENIA
WHERE data_zam > '2002-12-31'
```

Data

Id	data_zam	nazwa_firmy
36	2003-01-10	Sportex
37	2003-01-10	Latisana
38	2003-02-10	Sylwia
39	2003-02-10	Kross
40	2003-02-15	Kross
41	2003-02-15	Anna
42	2003-02-20	Anna
43	2003-02-20	Gardena
44	2003-02-26	Gardena

3.2.4. Złączenia tabel

Wszystkie dotychczasowe przykłady zapytań, mimo że niektóre wymagały pewnej wiedzy i finezji przy ich układaniu, charakteryzowały się istotnym ograniczeniem – działały tylko w odniesieniu do jednej tabeli. Nietrudno dostrzec, że takie działanie nie jest satysfakcjonujące, najczęściej zachodzi potrzeba wykorzystywania danych umieszczonych w różnych tabelach bazy danych, czyli wykonania złączeń tabel. Operacje złączenia (*join*) realizowane są w najprostszy sposób poprzez podanie w klauzuli WHERE kolumn, według których odbywa się połączenie. Przykład poniższy ilustruje połączenie dwóch tabel PRACOWNICY i WYDZIAŁY w celu sporządzenia listy pracowników wraz z nazwą wydziału, na którym pracownik jest zatrudniony:

SELECT *Id_pracownika*, *Nazwisko, Imię, nazwa_wydz*
FROM *PRACOWNICY, WYDZIALY*
WHERE *PRACOWNICY.Id_wydz = WYDZIAŁY.Id_wydz*

Data

ID_pracownika	nazwisko	imie	nazwa_wydz
115	Porada	Maria	sprzedaz
116	Nowak	Jan	sprzedaz
109	Pakulski	Damian	sprzedaz
100	Bielski	Leszek	marketing
120	Wyzga	Anatol	księgowośc
101	Adamska	Ewa	logistyka
112	Bielska	Janina	księgowośc
103	Kowalski	Piotr	logistyka
122	Burski	Adam	marketing
126	Frankowski	Jerzy	logistyka
125	Gawron	Anna	sekretariat
106	Mirski	Tadeusz	marketing
104	Wirski	Jakub	sprzedaz
117	Nawrot	Kamila	sprzedaz

Rys. 3.5. Obraz danych na ekranie aplikacji ISQL po złączeniu tabel PRACOWNICY i WYDZIAŁY

Analizując składnię polecenia SELECT użytego do złączenia tabel, należy zwrócić uwagę na różnice między podstawowym poleceniem SELECT, a podanym w przykładzie powyżej. Po słowie FROM podane zostały nazwy tabel, które mają być złączone, w klauzuli WHERE natomiast wymieniono warunki, na jakich ma się odbyć złączenie – według równych wartości w kolumnach Id_wydz obu tabel. Nazwy kolumn muszą być poprzedzone nazwami tabel, z których pochodzą. Tabela stojąca po lewej stronie znaku równości (PRACOWNICY) nosi nazwę *tabeli zewnętrznej*, a tabela po prawej stronie (WYDZIAŁY) jest nazywana *tabelą wewnętrzną*. Tak sformułowane złączenie nazywa się **złączeniem lewostronnym wewnętrznym** *(Inner Join)*. W wyniku złączenia występują tylko wiersze, dla których spełniony jest warunek złączenia.

Złączenia zewnętrzne

W złączeniach zewnętrznych, w odróżnieniu od przedstawionych powyżej wewnętrznych, w wyniku uwzględniane są wiersze, dla których nie jest spełniony warunek złączenia. W zależności od rodzaju złączenia zewnętrznego (lewostronne czy prawostronne) w tabeli wynikowej wiersze, dla których nie zostały znalezione wiersze spełniające warunek złączenia, uzupełniane są wartościami *null*. Przykładowo, wykaz zamówień z nazwami firm klientów utworzony za pomocą złączenia zewnętrznego lewostronnego będzie zawierać wartość *null* przy nazwie firmy, która nie składała żadnych zamówień (rys. 3.6).

SELECT *KLIENCI.Id_klienta, Nazwa_firmy, Id* AS *Numer_zamówienia*
FROM *KLIENCI* LEFT OUTER JOIN *ZAMÓWIENIA*

Data

id_klienta	nazwa_firmy	Numer_zamowienia
250	Kross	40
260	Falcon	(NULL)
280	Anna	19
280	Anna	20
280	Anna	34
280	Anna	41
280	Anna	42
290	Gardena	21
290	Gardena	22
290	Gardena	23
290	Gardena	35
290	Gardena	43
290	Gardena	44
211	Latisana	37
221	Hamilton	(NULL)
231	Warka	(NULL)

Rys. 3.6. Wynik działania złączenia zewnętrznego lewostronnego

Symetrycznie działa złączenie prawostronne (RIGHT OUTER JOIN) – w wyniku ukazywane są wszystkie wiersze tabeli prawej, natomiast wartościami *null* wypełniane są te wiersze tabeli stojącej po lewej stronie, dla których nie został spełniony warunek złączenia. Przykładowo, wykaz zamówień z nazwiskami pracowników, którzy je realizowali utworzony za pomocą złączenia zewnętrznego prawostronnego będzie zawierać wartość *null* przy nazwiskach osób nie zajmujących się obsługą zamówień (rys. 3.7).

SELECT *Id* AS *Numer_zamowienia*, *nazwisko, Imię*
FROM *ZAMÓWIENIA* RIGHT OUTER JOIN *PRACOWNICY*

W dialekcie Sybase dopuszczalne jest stosowanie składni nawiązującej do sposobu implementacji powiązań między tabelami, czyli mechanizmu klucza obcego. Polecenie złączenia tabel w takim przypadku nie wymaga wyszczególniania kolumn złączeniowych, lecz konstruowane jest za pomocą operatora KEY JOIN. Wracając do przykładu zilustrowanego rysunkiem 3.5, możemy stwierdzić, że ten sam rezultat można uzyskać, stosując prostszy zapis:

Data

Numer_zam	nazwisko	imie
37	Pakulski	Damian
44	Pakulski	Damian
(NULL)	Bielski	Leszek
(NULL)	Wyzga	Anatol
(NULL)	Adamska	Ewa
(NULL)	Bielska	Janina
(NULL)	Kowalski	Piotr
(NULL)	Burski	Adam
(NULL)	Frankowski	Jerzy
(NULL)	Gawron	Anna
(NULL)	Mirski	Tadeusz
11	Wirski	Jakub
14	Wirski	Jakub
17	Wirski	Jakub
18	Wirski	Jakub
23	Wirski	Jakub

Rys. 3.7. Wynik działania złączenia zewnętrznego prawostronnego

SELECT *Id_pracownika*, *Nazwisko, Imię, nazwa_wydz*
FROM *PRACOWNICY* KEY JOIN *WYDZIALY*

Gdy kolumny w łączonych tabelach mają tę samą nazwę, można wtedy stosować operator NATURAL JOIN, ale z zachowaniem ostrożności: jeżeli układający zapytanie nie panuje nad logiczną zawartością bazy danych, tzn. nad znaczeniem danych, to rezultat takiego zapytania może być całkowicie pozbawiony sensu. Z całą pewnością można poprzez operator NATURAL JOIN połączyć tabele PRACOWNICY i WYDZIAŁY, w obu tabelach bowiem występują kolumny o tej samej nazwie (*Id_wydz*) i połączenie danych pracownika z danymi wydziału, na którym pracuje jest sensowne. Można jednak wyobrazić sobie zastosowanie polecenia NATURAL JOIN do złączeń produkujących zupełnie bezużyteczne zbiory danych wynikowych, ze względu na różne znaczenie danych w kolumnach tabel, mimo tej samej nazwy kolumny i zgodności dziedziny.

Wszystkie przykłady związane ze złączeniami tabel dotyczyły działania na dwóch tabelach, co nie znaczy, że nie można łączyć ze sobą danych przechowywanych w większej liczbie tabel – czyli dokonywać **złączeń wielopoziomowych**. Przykładowo, aby utworzyć listę przyjętych zamówień zawierającą nazwę firmy zamawiającej, miasto, telefon (dane w tabeli KLIENCI), numer zamówienia, datę zamówienia (dane w tabeli ZAMÓWIENIA) oraz dane pracownika obsługującego zamówienie, czyli id_pracownika, nazwisko, imię (dane w tabeli PRACOWNICY), trzeba połączyć ze sobą trzy tabele (rys. 3.8):

SELECT *nazwa_firmy*, *KLIENCI.miasto*, *Id* AS *numer_zam*, *data_zam, PRACOWNICY.ID_pracownika* AS *Id_prac, Nazwisko, imie*
FROM *KLIENCI* KEY JOIN *ZAMÓWIENIA*
KEY JOIN *PRACOWNICY*

Data

nazwa_firmy	miasto	numer_zam	data_zam	Id_prac	Nazwisko	imie
Sportex	Głogów	10	2002-01-10	115	Porada	Maria
Sylwia	Lubin	12	2002-02-15	115	Porada	Maria
M&M	Wrocław	13	2002-03-11	116	Nowak	Jan
M&M	Wrocław	16	2002-03-18	116	Nowak	Jan
Sportex	Głogów	15	2002-04-20	109	Pakulski	Damian
Karo	Wrocław	11	2002-04-30	104	Wirski	Jakub
Kross	Wrocław	14	2002-05-02	104	Wirski	Jakub
Kross	Wrocław	17	2002-05-10	104	Wirski	Jakub
Kross	Wrocław	18	2002-06-15	104	Wirski	Jakub
Anna	Mirsk	19	2002-06-15	117	Nawrot	Kamila
Anna	Mirsk	20	2002-07-15	117	Nawrot	Kamila
Gardena	Maciejowy	21	2002-07-15	117	Nawrot	Kamila
Gardena	Maciejowy	22	2002-08-15	117	Nawrot	Kamila
Gardena	Maciejowy	23	2002-09-15	104	Wirski	Jakub
Kross	Wrocław	24	2002-09-15	104	Wirski	Jakub
Kross	Wrocław	25	2002-09-25	116	Nowak	Jan

Rys. 3.8. Wynik złączenia trzech tabel: KLIENCI, ZAMÓWIENIA, PRACOWNICY

W podanym przykładzie warto zwrócić uwagę na to, że nazwy niektórych kolumn zostały poprzedzone nazwami tabel, z których pochodzą; jest to wymóg w przypadku, gdy w łączonych tabelach występują kolumny o tych samych nazwach. W zapytaniu wykorzystany został operator KEY JOIN, ale oczywiście taki sam efekt przyniosłoby zastosowanie polecenia SELECT z klauzulą WHERE i jawnie podanym warunkiem złączenia.

Wszystkie operacje złączeń są podzbiorem **złączenia kartezjańskiego**, czyli iloczynu kartezjańskiego, w wyniku którego wszystkie wiersze jednej tabeli łączone są z wszystkimi wierszami drugiej. Na przykład złączenie przez iloczyn kartezjański tabel zawierających dane klientów i dane dotyczące zamówień:

SELECT *KLIENCI.Id_klienta, Nazwa_firmy, Id, Data_zam*
FROM *KLIENCI, ZAMÓWIENIA*

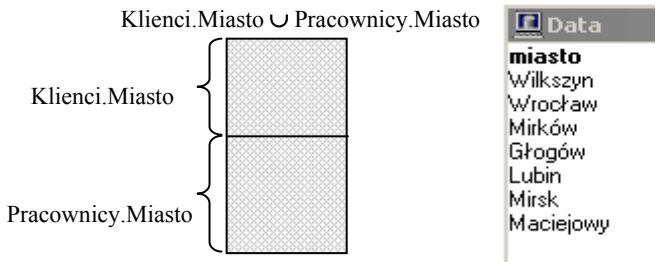
wygeneruje tabelę wynikową zawierającą zbiór połączonych danych wszystkich klientów (niezależnie od tego, czy składali zamówienie, czy nie) z każdym złożonym zamówieniem. Rozmiar zbioru wynikowego to liczba wierszy w pierwszej tabeli pomnożona przez liczbę wierszy w drugiej tabeli. Operacja taka rzadko kiedy ma zastosowanie praktyczne, zbiór wynikowy bowiem w zależności od rozmiaru tabel wejściowych może mieć bardzo duże rozmiary i zawierać dane nie zawsze mające sens praktyczny.

Operatory algebry relacyjnej

Standard języka SQL dopuszcza użycie operatorów algebry relacyjnej, czyli sumy mnogościowej (*UNION*), różnicy (*EXCEPT*) i przecięcia (*INTERSECT*). Nie wszystkie jednak dialekty mają te operatory zaimplementowane; w środowisku Sybase dopuszczalny jest jedynie operator **sumy mnogościowej** (UNION), pozostałe operatory

można jednak zastąpić innymi konstrukcjami języka SQL. Operacje mnogościowe mogą być wykonywane jedynie na relacjach wejściowych o takiej samej strukturze, tzn. mających taką samą liczbę atrybutów o takich samych dziedzinach w obu relacjach. Przykładowo, aby otrzymać wykaz miast będących siedzibą firm klientów i miast, w których mieszkają pracownicy, należy dodać elementy z kolumny miasto z obu tabel – PRACOWNICY i KLIENCI:

```
SELECT Miasto FROM PRACOWNICY
UNION
(SELECT Miasto FROM KLIENCI)
```

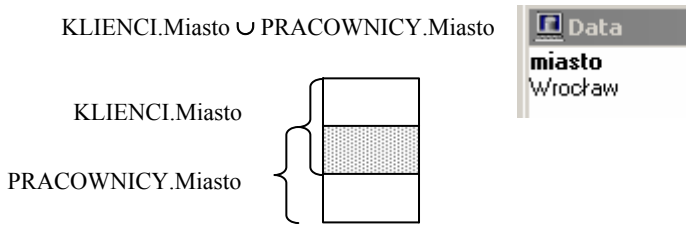


Rys. 3.9. Ilustracja działania operatora sumy mnogościowej (UNION)

Uwaga: Składnia polecenia podana w przykładzie domyślnie zakłada usunięcie duplikatów elementów danych. Jeżeli w wyniku mają znaleźć się wszystkie elementy danych, należy po operatorze UNION dodać ALL.

Operator przecięcia (INTERSECT) działa na dwóch relacjach wejściowych, produkując w wyniku relację zawierającą elementy wspólne z obu relacji. Jak wspomniano powyżej, w dialekcie Sybase nie jest on zaimplementowany, ale może być zastąpiony odpowiednio skonstruowanym zapytaniem języka SQL. Przykładowo, jeżeli potrzebny jest wykaz miast wspólnych dla siedzib firm klientów i zamieszkania pracowników, to można użyć zapytania:

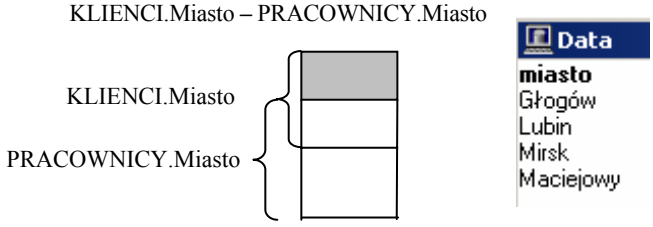
```
SELECT DISTINCT KLIENCI.Miasto FROM KLIENCI, PRACOWNICY
WHERE KLIENCI.Miasto = PRACOWNICY.Miasto
```



Rys. 3.10. Idea działania operatora przecięcia

Operator różnicy (EXCEPT) – podobnie jak w przykładzie opisanym powyżej, w dialekcie Sybase w jawnej postaci nie może być użyty. Operator ten, działając na dwóch relacjach, produkuje relację zawierającą elementy, które występują w jednej z relacji, a nie występują w drugiej. Potrzebne dane można uzyskać, stosując odpowiednie zapytanie SQL. Jeżeli na przykład potrzebny jest wykaz miast, w których mieszczą się siedziby firm, a nie mieszkają pracownicy, można użyć zapytania:

SELECT DISTINCT *miasto* FROM *KLIENCI*
WHERE *miasto* NOT IN (SELECT *miasto* FROM *PRACOWNICY*)



Rys. 3.11. Idea działania operatora różnicy

Uwaga: W przypadku operatora różnicy istotna jest, tak samo jak w działaniu odejmowania, kolejność relacji, dlatego też, konstruując zapytanie zastępujące operator EXCEPT, należy przemyśleć, o jaką różnicę chodzi. W odniesieniu do przykładu przedstawionego powyżej, zupełnie inny zestaw miast zostałby wybrany, gdyby zamienić kolejność tabel w zapytaniu, co zresztą spowodowałoby zmianę sensu zapytania: zamiast zestawu miast, w których mieszczą się siedziby firm, a nie mieszkają pracownicy, powstałoby zapytanie o wykaz miast, w których mieszkają pracownicy, a nie ma w nich siedzib firm, czyli:

SELECT DISTINCT *miasto* FROM *PRACOWNICY*
WHERE *miasto* NOT IN (SELECT *miasto* FROM *KLIENCI*)

3.2.5. Modyfikowanie zawartości bazy danych

W standardzie języka SQL w grupie instrukcji DML oprócz instrukcji umożliwiających wyszukiwanie danych z bazy danych znajdują się instrukcje umożliwiające zmianę zawartości tabel bazy danych. Są to trzy instrukcje:

INSERT – dodawanie nowych wierszy do tabeli,
UPDATE – modyfikowanie danych w tabeli,
DELETE – kasowanie wierszy z tabeli.

Składnia ogólna zdania INSERT

INSERT INTO *Nazwa tabeli* [(*lista kolumn*)]
VALUES (*lista wartości danych*)

Stosując powyższą składnię, można do tabeli o podanej nazwie wstawić pojedynczy wiersz. W przypadku, gdy w zdaniu INSERT nie jest podana lista kolumn, na liście wartości muszą wystąpić wartości dla wszystkich kolumn tabeli, w takiej kolejności, jak na schemacie tabeli i zgodne z typem danych określonym dla kolumny. W przypadku wstawiania danych do określonych kolumn należy wymienić nazwy tych kolumn. Wpisywanie danych z listy wartości odbywa się wtedy zgodnie z kolejnością zadeklarowaną na liście kolumn. Należy jednak zwrócić uwagę na to, które kolumny w schemacie tabeli zostały zadeklarowane jako kolumny, do których zapis danych jest wymagany – muszą one wystąpić na liście kolumn, a na liście wartości muszą być umieszczone odpowiadające im wartości danych, ponieważ dla takich kolumn nie jest dozwolona wartość *null*. Przykładowo, wpisanie nowego produktu do tabeli PRODUKTY(Id_p, nazwa, opis, rozmiar, kolor, ilość, cena_jedn) wymaga użycia instrukcji:

INSERT INTO *PRODUKTY*
VALUES (*20,'dres','damski','XS','czerwony',100,45*)

Uwaga: Przy wpisywaniu wartości znakowych lub dat należy tego typu dane umieszczać w apostrofach.

W tabeli *PRODUKTY* w odniesieniu do kolumny *kolor* dozwolone jest wprowadzanie wartości *null*, czyli w sytuacji, kiedy np. kolor jest trudny do zdefiniowania, można wprowadzić pozostałe dane o produkcie za pomocą drugiego wariantu zdania INSERT:

INSERT INTO *PRODUKTY* (*nazwa*, *opis*, *Id_p*, *ilość*, *rozmiar*, *cena_jedn*)
VALUES ('*dres*', '*męski*', *21*, *100*, '*XXL*', *75*)

Instrukcja INSERT w połączeniu ze zdaniem SELECT może służyć również do **kopiowania wierszy** z jednej tabeli do drugiej. Przykładowo, w celu przekopiowania do tabeli *NOTATNIK* (*Nazwisko, Imię, Telefon*) danych o klientach firmy przechowywanych w tabeli KLIENCI należy użyć polecenia:

INSERT INTO *NOTATNIK*
(SELECT *Nazwisko_k, Imie_k, Telefon*
FROM *KLIENCI*)

Składnia ogólna zdania UPDATE

UPDATE *Nazwa tabeli*
SET *nazwa kolumny1 = wartość danej 1* [, *nazwa kolumny 2 = wartość danej 2*]
[WHERE *warunek*]

Za pomocą instrukcji UPDATE dokonuje się modyfikacji wierszy w tabelach; w zależności od konstrukcji warunku WHERE modyfikacja może dotyczyć jednego lub kilku wierszy. Pominięcie klauzuli WHERE powoduje aktualizację wszystkich wierszy w tabeli. Układając polecenie aktualizacji, należy uwzględnić, że wprowadzone zmiany są trwałe, dlatego też ważną rzeczą jest właściwe skonstruowanie warunku wyboru wierszy, które mają zostać zaktualizowane. Jeżeli na przykład wszystkie znajdujące się w hurtowni artykuły mają podrożeć o 3%, to zaktualizowanie ceny jednostkowej nastąpi poprzez polecenie:

UPDATE *PRODUKTY*
SET *Cena_jedn = cena_jedn * 1.03*

Podwyżka płac o 10% dla wszystkich pracowników zatrudnionych na stanowisku specjalisty wymaga aktualizacji kilku wierszy, czyli odpowiedniego warunku wyboru:

UPDATE *PRACOWNICY*
SET *Uposażenie = Uposażenie * 1.10*
WHERE *Stanowisko* = '*specjalista*'

Za pomocą jednej instrukcji UPDATE można modyfikować kilka kolumn, zgodnie ze składnią, wymieniając je kolejno w poleceniu. Przykładowo: jeden z klientów hurtowni zmienił adres firmy oraz numer telefonu, co wymaga aktualizacji dwóch kolumn w tabeli KLIENCI:

UPDATE *KLIENCI*
SET Adres = '*Zielińskiego 44*', Telefon = '*44-45-467*'
WHERE *Id_klienta = 210*

Uwaga: Gwarancją aktualizacji tylko jednego wiersza jest konstruowanie kryterium wyboru z zastosowaniem kolumny klucza głównego; w przypadku kolumn nie będących kluczem wartości danych mogą się powtórzyć, co spowoduje modyfikację niezamierzonej liczby danych.

Składnia ogólna zdania DELETE

DELETE FROM *Nazwa tabeli*
[WHERE *warunek*]

Instrukcja DELETE umożliwia kasowanie wierszy z tabeli; gdy używana jest bez klauzuli WHERE, usuwana jest wtedy cała zawartość wyspecyfikowanej w poleceniu tabeli, natomiast zastosowanie klauzuli WHERE umożliwia określenie wierszy, które mają zostać usunięte. Usunięcie zawartości tabeli NOTATNIK:

```
DELETE FROM NOTATNIK
```

Należy pamiętać, że poleceniem kasowania usuwane są tylko wiersze z tabeli, natomiast definicja tabeli, czyli jej schemat nadal istnieje w bazie danych i w każdej chwili można do takiej tabeli wprowadzać nowe dane.

Usunięcie z tabeli PRODUKTY artykułów o nazwie „dres" i opisie „damski":

```
DELETE FROM PRODUKTY
WHERE nazwa = 'dres' AND opis = 'damski'
```

W przypadku wykonywania operacji kasowania wierszy obowiązuje podobna uwaga jak przy operacji aktualizacji – jeżeli skasowany ma zostać tylko jeden wiersz, warunek wyboru powinien zostać skonstruowany w odniesieniu do klucza głównego tabeli.

Język SQL – definiowanie obiektów bazy danych

W rozdziale trzecim omówione zostały instrukcje z grupy DML (*Data Manipulation Language*), czyli instrukcje umożliwiające wyszukiwanie danych zapisanych w tabelach oraz modyfikujące zawartość tabel. Do tworzenia zarówno samej bazy danych, jak i obiektów bazy danych, takich jak tabele, perspektywy (widoki), indeksy, dziedziny służą instrukcje z grupy DDL (*Data Definition Language*). Każdy z obiektów wchodzących w skład bazy danych musi być identyfikowalny, czyli musi posiadać nazwę. Standard SQL określa domyślny zbiór znaków, które mogą być używane do tworzenia nazw obiektów. Zbiór ten składa się z małych i dużych liter od A do Z, cyfr od 0 do 9 i znaku podkreślenia „_", z tym że różne środowiska bazodanowe dopuszczają możliwość wyspecyfikowania alternatywnych zbiorów znaków [23]. Przy nadawaniu nazw obiektom obowiązują następujące zasady:

- Identyfikator musi mieć dozwoloną długość, na ogół nie większą niż 128 znaków.
- Identyfikator musi rozpoczynać się od litery.
- Identyfikator nie może zawierać spacji (stąd w nazwach wieloczłonowych występuje znak podkreślenia).

Główne instrukcje języka DDL do tworzenia i zarządzania obiektami bazy danych:

CREATE DATABASE		DROP DATABASE
CREATE DOMAIN	ALTER DOMAIN	DROP DOMAIN
CREATE TABLE	ALTER TABLE	DROP TABLE
CREATE VIEW		DROP VIEW
CREATE INDEX		DROP INDEX

Pierwsze polecenie – **CREATE DATABASE** umożliwia założenie bazy danych, z tym, że szczegóły składni tego polecenia różnią się znacząco od siebie w zależności od środowiska bazodanowego, a nawet w obrębie jednego środowiska zależą od używanej wersji Systemu Zarządzania Bazą Danych. W systemach z wieloma użytkownikami proces zakładania bazy danych jest na ogół przypisany administratorowi (DBA). Pełna składnia polecenia zakładającego bazę danych w środowisku Sybase [23]:

```
CREATE DATABASE nazwa pliku bazy danych
        [[TRANSACTION] LOG OFF]
        [TRANSACTION] LOG ON [nazwa pliku]
        [MIRROR nazwa pliku]
        [CASE {RESPECT|IGNORE]
        [PAGE SIZE wielkość strony]
        [COLLATION etykieta]
        [ENCRYPTED {ON|OFF}]
        [PLANK PADDING {ON|OFF}]
        ASE [COMPATIBILE]]
        [JAVA {ON|OFF}]
        [JCONNECT {ON|OFF}]
```

Znaczenie poszczególnych klauzul zdania CREATE DATABASE:

Nazwa pliku – przy nadawaniu nazwy plikowi bazy danych, plikowi transakcji i plikowi kopii transakcji (*Mirror File Name)* należy używać apostrofów, gdyż nazwy są łańcuchem znaków. Można podać pełną ścieżkę dostępu do pliku, np.:

CREATE DATABASE '*C:\\ sybase\ '\moja_baza.db*'

W przypadku, gdy w zdaniu nie zostanie podana pełna ścieżka dostępu do pliku, zostanie on utworzony w aktualnej kartotece serwera bazy danych.

TRANSACTION LOG – klauzula ta, w zależności od wybranej opcji (OFF|ON) powoduje, że serwer zapisuje (lub nie) wszystkie operacje zmian w zawartości bazy danych wprowadzane przez użytkowników do pliku transakcji o podanej nazwie, czyli inaczej mówiąc – prowadzi dziennik transakcji. Jeżeli nie zostanie zdefiniowana pełna ścieżka dostępu do pliku, jest on umieszczany w tej samej kartotece co baza danych. Plik transakcji jest wykorzystywany w procesie tworzenia kopii zapasowej bazy danych, odtwarzania stanu bazy po awarii lub przy replikacji danych.

MIRROR – zastosowanie tej klauzuli powoduje prowadzenie kopii dziennika transakcji przez serwer; zazwyczaj w celu zwiększenia bezpieczeństwa danych kopia powinna znajdować się na oddzielnym urządzeniu. W ustawieniach domyślnych kopia nie jest tworzona.

CASE – klauzula ta związana jest z rozpoznawaniem przez serwer dużych i małych liter. W środowisku Sybase baza danych jest tworzona z domyślną nazwą użytkownika *DBA* i hasłem dostępu *sql*. W przypadku zastosowania klauzuli CASE RESPECT, przy logowaniu się użytkownika do bazy danych, hasło musi być wpisywane dużymi literami.

PAGE SIZE – klauzula służy do definiowania rozmiaru strony pamięci (umowna jednostka definiująca niepodzielne porcje danych, transmitowane z pamięci zewnętrznej do buforów wewnętrznych systemu) używanej przez serwer. Domyślna wartość rozmiaru strony wynosi 1024 bajty, a dopuszczalne wielkości to 512, 1024,

2048 i 4096 bajtów. Wybór wielkości strony związany jest z szacowanym rozmiarem bazy danych – im większy rozmiar bazy danych, tym korzystniejszy jest większy rozmiar strony.

COLLATION – klauzula umożliwia podanie etykiety do zbioru indeksów, zawierającego standardy języków narodowych (tzw. strony kodowe), w celu zdefiniowania standardów dla operacji porównywania.

ENCRYPTED – umożliwienie zaszyfrowania danych.

BLANK PADDING [ON|OFF] – w przypadku zastosowania tej klauzuli z opcją ON w operacjach porównywania łańcuchów znaków końcowe spacje są ignorowane. Przykładowo, dwa łańcuchy:

'*Kowalski*'

'*Kowalski*'

będą traktowane jako równe, jeżeli przy zakładaniu bazy danych zastosowana będzie klauzula BLANK PADDING ON. Ustawienie domyślne przyjmuje, że spacje są znaczące.

JAVA [ON|OFF] – umożliwia używanie bądź nie komponentów języka JAVA. Przy zastosowaniu ustawienia domyślnego klasy Javy są instalowane (opcja JAVA ON).

JCONNECT [ON|OFF] – umożliwia (opcja ON) bądź nie (opcja OFF) korzystanie z obiektów Sybase jCONNECT, czyli z łącza JDBC (*Java Database Connectivity*).

Przedstawiona powyżej składnia polecenia zakładającego bazę danych nie zawsze wykorzystywana jest z pełnym zestawem klauzul, należy jednak zwrócić uwagę, że w przypadku niewypisania klauzul w sposób jawny, serwer przyjmuje ustawienia domyślne, które nie zawsze odpowiadają wymaganiom użytkownika.

W środowisku Sybase wygodniej jest dla założenia bazy danych posłużyć się narzędziem Sybase Central, wybierając z zestawu funkcji w panelu *CREATE DATABASE*, co powoduje uruchomienie kreatora i pracę w trybie okienkowym. Wszystkie przedstawione powyżej klauzule realizowane są przez wybór odpowiednich opcji w kolejnych oknach udostępnianych przez kreator.

Po założeniu baza danych składa się z jednego pliku, wszystkie tworzone obiekty bazy danych będą umieszczane w tym pliku (plik główny). W wielu przypadkach rozwiązanie takie jest wygodne, ale w sytuacji, kiedy baza danych jest bardzo duża (Serwer ASA może obsługiwać bazy do 2 GB) istnieje możliwość podziału bazy na kilka plików, umieszczanych na jednym lub na różnych dyskach. Utworzenie dodatkowych plików bazy danych odbywa się poprzez polecenie:

CREATE DBSPACE *nazwa przestrzeni* AS *nazwa pliku*

Nazwa przestrzeni jest nazwą wewnętrzną dla pliku. W przypadku, gdy w poleceniu razem z nazwą pliku nie zostanie podana pełna ścieżka dostępu, zostanie on utworzony w tej samej kartotece, co główny plik bazy danych. Tak utworzona przestrzeń jest pusta; zapełnienie przestrzeni uzyskuje się poprzez kierowanie tworzonych tabel do odpowiednich przestrzeni.

Po założeniu bazy danych można przystąpić do tworzenia obiektów bazy danych, zaczynając od tabel, które będą zlokalizowane w bazie. Zakładanie tabel odbywa się poprzez polecenie **CREATE TABLE.** Składnia polecenia:

```
CREATE TABLE [GLOBAL TEMPORARY] TABLE [właściciel.] nazwa tabeli
  ({definicja kolumny [ograniczenia kolumny] | ograniczenia tabeli}, ...)
  [{IN|ON} nazwa przestrzeni tabel
  [ON COMMIT {DELETE|PRESERVE} ROWS]
```

Polecenie CREATE TABLE umożliwia tworzenie w bazie danych tabel stałych lub tymczasowych (*global temporary*) z podaniem identyfikatora właściciela tabeli. Schematy tabel, zarówno stałych jak i tymczasowych pozostają w bazie danych dopóty, dopóki nie zostaną w sposób jawny usunięte poprzez polecenie DROP TABLE *nazwa tabeli*. Podczas tworzenia schematu tabel można wybrać przestrzeń, w której tabela zostanie umieszczona (opcja IN|ON *nazwa przestrzeni*).

Znaczenie parametrów (w poleceniu pisane kursywą):

Definicja kolumny: *nazwa kolumny typ danych* [NOT NULL] [DEFAULT *wartość domyślna*]

Ograniczenia (*constrains*) kolumny:

UNIQUE – wymóg unikalnych wartości danych w kolumnie.

PRIMARY KEY – kolumna klucza głównego.

REFERENCES *nazwa tabeli* [(*nazwa kolumny*)] [*akcje*] – kolumna klucza obcego. Jeżeli w ograniczeniu podana jest tylko nazwa tabeli, klucz obcy zawierać będzie wartości z dziedziny klucza głównego tabeli, do której się odnosi, w przeciwnym przypadku, tzn. przy jawnie podanej nazwie kolumny (w tabeli głównej musi to być kolumna z ograniczeniem UNIQUE), w kluczu obcym oczekiwane są wartości z tej kolumny.

CHECK (*warunek*) – ograniczenie to umożliwia kontrolę danych wprowadzanych do kolumny poprzez sprawdzanie zgodności z zadeklarowanym warunkiem, np. płeć tylko ‘K’ lub ‘M’. W razie niezgodności danych z wyspecyfikowanym warunkiem operacja wstawiania danych nie zostanie wykonana.

COMPUTE (*wyrażenie*) – określa, że kolumna będzie zawierała wartości wyliczone w podanym wyrażeniu; systemowo zostaje zdefiniowana jako tylko do odczytu.

Wartości domyślne (DEFAULT):

wartości wpisywane do kolumny przez DBMS podczas wprowadzania danych, o ile instrukcja INSERT nie wyspecyfikuje dla takiej kolumny innej wartości. W charakterze wartości domyślnych mogą być używane zarówno łańcuchy znaków, liczby, wartość *null*, jak i funkcje pobierające bieżącą datę i czas. Na uwagę zasługuje możliwość przypisania kolumnie zawierającej wartości liczbowe domyślnej własności AUTOINCREMENT (DEFAULT AUTOINCREMENT), czyli automatycznego zwiększania liczby o 1 przy zapisie danych.

Akcje:
ON {UPDATE | DELETE}
{CASCADE | SET NULL | SET DEFAULT | RESTRICT}

Akcje umożliwiają realizację więzów integralności referencyjnej (rozdział 1). W zależności od decyzji podjętych w chwili definiowania tabeli, DBMS będzie wykonywał odpowiednie działania związane z operacjami kasowania i/lub aktualizacji danych w tabeli, czyli:

– usuwanie i/lub aktualizowanie w trybie kaskadowym,
– usuwanie i/lub aktualizowanie w trybie restrykcyjnym,
– usuwanie i/lub aktualizowanie ze wstawianiem wartości *null*,
– usuwanie i/lub aktualizowanie ze wstawianiem wartości domyślnej.

W celu zilustrowania sposobu układania zdań tworzących tabele wróćmy do bazy danych *Fitness2.db*. Do schematu bazy przedstawionego w poprzednim rozdziale dodane zostaną trzy tabele, z których pierwsza będzie przeznaczona do przechowywania danych dotyczących kategorii kwalifikacji zawodowych (*KATEGORIE*), druga (*KWALIFIKACJE*) do przechowywania wykazu kwalifikacji (kursy lub szkolenia), trzecia natomiast będzie zawierała informacje o pracownikach i posiadanych przez nich konkretnych umiejętnościach zawodowych (*PRAC_KWALIF*).

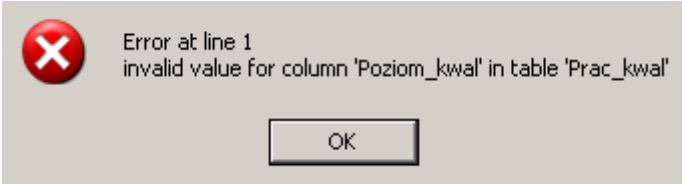
CREATE TABLE *KATEGORIE*
(*Id_kat* INTEGER NOT NULL DEAFAULT AUTOINCREMENT,
nazwa_kat CHAR(20) NOT NULL,
PRIMARY KEY (*Id_kat*))

CREATE TABLE *KWALIFIKACJE*
(*Id_kwal* INTEGER NOT NULL,
Nazwa_kwal CHAR (30) NOT NULL,
Opis (VARCHAR (50),
Id_kat INTEGER NOT NULL,
PRIMARY KEY (*Id_kwal*),
FOREIGN KEY (*Id_kat*) REFERENCES *KATEGORIE* ON DELETE SET NULL)

CREATE TABLE *PRAC_KWAL*
(*Id_pracownika* INTEGER NOT NULL,
Id_kwal INTEGER NOT NULL,
Poziom_kwal CHAR(15) CHECK (*Poziom_kwal* IN (‘*podst.*’, ‘*średni*’, ‘*zaawans.*’))
PRIMARY KEY (*Id_pracownika, Id_kwal*),
FOREIGN KEY (*Id_pracownika*) REFERENCES *PRACOWNICY*,
FOREIGN KEY (*Id_kwal*) REFERENCES *KWALIFIKACJE*)

Analizując polecenie zakładające tabelę *KATEGORIE*, należy zwrócić uwagę, że w definicji kolumny klucza głównego zastosowane zostało ograniczenie DEFAULT AUTOINCREMENT, czyli przy wprowadzaniu danych nie trzeba uwzględniać tej kolumny, będzie ona wypełniana przez System Zarządzania Bazą Danych automa-

tycznie liczbami zwiększanymi o 1 przy każdym zapisie, chyba że użytkownik w sposób jawny wpisze żądaną wartość. Z kolei w poleceniu zakładającym tabelę *KWALIFIKACJE*, w odniesieniu do kolumny klucza obcego zdefiniowana została akcja ON DELETE SET NULL, która skutkuje wpisaniem przez DBMS wartości NULL do tej kolumny we wszystkich wierszach, w których wystąpiła wartość klucza głównego kasowanego wiersza z tabeli nadrzędnej (*KATEGORIE*). Tabela *PRAC_KWAL* charakteryzuje się złożonym kluczem (dwukolumnowym) głównym oraz ograniczeniem CHECK zastosowanym do kolumny *Poziom_kwal*. Ograniczenie to precyzuje, jakie wartości mogą pojawić się w kolumnie. W przypadku próby wpisania wartości innych niż wyspecyfikowane na liście pojawi się komunikat systemowy o błędzie i zapis nie zostanie wykonany:



Rys. 4.1. Komunikat błędu podczas próby wpisu wartości danych niezgodnych z definicją schematu tabeli

Następną instrukcją z grupy DDL jest instrukcja umożliwiająca tworzenie dziedzin, lub własnych typów danych, traktowanych jako obiekty w bazie danych. Zalecane jest raczej tworzenie dziedzin niż własnych typów danych, gdyż jest to standard dla języka SQL. Wykorzystanie obiektów takich jak dziedziny korzystnie wpływa na zwiększenie spójności bazy danych; raz utworzona dziedzina może być wielokrotnie wykorzystywana (w zdaniu CREATE TABLE zamiast typu danych przypisywanego do kolumny używa się nazwy dziedziny). Każdy użytkownik definiujący dziedzinę automatycznie jest jej właścicielem (nie precyzuje się właściciela jak w przypadku tabel), ale zdefiniowane dziedziny są dostępne dla wszystkich użytkowników. Składnia polecenia:

```
CREATE {DOMAIN | DATATYPE} [AS nazwa dziedziny typ danych
  [[NOT] NULL]
  [DEFAULT wartość domyślna
  [CHECK (warunek)]
```

Nazwa dziedziny lub nazwa własnego typu danych musi spełniać reguły obowiązujące przy nadawaniu nazw obiektom bazy danych. Utworzona dziedzina może zostać zmieniona za pomocą instrukcji ALTER lub usunięta za pomocą instrukcji DROP, z tym że usunięcia może dokonać tylko właściciel obiektu lub administrator bazy danych. Przykładowa dziedzina dla kolumny *Płeć* w tabeli *PRACOWNICY*:

```
CREATE DOMAIN Płeć AS CHAR(1)
        CHECK (VALUE IN ('M', 'K'))
```

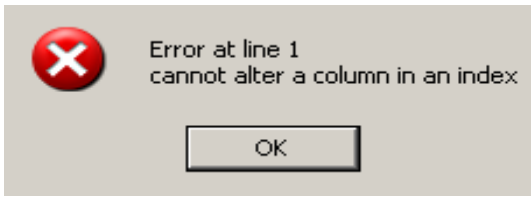
Standard ANSI/ISO dla języka SQL precyzuje również sposób zmiany struktury (schematu) tabel. Do tego celu służy polecenie ALTER TABLE, które umożliwia zmianę schematu w następującym zakresie:

- dodanie kolumny do tabeli,
- skasowanie kolumny z tabeli,
- dodanie nowych ograniczeń,
- skasowanie ograniczeń,
- ustawienie wartości domyślnych dla kolumny,
- skasowanie wartości domyślnych dla kolumny.

Składnia ogólna polecenia ALTER TABLE:

```
ALTER TABLE nazwa tabeli
  [ADD [COLUMN] nazwa kolumny typ danych [NOT NULL] [UNIQUE]
  [DEFAULT wartość domyślna ] [CHECK (warunek)
  [DROP [COLUMN] nazwa kolumny [RESTRICT | CASCADE]
  [ADD [CONSTRAINT [nazwa ograniczenia] definicja ograniczenia]
  [ALTER [COLUMN] SET DEFAULT wartość domyślna]
  ALTER [COLUMN] DROP DEFAULT]
```

Parametry występujące w poleceniu ALTER TABLE mają takie same znaczenie jak opisane w poleceniu CREATE TABLE. Dodawanie kolumny do tabeli jest składniowo podobne do definiowania kolumny przy zakładaniu schematu tabeli. Należy zwrócić uwagę na polecenie DROP COLUMN, czyli usuwanie kolumny z tabeli. Nie każdą kolumnę można usunąć, te kolumny, które zostały zadeklarowane jako klucze główne lub obce tabeli nie mogą być usuwane. W przypadku wydania takiego polecenia sygnalizowany jest błąd i operacja zostaje odrzucona (DBMS działa domyślnie zgodnie z trybem RESTRICT).



Rys. 4.2. Komunikat błędu podczas próby kasowania kolumny klucza tabeli

W takich przypadkach należy najpierw zdjąć z kolumny ograniczenie klucza głównego lub klucza obcego (DROP CONSTRAINT), a dopiero potem dokonywać zmian kolumny.

W dialekcie Sybase składnia polecenia ALTER TABLE nieco się różni od standardu, dlatego też poniżej podana jest składnia tego zdania obowiązująca dla środowiska Sybase:

```
ALTER TABLE [właściciel.] nazwa tabeli
  ADD definicja kolumny [ograniczenia kolumny]
  ADD ograniczenia tabeli
  MODIFY definicja kolumny
  MODIFY nazwa kolumny DEFAULT wartość domyślna
  MODIFY nazwa kolumny [NOT] NULL
  MODIFY nazwa kolumny CHECK NULL
  MODIFY nazwa kolumny CHECK (warunek)
  {DELETE | DROP} nazwa kolumny
  {DELETE | DROP CHECK}
  {DELETE | DROP} UNIQUE (nazwa kolumny, …)
  {DELETE | DROP} PRIMARY KEY
  {DELETE | DROP} FOREIGN KEY
  RENAME nowa nazwa tabeli
  RENAME nowa nazwa tabeli
```

Różnice w poleceniu ALTER TABLE dotyczą kilku funkcji rozpoczynających się od słowa MODIFY, dopuszczalne jest słowo DELETE zamiast DROP, jak również zaimplementowana jest opcja RENAME. Wykaz różnic:

- MODIFY *definicja kolumny* – umożliwia zmianę długości lub typów danych w kolumnie, należy jednak pamiętać, że nie każdy typ danych można przekonwertować na inny, np. znaki na typ liczbowy (akcja taka jest odrzucana, tabela pozostaje niezmieniona).
- MODIFY *nazwa kolumny* DEFAULT *wartość domyślna* – umożliwia zmianę wartości domyślnej dla podanej kolumny.
- MODIFY *nazwa kolumny* [NOT] NULL – zmienia ograniczenie NOT NULL dla podanej kolumny.
- MODIFY *nazwa kolumny* CHECK NULL – kasuje ograniczenie CHECK dla podanej kolumny.
- MODIFY *nazwa kolumny* CHECK (*warunek*) – zmienia istniejące ograniczenie CHECK na wyspecyfikowane w klauzuli.
- DELETE | DROP CHECK – kasuje ograniczenie CHECK dla całej tabeli.
- DELETE | DROP UNIQUE (*nazwa kolumny*, …) – znosi ograniczenie unikalności dla podanej kolumny.
- {DELETE | DROP} PRIMARY KEY – znosi ograniczenie klucza głównego dla danej kolumny.

• {DELETE | DROP} FOREIGN KEY – znosi ograniczenie klucza obcego dla danej kolumny.

• RENAME *nowa nazwa tabeli* – umożliwia zmianę nazwy tabeli.

• RENAME *nowa nazwa tabeli* – umożliwia zmianę nazwy kolumny w tabeli.

Przykład zdania ALTER TABLE zmieniającego schemat tabeli *PRAC_KWAL*, tzn. dodającego kolumnę o nazwie „*Zaświadczenie*" z ograniczeniem wprowadzania danych tylko dwóch wartości tak/nie:

```
ALTER TABLE PRAC_KWAL
  ADD COLUMN Zaświadczenie CHAR(3) CHECK (Zaświadczenie IN ('tak'/'nie'))
```

Jak już wspomniano, do usuwania tabel w bazie służy polecenie DROP, o składni ogólnej:

```
DROP TABLE nazwa tabeli [RESTRICT | CASCADE]
```

Polecenie to według standardu języka SQL usuwa z bazy wyspecyfikowaną tabelę zgodnie z wybranym trybem działania: restrykcyjnie lub kaskadowo. Należy zauważyć, że jeżeli wybrany zostanie tryb postępowania restrykcyjny (przyjmowany jako domyślny przez DBMS), to w sytuacji, kiedy tabela jest powiązana z innymi obiektami bazy danych polecenie zostanie przez DBMS odrzucone (tabela nie zostanie skasowana), natomiast w przypadku trybu kaskadowego usunięta będzie nie tylko wyspecyfikowana tabela, ale również wszystkie stowarzyszone z nią obiekty, a więc zmiany w bazie danych mogą być bardzo głęboko idące, co może nie być celowym zamiarem użytkownika. Jest rzeczą oczywistą, że wraz ze skasowaniem schematu tabeli kasowane są również wszystkie dane.

Polecenie DROP w dialekcie Sybase nie dopuszcza możliwości wyboru trybu kasowania. Zastosowanie polecenia DROP TABLE *nazwa tabeli* powoduje usunięcie tabeli, niezależnie od tego, czy była ona powiązana z innymi obiektami bazy danych, co z kolei może prowadzić do utraty spójności bazy danych, chociażby w sytuacji, kiedy zostanie skasowana tabela nadrzędna, do której klucza głównego odwołuje się kolumna klucza obcego tabeli powiązanej.

Następnym obiektem bazy danych tworzonym za pomocą instrukcji CREATE jest indeks, który bardzo ogólnie można określić jako mechanizm powodujący zwiększenie szybkości wyszukiwania danych. Składnia polecenia zakładającego indeks nie jest wprawdzie standardem języka SQL, ale wiele dialektów, w tym również Sybase, taką możliwość oferuje. Struktura i rodzaje indeksów zostaną dokładnie omówione w rozdziale 9. Składnia ogólna polecenia dla środowiska Sybase:

```
CREATE [UNIQUE] INDEX nazwa indeksu
  ON nazwa tabeli (nazwa kolumny [ASC | DESC], ...)
```

```
[{IN | ON} nazwa przestrzeni tabel
```

Zgodnie ze składnią, zastosowanie powyższego polecenia powoduje utworzenie uporządkowanego indeksu względem wyspecyfikowanych kolumn tabeli. Indeks istnieje tak długo, aż nie zostanie odwołany poleceniem DROP INDEX. Indeks jest przechowywany w tym samym pliku bazy danych co tabela, chyba że w poleceniu jawnie podana zostanie nazwa przestrzeni tabel, do której indeks ma zostać skierowany. W przypadku użycia ograniczenia UNIQUE w kolumnach indeksu nie będą powtarzać się wartości danych. Opcja ASC/DESC podaje kierunek uporządkowania wartości w kolumnach indeksu (rosnąco/malejąco). Serwer ASA automatycznie zakłada indeksy na kolumnach kluczy głównych oraz kolumnach z ograniczeniem UNIQUE.

Przykład polecenia zakładającego indeks na kolumnie *nazwisko* w tabeli *PRACOWNICY* i polecenia kasującego ten indeks:

CREATE INDEX *nazwiska* ON *PRACOWNICY* (*Nazwisko*)

DROP INDEX *nazwiska*

4.1. Perspektywy

Perspektywa (*view*) jest dynamicznym wynikiem działania jednej lub więcej operacji SELECT działających na tabelach bazy danych, w wyniku których powstaje tabela wirtualna, do której można kierować zapytania jak do tabel bazy danych. Perspektywa dla użytkownika bazy danych jawi się jak rzeczywista tabela, ze zbiorem kolumn i wierszy danych; w rzeczywistości nie przechowuje ona danych, lecz udostępnia dane zawarte w tabelach lub innych perspektywach. Składnia ogólna polecenia tworzącego perspektywy:

```
CREATE VIEW nazwa perspektywy [(nazwa kolumny,...)]
  AS SELECT WITH [CASCADED | LOCAL] CHECK OPTION
```

Perspektywa jest definiowana przez dowolne zdanie SELECT języka SQL (tu określane jako zapytanie definiujące), ale z wyłączeniem klauzuli ORDER BY. W definicji można wyspecyfikować nowe nazwy kolumn dla tworzonej tabeli wirtualnej. Klauzula WITH CHECK OPTION weryfikuje wprowadzane lub aktualizowane dane pod względem zgodności z kryterium zapytania definiującego. Opcja CASCADED/LOCAL używana jest w przypadkach budowania hierarchii perspektyw, czyli tworzenia perspektywy bazującej na innej perspektywie, również jako zabezpieczenie przed niepoprawnymi aktualizacjami. Różnica między składnią standardu a dialektem

Sybase w tym przypadku sprowadza się do tego, że w klauzuli WITH CHECK OPTION nie ma zaimplementowanej opcji CASCADED/LOCAL.

Przykład perspektywy udostępniającej dane pracowników z działu sprzedaży (kolumny pochodzą z dwóch tabel – *PRACOWNICY* i *WYDZIAŁY*):

CREATE VIEW *dział_sprzedaży*
AS SELECT *nazwisko, imię, stanowisko, uposażenie, nazwa_wydz*
FROM *PRACOWNICY* KEY JOIN *WYDZIALY*
WHERE *nazwa_wydz = 'sprzedaż'*

Po utworzeniu takiej perspektywy można odwoływać się do niej jak do tabeli bazy danych, czyli jeżeli wykonane zostanie polecenie:

SELECT * FROM *dział_sprzedaży*,

otrzymamy wynik pokazany na rys. 4.3

Data

nazwisko	imie	stanowisko	uposazenie	nazwa_wydz
Porada	Maria	referent	2000.00	sprzedaz
Nowak	Jan	menedżer	2500.00	sprzedaz
Pakulski	Damian	stażysta	1700.00	sprzedaz
Wirski	Jakub	st. referent	2100.00	sprzedaz
Nawrot	Kamila	st. referent	2300.00	sprzedaz

Rys. 4.3. Dane dla perspektywy *dział_sprzedaży*

Przykład perspektywy podającej wykaz wydziałów i liczbę pracowników na danym wydziale:

CREATE VIEW *wielkość_działów (wydzial, ilość_prac)*
AS SELECT *nazwa_wydz,* COUNT(*)
FROM *PRACOWNICY* KEY JOIN *WYDZIALY*
GROUP BY *nazwa_wydz*

Dane uzyskane po wykonaniu polecenia SELECT * FROM *wielkość_działów*:

Data

wydzial	ilosc_prac
logistyka	3
sprzedaz	5
marketing	3
księgowośc	2
sekretariat	1

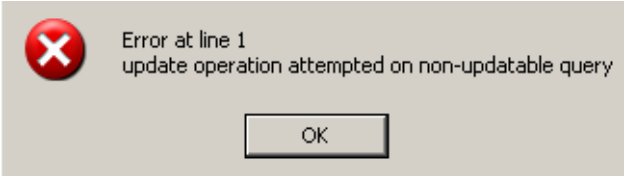
Rys. 4.4. Dane dla perspektywy *wielkość_działów*

Jak obrazują powyższe przykłady, w środowisku Sybase poprzez perspektywy można odwoływać się do wielu tabel jednocześnie, niemniej jednak obiekty te podlegają

pewnym ograniczeniom, zwłaszcza w przypadkach aktualizowania danych przez perspektywy. Ograniczenia, które muszą spełniać perspektywy w systemie Sybase [23]:

- Nie można w zapytaniu definiującym używać klauzuli ORDER BY.
- Nie można tworzyć indeksów ani procedur zdarzeń (*trigger*) w oparciu o perspektywy.
- Nie można aktualizować danych (polecenie UPDATE), wstawiać nowych wierszy (polecenie INSERT), kasować wierszy (polecenie DELETE) w perspektywach zawierających funkcje sumaryczne, klauzulę GROUP BY lub polecenie UNION.
- Polecenie INSERT nie jest wykonywane w przypadku perspektyw opartych na wielu tabelach. Jeśli perspektywa jest oparta na jednej tabeli, można używać polecenia wstawiania wierszy tylko w przypadku, gdy pozostałe kolumny tabeli bazy danych dopuszczają wartość *null*.
- Polecenie DELETE nie jest wykonywane w przypadku perspektyw opartych na wielu tabelach.

Analizując podane przykłady tylko w odniesieniu do perspektywy *dział_sprzedaży*, można stosować polecenie UPDATE (np. można aktualizować uposażenie pracowników działu sprzedaży lub zajmowane stanowisko) pamiętając o tym, że wykonanie operacji aktualizacji na perspektywie spowoduje aktualizację danych w tabeli bazy danych, natomiast polecenia INSERT lub DELETE nie będą realizowane; próby wykonania któregokolwiek z tych poleceń spowodują odrzucenie komendy i komunikat błędu:



Rys. 4.5. Komunikat błędu dla komend niezgodnych z ograniczeniami perspektywy

W obu przykładach perspektywy były tworzone bez zastosowania klauzuli WITH CHECK OPTION, co akurat w tych przypadkach, ze względu na ograniczone możliwości aktualizacyjne, nie ma większego znaczenia. W przypadku, gdy perspektywa oparta jest na jednej tabeli, klauzula WITH CHECK OPTION jest dobrym mechanizmem kontroli poprawności wykonywanych uaktualnień lub wstawień.

Podsumowując materiał zawarty w tym podrozdziale należy podkreślić, że tworzenie perspektyw jest wygodnym i bezpiecznym sposobem udostępniania danych użytkownikom, którzy nie muszą mieć przydzielonych uprawnień do korzystania z tabel bazy danych, mogą natomiast mieć zezwolenie na korzystanie z perspektyw, działają więc na takich zestawach danych, które są im potrzebne. Należy wziąć pod uwagę, że korzystanie z perspektyw opartych na wielu tabelach może być czasochłonne. System

Zarządzania Bazą Danych musi bowiem za każdym razem, kiedy użytkownik żąda dostępu do takiej perspektywy, dokonać łączenia tabel.

4.2. Transakcje

Transakcja jest logiczną, niepodzielną jednostką (porcją) pracy, na którą składają się pojedyncze zdania języka SQL. W wieloużytkownikowych systemach baz danych taki sposób pracy zapewnia bezpieczeństwo działania, gdyż podczas wykonywania transakcji zmiany bazy danych wykonywane przez transakcję są dla innych użytkowników niewidoczne dopóty, dopóki nie zostanie ona poprawnie zakończona [5, 7, 9, 18]. Główne cechy transakcji:

- niepodzielność – transakcja wykonywana jest w całości albo wcale,
- spójność – transakcje muszą zachowywać bazę w stanie spójnym,
- izolacja – transakcje są od siebie izolowane,
- trwałość – po pomyślnym wykonaniu transakcji zmiany bazy danych są zachowywane.

Niepodzielność transakcji zapewniana jest w Systemach Zarządzania Bazą Danych poprzez dziennik transakcji (*transaction log*); w przypadku zerwania transakcji baza danych jest przywracana do poprzedniego stanu na podstawie zawartości protokołu dziennika transakcji. Transakcje mogą się zakończyć w następujący sposób:

- zdaniem COMMIT, które powoduje zatwierdzenie zmian,
- zdaniem ROLLBACK, które odwołuje zmiany.

W niektórych przypadkach, np. awarii lub odłączenia od bazy danych, domyślną opcją jest polecenie ROLLBACK, czyli przywrócenie bazy do stanu poprzedniego. Po wykonaniu poleceń języka SQL z grupy DDL, takich jak ALTER, CREATE czy DROP, zmiany są zatwierdzane automatycznie. Ogólna składnia zdania wywołującego transakcję:

```
SET TRANSACTION
[READ ONLY | READ WRITE] |
[ISOLATION LEVEL READ UNCOMMITTED | READ COMMITED |
REAPEATABLE READ | SERIALIZABLE]
```

Kwalifikatory READ ONLY i READ WRITE wskazują, czy transakcja zawiera operacje tylko czytania, czy zarówno czytania jak i pisania. Należy jednak zauważyć, że transakcja READ ONLY może zawierać polecenia INSERT, UPDATE i DELETE, ale tylko w odniesieniu do tabel tymczasowych.

ISOLATION LEVEL określa dozwolony stopień interakcji ze strony innych transakcji podczas wykonywania nawiązanej transakcji. W środowisku Sybase składnia zdania nawiązującego transakcje do bazy danych jest nieco inna [23]:

BEGIN TRANSACTION *nazwa_transakcji*
SET OPTION ISOLATION LEVEL *nr poziomu izolacji*

Można wykorzystywać cztery poziomy izolacji, domyślnie ustawiany jest poziom najniższy, czyli 0, odpowiadający klauzuli READ UNCOMMITTED. Poziom 0 oznacza, że każde **zdanie** wchodzące w skład transakcji widzi tylko takie dane, które zostały zatwierdzone przed rozpoczęciem wykonania zdania (nie transakcji), co oznacza, że dane mogą być zmienione przez inną transakcję pracującą równolegle. Może więc wystąpić efekt tzw. „brudnego czytania" lub „czytania niepowtarzalnego". Ilustrując te przypadki:

- Brudne czytanie (*dirty read*) – załóżmy, że transakcja T1 modyfikuje wiersz danych. Zdanie transakcji T2 czyta ten wiersz zanim w transakcji T1 wystąpiło polecenie COMMIT zatwierdzające zmiany. Jeżeli z jakiś przyczyn transakcja T1 odwoła zmiany poprzez ROLLBACK, transakcja T2 będzie przetwarzała dane, które nigdy nie zostały zatwierdzone.
- „Czytanie niepowtarzalne" (*non-repetable read*) – transakcja T1 czyta wiersz danych. Transakcja T2 modyfikuje lub kasuje ten wiersz i zatwierdza zmiany. W przypadku próby przeczytania tego samego wiersza przez transakcję T1 okaże się, że wiersz jest zupełnie inny lub że takiego wiersza nie ma.

Zastosowanie najwyższego poziomu izolacji transakcji, czyli poziomu 3, powoduje, że każde zdanie transakcji widzi tylko dane (z uwzględnieniem zmian wprowadzonych przez operacje INSERT, UPDATE, czy DELETE), które zostały zatwierdzone zanim transakcja się rozpoczęła, co umożliwia uniknięcie zjawiska tzw. „wierszy widmowych":

„wiersze widmowe" (*phantom row*) – transakcja T1 czyta zbiór wierszy spełniających określone kryterium wyboru. Transakcja T2 wykonuje operację INSERT lub UPDATE (co może zwiększyć zbiór wierszy spełniających kryterium wyboru transakcji T1) i zatwierdza zmiany. Jeśli transakcja T1 powtórzy operację czytania, otrzyma zupełnie inny zbiór wierszy.

Zestawienie poziomów izolacji z zabezpieczeniem przed błędami wykonania transakcji podano w tabeli 4.1 (N – brak zabezpieczenia, T – zabezpieczenie przed błędem).

Tabela 4.1. Poziomy izolacji z zabezpieczeniem przed błędami wykonania transakcji

Poziom izolacji	0	1	2	3
Brudne czytanie	N	T	T	T
Niepowtarzalne czytanie	N	N	T	T
Wiersze widmowe	N	N	N	T

Zmianę poziomu izolowania transakcji podaje się w sposób jawny poprzez klauzulę SET ISOLATION LEVEL.

Serwer ASA, podobnie jak większość współczesnych relacyjnych DBMS, używa schematu blokowania (zapisywanego w tabeli blokad) do sterowania współbieżną pracą transakcji. W przypadku, gdy transakcja czyta lub zapisuje wiersz do tabeli bazy danych, automatycznie blokowany jest dostęp do tego wiersza. W zależności od poziomu izolacji transakcji stosowana jest blokada do odczytu lub blokada do zapisu.

- Blokada do odczytu – jest stosowana, gdy transakcja dokonuje tylko operacji odczytu, uniemożliwia modyfikacje zablokowanych danych przez inne transakcje, ale umożliwia innym transakcjom czytanie danych.
- Blokada do zapisu – serwer stosuje taką blokadę dla transakcji dokonujących wstawiania danych lub aktualizacji. Pozostałe transakcje nie mają możliwości ani zapisu, ani odczytu zablokowanych danych. Jest to tzw. blokada wyłączna.

Należy zwrócić uwagę, że ze względu na taki sposób sterowania współbieżnym wykonywaniem transakcji wybrany poziom izolowania transakcji powinien być adekwatny do działań, jaki transakcja wykonuje na bazie danych, aby w przypadku pracy współbieżnej nie powodować spowolnienia działania systemu. Blokady mogą bowiem dotyczyć całych tabel, wierszy lub kolumn w tabelach. Jeśli blokowane są duże elementy bazy danych, może zmniejszyć się efektywność operacji modyfikowania, natomiast w przypadku blokowania poszczególnych kolumn lub niektórych pól w wierszach wzrasta rozmiar tabeli blokad, co również niekorzystnie wpływa na wydajność całego systemu. Elementy danych nie muszą być blokowane przez cały czas trwania transakcji; blokada może trwać tylko podczas dostępu do danych, ale z kolei dynamiczne zwalnianie blokad i ponowne ich zakładanie może doprowadzić to tzw. zakleszczeń (*deadlock*). O zakleszczeniu można mówić wtedy, gdy transakcje wzajemnie się blokują, uniemożliwiając wykonanie którejkolwiek z nich. System Zarządzania Bazą Danych podejmuje w takiej sytuacji decyzję o wycofywaniu transakcji i umożliwieniu kontynuowania innych.

4.3. Procedury i funkcje

Procedury są obiektami składającymi się ze zdań języka SQL, zapisywanymi w bazie danych tak jak inne obiekty. Mogą być wykorzystywane przez wszystkie aplikacje i użytkowników, oczywiście pod warunkiem posiadania przez nich odpowiednich uprawnień. Procedury można podzielić na *procedury pamiętane* i *procedury zdarzeń* (*triggery*). Większość środowisk bazodanowych, w tym również Sybase, udostępnia możliwość tworzenia procedur zwracających wartości, podobnie jak funkcje w tradycyjnych językach programowania. Takie procedury w systemach baz danych

nazywane są również *funkcjami*. Ponieważ procedury i funkcje są obiektami bazy danych, a więc niezależnymi od aplikacji, sposób działania poprzez wykorzystywanie tych obiektów ma wiele zalet [9, 15, 23]:

- Standaryzacja – używanie tych samych procedur przez różnych użytkowników lub aplikacje powoduje wykonywanie pewnych działań w jednakowy, standardowy sposób.
- Efektywność – w sytuacji, gdy aplikacje korzystają z bazy danych zaimplementowanej na serwerze sieciowym, procedury i funkcje są wykonywane na komputerze serwera, a więc dla zrealizowania wymaganego dostępu do danych nie jest potrzebna komunikacja przez sieć. Taki sposób pracy jest szybszy, nieobciążony wpływem czynnika komunikacji sieciowej, w przeciwieństwie do wykonywania operacji po stronie klientów.
- Bezpieczeństwo – możliwość korzystania z procedur lub funkcji mają tylko użytkownicy, którym administrator bazy danych nadał odpowiednie uprawnienia, co pozwala na sterowanie dostępem do danych. Procedury zdarzeń, które zostaną dokładnie omówione w dalszej części rozdziału, są dodatkowym mechanizmem pozwalającym utrzymywać bazę w stanie spójnym, wymuszając niejako jednakowy sposób wykonywania operacji aktualizacyjnych.

Tworzenie procedury odbywa się poprzez zdanie CREATE PROCEDURE *nazwa_procedury*, po którym definiowane są *parametry wejściowe* (IN), *wyjściowe* (OUT), lub *wejściowo/wyjściowe* (INOUT). Tekst procedury rozpoczyna się od słowa BEGIN i kończy słowem END. Pomiędzy takimi „ramkami" umieszczany jest dowolny zestaw (ciąg) zdań języka SQL, gdzie poszczególne zdania muszą być oddzielane średnikami. Taki ciąg zdań traktowany jest jak zdanie złożone. Pomiędzy słowami BEGIN i END oprócz zdań SQL dopuszczalne są zdania sterujące, umożliwiające organizację przepływów logicznych i podejmowanie decyzji oraz lokalne deklaracje zmiennych, kursorów, tabel tymczasowych i identyfikacji błędów (*wyjątki*, *exception*) definiowanych przez użytkownika. Deklaracji dokonuje się przez zdanie DECLARE.

Składnia zdania deklaracji zmiennych

DECLARE *nazwa zmiennej typ danych*

W środowisku Sybase, deklarując obsługę sytuacji błędnych, korzysta się z wyjątków systemowych, których kody są przekazywane jako specjalne parametry OUT o nazwie SQLSTATE w następującym zdaniu:

DECLARE *nazwa wyjątku* EXCEPTION
FOR SQLSTATE [VALUE] *łańcuch znaków*

Przykładowe kody błędów:

00000 (no message)	02000 Row not found
01000 Warning	02W01 No data

Pełen wykaz kodów błędów znajduje się w dokumentacji technicznej środowiska Adapive Server Anywhere [23].

Wykaz zdań sterujących dopuszczalnych w procedurach lub funkcjach:

Zdania sterujące	**Składnia**

Wykonanie warunkowe	IF *warunek* THEN *lista zdań*
	ELSEIF *warunek* THEN
	Lista zdań
	ELSE
	Lista zdań
	END IF

CASE	CASE *wyrażenie* WHEN *wartość* THEN *Lista zdań*
Powtórzenia	WHILE *warunek* LOOP *Lista zdań* END LOOP
Wyjście z pętli	LEAVE *etykieta*
Pętla kursora	FOR *lista zdań* END FOR

Przykładowa procedura z zastosowaniem zdań sterujących; w zależności od parametru wejściowego w wyniku działania procedury powstaje lista firm klientów z adresami lub lista firm klientów wraz z numerami telefonów:

```
CREATE PROCEDURE firmy (IN parametr CHAR(1))
BEGIN
IF parametr = 'n' THEN
SELECT nazwa_firmy, adres, miasto
FROM KLIENCI
ELSE
        SELECT nazwa_firmy, telefon
        FROM KLIENCI
END IF
END
```

Wywołanie procedury (zarówno w przypadku pracy interaktywnej, jak i w aplikacjach i innych procedurach) odbywa się poprzez zdanie CALL *nazwa_procedury* (*wartości parametrów*). W powyższym przykładzie efektem wywołania CALL *firmy* ('*n*') będzie wykaz firm klientów z adresami, natomiast efektem wywołania procedury z jakimkolwiek innym parametrem będzie wykaz firm klientów z numerami telefonów.

Inny przykład procedury umożliwiającej wprowadzanie danych do tabeli *KWALIFIKACJE*:

```
CREATE PROCEDURE wpis_kwalifikacji
                (IN id INTEGER,
                 IN nazwa CHAR(30),
                 IN organizator VARCHAR(50)
                 IN kategoria INTEGER)
BEGIN
      INSERT INTO „DBA".KWALIFIKACJE (id_kwal, nazwa_kwal, opis, Id_kat
      VALUES (id, nazwa, organizator, kategoria)
END
```

W podanym przykładzie przykładowe wywołanie procedury: CALL *wpis_kwalifikacji* (60, 'kurs rachunkowości', 'TNOiK, podstawy', 1) spowoduje zapisanie nowego wiersza do tabeli KWALIFIKACJE, z takimi wartościami jak parametry wejściowe). Utworzona procedura jest obiektem bazy danych dopóty, dopóki nie zostanie jawnie usunięta zdaniem DROP PROCEDURE *nazwa_procedury*.

Przedstawione przykłady ilustrowały procedury z parametrami wejściowymi. W przypadku procedur z parametrami wyjściowymi odbiór wyników zwracanych przez procedurę do środowiska wywołującego jest możliwy w dwojaki sposób:

- wartości pojedyncze są zwracane jako parametry OUT lub INOUT procedury,
- tworzone są zbiory wyników.

Przykładowa procedura obliczająca średnią zarobków pracowników i zwracająca wynik obliczenia jako parametr wyjściowy OUT:

```
CREATE PROCEDURE Średnia_płaca (OUT średnia DECIMAL (20,2))
BEGIN
SELECT AVG (uposażenie) INTO średnia FROM PRACOWNICY
END
```

Jeżeli środowiskiem wywołującym jest aplikacja ISQL (interaktywny tryb pracy), należy przed jej wywołaniem utworzyć zmienną, która będzie przechowywała wartość parametru OUT:

```
CREATE VARIABLE średnia DECIMAL (20,2)
```

Następnie należy wywołać procedurę:

CALL *Średnia_płaca* (*średnia*)

Obliczoną wartość średnich zarobków otrzymamy w oknie danych po wykonaniu polecenia:

SELECT *średnia*

Podany przykład ilustruje działanie procedury zwracającej pojedynczą wartość poprzez parametr OUT. Inną możliwością są procedury zwracające zbiory wyników, podobnie jak zapytania.

Przykładowa procedura podająca listy pracowników wraz z zarobkami, w rozbiciu na wydziały:

CREATE PROCEDURE *lista_płac* (IN *nr_wydziału* INTEGER)
RESULT („*Identyfikator_pracownika*” INTEGER, „*pensja*” DECIMAL (20,2))
BEGIN
SELECT *Id_pracownika*, *uposażenie* FROM *PRACOWNICY*
WHERE *PRACOWNICY.Id_wydz = nr_wydzialu*
END

Po wywołaniu procedury: CALL *lista_płac* (*3*) z podaniem jako parametru wejściowego numeru wydziału, dla którego ma być sporządzone zestawienie, w aplikacji ISQL w oknie danych pojawi się wynik opisany komentarzami podanymi w klauzuli RESULT.

Przy omawianiu zdań sterujących dopuszczalnych w ciele procedur pojawiło się pojęcie *kursora*, które jest związane z przetwarzaniem wierszowym. Używając instrukcji języka SQL, nie ma możliwości przeglądania kolejnych wierszy będących wynikiem zapytania. Zgodnie z zasadami algebry relacyjnej zapytania działają na tabelach i wynikiem ich działania są również tabele – możliwość taką stwarza mechanizm kursora. Kursor jest buforem (obszarem roboczym), do którego są zapisywane kolejno wiersze będące wynikiem zapytania, z którego mogą one być pobierane w celu np. modyfikacji danych. Obsługa kursorów wymaga podobnych działań jak obsługa plików w tradycyjnym programowaniu:

- Deklarowanie kursora z przyporządkowaniem instrukcji SELECT

DECLARE *nazwa_kursora*
CURSOR FOR *zdanie SELECT*
[FOR {READ ONLY | UPDATE}]

- Otwarcie kursora za pomocą instrukcji OPEN (czyli uaktywnienie zdania SELECT przypisanego kursorowi)

OPEN *nazwa_kursora*

- Pobieranie kolejnych wierszy wyniku zapytania i przypisywanie ich do zmiennych za pomocą instrukcji FETCH, w celu przetworzenia danych

FETCH *nazwa_kursora* INTO *nazwa_zmiennej*

Standardowo instrukcja FETCH umieszczana jest w pętli, której działanie kończy się z chwilą sprowadzenia do kursora wszystkich wierszy wyniku.

- Zamknięcie kursora za pomocą instrukcji CLOSE *nazwa_kursora*

Przykład użycia kursora w procedurze pamiętanej:

```
BEGIN
DECLARE kurs_prac CURSOR FOR SELECT nazwisko FROM PRACOWNICY;
        DECLARE nazwisko_pracownika CHAR(40);
OPEN kurs_prac;
LOOP
        FETCH NEXT kurs_prac INTO nazwisko_pracownika;
        ...
END LOOP
CLOSE kurs_prac
END
```

Oczywiście kursory używane są nie tylko w procedurach pamiętanych, czy omawianych w dalszej części rozdziału procedurach zdarzeń. Deklaracje kursorów mogą wystąpić w dowolnym miejscu każdej aplikacji bazodanowej, niezależnie od tego, czy została skonstruowana za pomocą narzędzia typu RAD, czy instrukcje języka SQL zostały osadzone w języku programowania 3GL.

Funkcje są w zasadzie pewną klasą procedur, które zwracają pojedyncze wartości do środowiska wywołującego. Tworzy się je za pomocą zdania CREATE FUNCTION. Syntaktyka zdania tworzącego funkcję różni się bardzo niewiele od zdania tworzącego procedurę. Zasadnicze różnice polegają na tym, że:

- W zdaniu tworzącym funkcję nie występują słowa IN, OUT, INOUT, ponieważ wszystkie parametry są parametrami wejściowymi.
- W przypadku, kiedy funkcja zwraca do środowiska wywołującego dane, wymagane jest użycie słowa kluczowego RETURNS, po którym precyzuje się typy zwracanych danych.
- W przypadku, kiedy funkcja zwraca pojedynczą wartość używa się słowa kluczowego RETURN.

Przykładowa funkcja, która w efekcie swojego działania zwraca dane w postaci łańcucha znaków zawierającego nazwisko i imię rozdzielone spacją:

```
CREATE FUNCTION dane_osób (imię (CHAR 20), nazwisko (CHAR (30))
RETURNS (CHAR 51);
BEGIN
DECLARE dane_os CHAR(51);
```

SET *dane_os* = *imię* || ‘ ‘ || *nazwisko*;
RETURN *dane_os*;
END

Wywołanie funkcji w aplikacji ISQL w celu uzyskania listy imiennej pracowników hurtowni *Fitness2*:

SELECT *dane_osób* (*imię, nazwisko*) AS *dane_os* FROM *PRACOWNICY*

Efekt takiego wywołania:


Data

dane_os
Maria Porada
Jan Nowak
Damian Pakulski
Leszek Bielski
Anatol Wyzga
Ewa Adamska
Janina Bielska
Piotr Kowalski
Adam Burski
Jerzy Frankowski
Anna Gawron
Tadeusz Mirski
Jakub Wirski
Kamila Nawrot

Gdyby potrzebna była lista imienna klientów hurtowni, wówczas wywołanie funkcji musiałoby się odnosić do tabeli z danymi klientów, czyli:

SELECT *dane_osób* (*imię_k, nazwisko_k*) AS *dane_os*
FROM *KLIENCI*

Efektem tego wywołania będzie lista imienna klientów hurtowni:

Procedury zdarzeń (*wyzwalacze*, *triggery*) są to procedury definiujące akcje, które system powinien podjąć po wystąpieniu określonych zdarzeń dotyczących tabel bazy danych. Konstruowane są w celu zdefiniowania dodatkowych więzów integralności lub audytowania zmian bazy danych. Zdarzenia, z którymi związane są procedury zdarzeń dotyczą wstawiania, aktualizacji i usuwania wierszy z tabel. Składnia instrukcji umożliwiającej utworzenie *triggerów* (procedur zdarzeń):

CREATE TRIGGER *nazwa triggera czas zadziałania zdarzenia* [, *zdarzenie*, ...]
[ORDER integer] ON *nazwa tabeli*
[REFERENCING [OLD AS *stara nazwa*]
[NEW As *nowa nazwa*]]
[FOR EACH ROW {ROW | STATEMENT}]
[WHEN (*warunek wyboru*)]
[IF UPDATE (*nazwa kolumny*) THEN
[{AND | OR} UPDATE (*nazwa kolumny*)] …]
ciąg zdań (zdanie złożone)
[ELSEIF UPDATE (*nazwa kolumny*) THEN
[{AND | OR} UPDATE (*nazwa kolumny*)]…
ciąg zdań (*zdanie złożone*)
END IF]]

Czas zadziałania procedury określa się poprzez podanie jednego z parametrów: BEFORE | AFTER.

Zdarzenia, z którymi związana ma być procedura określa się poprzez wybranie odpowiedniej akcji: DELETE | INSERT | UPDATE | UPDATE OF *lista kolumn*.

Procedury zdarzeń uruchamiane są automatycznie przez DBMS (odpalane) przed (opcja BEFORE) lub po (opcja AFTER) wykonaniu operacji aktualizacji, wstawiania lub kasowania, wykonywanych na pojedynczym wierszu (opcja FOR EACH ROW) lub po wykonaniu całej instrukcji (opcja FOR EACH STATEMENT). Klauzula WHEN jest stosowana tylko dla triggerów wierszowych. Dla rozróżnienia starych i nowych wartości w wierszu stosuje się oznaczenia REFERENCING OLD i REFERENCING NEW. W przypadku triggerów zdaniowych należy podać nazwy tworzonych tymczasowych tabel przechowujących stare i nowe wartości wierszy.

Przykładowy trigger umożliwiający sprawdzenie, czy przyjmowany pracownik jest pełnoletni:

```
CREATE TRIGGER kontrola_daty_ur
BEFORE INSERT ON PRACOWNICY
REFERENCING NEW AS NOWI_PRACOWNICY
FOR EACH ROW
BEGIN
DECLARE skontroluj_datę_urodzenia EXCEPTION FOR SQLSTATE '99999';
        IF NOWI_PRACOWNICY.data_ur > '1985-01-01' THEN
        SIGNAL skontroluj_datę_urodzenia;
        END IF;
END
```

Procedury zdarzeń są, podobnie jak procedury pamiętane i funkcje, obiektami bazy danych, których usunięcia dokonuje się za pomocą instrukcji DROP TRIGGER *nazwa*.

Podsumowując tę partię materiału warto zwrócić uwagę na możliwość umieszczania tekstów procedur i funkcji w skryptach SQL, co znacznie ułatwia zmiany i modyfikacje. W przypadku korzystania z aplikacji ISQL skrypty tworzy się poprzez polecenie zapamiętania tekstu wpisywanej procedury wybierane z menu (opcja *SAVE US*), co powoduje utworzenie pliku typu *Nazwa*.sql, który w dowolnej chwili może być otwierany (opcja *OPEN*) i wykonywany (klawisz *EXECUTE*).

4.4. Sterowanie dostępem do danych

Każdy System Zarządzania Bazą Danych jest wyposażony w mechanizmy umożliwiające kontrolę i sterowanie dostępem do danych. Mechanizmy zabezpieczeń bazują na idei autoryzacji użytkowników, praw własności do obiektów oraz przydzielonych uprawnień. Autoryzacja użytkowników jest dokonywana na podstawie identyfikatorów oraz haseł przydzielanych każdemu użytkownikowi przez administratora bazy danych. Z każdym identyfikatorem związany jest przydzielony zakres uprawnień, sprawdzany każdorazowo przez DBMS z chwilą podejmowania działań przez użytkownika. Każdy z obiektów bazy danych ma swojego właściciela (użyt-

kownik, który obiekt utworzył) i początkowo jest to jedyna osoba, która może wykonywać operacje na obiekcie. W języku SQL istnieją dwie instrukcje umożliwiające przydzielanie i odwoływanie uprawnień do działania na obiektach bazy danych: GRANT (przydziel) i REVOKE (odwołaj).

Składnie zdania GRANT zaimplementowane w środowisku Sybase:

1. GRANT CONNECT TO *identyfikator_użytkownika*, ... IDENTIFIED BY *hasło*

Zdanie o takiej składni umożliwia utworzenie nowego użytkownika bazy danych lub zmianę hasła istniejącego użytkownika.

2. GRANT
DBA
| GROUP
| MEMBERSHIP IN GROUP *identyfikator_użytkownika*, …
| [RESOURCE]
…TO *identyfikator_użytkownika*

W powyższym zdaniu identyfikator DBA oznacza administratora bazy danych z pełnym zakresem uprawnień. Klauzula GROUP umożliwia tworzenie grup użytkowników z prawem posiadania członków, a klauzula MEMBERSHIP IN GROUP przypisanie poszczególnych użytkowników do grup. Poprzez słowo RESOURCE udzielane jest uprawnienie do zakładania tabel i perspektyw. Przykładowo, założenie grupy księgowi i przypisanie konkretnej osoby do tej grupy wymaga następujących sekwencji zdań:

- Utworzenie użytkownika *księgowi*

GRANT CONNECT TO *księgowi* IDENTIFIED BY *bilans*

- Nadanie użytkownikowi statusu grupy

GRANT GROUP TO *księgowi*

- Przypisanie członka do grupy

GRANT MEMBERSHIP IN GROUP *księgowi* TO *J_Bielska*

Użytkownik przypisany do grupy dziedziczy wszystkie uprawnienia przydzielone grupie.

3. GRANT {*lista uprawnień* | ALL PRIVILEGES}
ON *nazwa obiektu*
TO *identyfikator_użytkownika*, …[WITH GRANT OPTION]

Lista uprawnień:

ALTER – zezwolenie na zmiany struktury tabeli.

DELETE – zezwolenie na kasowanie wierszy w wyspecyfikowanej tabeli lub perspektywie.

INSERT – zezwolenie na wpisywanie wierszy do wyspecyfikowanej tabeli lub perspektywy.

REFERENCES [(*nazwy kolumn*)] – zezwolenie na tworzenie indeksów i kluczy obcych odnoszących się do wyspecyfikowanej tabeli. Jeżeli w klauzuli wystąpią nazwy kolumn, to uprawnienia dotyczą tylko kolumn wymienionych na liście (lista kolumn dotyczy tylko tabeli, nie perspektywy).

SELECT [(*nazwy kolumn*)] – zezwolenie na odczyt danych w wyspecyfikowanej perspektywie lub tabeli. Jeżeli określona zostanie lista kolumn, to zezwolenie dotyczy tylko tych kolumn (lista kolumn dotyczy tylko tabeli, nie perspektywy).

UPDATE [(*lista kolumn*)] – zezwolenie na aktualizację danych w tabelach lub perspektywach, z podobną uwagą odnośnie do listy kolumn jak w poprzednich klauzulach.

Klauzula WITH GRANT OPTION umożliwia przekazywanie swoich uprawnień innym użytkownikom.

Przykładowo, jeżeli istnieje użytkownik o identyfikatorze *kierownik*, który powinien mieć uprawnienia do odczytu tabeli *PRACOWNICY* oraz aktualizacji kolumny *Uposażenie*, polecenie nadania uprawnień ma postać:

GRANT SELECT, UPDATE (*Uposażenie*) ON *PRACOWNICY* TO *kierownik*
WITH GRANT OPTION

Zastosowanie klauzuli WITH GRANT OPTION umożliwia przekazanie uprawnień do odczytu oraz aktualizacji kolumny *Uposażenie* innym użytkownikom.

4. GRANT EXECUTE ON [*id_właściciela*] *nazwa_procedury* TO *id_użytkownika, ...*

Zdanie to oznacza udzielenie uprawnień do korzystania z procedur wyspecyfikowanym użytkownikom.

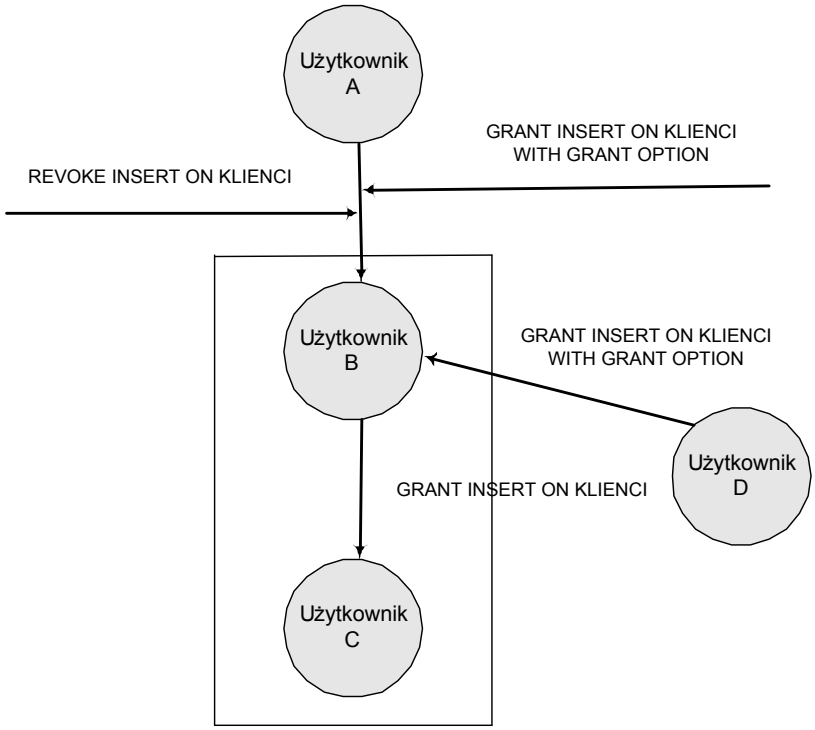
Odwoływanie uprawnień udzielonych przez zdanie GRANT odbywa się poprzez zdanie REVOKE. Składnie zaimplementowane w środowisku Sybase:

1. REVOKE
{CONNECT | DBA | GROUP | MEMBERSHIP IN GROUP *id_użytkownika*, ... | RESOURCE} FROM *id_użytkownika, ...*

Za pomocą tego zdania odwoływane są specjalne zezwolenia użytkowników.

2. REVOKE {ALL PRIVILEGES | *Lista_uprawnień*} [GRANT OPTION FOR]
ON *nazwa_obiektu* FROM *id_użytkownika*

Zdanie powyższe odwołuje przydzielone uprawnienia. Należy jednak zauważyć, że pojedyncze zdanie REVOKE nie odwołuje uprawnień nadanych przez wszystkich użytkowników i może zdarzyć się taka sytuacja, że pomimo odebrania jakiemuś użytkownikowi uprawnień nadal będzie on miał dostęp do określonego obiektu, co ilustruje rys. 4.6.



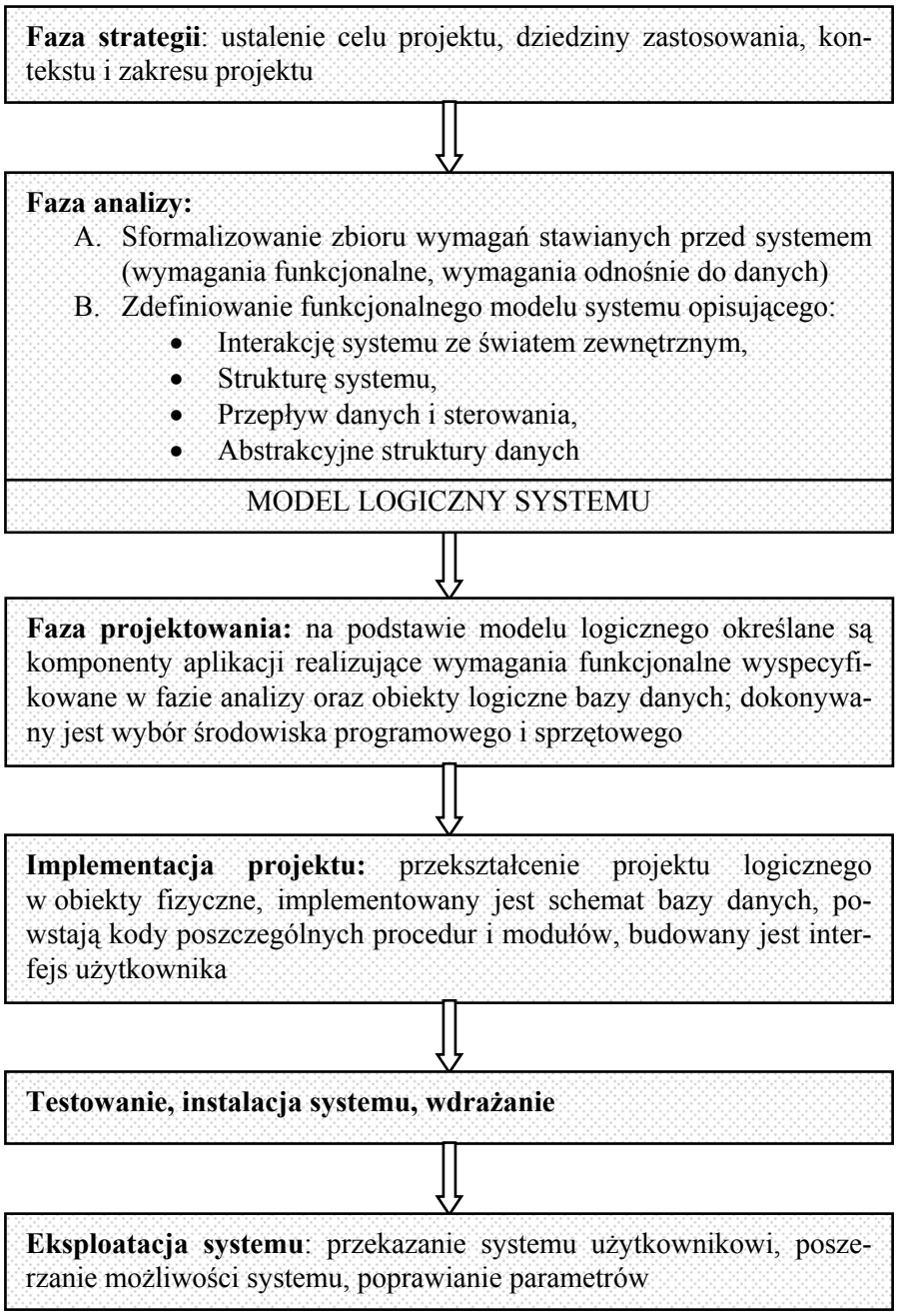
Rys. 4.6. Mechanizm odwoływania uprawnień

5 Etapy projektowania systemów baz danych

W poprzednich rozdziałach przedstawiono zagadnienia związane z budową, funkcjonowaniem, administrowaniem i użytkowaniem relacyjnych baz danych na podstawie środowiska Sybase. Rozdział ten i następne dotyczyć będą zagadnień modelowania, projektowania i implementacji projektów zarówno baz danych, jak i aplikacji bazodanowych, co wydaje się uzasadnione, gdyż nawet traktując bazę danych jako fundament, trudno pominąć resztę konstrukcji, czyli programy użytkowe, związane z tym fundamentem. Trudno sobie ponadto wyobrazić projektowanie bazy danych jako zadanie samo w sobie; raczej nie zdarzają się sytuacje, w których powstaje baza danych, z której nie będzie korzystała żadna aplikacja. Pomimo tego więc, że oddzielnym bytem jest baza danych i oddzielnym aplikacja z niej korzystająca, oba obiekty wzajemnie na siebie oddziałują i współpracują ze sobą, co należy uwzględnić już na najwcześniejszym etapie budowy systemu z bazą danych.

Nie jest celem niniejszego podręcznika przedstawienie teoretyczne metodologii projektowania systemów informatycznych. Zagadnienia te potraktowane są bardzo szczegółowo w wielu podręcznikach z zakresu inżynierii oprogramowania. Celem jest raczej zaprezentowanie praktycznego zastosowania określonych metodologii, techniki i narzędzi w procesie projektowania i budowania baz danych i współpracujących z nimi programów (aplikacji), które można traktować jako pewną klasę systemów informatycznych, czyli klasę systemów bazodanowych. Przystępując do realizacji określonego zadania, jakim jest w tym przypadku zaprojektowanie i zrealizowanie systemu z bazą danych, trzeba jednak mieć świadomość, że cały proces powinien przebiegać zgodnie z określonymi zasadami i regułami, które obowiązują przy tego typu przedsięwzięciach, czyli według określonego planu. Należy zauważyć, że wszystkie przedsięwzięcia (nie tylko informatyczne) są realizowane zgodnie z obowiązującymi procedurami, na ogół według ustalonego harmonogramu etapów, po zrealizowaniu których powstaje produkt finalny.

W odniesieniu do systemów informatycznych, a więc i do systemów z bazami danych wyróżnia się kilka etapów (faz) [1, 19, 22], tworzących plan postępowania przy konstruowaniu aplikacji bazodanowych. Schemat ten nazywany jest inaczej cyklem życia systemu informatycznego (rys. 5.1).



Rys. 5.1. Cykl życia systemu informatycznego

Pierwszą fazą cyklu życia systemu informatycznego jest *faza strategii*, którą często określa się jako studium wykonalności lub wstępnego badania środowiska użytkownika. Głównym zadaniem w tym etapie jest sprecyzowanie celu i zadań budowanego systemu. Dobrą praktyką jest określanie celu krótkim, syntetycznym zdaniem, zwłaszcza w odniesieniu do małych projektów, gdzie przez pojęcie system z bazą danych rozumie się bazę danych i program współpracujący z bazą, co nie jest ani błędem, ani nadużyciem – zgodnie bowiem z powszechnie przyjętą definicją, system jest to zbiór niezależnych składowych współpracujących ze sobą i tworzących jednolitą całość, reagujących na bodźce zewnętrzne. Wejściami do systemu są zasoby pozyskiwane z otoczenia, wyjściami są efekty działania systemu dostarczane do otoczenia. W przypadku aplikacji bazodanowych składowymi systemu są: sprzęt komputerowy (sieć komputerowa, procesory, terminale, drukarki), oprogramowanie (oprogramowanie sieciowe, systemy operacyjne, systemy zarządzania bazami danych, aplikacje), dane, użytkownicy oraz zbiór skodyfikowanych instrukcji obsługi systemu. Wracając do określenia celu, w zdaniu precyzującym cel podmiotem powinien być projektowany system, orzeczenie powinno opisywać zadanie, jakie system ma realizować, a dopełnienie określać obiekt, którego dotyczy zadanie systemu. Przykładowo: „System FITNESS 2 będzie wspomagał zarządzanie hurtownią odzieży sportowej", „System KADRY 1 będzie rejestrował urlopy i zwolnienia chorobowe pracowników", „System SSKF będzie wspomagał serwisowanie kas fiskalnych". Oczywiście określenie celu jednym zdaniem nie zawsze jest możliwe, zwłaszcza w takich sytuacjach, kiedy projektowany ma być duży system, współpracujący z innymi funkcjonującymi w danym środowisku, niemniej jednak nie powinno zajmować więcej niż jeden akapit tekstu, gdyż w tej fazie nadmiar szczegółów nie jest potrzebny, a może prowadzić do zagmatwania i niezrozumienia głównego zadania stawianego przed systemem.

Następnym krokiem jest wyspecyfikowanie funkcji, jakie system musi realizować, aby osiągnąć założony cel. Na tym etapie jest to po prostu opisowa lista funkcji, które mają być udostępnione użytkownikowi. Wymienione powinny zostać funkcje kluczowe, czyli takie, które umożliwiają realizację założonego celu systemu. Przykładowo, dla systemu wspomagającego zarządzanie hurtownią odzieży należałoby w fazie strategii wyspecyfikować następujące funkcje umożliwiające realizację celu:

- rejestracja i obsługa zamówień klientów,
- generowanie rachunków dla klientów,
- śledzenie stanów magazynu,
- prowadzenie statystyk sprzedaży.

W fazie strategii formułowane są również kryteria efektywności dla projektowanej aplikacji, co wiąże się z koniecznością identyfikacji użytkowników systemu i określenia zasięgu systemu. Często pomocny może okazać się wstępny diagram kontekstowy, umożliwiający zobrazowanie granic systemu (system jest przedstawiany jako pojedynczy proces) oraz otoczenie zewnętrzne, tzw. terminatory (osoby, jednostki organizacyjne lub inne systemy informatyczne), z którymi projektowany system będzie się

kontaktował. Diagram DFD (*Data Flow Diagram*, *diagram przepływu danych*), którego przypadkiem szczególnym jest diagram kontekstowy, zostanie omówiony dokładniej w rozdziale 6, prezentującym zasady tworzenia modelu systemu metodami strukturalnymi.

Zakończeniem fazy strategii jest wykonanie analizy kosztów, czyli przybliżonego oszacowania harmonogramu, kosztów budowy i wdrożenia nowego systemu. Oszacowanie kosztów projektu nie jest oczywiście rzeczą prostą, ponieważ należy uwzględnić szereg czynników wzajemnie ze sobą powiązanych, głównie zaś: wynagrodzenie zespołu zaangażowanego przy projekcie, które wiąże się z czasem realizacji projektu, koszty szkolenia użytkowników, koszty eksploatacji systemu, koszty konserwacji systemu. Ponadto już w pierwszym etapie powinno się uwzględniać różne warianty architektury systemu (przetwarzanie scentralizowane, rozproszone), ponieważ różnią się one kosztami.

Następną fazą cyklu życia systemu jest faza *analizy*. Podstawowym celem tego etapu jest otrzymanie *formalnego* opisu działania systemu na podstawie dwóch źródeł danych: statutu projektu wykreowanego na etapie strategii oraz wymagań użytkownika. Strukturalna specyfikacja funkcji systemu powstaje poprzez modelowanie reguł przetwarzania (konstruowanie modelu środowiska użytkownika), czyli ilustrację związków pomiędzy procesami, obiektami zewnętrznymi, zbiorami danych oraz przepływami (strumieniami) danych. Modele tworzone w fazie analizy są modelami graficznymi, najczęściej używanymi narzędziami analizy strukturalnej są wspomniane wcześniej diagramy DFD oraz diagramy ERD (*Entity Relationship Diagram, diagram związków encji*) służące do modelowania danych. Modele graficzne obrazują składowe systemu i sposoby komunikacji między nimi, nie jest to więc pełna dokumentacja projektowa. Inne szczegóły powinny się znaleźć w dokumentach pomocniczych, zawierających m.in. specyfikację procesów czy słownik danych. Wynikiem fazy analizy powinien być model systemu niezależny od technologii implementacyjnej. Na tym etapie dokonuje się również uszczegółowienia budżetu projektu oraz rachunku kosztów i zysków.

Warto zauważyć w tym miejscu, że termin „użytkownik systemu" rzadko odnosi się do pojedynczej osoby; najczęściej projektowany system jest przeznaczony dla różnych grup użytkowników. Przystępując więc do budowania modelu systemu zgodnie z wymaganiami użytkownika, należy zdecydować, w jaki sposób model będzie powstawał. Możliwe są dwa podejścia:

1. Centralistyczne – polegające na gromadzeniu wymagań poszczególnych użytkowników, utworzeniu jednej listy wymagań i opracowaniu na tej podstawie modelu systemu.

2. Integracyjne – polegające na budowaniu oddzielnych modeli, łączonych następnie w model globalny (integrowanie modeli lokalnych).

Podejście integracyjne stosowane jest najczęściej w sytuacjach, kiedy wymagania poszczególnych grup użytkowników znacznie się różnią.

Po fazie analizy następuje faza *projektowania*, w której dokonuje się przekształcenia modelu logicznego systemu w model implementacyjny. Na tym etapie dokonuje się wyboru platformy sprzętowej oraz programowej (systemu sieciowego, systemu operacyjnego, środowiska bazodanowego, narzędzia budowy aplikacji itp.). W przypadku dużych systemów opracowywane są przydziały zadań do odpowiednich procesorów, ustalana jest hierarchia modułów programowych oraz sposób interakcji międzymodułowej. Konceptualny model danych reprezentowany przez diagram ERD przekształcany jest w model logiczny, który w obszarze relacyjnych baz danych jest zbiorem tabel, spełniających warunek przynajmniej trzeciej postaci normalnej. Zasady przekształcania modelu konceptualnego na model logiczny oraz zasady normalizacji tabel omówiono szczegółowo w rozdziałach 7 i 8. Na tym etapie projektowane są również inne obiekty bazy danych, takie jak: procedury, funkcje, indeksy i perspektywy. Określone zostają grupy użytkowników i ich uprawnienia. W fazie tej opracowywany jest także projekt interfejsu użytkownika (sposób komunikacji z systemem). Projektanci systemu powinni uwzględnić dodatkowe czynniki, które mają wpływ na kształt budowanego systemu i dokładnie uzgodnić je z przyszłymi użytkownikami:

- Rozmiary i liczba danych – możliwa jest sytuacja, gdy użytkownik przewiduje znaczne zwielokrotnienie liczby danych przechowywanych w bazie danych, ze względu na rozwój firmy.
- Liczba transakcji – jest istotnym czynnikiem w sytuacjach, gdy transakcje są nierównomiernie rozłożone w czasie i mogą zdarzać się sytuacje znacznego zwiększenia obciążenia zarówno w określonych porach dnia, jak i w określonych dniach tygodnia czy miesiącach w roku.
- Czas reakcji systemu – realnie użytkownik precyzuje wymagania słownie, np. „90% wszystkich transakcji powinno być realizowane w czasie mniejszym niż 2 sekundy", „określony raport powinien być gotowy nie później niż o godzinie 20^{00} każdego dnia", „wszystkie transakcje dotyczące wpłat muszą być przetworzone do północy".
- Niezawodność systemu – użytkownik może określić średni czas między awariami (*mean time between failure* – MTBF), średni czas naprawy (*mean time to repair* – MTTR), lub po prostu określić, że system ma być dostępny w 99,5%, gdzie przyjmuje się, że miara dostępności systemu = MTBF/(MTBF + MTTR) [22].
- Ograniczenia środowiskowe – temperatura, wilgotność, zakłócenia elektryczne, wibracje, hałas, promieniowanie. Często zdarzają się sytuacje, że użytkownik nie precyzuje tego typu uwarunkowań, lecz zakłada milcząco, że system będzie funkcjonował w sposób zadowalający w jego środowisku.
- Bezpieczeństwo – użytkownik może określić wiele ograniczeń dotyczących dostępu do funkcji systemu dla pewnych grup użytkowników, specjalnego zabezpieczenia danych (kodowanie), lub prowadzenia kompletnego dziennika audytorskiego, czyli wykaz wszystkich transakcji wejściowych, odpowiedzi systemu, czy nawet zmian w plikach systemowych.

• Ograniczenia tzw. „polityczne” – użytkownik może określić markę sprzętu, który ma być zastosowany w projektowanym systemie, środowisko programistyczne, środowisko i technologie bazodanowe itp.

Można postawić pytanie, kto zajmuje się realizacją tej fazy, czyli projektem systemu: czy analitycy systemowi przekazują dokumentację zespołowi projektantów i „umywają ręce”? W przypadku dużych systemów na ogół są to odrębne, ale współpracujące ze sobą zespoły, jednak przy budowaniu systemów średnich i małych analityk i projektant to ta sama osoba, której zadaniem jest zarówno opracowanie modelu systemu, jak i wykonanie projektu z uwzględnieniem budżetu, wymagań funkcjonalnych i niefunkcjonalnych użytkownika oraz faktu, że systemy z upływem czasu zmieniają się i rozrastają.

Kolejną fazą jest faza *implementowania projektu*, czyli – inaczej mówiąc – budowanie systemu. W tej fazie odbywa się implementacja schematu bazy danych w wybranym środowisku (określony System Zarządzania Bazą Danych), czyli wygenerowanie skryptu *SQL* tworzącego zaprojektowane obiekty bazy danych. Oprócz bazy danych implementowany jest projekt aplikacji. W zależności od wybranego na etapie projektowania środowiska programistycznego, realizacja fazy implementowania projektu może polegać albo na kodowaniu zaprojektowanych modułów systemu, czyli napisaniu instrukcji w określonym języku programowania (programowanie strukturalne), albo w przypadku wykorzystywania bardzo obecnie popularnych narzędzi typu RAD (*Rapid Application Development*), gdzie istnieją możliwości programowania wizualnego i szybkiego generowania aplikacji – na budowaniu aplikacji z komponentów dostarczanych przez wybrane środowisko, z ewentualnym uzupełnieniem o samodzielne procedury. Ponieważ w czasach obecnych jednym z najważniejszych kryteriów branych pod uwagę w projektowaniu i budowaniu systemów i aplikacji jest czas ich realizacji, techniki programowania i narzędzia podnoszące wydajność programowania znajdują się w centrum uwagi. Stąd bierze się duża popularność narzędzi RAD umożliwiających szybkie budowanie aplikacji bazodanowych, takich jak Delphi, Visual Basic, Visual C++, czy produkt firmy Sybase/Powersoft, któremu poświęcono więcej uwagi w rozdziale 10, czyli PowerBuilder.

Po zakończeniu etapu budowania systemu, zanim zostanie on zainstalowany i wdrożony w środowisku użytkownika, musi zostać przetestowany. Stąd kolejna faza – faza *testowania, instalacji i wdrożenia* systemu. Trzeba nadmienić, że wielu autorów traktuje rozłącznie te etapy, ponieważ każdy z procesów, zarówno proces testowania, jak i instalacji oraz wdrożenia, ma swoją specyfikę i podlega ściśle określonym rygorom. Szczegółowe omówienie tych etapów wykracza poza ramy niniejszego podręcznika, którego głównym celem jest zapoznanie z metodologią wykonywania aplikacji bazodanowych, czyli procesem zamykającym się w czterech początkowych fazach cyklu życia systemu. Dlatego też, nie umniejszając znaczenia ani testowania, ani instalacji, ani wdrażania systemu, przedstawiono je łącznie i w sposób ogólny.

Testowanie systemu może odbywać się według różnych scenariuszy. Dwa najbardziej rozpowszechnione to testowanie wstępujące lub zstępujące. W przypadku testowania wstępującego testom podlegają poszczególne fragmenty systemu, które są łączone w większą całość poddawaną dalszemu procesowi testowania. W końcowej fazie testom podlega cały system. Odwrotny scenariusz, czyli testowanie zstępujące, polega na sprawdzaniu poprawności głównego modułu systemu i interfejsów do modułów niższego poziomu. Kolejno dołączane i testowane są moduły niższego rzędu. Niezależnie od wybranego scenariusza zakres testów powinien uwzględniać najważniejsze aspekty poprawności produktu:

1. Aspekt funkcjonalności – testowanie poprawności wykonywania założonych funkcji.

2. Aspekt efektywności – testowanie czasu reakcji systemu (czasu wykonywania poszczególnych funkcji na wymaganych rozmiarach bazy danych, przy założonej liczbie transakcji).

3. Aspekt niezawodności – testowanie zdolności regeneracji systemu po awariach różnego typu (awarie sprzętu, zasilania, błędów systemu operacyjnego lub sieciowego).

Testowanie systemu (aplikacji) powinno się odbywać według założonego planu testowania, który ma jasno precyzować cel testowania oraz sposób jego przeprowadzenia, czyli określać opis wejść i oczekiwanych wyjść i wyników. Ważne jest ustalenie procedury zgłaszania błędów pojawiających się podczas testów. Oczywistym jest, że testowanie powinno odbywać się w środowisku o konfiguracji identycznej z docelowym środowiskiem pracy systemu, czyli rozmiary pamięci operacyjnej, dyskowej, rodzaje kart graficznych, systemów operacyjnych, systemów sieciowych powinny być zgodne z konfiguracją przyszłych użytkowników.

W przypadku aplikacji i systemów bazodanowych testy powinny uwzględniać przede wszystkim:

- Sprawdzenie działania więzów integralności (usuwanie, dodawanie danych, wprowadzanie górnych i dolnych wartości z zakresu).
- Sprawdzanie mechanizmów pracy współbieżnej (testy powinny być nadmiarowe, tzn. jeżeli założenie projektowe przewiduje jednoczesną pracę 20 użytkowników, testy powinny obejmować większy zakres, np. 30–40 użytkowników). Testowanie pracy współbieżnej powinno obejmować:

– próbę jednoczesnego aktualizowania tego samego wiersza,

– próbę usunięcia wiersza edytowanego w tym samym czasie przez innego użytkownika,

– testowanie blokowania obiektów bazy danych,

– testowanie mechanizmów kontroli dostępu – reakcja na błędną nazwę użytkownika lub hasło,

– testowanie dostępu według przydzielonych uprawnień użytkownika.

Na koniec warto się zastanowić nad przekazaniem użytkownikowi wersji wstępnych sytemu, tzw. wersji *alfa* lub wersji *beta*, co w przypadku dużych systemów ko-

mercyjnych jest normalną strategią testowania produktu. Wersja *alfa* oznacza pierwszą edycję systemu, skierowaną do wybranych odbiorców, z założeniem, że produkt może mieć błędy. Wersja *beta* natomiast z założenia jest poprawną, wstępną wersją produktu, ale ponieważ wyczerpanie wszystkich możliwych przypadków testowych nie jest w praktyce realizowalne (dotyczyć to może zwłaszcza nietypowych sposobów użytkowania systemu lub nietypowych platform sprzętowych), wybrana grupa odbiorców użytkuje produkt, zgłaszając wykonawcy wszelkie anomalie działania. Korzyści płynące z udostępniania wstępnych wersji systemu są trudne do przecenienia – oprócz pogłębienia samego procesu testowania następuje sprawdzenie funkcjonalności i stopnia złożoności obsługi systemu – może się okazać, że pomimo poprawności działania system jest zbyt trudny w obsłudze, wymaga dużych nakładów na przeszkolenie użytkowników, czyli powinien jak najszybciej zostać zmodyfikowany, jeżeli ma być produktem satysfakcjonującym odbiorców.

Po pomyślnym zakończeniu testów można przystąpić do instalacji systemu w środowisku użytkownika. Często instalacja nowego produktu musi być poprzedzona uzupełnieniem dotychczasowego środowiska o dodatkowy sprzęt lub oprogramowanie. Może pojawić się konieczność konwersji dotychczasowych baz danych, programów i procedur na postać wymaganą przez nowy system, co jest zadaniem trudnym i czasochłonnym, niejednokrotnie sprowadzającym się do wykonania dodatkowych aplikacji konwertujących. W przypadku dużych systemów, a zwłaszcza systemów rozproszonych, proces instalacji może przebiegać etapami, w kolejnych lokalizacjach, nie będzie to więc zadanie jednorazowe, lecz rozłożone w czasie. Dopiero po zakończeniu etapu instalacji można przystąpić do szkolenia użytkowników, które znowu uzależnione jest od stopnia złożoności systemu. W przypadku prostych aplikacji czy systemów wystarczająca może okazać się dobrze opracowana dokumentacja techniczna i podręczniki użytkownika. W przypadku systemów dużych prowadzone są szkolenia w formie kursów, których harmonogram jest uzgadniany z użytkownikami.

Ostatnią fazą jest faza *eksploatacji* systemu, zamykająca cykl życia systemu przekazaniem produktu użytkownikowi. Należy pamiętać, że celem całego przedsięwzięcia było wykonanie dobrego systemu, spełniającego wymagania użytkownika; niestety, bardzo często okazuje się, że od początku eksploatacji systemu pojawiają się potrzeby zmian lub rozbudowy. Z całą pewnością ze względu na to, że organizacje czy firmy zmieniają się, systemy też muszą ewaluować. Nie może to być jednak ciągłe wykonywanie poprawek na każde żądanie użytkownika, bo w ten sposób system nie będzie nigdy gotowy. Inny tryb postępowania obowiązuje w przypadku konieczności usunięcia błędów – powinny być usuwane szybko, inny w przypadku rozszerzeń funkcjonalnych – powinna być opracowana kolejna wersja systemu.

Narzędzia wspomagające projektowanie systemów baz danych

Narzędzia wspomagające modelowanie i projektowanie systemów i aplikacji bazodanowych należą do tzw. narzędzi typu CASE (*Computer Aided Software Engineering, Inżynieria oprogramowania wspomagana komputerowo*). Dzisiejsze narzędzia tego typu to najczęściej zintegrowane pakiety wspomagające realizację wszystkich faz cyklu życia systemu. Charakteryzując bardzo ogólnie budowę zintegrowanych narzędzi CASE, można wymienić następujące komponenty wchodzące w skład pakietów:

- słownik danych elementarnych, przechowujący dane używane przez aplikację;
- narzędzia projektowania wspomagające analizę reguł przetwarzania, czyli modelowanie procesów;
- narzędzia projektowania umożliwiające budowę konceptualnego i logicznego modelu danych;
- narzędzia umożliwiające prototypowanie i testowanie aplikacji.

Omawiając niewątpliwe zalety wykonywania projektów z wykorzystaniem możliwości, jakie dają narzędzia CASE, trzeba mieć świadomość, że żaden z pakietów nie działa automatycznie, nie wyręcza projektanta w procesie planowania i tworzenia systemu, lecz jedynie wspomaga. Należy więc pamiętać, że poprawne posługiwanie się pakietami CASE wymaga niezbędnego, określonego zasobu wiedzy i umiejętności z zakresu projektowania zarówno baz danych, jak i aplikacji bazodanowych, ponadto również z zakresu obsługi określonego pakietu. Dlaczego więc narzędzia CASE zdobywają coraz większe uznanie i są chętnie stosowane przez analityków i projektantów? Korzyści płynące z wykonywania projektów informatycznych w środowisku CASE można pokrótce scharakteryzować w kilku punktach. Stosowanie narzędzi CASE:

- zmusza do przestrzegania jednakowych standardów przez wszystkich uczestników projektu,
- zapewnia łatwość integrowania projektu dzięki składowaniu poszczególnych elementów projektu w repozytoriach i słowniku danych,
- pozwala na oszczędność czasu, ponieważ rysowanie ręcznie niektórych diagramów może być trudne i pracochłonne,
- umożliwia automatyzację niektórych prac projektowych, gdyż niektóre z narzędzi ułatwiają transformowanie części projektu na wykonywalne kody, co znacznie przyspiesza fazę implementacji projektu, zmniejszając równocześnie ryzyko popełnienia błędów podczas kodowania.

Obecnie na rynku informatycznym istnieje spory wybór pakietów typu CASE, jednak – zgodnie z założeniami tego podręcznika – przedstawiony zostanie pakiet firmy Sybase – PowerDesigner.

Przykłady praktyczne zamieszczone w kolejnych rozdziałach zostały wykonane za pomocą wersji 6.1 pakietu. Jest to narzędzie oparte na metodologii inżynierii oprogramowania opracowanej przez Jamesa Martina, która opiera się na dwupoziomowym projektowaniu, czyli budowaniu modelu logicznego i fizycznego. W wersji 6.1

w skład pakietu PowerDesigner wchodzi program ProcessAnalyst umożliwiający modelowanie reguł przetwarzania, czyli zobrazowanie kluczowych funkcji aplikacji poprzez wzajemnie powiązane procesy. Projektant ma możliwość wyboru notacji: OMT, Yourdon/DeMarco, SSADM lub Gane&Sarson. ProcessAnalyst pozwala na kompletną definicję przepływu danych w systemie, analizę funkcjonalną z możliwością dekompozycji procesów na podprocesy, eliminację niejednoznaczności modelu oraz generowanie zaawansowanych raportów, stanowiących dokumentację projektową.

Kolejną składową pakietu jest program DataArchitect, wspierający proces modelowania danych poprzez kreowanie diagramów E-R, czyli konceptualnego (logicznego) modelu danych, niezależnego od docelowego systemu baz danych, i przekształcanie tego modelu w model fizyczny, czyli schemat bazy danych z zachowaniem wszystkich zależności zdefiniowanych w modelu logicznym. Model fizyczny związany jest z platformą wybraną przez projektanta. Na poziomie modelu fizycznego można dokonywać normalizacji tabel, tworzenia i modyfikacji procedur zdarzeń, tworzenia indeksów i perspektyw, czyli strojenia bazy danych. Końcowym etapem jest generowanie bazy danych, czyli kodu w języku definicji danych SQL. Pakiet umożliwia generowanie baz danych dla ponad 30 systemów, m.in.: Sybase SQL Server, Sybase SQL Anywhere, Oracle, Informix, Microsoft SQL Server, Progress, Microsoft Access, Paradox.

Warto wymienić jeszcze program AppModeler, współpracujący z pozostałymi modułami pakietu i umożliwiający automatyczne tworzenie aplikacji klient–serwer lub internetowych, czyli szybkie wykonanie prototypu. Obiekty aplikacji generowane są na podstawie fizycznego modelu danych oraz wybranej biblioteki klas dla środowiska PowerBuilder, Power++, Delphi, oraz Visual Basic. Ponadto z poziomu tego pakietu można generować strony WWW, umożliwiające dostęp do bazy danych.

Nie są to wszystkie programy wchodzące w skład pakietu, ale w odniesieniu do zagadnień związanych z procesem projektowania baz danych i aplikacji bazodanowych jest to zestaw umożliwiający projektantowi realizację zamierzeń. Należy również nadmienić, że przedstawiona na razie w sposób ogólny wersja 6.1 pakietu PowerDesigner wspomaga metodykę strukturalną projektowania, wersje najnowsze (PowerDesigner 9.5) są zdecydowanie związane z metodyką obiektową z wykorzystaniem standardów języka UML. Ponieważ jednak głównym celem niniejszego podręcznika nie jest szczegółowe przedstawianie metodyk projektowania, a tym bardziej ich porównywanie, lecz zobrazowanie procesu powstawania aplikacji współpracujących z bazą danych, działających efektywnie, zrozumiałych, łatwo pielęgnowalnych i skalowalnych, punkt ciężkości leży więc nie w wybranej metodzie, lecz w dobrym jej zrozumieniu i właściwym zastosowaniu. W praktyce, ponieważ wiadomo, że czas potrzebny na naukę narzędzi wpłynie negatywnie na czas zakończenia projektu, wybór metodologii zależy od preferencji analityków i projektantów. Trudno bowiem w sytuacji, kiedy zamierzeniem jest opracowanie konkretnego projektu w określonym czasie, wymagać od zespołu realizatorów uczenia się nowych metod i narzędzi wspomagających proces modelowania i projektowania.

6

Analiza systemowa – modelowanie reguł przetwarzania

Jak już wspomniano w poprzednim rozdziale, głównym celem etapu analizy, który jest podstawowym etapem projektowania systemu i w związku z tym wpływa zasadniczo na ostateczny kształt przyszłego produktu jest sformalizowanie wymagań użytkownika systemu, co wiąże się z koniecznością dobrego zrozumienia przez twórców systemu specyfiki środowiska użytkownika. Oczywista wydaje się potrzeba znalezienia platformy porozumienia pomiędzy światem projektanta a użytkownika, gdyż wymagania wyrażane nieformalnie za pomocą języka potocznego mogą być nieprecyzyjne bądź wieloznaczne. Taką platformę stanowi powstający w fazie analizy model logiczny systemu precyzujący, jakie funkcje system musi wykonywać, aby spełniać wymagania użytkownika oraz jakie są między nimi zależności. Na poziomie modelu logicznego precyzuje się, *co* system będzie robił, bez określania, *jak* będzie to robił. W większości przypadków modele funkcjonalne systemów tworzone są metodami graficznymi; jednym z podstawowych narzędzi analizy systemowej jest diagram DFD, czyli diagram przepływu danych [1, 19, 22]. Składowymi modelu środowiska użytkownika, czyli diagramu DFD są następujące elementy:

- Proces (*process*) — Zadanie, które ma wykonać instytucja lub aplikacja. W kontekście modelu systemu jest to fragment systemu przekształcający dane wejściowe na określone wyniki. Graficznie proces jest reprezentowany w zależności od przyjętej notacji jako okrąg, prostokąt z zaokrąglonymi rogami lub prostokąt, z nazwą wpisaną w środek figury. Jako przykład nazwy procesu można podać: „obliczenie podatku", „fakturowanie", „pobranie należności", „przyjmowanie zamówienia".
- Przepływ (*strumień, flow*) — Towary (rzeczy materialne) lub dane przepływające między obiektami zewnętrznymi, procesami lub magazynami. Inaczej mówiąc, przepływ opisuje ruch danych w systemie. Graficznie jest reprezentowany strzałką, która wskazuje kierunek przepływu. Każdy przepływ również powinien być nazwany, zgodnie ze znaczeniem pakietu danych, przenoszonego przez określony przepływ. Przykła-

dowo: „zapytanie klienta", „dane zamówienia", „dane reklamacji".

- Magazyn (*store*) — Magazyn obrazuje dane wykorzystywane przez system (poprzez zapisywanie, modyfikowanie lub odczytywanie). W przypadku systemów bazodanowych rolę magazynów pełnią pliki bazy danych. W odróżnieniu od przepływów, gdzie mamy do czynienia z ruchem danych, dane w magazynach pozostają w spoczynku. Graficznie magazyny danych obrazowane są najczęściej przez dwie równoległe linie, pomiędzy które wpisuje się nazwę magazynu. Nazwa magazynu pochodzi od nazwy pakietu, przenoszonego przez przepływ do/z magazynu. Przykładowo: „zamówienia", „reklamacje", „klienci".
- Obiekt zewnętrzny (*terminator, external entity*) — Osoba, instytucja, inny system znajdujący się poza systemem projektowanym, ale komunikujący się z nim. Obiekty zewnętrzne są źródłami lub odbiorcami informacji. W przypadku modelowanego systemu nie są istotne żadne związki między terminatorami, ponieważ znajdują się one poza granicami systemu. Graficznie terminatory obrazowane są przez prostokąt z wpisaną nazwą, np. „dział księgowości", „klient", „najemca".
- Elementy danych (*data items*) — Dane elementarne wchodzące w skład pakietów przenoszonych przez przepływy.

Model logiczny projektowanego systemu ilustruje związki między wymienionymi elementami.

6.1. Model środowiskowy

Pierwszym zadaniem analityka jest określenie granic powstającego systemu, czyli granicy pomiędzy systemem a środowiskiem jego działania oraz sposobów komunikacji systemu z otoczeniem, czyli interfejsów. W tym celu budowany jest tzw. *model środowiskowy*, który obrazuje, jakie obiekty zewnętrzne są w interakcji z powstającym systemem, dostarczając i odbierając dane lub sygnały. W drugim kroku powstaje tzw. model *behavioralny*, czyli model *zachowania*, który obrazuje wewnętrzne zachowanie systemu realizujące oczekiwania użytkownika.

W modelu środowiskowym występują trzy składowe:

1. Określenie celu systemu.
2. Diagram kontekstowy.
3. Lista zdarzeń.

Pierwszą składową, czyli określenie celu, omówiono w rozdziale poprzednim. Należy pamiętać, że określenie celu musi mieć krótką i zwięzłą formę tekstową.

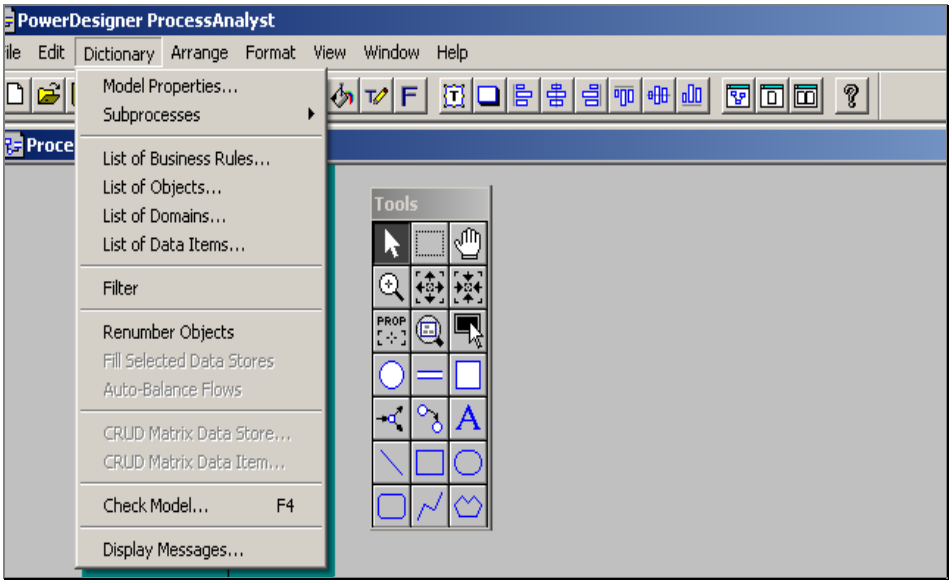
Składowa druga, czyli diagram kontekstowy, jest szczególnym przypadkiem diagramu przepływu danych, uwypuklającym pewne cechy projektowanego systemu, chociażby przez to, że diagram obrazuje zarówno dane, które system otrzymuje z zewnątrz i które muszą być przetworzone, jak i zwrotnie – dane, które produkuje system muszą zostać przesłane do obiektów zewnętrznych.

Składowa trzecia modelu środowiskowego to lista zdarzeń, czyli lista bodźców występujących w świecie zewnętrznym, na które system musi zareagować.

Omawiane w dalszej części przykłady modeli środowiskowych i modeli zachowania zostały skonstruowane za pomocą programu ProcessAnalyst z pakietu PowerDesigner, dlatego też ich przedstawienie musi poprzedzać krótka charakterystyka narzędzia oraz wskazówki dotyczące korzystania z programu [23].

6.2. Zasady pracy z programem ProcessAnalyst

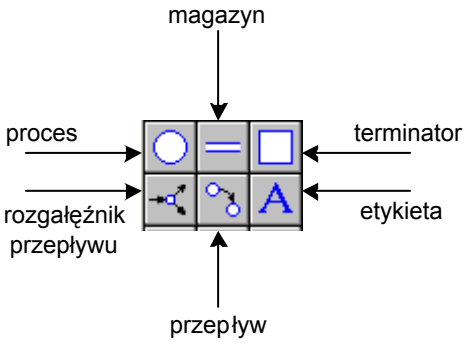
Po wywołaniu programu na ekranie pojawia się pole projektowe z widoczną paletą narzędzi i belką menu (rys.6.1). Układ menu jest podobny we wszystkich programach pakietu PowerDesigner.



Rys. 6.1. Belka menu i paleta narzędzi programu ProcessAnalyst

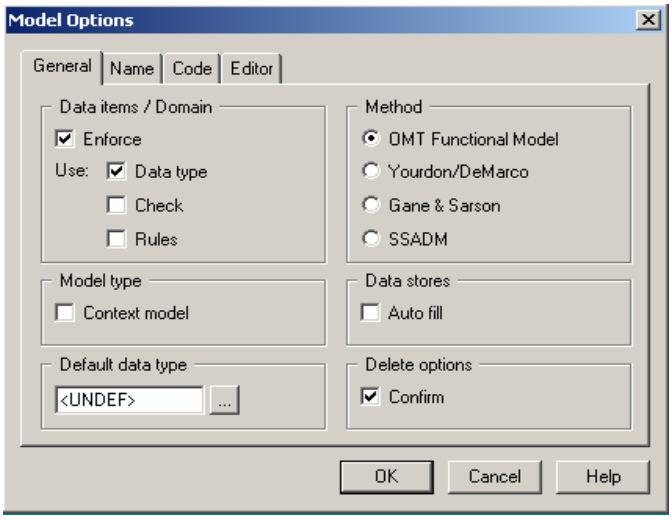
Rysunek 6.1 prezentuje opcje menu *Dictionary*, czyli *słownika* pakietu. Opcje ilustrują sposób działania pakietu – wszystkie elementy umieszczane w polu projektu są

traktowane jako obiekty, których charakterystyki przechowywane są w tzw. słowniku programu. Paleta narzędzi po wywołaniu programu znajduje się z lewej strony pola projektowego. Większość elementów składowych diagramu DFD dostępna jest na palecie, w środkowej jej części, co ilustruje rys. 6.2.



Rys. 6.2. Narzędzia modelowania na palecie programu ProcessAnalyst

Przed rozpoczęciem konstruowania modelu należy wybrać odpowiednią notację poprzez wybór z menu *File* opcji *Model Options*, co powoduje wyświetlenie okna z dostępnymi notacjami (rys. 6.3).



Rys. 6.3. Notacje dostępne w programie ProcessAnalyst

Uwaga: W przykładach zamieszczonych w niniejszym rozdziale modele zostały skonstruowane zgodnie z notacją OMT.

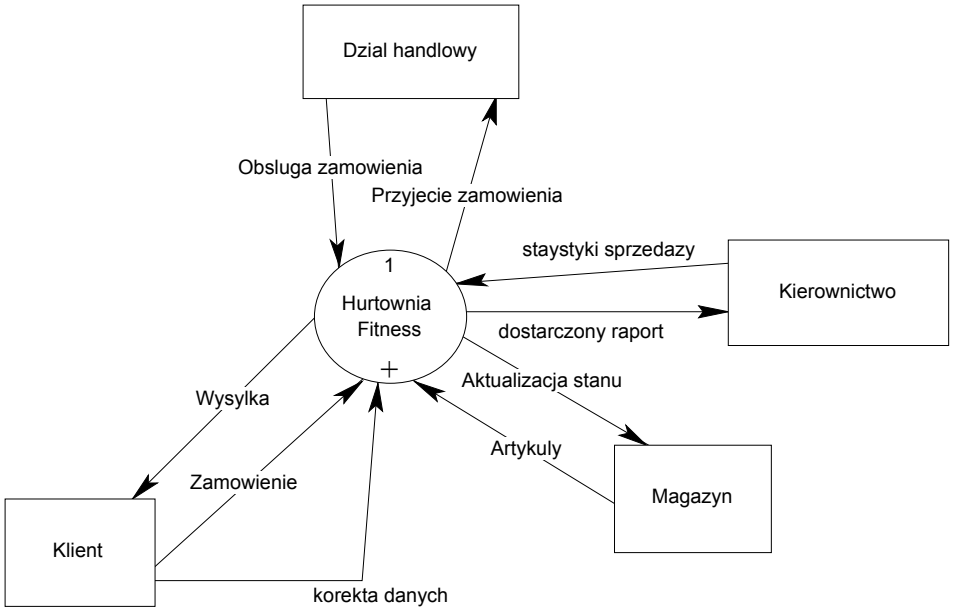
6.3. Przykłady modeli środowiskowych

Jak już wspomniano, dekompozycję funkcjonalną systemu rozpoczyna się od modelu środowiskowego, którego składową jest diagram kontekstowy; w środowisku ProcessAnalyst nazywany procesem nadrzędnym (*root process*). Proces nadrzędny obrazuje cały system, jest więc procesem grupującym wszystkie inne procesy wchodzące w skład analizowanego systemu. Proces nadrzędny kontaktuje się ze środowiskiem zewnętrznym reprezentowanym przez terminatory (encje zewnętrzne).

Przykład 6.1

Przykład dotyczy działalności hurtowni odzieży sportowej. Firma ma swoich klientów, którzy składają zamówienia na określone artykuły. Każde z zamówień jest obsługiwane przez określonego pracownika z działu obsługi klienta. Zamówienie jest realizowane przez wystawienie faktury i wysyłkę towaru. Celem systemu *Hurtownia Fitness* jest wspomaganie zarządzania hurtownią. Główne funkcje systemu:

- rejestrowanie i obsługa zamówień klientów,
- generowanie faktur dla klientów,
- obsługa stanu magazynu,
- sporządzanie statystyk sprzedaży.



Rys. 6.4. Projekt systemu *Hurtownia Fitness* – diagram kontekstowy

Zgodnie z zasadą działania pakietu, procesowi nadrzędnemu reprezentującemu ogólnie cały system został przyporządkowany numer 1. W następnych krokach proces

ten będzie poddany dekompozycji (aż do poziomu procesów elementarnych), w celu bardziej szczegółowego opisu funkcji systemu.

Lista zdarzeń dla systemu Hurtownia Fitness:

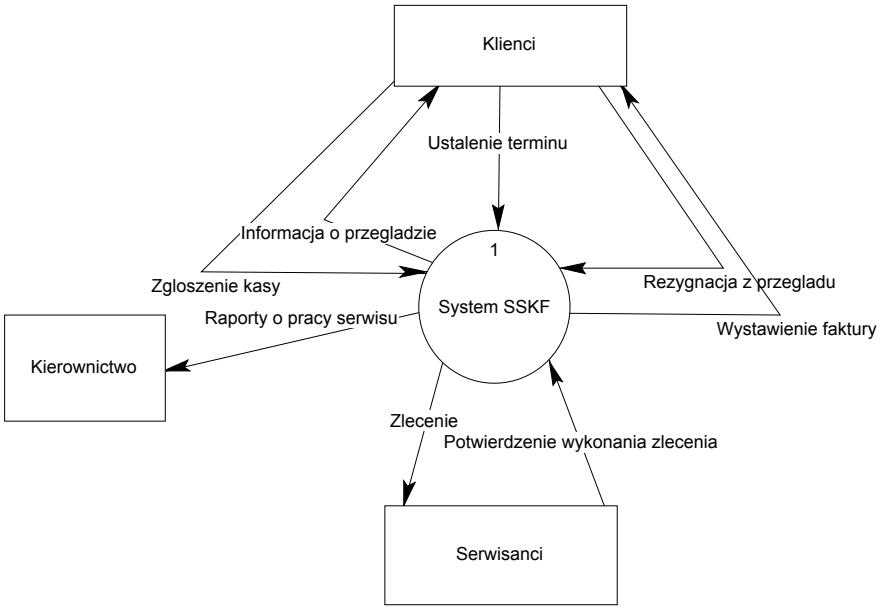
1. Klient składa zamówienie.
2. Klient informuje o zmianie adresu.
3. Magazyn dokonuje wysyłki zamówionych artykułów.
4. Magazyn dokonuje aktualizacji stanów.
5. Dział handlowy informuje o wysyłce.
6. Dział handlowy wystawia fakturę.
7. Kierownictwo zamawia statystyki sprzedaży.

Przykład 6.2

Kolejny przykład dotyczy firmy serwisującej kasy fiskalne w zakresie obowiązkowych przeglądów kas zainstalowanych u klientów. Przeglądy takie muszą być wykonywane w określonych przez przepisy podatkowe odstępach czasu, klient ma jednak prawo odmówić przeglądu. Przeglądu dokonują serwisanci danej kasy z odpowiednimi uprawnieniami. System powinien przechowywać informacje o zainstalowanych kasach i sygnalizować zbliżający się przegląd. Po zasygnalizowaniu terminu przeglądu firma serwisująca ustala z klientem termin przeglądu.

Model środowiskowy dla systemu wspomagającego pracę serwisu kas fiskalnych – system SSKF.

Celem systemu SSKF jest wspomaganie zarządzania serwisowaniem kas fiskalnych.



Rys. 6.5. Projekt systemu SSKF – diagram kontekstowy

Lista zdarzeń dla systemu SSKF:

1. Zgłoszenie kasy.
2. Informacja o przeglądzie.
3. Ustalenie terminu.
4. Rezygnacja z przeglądu.
5. Zlecenie dla serwisanta.
6. Potwierdzenie wykonania zlecenia.
7. Wystawienie faktury klientowi.
8. Raport dla kierownictwa.

Przykład 6.3

Przykład dotyczy sieciowego systemu prowadzenia ewidencji prac dyplomowych i rejestru absolwentów wydziału uczelni.

Obszar modelowania

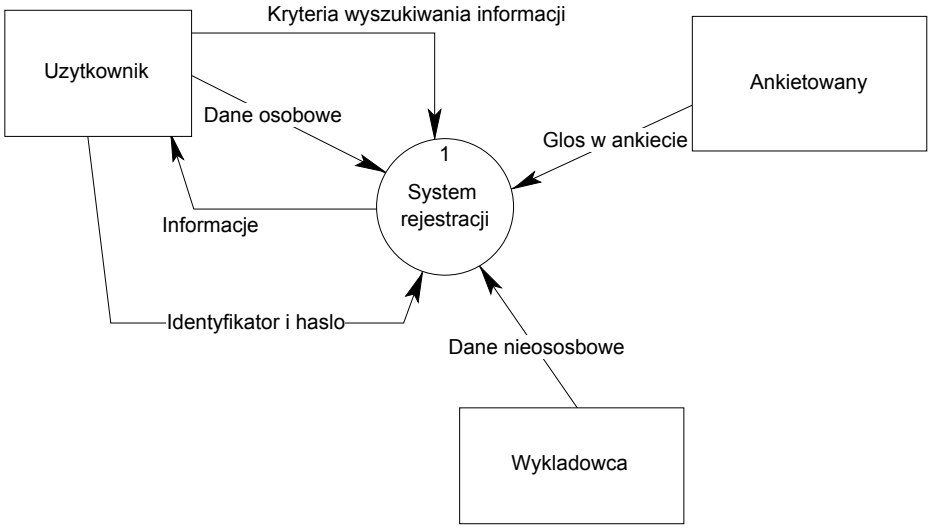
W skład wydziału wchodzą określone jednostki organizacyjne (zakłady, katedry, instytuty). Jednostki organizacyjne są odpowiedzialne za prowadzenie specjalności, wybieranych przez studentów określonego kierunku studiów prowadzonego przez wydział. Wykładowcy prowadzący zajęcia są zawsze związani z określoną jednostką organizacyjną. Oprócz zajęć kursowych (wykłady, ćwiczenia, laboratoria, projekty, seminaria) wykładowcy oferują tematy prac dyplomowych. Każda z prac dyplomowych, proponowana dla konkretnej specjalności, ma sprecyzowany zakres tematyczny, typ (magisterska lub inżynierska), wymagania dotyczące realizacji pracy oraz termin jej zakończenia, czyli datę obrony pracy. Tematy prac dyplomowych generowane są corocznie w semestrze zimowym. Wykładowcy często przeprowadzają wśród studentów ankiety, dotyczące zarówno sposobu prowadzenia zajęć kursowych, jak i prac dyplomowych. Każdy student po pomyślnej obronie pracy dyplomowej staje się absolwentem wydziału i uczelni. Okazuje się, że bardzo przydatne jest posiadanie aktualnego rejestru absolwentów – zarówno z punktu widzenia władz wydziału, jak i samych absolwentów. Dla takiego środowiska sprecyzowany został cel projektowanego systemu:

Celem systemu jest wspomaganie ewidencjonowania prac dyplomowych, ankietowania studentów oraz prowadzenia rejestru absolwentów wydziału.

Ogólne założenia projektowe

Docelowi użytkownicy systemu (wykładowcy, studenci, absolwenci) mogą zakładać sobie tzw. konta, na których będą przechowywane informacje na ich temat. Informacje te mogą być następnie wyszukiwane przez innych użytkowników systemu według określonych kryteriów (np. wyszukiwanie wykładowców z określonej specjalności, wyszukiwanie studentów o konkretnym nazwisku). Każdy użytkownik dokonujący rejestracji w systemie określa swój identyfikator (np. student lub absolwent – numer indeksu, wykładowca – numer PESEL lub inny identyfikator niezarezerwowa-

ny przez innego użytkownika). Oprócz tego każdy użytkownik precyzuje hasło umożliwiające autoryzację. Każdy wykładowca może umieszczać lub modyfikować w systemie ogłoszenia kierowane do studentów i absolwentów. Każde ogłoszenie ma swój tytuł, treść i datę wygaśnięcia. Z ogłoszeniem mogą być powiązane tzw. załączniki przeznaczone dla odbiorców (dokumenty, rysunki, materiały archiwalne). System umożliwia również zamieszczanie lub modyfikowanie ankiet dla studentów i absolwentów przez prowadzących zajęcia. Każda ankieta, podobnie jak ogłoszenie, ma swój tytuł, komentarz, datę wygaśnięcia oraz oczywiście treść w postaci punktów. Modyfikacje poszczególnych danych w systemie (konta, tematy prac dyplomowych, ogłoszenia, ankiety) zawsze wymagają autoryzacji użytkownika odpowiedzialnego za te dane.



Rys. 6.6. Diagram kontekstowy systemu rejestracji absolwentów i prac dyplomowych

Lista zdarzeń:

1. Żądanie założenia konta przez użytkownika (podanie danych osobowych).
2. Logowanie użytkownika w systemie (identyfikator i hasło).
3. Żądanie potrzebnych użytkownikowi informacji poprzez sprecyzowanie kryteriów wyboru informacji.
4. Oddanie głosu w ankiecie.
5. Zamieszczenie ogłoszeń lub ankiety przez wykładowcę.

Przykład 6.4

Projektowany system jest przeznaczony dla firmy pośredniczącej w hurtowej sprzedaży towarów.

Obszar modelowania

Firma, dla której projektowany jest system komputerowy zajmuje się pośrednictwem w obrocie hurtowym towarami. Udostępnia producentom powierzchnię w magazynie-hurtowni. Za dostawę towaru odpowiedzialny jest producent. Magazyn jest podzielony na działy, co ułatwia zarządzanie nim (przeprowadzanie inwentaryzacji towarów, wykonywanie statystyk itp.). Lista działów tworzona jest podczas wdrażania systemu.

Proces przyjęcia i wydania towaru z magazynu odbywa się w następujący sposób: gdy towar trafia do magazynu, jest umieszczany w obszarze składowania (do tego obszaru kupujący nie mają dostępu). Jeśli producent (dostawca) przywiezie do magazynu ilość towaru przekraczającą górny limit, to paleta jest przyjmowana, ale dostawca otrzymuje komunikat o przekroczeniu limitu, a system rejestruje przekroczoną ilość. Sprawdzanie limitów odbywa się każdego dnia o ustalonej porze (np. o północy). Za przechowywanie nadmiarowych ilości towarów naliczane są kary, ustalone z dostawcami.

Obszar sprzedaży obsługują magazynierzy – gdy w obszarze sprzedaży zaczyna brakować towaru, jest on wówczas przekładany z obszarów składowania. Magazynierzy muszą dysponować informacjami o położeniu towarów w obszarze składowania (najczęściej obszar składowania związany jest z danym działem magazynu).

Stan towaru w magazynie może się zwiększyć po przyjęciu towaru od producenta, wpisaniu ujemnych różnic inwentaryzacyjnych, a także w wyniku kontroli dat gwarancji lub przyjęcia towaru po przepakowaniu. Zmniejszenie stanu towaru następuje w wyniku sprzedaży towaru, uszkodzenia towaru w magazynie, przeterminowania towaru (zwrot do producenta), w wyniku inwentaryzacji, w wyniku kontroli dat gwarancji oraz podczas wydania towaru do przepakowania.

Operacja przepakowania może nastąpić w przypadku, gdy firma będzie chciała sprzedać towar w innych opakowaniach jednostkowych niż te, które oferuje producent, lub w przypadku tworzenia opakowań promocyjnych (np. do margaryny dodawany jest cukier waniliowy). Podczas operacji przepakowywania w systemie zmniejszana jest ilość towaru przepakowanego, po czym towar jest ponownie przyjmowany z innym kodem produktu.

W przypadku towarów przeterminowanych następuje zwrot towaru do producenta, który o fakcie zbliżania się daty przydatności towaru (w zależności od okresu trwałości produktu oraz indywidualnych wskazań dostawcy) jest informowany odpowiednio wcześniej. Zadaniem magazynierów jest sterowanie przemieszczaniem towaru wewnątrz magazynu zgodnie z zasadą, że towar o najwcześniejszej dacie przydatności znajduje się w obszarze sprzedaży.

Podstawową operacją zmniejszającą stany magazynu jest operacja sprzedaży, której algorytm działania polega na wczytaniu kodu kreskowego towaru i pobraniu ceny towaru. Po zakończeniu tej operacji zmniejszana jest ilość sprzedanego towaru w magazynie. Inne operacje zmniejszające stan magazynu różnią się od operacji

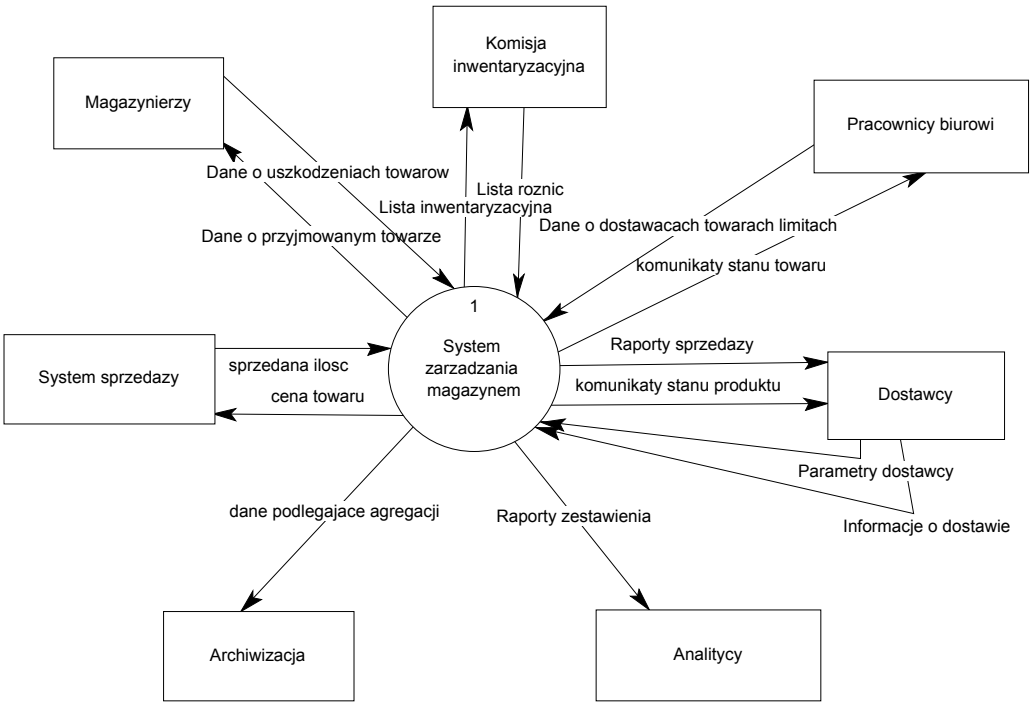
sprzedaży tym, że nie zachodzi potrzeba pobierania ceny towaru; w systemie operacje zmniejszające stany magazynowe są rozróżniane przez odpowiednie kody operacji, używane w zestawieniach statystycznych. Operacje zmniejszające stany magazynu mają związek z informacjami udzielanymi producentom. Są oni odpowiedzialni za dostarczanie towarów, muszą być więc informowani o ilości sprzedanego towaru, o ilości pozostającej w magazynie, jak również o wyczerpywaniu się zapasów.

Kolejnym istotnym aspektem działalności w tego typu firmach jest inwentaryzacja towarów. Może się ona odbywać w określonych przepisami odstępach czasu, na życzenie kierownictwa lub na życzenie dostawcy. Zakres inwentaryzacji może obejmować cały magazyn, wybrane działy, produkty wybranych dostawców lub określone kategorie towarów. Każdy rodzaj inwentaryzacji rozpoczyna się wydrukiem list stanów magazynowych. Następnie powołana komisja sprawdza faktyczny stan towarów i odnotowuje różnice. Ostatnim etapem jest odnotowanie zauważonych rozbieżności.

Niewątpliwie podczas prowadzenia tego rodzaju działalności istotne jest prowadzenie statystyk i zestawień, ułatwiających podejmowanie decyzji strategicznych związanych z firmą. Przykładem mogą być statystyki dzienne, dotyczące sprzedanych towarów, przyjętych produktów, ilości towaru w magazynie, lub statystyki o większym horyzoncie czasowym – odnoszące się przykładowo do czasu reakcji producenta na komunikaty dotyczące wyczerpywania lub przeterminowania towaru, czy też zestawienia najlepiej i najgorzej sprzedających się towarów.

Użytkownicy systemu (terminatory) oraz ich wymagania wobec systemu:

- Magazynierzy – z punktu widzenia tych użytkowników system musi umożliwiać przyjmowanie towaru do magazynu, wprowadzanie danych o uszkodzeniach towaru w magazynie, wydawanie zwrotów towaru do producentów.
- System sprzedaży – uzyskiwanie danych na temat cen produktów na podstawie kodu kreskowego.
- Pracownicy biurowi – możliwość zarządzania danymi dostawców, towarów, limitów stanu (stan minimalny, maksymalny, stan ostrzegawczy), możliwość administrowania listą działów, aktualizacja daty gwarancji towaru.
- Analitycy – system powinien umożliwiać tworzenie różnego rodzaju raportów i zestawień, na podstawie których wyznaczane są np. wskaźniki popytu na poszczególne produkty.
- Komisje inwentaryzacyjne – pobieranie list stanów magazynowych według ustalonego rodzaju inwentaryzacji, wprowadzanie wykrytych różnic inwentaryzacyjnych.
- Dostawcy – stanowiący najważniejszą grupę użytkowników, co wynika z przyjętego modelu działania firmy (odpowiedzialność za uzupełnianie stanów magazynowych jest w gestii dostawców). Dostawcy będą przesyłać do systemu informacje o nowych produktach, zmianach cen, nakazy wycofania towaru, system natomiast powinien wysyłać do dostawców komunikaty o bliskim osiągnięciu poziomu stanu



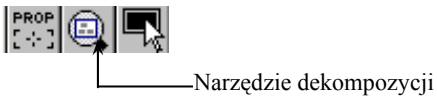
Rys. 6.7. Diagram kontekstowy *Systemu Zarządzania Magazynem*

(limit dolny, górny lub zdefiniowany poziom ostrzegawczy), informacje o terminach upływania dat ważności towarów oraz statystyki umożliwiające producentom kształtowanie ceny produktów i wielkość produkcji.

6.4. Modele zachowania

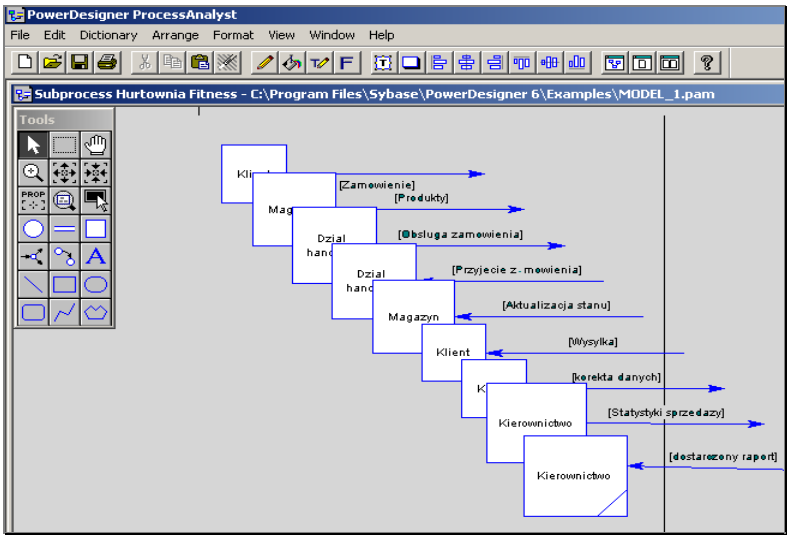
Model zachowania (model behawioralny) opisuje wewnętrzne zachowanie systemu w odpowiedzi na bodźce z otoczenia zewnętrznego, zobrazowanego za pomocą modelu środowiska. Inaczej mówiąc, model zachowania w postaci diagramu DFD obrazuje, w jaki sposób obiekty zewnętrzne (terminatory), procesy i magazyny danych są ze sobą powiązane, jakich transformacji danych wejściowych na dane wyjściowe dokonuje system, dokąd dostarczane są wyniki. Metoda budowania modelu zachowania za pomocą pakietu ProcessAnalyst opiera się na klasycznym podejściu zstępującym (*top-down*). Istota tej metody polega na dekomponowaniu pojedynczego procesu występującego na diagramie kontekstowym i reprezentującego cały system na podsystemy, czyli procesy realizujące poszczególne funkcje systemu. Dekompozycja po-

szczególnych procesów odbywa się do momentu osiągnięcia poziomu procesów elementarnych. Otrzymane diagramy przepływu danych, uporządkowane hierarchicznie poziomami, reprezentują główne składowe funkcjonalne systemu, nie zawierają jednak informacji szczegółowych. Kompletny model zachowania oprócz diagramu DFD wymaga utworzenia słownika danych, czyli uporządkowanego wykazu wszystkich danych mających związek z systemem oraz dokonania specyfikacji procesów. Należy pamiętać, że tylko procesy elementarne opisywane są za pomocą specyfikacji mających charakter opisu algorytmicznego. Metody notacji dla słownika danych oraz specyfikacji procesów omówiono w dalszej części rozdziału.



Rys. 6.8. Symbol narzędzia umożliwiającego dekompozycję procesu

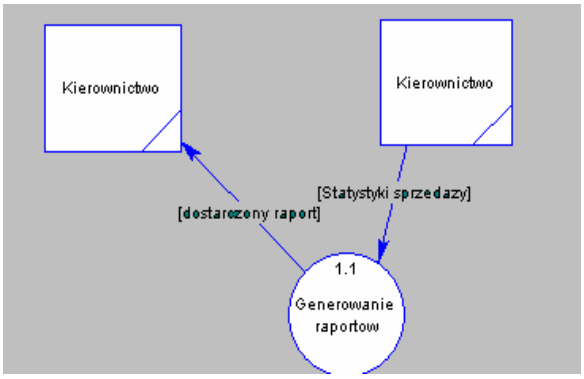
Dekompozycji procesu głównego w polu projektu programu ProcessAnalyst dokonuje się narzędziem dekompozycji znajdującym się na palecie narzędzi (rys. 6.8) lub przez użycie prawego przycisku myszki i wybranie z menu wyskakującego polecenia *Decompose*. Program otwiera nowe okno projektu, w którym widoczne będą wszystkie terminatory wraz z dołączonymi przepływami danych migrującymi z poziomu wyższego.



Rys. 6.9. Widok pola projektu po dekompozycji procesu głównego z przykładu 6.1

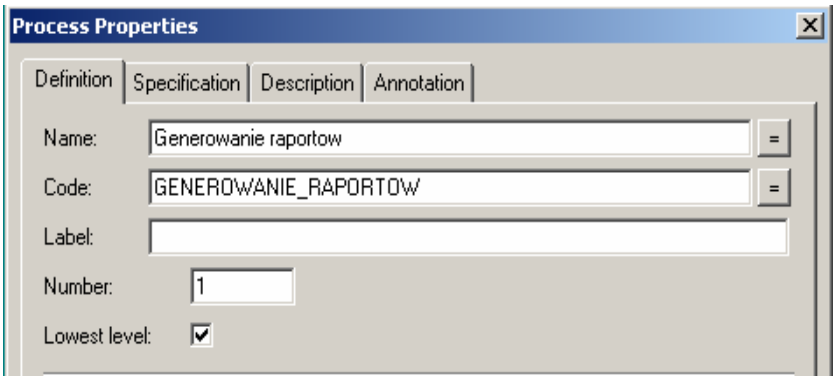
Następnym krokiem w procesie budowy diagramu DFD jest umieszczenie w modelu podprocesów wchodzących w skład procesu głównego. Podprocesy muszą być dołączone do przepływów widocznych w modelu. Zanim więc projektant użyje narzę-

dzia dekompozycji, powinien dokładnie przemyśleć, jakie podprocesy wchodzą w skład procesu nadrzędnego. W tym momencie jasna staje się rola i zadania narzędzia CASE – wspomaganie analityka czy projektanta, ale nie wyręczanie w pracy koncepcyjnej. Biorąc pod uwagę aspekt techniczny postępowania, w modelu z przykładu 6.1 został umieszczony symbol podprocesu generowania raportów, których żąda kierownictwo firmy. Podproces generowania raportów musi być związany z dwoma przepływami danych łączącymi obiekt zewnętrzny, jakim jest kierownictwo firmy z planowanym podprocesem generującym raporty dotyczące sprzedaży towarów.



Rys. 6.10. Fragment modelu dla przykładu 6.1 z podprocesem generowania raportów

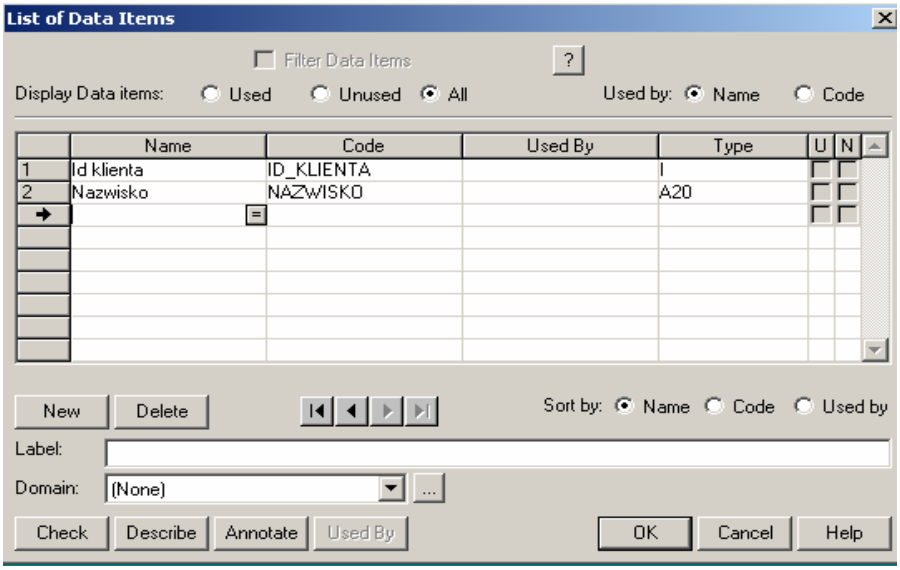
Jeśli projektant decyduje, że podproces nie będzie podlegał dalszej dekompozycji, czyli że jest to proces elementarny, należy tę decyzję zaznaczyć w oknie definicji procesu wybierając opcję *Lowest level* (rys. 6.11).



Rys. 6.11. Wybór poziomu dekompozycji w definicji podprocesu

Zgodnie z definicją, przepływy danych obrazują ruch danych w systemie, czyli między elementami modelu. Inaczej mówiąc, przepływy danych reprezentują związki

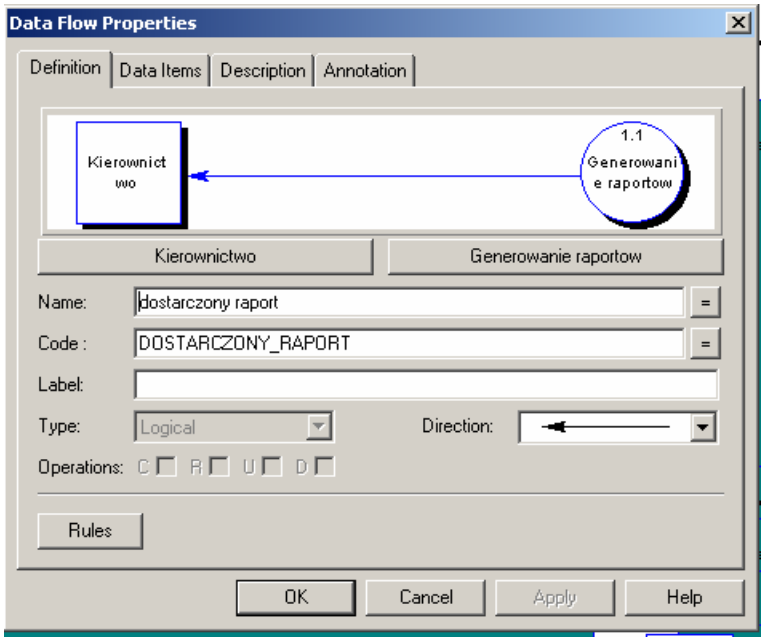
między procesami (funkcjami systemu). Przepływy mogą transportować pojedyncze elementy danych lub całe pakiety danych składające się z pojedynczych elementów. Na obecnym etapie tworzenia modelu przepływy mają jedynie swoje nazwy, które reprezentują znaczenie pakietów poruszających się wzdłuż przepływu. Program ProcessAnalyst wymaga, aby z każdym przepływem związane zostały faktyczne elementy danych, inaczej podczas sprawdzania modelu pojawią się komunikaty ostrzegające, że przepływ jest pusty (nie przenosi danych). Kolejnym etapem tworzenia modelu jest więc zdefiniowanie elementów danych związanych z systemem oraz dziedzin danych identyfikujących typ elementu danych, a następnie przypisanie elementów danych do odpowiednich przepływów. Definiowanie danych elementarnych odbywa się poprzez wybór z menu polecenia *Dictionary →List of Data Items*, co powoduje pojawienie się okna dialogowego umożliwiającego definiowanie danych (rys. 6.12).



Rys. 6.12. Definiowanie elementów danych za pomocą okna dialogowego

Po zdefiniowaniu danych elementarnych można je przyłączyć do odpowiednich przepływów, uwzględniając bardzo istotną uwagę: procesy i przepływy danych są **obiektami lokalnymi** – należą do określonego poziomu diagramu, natomiast terminatory i magazyny danych są **obiektami globalnymi** – występują na każdym poziomie dekompozycji.

Po określeniu elementów danych można na każdym poziomie dekompozycji skojarzyć odpowiednie elementy danych z przepływem. Odbywa się to poprzez uruchomienie edycji przepływu podwójnym kliknięciem myszki, powodującym otwarcie okna dialogowego, z zakładką *Data Items* (rys. 6.13).



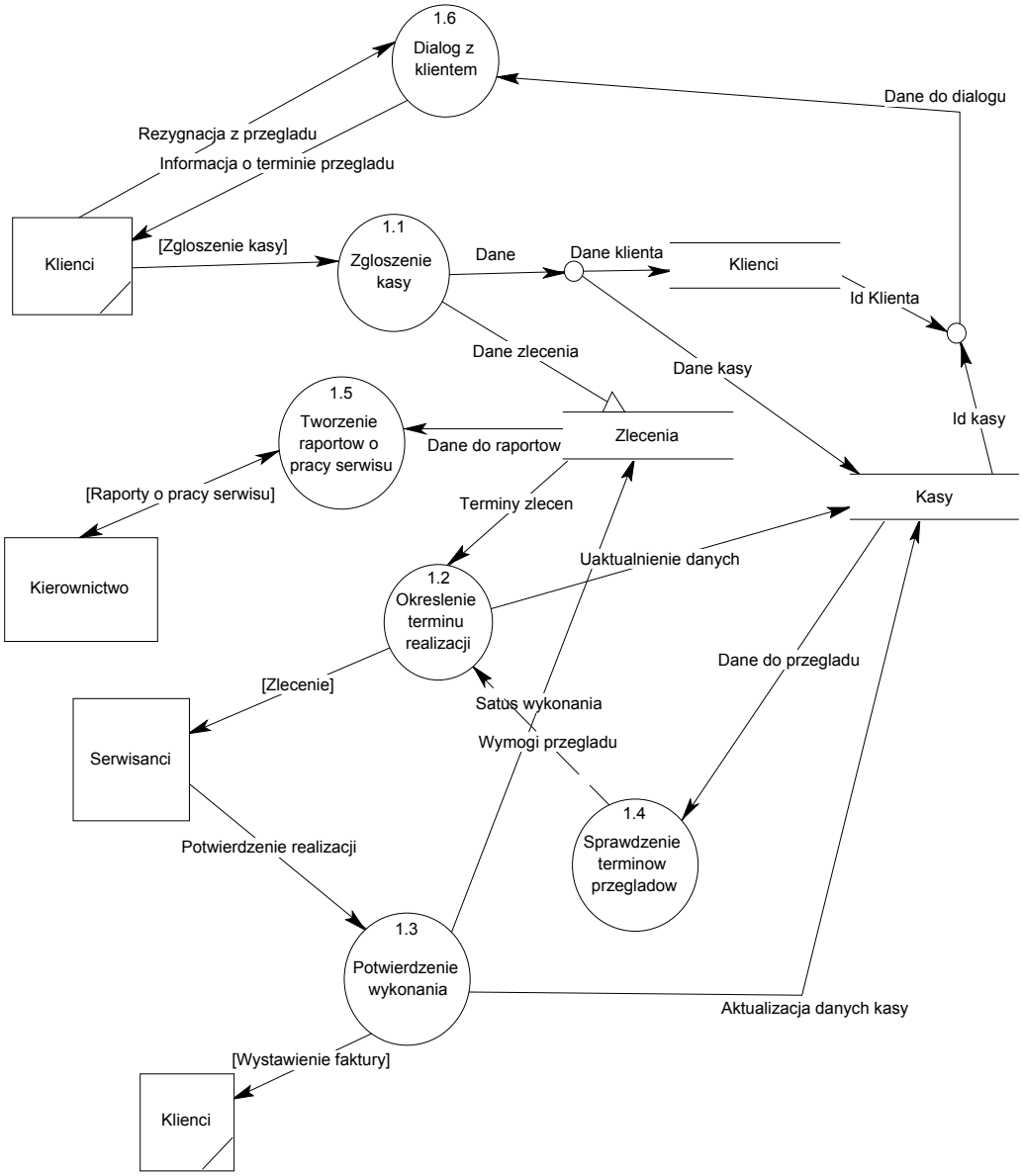
Rys. 6.13. Okno dialogowe umożliwiające skojarzenie elementów danych z przepływem

Kolejnymi obiektami, które wchodzą w skład modelu są magazyny danych reprezentujące dane w spoczynku. Inaczej mówiąc, magazyny danych są zbiorami (bardzo często są to pliki bazy danych), na których działają procesy, odczytując, modyfikując lub zapisując dane. Powiązanie między procesem a magazynem danych reprezentuje przepływ danych, zrozumiały więc jest sposób działania pakietu ProcessAnalyst umożliwiający w automatyczny sposób „umieszczenie" danych przenoszonych przez przepływ w docelowym magazynie. Taki sposób „wypełniania" magazynów danych wymaga ustawienia odpowiedniej opcji modelu w menu *File*. Okno dialogowe umożliwiające wybór automatycznego wypełniania magazynów zostało przedstawione na rysunku 6.3 – ustawienia opcji *Auto Fill* dokonuje się w panelu *Data Stores*.

Na każdym etapie budowania modelu można sprawdzać jego poprawność, wykorzystując opcję menu *Dictionary* → *Check model*. Należy pamiętać, że pakiet sprawdza poprawność modelu pod względem formalnym, a nie logicznym, tzn. czy procesy mają wejścia i wyjścia, czy z przepływami danych związane są pakiety lub elementy danych, czy przepływy danych są związane z innymi obiektami modelu itp. W przypadku wykrycia błędów tego typu drukowane są komunikaty o błędach i ostrzeżenia. Mechanizmem ułatwiającym kontrolę logiczną modelu jest tworzona automatycznie przez pakiet macierz krzyżowa, tzw. macierz CRUD (rys. 6.20).

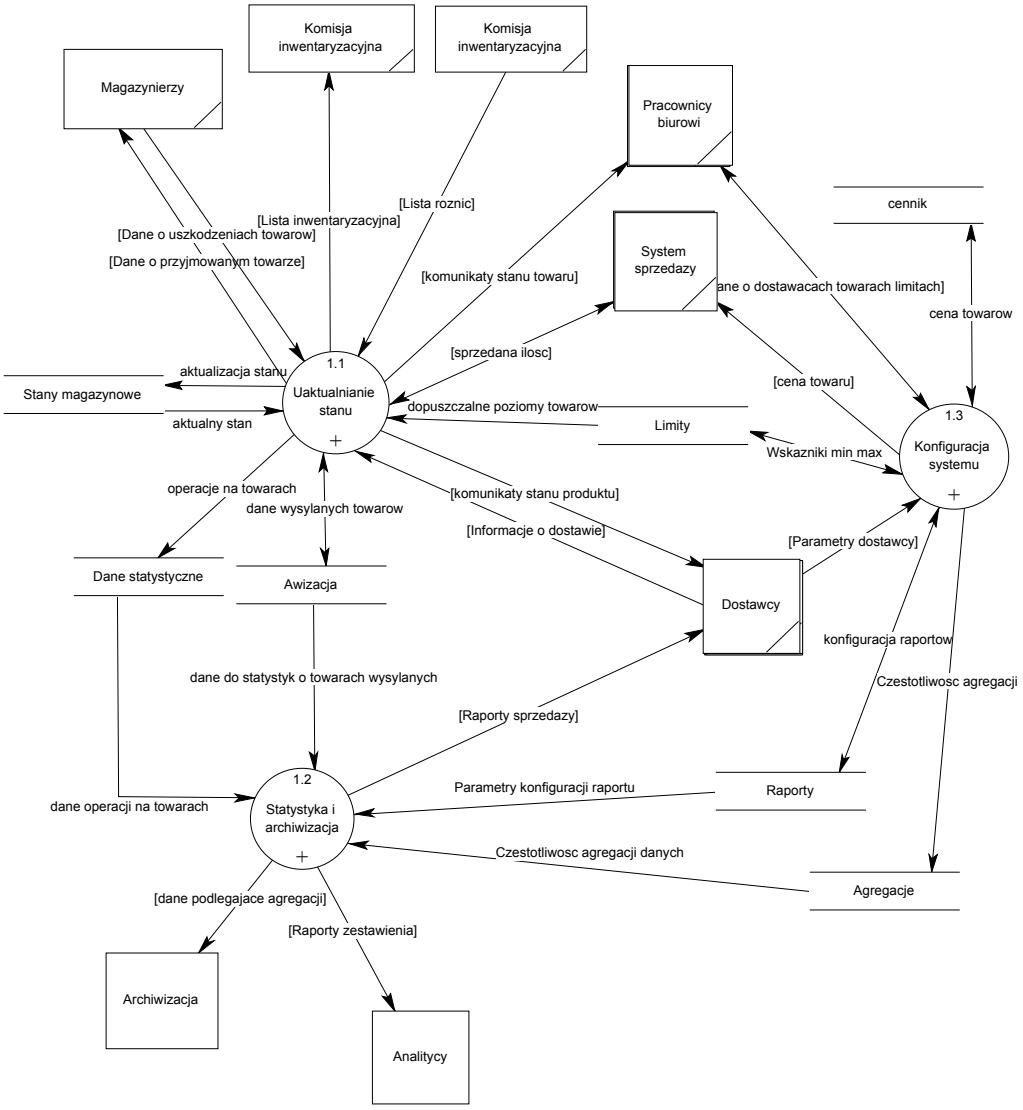
Na kolejnych stronach zamieszczono wybrane diagramy DFD, będące wynikiem dekompozycji diagramów kontekstowych omawianych w pierwszej części rozdziału.

Kontynuacja *przykładu 6.2* – diagram DFD dla systemu SSKF



Rys. 6.14. Diagram ogólny dla systemu SSKF (diagram DFD poziomu 0)

Kontynuacja *przykładu 6.4* – diagramy DFD dla Systemu Zarządzania Magazynem



Rys. 6.15. Diagram ogólny Systemu Zarządzania Magazynem – pierwszy poziom dekompozycji

Uwagi:

Proces główny został zdekomponowany na trzy procesy:

1.1. **Uaktualnianie stanu** – proces (podsystem) zapewniający poprawność stanów magazynowych dla poszczególnych towarów. Zmiana stanu może być spowodowana różnymi przyczynami, pochodzącymi z różnych źródeł (uszkodzenia, przyjęcia, różnice inwentarzowe).

1.2. **Statystyka i archiwizacja** – proces (podsystem) realizujący generowanie różnego rodzaju raportów i statystyk oraz archiwizację danych.

1.3. **Konfiguracja systemu** – proces (podsystem) obsługujący zarządzanie i administrowanie systemem, zarówno w zakresie ogólnym, jak i pod kątem indywidualnych potrzeb poszczególnych użytkowników.

Na diagramie DFD umieszczone zostały magazyny danych o następującym znaczeniu:

Stany magazynowe – przechowywanie ilości poszczególnych towarów z uwzględnieniem daty ważności oraz ceny.

Dane statystyczne – przechowywanie informacji o operacjach wykonywanych na produktach, np. sprzedaż określonej ilości określonego towaru, przeterminowanie określonego towaru.

Limity – przechowywanie ograniczeń nakładanych na towary zarówno ze strony dostawców, jak i obowiązujących umów.

Raporty – przechowywanie wymagań odnoszących się do zawartości oraz formatu raportów i statystyk.

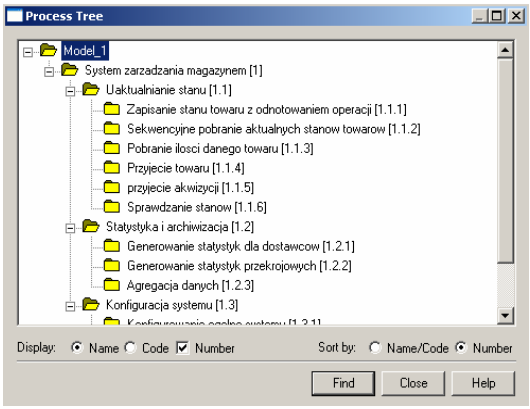
Akwizycja – dane dotyczące zapowiedzi wysłania towarów.

Agregacje – dane definiujące dokładność statystyk dla indywidualnego użytkownika.

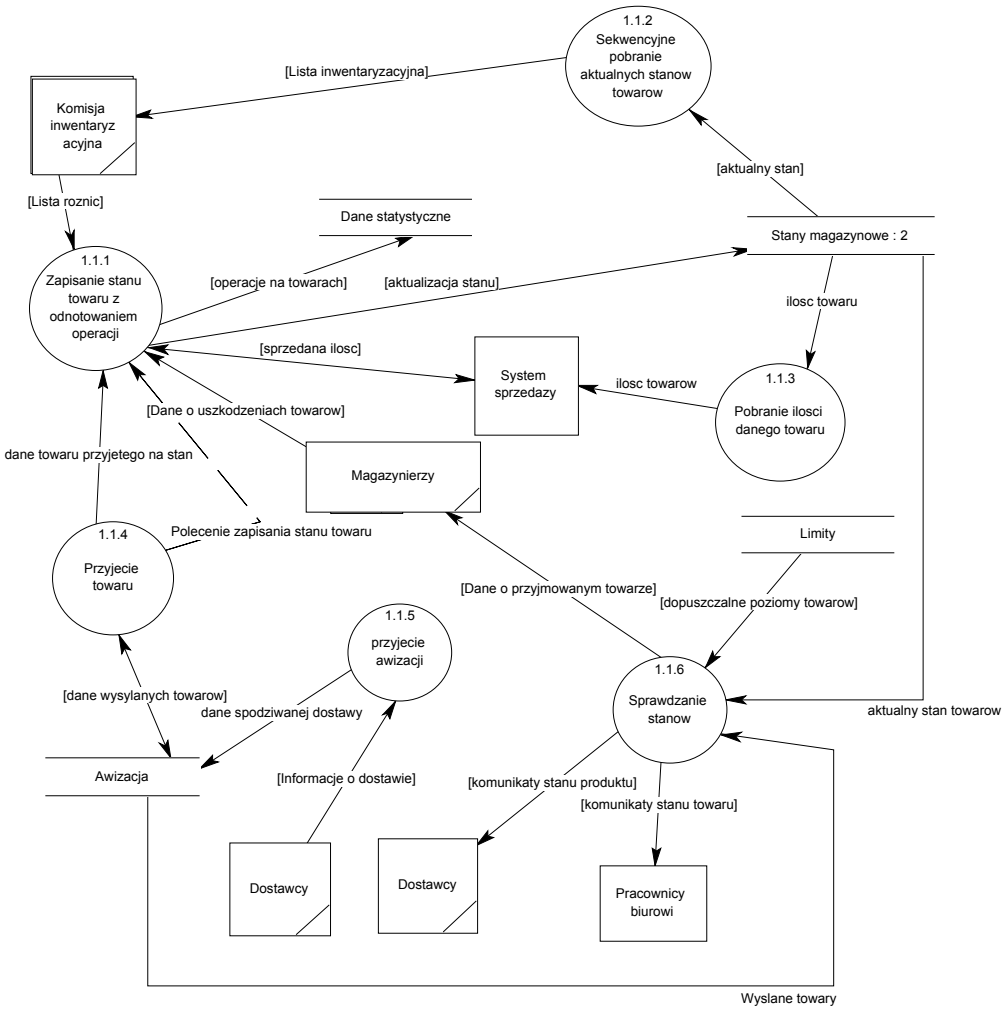
Cennik – wykaz cen poszczególnych produktów.

Związek między poszczególnymi obiektami DFD obrazują przepływy (strumienie) danych, których nazwy charakteryzują pakiety danych przenoszone przez przepływ. Warto zwrócić uwagę na mechanizm zapewnienia spójności modelu podczas dekompozycji: wszystkie przepływy danych umieszczane na diagramach wyższych poziomów są uwzględniane na diagramach poziomów niższych.

Procesy przedstawione na rys. 6.15 nie są procesami elementarnymi (graficznie obrazuje to znak + w symbolu reprezentującym proces), wykonany został więc kolejny i zarazem ostatni etap dekompozycji. W programie ProcessAnalyst ogólny obraz poziomów dekompozycji uwidacznia drzewo procesów, dostępne poprzez opcję *Dictionary* → *Subprocesses* → *Process Tree* (rys. 6.16).



Rys. 6.16. Drzewo procesów dla Systemu Zarządzania Magazynem



Rys. 6.17. Diagram DFD trzeciego poziomu dla Systemu Zarządzania Magazynem (dekompozycja procesu Uaktualnianie stanu)

Uwagi:

Diagram na rys. 6.17 przedstawia zdekomponowany proces 1.1. – **Uaktualnianie stanu**. Funkcje realizowane przez procesy elementarne są następujące:

1.1.1. **Zapisanie stanu towaru z jednoczesnym odnotowaniem operacji** – proces uaktualnia ilość towaru w magazynie o nazwie **Stany magazynowe** i zapisuje kody operacji w magazynie **Dane statystyczne**. Operacja zmiany stanu może zostać zainicjowana przez:

– magazyniera (przepakowanie towaru do większych opakowań, przeterminowanie towaru, zniszczenie, kradzież itp.),

– system sprzedaży (w przypadku sprzedania towaru),

– komisję inwentaryzacyjną (w przypadku stwierdzenia różnic inwentarzowych),
– proces przyjęcia towaru.

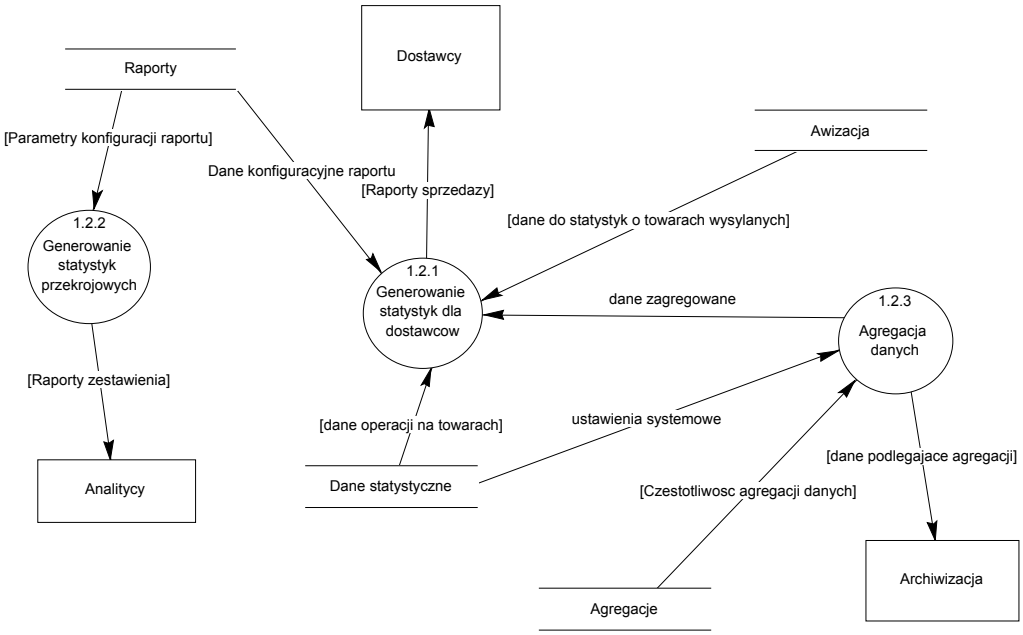
1.1.2. **Sekwencyjne pobranie aktualnych stanów towarów** – proces wykonujący listy inwentaryzacyjne (spis towarów, które powinny znajdować się w magazynie).

1.1.3. **Pobranie ilości danego towaru** – proces dostarcza danych dla systemu sprzedaży.

1.1.4. **Przyjęcie towaru** – realizuje wprowadzenie towaru do magazynu i zawiadomienie dostawcy o dotarciu towaru. Proces wysyła również formularz z danymi o dostawie do magazyniera. Proces przekazuje sterowanie do procesu 1.1.1, który dokonuje zapisu w magazynie **Stany magazynowe**.

1.1.5. **Przyjęcie awizacji** – proces obsługujący interfejs systemu do dostawców, umożliwiający dostawcom awizowanie wysyłki towaru (rodzaj, ilość, data dotarcia na miejsce przeznaczenia) oraz blokujący wysyłanie komunikatów o wyczerpaniu zapasów.

1.1.6. **Sprawdzenie stanów** – proces aktywowany czasowo lub na żądanie upoważnionego użytkownika. Realizuje zawiadamianie o przekroczeniu któregoś z poziomów ostrzegawczych (poziomu minimalnego bądź ostrzegawczego), dokonując sprawdzenia, czy dany towar nie został awizowany. Komunikuje się również z magazynierami w przypadku wykrycia towarów przeterminowanych.



Rys. 6.18. Diagram trzeciego poziomu dla Systemu Zarządzania Magazynem (dekompozycja procesu Statystyka i archiwizacja)

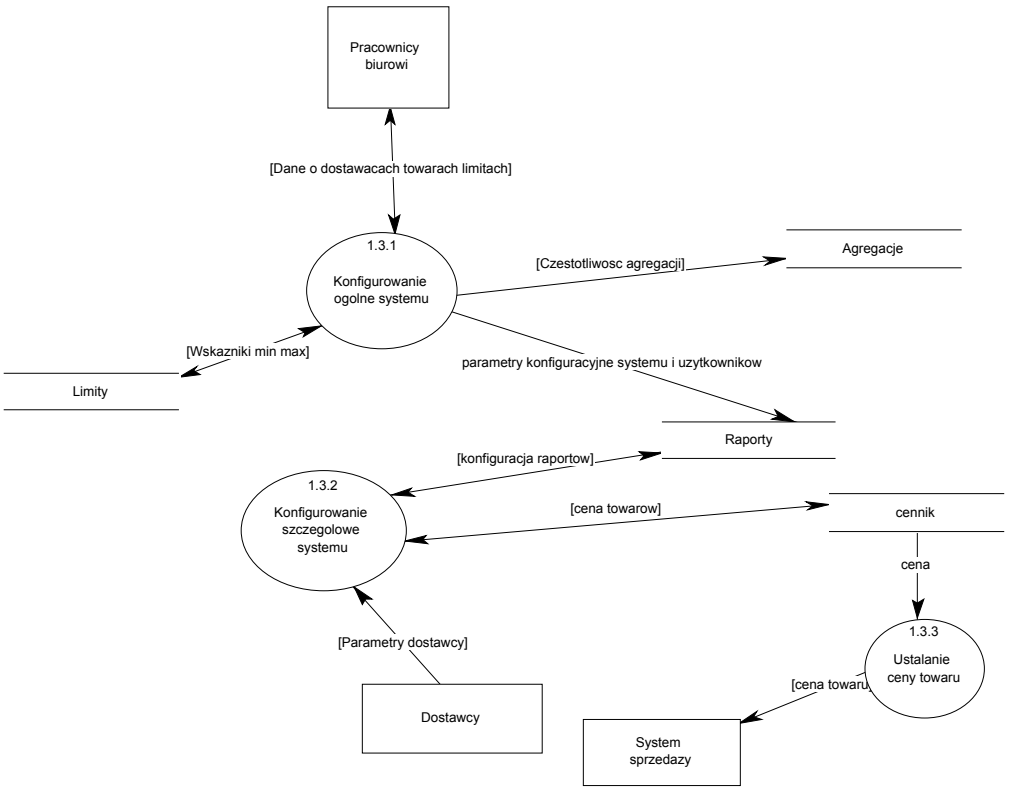
Uwagi:

Proces 1.2. **Statystyka i archiwizacja** zdekomponowany został na następujące procesy elementarne:

1.2.1. **Generowanie statystyk dla dostawców** – proces realizujący generowanie statystyk sprzedaży produktów konkretnego dostawcy. Dane dotyczące formatów raportów, rodzajów i częstotliwości ich generowania przechowywane są w magazynie danych Raporty. Z punktu widzenia użytkowników systemu jest to funkcja o dużym znaczeniu, gdyż w przypadku złych rozwiązań systemowych (złe profile zestawień, niewłaściwe horyzonty czasowe itp.) będą oni zmuszeni do stosowania własnych systemów informatycznych.

1.2.2. **Generowanie statystyk przekrojowych** – proces generujący zestawienia (np. obroty producentów) dla działu analizy firmy.

1.2.3. **Agregacja danych** – proces dokonujący agregowania danych dotyczących operacji na towarach według kryteriów dostawcy. Dane zbędne składowane są w zewnętrznym systemie archiwizacji.



Rys. 6.19. Diagram trzeciego poziomu dla Systemu Zarządzania Magazynem (dekompozycja procesu Konfiguracja systemu)

Uwagi:

Proces 1.3. **Konfiguracja systemu** zdekomponowany został na następujące procesy elementarne:

1.3.1. **Konfigurowanie ogólne systemu** – obsługuje interfejs umożliwiający administratorowi konfigurowanie całego systemu. Proces umożliwia wprowadzanie danych dostawców, z którymi podpisano umowy, danych dotyczących działów magazynu, danych towarów, które znajdują się w obrocie itp.

1.3.2. **Konfigurowanie szczegółowe systemu** – realizuje funkcje pozwalające personifikować poszczególne domyślne ustawienia systemu użytkownikowi, a konkretnie dostawcy. Każdy z dostawców potrzebuje zindywidualizowanych danych, prezentowanych w różnych formach i z różną częstotliwością, system zabezpiecza te potrzeby, umożliwiając indywidualne konfigurowanie takich ustawień w odniesieniu do własnych danych użytkownika, nie objętych warunkami umowy z firmą (np. stan minimalny i maksymalny towaru). Dostawcy mogą więc ustalać poziomy ostrzegawcze, rodzaje i częstotliwość raportów i statystyk czy sposób dostarczania dokumentów.

1.3.3. **Ustalanie ceny towaru** – proces dostarcza ceny poszczególnych towarów dla systemu sprzedaży. Proces ten jest o tyle ciekawy, że w systemie mogą występować różne ceny tego samego produktu, ze względu na możliwości obniżek przy większych zakupach, różnych cen dla różnych partii towaru, czy obniżek stosowanych przez producentów w przypadku zbliżającego się terminu ważności produktu. W tym ostatnim przypadku, ze względu na to, że system nie ma możliwości ustalania daty ważności sprzedawanego towaru, domyślnym trybem postępowania jest przesłanie niższej ceny do systemu sprzedaży aż do wyczerpania zapasów. Tryb taki nie gwarantuje jednak, że faktycznie sprzedany zostanie towar o żądanej dacie ważności. Niezbędne jest zatem wdrożenie procedury (poza systemem informatycznym) wykładania towarów do sprzedaży przez magazynierów.

Pakiet ProcessAnalyst wspomagający budowę diagramów DFD jest wyposażony w mechanizmy kontroli modelu. Oprócz mechanizmu kontroli spójności wbudowanego w narzędzie dekompozycji (zgodność przepływów danych na wszystkich poziomach diagramu DFD), program podczas tworzenia narzędziami graficznymi diagramu DFD w sposób automatyczny buduje macierz krzyżową, tzw. macierz CRUD, odwzorowującą związki pomiędzy procesami a magazynami danych lub pomiędzy danymi elementarnymi a procesami. Nazwa macierzy pochodzi od pierwszych liter operacji wykonywanych przez procesy na magazynach lub elementach danych:

C – Create, w przypadku zapisu do magazynu,

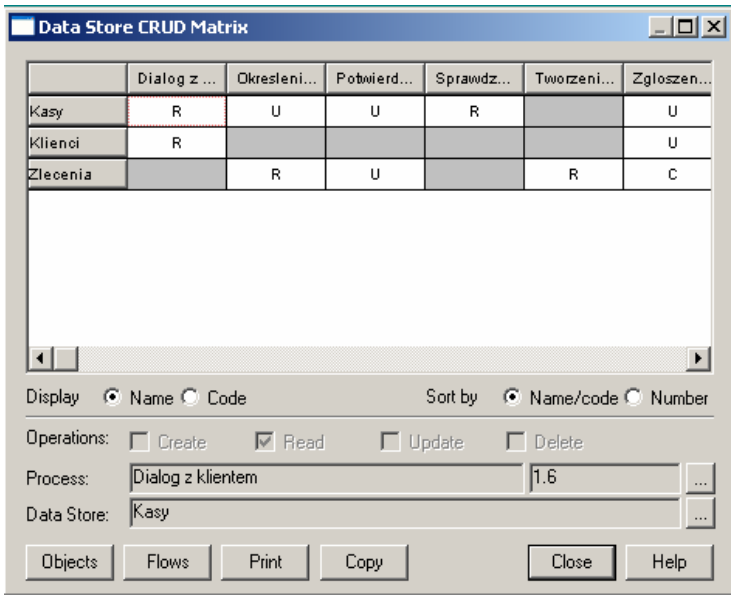
R – Read, w przypadku odczytu z magazynu,

U – Update, w przypadku modyfikacji zawartości magazynu,

D – Delete, w przypadku usuwania danych z magazynu.

Analizując zawartość macierzy CRUD, której nagłówkami kolumn są nazwy procesów, a nagłówkami wierszy nazwy magazynów lub elementów danych, natomiast na przecięciu kolumn i wierszy występują odpowiednie litery oznaczające wykony-

waną operację, można przeprowadzić dodatkową kontrolę modelu pod kątem wykorzystania magazynów danych przez procesy. W praktyce magazyny danych najczęściej oznaczają tablice bazy danych, do których odbywa się zapis, aktualizacja, kasowanie i odczyt danych. Są to podstawowe operacje, które powinny być dostępne dla użytkowników, jeżeli nie w sposób bezpośredni i nieograniczony, to poprzez nałożenie odpowiednich warunków i ograniczeń. Zawartość macierzy CRUD wyświetlana jest poprzez opcję *Dictionary → CRUD Matrix Data Store* lub *Dictionary → CRUD Matrix Data Item*.



Rys. 6.20. Fragment macierzy CRUD dla systemu SKKF

6.4.1. Słownik danych

Oprócz zaprezentowanych diagramów DFD w skład modelu zachowania (modelu behawioralnego) wchodzi słownik danych, stanowiący uporządkowany wykaz wszystkich elementów danych mających związek z systemem [19, 22]. Na podstawie słownika danych opisywane są:

- przepływy i magazyny uwidocznione na diagramach DFD;
- złożone pakiety danych transmitowane wzdłuż przepływów;
- pakiety danych w magazynach;
- właściwe wartości i jednostki elementarnych porcji danych dla przepływów i magazynów danych, często z określeniem precyzji pomiaru;
- szczegóły związków między danymi w poszczególnych magazynach.

Pełna definicja elementu danych powinna określać:

- znaczenie elementu danych w kontekście aplikacji użytkownika, na ogół w formie komentarza;
- budowę elementu danych;
- zbiór wartości, jakie może przyjmować element danych, w przypadku danych elementarnych, czyli niepodlegających dekompozycji.

Dla precyzji zapisu w słowniku danych niezbędne jest przyjęcie odpowiedniej notacji, czyli sposobu zapisu. Oczywiście istnieje wiele schematów notacyjnych, ale jednym z najprostszych i najpopularniejszych jest schemat składający się z następujących operatorów [22]:

=	jest złożony z
•	i
()	element opcjonalny
{}	element powtarzalny (iteracja)
[ ]	wybór z możliwości alternatywnych
|	rozdzielenie możliwości alternatywnych
**	początek i zakończenie komentarza (minispecyfikacji elementu danych)
@	element identyfikujący

Należy pamiętać, że słownik danych powstaje na etapie analizy wymagań dotyczących systemu, nie zaś projektowania fizycznej implementacji, uwzględnia więc istnienie elementów danych, a nie ich postać fizyczną (np. pensja to siedem cyfr dziesiętnych z dwoma miejscami po przecinku).

Podczas uzgadniania z przyszłym użytkownikiem definicji słownika mogą pojawić się sytuacje, kiedy odnośnie tej samej pozycji danych można użyć kilka różnych nazw. Zadaniem analityka jest wybór nazwy podstawowej (preferowanej), pozostałe zaś będą występować jako synonimy (aliasy), z odsyłaczami do definicji nazwy podstawowej, np. *wykładowca = synonim dla nauczyciela akademickiego*. Umieszczanie w słowniku danych zbyt wielu synonimów nie jest dobrym rozwiązaniem, lepiej doprowadzić do konsensusu i przyjęcia jednej, wspólnej nazwy.

Następna uwaga dotyczy zapisu iteracji, czyli powtórzeń składnika elementu danych. W rzeczywistości mogą pojawić się sytuacje, w których będzie konieczne ograniczenie iteracji, zarówno dolnego jak i górnego (lub obu), wynikające z reguł i przepisów obowiązujących w danym środowisku lub po prostu z żądań użytkownika. Ograniczenia zapisuje się w słowniku w sposób następujący: 1{*element danych*}10, co oznacza minimum jeden element danych, maksimum dziesięć.

Po zastosowaniu przyjętych konwencji, słowniki danych dla systemów SSFK oraz Systemu Zarządzania Magazynem przedstawiają się następująco:

Kontynuacja *przykładu 6.2* – słownik danych dla systemu SSFK

Nr unikatowy = * Numer nadany przez Urząd Skarbowy*
16{cyfra}16
Typ = **Model i typ kasy**
1{litera| cyfra | znak_1}20
Data instalacji = **Data zainstalowania kasy u klienta**
rok + miesiąc + dzień
Data ostatniego przeglądu = rok + miesiąc + dzień
Adres instalacji = **Adres, pod którym zainstalowana jest kasa**
ulica + numer domu + (numer mieszkania) + kod pocztowy + nazwa miejscowości
Data realizacji = **Data realizacji zlecenia**
rok + miesiąc + dzień
Godzina realizacji = **Godzina realizacji zlecenia**
godzina + minuta
Id zlecenia = **Identyfikator zlecenia: inicjały serwisanta, data, numer kolejny w ciągu dnia**
2{litera}3 + 8{cyfra}8 + znak_2 + {cyfra}
Dzień = **Numer dnia w miesiącu. Zakres 01-31**
2{cyfra}2
Miesiąc = **Numer miesiąca w roku. Zakres 01-12**
2{cyfra}12
Rok = **Rok. Wartość minimalna 2002**
4{cyfra}4
Godzina = **Numer godziny w dobie. Zakres 00-23**
2{cyfra}2
Minuta = **Numer minuty w godzinie. Zakres 00-59**
2{cyfra}2
Serwisant = **Imię i nazwisko serwisanta wykonującego zlecenie**
3{litera}15 + znak_3 + 3{litera}30
Status = **Status wykonania zlecenia: wykonane lub nie**
wartość [T|N]
Id klienta = **Identyfikator klienta, skrót nazwy**
1{litera}10
Adres = **Dane adresowe klienta**
ulica + numer domu + (numer mieszkania) + kod pocztowy + nazwa miejscowości
Ulica = 1{[litera | znak_1]}32
Numer domu = 1{cyfra}4
Numer mieszkania = 1{cyfra}4
Kod pocztowy = 2{cyfra}2 + znak_1 + 3{cyfra}3
Nazwa miejscowości = 1{[litera | znak_1]}32

NIP = **Numer Identyfikacji Podatkowej Użytkownika**
3{cyfra}3 + znak_2 + 2{cyfra}2 + znak_2 + 2{cyfra}2 + znak_2 + 2{cyfra}2
Nr konta = **Numer konta bankowego klienta**
{cyfra | znak_1}
Znak_1 = [_ | - | .]
Znak_2 = -
Litera = [a-z | A-Z]
Cyfra = 0-9

Kontynuacja *przykładu 6.4* – fragment słownika danych dla Systemu Zarządzania Magazynem

Cyfra = [0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9]
Litera = [a...z | A...Z]
Znak = [! | @ | # | $ | % | & | * | (|) | _ | + | - | = | { | } | [|] | „ | ; | ‘ | < | > | , | . | ? | / | ^]
Dostawca = @Id_dostawcy + Nazwa + Adres + (e-mail) + Telefon + Osoba_kontaktowa
@Id_dostawcy = 1{cyfra}8
Nazwa = **nazwa firmy**
{litera}
Adres = **Adres firmy**
Ulica + Numer budynku + (Numer lokalu) + Kod + Miejscowość
Ulica = {litera}
Numer budynku = {cyfra} + (litera)
Numer lokalu = {cyfra} + (litera)
Kod = **Kod pocztowy**
cyfra + cyfra + znak - + cyfra + cyfra + cyfra
Miejscowość = {litera}
e-mail = {litera + (znak .| znak _)} + znak @ + {litera + (znak . | znak _)}
Telefon = (znak (+ {cyfra} + znak)) + {cyfra}
Osoba_kontaktowa = **Osoba z firmy odpowiedzialna za kontakty z obsługą systemu**
Imię + Nazwisko
Imię = {litera}
Nazwisko = {litera}
Cena towaru = Id_towaru + Cena netto + VAT + Ilość_dla_upustu + Data ważności
Id towaru = **Identyfikator towaru, którego dotyczy kalkulacja ceny**
{cyfra}
Cena netto = {cyfra} + znak , + cyfra + cyfra
VAT = **Wielkość podatku VAT naliczanego dla danego towaru**
{cyfra}
Ilość_dla_upustu = **Minimalna liczba sztuk towaru gwarantująca upust cenowy**
{cyfra}

Upust = **Wartość określająca, o ile zmniejszy się cena w przypadku upustu**
{cyfra}
Data ważności = **Data końcowa obowiązywania kalkulacji cenowej dla danego towaru w formacie dd-mm-rrrr**
cyfra + cyfra + znak - + cyfra + cyfra + znak - + cyfra + cyfra + cyfra + cyfra
Raporty = Id_towaru + Id_dostawcy + Id_producenta + Okres + Status_raportu + Kod_formy + Kod_dostarczania
Id_towaru = **Identyfikator towaru, którego dotyczy raport**
{cyfra}
Id_dostawcy = **Identyfikator dostawcy, dla którego tworzony jest raport**
1{cyfra}8
Id_producenta = **Identyfikator producenta, którego dotyczy raport**
{cyfra}
Okres = **Częstotliwość generowania raportu**
{cyfra}
Status_raportu = **Rozróżnienie raportów wewnętrznych dla analityków i zewnętrznych dla dostawców**
{cyfra}
Kod_formy = **Kod identyfikujący postać raportu**
{cyfra}
Kod_dostarczania = **Kod określający sposób przesyłania raportu (poczta, e-mail, www, fax, itp.)**
{cyfra}

Ze zrozumiałych względów, w odniesieniu do powyższego przykładu nie został przytoczony kompletny słownik danych, definicje mają jedynie ilustrować sposób budowania słownika.

Warto zauważyć, że w obu przykładach zastosowano tę samą notację, styl definiowania danych natomiast nieco się różni. Zależy on od indywidualnych preferencji twórcy (analityka systemu) i ma oczywiście wpływ na ostateczny wygląd słownika. Z punktu widzenia projektantów i programistów jest to sprawa wtórna, istotna jest natomiast czytelność, spójność i kompletność zamieszczonych definicji.

6.4.2. Specyfikacja procesów

Specyfikacja procesów stanowi algorytmiczną definicję procesu, czyli opisuje, co dzieje się wewnątrz procesu w celu przekształcenia danych wejściowych w dane wyjściowe. Specyfikacje tworzone są dla procesów elementarnych, a więc dla procesów na najniższym poziomie diagramu przepływu danych. Istnieje wiele sposobów specyfikowania procesów, niezależnie jednak od wybranej konwencji specyfikacja musi zawierać:

– numer i nazwę procesu,
– dane wejściowe,
– dane wyjściowe,
– opis algorytmu.

Do opisu algorytmu najczęściej wykorzystuje się *strukturalny język polski* (czyli podzbiór języka naturalnego), będący w zasadzie zbiorem konstrukcji spotykanych w językach programowania (tzw. pseudokod). Często również specyfikacje procesów tworzone są poprzez określenie tzw. warunków początkowych i końcowych, metodą konstruowania tablic decyzyjnych, czy nawet wykresów lub schematów blokowych.

Wybór sposobu specyfikacji zależy od preferencji analityka, ale należy pamiętać, w jakim celu tworzone są specyfikacje – zapoznać się z nimi i dokonać weryfikacji musi przyszły użytkownik systemu, specyfikacje muszą więc być dla niego zrozumiałe.

Kontynuacja *przykładu 6.2* – specyfikacja procesów dla systemu SSKF, z zastosowaniem pseudokodu wzorowanego na programowaniu strukturalnym

1.1. **Zgłoszenie kasy**

```
GET Id_klienta FROM Zgloszenie kasy
GET Id_klienta FROM STORE Klienci AS klient
IF (klient = NULL)
{
        GET Nazwa + Adres + Telefon + NIP + (Nr_konta) AS dane_klienta
        SAVE dane_klienta TO STORE Klienci
}
GET Nr_unikatowy + typ + Data_instalacji + Data_ostatniego_przeglądu + Adres_instalacji AS dane_kasy
SAVE dane_kasy TO STORE Kasy
```

1.2. **Określenie terminu realizacji**

```
DO
{
        SEND pytanie o termin realizacji to TERMINATOR Klient
        GET odpowiedź na pytanie o termin AS odpowiedz
        IF (odpowiedz = odmowa)
        {
                SEND odmowa TO STORE Kasy
                EXIT
        }
        ELSE
        {
```

```
                SET odpowiedz AS proponowana data
                FIND proponowana data IN STORE Zlecenia AS termin
                IF (termin! = NULL)
                {
SEND przekazanie serwisowi zlecenia TO TERMINATOR Serwisanci
                }
        }
WHILE (!termin)
```

1.3. **Potwierdzenie wykonania**

```
GET potwierdzenie wykonania FROM TERMINATOR Serwisanci AS potwierdzenie
IF (potwierdzenie)
{
        SEND wysłanie rachunku TO TERMINATOR Klient
        SET data_ostatniego_przegladu AS aktualna_data
        SAVE data_ostatniego_przeglądu TO STORE Kasy
}
```

1.4. **Sprawdzenie terminów przeglądów**

```
GET Nr_unikatowy + Data_ostatniego_przeglądu + Adres_instalacji FROM STORE
Kasy
IF (Data_ostatniego_przeglądu > (Data_ustawowa – 7 dni)
{
        SEND sygnał „kasa wymaga przeglądu” TO PROCESS 1.2 Określenie termi-
nu realizacji
}
```

1.5. **Tworzenie raportów o pracy serwisu**

```
IF (data systemowa > 27 danego miesiąca)
{
        FOR Serwisanci {
                DO
                {
                FIND Serwisant, data_realizacji, status) FROM STORE Zlecenia
                IF (data_realizacji = bieżący miesiąc AND status = T)
                } WHILE TRUE
}
SEND gotowy raport TO TERMINATOR Kierownictwo
```

Kontynuacja *przykładu 6.4* – specyfikacja niektórych procesów dla Systemu Zarządzania Magazynem wzorowana na składni języka C/C++ (rys 6.18, procesy związane ze statystyką i archiwizacją). Do zbioru słów kluczowych włączono *foreach* (*obiekt* in *obiekty*), powodujące za każdym wykonaniem pętli podstawienie pod zmienną obiekt kolejno każdego obiektu ze zbioru obiektów. Magazyny danych są zaznaczone przez podkreślenie.

1.2.1. **Generowanie statystyk dla dostawców**

Opis: Proces generuje zestawienia statystyczne (raporty) dla jednego dostawcy.
Wejście: Id_dostawcy
Wyjście: raport
Algorytm:

```
FOREACH (raport IN Raporty.Zapytanie (kod_raportu = Dostawcy, Id_dostawcy))
        GENERUJ Statystyki (raport)
```

1.2.2. **Generowanie statystyk przekrojowych**

Opis: Proces generuje statystyki dla konkretnych konfiguracji raportu
Wejście: parametry konfiguracji
Wyjście: raport
Algorytm:

```
        Id_producenta = raport.Id_producenta;
        Id_towaru = raport.Id_towaru;
        Dane = Dane Statystyczne.Zapytanie (Id_producenta, Id_towaru);
        SWITCH (raport.Postać_raportu)
        { CASE SPRZEDAŻ:
                operacja = Operacje.Zapytanie (SPRZEDAŻ);
                Id_operacji = operacja.Id_poeracji;
                FOREACH (dana IN dane)
                {
                        IF (dana.Data >= aktualnaData – raport.okres &&
                            Dana.Id_operacji == Id_operacji)
                }
                        PRINT (dana.Data, dana.Godzina, dana.ilość);
                }
        }
        BREAK;
        CASE USZKODZENIA:
                {
                        IF (stan.Data_ważności < AktualnaData)
                            Przeterminowane += stan;
                }
        }
```

RETURN *przeterminowane*

Pokazane przykłady obrazują sposób specyfikacji procesów za pomocą strukturalnego języka polskiego, zawierają opis procedur prowadzących od danych do wyników. Sposób ten jest zalecany w przypadkach, gdy związki między danymi wejściowymi i wynikami są złożone. Należy jednak zauważyć, że specyfikacje muszą być czytelne dla użytkownika, a więc nie mogą być zbyt skomplikowane (zbyt wiele poziomów zagnieżdżenia, nieczytelny układ edytorski, zbyt obszerne). Jeśli specyfikacje opracowane w takiej konwencji stają się bardzo złożone, należy zastanowić się nad inną metodyką specyfikacji procesów lub wręcz przeanalizować powtórnie diagram DFD pod kątem dalszej dekompozycji procesów.

W przypadku, gdy funkcja realizowana przez proces może być wykonywana na kilka sposobów, specyfikacje procesów powstają poprzez określenie tzw. *warunków początkowych i końcowych*, bez precyzowania poszczególnych algorytmów. Warunki początkowe określają wszystko, co musi być spełnione przed rozpoczęciem procesu (dane wejściowe, związki między różnymi magazynami lub wewnątrz określonego magazynu), warunki końcowe określają stan po zakończeniu działania procesu (opis wyników, zmiany dokonane w magazynach, związki między wynikami a wartościami w magazynach). Należy pamiętać, że stosując taką metodę specyfikacji należy sprecyzować, oprócz warunków początkowych i końcowych dla sytuacji normalnych, warunki początkowe i końcowe dla sytuacji błędnych.

Przykładowo, proces, którego funkcją jest drukowanie faktur można opisać poprzez specyfikację:

Proces 1. Wydruk faktury
WARUNEK POCZĄTKOWY 1
Istnieje *faktura* w magazynie *Faktury*
Której *Nr_faktury* pasuje
Do *Nr_faktury* w magazynie *Usługi na fakturze*
WARUNEK KOŃCOWY 1
Drukowanie zestawienia dla *Nr_faktury* zawierajacego:
Nr_faktury + PESEL_klienta + Nazwisko + Imię + Kod + Miejscowość + Ulica + Data_wystawienia + Opis_usługi + Cena
WARUNEK POCZĄTKOWY 2
Istnieje *faktura* z *Nr_faktury* nie pasującym
Do *Nr_faktury* w magazynie *FAKTURY*
WARUNEK KOŃCOWY 2
Generowanie komunikatu: „Nie istnieje faktura o numerze *Nr_faktury*”

Jeszcze innym sposobem umożliwiającym specyfikację procesów jest budowanie *tablic decyzyjnych*. Metoda ta jest najczęściej przydatna w sytuacjach, gdy decyzje, od których zależy wykonanie odpowiednich akcji przez proces, oparte są na kilku zmien-

nych mogących przyjmować różne wartości. Tablica decyzyjna zbudowana jest z kolumny zawierającej wszystkie zmienne wejściowe oraz wszystkie akcje, które może podjąć proces. W następnych kolumnach tabeli umieszczane są wszystkie możliwe kombinacje wartości zmiennych wraz z towarzyszącymi im akcjami – są to tzw. reguły. Dla N zmiennych o wartościach binarnych otrzymuje się więc 2^N reguł. Tablica decyzyjna opisująca proces udzielania rabatu klientom w zależności od ilości zakupionego towaru, wartości transakcji oraz statusu klienta (klient stały lub okazjonalny) może mieć na przykład postać jak na rys. 6.21.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Status klienta	S	S	O	O	S	S	O	O
Ilość > 1000 szt.	T	T	T	T	N	N	N	N
Wartość > 1500	T	N	T	N	T	N	T	N
Rabat 1						*X*	*X*	
Rabat 2		*X*	*X*					
Rabat 3	*X*				*X*			
Bez rabatu				*X*				*X*

Rys. 6.21. Tablica decyzyjna jako specyfikacja procesu

W praktyce warunki (zmienne wejściowe) nie zawsze są typu binarnego, co wpływa na rozmiar tablicy decyzyjnej (aby określić liczbę reguł, należy pomnożyć przez siebie liczby możliwych wartości kolejnych zmiennych; przykładowo, jeśli zmienna 1 przyjmuje dwie wartości, zmienna 2 trzy wartości, zmienna 3 cztery wartości, to w tablicy decyzyjnej muszą pojawić się 24 reguły).

Specyfikacje procesów, niezależnie od wybranej metody ich opracowania, są jedną z bardziej pracochłonnych czynności w cyklu budowania modelu systemu. Będąc składową modelu, jednocześnie stanowią test poprawności dla diagramów przepływu danych. Często okazuje się (w trakcie opracowywania specyfikacji), że diagram DFD powinien zostać rozbudowany o dodatkowe przepływy danych, lub że niektóre procesy powinny zostać zdekomponowane, gdyż realizują kilka rozłącznych funkcji. W trakcie pisania specyfikacji najczęściej ujawniają się braki procesów modyfikujących lub usuwających dane z magazynów.

Na tym etapie można przyjąć, że model systemu (inaczej mówiąc, model reguł przetwarzania) w odniesieniu do systemów bazodanowych jest kompletny i udokumentowany. Należy pamiętać, że w procesie kreowania modelu za pomocą narzędzi CASE większa część dokumentacji przechowywana jest w repozytorium pakietów – w przypadku programu ProcessAnalyst w plikach z rozszerzeniem *.PAM, co pozwala na wykorzystanie zdefiniowanych obiektów i struktur w następnych etapach projektowania. Polecenie zachowania modelu wykonywane jest poprzez wybór menu *File* → *Save* lub *File* → *Save As*.

Następnym krokiem jest opracowanie modelu danych, czyli układu danych (struktury i powiązania), przechowywanych w systemie.

7

Modelowanie danych, projektowanie bazy danych

Począwszy od lat siedemdziesiątych fundamentalną składową systemów informatycznych stały się systemy bazy danych, które zastąpiły przetwarzanie danych zorganizowane w systemach plików. Etapy cyklu życia bazy danych są więc ściśle związane z cyklem życia całego systemu informatycznego. Po zakończeniu etapu gromadzenia wymagań użytkowników odnośnie do systemu oraz fazy analizy następuje przejście do etapu projektowania.

Integrację cyklu życia bazy danych z cyklem życia całego systemu od momentu przejścia do fazy projektowania ilustruje rys. 7.1. Na rysunku uwidoczniono trzy fazy projektowania bazy danych:

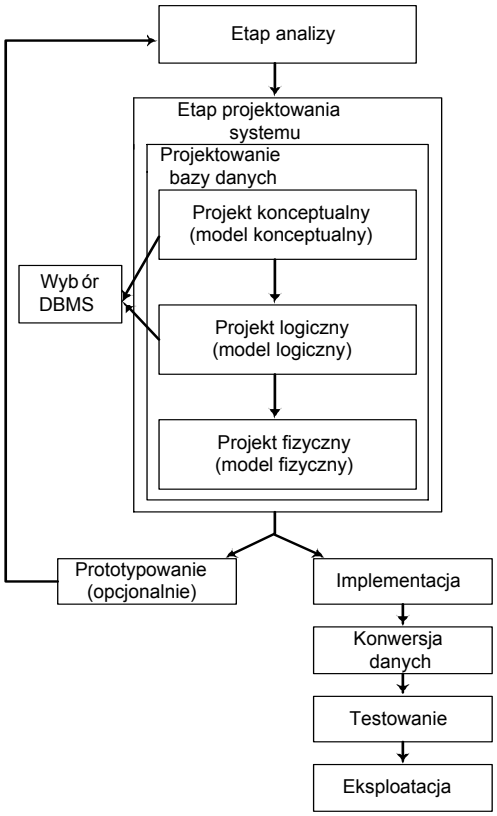
- budowanie modelu konceptualnego,
- budowanie modelu logicznego,
- budowanie modelu fizycznego.

Faza pierwsza to budowanie modelu konceptualnego, które polega na konstruowaniu modelu informacji (danych) występujących w obszarze przedsięwzięcia, jakim jest projekt systemu. Powstaje on na podstawie udokumentowanych wymagań użytkownika, czyli w wyniku etapu analizy. Model konceptualny jest całkowicie oderwany zarówno od uwarunkowań implementacyjnych, takich jak typ bazy danych (hierarchiczna, sieciowa, relacyjna, czy relacyjno-obiektowa), jak i konkretnych platform sprzętowych i programowych.

W fazie drugiej powstaje model logiczny danych uwzględniający specyfikę modelu danych (np. model relacyjny), ale również bez uwzględnienia jakichkolwiek uwarunkowań konkretnej implementacji fizycznej.

W fazie trzeciej powstaje model fizyczny, czyli opis implementacji modelu logicznego w konkretnym (wybranym) środowisku bazodanowym z uwzględnieniem organizacji plików, indeksów, więzów integralności oraz mechanizmów bezpieczeństwa.

W odniesieniu do zagadnienia projektowania baz danych można wyraźnie wyróżnić dwie strategie postępowania:



Rys. 7.1. Etapy projektowania bazy danych w cyklu życia systemu

1. Wstępująca (*bottom-up*) – stosowana w projektowaniu małych, niezbyt skomplikowanych baz danych, a polegająca na wyodrębnieniu danych elementarnych z całego zbioru danych, analizie związków funkcjonalnych między tymi danymi i następnie na tej podstawie zaprojektowaniu tabel oraz powiązań między nimi. Takie podejście, znane również jako proces normalizacji, zostanie szerzej omówione w rozdziale 8. Zasady i algorytmy normalizacji, oprócz umożliwienia wstępującego projektowania małych baz danych, pełnią rolę narzędzia weryfikacji projektów dużych baz danych.

2. Zstępująca (*top-down*) – najbardziej odpowiednia dla systemów dużych i skomplikowanych. W tym podejściu nie rozpoczyna się od analizy danych szczegółowych, lecz od wydzielenia abstrakcyjnych jednostek wysokiego poziomu, aby w kolejnych podejściach wyodrębnić jednostki niższych poziomów, określić powiązania między nimi, czyli coraz bardziej uściślać model opisujący dany wycinek rzeczywistości, dla którego powstaje baza danych. Metoda ta bazuje na koncepcji modelowania związków encji (*Entity Relationship Model*), opierającej się na założeniu, że cały obszar modelowania da się przedstawić za pomocą obiektów (*encji, entity*), opisanych poprzez ich

własności (*atrybuty, attributs*) oraz wzajemnego oddziaływania na siebie obiektów (*powiązania, relationships*), którego zasady także są zapisywane w modelu.

7.1. Diagramy E-R

Graficzną reprezentację modelu logicznego danych relacyjnych stanowi tzw. diagram E-R (*Entity Relationship Diagram, Diagram związków encji*), podstawa współczesnego modelowania danych. Za twórcę tej metodyki uważany jest Peter Chen [2, 3, 12, 19, 22]. Notacja zaproponowana przez Chena w zastosowaniu do złożonych projektów wymaga sporządzenia bardzo dużej liczby pojedynczych diagramów, ponieważ każdy z nich reprezentuje jeden związek pomiędzy encjami. Dlatego też obecnie najbardziej rozpowszechnioną notacją są tzw. szczegółowe diagramy E-R, przedstawiające wszystkie związki danej encji. Należy zauważyć, że przedstawione w poprzednim rozdziale narzędzie modelowania systemu, jakim jest diagram DFD, wprawdzie dobrze opisuje system, ale nie pokazuje zależności pomiędzy danymi, które często są bardzo złożone (magazyny danych są pojęciami czysto abstrakcyjnymi, bez wyspecyfikowanej struktury składnicy); zbudowanie modelu danych staje się więc koniecznością. W praktyce często okazuje się, że model danych sugeruje zmiany w modelu systemu.

W przypadku technologii CASE zaimplementowanej w pakiecie DataArchitect Sybase, model konceptualny (*Conceptual Data Model*, CDM) związany jest z relacyjnym modelem danych, ale nie z konkretnym systemem zarządzania bazą danych, tak więc po zbudowaniu diagramu związków encji można na jego podstawie generować modele relacyjne danych (*Physical Data Model*, PDM) związane z różnymi środowiskami bazodanowymi, a następnie dla każdego z modeli PDM – skrypt *SQL* umożliwiający fizyczną implementację modelu (schematu bazy danych) w wybranym środowisku.

Model związków encji reprezentowany przez diagram E-R jest to, inaczej mówiąc, logiczny model danych, w którym obiekty modelowanego środowiska są przedstawiane za pomocą jednoznacznie identyfikowalnych encji. Własności encji są opisywane za pomocą atrybutów. Pomiędzy encjami występują powiązania wynikające z reguł działania danego środowiska, a zatem przy ustalaniu powiązań między encjami niezbędne jest zapoznanie się ze specyfikacją procesów przetwarzających dane, opracowane na etapie analizy. Ze specyfikacji procesów powinny wynikać reguły powiązań (które obiekty są ze sobą związane i w jaki sposób) oraz zasady współdziałania między encjami (dwiema lub kilkoma). Składowymi modelu danych, czyli diagramu E-R są następujące elementy:

- Encja (klasa encji) (*entity*) — Logicznie wyodrębniony zestaw elementów danych – dane odnoszące się do osoby, miejsca zdarzenia, pro-

duktu, przechowywane w systemie. Grupa elementów danych powtarzająca się w przepływie danych lub w magazynie na ogół jest encją.

- Wystąpienie encji (*entity instance*) — Konkretna jednostka należąca do klasy encji. Każde wystąpienie encji ma określone wartości wszystkich atrybutów encji. Encja musi mieć więcej niż jedną instancję, gdyż w przypadku jednej instancji byłaby to stała systemowa. Każde wystąpienie encji musi być jednoznacznie identyfikowalne poprzez jeden lub kilka atrybutów. Przykładowo, wystąpieniem encji STUDENT jest konkretny student Jan Nowak identyfikowany numerem indeksu.
- Identyfikator encji (*entity identifier*) — Pojedynczy atrybut lub kombinacja atrybutów jednoznacznie identyfikująca poszczególne wystąpienia encji.
- Atrybut (*attribute*) — Element danych opisujący właściwości encji lub związku. Każda encja musi mieć co najmniej jeden atrybut.
- Dziedzina atrybutu (*domain*) — Typ danych lub zakres wartości dozwolony dla atrybutu.
- Nadklasa (*supertype*) — Klasa encji będąca generalizacją podzbiorów encji, przykładowo klasa encji STUDENT może być generalizacją podzbiorów STUDENT DZIENNY, STUDENT ZAOCZNY, STUDENT WIECZOROWY.
- Podklasa (*subtype*) — Klasa encji będąca podzbiorem nadklasy. Podklasy dziedziczą atrybuty i związki nadklasy, ale mogą mieć dodatkowo (i najczęściej mają) swoje własne atrybuty i związki.
- Związek (*relationship*) — Połączenie encji (dwóch lub kilku), określające pamiętane w systemie współdziałanie między encjami. Związki pomiędzy dwoma encjami noszą nazwę binarnych, związki pomiędzy większą liczbą encji są to związki *n*-stronne, w których stopień związku określa liczba encji biorących w nim udział (związki trójstronne itp.). Gdy powiązania zachodzą pomiędzy instancjami tej samej encji, mamy do czynienia ze związkiem rekurencyjnym. Związki ustanawiane są na podstawie zależności wyspecyfikowanych w modelu systemu: w specyfikacji procesów zapisywane są reguły tworzenia, aktualizowania i usuwania encji i związków. Niektóre ze związków mogą zawierać atrybuty opisujące dodatkowo sam związek, a nie związane z nim encje.
- Liczność — Wskazanie liczby instancji encji skojarzonych

(*cardinality*)	w związku. Ze względu na liczność wyróżnia się związki: jeden do jeden, jeden do wiele, wiele do wiele. Określenie liczności związku musi nastąpić dla obu końców związku.
• Opcjonalność (*modality*)	Określa, czy wystąpienie encji w danym związku jest wymagane, czy też opcjonalne. Liczbowo jest to parametr przyjmujący wartość 0 dla uczestnictwa opcjonalnego, 1 dla wymaganego. Określenie uczestnictwa również musi nastąpić dla obu końców związku.
• Normalizacja	Proces usuwania nadmiarowych, niespójnych elementów danych, którego celem jest uzyskanie modelu, w którym każda encja reprezentuje tylko jeden obiekt rzeczywisty. Proces normalizacji jest formalną techniką analizy zależności funkcjonalnych pomiędzy elementami danych oraz metodyką postępowania dla zachowania zgodności schematów tabel z wymogami postaci normalnych.

Powyższe zestawienie charakteryzuje pokrótce najważniejsze pojęcia związane z modelowaniem danych za pomocą diagramów E-R. W dalszej części rozdziału definicje pojęć zostaną rozszerzone, a ich zastosowanie zilustrowane przykładami. Najpierw jednak zaprezentowane zostanie narzędzie, za pomocą którego proces modelowania danych będzie przeprowadzany, czyli moduł DataArchitect wchodzący w skład zintegrowanego pakietu PowerDesigner. Przykłady przytaczane w dalszych częściach podręcznika zostały opracowane za pomocą wersji 6.1 programu; należy nadmienić, że wyższe wersje pakietu umożliwiają modelowanie danych w środowisku graficznym również w podejściu obiektowym z odwołaniem do standardów UML.

7.2. Zasady pracy z programem DataArchitect – model konceptualny

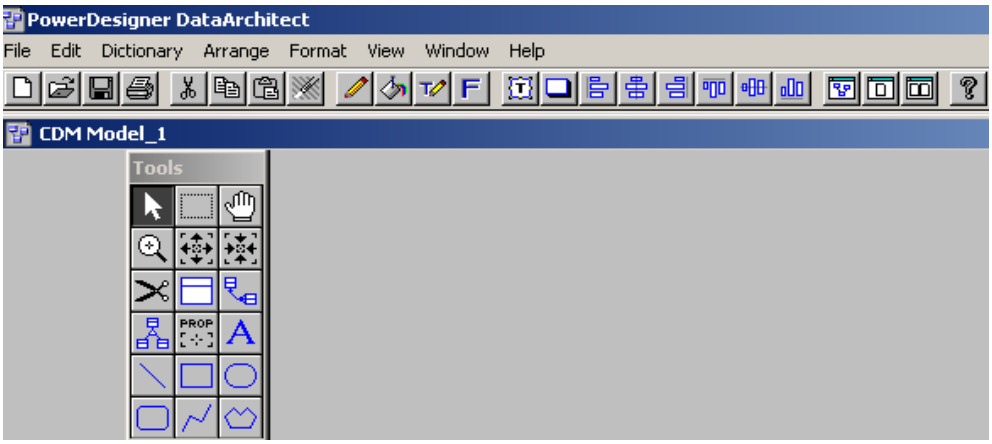
Praca z programem DataArchitect rozpoczyna się od budowania modelu konceptualnego (CDM), który reprezentuje ogólną strukturę danych w systemie informatycznym, czyli relacje między różnymi typami informacji z pominięciem aspektu implementacji fizycznej [23]. Pakiet wyposażony jest w procedury generacyjne, które umożliwiają transformację modelu konceptualnego w model relacyjny (PDM) związany z wybranym Systemem Zarządzania Bazą Danych (DBMS). W następnym etapie, po zakończeniu pracy nad projektem logicznym bazy danych (normalizacją tabel, utworzeniu perspektyw, założeniu indeksów) program umożliwia wygenerowanie

skryptów zakładających zaprojektowaną bazę danych w wybranym środowisku. Rozpoczynając pracę z programem, należy mieć na uwadze, że jest to, zgodnie z definicją, narzędzie wspomagające projektanta, a więc takie aspekty jak:

- informacje, które powinny być pamiętane w bazie danych,
- obiekty zewnętrzne, których dane mogą być potrzebne, a więc powinny być umieszczone w bazie danych,
- działanie organizacji, dla której projektowany jest system,

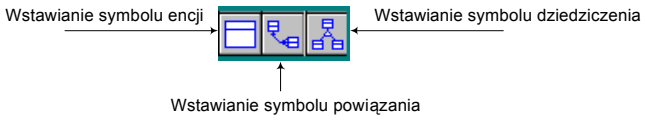
powinny być łatwe do wyabstrahowania z modelu systemu (diagram kontekstowy, diagram DFD, specyfikacja procesów, słownik danych), ale jest to zadanie projektanta, a nie pakietu. Pakiet, stanowiąc moduł zintegrowanego narzędzia CASE, zgodnie z założeniem ma zdolność importowania obiektów istniejących w innych modelach, ale tylko obiektów wskazanych przez projektanta.

Po wywołaniu programu DataArchitect, podobnie jak w przypadku modułu ProcessAnalyst, otwierane jest okno udostępniające przestrzeń projektową, wraz z paletą narzędzi oraz belką menu programu.



Rys. 7.2. Belka menu i paleta narzędzi programu DataArchitect

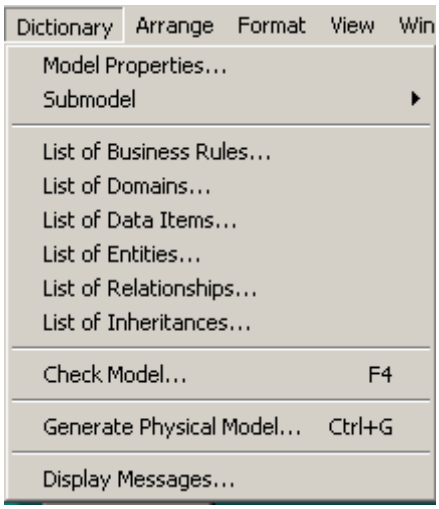
Łatwo zauważyć, że belka menu jest identyczna jak dla programu ProcessAnalyst. Różnice wystąpią oczywiście w opcjach menu *Dictionary*. Zaczynając jednak od palety narzędzi bezpośrednio związanych z budowaniem i modyfikowaniem diagramu E-R, istotne są narzędzia pozwalające umieszczać w polu projektu encje, powiązania między encjami oraz określać dziedziczenie (rys. 7.3).



Rys. 7.3. Narzędzia z palety bezpośrednio związane z budowaniem modelu CDM

Cały czas należy pamiętać, że opis obiektów umieszczanych w polu projektu przenoszony jest do słownika pakietu, dlatego też, kasując graficzną reprezentację obiektu, trzeba również dokonać kasowania w słowniku.

Na osobną uwagę zasługują opcje menu słownika danych (*Dictionary*) – podobnie jak w przypadku programu ProcessAnalyst w opcjach tych zawarta jest idea i główne funkcje programu (rys. 7.4). Ze zrozumiałych względów podręcznik niniejszy nie prezentuje dokładnego opisu technicznego pakietu, lecz jedynie jego główne funkcje używane w procesie modelowania danych. Kompletny opis pakietu zainteresowani znajdą w dokumentacji technicznej firmy Sybase [23] lub chociażby w obszernym systemie pomocy (*Help*) programu.



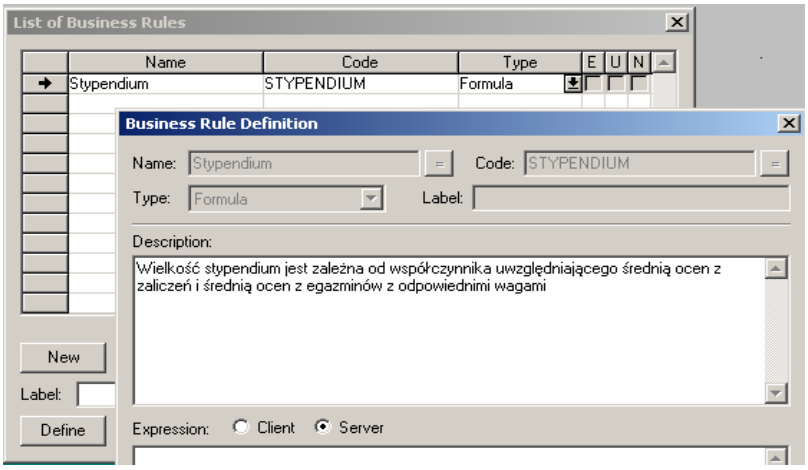
Rys. 7.4. Rozwinięcie menu *Dictionary*

Opcje menu słownika pakietu (*Dictionary*) udostępniają listy wszystkich obiektów wchodzących w skład modelu: encji, dziedzin, elementów danych (atrybutów), reguł biznesowych, powiązań oraz dziedziczeń. Oprócz danych słownikowych poprzez opcje tego menu można uruchomić funkcję sprawdzania poprawności modelu (oczywiście kontrola dotyczy tylko poprawności formalnej) oraz funkcję generowania modelu fizycznego (PDM) dla wybranego środowiska bazodanowego. W tym miejscu należy zwrócić uwagę, że program traktuje każde z definiowanych powiązań i dziedziczeń jako obiekty, których opis jest przechowywany w słowniku. O ile większość list słownika dotyczy pojęć uprzednio zdefiniowanych, o tyle nowym pojęciem są reguły biznesowe (*Business Rules*). Na poziomie modelu konceptualnego można traktować reguły biznesowe jako słowny opis zasad działania obowiązujących w modelo-

wanej rzeczywistości. Każda z reguł traktowana jest jako obiekt modelu, który powinien zostać związany z innym obiektem (encją, powiązaniem, atrybutem itp.). W modelu konceptualnym/logicznym reguły zazwyczaj nie są wyposażane w wykonywalny kod – kod umożliwiający implementację reguł definiowany jest na poziomie modelu PDM. Zapisując regułę jako obiekt wchodzący w skład modelu CDM, należy jednak pamiętać, że – podobnie jak inne obiekty – musi mieć ona nazwę oraz deklarację typu zgodną ze standardami programu DataArchitect. Każda reguła biznesowa musi być przypisana do jednego z czterech typów:

- Fakt – przykładowo: student może studiować na więcej niż jednym wydziale.
- Definicja – przykładowo: studentem jest osoba legitymująca się indeksem lub ważną legitymacją studencką.
- Wzór – przykładowo: wielkość stypendium jest zależna od współczynnika uwzględniającego średnią ocen z zaliczeń i średnią ocen z egzaminów z odpowiednimi wagami.
- Walidacja – przykładowo: najniższe stypendium za wyniki w nauce nie może być mniejsze niż najniższe stypendium socjalne.

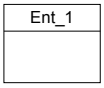
W tym miejscu warto przypomnieć, że program DataArchitect jest modułem zintegrowanego narzędzia CASE, tak samo jak przedstawiony uprzednio program ProcessAnalyst. Dlatego też, jeżeli jakiekolwiek reguły biznesowe (to samo dotyczy innych obiektów, szczególnie elementów danych) zostały utworzone na poziomie modelu systemu, można je zaimportować, wykorzystując opcję *Import* z menu *File*. Jeśli tworzona jest nowa reguła, należy uruchomić standardową ścieżkę: z menu *Dictionary* → *List of Business Rules*, co powoduje pojawienie się okna umożliwiającego definiowanie reguł (rys. 7.4), tzn. wpisanie i zakodowanie nazwy oraz określenie typu reguły. Za pomocą klawisza *Define* uruchamiane jest następne okno umożliwiające wpisanie treści słownej reguły. Kod implementacyjny na tym etapie nie musi być wpisywany.



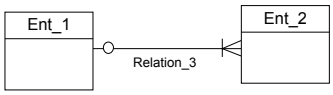
Rys. 7.5. Definiowanie reguł biznesowych

Graficzne reprezentacje składowych diagramu ER stosowane w różnego typu narzędziach CASE mogą się różnić. W programie DataArchitect stosowane są następujące konwencje:

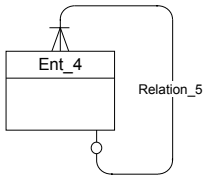
• Encja – reprezentowana jest poprzez prostokąt z wpisaną nazwą encji (pakiet standardowo nadaje obiektowi numer, ale z zasady projektant powinien nadać encji nazwę odzwierciedlającą typ lub klasę encji).



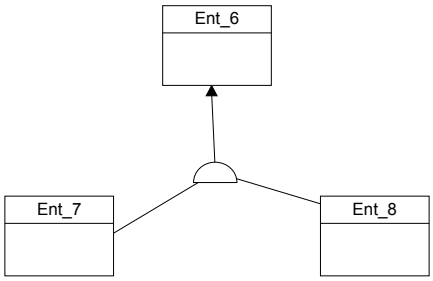
• Związki – reprezentowane są poprzez linię łączącą ramki encji. W zależności od typu związku (jeden do jeden, jeden do wielu, wiele do wielu) koniec linii odpowiadający określeniu „wiele" przybiera kształt „kurzej stopki" (*crowsfoot*). Dla zaznaczenia uczestnictwa encji w związku (wymagane/opcjonalne) używany jest odpowiednio znak „|" (uczestnictwo wymagane) lub znak „○" (uczestnictwo opcjonalne).



• Związki rekurencyjne reprezentowane są linią krzywą rozpoczynającą się i kończącą w obszarze pojedynczej encji.



• Dziedziczenie – encja Ent_6 jest nadklasą, mającą dwie encje potomne Ent_7 i Ent_8.



Przedstawione ogólne konwencje stosowane w programie DataArchitect i dotyczące składowych diagramu E-R są ilustracją wykorzystania odpowiednich narzędzi z palety programu. Przystępując do budowania modelu danych dla konkretnego wycinka rzeczywistości (inaczej mówiąc, przystępując do określenia potrzeb informacyjnych organizacji czy firmy), należy każdy z obiektów zdefiniować zgodnie z wynikami analizy, odzwierciedlającymi istotne założenia systemu, mając na uwadze, że jakość systemu informatycznego zależy od jakości modelu danych.

Podany przykład budowania diagramu E-R za pomocą programu DataArchitect ilustruje zarówno ideę modelowania danych, jak i możliwości narzędzia, czyli technologii CASE.

Przykład 7.1

Zadaniem projektanta jest zbudowanie modelu danych i następnie schematu bazy danych, będącej fundamentem aplikacji wspomagającej zarządzanie małą firmą usługową umożliwiającą swoim klientom czynne spędzanie wolnego czasu w siłowni, na basenie, zajęciach aerobiku itp. Tego typu firmy są powszechnie nazywane klubami fitness. Głównym zadaniem programu jest wspomaganie bieżącej obsługi klienta, a więc zautomatyzowanie pracy recepcji klubu. Praca ta polega na przyjmowaniu rezerwacji od klientów na konkretne zajęcia. Rezerwacji można dokonać tylko na te zajęcia, na które są wolne miejsca, co zależy od liczby miejsc na sali, która jest przydzielona do zajęć i oczywiście od już dokonanych rezerwacji. Zajęcia mogą się odbywać z indywidualnym trenerem lub samodzielnie, ale zawsze jest pracownik, odpowiedzialny za konkretne zajęcia, wykonujący rutynowe czynności związane z obsługą sali i sprzętu. Zasadą jest, że klienci, którzy chcą korzystać z usług klubu muszą założyć rachunek, na który z góry wpłacają ustaloną kwotę pieniędzy, ponieważ po dokonaniu rezerwacji opłata za określony typ zajęć jest automatycznie odejmowana od rachunku. Gdy klient rezygnuje z rezerwacji, kwota zapłacona za zajęcia jest wówczas zwracana na jego rachunek. Główne czynności recepcji to:

- udzielanie klientom informacji o zajęciach,
- rezerwacja zajęć,
- odwoływanie rezerwacji,
- zapisywanie nowych klientów,
- przyjmowanie wpłat od klientów,
- drukowanie harmonogramów zajęć,
- drukowanie wykazów rezerwacji.

Ponieważ firma Fitness Club jest firmą małą, dyżury recepcyjne pełnią więc kolejno wszyscy jej pracownicy.

Pierwszym krokiem budowania diagramu E-R jest zdefiniowanie zbioru encji. Encję wyodrębnia się na podstawie opisu działania firmy, korzystając z definicji encji,

która mówi, że jest to logicznie wyodrębniony zestaw danych, opisujący obiekt z modelowanej rzeczywistości, czy inaczej – encja jest to obiekt, o którym informacje powinny być przechowywane w systemie. W praktyce często rzeczowniki powtarzające się w opisie są encjami. Po uważnej analizie opisu działania Fitness Club można przyjąć, że klasą encji jest KLIENT: na pewno dane klientów (a więc instancje encji) są danymi istotnymi, klient pełni główną rolę w przedsięwzięciu, ponadto łatwo można wyodrębnić zestaw atrybutów opisujących tę encję.

Powtarzającym się w opisie rzeczownikiem jest nieco enigmatyczne określenie „recepcja". Przyglądając się uważnie rzeczywistości, czyli zadaniom i czynnościom tzw. recepcji, nie mamy wątpliwości, że recepcja to zbiór konkretnych pracowników obsługujących klientów (rezerwacje, odwołania, wpłaty). Zdanie takie znajduje się zresztą w opisie. Można więc wyodrębnić następną klasę encji: PRACOWNIK.

Istota działania firmy jest związana z oferowanymi zajęciami, oferta może klientów przyciągać lub zniechęcać, ale niewątpliwie musi być dostępna. Jeżeli tak, to w modelu powinna pojawić się encja ZAJĘCIA.

W opisie środowiska użytkownika wyraźnie określono, że zajęcia odbywają się na salach i to różnej wielkości, a więc kolejna encja to SALA.

Większa część opisu została dokładnie przeanalizowana, pozostają jednak jeszcze dwie kwestie:

1. Pierwsza kwestia dotyczy rezerwacji zajęć przez klientów. Termin „rezerwacja" w zasadzie określa czynność, ale należy pamiętać, że encja to nie tylko osoba lub przedmiot – to również pojęcia abstrakcyjne dotyczące zdarzeń, takie jak transakcja czy rezerwacja. W tym przypadku rezerwacja powinna być encją opisaną odpowiednimi atrybutami. Kolejną klasą encji jest więc REZERWACJA.

2. Następna kwestia związana jest z tzw. myśleniem zdroworozsądkowym – opis środowiska użytkownika nie precyzuje bowiem tej kwestii, ale każdemu w miarę uważnemu czytelnikowi nasunie się pytanie: jeżeli klientom oferowane są różne typy zajęć, często bardzo specjalizowane, to co ze sprzętem? Wątpliwości tego typu można rozwiać, konsultując się z przyszłym użytkownikiem systemu; akurat w tej kwestii wystarczyłby telefon z zapytaniem. Załóżmy więc, że otrzymaliśmy odpowiedź, że każda sala ma przypisany sobie określony zestaw przyrządów (sprzętu). Musimy zatem dane dotyczące sprzętu ująć w modelu jako encję SPRZĘT.

Jeżeli przyjmiemy, że zbiór encji został określony, to następnym krokiem jest ustalenie związków pomiędzy encjami, czyli odzwierciedlenie reguł działania danego środowiska. Cały czas należy pamiętać, że budujemy abstrakcyjny model danych, a więc nie są ważne sposoby implementacji związków, lecz jedynie ich wystąpienia. W opisie środowiska użytkownika podobnie jak rzeczowniki często oznaczają encje, tak czasowniki oznaczają związek, chociaż nie jest to regułą.

Prowadząc analizę opisu pod kątem ustalenia związków pomiędzy encjami, dość łatwo zaobserwować, że istnieje powiązanie pomiędzy encją KLIENT a encją REZERWACJA, które słownie można zdefiniować jako:

KLIENT dokonuje REZERWACJI.

Wiadomo również, że rezerwacje dotyczą określonych zajęć, a więc musi istnieć związek pomiędzy rezerwacjami a oferowanymi zajęciami. Związek ten można słownie zdefiniować jako:

REZERWACJA dotyczy TYPU ZAJĘCIA.

Zajęcia odbywają się w określonych salach, czyli do określonej sali przydzielone są określone typy zajęć, a więc:

SALA ma przypisane TYPY ZAJĘĆ.

Kwestia sprzętu – z uszczegółowionego opisu wynika, że pomiędzy salą a sprzętem również zachodzi związek:

SPRZĘT jest przydzielony do SALI.

Nieco trudniejszą sprawą wydaje się być ustalenie, czy istnieją i jakiego rodzaju związki pomiędzy pracownikami Fitness Club a klientami. Bezpośredni związek pomiędzy pracownikiem klubu a klientem występuje w sytuacji, kiedy klient życzy sobie indywidualnego trenera; wtedy można zapisać:

PRACOWNIK trenuje KLIENTA.

Następne związki dotyczą encji PRACOWNIK. Pracownicy klubu z definicji związani są z przydzielonymi im zajęciami, a więc:

PRACOWNIK obsługuje ZAJĘCIA.

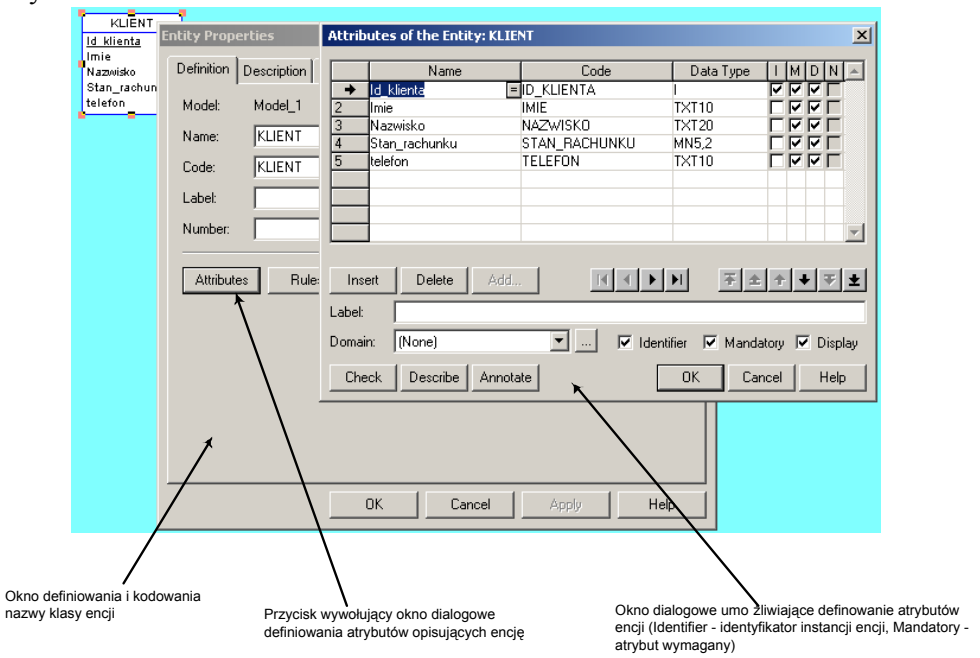
oraz

PRACOWNIK przyjmuje rezerwację.

Zakończony został wstępny etap związany z budową modelu danych: wyabstrahowane zostały klasy encji (nadane nazwy) oraz sprecyzowane werbalnie związki pomiędzy nimi. W kolejnym etapie należy przypisać klasom encji atrybuty, które je opisują, zdefiniować jednoznaczne identyfikatory instancji encji oraz doprecyzować definicję związków, czyli określić typ związku, liczność i uczestnictwo encji w związku. Mając przemyślaną koncepcję modelu i zarysowane główne obiekty wchodzące w jego skład, wygodniej będzie posłużyć się programem DataArchitect w konstruowaniu diagramu E-R niż opracowywać go „ręcznie".

Po wywołaniu programu DataArchitect pracę nad diagramem E-R rozpoczynamy od umieszczenia w polu projektu symboli encji. Służy do tego celu odpowiednie narzędzie z palety narzędzi (rys. 7.3). Definiowanie encji polega na nadaniu jej nazwy, przydzieleniu atrybutów, określeniu dziedzin atrybutów oraz określeniu atrybutu (lub atrybutów) identyfikujących. Każdy z tych kroków wykonywany jest poprzez wywołanie odpowiednich okien dialogowych. Wszystkie decyzje definicyjne odnotowywa-

ne są przez program w słowniku (*Dictionary*). Opisany schemat postępowania ilustruje rysunek 7.6.



Rys. 7.6. Widok pola projektu z oknami dialogowymi definiowania encji

Na rysunku 7.6 przedstawiono realizację procesu modelowania danych w zakresie definiowania encji: ustalenie atrybutów opisujących klasę encji, ustalenie atrybutu identyfikującego, jak również decyzje precyzujące, czy atrybut jest wymagany, czy opcjonalny (oznacza to w praktyce podjęcie decyzji, czy dla każdej instancji encji wartość danego atrybutu musi być znana (atrybut wymagany) – przykładowo, czy każdy z klientów musi podać numer telefonu kontaktowego). Dla każdego atrybutu określany jest również typ danych (dziedzina). W omawianym przykładzie założono, że każdy obiekt będzie identyfikowany poprzez identyfikator nadany w ramach systemu. Również dziedzina atrybutu jest równoznaczna z typem danych – nie zachodziła potrzeba bardziej precyzyjnej definicji dziedzin. W identyczny sposób definiowane są pozostałe encje wchodzące w skład modelu (rys. 7.7). Po zakończeniu etapu definiowania encji należy określić wyabstrahowane związki pomiędzy poszczególnymi encjami.

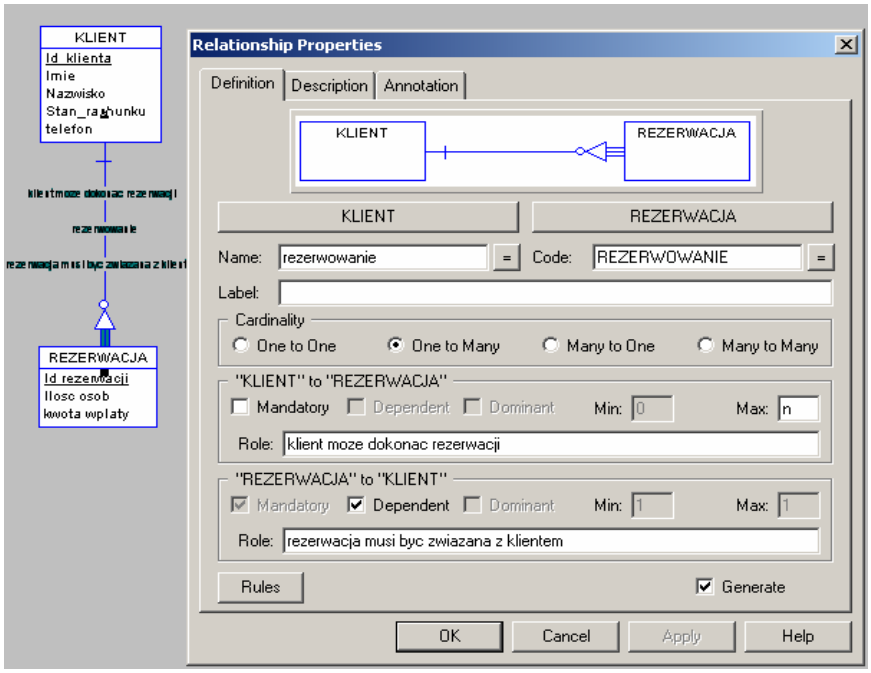
Uwaga: Gdy jako pierwszy zbudowany został diagram DFD, wtedy określony dla modelu systemu zbiór elementów danych związanych z przepływami i magazynami może zostać zaimportowany ze słownika programu ProcessAnalyst.

Sposób definiowania związków pokazano na przykładzie związku między encją KLIENT i REZERWACJA. Słownie związek został sformułowany w sposób bardzo

ogólny: KLIENT dokonuje REZERWACJI. W celu zaznaczenia tego związku na dia-

Rys. 7.7. Encje wchodzące w skład diagramu E-R dla firmy Fitness Club

gramie E-R należy posłużyć się odpowiednim narzędziem z palety narzędzi (rys. 7.7) i wyrysować powiązanie od encji KLIENT do encji REZERWACJA, a następnie, poprzez kliknięcie myszką symbolu związku, wywołać okno dialogowe umożliwiające jego zdefiniowanie (rys. 7.8).



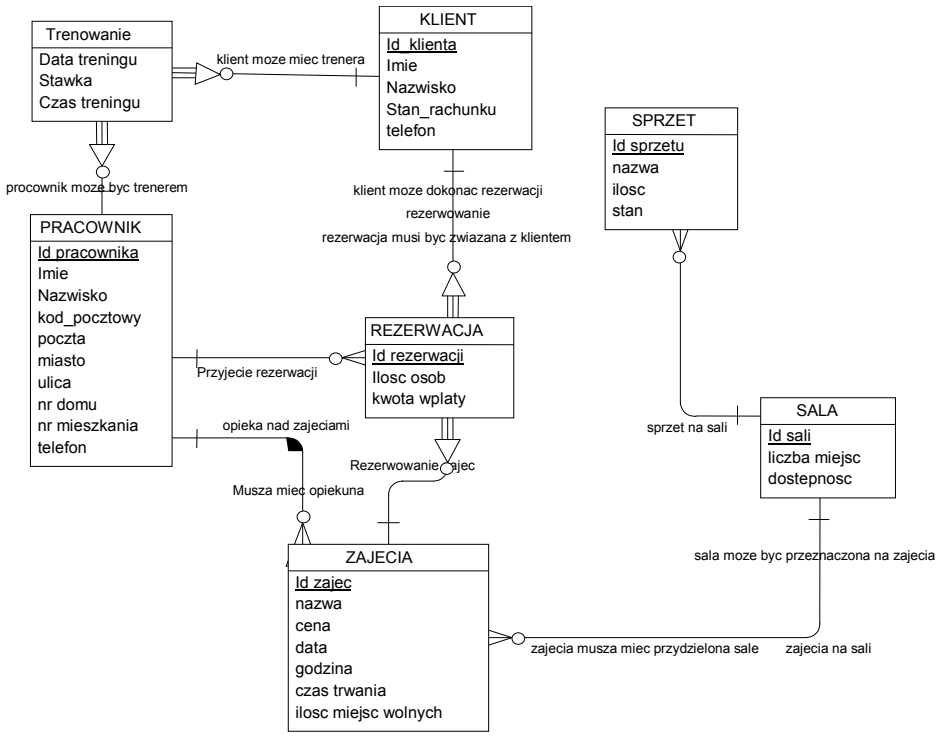
Rys. 7.8. Okno dialogowe definiowania związku

Pełna definicja związku zawiera określenie liczności związku (inaczej typu związku ze względu na liczbę instancji encji uczestniczących w związku). Jak widać, według tego kryterium związki mogą być typu: jeden do jeden (*one to one*), jeden do wiele (*one to many*), wiele do jednego (*many tu one*) lub wiele do wiele (*many to ma-*

ny). Określenie liczebności związku wynika z reguł działania modelowanej rzeczywistości. Aby prawidłowo zakwalifikować związek, należy umieć odpowiedzieć na pytania odnoszące się do obu stron związku:

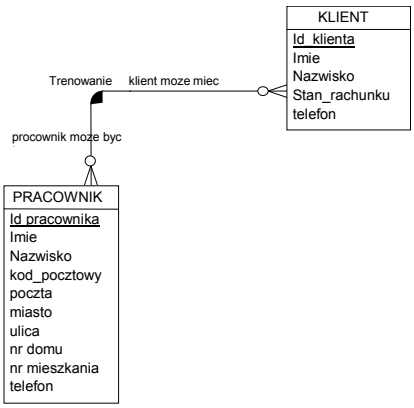
- Ilu rezerwacji może dokonać klient klubu? – odpowiedź: kilka, dowolną liczbę (wiele).
- Czy rezerwacja jest związana z konkretnym klientem? – odpowiedź: musi być związana z jednym klientem; nawet jeśli dotyczy grupy osób, ktoś jeden dokonuje rezerwacji, uiszcza opłatę i ma prawo rezerwację odwołać.

Związek dla powyższego przykładu określony został jako jeden do wielu. Jest to zresztą najczęściej pojawiający się typ związku. Następnie w oknie dialogowym należy określić uczestnictwo instancji każdej z encji (opcjonalne lub wymagane) w definiowanym związku. Uczestnictwo klientów w związku zostało określone jako opcjonalne (klient może, ale nie musi dokonać rezerwacji), natomiast rezerwacja musi być przypisana konkretnemu klientowi, stąd ten koniec związku opisany został jako wymagany (*mandatory*). Dodatkowo wybrany został parametr określający, że wystąpienie encji REZERWACJA jest zależne (*dependent*) od wystąpienia encji KLIENT (nie może zostać założona rezerwacja bez istniejącego klienta). W oknie dialogowym podaje się również nazwę związku, która powinna wskazywać, dlaczego takie powiązanie istnieje. Można, chociaż nie jest to wymagane, opisać słownie znaczenie (rolę) instancji encji w związku. W podobny sposób należy zdefiniować wszystkie związki w modelu danych.



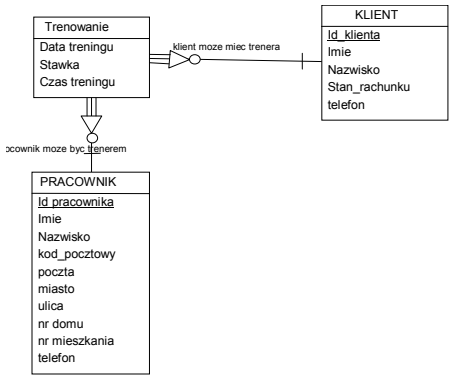
Rys. 7.9. Diagram E-R dla firmy Fitness Club

W przedstawionym diagramie E-R wszystkie związki zostały zakwalifikowane jako związki jeden do wielu. Może się jednak zdarzyć, że pojawią się związki wiele do wiele. Taki typ związku w diagramie E-R, budowanym za pomocą programu DataArchitect, jest reprezentowany dwojako: albo tylko jako związek wiele do wiele, albo za pomocą tzw. encji asocjacyjnej, w której umieszczane są atrybuty opisujące związek. Aby zilustrować tę sytuację, załóżmy, że pracownik może być trenerem dla różnych klientów, ale również, że ambitni klienci mogą pracować z kilkoma trenerami, w różnych terminach i za różną stawkę. Fragment modelu E-R przedstawiony na rys. 7.9 obrazujący związek między pracownikiem a klientem musi ulec zmianie (rys. 7.10).



Rys. 7.10. Reprezentacja związku wiele do wiele

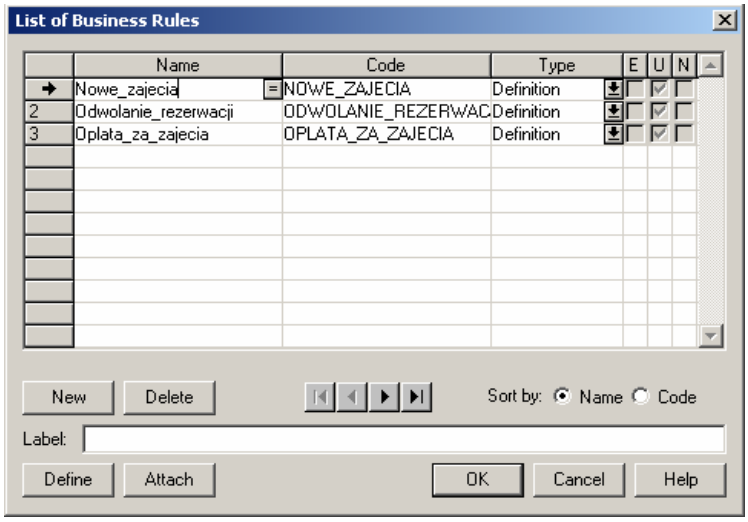
Związek wiele do wiele może zostać zastąpiony przez encję asocjacyjną. W tym celu należy wywołać edycję związku poprzez kliknięcie prawym klawiszem myszki symbolu związku. Z menu kontekstowego należy wybrać opcję *Change to Entity* (*Zamień na encję*). Program umieści symbol encji asocjacyjnej z nazwą odpowiadającą nazwie nadanej związkowi. Jeżeli związek zawiera atrybuty, które go opisują, należy dodać je do encji (rys. 7.11).



Rys. 7.11. Encja asocjacyjna dla związku wiele do wiele

Najrzadziej pojawiającym się typem związku jest typ jeden do jeden. W praktyce związek ten oznacza, że prawdopodobnie jedna z encji jest niepotrzebna.

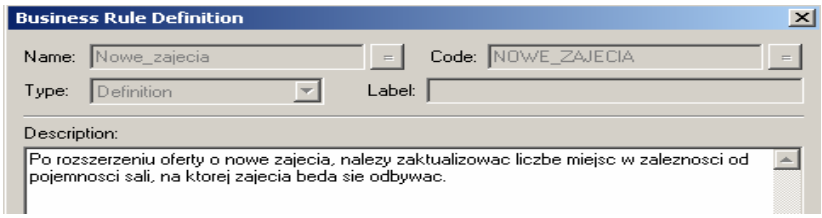
Końcowym etapem pracy nad modelem jest umieszczenie zestawu reguł biznesowych, precyzujących zasady obowiązujące w modelowanej rzeczywistości i związanie ich z odpowiednimi obiektami. Dla omawianego przykładu zostały wyspecyfikowane odpowiednie reguły (rys. 7.12):



Rys. 7.12. Zestaw reguł dla modelu danych firmy Fitness Club

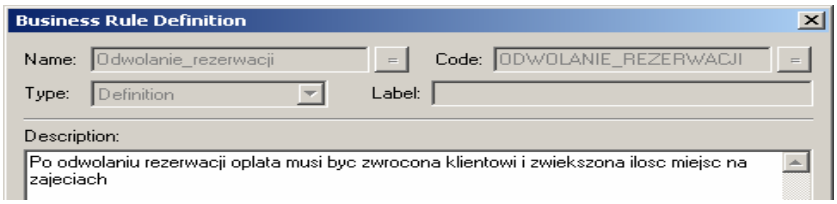
Znaczenie reguł:

1. Nowe zajęcia:



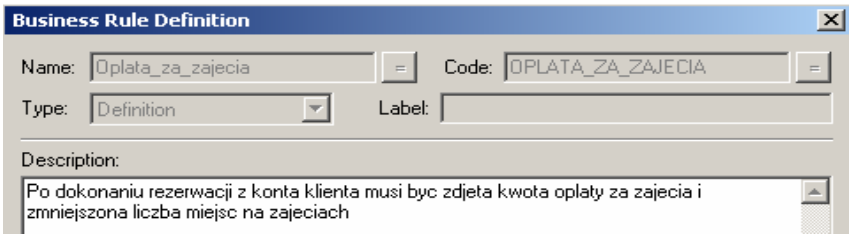
Reguła została dołączona do encji ZAJĘCIA.

2. Odwołanie rezerwacji:



Reguła została dołączona do encji REZERWACJE.

3. Opłata za zajęcia:



Reguła została dołączona do encji REZERWACJE.

Po zakończeniu pracy nad modelem konceptualnym należy wykorzystać opcję *Check Model* z menu *Dictionary*, w celu sprawdzenia poprawności formalnej modelu. Jeżeli model nie zawiera błędów formalnych, można przystąpić do drugiej fazy projektu – generowania modelu fizycznego, na podstawie którego zaimplementowana zostanie baza danych.

7.3. Zasady pracy z programem DataArchitect – model fizyczny

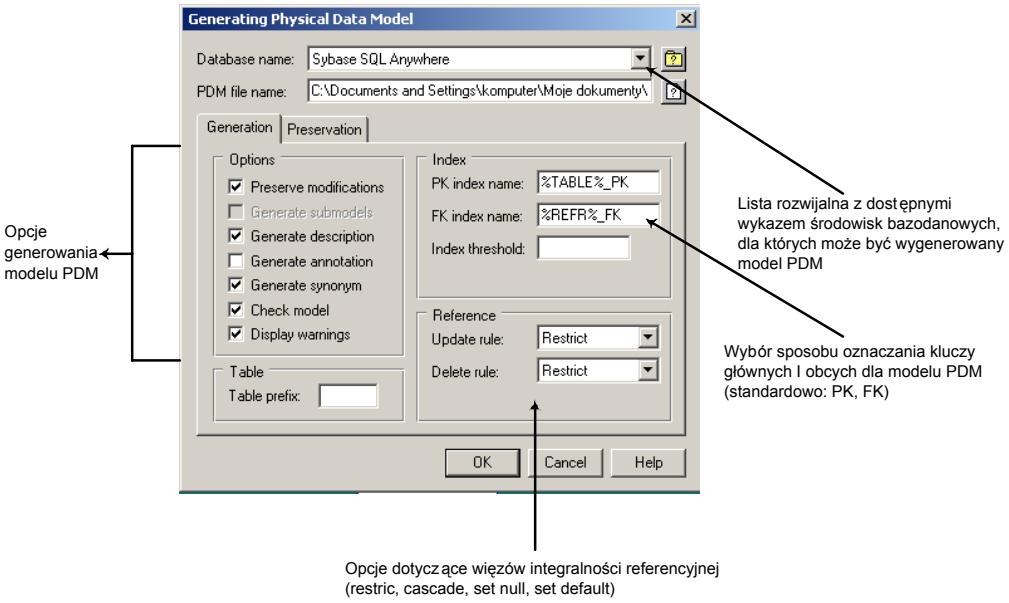
Model fizyczny (relacyjny) danych (PDM) związany jest z określonym środowiskiem bazodanowym (Systemem Zarządzania Bazą Danych). Generowany jest na podstawie modelu konceptualnego, z uwzględnieniem specyfiki wybranego środowiska. Posługując się programem DataArchitect, na podstawie jednego modelu CDM można wygenerować kilka modeli PDM, dla różnych środowisk bazodanowych. Na poziomie modelu PDM przeprowadza się optymalizację charakterystyki (tzw. strojenie pod kątem poprawy wydajności) bazy danych poprzez wprowadzenie indeksów, utworzenie perspektyw, modyfikację więzów integralności referencyjnej, wprowadzenie triggerów czy procedur, kierując się wynikami analizy użycia danych, czyli odpowiedziami na pytania:

- Jaki rodzaj informacji będzie pobierany z bazy danych?
- Jak często?
- W jakiej formie?

Pojęcia z zakresu modelowania związków encji mają swoje odpowiedniki w modelu relacyjnym:

Model CDM	**Model PDM**
Encja	Tabela
Wystąpienie encji	Wiersz tabeli
Atrybut	Kolumna tabeli
Atrybut identyfikujący	Klucz główny
Związek	Klucz obcy

Generowanie modelu relacyjnego na podstawie modelu konceptualnego odbywa się poprzez opcję menu *Dictionary → Generate Phisical Model*. Jak już stwierdzono, należy zdecydować, poprzez wybór z listy rozwijalnej okna dialogowego, jaki ma być docelowy DBMS (System Zarządzania Bazą Danych).



Rys. 7.13. Okno dialogowe z opcjami generacyjnymi modelu PDM

Ustawienie okna dialogowego takie, jak na rys. 7.13 powoduje, że model PDM będzie generowany z wyświetleniem komunikatów (*Display Warnings*), generowanie zostanie poprzedzone sprawdzeniem modelu (*Check Model*), w modelu PDM klucze główne będą oznaczone przez PK (*Primary Key*), klucze obce przez FK (*Foreign Key*). Opcja *Preserve Modifications* powoduje, że kolejne polecenia generowania modelu PDM na skutek zmian w modelu konceptualnym będą skierowane do pliku z tą samą nazwą, z zachowaniem poprzednich ustaleń.

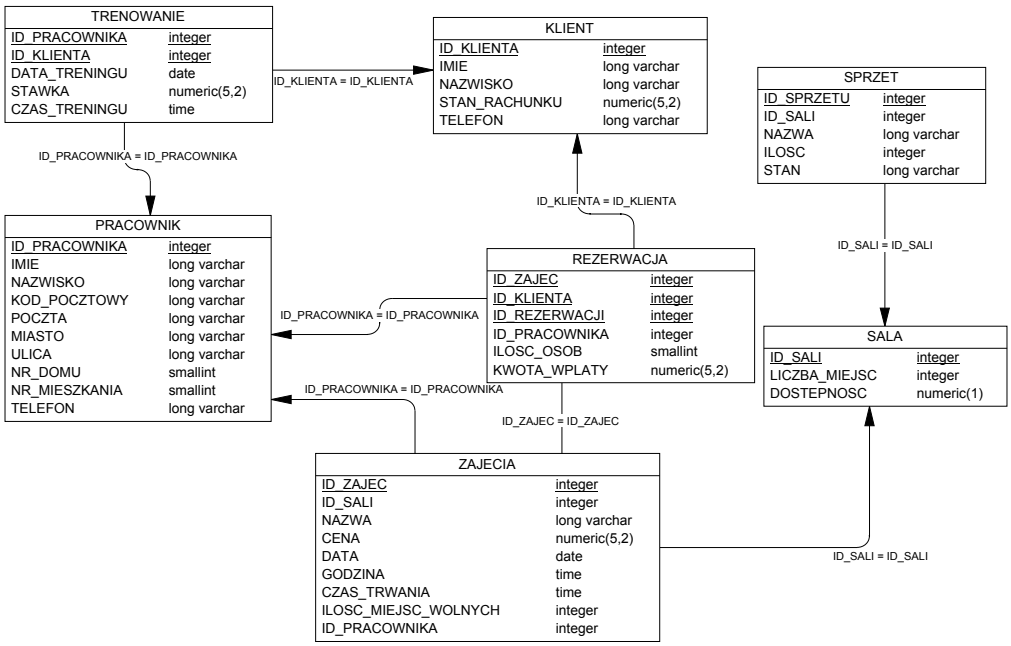
Na szczególną uwagę zasługuje obszar okna dialogowego umożliwiający definiowanie reguł integralności referencyjnej. Przywołując treść rozdziału 1. odnoszącą się do implementacji związków pomiędzy obiektami (encjami) oraz reguł dotyczących więzów integralności przypomnijmy, że *integralność referencyjna* związana jest z ogranicze-niami wartości klucza obcego. W kluczu obcym mogą wystąpić dwa rodzaje wartości:

- wartość z dziedziny klucza głównego, gdy istnieje związek pomiędzy danymi w tabelach,
- wartość *null*, jeżeli nie ma takiego związku lub gdy stwierdzamy, że związek jest nieznany.

Reguły dotyczące więzów nie tylko określają, czy w kluczu obcym mogą pojawić się wartości *null*, czy też muszą wystąpić wartości z dziedziny klucza głównego. Reguły dotyczące integralności referencyjnej muszą precyzować, co będzie się działo w przypadku usuwania czy modyfikacji danych w tabelach, które są ze sobą powiązane. Dysponujemy czterema możliwościami:

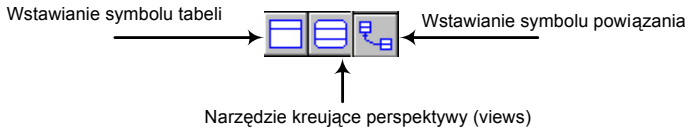
- Usuwanie i aktualizowanie ograniczone (*restricted*) – podejście restrykcyjne: nie można kasować wierszy z tabeli, jeżeli w innej tabeli występują wiersze powiązane, nie można zmienić wartości klucza głównego w tabeli, jeżeli z tą wartością powiązane są wiersze w innej tabeli.
- Usuwanie kaskadowe (*cascades*) – po usunięciu wierszy z jednej tabeli wszystkie wiersze powiązane zostają automatycznie usunięte; podobnie w przypadku aktualizacji zmiana wartości klucza głównego powoduje kaskadową aktualizację wartości klucza obcego.
- Reguła wstaw *null* (*set null*) – po usunięciu wierszy z jednej tabeli, w powiązanych wierszach w kolumnie klucza obcego ustawiona zostanie wartość *null*.
- Reguła wstaw wartość domyślną (*set default*) – po usunięciu wierszy z jednej tabeli, w powiązanych wierszach w kolumnie klucza obcego ustawiona zostanie wartość wybrana wartość domyślna.

Uwaga: Wybrane reguły obowiązują dla całego modelu PDM. Jeśli do niektórych fragmentów powinny obowiązywać inne reguły, należy dokonać modyfikacji poszczególnych powiązań.



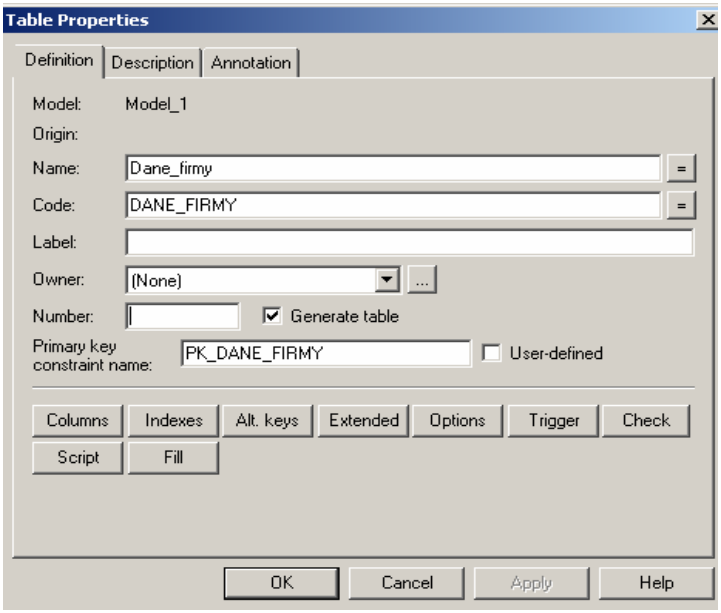
Rys. 7.14. Relacyjny model danych odpowiadający modelowi konceptualnemu z przykładu 7.1

Na rysunku 7.14 przedstawiono model PDM wygenerowany na podstawie modelu CDM, zgodnie z ustawieniami okna dialogowego (rys. 7.13). Prace nad modelem PDM odbywają się według tych samych zasad jak w przypadku modelu CDM, dotyczą jednak innych obiektów. Paleta narzędzi oferuje inne możliwości (rys. 7.15).



Rys. 7.15. Paleta narzędzi wspomagających budowę i modyfikacje modelu PDM

Chcąc rozszerzyć model PDM o dodatkową tabelę, możemy umieścić symbol tabeli w polu projektu, za pomocą narzędzia z palety, a następnie zdefiniować wprowadzony obiekt za pomocą okna dialogowego, którego opcje są adekwatne do wymagań definicji tabeli (rys. 7.16).

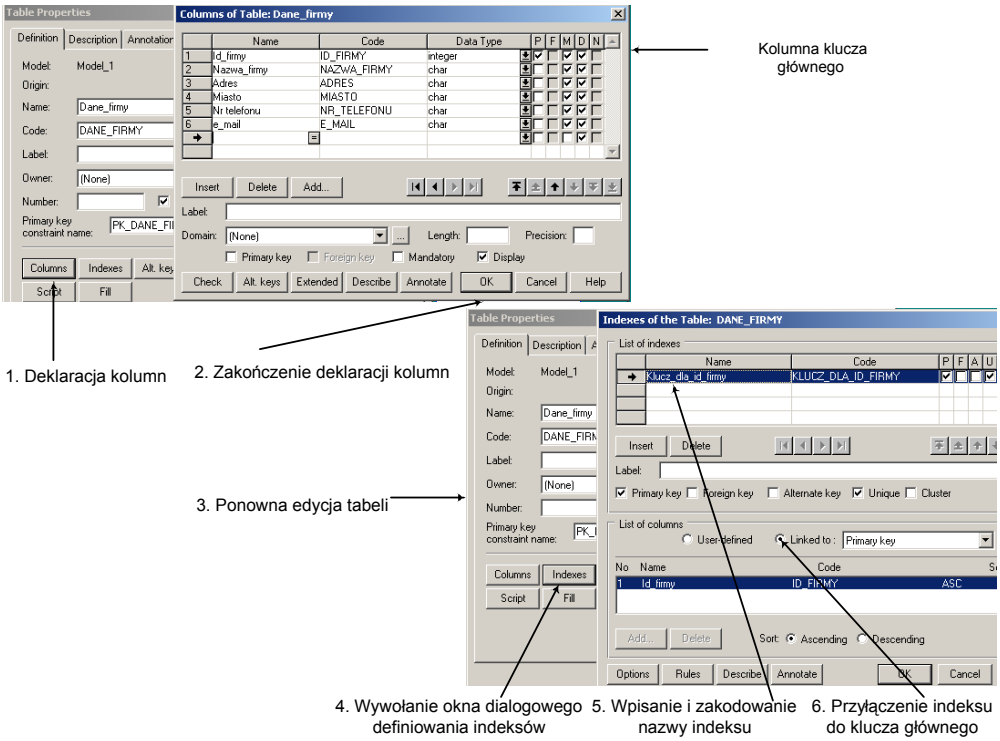


Rys. 7.16. Okno dialogowe definiowania tabeli w modelu PDM

Za pomocą okna dialogowego, dla nowo tworzonej tabeli, należy określić kolejno nazwę tabeli, zadeklarować kolumny (lub dołączyć z listy kolumny już istniejących), określić, która kolumna (lub zestaw kolumn) będzie pełniła rolę klucza głównego. Należy zwrócić uwagę, że dla tabel, wygenerowanych przez pakiet na podstawie encji modelu CDM, do klucza głównego automatycznie dołączany jest indeks. W większości systemów baz danych klucze główne są indeksowane automatycznie, co powoduje

przyspieszenie wyszukiwania danych według kryteriów odnoszących się do wartości klucza głównego. Dołączając tabelę na poziomie modelu PDM, należy więc do kolumny, pełniącej rolę klucza głównego dodać indeks, według następującego schematu (rys. 7.17):

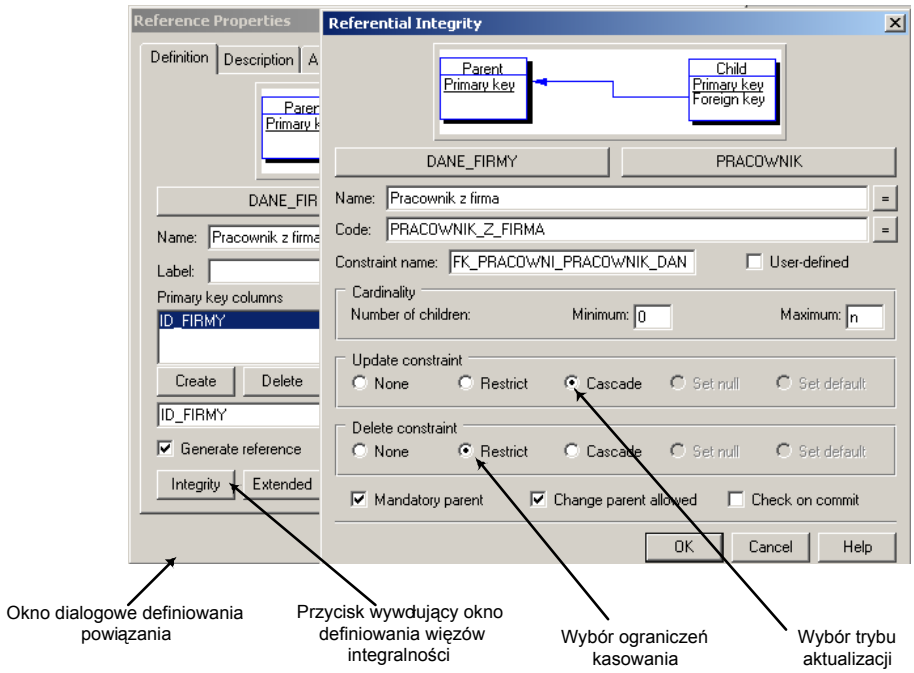
- Zadeklarować kolumny wchodzące w skład tworzonej tabeli (podanie nazwy, zakodowanie nazwy, określenie typu danych, określenie, czy wprowadzanie danych do kolumny jest wymagane, czy opcjonalne, wybór klucza głównego).
- Zakończyć definiowanie tabeli.
- Ponownie wywołać do edycji tabelę.
- Przyciskiem *Indexes* uruchomić okno dialogowe umożliwiające deklarowanie indeksów.
- Wpisać i zakodować nazwę indeksu.
- Za pomocą opcji *Linked to* przyłączyć utworzony indeks do klucza głównego tabeli.



Rys. 7.17. Algorytm definiowania tabeli w obszarze modelu PDM

Jeśli nowa tabela ma zostać połączona z inną, to – podobnie jak w modelu CDM – wykorzystuje się narzędzie z palety narzędzi umożliwiające graficzne zaznaczenie powiązania (rys. 7.15), po czym powiązanie takie należy w pełni zdefiniować poprzez okno dialogowe.

Uwaga: To samo okno dialogowe używane jest do edycji istniejących powiązań, wygenerowanych na podstawie modelu CDM. Warto pamiętać, że w modelu PDM może zachodzić potrzeba ponownego zdefiniowania niektórych powiązań, ze względu na to, że więzy integralności referencyjnej zostały ustalone generalnie dla całego modelu.



Rys. 7.18. Definiowanie i edycja powiązań między tabelami

Rysunki 7.16, 7.17 i 7.18 ilustrują dodanie do relacyjnego modelu danych z przykładu 7.1 tabeli, o nazwie DANE_FIRMY, powiązanej z tabelą PRACOWNIK według następujących kryteriów: FIRMA zatrudnia wielu pracowników; jeżeli z firmą związani są pracownicy, nie jest dozwolone kasowanie danych firmy; jeżeli dane firmy ulegną zmianie (*Id_firmy*) kaskadowo, to zmienione zostaną wartości w kluczu obcym tabeli PRACOWNIK.

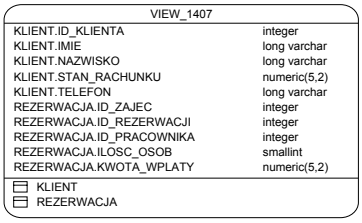
Następnymi obiektami, które występują na poziomie modelu PDM są perspektywy, odzwierciedlające różne sposoby postrzegania i wykorzystywania danych przez różnych użytkowników. Dokładne zasady i cele tworzenia perspektyw przedstawiono w rozdziale 4. Warto pamiętać, że utworzenie perspektywy jest ekwiwalentne z włączeniem zapytania SQL (w formie obiektu) do modelu danych. Oznacza, to, że podczas generowania skryptu umożliwiającego założenie bazy danych wraz z poleceniami utworzenia tabel, kluczy głównych, implementacji powiązań (klucze obce) wygenerowane zostaną polecenia utworzenia perspektyw. Przesłanki do tworzenia perspektyw wynikają z analizy wykorzystania zestawów danych przez różne grupy użytkowników (różne grupy pracują

na różnych zestawach danych, niekoniecznie na całej bazie danych, okresowo zadawane powtarzalne zapytania, zwłaszcza zawierające funkcje agregujące itp.). Program DataArchitect umożliwia tworzenie perspektyw w trybie graficznym, co dla wielu projektantów jest trybem znacznie wygodniejszym niż układanie zapytań w języku SQL.

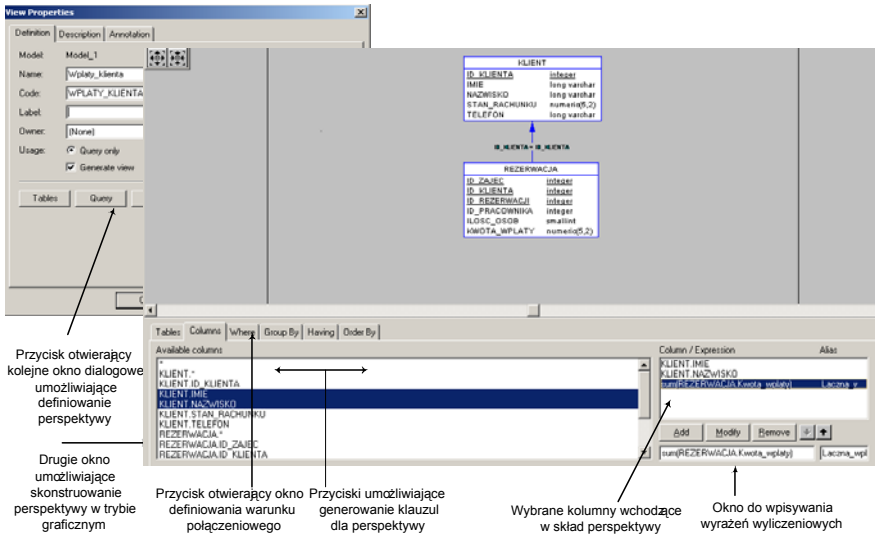
W odniesieniu do przykładu 7.1 zasadne wydaje się utworzenie perspektywy umożliwiającej śledzenie stanu wpłat poszczególnych klientów. Ponieważ dane klientów są przechowywane w tabeli KLIENT, natomiast dane dotyczące wpłat bieżących w tabeli REZERWACJA, potrzebne zestawienia może więc udostępnić perspektywa oparta na tych dwóch tabelach.

W celu utworzenia takiej perspektywy należy:

- W polu projektu zaznaczyć wybrane tabele (pierwszą poprzez kliknięcie myszką, drugą – kliknięcie z klawiszem *Shift*).
- Wybrać opcje menu *Dictionary* →*Views* →*New*, co skutkuje umieszczeniem w obszarze modelu PDM wstępnego schematu perspektywy, w którym występują wszystkie kolumny z wybranych tabel:

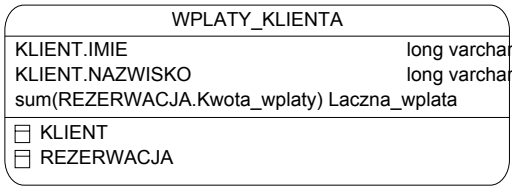


- Dostosować perspektywę do rzeczywistych potrzeb prezentacji danych poprzez kolejne wywołanie okien dialogowych umożliwiające definiowanie obiektu.

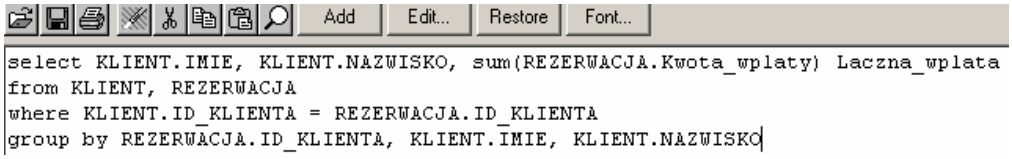


Rys. 7.19. Okna dialogowe definiujące perspektywę

Efektem zastosowania ustawień pokazanych na rys. 7.19 jest perspektywa udostępniająca zestawienie danych klientów (*Imię, Nazwisko*) oraz sumę wpłat wniesionych przez każdego z klientów (*sum*(*REZERWACJA.Kwota_wpłaty*)). Konstruowanie treści zapytania tworzącego perspektywę odbywa się za pomocą przycisków udostępnianych w drugim oknie dialogowym. Każdemu z przycisków odpowiada fragment kodu SQL, który jest generowany przez program DataArchitect. W każdej chwili można poprzez przycisk SQL w drugim oknie dialogowym zobaczyć składnię SQL dla budowanej graficznej prezentacji perspektywy. W omawianym przykładzie graficzny obraz perspektywy w modelu PDM wygląda następująco:



co odpowiada zapytaniu w języku SQL (wygenerowanemu przez program):



Kolejnymi obiektami wchodzącymi w skład modelu PDM są procedury zdarzeń (*triggery*), które generalnie w programie DataArchitect wykorzystywane są do zaimplementowania więzów integralności. Generowane są automatycznie, zgodnie z ustawieniami zadeklarowanymi przy generowaniu modelu relacyjnego. Podobnie jak inne obiekty bazy danych, triggery mogą być modyfikowane, projektant może również tworzyć swoje własne procedury zdarzeń, wykorzystując zestaw szablonów programu. W takich sytuacjach kody procedur zdarzeń mogą być generowane w formie skryptu bądź bezpośrednio do przyłączonej bazy danych. Taki algorytm postępowania dokładnie opisuje dokumentacja techniczna programu DataArchitect.

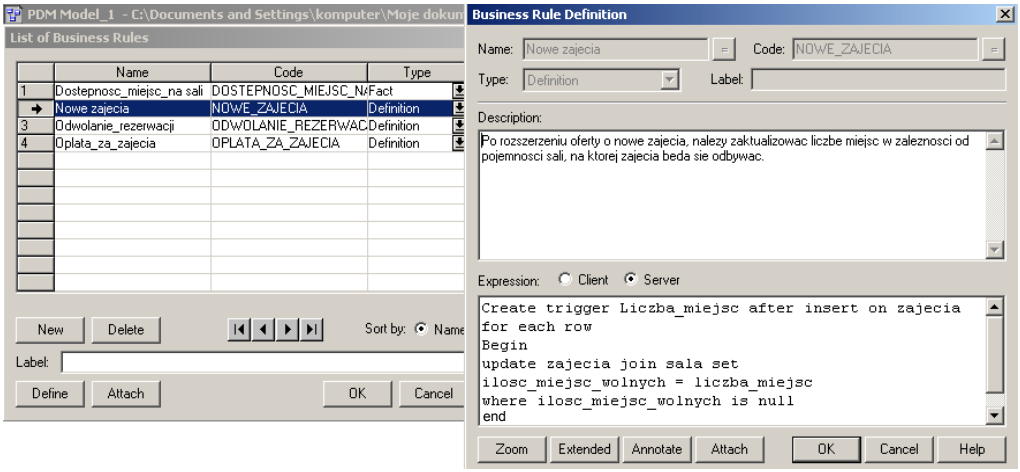
W przypadku rozważanego modelu danych dla firmy Fitness Club, triggery mogą mieć zastosowanie do implementacji reguł biznesowych. Przypomnijmy zdefiniowane w modelu konceptualnym reguły biznesowe dla modelowanej rzeczywistości:

- Po rozszerzeniu oferty o nowe zajęcia liczba dostępnych miejsc na tych zajęciach odpowiada pojemności sali, na której będą odbywać się zajęcia (*Nowe_zajecia*).
- Po dokonaniu rezerwacji z konta klienta musi być zdjęta kwota opłaty za zajęcia i zmniejszona liczba wolnych miejsc na zajęciach (*Oplata_za_zajecia*).
- Po odwołaniu rezerwacji opłata musi być zwrócona klientowi i zwiększona liczba wolnych miejsc na zajęciach (*Odwolanie_rezerwacji*).

W modelu CDM reguły miały postać słownego opisu, w modelu PDM należy nadać im realizowalność techniczną. Analizując treść poszczególnych reguł pod kątem mechanizmów operowania na bazie danych, należy zauważyć, że dotyczą one wstawiania lub kasowania danych zgromadzonych w tabelach; ponadto operacje dotyczące jednej tabeli powinny wywoływać określone zmiany w innej tabeli. Z tego powodu wygodnym sposobem implementacji takich reguł biznesowych jest zdefiniowanie triggerów, które są wykonywane automatycznie w określonych sytuacjach.

Lista reguł biznesowych została utworzona już w modelu konceptualnym, na poziomie modelu relacyjnego należy doprecyzować definicję każdej z nich poprzez rozwinięcie listy słownikowej ze spisem reguł i uruchomienie okna dialogowego dla każdej z nich (przycisk *Define*).

Dla reguły pierwszej w oknie dialogowym wybrane zostało miejsce implementacji reguły (serwer); w dolnej części okna wpisana została treść triggera realizującego słownie zdefiniowaną regułę (rys. 7.20).



Rys 7.20. Definiowanie reguł biznesowych

Zgodnie z intencją projektanta treść triggera określa, że po wpisaniu (*after insert*) nowej pozycji do tabeli ZAJECIA, automatycznie modyfikowana będzie kolumna tej tabeli (*ilosc_miejsc_wolnych*) poprzez umieszczenie w niej zawartości kolumny (*liczba_miejsc*) z tabeli SALA.

Następnym etapem jest związanie definicji reguły biznesowej z odpowiednim triggerem dla tabeli ZAJECIA, czyli wstawienie reguły biznesowej do ciała procedury zdarzenia. Ogólny sposób realizacji tego etapu dla programu DataArchitect polega na wykonaniu kilku kroków:

- Uruchomieniu opcji menu *Dictionary* → *Triggers and Procedures* → *List of Triggers*, co powoduje otwarcie okna dialogowego umożliwiającego wybór tabeli, z którą

związany będzie trigger, wybranie z listy triggerów określonych dla danej tabeli tego, który zostanie zmodyfikowany poprzez włączenie treści reguły biznesowej do ciała triggera, lub określenie nowego triggera (*user define*) dla danej tabeli.

• Przyciskiem Zoom wywoływane jest kolejne okno dialogowe, umożliwiające włączenie reguły biznesowej do ciała triggera, co obrazuje rys. 7.21.

• Kolejność kroków po otwarciu drugiego okna dialogowego jest następująca:

– dokonać wyboru z listy kategorii (w omawianym przypadku kategorią jest *business rules*),

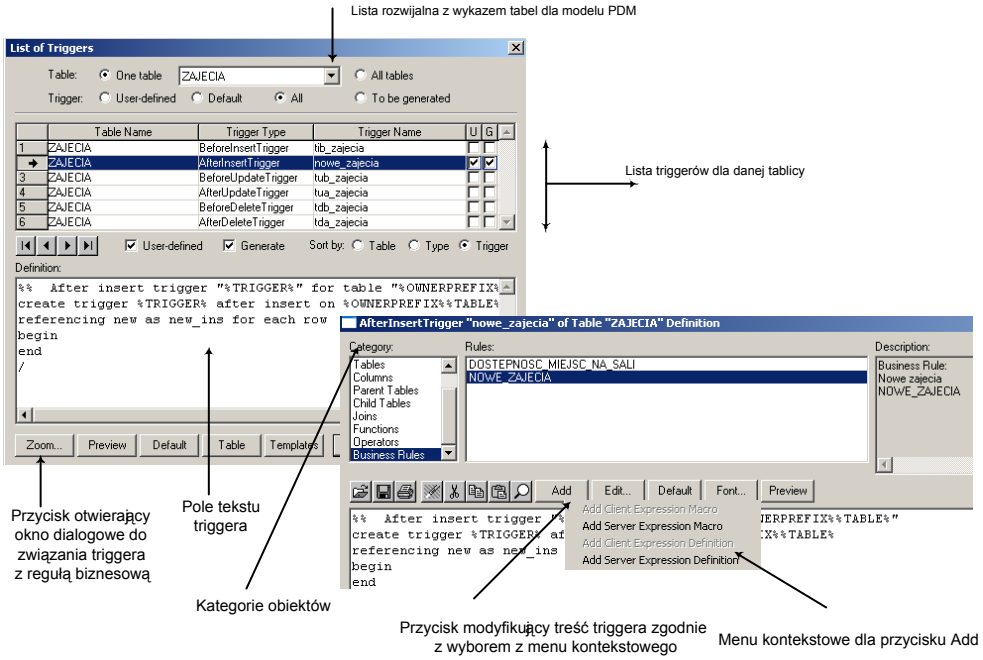
– z listy reguł biznesowych, widocznych w prawym panelu, należy wybrać żądaną,

– umieścić kursor w dolnej części okna, w miejscu, w którym ma być umieszczona treść reguły,

– uruchomić przycisk Add,

– z menu kontekstowego wybrać odpowiednią opcję (w tym przypadku *Add Server Expression Definition*), co spowoduje przeniesienie wyrażenia definiującego regułę biznesową do ciała triggera.

• Zakończyć całą procedurę przyciskiem OK.



Rys. 7.21. Włączanie reguły biznesowej do procedury zdarzenia (triggera)

Cała procedura w formie opisowej instrukcji wydawać się może dość skomplikowana, ale jeżeli kolejne kroki wykonywane są podczas pracy z programem DataArchi-

tect, to okazuje się, że algorytm postępowania nie jest ani trudny merytorycznie, ani zbyt czasochłonny. Trudność polega raczej na dobrym sprecyzowaniu wyrażenia realizującego samą regułę biznesową, niż na stronie technicznej całego zagadnienia.

W identyczny sposób muszą zostać zaimplementowane dwie następne reguły biznesowe, związane z tabelą REZERWACJA.

Proponowane wyrażenia dla reguły drugiej (*Oplata_za_zajecia*):

```
Create trigger rezerwacja_dodaj after insert order1 on "DBA".REZERWACJA
For each row
Begin
Update ZAJECIA join REZERWACJA
set
ZAJECIA.Ilosc_miejsc_wolnych = Ilosc_miejsc_wolnych – REZERWACJA.Ilosc_osob
Where REZERWACJA.kwota_wplaty is null
End
Create trigger klient_placi after insert order 2 on "DBA".REZERWACJA
For each row
Begin
Update KLIENT join REZERWACJA,ZAJECIA
set
KLIENT_Stan_rachunku = Stan_rachunku – Cena * REZERWACJA.Ilosc_osob
Where (REZERWACJA.Kwota_wplaty is null)
and (ZAJECIA.Id_zajec = REZERWACJA.Id_zajec)
Create trigger ksieguj_wplate after insert order 3 on "DBA".REZERWACJA
For each row
Begin
Update REZERWACJA join ZAJECIA
set
REZERWACJA.Kwota_wplaty = ZAJECIA.Cena * REZERWACJA.Ilosc_osob
Where REZERWACJA.Kwota_wplaty is null
```

Po dołączeniu tych wyrażeń do zestawu triggerów dla tablicy REZERWACJA zostanie zaimplementowana reguła biznesowa, precyzująca zasady dokonywania rezerwacji zajęć w modelowanej rzeczywistości (pobranie kwoty z konta klienta, odnotowanie wpłaty i zmniejszenie liczby miejsc na zajęciach).

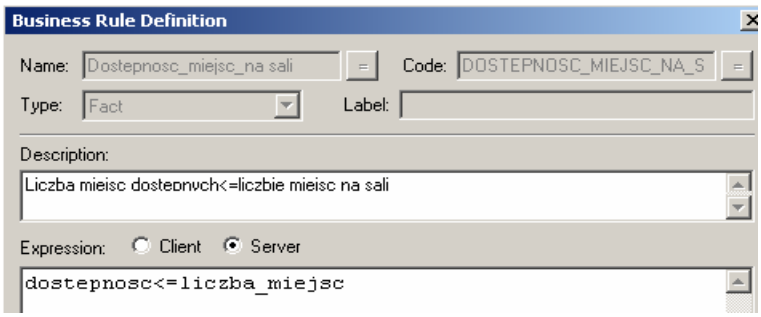
Dla reguły trzeciej (*Odwolanie_rezerwacji*):

```
Create trigger zwrot_pieniedzy before delete order 2 on "DBA".REZERWACJA
Referencing old as old_rezerwacja
For each row
Begin
Update KLIENT
```

```
Set
KLIENT.Stan_rachunku = KLIENT.Stan_rachunku + old_rezerwacja.kwota.wplaty
Where old_REZERWACJA.Id_klienta = KLIENT.Id_klienta
End
Create trigger usun_rezerwacje before delete order 1 on "DBA".REZERWACJA
Referencing old as old_rezerwacja
For each row
Begin
Update ZAJECIA
Set
ZAJECIA.Ilosc_miejsc_wolnych = Ilosc_miejsc_wolnych + REZERWACJA.Ilosc_osob
Where old_rezerwacja.Id_zajec = ZAJECIA.Id_zajec
End
```

Triggery zostają wywołane przed zdarzeniem cofnięcia rezerwacji. Kwota zapłacona przez klienta jest zwracana na jego konto, a liczba zajęć odpowiednio zwiększona.

Uwaga 1: Podane przykłady implementacji reguł biznesowych dotyczą konkretnej rzeczywistości. Oczywistym jest, że dla każdej firmy czy organizacji reguły te będą inne. Nie zawsze reguły biznesowe są implementowane poprzez procedury zdarzeń. W wielu przypadkach wyrażenia definiowane dla konkretnych reguł mają postać prostych walidacji lub wyrażeń. Na przykład, dla omawianego obszaru modelowania można by wyłonić taką zasadę: liczba dostępnych miejsc na sali nie może przekraczać jej pojemności. Przełożenie tego stwierdzenia na język modelu owocuje następującą regułą biznesową:



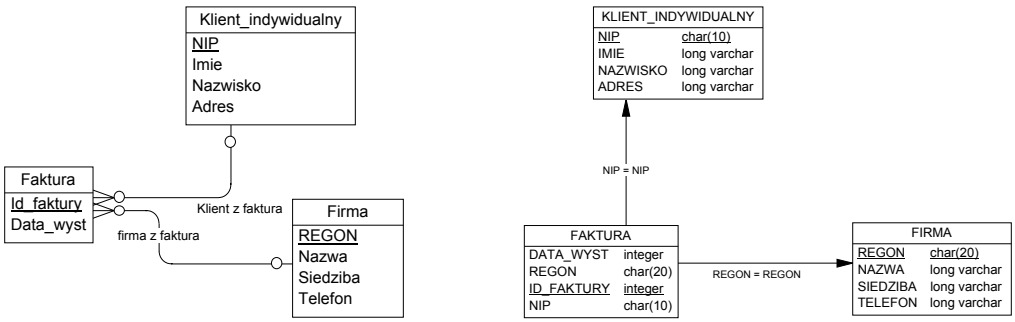
dołączoną do tabeli SALA.

Często projektant nie artykułuje pewnych zasad istniejących w modelowanej rzeczywistości jako reguł biznesowych, uwzględniając jednak ich istnienie w modelu. Najlepszym przykładem może być określanie atrybutów identyfikujących instancję encji (w modelu PDM kluczy głównych). Nie zapisuje się na ogół w formie reguły biznesowej stwierdzeń dotyczących sposobów identyfikowania instancji encji (przykładowo: pracownik jest identyfikowany poprzez numer PESEL, autor jest identyfi-

kowany przez imię i nazwisko – są to fakty w określonej rzeczywistości), reguła ta jest uwzględniana w formie identyfikatora encji lub klucza głównego tabeli.

Uwaga 2: Omawiany szerzej przykład 7.1 z oczywistych względów nie ilustruje wszystkich sytuacji, mogących pojawić się w modelowanej rzeczywistości. Jedną z takich sytuacji jest występowanie związków tzw. wzajemnie wykluczających się. Związki tego typu artykułowane są słownie poprzez określenie albo/albo. Dla zobrazowania modelowania i implementacji związków wykluczających się przeanalizujemy czysto ilustracyjny przykład:

Firma handlowa wystawia faktury swoim klientom. Z punktu widzenia firmy istotne jest rozróżnienie, która faktura została wystawiona klientowi indywidualnemu, a która podmiotowi gospodarczemu. Z punktu widzenia zarówno użytkownika, jak i projektanta klient indywidualny jest innym obiektem niż klient będący podmiotem gospodarczym (atrybuty opisujące klientów indywidualnych różnią się od atrybutów opisujących firmy). Wiadomo, że każda wystawiona faktura musi być przypisana konkretnemu klientowi, czyli – inaczej mówiąc – każda faktura musi mieć swojego właściciela: albo klienta indywidualnego, albo firmę. Niestety, sytuacji takiej, posługując się programem DataArchitect, nie można przedstawić na diagramie ER w sposób graficzny. W modelach wykonywanych ręcznie związki wykluczające najczęściej obrazowane są tzw. łukiem wykluczającym, przechodzącym przez linie odzwierciedlające związki między encjami. W przypadku modelu budowanego w środowisku CASE warto więc dołączyć reguły biznesowe, precyzujące taki rodzaj związków i określające sposób ich implementacji, który zależy od atrybutów identyfikujących instancje encji, a ściślej od dziedzin tych atrybutów. Diagram ER i model PDM dla przedstawionego kontekstu pokazano na rys.7.22.



Rys. 7.22. Związek wykluczający w modelu CDM i PDM

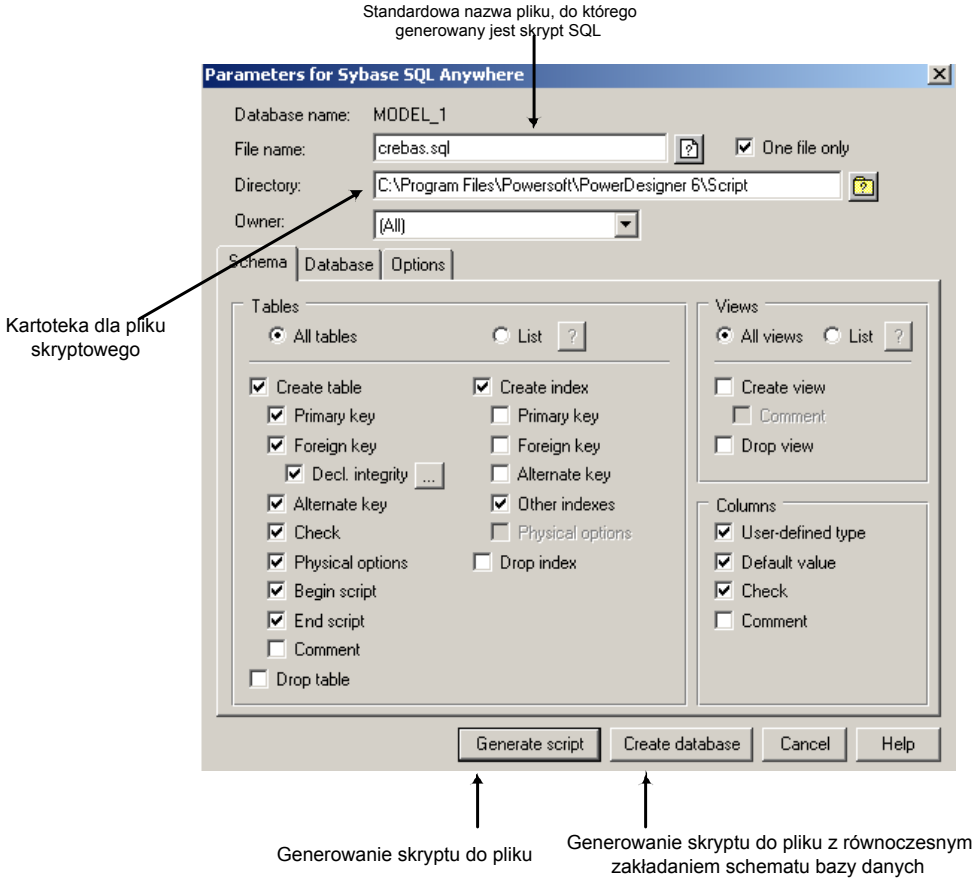
Niepokój budzić może definicja uczestnictwa encji w związku – uczestnictwo w modelu określono jako obustronnie opcjonalne, mimo że opis słowny zawiera stwierdzenie, że faktura musi mieć właściciela. Mając na uwadze sposób implementacji związków w modelu relacyjnym (poprzez klucze obce), gdyby diagram ER określał, że uczestnictwo encji FAKTURA jest w obu związkach wymagane, po

transformacji modelu konceptualnego na relacyjny powstałaby sytuacja patowa – do każdej faktury musiałby być wprowadzany w odpowiednie pola kluczy obcych zarówno identyfikator klienta indywidualnego, jak i podmiotu gospodarczego. Uwzględniając taką sytuację, zaproponowano podejście często stosowane w praktyce [2], polegające na odpowiedniej modyfikacji modelu relacyjnego w zależności od tego, czy klucze encji mają wspólną dziedzinę, czy nie. W przypadku pierwszym (wspólna dziedzina) w tabeli odpowiadającej encji FAKTURA należy umieścić dodatkową kolumnę, która będzie zawierała identyfikator związku (np. wartości KI lub PG, lub znacznik 0/1) i jedną kolumnę klucza obcego. W przypadku drugim (dziedziny różne) muszą wystąpić oba klucze obce, ale dopuszczające wartość *null*. Zapewnienie stanu spójnego bazy danych nakłada jednak na aplikację użytkową konieczność kontroli w postaci dodatkowego warunku spójności, takiego, że tylko do jednego z kluczy może odbywać się zapis i dokładnie jeden zapis musi mieć miejsce.

Ponieważ, jak już wspomniano, w przypadku wykorzystywania programu DataArchitect w budowaniu modelu danych nie ma możliwości graficznego odzwierciedlenia związku wykluczającego, warto zilustrować taką sytuację poprzez dołączenie reguły biznesowej (nawet w formie opisowej) do obu związków, co ma znaczenie zwłaszcza wtedy, gdy analityk kreujący model konceptualny nie jest równocześnie projektantem bazy danych i jego rola kończy się wraz z przekazaniem dokumentacji związanej z modelem konceptualnym i logicznym.

Wracając do przykładu wyjściowego, praca nad modelem relacyjnym w zasadzie dobiega końca. Jeżeli projektant uzna, że model PDM został ukończony i sprawdzony pod względem formalnym (menu *Dictionary* → *Check model*), to następnym krokiem jest generowanie skryptu zakładającego schemat bazy danych. Oczywiście model musi być sprawdzony również pod względem poprawności merytorycznej, a zwłaszcza pod kątem zgodności tabel z wymogami normalizacji. Zagadnienie normalizacji ze względu na ważkość tej tematyki zostanie szczegółowo przedstawione w następnym rozdziale. Etap sprawdzania poprawności tabel względem reguł normalizacji nie jest wspomagany przez program DataArchitekt, dlatego też, aby nie tracić z oczu całego toku postępowania i pracy nad modelem wykonywanej za pomocą narzędzia CASE, przejdziemy do końcowego etapu, czyli generowania skryptu *SQL* zakładającego schemat bazy danych, zwłaszcza że dysponując modelem PDM w formie pliku, łatwo można w nim dokonywać stosownych poprawek i ponownie przeprowadzić generowanie skryptu, jeżeli zajdzie taka potrzeba. Polecenie generowania skryptu jest uruchamiane poprzez wybór z menu *Database*, opcji *Generate Database*, co – zgodnie ze standardem pracy programu DataArchitect – powoduje uruchomienie okna dialogowego umożliwiającego ustawienie parametrów generowania (rys. 7.23). Okno dialogowe jest udostępniane z ustawieniami standardowymi, które można zmienić zgodnie z własnymi preferencjami. Szczegółowy opis wszystkich parametrów generacyjnych

jest zawarty w dokumentacji pakietu PowerDesigner. Ustawienia parametrów

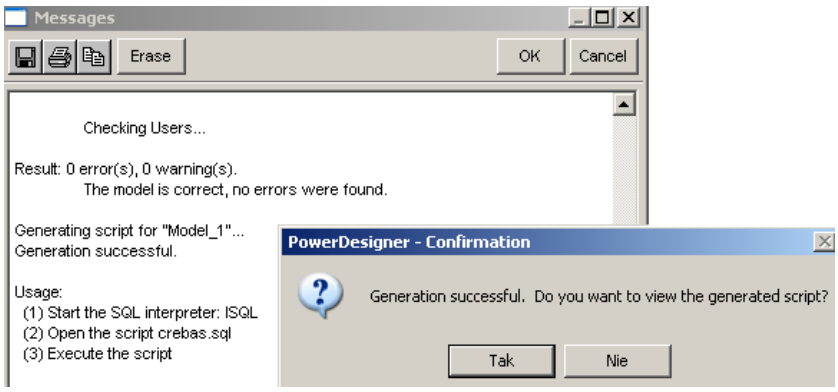


Rys. 7.23. Okno dialogowe z parametrami generowania skryptu *SQL*

generacyjnych są intuicyjnie zrozumiałe i wystarczająco czytelnie skomentowane przez etykiety okna, wątpliwości mogą budzić przyciski generuj skrypt (*Generate script*) i twórz bazę danych (*Create database*).

Różnica w ich działaniu jest zasadnicza – pierwszy z przycisków jedynie generuje skrypt SQL do pliku o standardowej nazwie *crebas.sql*; dla założenia schematu bazy danych wymagane jest otwarcie pliku poprzez interpreter SQL, np. przez aplikację *ISQL*, i uruchomienie go. Użycie drugiego przycisku powoduje automatyczne zakładanie schematu bazy danych w wybranym i przyłączonym obszarze bazy, z jednoczesnym zapisem skryptu do pliku *crebas.sql*. Rezultat generowania oraz algorytm postępowania w celu uruchomienia skryptu przedstawia okno komunikatu, po którym

pojawia się okno potwierdzenia generowania z opcjami natychmiastowego lub nie przeglądania skryptu (rys. 7.24).



Rys. 7.24. Okna komunikatu i potwierdzenia generacji skryptu *SQL*

Uwaga: Skrypt jest wygenerowany w dialekcie *SQL*, odpowiadającym wybranemu środowisku bazodanowemu, co jest konsekwencją wyboru środowiska dla modelu PDM. Nie ma żadnych przeszkód, aby na podstawie jednego modelu CDM wygenerować różne modele PDM i następnie skrypty *SQL* dla różnych środowisk.

7.4. Obiektowość w modelu relacyjnym

Obiektowość w modelu relacyjnym wyraża się poprzez możliwość definiowania encji nadrzędnych i encji potomnych według ogólnych zasad dziedziczenia dla podejścia obiektowego. Zgodnie z tymi zasadami encja nadrzędna zawiera atrybuty wspólne, encje potomne natomiast zawierają atrybuty charakterystyczne dla nich, dziedzicząc atrybuty wspólne. Program DataArchitect umożliwia wprowadzanie nadklasy encji i podklas do diagramu ER za pomocą narzędzia dziedziczenia z palety narzędzi, a także przeprowadzenie szczegółowej definicji dziedziczenia w standardowy dla pakietu sposób, czyli poprzez okna dialogowe.

Ponieważ w środowisku, którego dotyczył przykład 7.1 nie zachodziła potrzeba definiowania dziedziczenia, mechanizmy te zostaną zobrazowane prostym i wyłącznie ilustracyjnym przykładem dotyczącym innego obszaru modelowania, z hipotetycznie przyjętymi założeniami. Nie będzie to cały model, lecz wycinek związany z zagadnieniem dziedziczenia i jego transformacją na model relacyjny.

Przykład 7.2

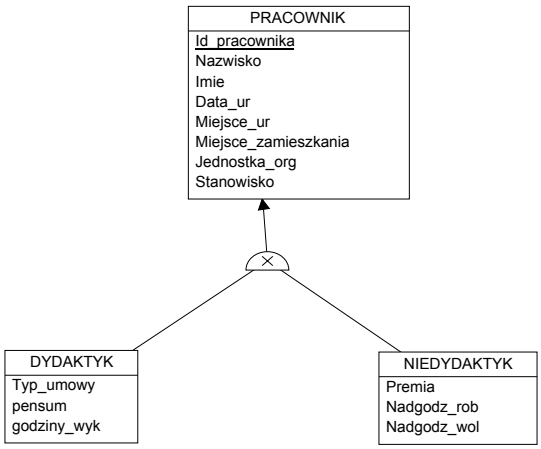
Obszar modelowania dotyczy środowiska uczelnianego, analizowanego pod kątem systemu płac. Pierwszym spostrzeżeniem jest podział pracowników na dwie ogólne

kategorie: pracownicy dydaktyczni i niedydaktyczni (w rzeczywistości podział jest bardziej skomplikowany, ten podział przyjęto jedynie przykładowo). Obie kategorie pracowników można opisać wspólnymi atrybutami, takimi jak: identyfikator pracownika, nazwisko, imię, data urodzenia, miejsce zamieszkania, jednostka organizacyjna, stanowisko. Każda z kategorii charakteryzuje się dodatkowo swoimi własnymi atrybutami, na przykład: typ umowy o pracę dla pracowników dydaktycznych może być mianowaniem bądź umową o pracę, każdy z pracowników dydaktycznych ma określone pensum godzinowe, w odniesieniu do każdego pracownika odnotowywana jest liczba zrealizowanych godzin dydaktycznych. Podobnie dla pracowników niedydaktycznych atrybutem jest przysługująca premia, odnotowywane są liczby nadgodzin przepracowanych w dni robocze i w dni wolne.

Z punktu widzenia analityka i projektanta sytuacja taka umożliwia wyodrębnienie nadklasy encji: PRACOWNIK oraz dwóch podklas encji: PRACOWNIK DYDAKTYCZNY i PRACOWNIK NIEDYDAKTYCZNY.

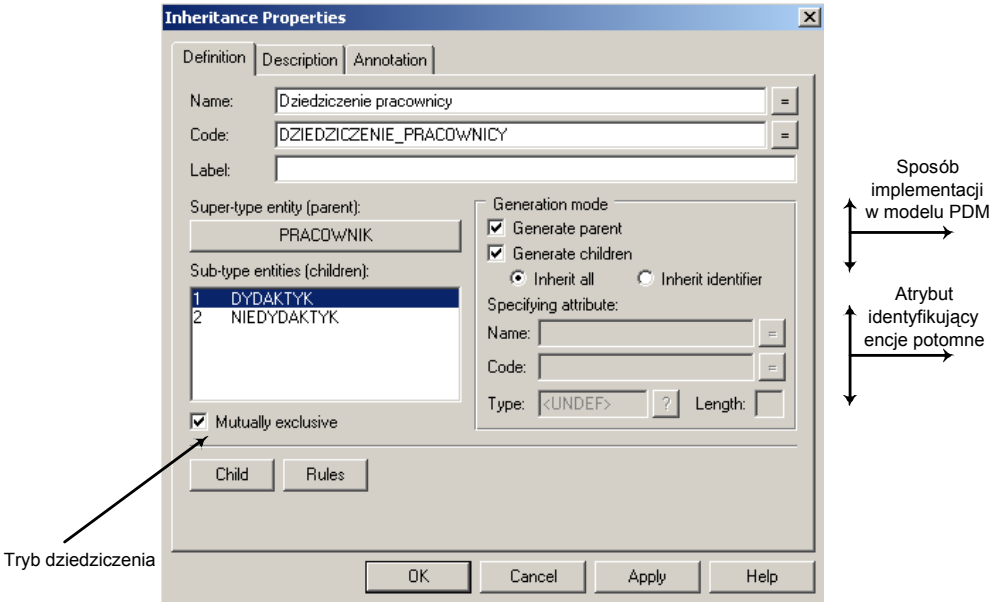
Konstruowanie diagramu E-R przebiega według znanej procedury:

- Umieszczenie w polu projektu encji oraz ich zdefiniowanie (nazwy, atrybuty, dziedziny).
- Zaznaczenie za pomocą narzędzia dziedziczenia z palety narzędzi związku pomiędzy encją rodzicem (PRACOWNIK) najpierw z jedną encją potomną, a następnie z drugą. Diagram E-R wygląda następująco:



- Kolejnym krokiem jest zdefiniowanie właściwości dziedziczenia poprzez wywołanie okna dialogowego dla obiektu dziedziczenia (rys. 7.25). Należy przypomnieć, że program DataArchitect traktuje dziedziczenie umieszczone w modelu CDM jako obiekt (podobnie jak związek, encja, czy perspektywa), którego nazwa i definicja mu-

szą zostać zapisane w słowniku programu. Okno dialogowe w przypadku definiowania dziedziczenia oprócz standardowych definicji zawiera opcje umożliwiające wybór parametrów transformacji modelu CDM na model PDM.



Rys. 7.25. Okno dialogowe definiujące dziedziczenie

Należy zwrócić uwagę na wybór trybu dziedziczenia – na rys. 7.25 zaznaczono tryb *Mutually exclusive*, co oznacza dziedziczenie wykluczające: nie można być jednocześnie pracownikiem dydaktycznym i niedydaktycznym. W modelu CDM dziedziczenie wykluczające zostanie automatycznie oznaczone przez pojawienie się znaku × wewnątrz symbolu dziedziczenia.

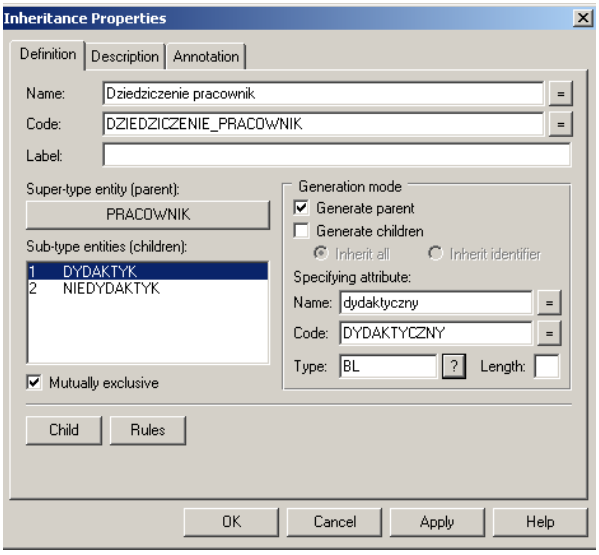
Kolejne deklaracje dotyczą sposobu transformacji modelu CDM na model PDM, tzn. czy w modelu relacyjnym ma być generowana jedna tabela zawierająca kolumny z encji rodzica i encji potomnych, czy generowane mają być tabele odpowiadające wszystkim encjom, czy też tylko tabele odpowiadające encjom potomnym.

W pierwszym przypadku (tylko jedna tabela) zostanie uaktywniona dolna część okna, w celu zdefiniowania atrybutu identyfikującego (rozróżniającego) encje potomne. W uaktywnione pola należy wpisać nazwę atrybutu oraz jego dziedzinę. W odniesieniu do omawianego przykładu właściwą dziedziną jest typ Boolean, ponieważ istnieją tylko dwie możliwości: pracownik jest pracownikiem dydaktycznym lub nie. Okno dialogowe dla takiego przypadku oraz – będący konsekwencją wybranych opcji – model PDM ilustruje rys. 7.26.

Uwaga: Planując realizację dziedziczenia w modelu relacyjnym w jednej tabeli należy uwzględnić, że atrybuty encji potomnych nie mogą zostać zdefiniowane jako

wymagane, lecz opcjonalne. Jako wymagane mogą być definiowane jedynie atrybuty encji rodzica. Wiąże się to z późniejszymi zapisami do bazy danych – kolumny tabel odpowiadające atrybutom wymaganym będą określone jako *not null*.

W praktyce taki sposób implementacji dziedziczenia stosuje się w przypadkach, gdy encje potomne mają małą liczbę atrybutów własnych, aby nie doprowadzić do sytuacji nierównomiernych zapisów do tabeli, z wieloma miejscami pustymi.



Ustawienia okna dialogowego dla trybu generowania jednej tabeli w modelu PDM

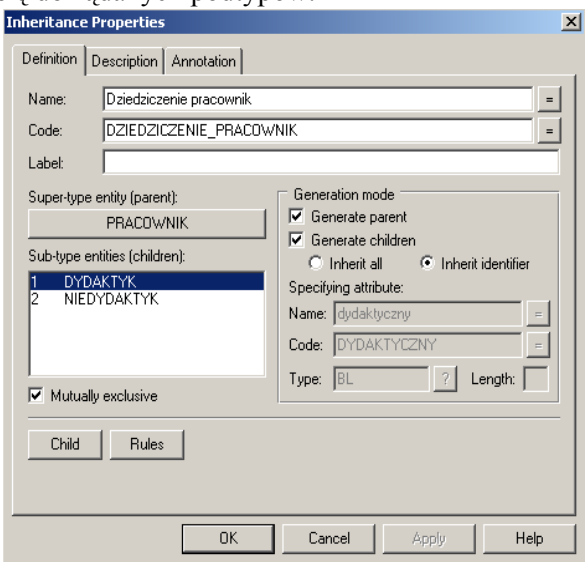
Model PDM – implementacja dziedziczenia w jednej tabeli

Rys. 7.26. Okno dialogowe definiujące dziedziczenie i opcje generowania jednej tabeli w modelu PDM

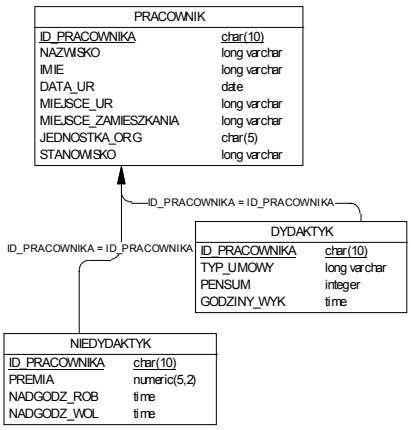
W przypadku drugim (generowane wszystkie tabele) należy podjąć decyzję, czy w tabelach odpowiadających encjom potomnym mają się znaleźć wszystkie dziedziczone atrybuty, czy jedynie klucze obce, poprzez które realizowane będzie powiązanie pomiędzy tabelą nadrzędną a tabelami podrzędnymi. Wybór opcji z dziedziczeniem wszystkich atrybutów prowadzi nieuchronnie do ogromnej redundancji danych – te same wartości będą przechowywane w kilku tabelach; z praktycznego punktu widzenia trudno znaleźć uzasadnienie dla takiej opcji. Ustawienia okna dialogowego oraz model PDM dla przypadku generowania trzech tabel powiązanych poprzez klucze ilustruje rys. 7.27. W przypadku generowania wszystkich tabel przy wprowadzaniu danych zachodzi konieczność uwzględnienia (podobnie jak w przypadku związków wykluczających) dodatkowych warunków spójności. Często wykorzystywanym mechanizmem jest perspektywa, zarówno dla odczytu, jak i wprowadzania danych.

Mniejsze wątpliwości budzi wariant generowania tabel odpowiadających encjom potomnym z dołączeniem kolumn dziedziczonych po encji rodzicu do każdej z tabel.

Tryb ten jest często stosowany w praktyce ze względu na późniejszy sposób pracy programów użytkowych z bazą danych – programy pracują na tabelach odnoszących się do żądanych podtypów.



Ustawienia dla trybu generowania osobnych tabel w modelu PDM



Model PDM –tabele powiązane poprzez klucz

Rys. 7.27. Okno dialogowe z opcją generowania wszystkich tabel w modelu PDM

Wszystkie przedstawione opcje implementowania dziedziczenia mają zarówno wady, jak i zalety, których nie można oszacować bezwzględnie, czyli bez pominięcia kontekstu (celu, jakiemu ma służyć projektowana baza danych, funkcji, jakie będą realizowane przez aplikacje użytkowe w odniesieniu do bazy danych, rodzaju pamiętanych danych itp.). Projektant może podjąć decyzję, która z opcji jest najwłaściwsza, opierając się na wynikach analizy całego systemu i mając na uwadze, że baza danych jest jedną z jego składowych, fundamentalną, ale nie samoistną.

7.5. Modele danych dla wybranych przykładów

Prezentowane w niniejszym rozdziale przykłady modeli danych są uzupełnieniem przykładów z rozdziału 6 dotyczących modelowania systemów, składową modelu systemu jest bowiem model danych. Projektant systemu często staje przed dylematem: czy pierwszy powinien być budowany model systemu, czy model danych? Odpowiedź na to pytanie nie jest oczywista. Często ważniejszą rzeczą jest zaprezentowanie użytkownikowi funkcji, jakie będzie realizował system, niż zobrazowanie struktury da-

nych, które system będzie przetwarzał. Z drugiej strony dane są przedmiotem przetwarzania, często więc zdarza się sytuacja, że model danych implikuje zmiany w modelu systemu. Optymalną strategią wydaje się więc podejście równoległe – budując model systemu, równocześnie budować model danych, pamiętając, że oba modele powinny być równoważone zgodnie z regułami:

- Każdemu magazynowi na diagramie DFD musi odpowiadać na diagramie ERD typ obiektu (w tym encje asocjacyjne) lub związek.
- Musi być zachowana zgodność nazw obiektów na obu typach diagramów.
- Pozycje słownika danych muszą mieć zastosowanie w obu modelach.
- Procesy muszą nadawać wartość elementom danych, będącym atrybutami obiektów w modelu danych.
- Procesy muszą dodawać, usuwać i modyfikować instancje obiektów modelu danych.

Model danych dla *przykładu 6.2* – Systemu Serwisowania Kas Fiskalnych

Opracowanie modelu danych dla tego systemu wymagało dokładnego zapoznania się z zasadami i przepisami obowiązującymi użytkowników kas fiskalnych, przepisy te rzutują bowiem na sposób pracy serwisu takich urządzeń oraz przetwarzane dane.

- Pierwszym objętym uregulowaniami ustawowymi pojęciem jest fiskalizacja kasy. Fiskalizacja kasy oznacza moment dokonania jednokrotnej i niepowtarzalnej czynności inicjującej pracę modułu fiskalnego kasy z pamięcią fiskalną kasy zakończonej wydrukiem dobowego raportu fiskalnego. Podczas tego procesu do pamięci fiskalnej zapisywany jest numer identyfikacji podatkowej użytkownika oraz unikatowy numer kasy. Numer unikatowy jest to unikalny i niepowtarzalny numer, poprzedzony dwuliterowym prefiksem, nadawany przez Ministerstwo Finansów, jednoznacznie identyfikujący każdą kasę. W momencie fiskalizacji drukowane są automatycznie następujące dokumenty:

– zawiadomienie podatnika o miejscu instalacji kasy,

– zawiadomienie serwisu o miejscu instalacji kasy.

Dokumenty te w ciągu siedmiu dni od daty fiskalizacji muszą być dostarczone do odpowiedniego urzędu skarbowego.

- Każda kasa musi być poddawana przeglądowi technicznemu co pół roku. Zarówno fiskalizacja, jak i przegląd techniczny muszą być wykonywane przez serwisanta, który posiada uprawnienia na wybrany model kasy.
- W przypadku zmiany numeru identyfikacji podatkowej użytkownika kasy zachodzi konieczność wymiany pamięci fiskalnej kasy i ponownej fiskalizacji. Pamięć fiskalna przed wymianą musi być odczytana w obecności podatnika, urzędnika skarbowego i serwisanta kasy.

Firmy serwisujące kasy fiskalne należą do kategorii małych firm, zatrudniających nie więcej niż kilkunastu pracowników. Przy zakupie kasy przez klienta do systemu muszą być wpisane dane klienta, dane zakupionej kasy (numer fabryczny, numer uni-

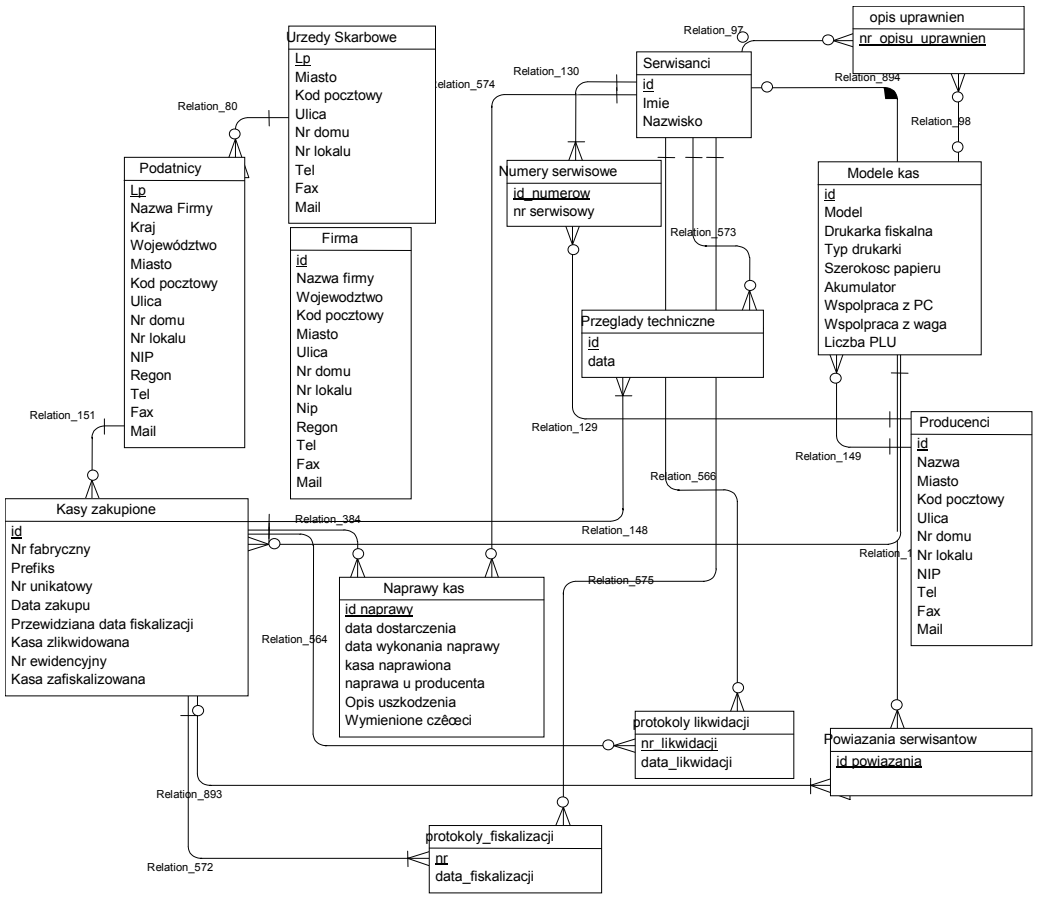
katowy, data zakupu, typ i model kasy oraz przewidziana data fiskalizacji), dane urzędu skarbowego, do którego należy użytkownik kasy. System powinien prowadzić ewidencję terminów fiskalizacji oraz przeglądów technicznych. Przeglądów technicznych dokonują serwisanci danej kasy – klient ma prawo zmiany serwisanta. System powinien mieć możliwość prowadzenia ewidencji napraw oraz wydruku protokołów przyjęć do naprawy, przekazania do producenta i wydania sprzętu po naprawie.

Osobnym zadaniem serwisu jest doradztwo w kwestii wyboru przez klienta odpowiedniego typu kasy fiskalnej, w zależności od rzeczywistych potrzeb. Urządzenia fiskalne dzielą się na kilka typów:

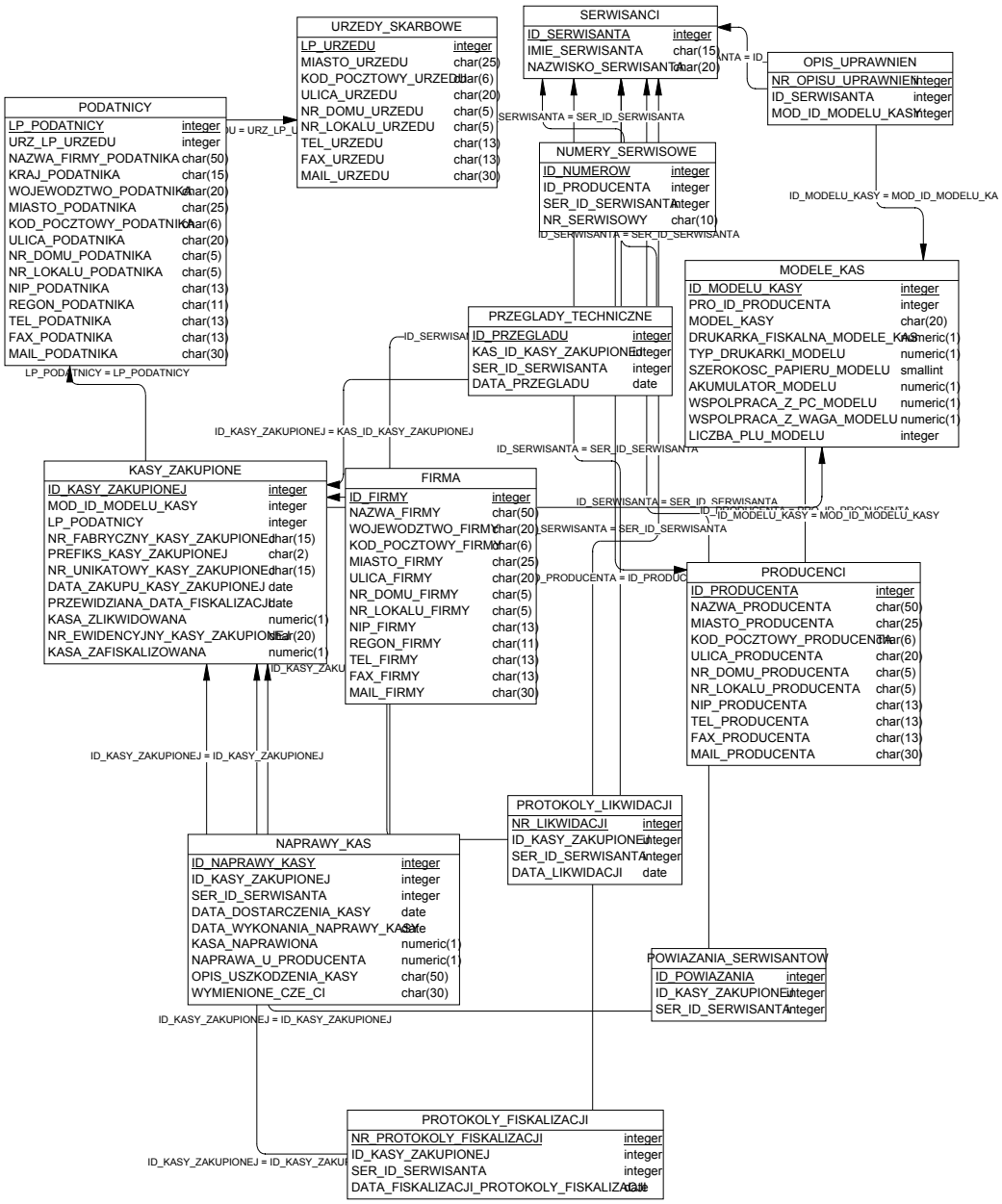
– kasy przenośne,
– kasy jednostanowiskowe,
– kasy systemowe,
– drukarki fiskalne (urządzenia współpracujące z komputerem).

Przytoczony dokładny opis obszaru modelowania ma na celu uzmysłowienie, że – ze względu na obowiązujące zasady i przepisy – system serwisowania kas fiskalnych musi się różnić od standardowych systemów wspomagających zarządzanie małymi firmami.

Model danych (diagram E-R) dla omawianego przykładu jest przedstawiony na rys. 7.28, odpowiadający mu zaś model relacyjny na rys. 7.29.



Rys. 7.28. Model CDM dla Systemu Serwisowania Kas Fiskalnych



Rys. 7.29. Model relacyjny dla Systemu Serwisowania Kas Fiskalnych

Model danych dla *przykładu 6.4* – Systemu Zarządzania Magazynem

Na diagramie DFD dla systemu występuje siedem magazynów. Ogólna charakterystyka struktury magazynów, zgodnie z wcześniejszą analizą, sprowadza się do następujących relacji:

1. **Stany magazynowe**
Encje składowe: *Stany Magazynowe*, *Towar*, *Dostawca*.
Związki składowe: *Ilość* (Stany Magazynowe, Towar), *Kto dostarczył* (Towar, Dostawca).

2. **Dane Statystyczne**
Encje składowe: *Dane Statystyczne*, *Operacje*, *Producent*, *Dostawca*.
Związki składowe: *Typ operacji* (Dane Statystyczne, Operacje), *Wywołujący operację* (Dane Statystyczne, Producent), *Właściciel jednostki* (Producent, Dostawca).

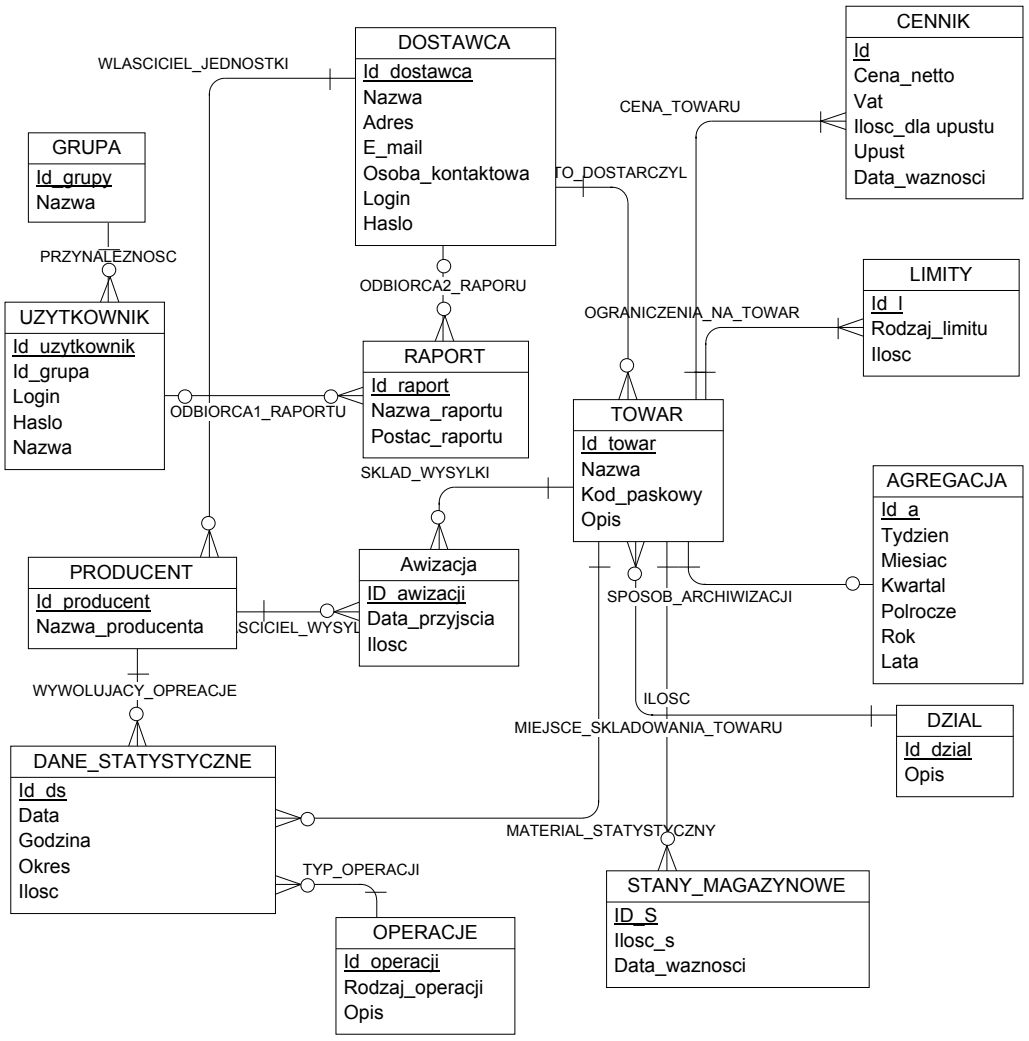
3. **Limity**
Encje składowe: *Limity*, *Towar*, *Dostawca*, *Dział*.
Związki składowe: *Ograniczenia na towar* (Limity, Towar), *Miejsce składowania towaru* (Dział, Towar), *Dostarczyciel towaru* (Towar, Dostawca).

4. **Raporty**
Encje składowe: *Raport*, *Towar*, *Dostawca*, *Producent.*
Związki składowe: *Odbiorca raportu* (Raporty, Dostawca, Użytkownik), *Zawartość raportu* (Raporty, Towar), *Zakres raportu* (Raporty, Producent).

5. **Awizacja**
Encje składowe: *Towar*, *Dostawca*, *Producent.*
Związki składowe: *Właściciel wysyłki* (Awizacja, Producent), *Skład wysyłki* (Awizacja, Towar), *Właściciel jednostki* (Producent, Dostawca).

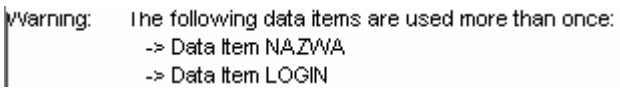
6. **Agregacja**
Encje składowe: *Agregacja*, *Towar*.
Związki składowe: *Sposób archiwizacji* (Agregacja, Towar).

7. **Cennik**
Encje składowe: *Cennik*, *Towar*.
Związki składowe: *Cena towaru* (Cennik, Towar).
Model danych dla Systemu Zarządzania Magazynem przedstawiono na rys. 7.30.

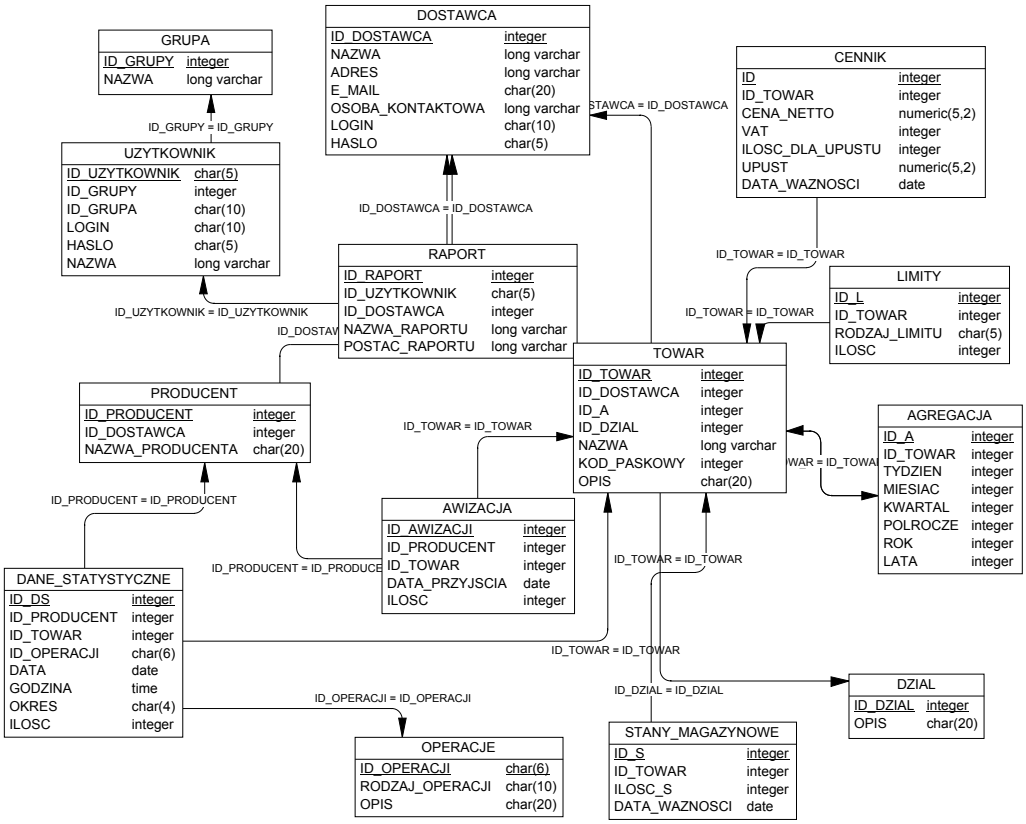


Rys. 7.30. Diagram ER dla Systemu Zarządzania Magazynem

Model konceptualny został sprawdzony pod względem poprawności formalnej – nie zawiera błędów, program DataArchitect wygenerował jedynie komunikaty ostrzegające, o treści:



ponieważ te same nazwy atrybutów pojawiły się w kilku encjach. Nie jest to jednak sytuacja błędna, nie ma więc przeszkód, aby wygenerowany został model PDM, który ilustruje rys. 7.31.



Rys. 7.31. Model relacyjny dla Systemu Zarządzania Magazynem

Kończąc rozdział dotyczący konstruowania modeli danych, warto zwrócić uwagę, że budowanie różnych typów modeli (diagram kontekstowy, DFD, ERD) umożliwia rozważenie systemu na różne sposoby i w różnych aspektach, ponieważ nie są to działania rozłączne. Często jeden typ modelu pozwala dostrzec zależności nie uchwycone w innym, model danych może spowodować konieczność zmian w modelu funkcjonalnym systemu i na odwrót.

8 Normalizacja

Proces normalizacji jest formalną techniką pozwalającą na uzyskanie modelu danych (relacyjnego), którego elementy będą spełniały określone wymogi, wynikające z definicji postaci normalnych, wprowadzonych przez Codda. W rozdziale 7 niniejszego podręcznika przedstawiono tzw. zstępującą (*top-down*) metodologię projektowania baz danych, polegającą na wyabstrahowaniu głównych encji i zależności pomiędzy nimi. Zawsze jednak stosując tę metodologię, należy przeprowadzić proces walidacji modelu, do czego służą techniki normalizacyjne [5, 6, 9]. Niezależnie od wspomagania procesu modelowania danych, normalizacja może być wykorzystywana jako samodzielna metodologia projektowania relacyjnej bazy danych. Podejście takie jest nazywane metodologią wstępującą (*bottom-up*), ponieważ pierwszym krokiem jest zgromadzenie wszystkich elementów danych, które mają być umieszczone w bazie danych, a następnie na podstawie analizy zależności pomiędzy elementami danych projektowany jest schemat bazy.

Podstawą normalizacji jest zidentyfikowanie związków logicznych pomiędzy elementami danych, zwanych zależnościami funkcyjnymi lub inaczej – związkami determinowania. W odniesieniu do tabel relacyjnej bazy danych identyfikowane zależności dotyczą kluczy głównych tabel i kolumn niekluczowych. Może się pojawić pytanie, czy konieczne jest przestrzeganie wymogów postaci normalnych? Okazuje się, że jeżeli tabele wchodzące w skład bazy danych nie spełniają takich wymogów, tzn. schemat bazy danych jest niepoprawny, to w określonych sytuacjach pojawią się nieprawidłowości uniemożliwiające właściwą pracę z bazą danych lub wręcz dyskwalifikujące produkt. Nieprawidłowości te – ogólnie mówiąc – sprowadzają się do czterech zasadniczych przypadków:

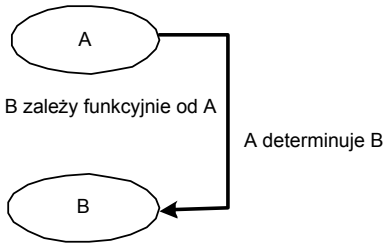
- redundancji danych (powtarzanie się elementów lub grup danych),
- anomalii przy modyfikacji danych,
- anomalii przy wstawianiu danych,
- anomalii przy usuwaniu danych.

Proces normalizacji ma na celu usunięcie takich nieprawidłowości, dobrze zaprojektowany schemat bazy danych jest bowiem gwarancją prawidłowego jej działania.

Jak już wspomniano, proces normalizacji opiera się na ustalaniu zależności funkcyjnych pomiędzy atrybutami w relacji. Definicja zależności funkcyjnej przedstawia się następująco:

Załóżmy, że A i B są atrybutami relacji R. Atrybut B jest funkcjonalnie zależny od atrybutu A (A→B), jeżeli każda wartość A jest związana z dokładnie jedną wartością B. Inaczej mówiąc, A determinuje B, czyli B jest funkcyjnie zależne od A.

Zależności funkcyjne mogą występować nie tylko pomiędzy pojedynczymi atrybutami, lecz także pomiędzy grupami atrybutów. Często zależności funkcyjne przedstawia się w formie graficznej, za pomocą prostych diagramów (rys. 8.1).



Rys. 8.1. Diagram zależności funkcyjnych

Rozważając zależności funkcyjne pomiędzy atrybutami (grupami atrybutów), należy zwrócić uwagę, że zależności te nie są związane z wartościami, które w danym czasie przyjmują atrybuty, lecz są własnością schematu relacji, obowiązującą dla zbioru wszystkich możliwych wartości atrybutów.

W przypadku analizy zbioru danych pod kątem budowania schematu relacji, wyodrębnienie zależności funkcyjnych związane jest z identyfikacją zależności pomiędzy kluczami głównymi (ewentualnie kluczami kandydującymi) i atrybutami niekluczowymi. Oznacza to przyjęcie założenia, że każda relacja musi mieć desygnowany klucz główny. Proces normalizacji zbioru danych przebiega w kilku formalnie wyodrębnionych etapach; po każdym z etapów relacja przyjmuje tak zwaną postać normalną (*Normal Form*); począwszy od pierwszej postaci normalnej (*First Normal Form*, *1NF*), do piątej postaci normalnej (*Fifth Normal Form*, *5NF*). Przyjęcie przez relację kolejnej postaci normalnej jest możliwe po spełnieniu kryteriów wcześniejszych postaci. W praktyce, w typowych projektach baz danych, przyjmuje się, że trzecia postać normalna (*3NF*) jest wystarczająca dla zapewnienia poprawności schematu bazy danych.

W wielu podręcznikach z zakresu relacyjnych baz danych proces normalizacji jest przedstawiany w sposób sformalizowany matematycznie. Wydaje się jednak, że dla zrozumienia reguł normalizacyjnych oraz ich zastosowania w metodzie wstępującej projektowania schematu bazy danych wystarczające jest omówienie na przykładzie praktycznym idei normalizacji z wykorzystaniem metod graficznych, czyli diagramów zależności.

Zanim przejdziemy do ilustracji praktycznych całego procesu normalizacji, zdefiniujemy pojęcie tabeli nieznormalizowanej i relacji (tabeli) w pierwszej postaci normalnej.

Pierwsza postać normalna (*1NF*)

- Tabela nieznormalizowana, to tabela, która zawiera powtarzalne grupy danych, gdzie grupa powtarzalna oznacza wystąpienie w wierszu kolumn umożliwiających przechowywanie wielu wartości tego samego atrybutu.
- Tabela spełnia wymogi pierwszej postaci normalnej (*1NF*), jeżeli na przecięciu wierszy i kolumn znajdują się wartości elementarne (atomowe).

Przykład 8.1

Projektując schemat bazy danych metodą wstępującą (*bottom-up*), czyli wykorzystując techniki normalizacji, należy zacząć od zgromadzenia zbioru danych, który będzie przechowywany w relacyjnej bazie danych. Rozważany wycinek rzeczywistości związany jest z ubezpieczeniami domów i mieszkań. Przykład dotyczy hipotetycznego towarzystwa ubezpieczeniowego. Uwzględniono w nim najistotniejsze dane znajdujące się na polisach ubezpieczeniowych, pomijając szczegóły nieistotne z punktu widzenia metodologii.

Towarzystwo Ubezpieczeń Nieruchomości oferuje klientom (osobom fizycznym) kompleksowe ubezpieczenie nieruchomości, które może obejmować:

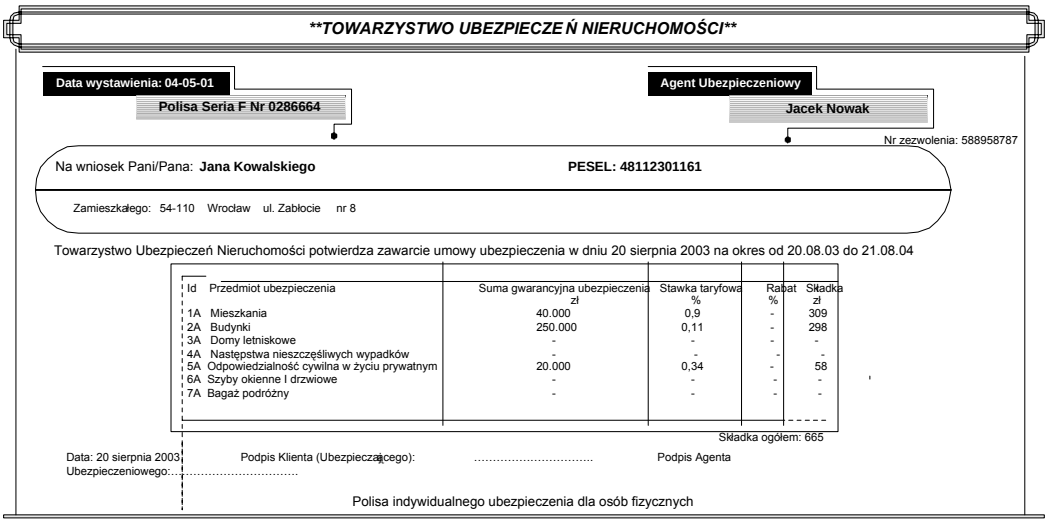
- **mieszkania**, czyli ruchomości domowe znajdujące się w mieszkaniu, piwnicy, garażu – od kradzieży, ognia, zalania i innych zdarzeń losowych;
- **budynki i lokale mieszkalne** – od ognia, powodzi i innych żywiołów;
- **domy letniskowe** – od ognia, powodzi i innych żywiołów;
- **następstwa nieszczęśliwych wypadków**;
- **odpowiedzialność cywilną ubezpieczającego** w życiu prywatnym;
- **bagaż podróżny**, czyli rzeczy przenoszone lub przewożone przez ubezpieczającego;
- **szyby**, czyli szyby okienne i drzwiowe w budynkach mieszkalnych.

Sumę ubezpieczenia, odrębną dla każdej grupy mienia, następstwa nieszczęśliwych wypadków oraz sumę gwarancyjną dla odpowiedzialności cywilnej określa ubezpieczający w porozumieniu z Towarzystwem Ubezpieczeń Nieruchomości, reprezentowanym przez agenta wystawiającego polisę. W ubezpieczeniach od nieszczęśliwych wypadków kwota ubezpieczenia dla każdej osoby musi się zawierać w ustalonych przez Towarzystwo granicach, w przypadku odpowiedzialności cywilnej ustalana jest natomiast górna granica sumy gwarancyjnej.

Z tytułu zawartej umowy Towarzystwo Ubezpieczeń Nieruchomości wypłaca należne odszkodowanie w kwocie odpowiadającej wysokości szkody, nie większej jednak niż kwota ubezpieczenia określona dla poszczególnych ubezpieczeń.

Odpowiedzialność Towarzystwa liczy się od dnia wystawienia polisy i trwa jeden rok.

Polisy ubezpieczeniowe są wystawiane na standardowych drukach, podpisywanych przez klienta i przez agenta ubezpieczeniowego. Pojedynczą (uproszczoną) polisę przedstawiono na rys. 8.2. Towarzystwo Ubezpieczeń Nieruchomości dotychczas przechowywało kolekcję polis w segregatorach, ponieważ jednak rynek ubezpieczeń rozwija się dynamicznie, postanowiono wdrożyć system z bazą danych umożliwiający automatyzację prac związanych z wystawianiem polis, obliczaniem należnych odszkodowań, uaktualnianiem danych klientów Towarzystwa, jak również pozyskiwaniem nowych klientów, którzy jeszcze nie podjęli decyzji o podpisaniu umowy (polisy).



****TOWARZYSTWO UBEZPIECZEŃ NIERUCHOMOŚCI****

Data wystawienia: 04-05-01

Polisa Seria F Nr 0286664

Agent Ubezpieczeniowy

Jacek Nowak

Nr zezwolenia: 588958787

Na wniosek Pani/Pana: **Jana Kowalskiego** **PESEL: 48112301161**

Zamieszkałego: 54-110 Wrocław ul. Zabłocie nr 8

Towarzystwo Ubezpieczeń Nieruchomości potwierdza zawarcie umowy ubezpieczenia w dniu 20 sierpnia 2003 na okres od 20.08.03 do 21.08.04

Id	Przedmiot ubezpieczenia	Suma gwarancyjna ubezpieczenia zł	Stawka taryfowa %	Rabat %	Składka zł
1A	Mieszkania	40.000	0,9	-	309
2A	Budynki	250.000	0,11	-	298
3A	Domy letniskowe	-	-	-	-
4A	Następstwa nieszczęśliwych wypadków	-	-	-	-
5A	Odpowiedzialność cywilna w życiu prywatnym	20.000	0,34	-	58
6A	Szyby okienne i drzwiowe	-	-	-	-
7A	Bagaż podróżny	-	-	-	-

Składka ogółem: 665

Data: 20 sierpnia 2003 Podpis Klienta (Ubezpieczającego): ..... Podpis Agenta Ubezpieczeniowego: .....

Polisa indywidualnego ubezpieczenia dla osób fizycznych

Rys. 8.2. Uproszczony wzór polisy ubezpieczeniowej

Dane z przykładowych polis przetransferowane do tabeli nieznormalizowanej zawiera tabela przedstawiona na rys. 8.3.

Nr polisy	*Seria*	*Data*	*PESEL*	*Nazwisko_k*	*Imię_k*	*Kod*	*Miasto*	*Ulica*	*Nr_d*	*Nr_m*	*Id_pu*	*Opis*	*Suma_gwar*	*Stawka_t*	*Rabat*	*Składka*	*Nr_zezw*	*Nazwisko_a*	*Imię_a*
0286664	F	20-08-03	48112301161	Kowalski	Jan	54-110	Wroclaw	Zabłocie	8	-	1A 2A 5A	Mieszkania Budynki OC	40.000 250.000 20.000	0.9 0.11 0.34	- - -	309 298 20	588958787	Nowak	Jacek
0286670	F	21-08-03	50100802622	Adamska	Ewa	51-620	Legnica	Magnolii	11	2	1A 3A 7A	Mieszkania Domy letn. Bagaż podr	35.000 50.000 25.000	0.8 0.9 0.12	- - -	280 397 150	588958787	Nowak	Jacek
0289000	G	17-07-03	68052801131	Kowalski	Jan	52-240	Wrodaw	Piękna	30	-	2A 6A	Budynki Szyby	500.000 70.000	0.10 0.8	10 -	600 250	455789011	Malina	Kamil
0289060	G	19-07-03	68052801131	Kowalski	Jan	52-240	Wroclaw	Piękna	30	-	1A 4A	Mieszkania NW	70.000 30.000	0.9 0.1	10	396 154	455789011	Malina	Kamil
0289065	G	20-07-03	55120613134	Mirska	Anna	54-123	Wrodaw	Orla	17	-	1A 2A	Mieszkania Budynki	45.000 300.000	0.9 0.11	-	350 311	455789011	Malina	Kamil

Rys. 8.3. Nieznormalizowana tabela POLISY

Łatwo zauważyć, że w nieznormalizowanej tabeli POLISY kolumny Id_pu, Opis, Suma_gwar, Stawka_t, Rabat, Składka zawierają grupy powtarzalne danych (na przecięciu kolumny i wiersza nie występują wartości atomowe). Jednym z algorytmów

umożliwiających przekształcenie nieznormalizowanego zbioru danych w tabelę spełniającą wymogi pierwszej postaci normalnej jest usunięcie grup powtarzalnych poprzez wprowadzenie odpowiednich danych do pustych kolumn, czyli duplikowanie danych w wierszach zawierających grupy powtarzalne (rys. 8.4). Otrzymana w taki sposób dwuwymiarowa tabela zawiera na każdym przecięciu wiersza i kolumny wartości atomowe, czyli spełnia wymagania pierwszej postaci normalnej (*1NF*). Tabela taka jest relacją, możliwe jest umieszczenie jej w relacyjnej bazie danych, ale na pierwszy rzut oka widać dużą redundancję danych, jak również pozostałe wady wymienione na początku rozdziału, czyli anomalie przy modyfikacji danych, zarówno przy wstawianiu danych do tabeli, jak i ich usuwaniu.

Nr polisy	*Seria*	*Data*	*PESEL*	*Nazwisko_k*	*Imię_k*	*Kod*	*Miasto*	*Ulica*	*Nr_d*	*Nr_m*	*Id_pu*	*Opis*	*Suma_gwar*	*Stawka_t*	*Rabat*	*Składka*	*Nr_zezw*	*Nazwisko_a*
0286664	F	20-08-03	48112301161	Kowalski	Jan	54-110	Wroclaw	Zabłocie	8	-	1A	Mieszkania	40.000	0.9	-	309	588958787	Nowak
0286664	F	20-08-03	48112301161	Kowalski	Jan	54-110	Wroclaw	Zabłocie	8	-	2A	Budynki	250.000	0.11	-	298	588958787	Nowak
0286664	F	20-08-03	48112301161	Kowalski	Jan	54-110	Wroclaw	Zabłocie	8	-	5A	OC	20.000	0.34	-	20	588958787	Nowak
0286670	F	21-08-03	50100802622	Adamska	Ewa	51-620	Legnica	Magnolii	11	2	1A	Mieszkania	35.000	0.8	-	280	588958787	Nowak
0286670	F	21-08-03	50100802622	Adamska	Ewa	51-620	Legnica	Magnolii	11	2	3A	Domy letn.	50.000	0.9	-	397	588958787	Nowak
0286670	F	21-08-03	50100802622	Adamska	Ewa	51-620	Legnica	Magnolii	11	2	7A	Bagaż podr	25.000	0.12	-	150	588958787	Nowak
0289000	G	17-07-03	68052801131	Kowalski	Jan	52-240	Wrodaw	Piękna	30	-	2A	Budynki	500.000	0.10	10	600	455789011	Malina
2890000	G	17-07-03	68052801131	Kowalski	Jan	52-240	Wrodaw	Piękna	30	-	6A	Szyby	70.000	0.8	-	250	455789011	Malina
0289060	G	19-07-03	68052801131	Kowalski	Jan	52-240	Wroclaw	Piękna	30	-	1A	Mieszkania	70.000	0.9	10	396	455789011	Malina
0289060	G	19-07-03	68052801131	Kowalski	Jan	52-240	Wroclaw	Piękna	30	-	4A	NW	30.000	0.1		154	455789011	Malina
0289065	G	20-07-03	55120613134	Mirska	Anna	54-123	Wrodaw	Orla	17	-	1A	Mieszkania	45.000	0.9		350	455789011	Malina
0289065	G	20-07-03	55120613134	Mirska	Anna	54-123	Wrodaw	Orla	17	-	2A	Budynki	300.000	0.11		311	455789011	Malina

Rys. 8.4. Tabela POLISY w pierwszej postaci normalnej

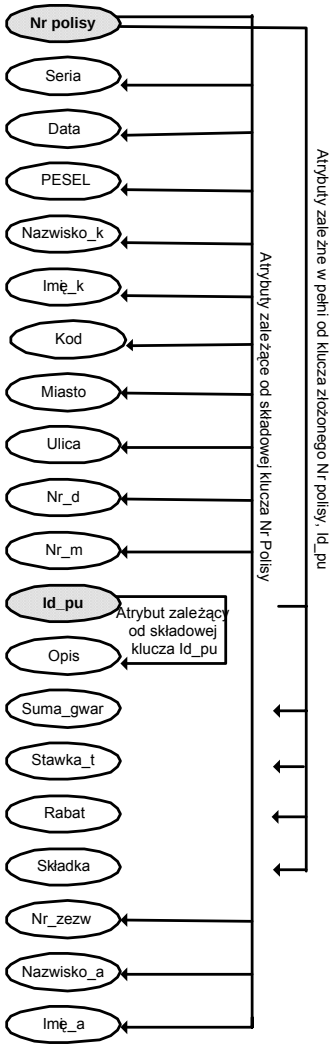
W omawianym przykładzie usunięcie danych klienta, który wykupił polisę, spowoduje usunięcie danych dotyczących polisy. W razie zmiany danych, np. adresu klienta, aktualizowane muszą być wszystkie wiersze zawierające taki adres, co może łatwo doprowadzić do sytuacji, w której klient będzie figurował zarówno ze starym, jak i nowym adresem, czyli baza danych stanie się niespójna. Wstawianie danych może również sprawiać problemy: nie jest możliwe ani prowadzenie rejestru potencjalnych klientów, ani agentów ubezpieczeniowych, którzy nie sprzedali jeszcze żadnej polisy, czy poszerzenia oferty możliwych przedmiotów ubezpieczenia. Koniecznym jest więc doprowadzenie relacji do kolejnej postaci normalnej poprzez podział struktury danych (rozkład odwracalny) na większą liczbę tabel, ale z zachowaniem związków między elementami danych, czyli z zachowaniem możliwości odwrócenia rozkładu.

Druga postać normalna (*2NF*)

- Relacja, która jest w pierwszej postaci normalnej i w której każdy atrybut niekluczowy jest w pełni funkcyjnie zależny od klucza głównego spełnia wymogi drugiej postaci normalnej.
- Należy zauważyć, że zacytowana definicja dotyczy tabel, które mają złożone klucze główne, w tym przypadku bowiem może się zdarzyć taka sytuacja, że niektóre kolumny będą zależeć od całego klucza, inne zaś od jego składowych. Przekształcenie

z pierwszej postaci normalnej do drugiej postaci normalnej będzie polegać na usunięciu częściowych zależności funkcyjnych, czyli na takim podziale struktury danych, aby każdy z atrybutów niekluczowych zależał w pełni od klucza głównego.

Kontynuując przykład 8, w odniesieniu do tabeli POLISY, należy określić klucz główny, a następnie zbadać zależności atrybutów niekluczowych od klucza. Kluczem głównym dla tej tabeli jest klucz złożony z dwóch kolumn: **Nr polisy** i **Id_pu**. Posiłkując się definicją zależności funkcyjnych, dla tabeli POLISY skonstruowano diagram zależności funkcyjnych (rys. 8.5).



Rys. 8.5. Diagram zależności funkcyjnych dla tabeli POLISY

Analizując diagram zależności funkcyjnych łatwo zauważyć, że tabela POLISY nie jest w drugiej postaci normalnej, ponieważ występują w niej zarówno zależności funkcyjne pomiędzy całym złożonym kluczem głównym a atrybutami niekluczowymi, jak i zależności pomiędzy składowymi klucza głównego a atrybutami niekluczowymi. Przykładowo, kolumna *Opis* jest zależna funkcyjnie tylko od jednej składowej klucza głównego *Id_pu*, a nie od obu. Przekształcenie tabeli POLISY do drugiej postaci normalnej wymaga jej rozdzielenia na osobne tabele zawierające elementy determinujące (składowe klucza głównego i cały klucz główny) oraz elementy zależne od nich. W omawianym przykładzie otrzymujemy więc tabele przedstawione na rys. 8.6.

POLISY

Nr polisy	Seria	Data	PESEL	Nazwisko_k	Imię_k	Kod	Miasto	Ulica	Nr_d	Nr_m	Nr_zezw	Nazwisko_a	Imię_a
028664	F	20-08-03	48112301161	Kowalski	Jan	54-110	Wrocław	Zabłocie	8	-	588958787	Nowak	Jacek
0286670	F	21-08-03	50100802262	Adamska	Ewa	51-620	Legnica	Magnolii	11	2	588958787	Nowak	Jacek
0289000	G	17-07-03	68052801131	Kowalski	Jan	52-240	Wrocław	Piękna	30	-	456789011	Malina	Kamil
0289060	G	19-07-03	68052801131	Kowalski	Jan	52-240	Wrocław	Piękna	30	-	456789011	Malina	Kamil
0289065	G	20-07-03	56120613134	Mirska	Anna	54-123	Wrocław	Orla	17	-	456789011	Malina	Kamil

ZAKRES UBEZPIECZENIA

Nr polisy	Id_pu	Suma_gwar	Stawka_t	Rabat	Składka
028664	1A	40.000	0.9	-	309
028664	2A	250.000	0.11	-	298
028664	5A	20.000	0.34	-	20
...	...	...	...	...	...
0289065	1A	45.000	0.9	-	350
0289065	2A	300.000	0.12	-	311

PRZEDMIOT UBEZPIECZENIA

Id_pu	Opis
1A	Mieszkania
2A	Budynki
3A	Domy letniskowe
4A	Nieszczęśliwe wypadki
5A	Odpowiedzialność cywilna
6A	Szyby okienne I drzwiowe
7A	Bagaż podróżny

Rys. 8.6. Druga postać normalna po przekształceniu tabeli POLISY

W omawianym przykładzie tabele ZAKRES UBEZPIECZENIA i PRZEDMIOT UBEZPIECZENIA są całkowicie wolne od redundancji danych, wszystkie kolumny nie będące kluczami są w pełni zależne od klucza głównego, tabele te nie muszą więc być dalej przekształcane. Wątpliwości co do prawidłowości struktury nasuwają się po analizie tabeli POLISY. W odniesieniu do tej tabeli można wyraźnie stwierdzić, że ilekroć ten sam agent wystawi polisę, należy powtórzyć jego dane, podobnie w przypadku kilkukrotnego wykupywania polisy przez tego samego klienta jego dane muszą być powtarzane. Sytuacja taka związana jest z istniejącymi w obrębie relacji tzw. zależnościami przechodnimi (tranzytywnymi). Zależność przechodnią można zdefiniować następująco:

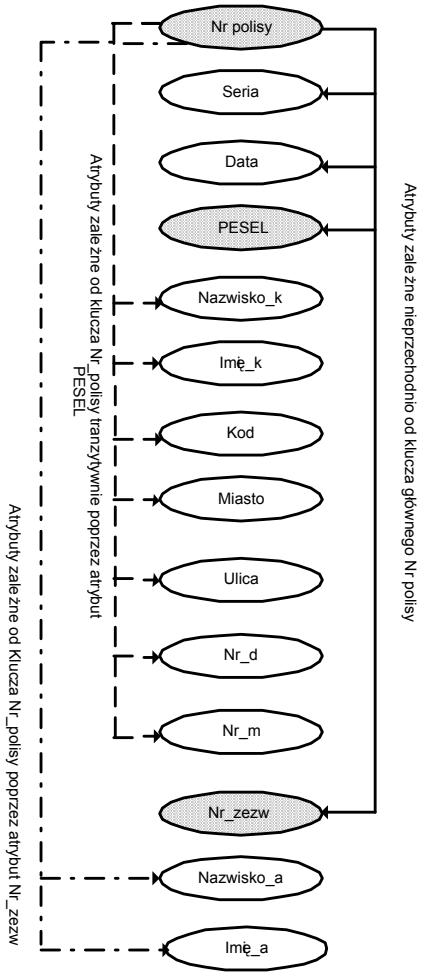
Gdy A, B, C są atrybutami relacji R, w której zachodzą zależności: $A \rightarrow B$ i $B \rightarrow C$, wtedy C zależy przechodnio (tranzytywnie) od A poprzez B.

Pojęcie zależności przechodniej związane jest z trzecią postacią normalną relacji.

Trzecia postać normalna (*3NF*)

• Relacja, która jest w drugiej postaci normalnej i której każdy atrybut niekluczowy zależy od klucza bezpośrednio, a nie tranzytywnie, spełnia wymogi trzeciej postaci normalnej.

Wyniki analizy tabeli POLISY pod kątem zależności funkcyjnych i tranzytywnych obrazuje diagram przedstawiony na rys. 8.7.



Rys. 8.7. Diagram zależności funkcyjnych i tranzytywnych w relacji POLISY

Diagram wskazuje zależności między kolumną klucza głównego *Nr_polisy*, od którego funkcjonalnie zależą kolumny niekluczowe: *Seria*, *Data*, *PESEL*, *Nr_zezw*.

Pozostałe zależności są zależnościami tranzytywnymi, i tak kolumny: *Nazwisko_k*, *Imię_k*, *Kod*, *Miasto*, *Ulica*, *Nr_d*, *Nr_m* zależą od klucza głównego przechodnio poprzez kolumnę *PESEL*, natomiast kolumny: *Nazwisko_a*, *Imię_a* zależą od klucza poprzez kolumnę *Nr_zezw*. Zgodnie z definicją trzeciej postaci normalnej tabela POLISY musi zostać rozdzielona na osobne tabele, w których będą występować tylko zależności funkcjonalne kolumn niekluczowych od klucza głównego. W tabelach tych rolę kluczy głównych pełnić będą odpowiednio *Nr_polisy*, *PESEL* oraz *Nr_zezw*. (rys. 8.8).

POLISY

Nr polisy	Seria	Data	PESEL	Nr_zezw
028664	F	20-08-03	48112301161	588958787
0286670	F	21-08-03	50100802262	588958787
0289000	G	17-07-03	68052801131	456789011
0289060	G	19-07-03	68052801131	456789011
0289065	G	20-07-03	56120613134	456789011

AGENCI

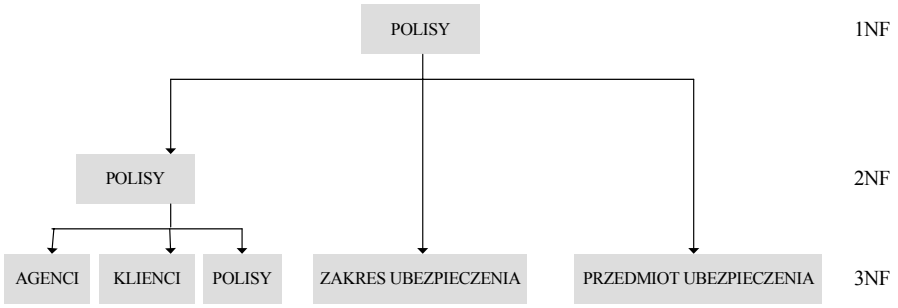
Nr_zezw	Nazwisko_a	Imię_a
588958787	Nowak	Jacek
456789011	Malina	Kamil

KLIENCI

PESEL	Nazwisko_k	Imię_k	Kod	Miasto	Ulica	Nr_d	Nr_m
48112301161	Kowalski	Jan	54-110	Wrocław	Zabłocie	8	
50100802262	Adamska	Ewa	51-620	Legnica	Magnolii	11	2
68052801131	Kowalski	Jan	52-240	Wrocław	Piękna	30	
56120613134	Mirska	Anna	54-123	Wrocław	Orla	17	

Rys. 8.8. Tabele w trzeciej postaci normalnej otrzymane po przekształceniu tabeli POLISY

Schematy tabel spełniające wymogi trzeciej postaci normalnej, wolne od redundancji danych i anomalii aktualizacyjnych tworzą poprawny i wystarczający z praktycznego punktu widzenia schemat bazy danych. Proces dekompozycji tabeli POLISY (poprzez wykonanie szeregu operacji rzutowania w sensie algebry relacji) do tabel spełniających wymogi trzeciej postaci normalnej ilustruje diagram przedstawiony na rys. 8.9.



Rys. 8.9. Dekompozycja relacji POLISY od pierwszej do trzeciej postaci normalnej

Zauważmy, że wejściowa relacja POLISY może zostać odtworzona poprzez operacje złączenia, z wykorzystaniem mechanizmu klucz główny/klucz obcy.

Wychodząc od jednej relacji, poprzez ciąg rzutów, zaprojektowano poprawny schemat bazy danych. Warto mieć jednak świadomość, że normalizacja jako samodzielna metoda projektowania bazy danych może w rzeczywistości okazać się trudna i czasochłonna, zwłaszcza w przypadku dużych zbiorów danych, chociażby z powodu konieczności pełnego określenia całego zbioru danych, który będzie przechowywany w bazie danych. Dlatego też najczęściej normalizacja używana jest jako metoda sprawdzania poprawności modelu danych uzyskanego metodą zstępującą, czyli diagramu ER.

Uwaga: W licznych pozycjach literaturowych z zakresu teorii relacyjnych baz danych omawiana jest szczegółowo rozszerzona postać normalna, tzw. postać normalna Boyce'a–Codda (*BCNF, Boyce–Codd Normal Form*), która narzuca dodatkowe ograniczenia, sformułowane w postaci warunku, że **każda** kolumna, od której zależy **jakakolwiek** inna koluma musi być **kluczem unikalnym** (czyli kluczem potencjalnym relacji). Warunek ten jest mocniejszym ograniczeniem niż reguły omówione powyżej, niemniej jednak nie stoi w sprzeczności z klasyczną definicją trzeciej postaci normalnej. Relacja, która jest postaci normalnej BCNF, zawsze jest w trzeciej postaci normalnej, chociaż odwrotna sytuacja nie zawsze jest prawdziwa. Decyzje, czy lepiej pozostawić tabele w trzeciej postaci normalnej, czy dążyć do spełnienia wymogów postaci normalnej BCNF zależą od wielkości redundancji danych, jaka w niektórych przypadkach może zaistnieć, niemniej jednak w praktyce obowiązującym kryterium dla schematów tabel jest spełnienie wymogów klasycznej trzeciej postaci normalnej. Należy pamiętać, że zbyt daleko posunięta dekompozycja tabel może spowodować spadek wydajności systemu ze względu na konieczność wykonywania operacji złączeń.

Czwarta postać normalna (*4NF*)

Czwarta postać normalna związana jest z innym typem zależności między atrybutami relacji, a mianowicie z zależnościami wielowartościowymi. Zależności wielowartościowe (niefunkcyjne) można zdefiniować następująco:

Zależność wielowartościowa między atrybutami A, B i C relacji R zachodzi wtedy, gdy dla każdej wartości A istnieje zbiór odpowiadających wartości B (A→>B) i zbiór wartości C (A→>C). Zbiory wartości B i C są od siebie niezależne.

Zależności wielowartościowe w praktyce występują dość rzadko, tabele z przykładu 8.1 takich zależności nie zawierają, dlatego też posłużymy się prostym przykładem ilustracyjnym, na którym wyjaśnione zostaną reguły czwartej postaci normalnej, której definicja brzmi:

- Tabela spełnia wymogi czwartej postaci normalnej, jeżeli nie zawiera wielu zależności wielowartościowych, a jedynie pojedynczą.

Przykład 8.2

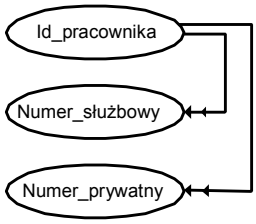
W bazie danych mają być przechowywane dane dotyczące pracowników oraz możliwości kontaktu telefonicznego z każdym z pracowników. Przez kontakt telefoniczny rozumiany jest zestaw służbowych numerów telefonicznych (nie można wykluczyć, że będzie ich kilka, z racji chociażby pracy w kilku firmach) oraz zestaw numerów prywatnych (zarówno stacjonarnych, jak i komórkowych). W przypadku zebrania takich danych w jednej tabeli, dla zachowania spójności informacji, każdy wiersz musiałby zawierać zapis kombinacji wszystkich możliwych numerów telefonów związanych z danym pracownikiem. Rysunek 8.10 przedstawia taką relację dla sytuacji, gdy pierwszy z pracowników o numerze identyfikacyjnym 0122447 jest dostępny pod dwoma numerami służbowymi i dwoma prywatnymi, drugi pracownik o numerze identyfikacyjnym 0230112 jest dostępny pod jednym numerem służbowym i dwoma prywatnymi, trzeci zaś o numerze identyfikacyjnym 0340224 – pod dwoma służbowymi i jednym prywatnym.

KONTAKTY TELEFONICZNE

Nazwisko	Id_pracownika	Nr_służbowy	Nr_prywatny
	0122447	327-27-07	348-42-56
	0122447	327-28-08	348-42-56
	0122447	327-27-07	601-281-449
	0122447	327-28-08	601-281-449
	0230112	330-30-31	751-35-40
	0230112	330-30-31	600-444-644
	0340224	350-58-50	607-337-670
	0340224	350-58-52	607-337-670

Rys. 8.10. Relacja KONTAKTY TELEFONICZNE

Diagram zależności dla relacji KONTAKTY TELEFONICZNE przedstawiono na rys. 8.11.



Rys. 8.11. Zależności wielowartościowe
występujące w relacji KONTAKTY TELEFONICZNE

Spełnienie reguł czwartej postaci normalnej wymaga dekompozycji relacji na dwie tabele: KONTAKTY SŁUŻBOWE i KONTAKTY PRYWATNE, zawierające pojedyncze zależności wielowartościowe:

KONTAKTY SŁUŻBOWE

Id_pracownika	Nr_służbowy
0122447	327-27-07
0122447	327-28-08
0230112	330-30-31
0340224	350-58-50
0340224	350-58-52

KONTAKTY PRYWATNE

Id_pracownika	Nr_prywatny
0122447	348-42-56
0122447	601-281-449
0230112	751-35-40
0230112	600-444-644
0340224	607-337-670

Piąta postać normalna (*5NF*)

Piąta postać normalna dotyczy tzw. zależności złączeniowych (połączeniowych), które są uogólnieniem zależności wielowartościowych. Dotyczy relacji, w których występuje więcej niż dwie zależności wielowartościowe. W rzeczywistości sytuacje wymagające zastosowania procedury normalizacyjnej zapewniającej piątą postać normalną nie występują prawie wcale, niemniej jednak postać ta prezentowana jest w literaturze przedmiotu. Ogólna definicja mówi, że:

- Relacja jest w piątej postaci normalnej, jeżeli jest w czwartej postaci normalnej i nie występują w niej zależności połączeniowe, czyli inaczej mówiąc, jeżeli istnieje jej rozkład odwracalny na mniejsze tabele.

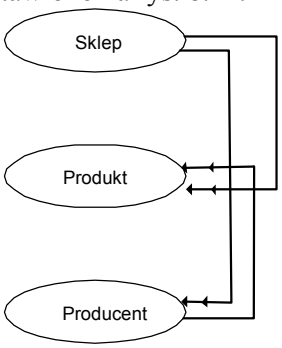
Przykład 8.3

Rozważmy relację pomiędzy producentami, produktami i sklepami sprzedającymi te produkty:

ASORTYMENT

Sklep	Produkt	Producent
ASTRA	Ser twarogowy	OSM Pątnica
WSS Południe	Śmietana kremowa	OSM Pątnica
WSS Południe	Ser homogenizowany	OSM Pątnica
WSS Poludnie	Ser twarogowy	OSM Łowicz
ASTRA	Śmietana kremowa	OSM Łowicz
ASTRA	Ser twarogowy	OSM Łowicz
PANDA	Śmietana kremowa	OSM Czarnków
ASTRA	Śmietana kremowa	OSM Czarnków

Zasady są takie, że sklepy zawierają kontrakty z różnymi producentami, producenci produkują podobny (często pokrywający się asortyment towarów), asortyment produktów dostępnych w sklepach pochodzi od różnych producentów. Diagram zależności dla tej relacji przedstawiono na rys. 8.12.



Rys. 8.12. Diagram zależności dla relacji ASORTYMENT

W omawianym przykładzie w jednej relacji występuje kilka zależności wielowartościowych; zgodnie z wymogami piątej postaci normalnej zależności te powinny być usunięte poprzez dekompozycję relacji – w omawianym przypadku na trzy oddzielne tabele (rys 8.13).

Sklep	Produkt
ASTRA	Ser twarogowy
WSS Południe	Śmietana kremowa
WSS Południe	Ser homogenizowany
WSS Poludnie	Ser twarogowy
ASTRA	Śmietana kremowa
PANDA	Śmietana kremowa

Sklep	Producent
ASTRA	OSM Pątnica
ASTRA	OSM Łowicz
ASTRA	OSM Czarnków
WSS Poludnie	OSM Łowicz
WSS Południe	OSM Pątnica
PANDA	OSM Czarnków

Producent	Produkt
OSM Pątnica	Ser twarogowy
OSM Pątnica	Śmietana kremowa
OSM Pątnica	Ser homogenizowany
OSM Łowicz	Ser twarogowy
OSM Łowicz	Śmietana kremowa
OSM Czarnków	Śmietana kremowa

Rys. 8.13. Piąta postać normalna relacji ASORTYMENT

W podsumowaniu tego rozdziału jeszcze raz należy podkreślić, że normalizacja jest procesem kreowania poprawnego schematu bazy danych, niemniej jednak mogą się zdarzyć sytuacje świadomego odejścia od zasad normalizacji, wynikające z wymogów funkcji realizowanych na danych. Zawsze jednak muszą to być przemyślane decyzje projektanta, wynikające najczęściej z analizy wydajności aplikacji. Najbezpieczniejszą ścieżką jest normalizacja bazy danych, a następnie zastosowanie częściowej denormalizacji, z wprowadzeniem ograniczeń i więzów zapewniających utrzymanie spójności bazy danych.

9

Projektowanie warstwy fizycznej relacyjnej bazy danych

Projektowanie fizyczne relacyjnej bazy danych jest trzecią i ostatnią fazą procesu projektowego. Wiąże się z decyzjami, w jaki sposób projekt logiczny ma zostać zaimplementowany w docelowym środowisku bazodanowym. Proces ten można zakwalifikować jako proces ze sprzężeniem zwrotnym – duża część decyzji implementacyjnych wynika z cech wybranego środowiska (Systemu Zarządzania Bazą Danych) i odwrotnie, na wybór środowiska mają wpływ wyniki analizy sposobu użycia danych, satysfakcjonujące przyszłego użytkownika parametry wydajnościowe, czy możliwości zapewnienia odpowiedniego bezpieczeństwa danych. Należy jednak pamiętać, że w każdym środowisku może istnieć kilka sposobów implementacji tego samego modelu, w związku z czym projektant powinien dysponować wiedzą o możliwościach funkcjonalnych oferowanych przez określony SZBD w zakresie tworzenia relacji, definiowania kluczy głównych i obcych, dziedzin czy też więzów integralności.

Głównym celem projektu fizycznego jest opracowanie szczegółowych wytycznych implementacyjnych dla projektu logicznego, czyli wybór optymalnej organizacji plików pamięci zewnętrznej, w których pamiętane będą obiekty bazy danych, dobór indeksów zapewniających efektywność korzystania z bazy danych, sposób implementacji więzów integralności oraz mechanizmów bezpieczeństwa danych.

Składowymi wejściowymi, stanowiącymi źródło informacji niezbędne dla prawidłowego przebiegu procesu projektowania fizycznego są:

- Projekt logiczny, niezależny od szczegółów implementacyjnych, tzn. od funkcjonalnych możliwości Systemu Zarządzania Bazą Danych i programów aplikacyjnych, składający się z diagramu ER, schematu relacyjnego oraz dokumentacji opisującej model, np. słownika danych.
- Sprecyzowane wymagania co do wydajności aplikacji.
- Oszacowania wielkości tabel.
- Oszacowania obciążenia bazy danych (częstotliwość dostępu do tabel, liczba wykonywanych operacji).

- Lista więzów integralności, które mają być zaimplementowane w bazie danych.
- Lista atrybutów wyliczanych, utworzona na podstawie porównania kosztów wyliczania atrybutów (czas wyliczania) z kosztem dodatkowej zajętości pamięci (denormalizacja).

Mówiąc o efektywności czy wydajności bazy danych, należy mieć świadomość, że będzie ona związana z charakterem aplikacji pracujących z bazą danych, ponieważ dla różnych systemów obowiązywać będą różne priorytety. Przykładowo, jeżeli mamy do czynienia z systemem rezerwacji biletów lotniczych, priorytetową sprawą będzie przepustowość systemu, czyli liczba transakcji, którą system może przetworzyć w określonym przedziale czasu. Z punktu widzenia użytkownika istotnym kryterium jest czas odpowiedzi systemu, rozumiany jako czas realizacji pojedynczej transakcji, ale z kolei na tak rozumiany czas odpowiedzi mają wpływ inne czynniki, będące poza kontrolą projektanta, np. obciążenie systemu. Innym kryterium może być wielkość pamięci zewnętrznej zajmowanej przez bazę danych – w określonych sytuacjach projektant może dążyć do minimalizacji zajętości dysku, ponieważ według oszacowań wstępnych czynnik ten może mieć wpływ na przepustowość systemu. W praktyce często po zaimplementowaniu wstępnej wersji projektu fizycznego system jest monitorowany i strojony w celu poprawy wydajności i efektywności działania.

Ogólny algorytm postępowania w projektowaniu fizycznym bazy danych sprowadza się do następujących kroków:

1. Translacja logicznego modelu danych dla docelowego środowiska bazodanowego, czyli: dobór nazw tabel, utworzenie listy atrybutów, określenie kluczy głównych, kandydujących i obcych, utworzenie listy atrybutów wyliczanych i sposobów ich wyliczenia, określenie więzów integralności referencyjnej i więzów dodatkowych, określenie dziedzin, ustalenie wartości domyślnych dla atrybutów, ustalenie, które z atrybutów mogą przyjmować wartości *null*. Sposób implementacji tego etapu jest ściśle uzależniony od wybranego Systemu Zarządzania Bazą Danych – w niniejszym podręczniku, w rozdziałach poprzednich przedstawione zostały możliwości implementacji w środowisku Sybase poprzez instrukcje DDL.
2. Zaprojektowanie fizycznej warstwy bazy danych, oparte na:
   a) analizie transakcji,
   b) wyborze organizacji plików bazy danych,
   c) wyborze indeksów,
   d) oszacowaniu wymaganego obszaru dyskowego.
3. Zaprojektowanie perspektyw.
4. Zaprojektowanie mechanizmów bezpieczeństwa.
5. Ewentualne wprowadzenie kontrolowanej redundancji danych.
6. Monitorowanie i strojenie systemu.

Realizacja drugiego kroku algorytmu projektowania warstwy fizycznej bazy danych wiąże się z koniecznością poznania sposobów pamiętania danych, czyli organizacją plików danych na dyskach.

9.1. Fizyczne przechowywanie danych – organizacja plików

Warstwa fizyczna bazy danych związana jest z pojęciami dotyczącymi sposobu przechowywania danych na komputerowych nośnikach pamięci – najczęściej dyskach, magnetycznych lub optycznych (tzw. pamięć pomocnicza). Baza danych w pamięci pomocniczej przechowywana jest w jednym lub kilku *plikach*, z których każdy składa się z *rekordów* (jeden lub więcej), a każdy rekord składa się z *pól* (jednego lub kilku). Każde z pól rekordu przechowuje jedną wartość atrybutu. Należy rozróżnić pojęcie *rekordu logicznego*, odpowiadającego w zasadzie wierszowi tabeli, od *rekordu fizycznego*, określanego inaczej *stroną* lub *blokiem pamięci*.

Rekord fizyczny (strona, blok) jest jednostką transferu pomiędzy dyskiem a pamięcią główną i na odwrót. Ogólnie mówiąc, strona pamięci zawiera w zależności od rozmiaru rekordów kilka rekordów logicznych, chociaż może się zdarzyć sytuacja, kiedy rekord logiczny zajmuje całą stronę pamięci. Każda strona pamięci posiada swój adres. W przypadku polecenia odczytu wierszy z określonej tabeli, SZBD lokalizuje żądane rekordy logiczne na stronach pamięci i kopiuje strony do bufora pamięci głównej. Następnie odpowiednie komponenty SZBD wyszukują w buforze wymagane rekordy. W przypadku polecenia zapisu zawartość bufora jest kopiowana na odpowiednią stronę pamięci. Operacje wyszukiwania i zapisywania danych wiążą się z wykonywaniem operacji we/wy, które są operacjami czasochłonnymi, dlatego też sposób organizacji plików przechowujących dane i związany z tym sposób dostępu do danych ma ogromne znaczenie i wpływ na efektywność bazy danych.

Jeżeli termin *organizacja pliku* zdefiniujemy jako sposób ułożenia danych w rekordach i w pliku oraz na stronach pamięci zewnętrznej, to *metody dostępu do danych* można określić jako czynności związane z pamiętaniem i wyszukiwaniem rekordów w pliku.

Jest rzeczą oczywistą, że określone metody dostępu związane są z określoną organizacją pliku (trudno mówić o dostępie indeksowanym w odniesieniu do pliku bez indeksu), dlatego też często termin *organizacja pliku* jest stosowany wymiennie z terminem *metoda dostępu do danych*.

Podstawowe formy organizacji plików:

- pliki nieuporządkowane,
- pliki uporządkowane,
- pliki haszowane,
- klastry.

Pliki nieuporządkowane – to najprostsza forma organizacji danych. Rekordy są umieszczane w takim pliku w kolejności ich wstawiania. Każdy nowy rekord jest umieszczany na ostatniej stronie pliku lub – w przypadku braku miejsca na niej – na nowo tworzonej stronie, dodawanej do pliku.

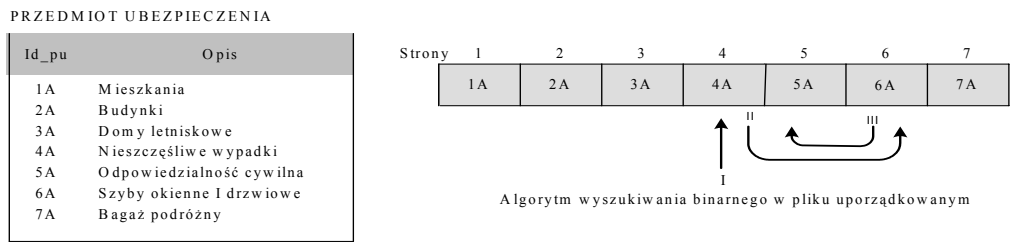
Organizacja taka jest efektywna w przypadku wstawiania danych, natomiast w przypadku wyszukiwania określonego rekordu plik musi zostać przeszukany liniowo, tzn. poprzez wczytywanie kolejnych stron pamięci, aż do zlokalizowania żądanych danych.

Podobnie czasochłonną operacją jest kasowanie danych. W tym przypadku, po odszukaniu strony zawierającej rekordy do kasowania, rekordy te są zaznaczane jako kasowane i strona jest ponownie zapisywana na dysk. Obszar zaznaczony jako skasowany nie jest wykorzystywany przy zapisie, dopóki nie nastąpi jawne przeorganizowanie obszaru dyskowego (przez administratora bazy danych). Sytuacje takie wpływają negatywnie na efektywność systemu, niemniej jednak organizacja danych w postaci plików nieuporządkowanych może być zalecana w przypadkach, gdy:

– dokonywane jest początkowe wprowadzanie dużej liczby danych do tabel,
– z założenia relacja mieści się na niewielkiej liczbie stron,
– w przetwarzaniu danych uczestniczą wszystkie wiersze,
– przewidywany jest dodatkowy mechanizm usprawniający dostęp do danych, tzn. indeks.

Pliki uporządkowane – w plikach tych rekordy są sortowane według wartości jednego lub kilku pól i tworzą sekwencję uporządkowaną względem określonego klucza. W praktyce, w większości systemów baz danych taką strukturę stosuje się rzadko, chyba że uporządkowanie dotyczy klucza głównego.

Przy takiej organizacji plików wstawianie i kasowanie danych jest czasochłonne, ale wyszukiwanie danych według wartości klucza porządkowania jest szybkie. Najczęściej stosowanym algorytmem wyszukiwania dla plików uporządkowanych jest wyszukiwanie binarne, realizowane poprzez rekurencyjne zawężanie obszaru wyszukiwania o połowę. Działanie algorytmu wyszukiwania binarnego ilustruje prosty przykład, nawiązujący do tabeli PRZEDMIOT UBEZPIECZENIA z rozdziału poprzedniego; dla uproszczenia przyjęte zostało założenie, że na każdej stronie pamięci znajduje się jeden rekord (rys. 9.1).



Rys. 9.1. Wyszukiwanie binarne w pliku uporządkowanym

Realizacja zapytania:

```
SELECT * FROM PRZEDMIOT UBEZPIECZENIA
WHERE Id_pu = 5A
```

Według algorytmu wyszukiwania binarnego:

I. Pobranie środkowej strony pliku (w przykładzie strona 4)→ porównanie wartości szukanej z wartością klucza na stronie → gdy równe wyszukiwanie jest kończone.

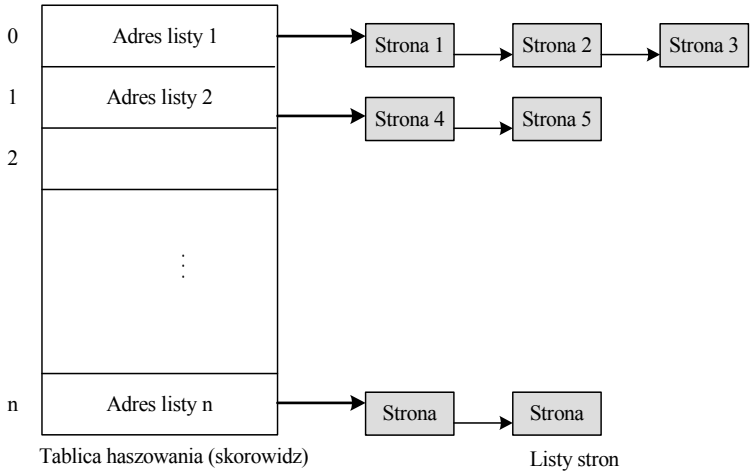
II. W przypadku, gdy wartość klucza w pierwszym rekordzie na stronie jest większa niż wartość poszukiwana, obszar przeszukiwania zawęża się do stron wcześniejszych, w przeciwnym razie do stron dalszych (w przykładzie 5A>4A).

III. Kroki I i II są powtarzane aż do momentu znalezienia żądanej wartości (w przykładzie pobierana jest strona 6).

Generalnie, przeszukiwanie binarne jest efektywniejsze niż liniowe dzięki zawężeniu o połowę obszaru przeszukiwania. Wracając do operacji wstawiania i kasowania danych – operacje te na plikach uporządkowanych są dosyć skomplikowane i czasochłonne ze względu na konieczność zachowania uporządkowania rekordów. Przy wstawianiu nowego rekordu do pliku musi zostać odnaleziona właściwa dla niego pozycja w sekwencji, a następnie obszar umieszczenia. Jeśli na stronie pamięci jest wystarczająca ilość miejsca na wpisanie nowego rekordu, strona jest przeorganizowywana i zapisywana na dysk. Jeżeli natomiast na stronie nie ma miejsca, część rekordów musi zostać przesunięta na następną stronę pamięci, co powoduje przesunięcie kolejnych rekordów z tej strony na następną itd. Zrealizowanie operacji wpisywania może okazać się w wielu sytuacjach (zwłaszcza w przypadku dużych plików danych) bardzo czasochłonne. Podobnie w przypadku kasowania rekordów każdorazowo musi nastąpić porządkowanie pliku. W praktyce jednym z rozwiązań poprawiających efektywność wstawiania danych do plików uporządkowanych jest tworzenie tymczasowego pliku nadmiarowego (*overflow file*, *transaction file*), nieuporządkowanego, do którego dane są zapisywane w kolejności pojawienia się, i okresowe łączenie obu plików [9]. Rozwiązanie takie poprawia efektywność wstawiania danych, może natomiast pogarszać efektywność wyszukiwania, gdyż w przypadku negatywnego wyniku wyszukiwania binarnego plik nadmiarowy musi być przeszukiwany liniowo.

Pliki haszowane (mieszane) – struktura taka jest dostępna w niektórych systemach bazodanowych poprzez komendę DDL. Ogólnie rzecz ujmując, utworzenie pliku haszowanego jest możliwe, jeżeli w systemie dostępna jest funkcja haszująca, wyliczająca adres strony lub listy stron, na której ma zostać umieszczony rekord na podstawie wartości znajdującej się w tzw. kluczu haszowania (wybrane pole lub pola rekordu). Jeżeli strony pamięci powiązane są w listę, to rekordy są umieszczane na liście w kolejności napływania. Inaczej mówiąc, funkcja haszująca konwertuje logiczną wartość klucza na adres fizyczny. Wewnętrzna struktura pliku haszowanego składa się z tablicy haszowania (rodzaj skorowidza) zawierającej numer listy stron pamięci oraz fizyczny adres listy (rys. 9.2).

Funkcja haszująca powinna być tak dobrana, aby rekordy rozmieszczane były w pliku równomiernie oraz aby rozmiar tablicy haszowania był jak najmniejszy. Istnieje wiele funkcji haszujących, jednak stosowane w praktyce algorytmy sprowadzają się do czterech typów:



Rys. 9.2. Struktura pliku haszowanego

1. Wybór cyfr (*Digit selection*): z pola klucza wybierany jest podzbiór bitów, znaków lub cyfr, które ustalają porządek haszowania. Przykładowo, przy telefonicznym składaniu zamówień dwie ostatnie cyfry numeru telefonicznego mogą zostać przypisane do nazwiska, wyznaczając tym samym listę, na którą trafia nazwisko.

2. Dzielenie (*Division*): wejście jest traktowane jako liczba, która jest dzielona przez inną (wcześniej określoną) liczbę, a reszta dzielenia wyznacza numer listy; jest to funkcja MODULO, czyli Hash (x) = MOD (x,m). Oczywiście w przypadku ciągów znaków musi nastąpić ich konwersja na cyfry (np. przypisanie zajmowanej pozycji w alfabecie, czy wartości ASCII).

3. Mnożenie (*Multiplication*): wejściem jest liczba, mnożona przez inną (wcześniej zdefiniowaną) liczbę, z wyniku wybierany jest podzbiór cyfr (najczęściej środkowych) wyznaczających numer listy.

4. Składanie (*Folding*): cyfry wejściowe są dzielone na podzbiory i następnie dodawane do siebie. Najczęściej algorytm ten występuje łącznie z mnożeniem lub dzieleniem.

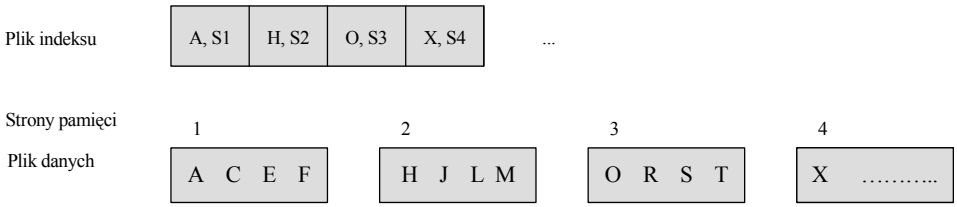
W systemach udostępniających możliwość haszowania plików musi istnieć mechanizm obsługi kolizji. Kolizja następuje w momencie odwzorowania różnych wartości logicznych na tę samą wartość klucza haszowania (czyli na ten sam adres listy). Problem pojawia się wówczas, gdy na stronach z listy brakuje miejsca dla zapisu rekordu. W Systemach Zarządzania Bazami Danych stosowane są różne mechanizmy obsługi kolizji – od najprostszego polegającego na tworzeniu obszarów przepełnień, poprzez haszowanie wielokrotne lub haszowanie dynamiczne, gdzie rozmiar listy zmieniany jest w odpowiedzi na operacje modyfikacji.

Wyszukiwanie rekordów odbywa się podobnie, tzn. na podstawie wartości klucza haszowania. W przypadku plików haszowanych wyszukiwanie według klucza haszo-

wania jest bardzo szybkie, ale przy innych kryteriach może okazać się nieefektywne bądź wręcz niemożliwe do zrealizowania.

Indeksy – są to mechanizmy przyspieszające wyszukiwanie danych (lokalizację poszczególnych rekordów w pliku) i przez to poprawiające czas realizacji zapytań użytkowników, bardziej uniwersalne niż metody haszowania. Ideę indeksu można przyrównać do skorowidzu w książce, umożliwiającego szybką lokalizację określonych pojęć na odpowiednich stronicach książki. W przypadku baz danych najprostszy mechanizm indeksowania polega na dodaniu do pliku danych dodatkowego pliku, czyli indeksu. Plik indeksu składa się z posortowanych wartości logicznych klucza (w odniesieniu do którego zakładany jest indeks) oraz adresu rekordu, w którym dana wartość klucza się znajduje [5, 9]. Mechanizmy indeksowania mogą być realizowane w różny sposób, niemniej jednak w większości pozycji literaturowych jako główne przyjmuje się następujące typy i struktury indeksów:

- Indeks pierwotny (*primary index*) związany jest z tzw. indeksowo-sekwencyjną metodą dostępu do danych (ISAM). W przypadku indeksu pierwotnego plik danych jest porządkowany według wartości klucza (patrz opis pliku uporządkowanego), plik indeksu natomiast zawiera uporządkowany zbiór wartości kluczy, będących pierwszymi na stronach pamięci wraz z adresami tych stron (rys. 9.3).



Rys. 9.3. Idea indeksu pierwotnego

Wyszukiwanie żądanego rekordu odbywa się poprzez plik indeksu, który zawiera posortowane klucze; plik indeksu przeszukiwany jest najczęściej metodą przeszukiwania binarnego. Po zlokalizowaniu adresu strony, na której znajduje się rekord o określonej wartości klucza, żądany rekord jest wyszukiwany na stronie pamięci.

Wyszukiwanie rekordów zorganizowanych w pliki indeksowo-sekwencyjne jest stosunkowo szybkie, zwłaszcza gdy plik indeksu nie jest duży. Większe problemy mogą się pojawić przy wstawianiu i kasowaniu rekordów, czyli podczas operacji aktualizacji. Pomijając konieczność aktualizacji dwóch plików: indeksowego i pliku danych, pozostaje sprawa organizacji obszaru przepełnienia (podobnie jak w przypadku kolizji przy haszowaniu). Indeks pierwotny ma charakter statyczny, w momencie jego powstawania muszą zostać zabezpieczone przez System Zarządzania Bazą Danych sytuacje, gdy na stronie pamięci brakuje miejsca na dopisanie nowych rekordów, czyli organizację obszaru przepełnienia: poprzez częściowe zapełnianie stron pamięci albo poprzez dołączanie dodatkowych stron.

Zauważmy, że ze względu na uporządkowanie danych w pliku głównym, dla jednoznacznej lokalizacji rekordu na stronie wystarczające jest, aby w pliku indeksu umieszczane były tylko rekordy pierwsze na każdej stronie. Indeks o takiej strukturze nosi nazwę *indeksu rzadkiego*. Przeciwstawnym pojęciem jest *indeks gęsty*, zawierający adresy wszystkich wartości klucza.

• Indeks wtórny (*secondary index*) w przeciwieństwie do indeksu pierwotnego nie wymaga porządkowania pliku danych, ani spełnienia warunku unikalnych wartości klucza. Plik indeksu natomiast jest uporządkowany. Istnieje kilka technik umożliwiających realizację indeksu wtórnego:

i) zakładanie indeksu gęstego zawierającego wszystkie wartości klucza (łącznie z ich duplikatami) wraz z ich adresami; inaczej mówiąc, jest to tworzenie mapy wszystkich rekordów pliku danych;

ii) zakładanie indeksu bez duplikowania wartości klucza, lecz z wielowartościowymi wskaźnikami adresowymi (każda z wartości określa kolejną lokalizację duplikatu wartości klucza w pliku danych);

iii) zakładanie indeksu bez duplikatów wartości klucza, ale z wielopoziomowym wskaźnikiem adresowym, tzn. wskazującym listę zawierającą adresy do poszczególnych rekordów w pliku danych, zawierających określoną wartość klucza.

Indeksy wtórne, które mogą być zakładane na różnych atrybutach relacji, przyspieszają operacje wyszukiwania danych, natomiast w operacjach aktualizacji wymuszają określone działania SZBD, które powodują obniżenie efektywności przetwarzania:

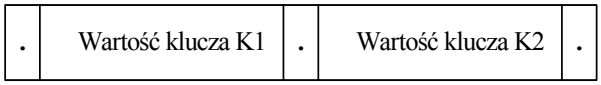
i) konieczność dodawania rekordów do wszystkich plików indeksów w momencie wstawiania wiersza do tabeli;

ii) aktualizowanie pliku indeksu w momencie aktualizowania tabeli;

iii) zwiększenie czasu optymalizacji zapytań w wyniku konieczności analizowania przez optymalizatory systemowe wszystkich indeksów wtórnych.

Dodatkowo, nie bez znaczenia może być fakt zwiększenia zajętości obszaru dyskowego, potrzebnego dla pamiętania plików indeksowych.

• Indeksy wielopoziomowe – B-drzewa są szczególnym typem indeksu gęstego. Struktura drzewiasta indeksu, która jest bardziej elastyczna niż struktury opisane poprzednio, eliminuje niedogodności związane z operacjami wstawiania i usuwania wierszy do tabel. Termin B-drzewo oznacza drzewo wyważone (*balanced*), czyli takie, w którym odległość od korzenia drzewa do węzłów końcowych (liście) jest jednakowa dla wszystkich gałęzi. Drzewo składa się z węzłów, zawierających jedną lub kilka wartości klucza, na przemian ze wskaźnikami do innych rekordów w drzewie, czyli węzeł ma postać:



Węzły są ułożone w hierarchię na takiej zasadzie, że każdy węzeł nie będący korzeniem posiada jednego rodzica i zero lub kilka węzłów potomnych. Węzły nie posiadające węzłów potomnych stanowią najniższy poziom drzewa, tzw. liście. Wysokość drzewa jest odległością od korzenia do liści, stopień drzewa określa liczbę dozwolonych węzłów potomnych (w przypadku, gdy dozwolona liczba węzłów potomnych wynosi 2, mamy do czynienia z drzewami binarnymi).

W odniesieniu do struktury B-drzewa obowiązują następujące reguły [9]:

i) Korzeń musi mieć co najmniej dwa węzły potomne.

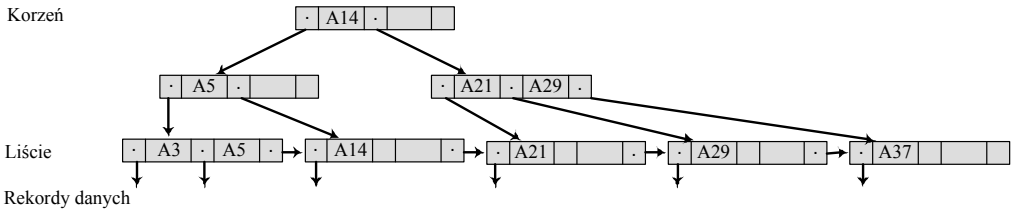
ii) Dla drzewa o stopniu n, dla każdego węzła (z wyjątkiem korzenia i liści) liczba wskaźników i węzłów potomnych zawiera się między $n/2$ i n. Gdy $n/2$ nie jest liczbą całkowitą, to wynik jest zaokrąglany.

iii) Dla drzewa o stopniu n, liczba wartości klucza oraz liczba wskaźników w liściu muszą być zawarte między $(n–1)/2$ i $(n–1)$. Gdy $(n–1)/2$ nie jest liczbą całkowitą, wynik jest zaokrąglany.

iv) Liczba wartości klucza w każdym z węzłów nie będących liśćmi jest o 1 mniejsza niż liczba wskaźników.

v) Wartości kluczy w węzłach końcowych (liściach) są uporządkowane.

Przykładowy indeks o strukturze B-drzewa ilustruje rysunek 9.4.

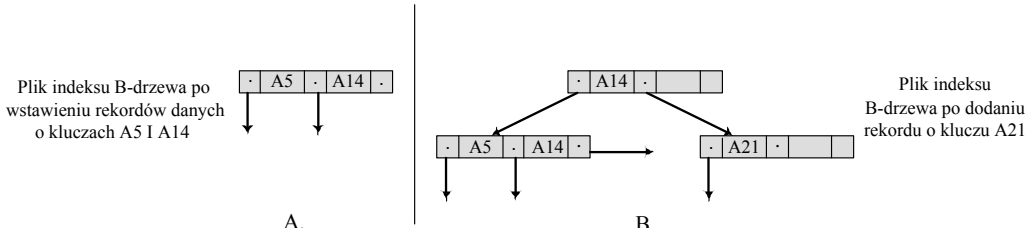


Rys. 9.4. Przykładowe B-drzewo

Przeszukiwanie drzewa odbywa się według następującej zasady: gdy szukana wartość klucza jest mniejsza lub równa wartości w węźle, pobierany jest wskaźnik adresowy z lewej strony wartości klucza, jeżeli większa – to wskaźnik z prawej strony. Przykładowo, jeżeli szukana wartość klucza wynosi A29, wyszukiwanie zaczyna się od korzenia; po porównaniu wartości kluczy pobrany zostanie wskaźnik z prawej strony (A29>A14), wskazujący na drugi poziom drzewa. Na drugim poziomie węzeł zawiera dwie wartości klucza – A21 i A29. Wskaźnik z lewej strony klucza A29 wskazuje węzeł końcowy, zawierający adres rekordu o kluczu A29.

W praktyce węzeł jest stroną pamięci, oczywistym jest więc, że może przechowywać więcej niż dwie wartości klucza i trzy wskaźniki. Liczba rekordów indeksu przechowywanych na stronie zależy od rozmiaru pola klucza oraz wskaźników adresowych (najczęściej 4 bajty). Indeksy o strukturze B-drzewa zapewniają taki sam czas wyszukiwania dla dowolnego rekordu danych, co wynika z założenia zrównoważenia

drzewa (każda gałąź ma tę samą wysokość). Ponadto, ponieważ indeks jest indeksem gęstym, plik danych nie musi być sortowany. Operacje wstawiania i usuwania rekordów wymagają jednak modyfikacji B-drzewa, co często wiąże się z koniecznością dostawiania nowych stron pamięci dla kolejnych węzłów drzewa na każdym poziomie (rys. 9.5).



Rys. 9.5. Modyfikacja pliku indeksu B-drzewa przy wstawianiu rekordów

Klastry – niektóre z Systemów Zarządzania Bazą Danych umożliwiają tworzenie obszarów fizycznych, w których pamiętane są tablice często łączone w zapytaniach do bazy danych; inaczej mówiąc, klastry umożliwiają implementację złączeń [5, 9]. Proces budowania klastrów zależy od SZBD i ogólnie mówiąc – zaczyna się od polecenia utworzenia klastra i zadeklarowania tzw. *klucza klastra*, czyli kolumny, według której łączone są tabele umieszczane w klastrze, następnie tworzone tabele są przypisywane do klastra (umieszczane w klastrze). Wybór takiej struktury danych zależy od analizy transakcji przeprowadzanych w odniesieniu do danych gromadzonych w bazie, w przypadku bowiem przeszukiwania bazy danych według kryteriów innych niż zaimplementowane złączenia może nastąpić znaczne pogorszenie czasu dostępu. Ogólne kryteria, które mogą być pomocne w decyzji, czy tworzyć klastry, można ująć następująco:

i) Oszacowanie częstotliwości złączeń tabel – jeżeli zachodzi ono okazjonalnie, nie ma potrzeby umieszczania danych w obszarze klastra,

ii) Oszacowanie częstotliwości modyfikacji klucza klastra – modyfikacja wartości klucza klastra jest czasochłonna,

iii) Oszacowanie sposobu użycia danych – gdy jedna z łączonych tabel jest często przeszukiwana w całości, operację taką wykonuje się dłużej na tabeli klastrowanej,

iv) Oszacowanie rozmiaru danych – w przypadku zbyt dużych rozmiarów klastra (więcej niż jedna, dwie strony pamięci) dostęp do pojedynczego wiersza tabel klastrowanych wymaga więcej odczytów z dysku, niż w przypadku tabel nieklastrowanych.

9.2. Szacowanie rozmiaru danych i analiza użycia

Rozważania dotyczące oszacowań rozmiarów danych oraz sposobu użycia danych muszą się odnosić do konkretnego projektu, z uwzględnieniem wcześniejszych założeń dotyczących zarówno bazy danych, jak i aplikacji oraz wybranego środowiska implementacyjnego, dlatego też w rozdziale niniejszym przedstawiono jedynie ogólne wskazówki metodologiczne, które mogą być wykorzystane w projektowaniu warstwy fizycznej bazy danych.

Szacowanie rozmiaru danych – celem oszacowania rozmiaru danych jest określenie zajętości obszaru dyskowego, zajmowanego przez bazę danych. Szacowanie ma wpływ na wybór środowiska bazodanowego oraz konfiguracji sprzętowej, ale i odwrotnie – określone środowisko z góry narzuca sposób szacowania. Ogólnie mówiąc, szacowanie bazuje na rozmiarze wiersza relacji i liczbie wierszy w relacji. Dysponując schematem bazy danych, w którym dla każdej tabeli ustalona została liczba kolumn w wierszu oraz rozmiar każdej kolumny, łatwo wyliczyć długość wiersza (suma rozmiarów kolumn). Zakładając, że potrafimy na podstawie modelu logicznego określić maksymalną liczbę wierszy w tabeli, można oszacować rozmiar tabeli (iloczyn liczby wierszy przez długość wiersza), chociaż od razu należałoby założyć, że tabela będzie rosła, a więc oszacowanie powinno uwzględnić współczynnik przyrostu danych w czasie. Szacowanie rozmiaru pamięci dyskowej potrzebnej dla danej tabeli zależy natomiast od wybranego środowiska implementacyjnego: oprócz wyliczenia rozmiaru tabeli należy bowiem uwzględnić wszystkie nagłówki systemowe, których wielkość zależy od konkretnego środowiska (nagłówki stron pamięci, nagłówek pliku, nagłówki rekordu) oraz rozmiar obszaru na stronie pamięci zarezerwowany systemowo dla aktualizacji danych. Dane potrzebne do tego typu wyliczeń są dostępne w opisach technicznych poszczególnych środowisk bazodanowych.

Analiza użycia danych ma na celu sprecyzowanie rodzajów i częstotliwości transakcji wykonywanych na bazie danych. Przez transakcje rozumiane są operacje wstawiania, aktualizacji kasowania i odczytu. Analiza tego rodzaju ma pomóc w określeniu wpływu na wydajność systemu poszczególnych operacji w zależności od częstotliwości ich wykonywania oraz zakresu i sposobu działania i w rezultacie na dobraniu odpowiednich struktur plików oraz metod dostępu do nich. W przebiegu procesu analizy użycia danych można wyróżnić kilka etapów:

Etap 1. Określenie rodzaju transakcji.

Etap 2. Oszacowanie częstotliwości wykonywania transakcji.

Etap 3. Oszacowanie zakresu działania transakcji.

Etap 4. Wyodrębnienie atrybutów będących przedmiotem działania transakcji.

Etap 1. W wielu przypadkach w fazie projektowania warstwy fizycznej bazy danych nie ma możliwości sporządzenia pełnej listy transakcji, tak więc analiza użycia sprowadza się do wyodrębnienia „najważniejszych”. Okazuje się bowiem w praktyce,

że 20% najbardziej aktywnych zapytań do bazy danych wykorzystuje 80% wszystkich dostępów do bazy danych (zasada 80/20, Wiedrehold, 1983).

Konstruując model logiczny bazy danych przeprowadza się jego walidację, m.in. poprzez sprawdzenie możliwości realizacji ustalonych aktualizacji oraz zapytań do bazy danych. Wykorzystując taką listę, można zbudować prostą macierz krzyżową [9], służącą do wizualizacji związków pomiędzy transakcjami a tabelami (relacjami) bazy danych (rys. 9.6).

Transakcja / Tabela (relacja)	T1				T2				T3				...				Tn			
	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D
R1						X	X		X											X
R2		X				X														
R3									X											
...																				
Rn									X					X						

Rys. 9.6. Macierz krzyżowa dla tabel i transakcji (I – Insert, R – Read, U – Update, D – Delete)

Etap 2. Następnym krokiem powinno być oszacowanie częstotliwości wykonywania transakcji. Powinno się ją określać nie tylko jako średnią i maksymalną częstotliwość w ciągu godziny i doby, ale także w odniesieniu do szczytowego obciążenia systemu. Jest rzeczą oczywistą, że transakcje wykonywane z dużą częstotliwością muszą mieć zapewniony efektywny sposób dostępu do danych, na których operują, może jednak zaistnieć sytuacja, kiedy transakcji nie wykonuje się często, ale musi być wykonana w określonym czasie (pokrywającym się z czasem szczytowego obciążenia systemu) albo wcale, lub może mieć zdefiniowany czas zakończenia. Rozważania takie prowadzą do układania harmonogramów wykonywania transakcji, często także do przebudowywania struktury bazy danych (strojenie systemu) w celu poprawienia parametrów wydajnościowych.

Etap 3. Wyodrębnione transakcje podlegają dużo bardziej szczegółowej analizie, uwzględniającej zasięg działania transakcji (szacunkowa liczba wierszy używanych przez transakcje – zarówno wyszukujące, jak i aktualizujące dane), z wyodrębnieniem zakresu w obciążeniu szczytowym.

Etap 4. Następnym krokiem jest wykonanie analizy schodzącej do poziomu atrybutów (kolumn), ze względu na konieczność określenia priorytetów indeksowania kolumn (indeksy wtórne). Na tym etapie wyodrębniane są atrybuty nie tylko podlegające aktualizacji, ale również atrybuty występujące w kryteriach wyszukiwania (w klauzuli WHERE) zarówno dla odczytu, jak i aktualizacji oraz atrybuty, według

których odbywa się łączenie tabel, grupowanie danych (poprzez funkcje sumaryczne). Wyodrębnione atrybuty (kolumny) są potencjalnymi kandydatami do indeksowania.

Dysponując wynikami oszacowań rozmiarów tabel, analizy użycia danych oraz wiedzą na temat struktur plików danych i ich własności, cel jakim jest zaprojektowanie fizycznej warstwy bazy danych staje się możliwy do zrealizowania. Dalsze wskazówki metodologiczne koncentrują się na zasadności stosowania struktur sekwencyjnych lub indeksowanych – ze względu na największą popularność tych rozwiązań – większość systemów baz danych (w tym środowisko Sybase) umożliwia stosowanie indeksów, nieliczne natomiast umożliwiają haszowanie lub budowanie klastrów. Punktem wyjścia jest ustalenie wstępnej listy indeksów według następujących wskazówek ogólnych:

- Nie jest opłacalne stosowanie indeksów dla małych tabel. Bardziej efektywne jest przeszukiwanie takich tabel niż pamiętanie dodatkowej struktury indeksu.
- Indeksowanie klucza głównego tabeli. Wprawdzie w większości nowych SZBD indeksowanie klucza odbywa się automatycznie, ale nie zawsze jest to regułą.
- Zakładanie indeksu wtórnego na kluczu obcym, jeżeli wykonywane są częste łączenia tabel. Podobnie jak w poprzednim przypadku, wiele SZBD automatycznie indeksuje pola klucza obcego.
- Zakładanie indeksów wtórnych na atrybutach często występujących w kryterium wyboru (klauzula WHERE), atrybutach będących podstawą grupowania wierszy (klauzula GROUP BY) lub sortowania (klauzula ORDER BY).
- Zakładanie indeksów wtórnych na atrybutach będących argumentami funkcji agregujących.
- Niezakładanie indeksów na atrybutach często aktualizowanych tabel.
- Niezakładanie indeksów na atrybutach tabel, w przypadku gdy transakcje przetwarzają duże zestawy wierszy (więcej niż 20%).
- Nieindeksowanie atrybutów zawierających długie ciągi znaków.

Ostatnie dwa etapy projektowania warstwy fizycznej bazy danych dotyczą konstruowania perspektyw oraz wyboru mechanizmów bezpieczeństwa. Perspektywy muszą odzwierciedlać różne potrzeby widzenia i przetwarzania danych przez użytkowników lub grupy użytkowników, sprecyzowane w fazie ustalania wymagań wobec systemu. Również w tej fazie określane są żądania użytkowników dotyczące bezpieczeństwa danych. Mechanizmy zabezpieczeń zależą od konkretnego SZBD, ale ogólnie można wyróżnić zabezpieczenia na poziomie systemu (nazwy użytkowników, hasła) oraz na poziomie danych (udostępnianie określonych zakresów danych, przydzielanie uprawnień do określonych operacji).

9.3. Poprawianie wydajności, strojenie systemu

Pojęcie wydajności systemu bazodanowego jest ściśle związane z charakterem aplikacji obsługujących bazę danych. Wszystkie rozważania dotyczące projektu warstwy fizycznej bazy danych muszą być prowadzone pod kątem priorytetów aplikacji. Nie zawsze bowiem można pogodzić szybki dostęp do danych z szybką aktualizacją danych. Wyniki oszacowań rozmiaru danych i analiza użycia są podstawą do zaprojektowania struktury warstwy fizycznej bazy danych. Następnym, rutynowym krokiem w procesie projektowym jest strojenie bazy danych, w praktyce bowiem może okazać się, że decyzje projektowe nie zawsze były trafne lub nie wszystkie mechanizmy umożliwiające poprawę wydajności systemu zostały wykorzystane.

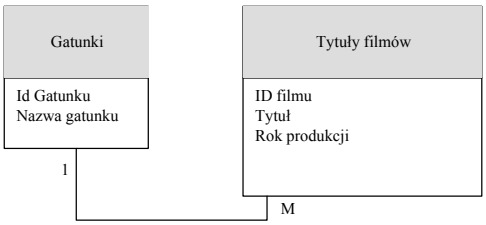
Przede wszystkim należy uważnie przeanalizować listę potencjalnych indeksów i zbadać (eksperymentalnie) ich wpływ na operacje aktualizowania danych, może bowiem okazać się, że obecność indeksu nie ma znaczącego wpływu lub znacznie pogarsza efektywność wykonywania niektórych transakcji. Podobnie okazać się może, że w operacjach wyszukiwania danych indeks nie zawsze poprawia czas realizacji transakcji. Przykładowo, jeżeli realizowane jest zapytanie wyszukiwania z kryterium złożonym, w którym występuje klauzula OR, założenie indeksu tylko na jednym z atrybutów nie poprawia efektywności wyszukiwania – tabela, do której odnosi się zapytanie będzie przeszukiwana sekwencyjnie. Obecnie większość SZBD jest wyposażona w *optymalizatory zapytań* (w środowisku Sybase jest to Query Optimizer; patrz dokumentacja techniczna systemu Sybase), które dokonują analizy zapytania (pod kątem kosztu dostępów do dysku i obciążenia procesora) i podają najbardziej efektywny plan jego wykonania, łącznie z propozycją stosowania indeksów. Jest to narzędzie godne polecenia zarówno w przypadku układania złożonych zapytań, jak i w przypadku zbyt powolnego wykonywania się zapytań.

Następną kwestią związaną z indeksami jest wprowadzanie danych do bazy danych. W przypadku wpisywania dużej liczby danych do tabeli z założonym indeksem lub indeksami efektywniejsze jest zwolnienie czasowo indeksu i przywrócenie go po zakończeniu operacji INSERT.

Następną możliwą opcją wykorzystywaną przy strojeniu bazy danych jest tzw. denormalizacja bazy danych, czyli odejście od założonego w schemacie bazy danych stopnia normalizacji dla niektórych tabel, co prowadzi do kontrolowanej redundancji danych. Tego typu podejście stosowane jest najczęściej w odniesieniu do tabel zawierających opisy instancji obiektów, często nazywanych tabelami odniesień lub słownikowymi, w których dane nie zmieniają się często i jest ich niewiele. W sytuacji, gdy przetwarzanie wymaga wykonywania częstych złączeń wierszy takiej tabeli z inną stosuje się dodanie informacji przechowywanej w tabeli odniesień do tabeli z nią powiązanej (rys. 9.7 a, b). Rozwiązanie takie zmniejsza czas wyszukiwania, ale nie jest wolne od wad – tych

wszystkich, którym zapobiega normalizacja. Dodatkowe koszty tego typu rozwiązań będą dotyczyć przede wszystkim utrzymania bazy w stanie spójnym.

a)



b)

Filmy

Id filmu	Tytuł	Rok produkcji	Id gat	Nazwa gatunku
11PG3	Potop	1974	1	Historyczny
12TF5	Czarna wdowa	2001	2	Horror
23HK5	Zezowate szczęście	1950	5	Komedia
13PG5	Misery	1989	2	Horror

Rys. 9.7. Przykład denormalizacji schematu bazy danych:
a) tabele przed denormalizacją, b) efekt denormalizacji

Innym przykładem odejścia od reguł relacyjnych baz danych na rzecz poprawy wydajności jest przechowywanie danych obliczanych. Z zasady w bazie danych nie powinny być przechowywane wartości, które są wyliczane. Jeżeli jednak okazuje się, że dane potrzebne do wyliczeń znajdują się w różnych plikach, a wyliczenia odbywają się często, to ze względu na czas dostępu do danych sensownym staje się bezpośrednie przechowywanie wyników obliczeń.

Uwaga: Wprowadzenie redundancji danych poprzez denormalizację schematu bazy danych powinno być stosowane w ściśle uzasadnionych wypadkach, kiedy inne metody poprawy wydajności okażą się nieskuteczne. Z zasady najbardziej niebezpieczne są denormalizacje dużych plików, które podlegają częstym aktualizacjom. Zmiany schematu powinny być udokumentowane i skomentowane.

Wszystkie zabiegi przedstawione w niniejszym rozdziale mają na celu poprawę wydajności projektowanego i implementowanego systemu z bazą danych. Należy pamiętać, że proces strojenia nie jest procesem statycznym – SZBD wyposażone są w narzędzia umożliwiające administratorowi systemu ciągłe monitorowanie i strojenie systemu. W dokumentacji Sybase opisującej serwer ASA (Adaptive Server Anywhere) wyszczególnione są opcje mające istotny wpływ na wydajność przetwarzania, w zależności od ich ustawienia. Do takich opcji należą:

- Protokół transakcji – serwer może lub nie prowadzić protokół transakcji (*transaction log*), zapisywany w pliku (*.log). Opcja ta ma istotny wpływ na szybkość przetwarzania – pomimo tego, że serwer musi zarządzać dziennikiem transakcji, protokołowanie przyspiesza pracę serwera poprzez kompleksowy tryb obsługi aktualizacji bazy danych. Bez protokołowania każda zmiana danych musi być zapisana w pliku danych i sprawdzona po zakończeniu każdej pojedynczej transakcji (*checkpoint*), co jest bardzo czasochłonne. Na poprawę zarówno wydajności, jak i niezawodności systemu wpływa również alokacja protokołu; powinien on być umieszczony na innym urządzeniu dyskowym niż dane.

• Przydział pamięci typu *cache* serwerowi – serwer powinien dysponować możliwie dużą pamięcią, ponieważ operowanie na pamięci jest nieporównywalnie szybsze niż odczyty z dysku.

• Umieszczenie plików tymczasowych (używanych przez serwer podczas operacji sortowania, grupowania lub unii) na innym urządzeniu dyskowym niż dane.

• W przypadku sekwencyjnego dostępu do danych lub dużych baz danych wybór dużego rozmiaru strony pamięci (4 Kb).

• Niestosowanie automatycznego trybu zatwierdzania transakcji (*autocommit mode*), który jest domyślnie przyjętym trybem dla interfejsów ODBC i JDBC. Tryb ten wymusza bowiem traktowanie każdego pojedynczego polecenia *SQL* jako oddzielnej transakcji.

Nie są to oczywiście wszystkie możliwości strojenia bazy danych – administrator bazy danych dysponuje opisami technicznymi środowiska, zawierającymi pełną listę parametrów i opcji zarówno inicjalizacyjnych, jak i modyfikujących pracę systemu. Do zadań administratora należy również bieżące monitorowanie bazy danych, możliwe dzięki narzędziom systemowym, w zakresie:

– nazw aktualnych użytkowników,
– rodzaju wykonywanych aplikacji,
– statystyk operacji we/wy,
– statystyk użycia bazy danych,
– analizy planów wykonywania poleceń *SQL*.

Wszystkie działania związane ze strojeniem systemu, zarówno wstępne jak i bieżące, mają na celu oszczędność czasu i zwiększenie satysfakcji użytkowników. Jeszcze raz należy podkreślić, że proces strojenia jest procesem dynamicznym i ciągłym, który musi uwzględniać zmiany otoczenia i badać ich wpływ na zachowanie systemu.

10

Aplikacje

Traktując etap projektowania i implementacji bazy danych jako zakończony, czyli przyjmując, że fundament, na którym budowane będą aplikacje jest gotowy, można przystąpić do realizacji pozostałej części systemu bazodanowego, czyli aplikacji. W niniejszym rozdziale oprócz ogólnych wskazówek metodologicznych przedstawione zostanie narzędzie RAD, przeznaczone do tworzenia oprogramowania klient–serwer, standardowo związane ze środowiskiem Sybase, czyli PowerBuilder.

Projektując aplikację, opieramy się na opracowanym modelu systemu (rozdz. 6), umożliwiającym określenie obiektów aplikacji (formularzy, raportów oraz bloków kodu programu) na podstawie zamodelowanych reguł przetwarzania. W tym miejscu należy wspomnieć, że narzędzie Case – AppModeler umożliwia automatyczne generowanie aplikacji zarówno dla środowiska PowerBuilder, jak i dla Delphi, Power ++, Visual Basic oraz Web na podstawie modelu fizycznego bazy danych. Oczywiście takie aplikacje stanowią jedynie podstawę do dalszych uzupełnień, niemniej jednak mogą być pomocne, chociażby ze względu na łatwość i szybkość ich uzyskania.

Przedstawione poniżej uwagi dotyczące tworzenia aplikacji koncentrują się w zasadzie na wskazówkach dotyczących projektowania formularzy (umożliwiających pobieranie, prezentowanie i operowanie na danych) oraz raportów, z tego względu, że PowerBuilder jest narzędziem programowania wizualnego, które w ostatnich czasach zdecydowanie wypiera tradycyjne programowanie strukturalne. Główne zasady cechujące dobry interfejs graficzny (*Graphical User Interface*, GUI) można przedstawić następująco:

- Zachowanie spójności (standaryzacja interfejsu) – zasada ta dotyczy nie tylko zbioru formularzy danej aplikacji, czyli określenia czcionek, kolorów, położenia ikon i przycisków, wielkości obiektów oraz standardów prowadzenia dialogu z użytkownikiem. Równie ważne jest, aby aplikacja przestrzegała konwencji przyjętych dla innych aplikacji powszechnie stosowanych w danym środowisku.
- Ograniczenie zbędnych funkcji i komunikatów – funkcje powinny wynikać z rzeczywistych potrzeb użytkownika, komunikaty natomiast nie powinny komentować sytuacji oczywistych, lecz ostrzegać przed wykonaniem czynności niemożliwych do cofnięcia lub krytycznych dla pracy systemu. Jeżeli formularz ma służyć jedynie

do wyświetlania danych, nie należy umieszczać w nim opcji umożliwiających wprowadzanie danych, nawet jeżeli będą one nieaktywne.

- Formularze powinny prezentować informacje należące do jednej klasy, przykładowo w jednym formularzu nie powinny znajdować się dane dotyczące zwolnienia lekarskiego pacjenta i wypisanych mu recept.
- Interfejs powinien umożliwiać różne sposoby dostępu do funkcji systemu – przyciski na formularzach powinny mieć odpowiedniki w postaci opcji menu, w uzasadnionych sytuacjach powinny zostać utworzone paski narzędzi i klawisze skrótów oraz menu podręczne wywoływane prawym klawiszem myszki.
- Najważniejsze pola w formularzu powinny być etykietowane.
- W interfejsie powinna zostać wykorzystana możliwość wskazówek wizualnych poprzez ustawianie fokusu na odpowiednich polach, wykorzystanie wartości domyślnych, zmiany kształtu wskaźnika myszy.
- Elementy formularza logicznie ze sobą powiązane powinny być umieszczane obok siebie; ułatwia to zgodny z logiką i intuicją sposób działania użytkownika.
- Użytkownik powinien mieć możliwość cofnięcia danej operacji (opcja *Cancel*).
- Interfejs powinien zostać wyposażony w mechanizm podpowiedzi (*hints*) informujący o funkcji danej kontrolki lub żądanym formacie danych.
- Interfejs powinien umożliwiać korzystanie z systemu pomocy (*help*), zawierającego dokumenty opisujące działanie aplikacji.
- W przypadku formularzy do wprowadzania danych istotna jest kolejność przechodzenia pomiędzy polami formularza, powinien więc zostać zachowany logiczny porządek: od lewej do prawej strony i z góry do dołu.
- Dobrą tradycją są komunikaty informujące o rodzaju błędu popełnionego przez użytkownika i sposobu jego poprawienia.
- W przypadku długotrwałych działań systemu przy realizacji określonej funkcji użytkownik powinien być o tym informowany (komunikaty o procentowym wykonaniu funkcji lub czasie potrzebnym do zakończenia operacji).
- Istotną sprawą jest kolorystyka formularzy – biorąc pod uwagę, że system jest narzędziem pracy (dla niektórych grup użytkowników, takich jak operatorzy czy administratorzy jest to praca wielogodzinna), kolory zarówno tła jak i elementów formularzy powinny być stonowane, nie męczące wzroku.

Podsumowując, można najogólniej stwierdzić, że dobry interfejs użytkownika może być kluczem do sukcesu całej aplikacji, jeżeli natomiast interfejs okaże się nieprzyjazny (nie działający zgodnie z intuicją i logiką), zbyt trudny do opanowania przez użytkowników, mało elastyczny, to nawet najlepiej działający system nie ma szans na wdrożenie i eksploatację. Najdobitniej o roli interfejsu, udostępniającego przecież użytkownikowi funkcje systemu, mogą świadczyć liczne pozycje literaturowe prezentujące szczegółowo metodykę i zasady projektowania interfejsów użytkownika.

10.1. Konstruowanie aplikacji w środowisku PowerBuilder

Szczegółowe przedstawienie wszystkich możliwości narzędzia, jakim jest PowerBuilder oraz całego cyklu budowy aplikacji w tym środowisku przekracza zakres niniejszego podręcznika. Istnieje zresztą podręcznik, który cele te realizuje w pełni, a mianowicie *PowerBuilder 6 – oficjalny podręcznik*, autorstwa Steve Erlanka i Craiga Levina, do którego powinni zaglądać wszyscy zainteresowani. Rozdział niniejszy stanowi wprowadzenie do metod tworzenia systemów i poruszania się w środowisku PowerBuilder.

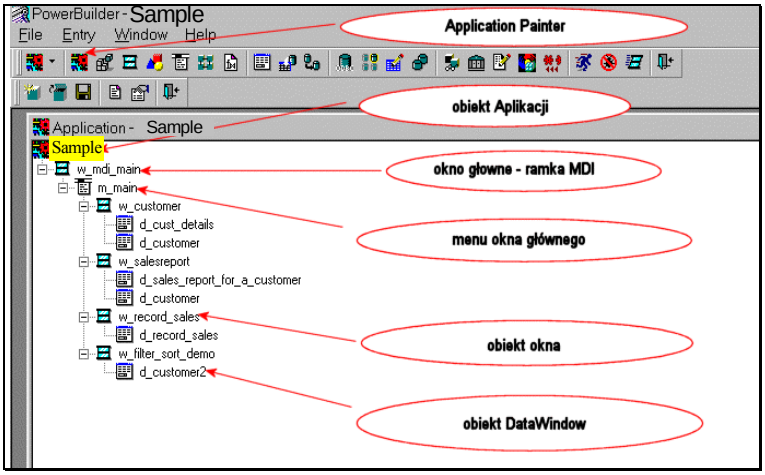
PowerBuilder jest obiektowym narzędziem programowania czwartej generacji zaliczanym do narzędzi typu RAD, z możliwościami pracy w kilku systemach operacyjnych, m.in. Windows 9x, Windows NT, MacOS, Sun Solaris. Jest on standardowo rozprowadzany z systemem zarządzania bazami danych Sybase, lecz może się komunikować z innymi bazami danych (np. Oracle, DB2). PowerBuilder obsługuje interfejs ODBC, będący standardem dostępu do relacyjnych baz danych; informacje o bazach danych dostępnych dla systemu są zapisywane w pliku kontrolnym poprzez ustawienie tzw. profili (sposób tworzenia profili zostanie omówiony w dalszej części rozdziału). Dla PowerBuildera bazy te stanowią źródła danych. Aplikacja skonstruowana za pomocą PowerBuildera składa się z wielu obiektów, z przypisanymi atrybutami, które określają wygląd danego obiektu, zdarzeniami, które określają zachowanie obiektu oraz skryptami, które są przypisywane do zdarzeń i wykonywane podczas zaistnienia zdarzenia. Najpopularniejszym obiektem występującym praktycznie we wszystkich graficznych środowiskach programistycznych są okna, reagujące na czynności wykonywane za pomocą myszki i klawiatury, służące do wyświetlania i modyfikowania danych oraz pobierania odpowiedzi użytkownika. Oknem typowym dla środowiska PowerBuilder, a nie występującym w innych środowiskach, jest obiekt o nazwie DataWindow, bezpośrednio połączony z bazą danych, umożliwiający elastyczny i prosty sposób operowania danymi.

10.1.1. Środowisko PowerBuildera

Prawie wszystkimi obiektami w PowerBuilderze manipuluje się za pomocą edytorów (ang. *painters*). Przykładowo, jeżeli programista chce stworzyć okno w aplikacji, musi użyć specjalnie dedykowanego do tego celu edytora WindowPainter, jeżeli chce stworzyć menu w kreowanym programie, powinien skorzystać z edytora Menu Painter itd. Podstawowymi edytorami występującymi w PowerBuilderze są:

 Application Painter (edytor główny aplikacji) – wszystkie komponenty tworzonego systemu (skrypty, tworzone okna, kreowane menu itp.) są zgrupowane razem w obiekcie aplikacji. Obiekt aplikacji steruje zachowaniem elementów (obiektów),

które wchodzą w jego skład. Obiekty te są przechowywane w bibliotekach aplikacji (pliki z rozszerzeniem *.pbl). Rysunek 10.1 obrazuje załadowany Application Painter z otwartą aplikacją, prezentując hierarchię obiektów w aplikacji (np. okno główne podlega aplikacji, obiekt typu DataWindow podlega „pod" okno itd).



Rys. 10.1. Application Painter – edytor główny aplikacji

Menu Painter (Edytor menu)

DataWindow Painter (Edytor obiektów *DataWindow*) – obiekty DataWindow służą do pobierania, prezentacji i operacji na danych. Pobierają one informacje z baz danych lub z określonych plików (np. *.dbf lub za pomocą DDE). Są również stosowane w przypadku publikowania informacji w Internecie. Za ich pomocą można wyświetlać rezultaty zapytań SQL w określony sposób: w formie wykresów, raportów, zestawień itd. Przy ich użyciu można także modyfikować określone wiersze tabel w bazie danych. Obiekty DW stosuje się w zwykłych oknach. Stworzenie obiektu DW składa się z kilku etapów:

Etap 1 polega na utworzeniu obiektu DW za pomocą Edytora DataWindow Painter (określenie stylu wyświetlania danych, kryteriów sortowania i filtrowania, definicja źródła danych pobieranych przez DW itd). Źródłem danych dla obiektu DW jest baza danych, w której znajduje się tabela.

Etap 2 polega na utworzeniu obiektu **dialogowego** typu DataWindow, który jest osadzony w obiekcie okna.

Etap 3 polega na połączeniu obiektu **dialogowego** DW z utworzonym obiektem DW w bibliotece. Oba obiekty będą stanowiły „całość".

DataBase Painter (Edytor baz danych)

Library Painter (Edytor bibliotek)

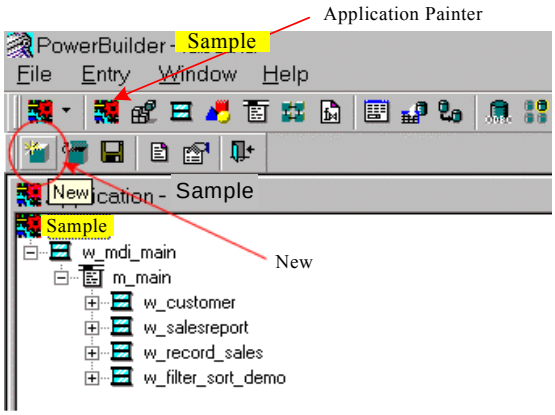
PowerScript Painter (Edytor skryptów)

Skrypty są to bloki programu wbudowane w obiekty (np. w obiekt aplikacji, w okna itp.). Są one powiązane ze zdarzeniami występującymi w obiektach. Oznacza to, że skrypty zostaną wykonane wtedy, gdy określone zdarzenie zajdzie. Przykładowo, gdy uruchamiamy tworzony program, zostaje wywołane zdarzenie ***open*** obiektu aplikacji i wykonany odpowiedni skrypt związany z tym zdarzeniem (zazwyczaj łączący program z bazą danych i otwierający okno główne).

Function Painter (Edytor Funkcji)

10.1.2. Początek pracy – tworzenie obiektu aplikacji

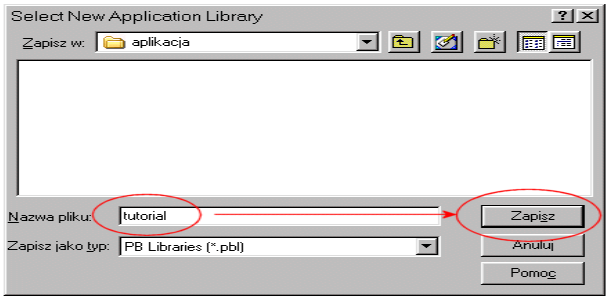
Pierwszym krokiem na drodze do zbudowania programu współpracującego z bazą danych jest utworzenie obiektu aplikacji i zapisanie go w pliku biblioteki. W tym celu należy odpowiednim przyciskiem z paska ikon uruchomić Application Painter. W edytorze powinna znajdować się aplikacja, z którą pracowano ostatnio (rys. 10.2). W celu utworzenia nowej aplikacji należy posłużyć się przyciskiem **New** (rys. 10.2).



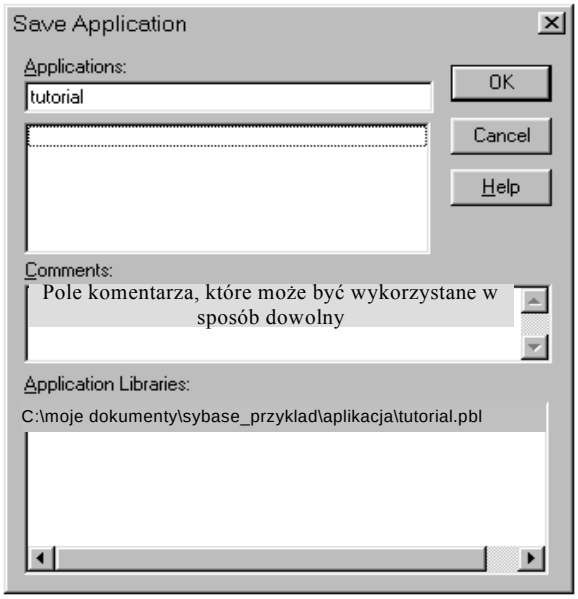
Rys. 10.2. Tworzenie obiektu aplikacji

Wyświetlone zostanie standardowe okno systemu operacyjnego umożliwiające określenie folderu, w którym zostanie zapisana biblioteka aplikacji z wybraną nazwą.

Następnym krokiem po utworzeniu biblioteki jest umieszczenie (zapisanie) w niej aplikacji. Krok ten wykonuje się za pomocą kolejnego okna – **Save Application**. W polu *Applications* należy wpisać nazwę obiektu aplikacji (w omawianym przypadku obiekt aplikacji nazywa się tak samo jak biblioteka, mimo iż nie jest to konieczne). W polu *Comments* można wpisać dowolny komentarz. W polu *Application Libraries* powinna znajdować się nowo stworzona biblioteka „tutorial.pbl". Wygląd okna obrazuje rysunek 10.4.

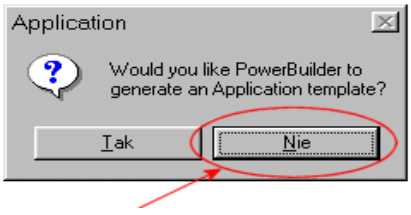


Rys. 10.3. Okno wyboru folderu i nazwy biblioteki (nazwa wybrana: Tutorial)



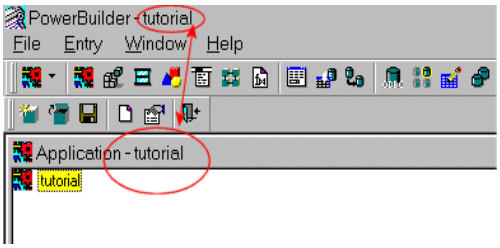
Rys. 10.4. Zapisywanie nowej aplikacji (przycisk OK)

W odpowiedzi na polecenie zapisu nowej aplikacji PowerBuilder za pomocą okna komunikatu proponuje skorzystanie z kreatora szablonów dla aplikacji. Jeśli aplikacja jest konstruowana „od zera", ścieżka ta nie powinna być wykorzystywana (rys. 10.5).



Rys 10.5. Okno komunikatu
umożliwiające generowanie szablonów aplikacji

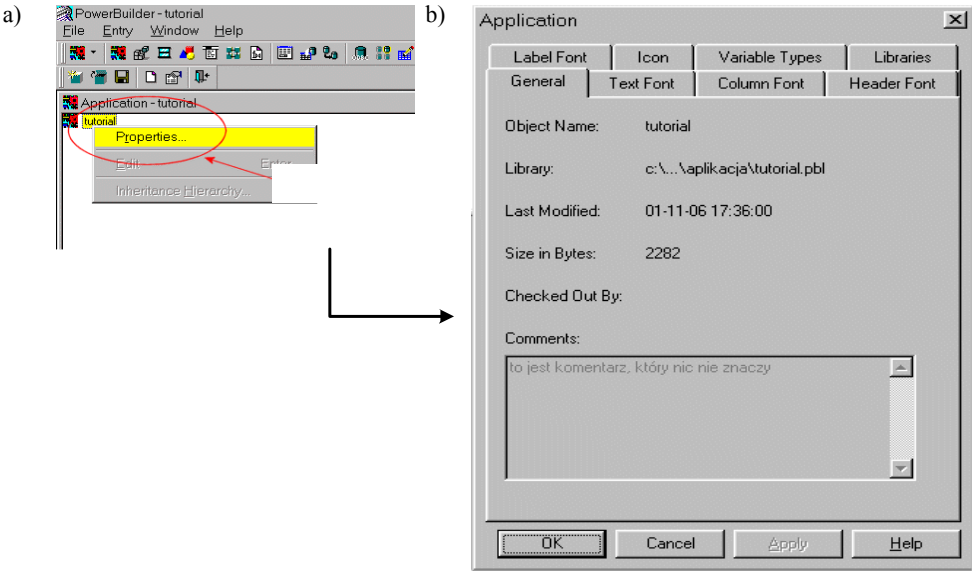
Po utworzeniu biblioteki o nazwie tutorial.pbl oraz aplikacji o tej samej nazwie znajdującej się w bibliotece, aplikacja tutorial staje się tzw. aplikacją bieżącą, co widać u góry ekranu (rys. 10.6).



Rys. 10.6. Widok ekranu z aplikacją bieżącą

Dostosowanie właściwości aplikacji do własnych potrzeb w zakresie ustawienia:

- domyślnych czcionek do wyświetlania nagłówków, kolumn tekstu i etykiet podczas pracy z tabelami,
- komentarza do aplikacji,
- ikony aplikacji,
- lokalizacji plików bibliotek, jeżeli składniki aplikacji zostały rozbite na kilka plików, odbywa się poprzez kliknięcie prawym przyciskiem myszy na obiekcie aplikacji, co powoduje wywołanie menu podręcznego, z którego należy wybrać pozycję *Properities* (rys. 10.7a), a następnie skorzystać z opcji uwidocznionych w karcie właściwości (rys. 10.7b).

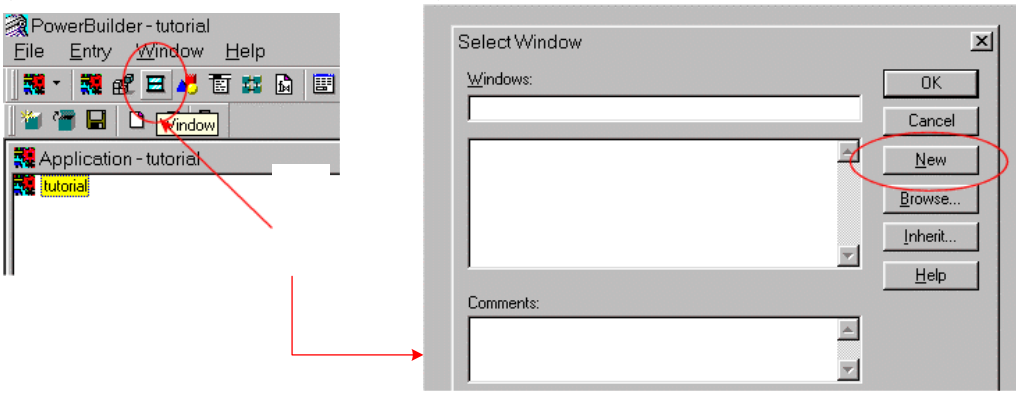


Rys. 10.7. Ustawianie właściwości aplikacji: a) wywołanie menu podręcznego, b) okno wyboru właściwości aplikacji

10.1.3. Tworzenie okna w aplikacji

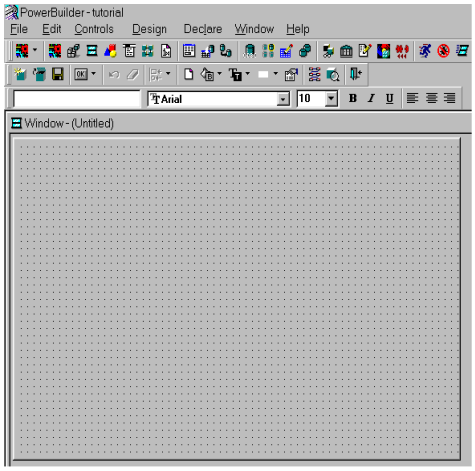
Tworzenie okna jest następnym etapem konstruowania aplikacji. W aplikacji nie ma jeszcze żadnych okien, należy więc uruchomić edytor okien przyciskiem Window Painter znajdującym się na belce (rys. 10.8a) i następnie wybrać opcję New (rys. 10.8b), co umożliwi utworzenie nowego obiektu, który zostanie zapisany w bieżącym obiekcie aplikacji, czyli w bibliotece wybranej w „strefie" Application Libraries (tutorial.pbl). Celem działania jest utworzenie okna z przyciskiem polecenia umożliwiającym jego zamykanie – poprzez wywołanie odpowiedniego zdarzenia, powodującego uruchomienie skryptu zamykającego okno.

a) b)



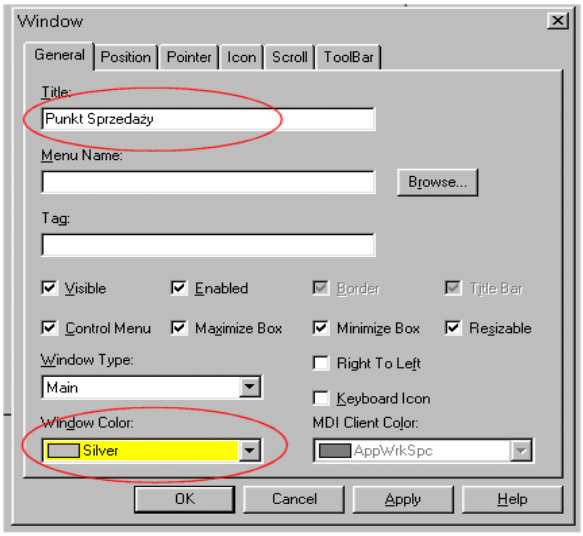
Rys. 10.8. Tworzenie okna: a) wybór edytora okien, b) okno definicji nowego obiektu

Wybór opcji pokazanych na rysunku 10.8 powoduje otwarcie obszaru roboczego edytora okien, w którym widoczny będzie tworzony obiekt (rys. 10.9).



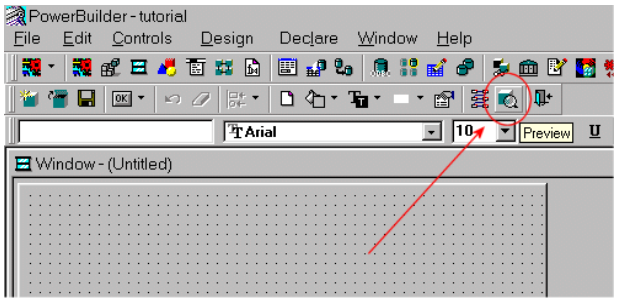
Rys. 10.9. Obszar roboczy edytora okien

Podobnie jak inne obiekty, okno posiada swoje właściwości, które można modyfikować, wywołując menu podręczne poprzez kliknięcie prawym przyciskiem myszki na obszarze roboczym okna i wybranie opcji *Properities*, co powoduje pojawienie się okna dialogowego (rys. 10.10) z kartą właściwości obiektu. Pierwszym krokiem powinno być nadanie nazwy obiektowi, czyli zmiana tytułu okna z „untitled" na własną nazwę, np. „Punkt sprzedaży", następnie – kierując się wytycznymi odnośnie do projektowania interfejsu – należy wybrać kolor tła okna, np. srebrny (*SILVER*). Zakończenie definiowania właściwości obiektu potwierdza przycisk OK.



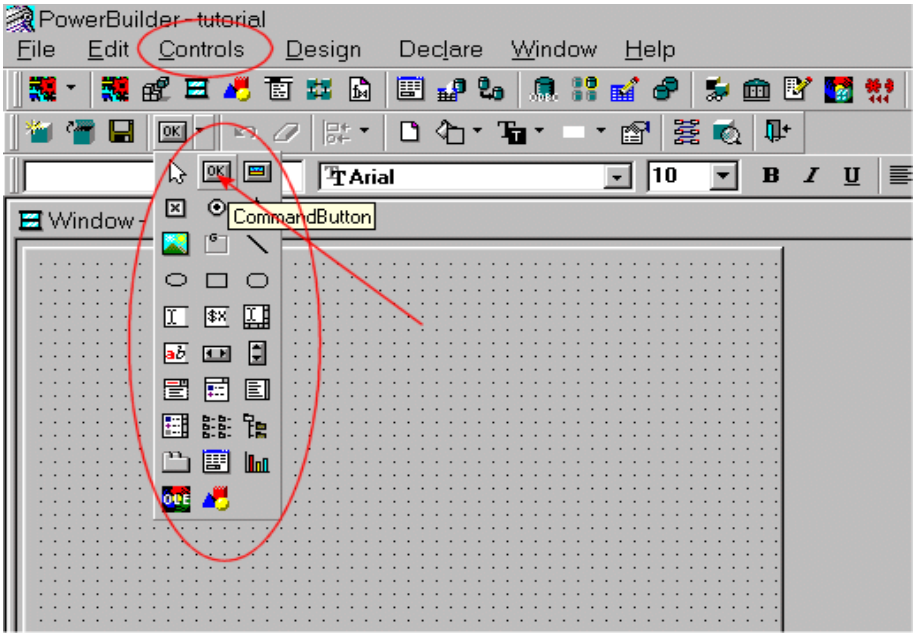
Rys. 10.10. Okno dialogowe do definiowania właściwości tworzonego obiektu – okna aplikacji

W każdym momencie pracy nad obiektem istnieje możliwość podglądu postaci graficznej obiektu poprzez wykorzystanie przycisku „lupy" (*PREVIEW*), znajdującego się na belce PowerBuildera (rys. 10.11).



Rys. 10.11. Lokalizacja przycisku podglądu obiektu

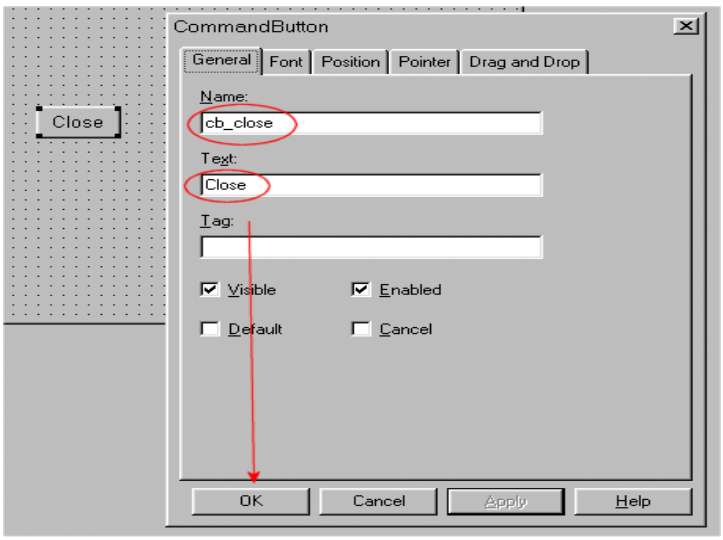
Zgodnie z zamierzonym celem, do tworzonego okna należy dodać przycisk zamykający (obiekt dialogowy). Wszystkie obiekty dialogowe, które można umieścić w oknie znajdują się na liście rozwijalnej na pasku narzędzi edytora lub w menu *Controls* (rys. 10.12).



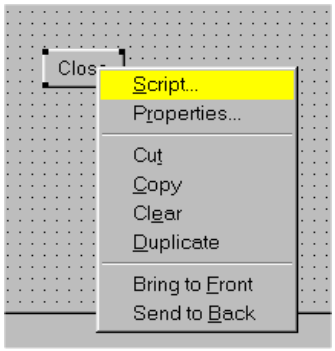
Rys. 10.12. Lista rozwijalna z obiektami dialogowymi

Umieszczenie przycisku dialogowego w polu okna polega na wyborze poprzez kliknięcie myszką żądanej ikony z listy rozwijalnej i „wklejenie" jej w obszar roboczy tworzonego okna (również poprzez kliknięcie myszką), lub wybór adekwatnej opcji z menu *Controls*. Nowy przycisk posiada swoje właściwości, które zmienia się tak, jak w przypadku innych obiektów. Po otwarciu karty właściwości dla przycisku należy przejść na zakładkę *General* w celu zmiany nazwy obiektu z *cb_1* (domyślna nazwa) na *cb_close*. Prefix *cb* pochodzi od słów CommandButton. Nadana nazwa *cb_close* będzie używana w odniesieniu do przycisku w skryptach pisanych dla tego przycisku. W polu *Text* wpisywany jest opis słowny, jaki ma być widoczny na przycisku.

Następnym krokiem jest dodanie skryptu dla tworzonego przycisku, którego wykonanie spowoduje zamknięcie okna. Otwarcie edytora skryptów następuje poprzez kliknięcie prawym klawiszem myszki w obiekt przycisku (rys. 10.14) lub poprzez użycie przycisku *Script* , na pasku *PainterBar* edytora *PowerScript Painter*.

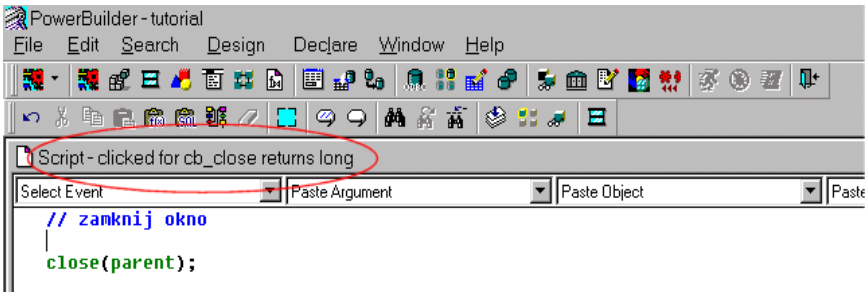


Rys. 10.13. Karta właściwości przycisku



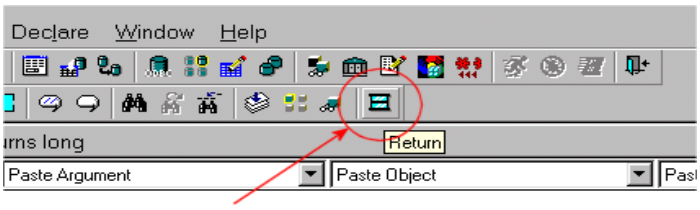
Rys. 10.14. Menu podręczne obiektu przycisk

Z listy rozwijanej *Select Event* w edytorze skryptów należy wybrać zdarzenie *clicked*. Wygląd belki tytułowej edytora obrazuje rysunek 10.15.



Rys. 10.15. Belka tytułowa edytora *Power Script Painter*

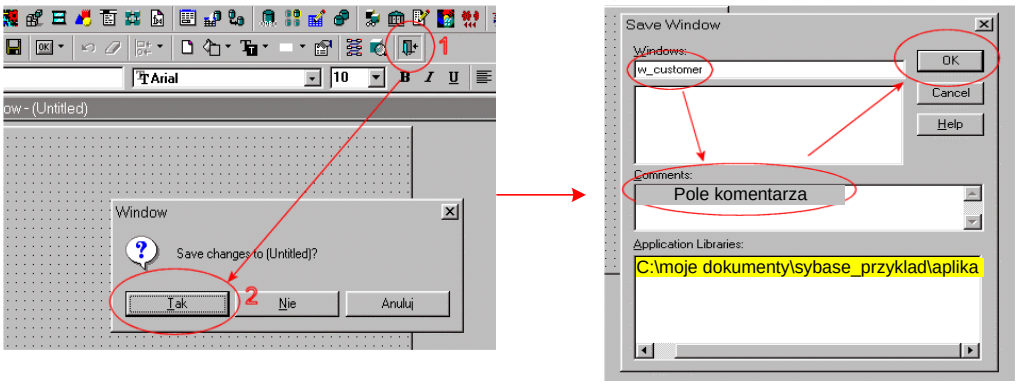
Tekst skryptu uwidoczniony na rysunku 10.15 powoduje zamknięcie okna rodzica, czyli okna, w którym znajduje się obiekt dialogowy. Komentarz zaczynający się znakiem // można, oczywiście, pominąć. Po zakończeniu wpisywania tekstu skryptu można użyć przycisk *Return* na pasku *PainterBar* (rys. 10.16) lub wybrać opcję *Return* z menu *File* w celu kompilacji skryptu.



Rys. 10.16. Położenie przycisku *Return* na pasku *PainterBar*

Ponieważ użycie przycisku *Return* powoduje kompilację skryptu, w przypadku błędów pojawią się komunikaty opisujące błędy; jeżeli skrypt jest poprawny, nastąpi powrót do edytora WindowPainter. Efekty pracy należy zapisać poprzez uruchomienie opcji *Save* z menu *File* lub kliknięcie ikony *Close* (rys. 10.17a), następnie w oknie dialogowym podanie nazwy obiektu, ewentualnego komentarza oraz ulokowania obiektu (rys. 10.17b).

a) b)

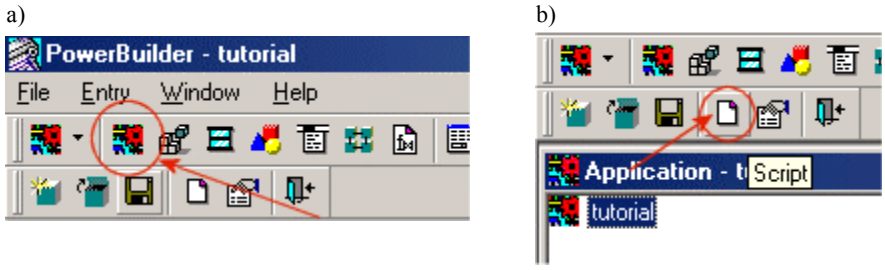


Rys.10.17. Zapisywanie obiektu: a) polecenie *Close*, b) okno definiujące sposób zachowania obiektu

10.1.4. Dodanie skryptu do obiektu aplikacji

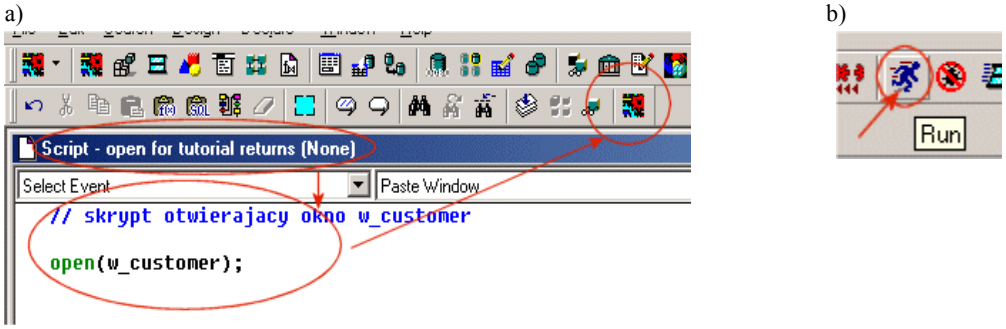
Aplikacja w dalszym ciągu nie może być uruchomiona, ponieważ nie dodano skryptu do obiektu aplikacji. Aplikacja wymaga istnienia odpowiednich procedur, które działają w trakcie jej uruchomiania. W tym celu należy dodać skrypt, który

spowoduje otwarcie nowo powstałego okna (*w_customer*), w trakcie uruchamiania aplikacji (zdarzenie *open* dla obiektu aplikacji). Dodanie skryptu odbywa się poprzez wywołanie edytora aplikacji przyciskiem *Application* na pasku PowerBar, co powoduje otwarcie edytora aplikacji, w którym poprzez przycisk *Script* uruchamiany jest edytor skryptów (rys 10.18a, b).



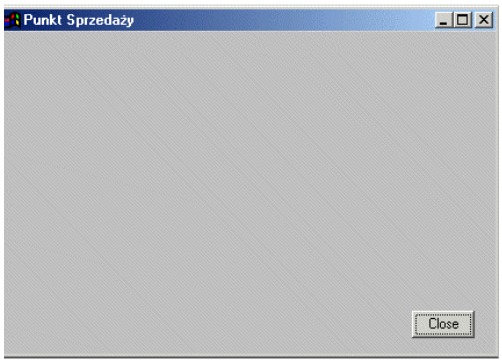
Rys. 10.18. Dodanie skryptu do aplikacji: a) wywołanie edytora aplikacji, b) uruchomienie edytora skryptów

W polu skryptu należy umieścić treść skryptu dla zdarzenia *Open* obiektu aplikacji (rys. 10.19a) i skompilować go w sposób opisany w podrozdziale 10.1.3, co umożliwi uruchamianie aplikacji poprzez przycisk Run na pasku PowerBar (rys. 10.19b) lub wciśnięcie klawiszy Ctrl + R.



Rys. 10.19. Skrypt otwierający aplikację: a) tekst skryptu, b) przycisk uruchamiający aplikację

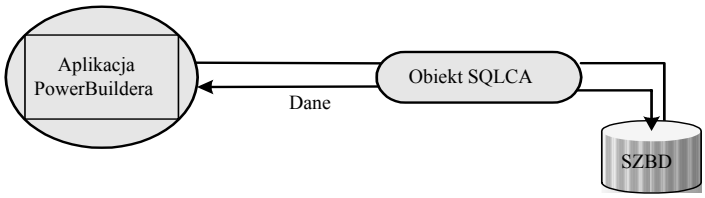
Zmiany w obiekcie aplikacji muszą zostać zapisane w sposób identyczny z opisanym w podrozdziale 10.1.3. Po uruchomieniu aplikacji wyświetlane będzie utworzone okno z przyciskiem umożliwiającym jego zamknięcie. Postać aplikacji obrazuje rys. 10.20.



Rys. 10.20. Wygląd budowanej aplikacji

10.1.5. Połączenie z serwerem baz danych w PowerBuilderze

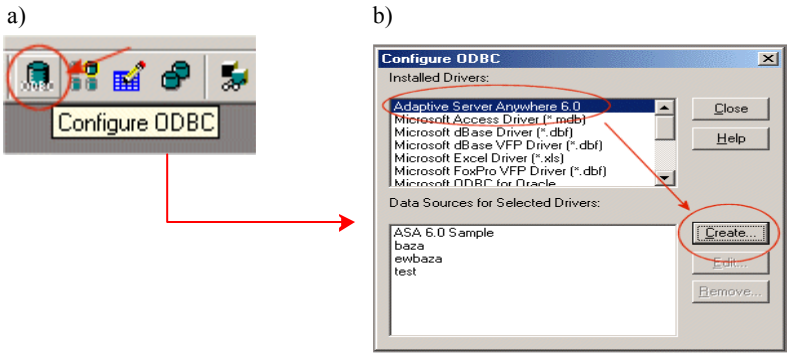
Aplikacja bazodanowa tworzona w środowisku PowerBuilder pobiera rekordy z bazy danych. Wcześniej jednak musi ona z tą ostatnią uzyskać „połączenie". Realizacja połączenia odbywa się poprzez tzw. profil bazy danych. Tworzenie profilu powoduje modyfikację pliku PB.INI, w którym zapisane są informacje o tym, z jaką bazą danych i w jaki sposób aplikacja się łączy. W środowisku PowerBuilder aplikacje wymagają oprócz zdefiniowania odpowiedniego interfejsu także zdefiniowania obszaru roboczego, tzw. obiektu transakcji (obiekt SQLCA, rys. 10.21). Obiekt transakcji zawiera informacje identyfikujące bazę danych oraz realizuje wymianę danych pomiędzy aplikacją a bazą [11]. Podczas tworzenia profilu bazy danych parametry definiujące obiekt SQLCA zostają zapisane do pliku PB.INI.



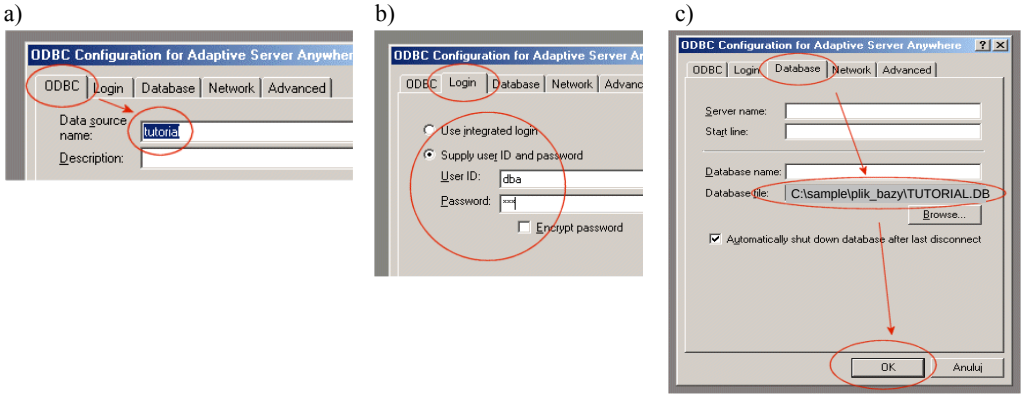
Rys. 10.21. Współpraca PowerBuildera z bazą danych

Tworzenie profilu bazy danych odbywa się według następującego algorytmu:

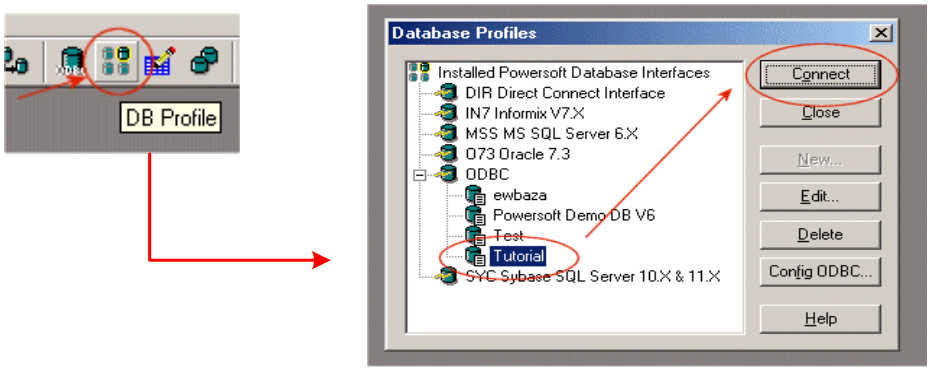
- Poprzez ikonę konfiguracji ODBC wywoływane jest okno dialogowe umożliwiające utworzenie nowego interfejsu ODBC (rys. 10.22a, b).
- Dalszym etapem jest konfiguracja ODBC, czyli wpisanie wybranej nazwy profilu (np. tutorial) jako źródła danych, podanie nazwy i hasła użytkownika oraz lokalizacji bazy danych (przycisk *Browse*). Opisane kroki ilustrują kolejno rysunki 10.23a, b, c.



Rys. 10.22. Tworzenie interfejsu ODBC: a) wywołanie okna dialogowego, b) polecenie tworzenia



Rys. 10.23. Konfiguracja ODBC: a) nazwa profilu, b) login użytkownika, c) plik bazy



Rys. 10.24. Przyłączanie bazy danych za pomocą zdefiniowanego profilu

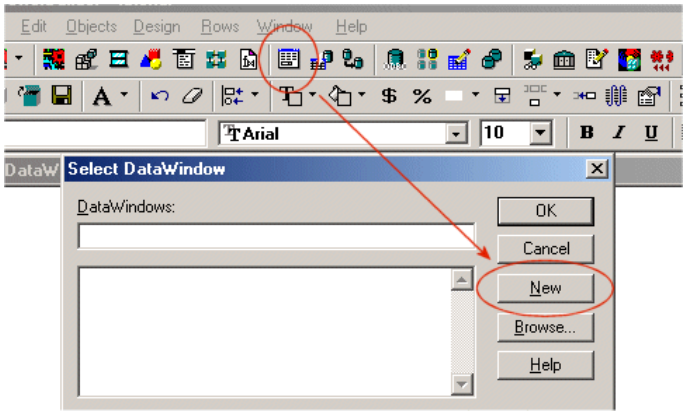
Zakończenie opisanej procedury powoduje określenie bazy bieżącej (domyślnej) dla programu PowerBuilder. Przy ponownym uruchomieniu tego środowiska programistycznego nie trzeba wykonywać powyższych operacji, chyba że zaistnieje potrzeba zdefiniowania innego źródła danych dla tworzonych aplikacji. Działanie utworzonego profilu (zdolność do przyłączenia bazy danych) można sprawdzić poprzez użycie ikony DB PROFILE i wybór utworzonego profilu w oknie dialogowym (rys. 10.24).

10.1.6. Tworzenie obiektu DataWindow

Obiekty DataWindow stanowią łącze z danymi w bazie danych; za ich pomocą można pobierać rekordy z tabel, aktualizować informacje w bazie, wyświetlać wyniki zapytań *SQL*, budować raporty itd.

Istnieją trzy sposoby komunikowania się aplikacji z bazą danych:

- wbudowanie instrukcji *SQL* bezpośrednio w skrypt (metoda stosowana w przypadku pisania aplikacji w takich językach jak COBOL czy C),
- przedstawianie wyników w postaci graficznej (DataWindow zapewnia taki mechanizm – zawiera wewnątrz zapytania *SQL*, sprecyzowane podczas projektowania),
- użycie obiektu DataStore (tego obiektu niniejszy podręcznik nie prezentuje – odsyłam wszystkich zainteresowanych do pozycji [11] lub materiałów publikowanych w Internecie).



Rys. 10.25. Wywołanie okna dialogowego definiującego nowy obiekt DataWindow

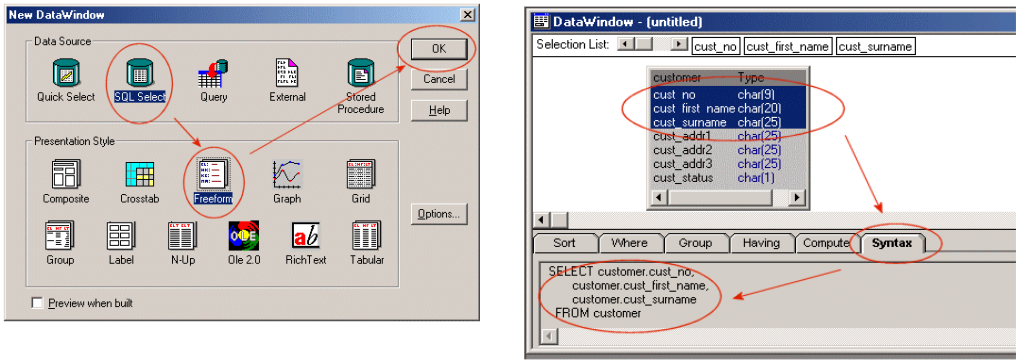
Istotę działania DataWindow najlepiej zobrazować na przykładzie. W celu utworzenia obiektu DataWindow (pobierającego i wyświetlającego dane z bazy danych) należy poprzez przycisk *Data Window* na pasku *PowerBar* wyświetlić okno *Select Data Window*. W oknie tym można modyfikować istniejące obiekty DW lub tworzyć

nowe. W przypadku tworzenia nowego obiektu należy przyciskiem *New* wywołać kolejne okno dialogowe: *New Data Window* (rys. 10.25).

Pierwsze okno dialogowe definiujące nowy obiekt DW jest podzielone na dwie sekcje: *Data Source* i *Presentation Style* (rys. 10.26a). W sekcji *Data Source* określa się sposób dostępu do danych (SQL, zapytanie (*Query*), procedurę itp.). W sekcji *Presentation Style* należy wybrać sposób prezentacji danych, np. formularz (*Freeform*), etykiety (*Label*), tabularyczny (*Tabular*), pełnotekstowy (*Rich Text*).

Kolejne okno (*Select Tables*) umożliwia określenie tabel, z których będą pobierane informacje do obiektu DW. Po wybraniu żądanej tabeli zostanie wyświetlona lista wszystkich jej kolumn. Należy wybrać kolumny, z których dane będą prezentowane w formularzu. Zakładka *Syntax* umożliwia podgląd dynamicznie tworzonego zapytania *SQL* (rys. 10.26b).

a) b)



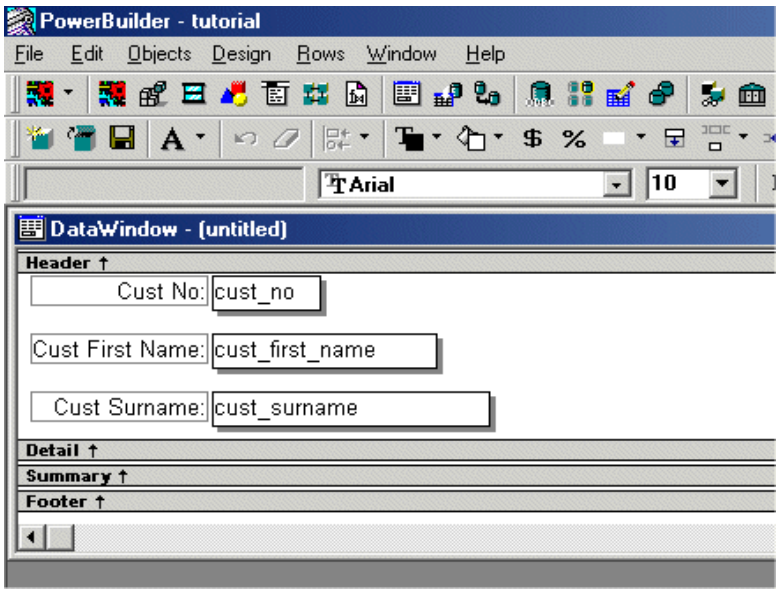
Rys. 10.26. Okna dialogowe definiujące: a) źródło danych i styl prezentacji, b) tabele i kolumny

Przycisk *SQL* na pasku Painter Bar 

umożliwia przełączenie na tryb projektowania sposobu prezentacji wyników. Kolumny wybrane poprzednio są rozmieszczane w formularzu zgodnie ze stylem prezentacji *FreeForm*. Obszar roboczy jest podzielony na następujące obszary (rys. 10.27):

- nagłówek (*Header*),
- szczegóły (*Detail*),
- podsumowanie (*Summary*),
- stopka (*Footer*).

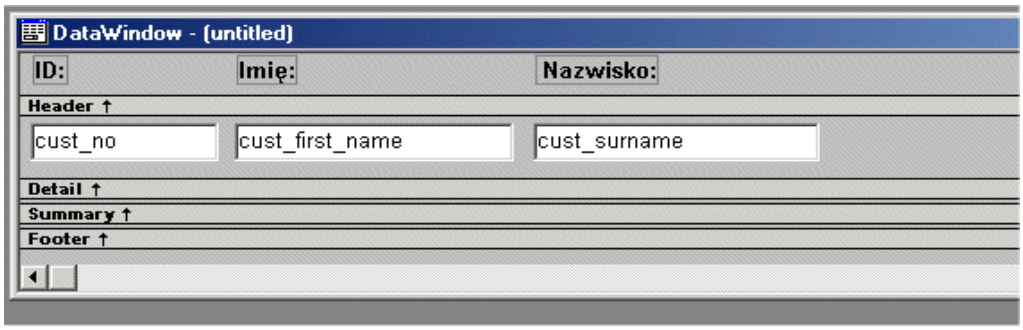
Wielkość każdego z pasm może być dowolnie modyfikowana za pomocą lewego klawisza myszki. Tekst umieszczony w nagłówku pojawi się u góry ekranu lub strony. Tekst ten zostanie umieszczony tylko raz. Każdy obiekt (tekst, obrazek) z pasma szczegółów zostanie powtórzony dla każdego pobranego rekordu tabeli. Pozostałe pasma są zwykle stosowane do wyliczania sum pośrednich i całkowitych.



Rys. 10.27. Widok formularza w trybie projektowym

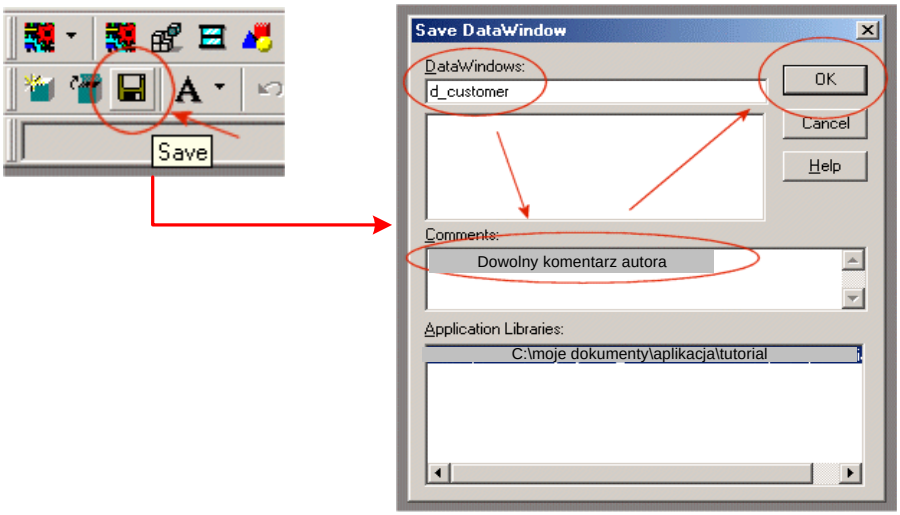
Przycisk  (*Preview*) umożliwia podgląd obiektu DataWindow z przykładowymi danymi.

Pola kolumn dla stylu *FreeForm* widoczne są w postaci cieniowanej ramki. Tekst etykiet jest wyświetlany za pomocą szarej ramki.



Rys. 10.28. Wygląd obiektu DataWindow z przykładowymi danymi

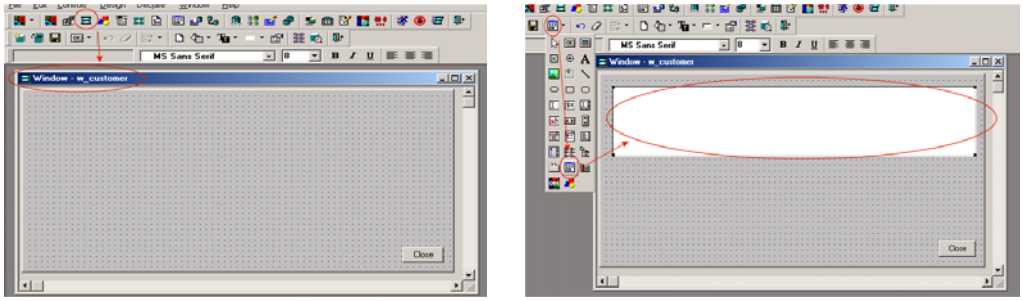
Po zakończeniu definiowania obiektu należy go zapisać w standardowy dla środowiska PowerBuilder sposób, czyli poprzez przycisk *Save* i okno dialogowe *Save Data Window* (rys. 10.29a i b).



Rys. 10.29. Zapisywanie obiektu: a) przycisk *Save*, b) okno dialogowe

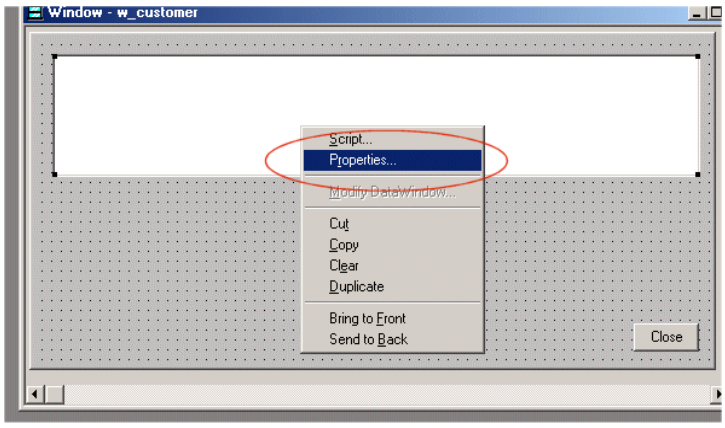
Kolejny etap polega na dodaniu obiektu *DataWindow* do utworzonego wcześniej okna. Obiekty DW nie są umieszczane bezpośrednio w oknie, zamiast tego w oknie umieszczany jest obiekt dialogowy DW, z którym potrzebny obiekt DW musi zostać związany. Połączenie tworzonych obiektów odbywa się poprzez edytor WindowPainter (rys. 10.30a). Obiekt dialogowy DataWindow znajduje się na liście rozwijalnej obiektów dialogowych (rys. 10.31b).

a) b)



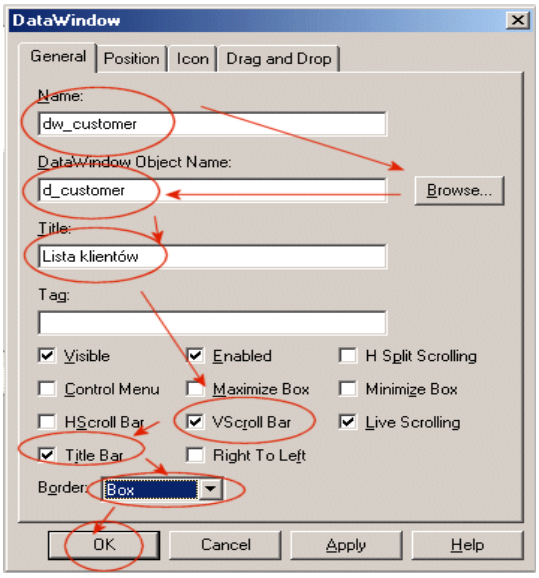
Rys. 10.30. Proces łączenia okna z obiektem DW: a) widok okna w edytorze Window Painter, b) widok okna z obiektem dialogowym

Wywołanie dialogu umożliwiającego połączenie obiektu dialogowego z obiektem DataWindow realizowane jest poprzez kliknięcie prawym przyciskiem myszki na obiekt dialogowy (wywołanie menu podręcznego, opcja *Properties*) (rys. 10.31).



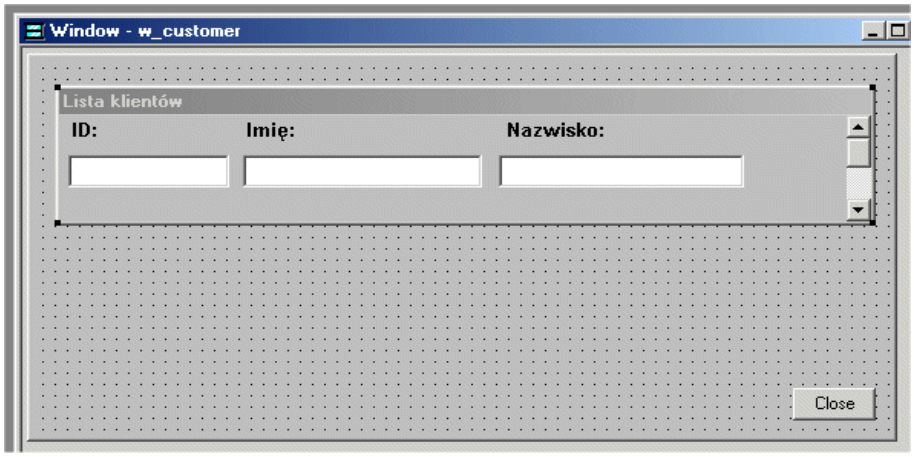
Rys. 10.31. Menu podręczne, opcja *Properties*

Poprzez opcję *Properties* wywoływane jest okno dialogowe, którego zakładka *General* umożliwia nadanie obiektowi dialogowemu odpowiednich właściwości (rys. 10.32). W odniesieniu do pola DataWindow Object Name warto skorzystać z przycisku *Browse*, uwidaczniającego wszystkie obiekty DataWindow znajdujące się w bibliotece.



Rys. 10.32. Zakładka *General* okna dialogowego definiującego właściwości obiektu dialogowego

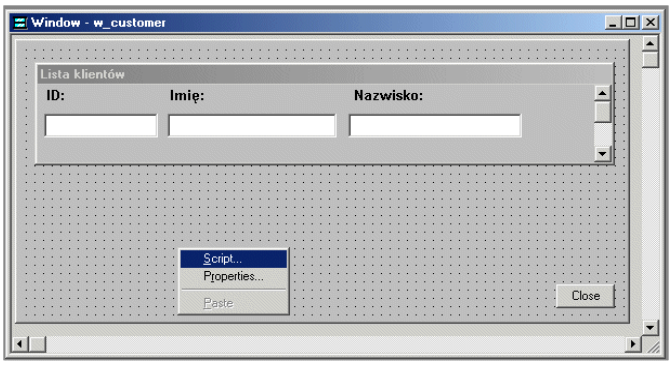
Po zakończeniu definiowania obiekt DataWindow pojawi się na obiekcie dialogowym DataWindow (rys. 10.33). Początkowo nie będzie zawierał danych, ponieważ nie nastąpiło połączenie z bazą danych, lecz zakończone zostały jedynie prace projektowe i edytorskie.



Rys. 10.33. Obiekt DataWindow umieszczony w obiekcie dialogowym okna

W celu nawiązania współpracy z bazą danych należy użyć obiektu transakcji (SQLCA), aby przekazać potrzebne informacje z bazy do obiektu DataWindow. Niezbędnym krokiem jest więc napisanie dodatkowego skryptu, który będzie pobierał informacje z bazy danych, i umieszczenie go w zdarzeniu *open* okna. Rodzaj pobieranych danych został określony podczas projektowania DataWindow.

W przypadku, gdy dane powinny być wyświetlane w momencie otwarcia okna, należy związać instrukcję *Retirieve*() ze zdarzeniem otwierania okna, czyli odpowiednio zmodyfikować skrypt dla zdarzenia *open* okna. Modyfikacji skryptu dokonuje się poprzez wywołanie edytora skryptów prawym klawiszem myszki (rys. 10.34) i wybranie opcji *Open* z listy rozwijalnej zdarzeń (*Select Event*).

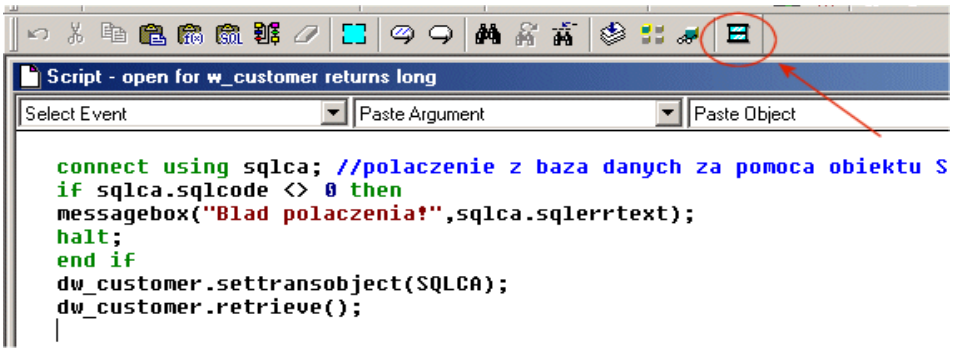


Rys. 10.34. Wywołanie edytora skryptów

Zawartość skryptu przyłączającego bazę danych za pomocą instrukcji *CONNECT* i obiektu SQLCA oraz instrukcji *Retrieve* pobierającej dane jest następująca:

```
connect using sqlca; //polaczenie z baza danych za pomoca
obiektu SQLCA
if sqlca.sqlcode <> 0 then
messagebox("Blad polaczenia!",sqlca.sqlerrtext);
halt;
end if
nazwa_obiektu_DataWindow.settransobject(SQLCA);
nazwa_obiektu_DataWindow.retrieve();
```

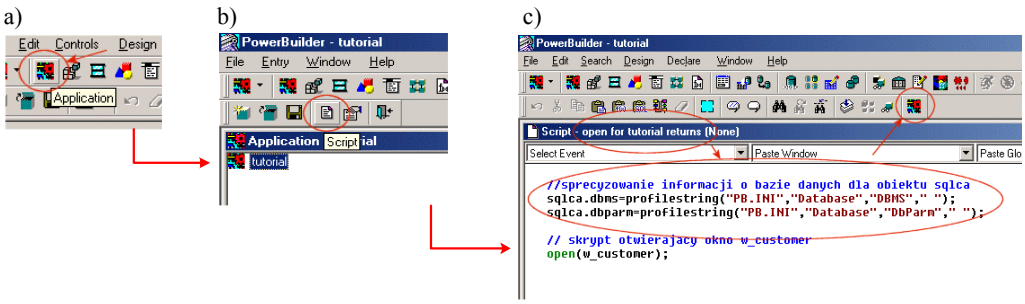
Jeśli w danym oknie istnieje kilka obiektów DataWindow, to każdy z nich musi mieć przypisaną instrukcję *Retrieve*. Skrypt zawiera również obsługę błędu przy uzyskiwaniu połączenia z bazą danych: jeżeli połączenie nie uda się, zostaje wyświetlony odpowiedni komunikat informacyjny. Kompilacja skryptu odbywa się poprzez przycisk Return.



Rys. 10.35. Ekran edytora skryptów ze skryptem przyłączającym bazę danych i instrukcją *Retrieve*

Ostatnim etapem jest określenie, z jakiej bazy danych ma korzystać aplikacja oraz podanie jej lokalizacji. Przywołując opis tworzenia profilu bazy danych – podczas tworzenia profilu pewne parametry (typ systemu DBMS oraz informacje o połączeniu) zostały umieszczone w pliku konfiguracyjnym PB.INI, stamtąd też powinna pobrać je aplikacja. Ponieważ, z założenia, w podręczniku omówiono jednorodne środowisko bazodanowe Sybase, dla uzyskania połączenia z bazą danych tylko dwa parametry muszą być pobrane z pliku PB.INI, a mianowicie *DBMS* i *Dbparam*. Parametry te należy przypisać obiektowi transakcji SQLCA, poprzez skrypt związany ze zdarzeniem *open* obiektu aplikacji. Modyfikacja skryptu otwierającego obiekt aplikacji przebiega w standardowy sposób: wywołanie edytora aplikacji, następnie edytora skryptów, wpisanie zmian do skryptu zdarzenia *open* obiektu aplikacji i następnie skompilowanie skryptu (rys. 10. 36a, b, c). Pobranie przez PowerBuilder parametrów połączeniowych odbywa się poprzez funkcję *ProfileString*(). Treść linii skryptu powodująca pobranie parametrów połączeniowych:

```
SQLCA.DBMS = ProfileString ("PB.INI","Database","DBMS", " ")
SQLCA.DBParm = ProfileString ("PB.INI","Database","DbParm"," ")
```



Rys. 10.36. Modyfikacja skryptu otwierającego obiekt aplikacji: a) wywołanie edytora aplikacji, b) wywołanie edytora skryptów, c) zmodyfikowana treść skryptu

Po poprawnym zakończeniu wszystkich etapów konstruowania aplikacji można przystąpić do jej uruchomienia (przycisk Run na pasku narzędzi) oraz testowania. Po uruchomieniu aplikacji w oknie pokazanym na rys. 10.34 powinny pojawić się dane. Uruchomienie aplikacji powoduje wykonanie (hierarchicznie) szeregu zdarzeń obsługiwanych przez przypisane im skrypty:

1. Wyzwolenie zdarzenia *open* aplikacji. W tym zdarzeniu ładowane są określone parametry do obszaru roboczego SQLCA, a także otwierane jest okno główne aplikacji.

2. Zdarzenie poprzednie wyzwala kolejne zdarzenie *open* dla obiektu DataWindow, w który łączy się z bazą danych za pomocą SQLCA, a następnie poprzez skrypt związany z obiektem pobierane są informacje z bazy danych.

Wykonana aplikacja jest bardzo prosta, ponieważ celem rozdziału było zobrazowanie sposobu pracy w środowisku PowerBuilder oraz idei programowania sterowanego zdarzeniami. Nie zostały, ze zrozumiałych względów, przedstawione pełne możliwości narzędzia – zarysowany szkielet aplikacji powinien być dalej rozwijany poprzez dodanie kolejnych DataWindow z możliwościami wyszukiwania danych według określonych kryteriów, wyszukiwania danych z tabel połączonych, możliwościami aktualizacji danych czy tworzenia raportów i ich drukowania, co wiąże się z koniecznością poznania języka skryptowego PowerScript. Niebagatelną sprawą jest również dobrze dostosowany do potrzeb użytkownika interfejs graficzny oraz sposób obsługi błędów. Projektantom i programistom, których zaprezentowane w dużym skrócie środowisko PowerBuilder zainteresowało, odsyłam do pogłębienia swojej wiedzy według wskazówek zawartych w pracy [11]. Według tych wskazówek opracowana została aplikacja prezentowana w następnym podrozdziale.

10.2. Przykładowa aplikacja w środowisku PowerBuilder

Przytoczony przykład aplikacji zrealizowanej za pomocą PowerBuildera jest kontynuacją przykładu z rozdziałów wcześniejszych, a mianowicie systemu serwisowania kas fiskalnych (SSFK), którego model zarówno w zakresie funkcjonalnym, jak i danych został szczegółowo przedstawiony. Na podstawie modelu systemu i modelu danych przyjęto następujące założenia implementacyjne:

- Architektura klient–serwer z możliwością pracy jednostanowiskowej (strona klienta i strona serwera na jednym komputerze).
- Serwer bazy danych – Adaptive Server Anywhere.
- Środowisko aplikacji – PowerBuilder.

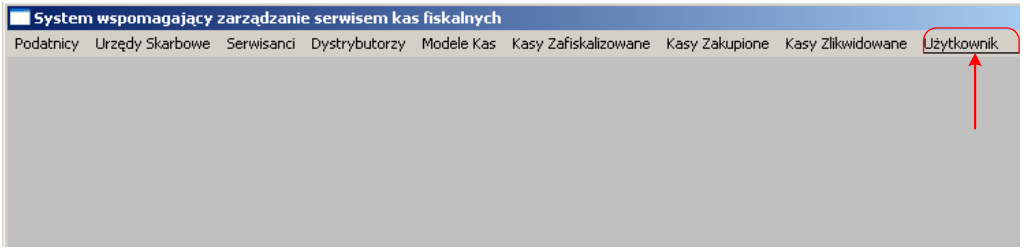
Założenia funkcjonalne

Aplikacja została zrealizowana w formie okna z menu wybieralnym, podzielonym na dziewięć kategorii:

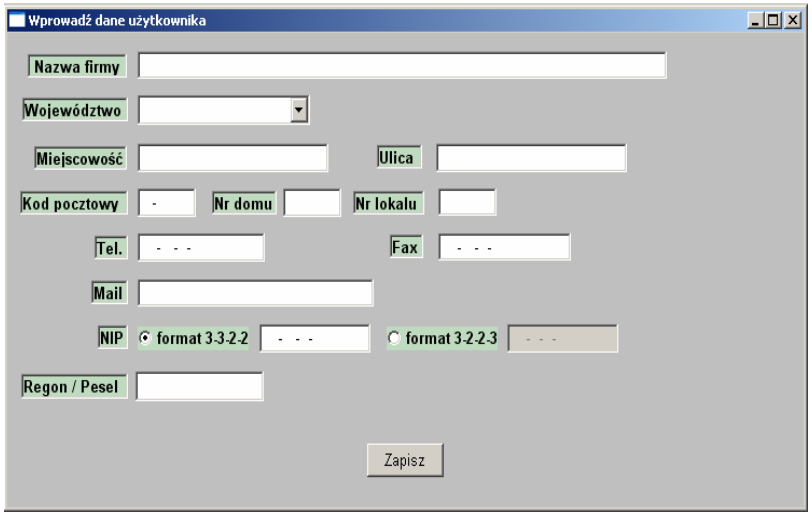
1. Dane podatników:
   1.1. Otwórz dane podatników.
   1.2. Dodaj nowego podatnika.
2. Urzędy Skarbowe:
   2.1. Otwórz dane Urzędów.
   2.2. Dodaj nowy Urząd.
3. Serwisanci:
   3.1. Otwórz dane serwisantów.
   3.2. Dodaj nowego serwisanta.
   3.3. Dodaj uprawnienia dla serwisanta.
4. Dystrybutorzy kas:
   4.1. Otwórz dane dystrybutorów.
   4.2. Dodaj nowego dystrybutora.
5. Modele kas:
   5.1. Otwórz dane kas.
   5.2. Dodaj nowy model.
6. Kasy zafiskalizowane:
   6.1. Otwórz dane kas.
   6.2. Likwidacja kasy.
   6.3. Przegląd techniczny kasy.
   6.4. Otwórz przeglądy techniczne.
   6.5. Kasy naprawiane.
   6.6. Kasy naprawione.
   6.7. Zmień serwisanta kasy.
7. Kasy zakupione:
   7.1. Otwórz dane kas.

7.2. Fiskalizacja kasy.
7.3. Dodaj nową kasę.
8. Kasy zlikwidowane:
8.1. Otwórz dane kas.
9. Użytkownik:
9.1. O aplikacji.
9.2. Użytkownik systemu.
9.3. Zakończenie pracy.

Początek pracy aplikacji wiąże się z koniecznością wprowadzenia danych użytkownika korzystającego z systemu, po czym można rozpocząć pracę z systemem. Menu główne aplikacji przedstawiono na rysunku 10.37, a okno służące do wprowadzenia danych użytkownika na rys. 10.38.



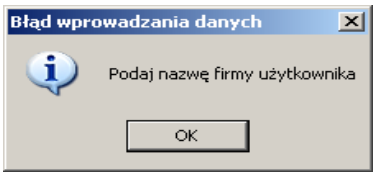
Rys. 10.37. Menu główne aplikacji



Rys. 10.38. Okno wprowadzające dane użytkownika systemu

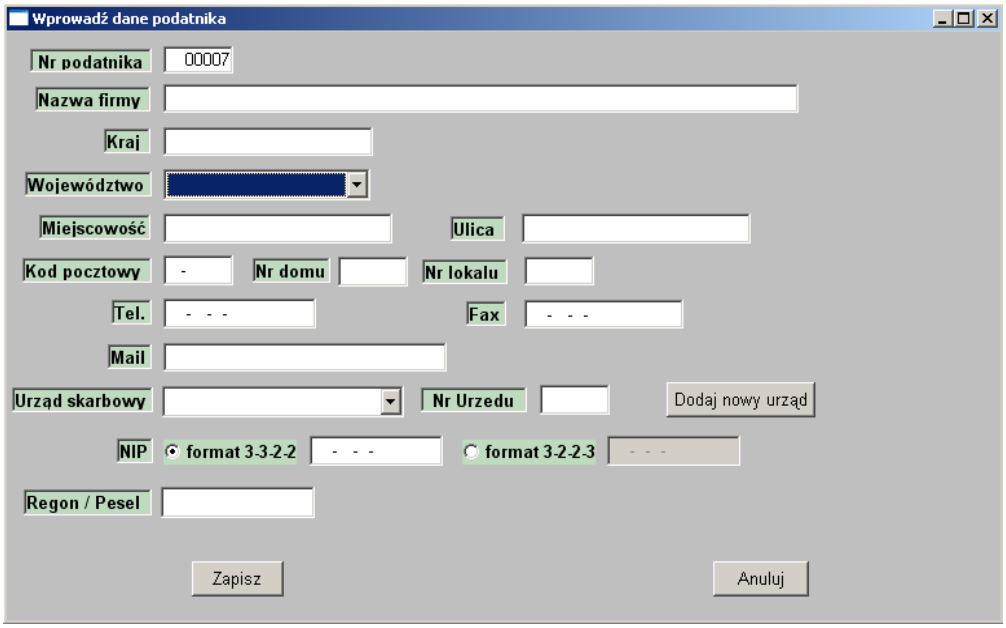
Po wprowadzeniu danych użytkownika należy kolejno wprowadzić pozostałe dane, tzn. dane podatników, urzędów skarbowych, serwisantów, modeli kas i dystrybu-

torów. W formularzach służących do wprowadzania danych zastosowano możliwość korzystania z list rozwijalnych, co znacznie ułatwia obsługę systemu. Każde z okien służących do wprowadzania danych posiada przycisk *Zapisz* i *Anuluj*. Przed zrealizowaniem operacji zapisu dane są sprawdzane na poziomie aplikacji, a następnie przez serwer bazy danych. W przypadku stwierdzenia błędu użytkownik jest informowany stosownym komunikatem (rys. 10.39) i operacja zapisu zostaje wycofana. Obsługa błędów została zrealizowana na poziomie aplikacji, ze względu na możliwości opracowania bardziej zrozumiałych dla użytkownika treści komunikatów.



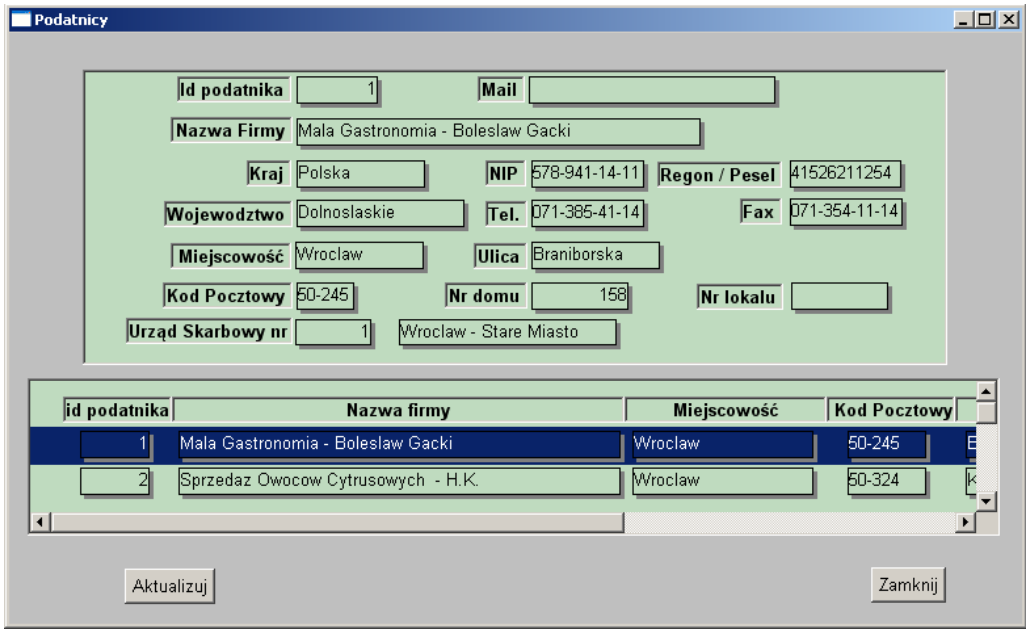
Rys. 10.39. Przykład komunikatu o błędzie wprowadzania danych podatnika

Okno umożliwiające wprowadzenie danych podatnika wywoływane jest z menu *Podatnicy* poprzez wybór opcji *Dodaj nowego podatnika*. Numer systemowy, pełniący rolę klucza głównego nadawany jest automatycznie, bez możliwości edycji. Użytkownik ma możliwość wybrania formatu numeru NIP (dostępne formaty: 3-3-2-2 oraz 3-2-2-3). Okno formularza wyposażone jest w przycisk realizujący zapis danych do bazy oraz w przycisk anulujący operację (rys. 10.40).



Rys. 10.40. Okno służące do wprowadzania danych podatnika

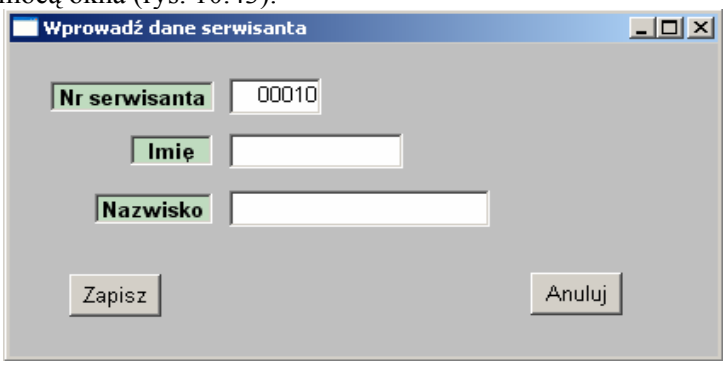
Informacje o podatnikach, których dane znajdują się w systemie są dostępne z poziomu menu *Podatnicy* poprzez opcję *Otwórz dane podatników*. Formularz zbudowany jest z dwóch okien danych (dwa obiekty DataWindow). Okno górne wyświetla szczegółowe dane wybranego podatnika, w oknie dolnym widnieje lista wszystkich podatników; wybór podatnika następuje po kliknięciu myszką na dowolny wiersz okna dolnego (rys. 10.41). Z poziomu okna wyświetlającego dane podatników można poprzez przycisk *Aktualizuj* przejść do aktualizacji danych. Okno aktualizacji danych wyglądem niewiele się różni od okna wprowadzającego dane. W odniesieniu do niektórych pól (NIP, dane urzędu skarbowego podatnika w przypadku, gdy przynależą do niego niezlikwidowane kasy fiskalne, kraj), jako mechanizm utrzymujący bazę w stanie spójnym zastosowano blokadę aktualizacji. Podobne mechanizmy zastosowano względem danych o urzędach skarbowych, dystrybutorach kas oraz użytkowniku systemu.



Rys. 10.41. Okno prezentujące zbiór podatników

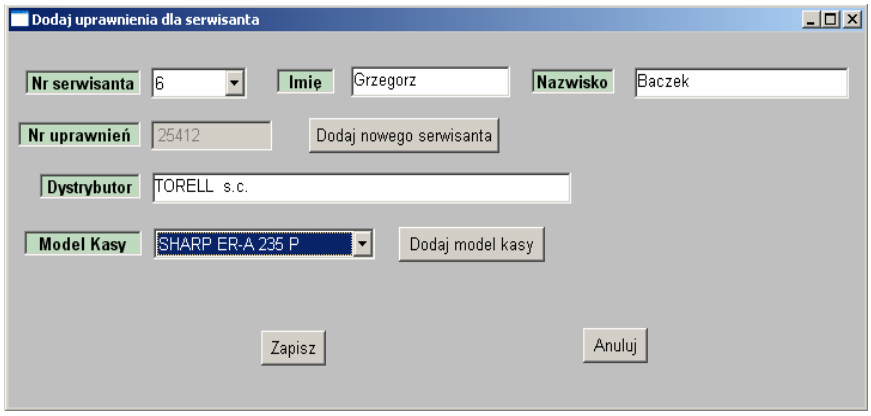
Zapis pełnych danych serwisantów, tzn. dane personalne oraz uprawnienia nadane przez dystrybutora kas, jest realizowany dwuetapowo. Etap pierwszy, polegający na wprowadzeniu do bazy danych imienia i nazwiska serwisanta, realizowany jest poprzez menu *Serwisanci* i opcję *Dodaj nowego serwisanta*. Okno umożliwiające wpisanie danych serwisanta przedstawiono na rys. 10.42. W etapie drugim określonemu serwisantowi przypisywane są odpowiednie uprawnienia nadane przez dystrybutora

kas na konkretny model lub modele kas (opcja *Dodaj uprawnienia dla serwisanta*), również za pomocą okna (rys. 10.43).



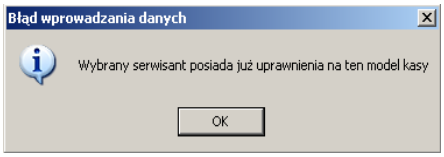
Rys. 10.42. Okno do wprowadzania danych serwisantów

Podczas przypisywania uprawnień konkretnemu serwisantowi jego dane wybierane są z listy rozwijalnej (dane bez możliwości edycji). Z poziomu okna dodawania uprawnień można wrócić do poziomu dodawania serwisantów (przycisk *Dodaj nowego serwisanta*) oraz do poziomu dodawania modelu kasy (przycisk *Dodaj model kasy*). Oczywiście oba przypadki są realizowane jako samodzielne opcje menu (patrz menu główne aplikacji, rys. 10.37).



Rys. 10.43. Okno dodawania uprawnień dla serwisanta

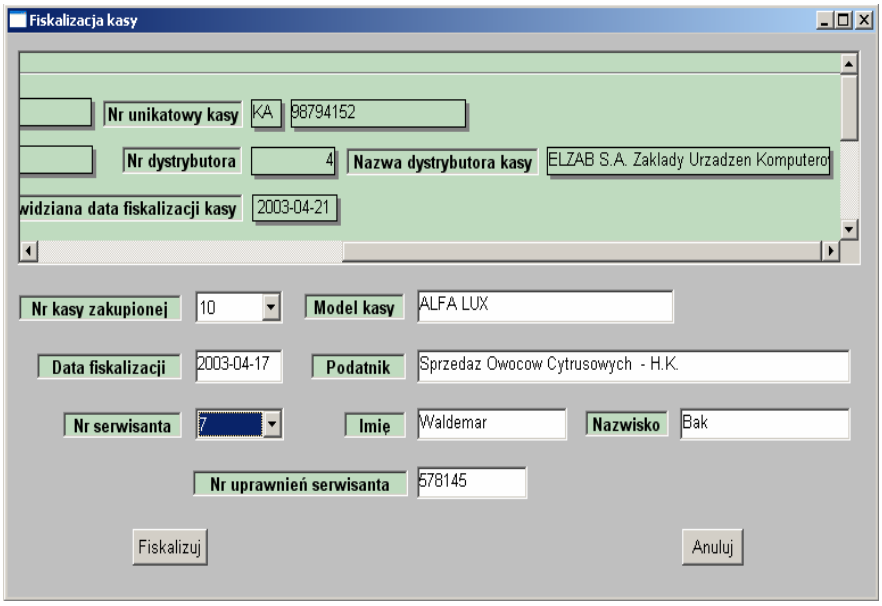
Pole okna zaetykietowane *Dystrybutor* jest wypełniane automatycznie po wyborze z listy rozwijalnej modelu kasy. Jeśli serwisant posiada już określone uprawnienia, to aplikacja generuje komunikat błędu (rys. 10.44).



Rys. 10.44. Komunikat o błędzie wprowadzania danych

Dane wszystkich serwisantów prezentowane są w podobny sposób jak dane podatników, tzn. poprzez okno zawierające dwa obiekty DataWindow. Obiekt górny wyświetla dane wybranego serwisanta, dolny natomiast listę wszystkich serwisantów.

Jedną z ważniejszych funkcji aplikacji jest umożliwienie fiskalizacji zakupionych kas. Okno fiskalizacji kas posiada jeden obiekt DataWindow, w którym wyświetlane są dane wybranej kasy, lista wyboru kas zakupionych oraz lista wyboru serwisantów (rys. 10.45).

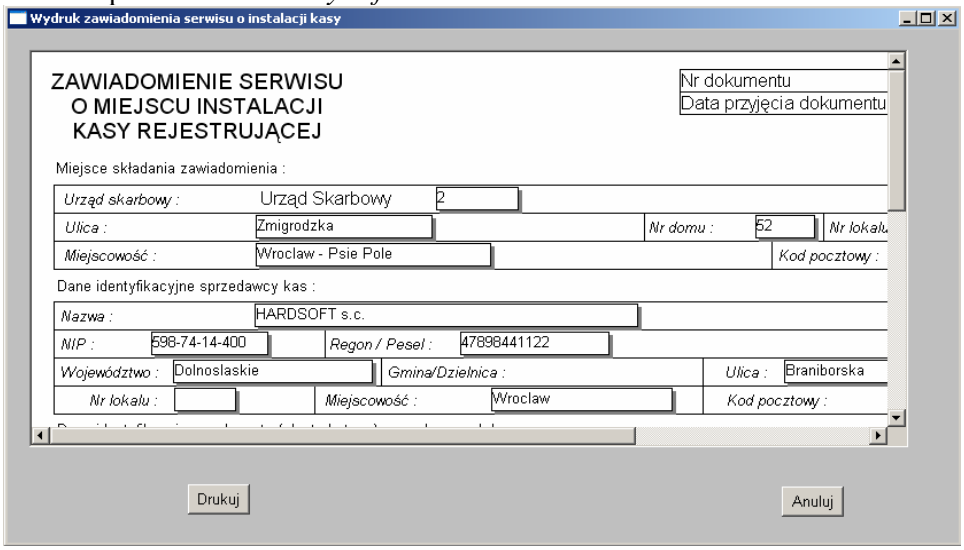


Rys. 10.45. Fiskalizacja zakupionej kasy

Fiskalizacji kasy musi dokonać serwisant posiadający uprawnienia na dany model kasy, w przeciwnym przypadku system nie pozwoli na zapisanie operacji. Ponieważ operacja jest nieodwracalna, przed jej zatwierdzeniem generowane jest okno komunikatu z żądaniem potwierdzenia zapisu. Po zatwierdzeniu operacji kasa zostaje zafiskalizowana; przestaje być widoczna w oknie *Kasy zakupione*, jej dane będą udostępniane w oknie *Kasy zafiskalizowane*. Serwisant, który dokonał fiskalizacji kasy staje się

automatycznie serwisantem uprawnionym do dokonywania przeglądów technicznych oraz napraw tej kasy, chyba że poprzez opcję *Zmień serwisanta* nastąpi taka zmiana.

Równocześnie z potwierdzeniem fiskalizacji otwarte zostają okna wydruku zawiadomień dla urzędu skarbowego ze strony podatnika i serwisu, zawierające przyciski *Drukuj* i *Anuluj* (rys. 10.46). Wszystkie dokumenty drukowane przez system są zgodne z wymaganiami urzędów i przepisów. Możliwość wydruku zawiadomień dostępna jest także z poziomu okna *Kasy zafiskalizowane*.



Rys. 10.46. Wydruk zawiadomienia o miejscu zainstalowania kasy

W podobny sposób zrealizowane zostały pozostałe funkcje aplikacji. Aplikacja składa się z jednego menu wybieralnego, 41 obiektów DataWindows i 47 niezależnych okien, kwalifikuje się więc do klasy systemów niedużych. Niemniej jednak założenia funkcjonalne zostały zrealizowane w sposób czytelny, system ma jeszcze jedną zaletę wynikającą z wybranej techniki realizacji i środowiska implementacyjnego – łatwość rozbudowy i modyfikacji. Przystępując do realizacji systemu wykonawca musiał nie tylko opanować zagadnienia związane z projektowaniem baz danych, ale również poznać obiektowe środowisko PowerBuildera, a szczególnie język skryptowy PowerScript, oraz techniki związane z projektowaniem i realizacją interfejsu GUI.

Literatura

[1] BARKER R., *CASE Method. Modelowanie funkcji i procesów*, WNT, Warszawa 1996.
[2] BARKER R., *CASE Method. Modelowanie związków encji*, WNT, Warszawa 1996.
[3] BATINI C., CERI S., NAVATHE S., *Conceptual Database Design: An Entity-Relationship Approach,* Benjamin Cummings, Redwood City CA, 1992.
[4] BEN-NATHAN R., *Objects on the Web: Designing, Building, and Deploying Object-Oriented Applications for the Web*, McGraw-Hill, New York 1997.
[5] BEYNON-DAVIES P., *Networks*, *Systemy baz danych,* WNT, Warszawa 1998.
[6] CELKO J., *Joe Celko's Data & Databases: Concepts in Practice*, Morgan Kaufmann, San Francisco, CA 1999.
[7] CHANDY K.M., BROWNE J.C., DISSLY C.W., UHRIG W.R., *Analytic models for rollback and recovery strategies in data base systems,* IEEE Trans, Software Engineering 1975, Vol. 1, No 1.
[8] CODD E. F., *The Relational Model for Database Management: Version 2,* Addison Wesley, Reading 1990.
[9] CONNOLY T., BEGG C., *DataBase Systems, A Practical Approach to Design Implementation, and Management*, Addison Wesley, Harlow 2002.
[10] EARL M.J., *Management Strategies for Information Technology*, Prentice Hall, Hempstead 1989.
[11] ERLANK S., LEVIN C., *PowerBuilder 6.0, Oficjalny podręcznik*, MIKOM, Warszawa 1998.
[12] GOGOLLA M., HEHENSTEIN U., *Towards a semantic view of the Entity-Relationship model,* ACM Trans. Database Systems 1991, Vol. 16, No 3.
[13] GRUBER M., *SQL*, 2nd ed., Helion, Gliwice 2000.
[14] HALL L.C., *Techniczne podstawy systemów klient/serwer*, WNT, Warszawa 1996.
[15] HENDERSON K., *Bazy danych w architekturze klient/serwer*, Robomatic, Wrocław 1998.
[16] LADANYJ H., *SQL, Księga eksperta*, Helion, Gliwice 2000.
[17] LAURIE B., LAURIE P., *Apache, Przewodnik encyklopedyczny*, Helion, Gliwice 2001.
[18] LYNCH N., MERRIT M., WEIHL W., FECHET A., *Atomic Transaction*, Morgan Kaufmann, San Francisco, CA 1994.
[19] ROBERTSON J., ROBERTSON S., *Pełna analiza systemowa*, WNT, Warszawa 1999.
[20] ULLMAN S., *Systemy baz danych*, WNT, Warszawa 1988.
[21] WITKOWSKI M., *Apache i inne plemiona*, PC Kurier 2002, nr 1.
[22] YOURDON E., *Współczesna analiza strukturalna*, WNT, Warszawa 1996.
[23] Dokumentacja systemu Sybase.
[24] Strona internetowa DBMS ONLINE http://www.intelligententerprise.com
[25] Strona internetowa http://httpd.apache.org.
[26] Strona internetowa firmy Sybase http://www.sybase.com.pl