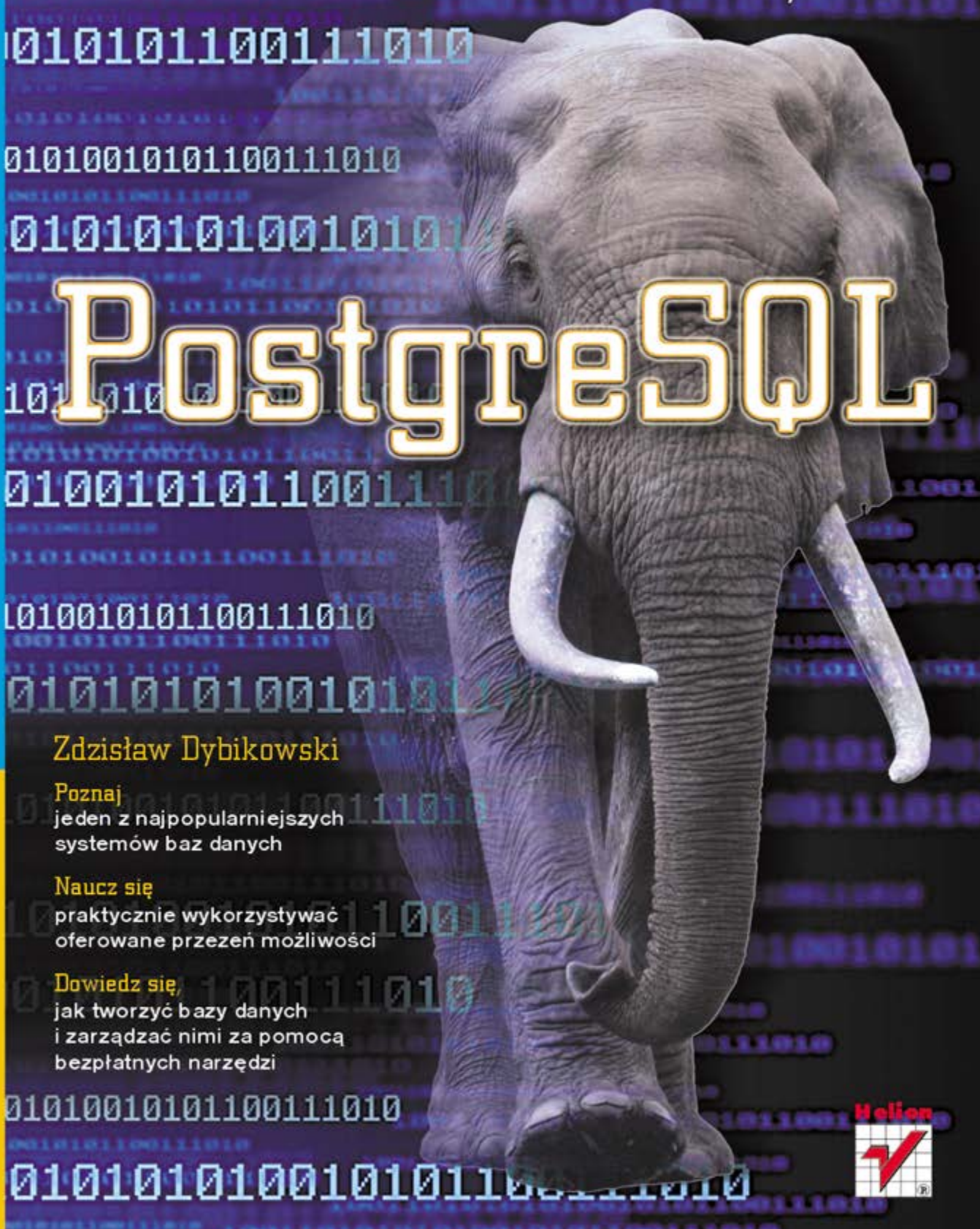
Jeśli baza danych, to tylko z PostgreSQL!

Wydanie II



PostgreSQL

Zdzisław Dybikowski

Poznaj
jeden z najpopularniejszych systemów baz danych

Naucz się
praktycznie wykorzystywać oferowane przezeń możliwości

Dowiedz się,
jak tworzyć bazy danych i zarządzać nimi za pomocą bezpłatnych narzędzi

Helion

Redaktor prowadzący: Michał Mrowiec

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?posql2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-246-3068-4

Printed in Poland.

Spis treści

Rozdział 10.

Budowa aplikacji bazodanowych

Budowanie każdego programu jest trudnym zagadnieniem. Tworzenie zaś aplikacji bazodanowej jest szczególnie trudne. Spróbuję w tej części książki streścić pokrótce zagadnienia, które należałoby rozpatrzyć, aby odnieść sukces w dziedzinie budowy programów i baz danych.

PostgreSQL stanowi doskonałą bazę danych, na której można oprzeć budowę i działanie bardzo wymagających aplikacji, i to o strategicznym znaczeniu dla firm. Coraz częściej firmy decydują się na użycie PostgreSQL właśnie ze względu na jego wysoką wydajność, niezawodność i uniwersalność dostępu do danych.

Zasadniczym problemem, przed jakim stają twórcy baz danych, jest prawidłowe modelowanie zjawisk świata rzeczywistego, które mają być przełożone na informacje uzyskiwane z bazy danych. Problem ten można rozpatrzyć na płaszczyźnie tego, co mamy gromadzić w bazie danych, w jaki sposób to czynić, aby samo gromadzenie danych nie stało się udręką użytkowników, oraz w jaki sposób przechowywać zebrane dane i odzyskiwać z nich niezbędne nam informacje.

Cechą każdego modelu jest to, że może on tylko w sposób uproszczony opisywać rzeczywistość. Z reguły jest to rzeczywistość z dziedziny obrotu gospodarczego i zjawisk pokrewnych, ale również otaczającego nas świata, codziennych czynności, rozkładów zajęć, hobby i temu podobnych czynności życiowych. Stąd też założyć można, że model nigdy nie będzie doskonały, jak również i to, że ze zbudowanej bazy danych będzie mogła korzystać dość ograniczona liczba użytkowników, gdyż zakres przechowywanej informacji z natury rzeczy musi być ograniczony. Kolejną cechą modelu jest to, że modelowana rzeczywistość zmienia się, stąd też pomimo pewnej elastyczności modelu czas jego życia, jak również budowanej na nim bazy danych jest ograniczony.

Model bazy danych a PostgreSQL

Model bazy danych jest elementem abstrakcyjnym i niezależnym od sprzętu i oprogramowania systemu operacyjnego czy też bazy danych. Jednakże już na etapie tworzenia takiego modelu należy wziąć pod uwagę pewne ograniczenia sprzętowe i programowe.

Zacznijmy od przymierzenia się z abstrakcyjnym modelem działania firmy handlowej do bazy danych PostgreSQL.

Pierwszym elementem, który należy uwzględnić, jest **granica zasobów**. To pojęcie powinno uwzględniać zarówno ilość składowanej informacji, długość życia informacji zawartej w bazie danych, jak również szybkość dostępu do informacji. Rozpatrując te zagadnienia, należy stwierdzić, że PostgreSQL, posiadający pewne perspektywy rozwoju, powinien spełnić założenia o nieprzekraczaniu przez model funkcjonowania przedsiębiorstwa granicy zasobów. Jednakże to stwierdzenie jest prawdziwe jedynie dla małych i średnich przedsiębiorstw. W przypadku dużych organizacji gospodarczych centrum, z którego można wydobywać informacje, musi być z konieczności rozproszone.

Z ograniczeniami zasobów można walczyć poprzez modernizowanie sprzętu komputerowego, na którym osadzone jest oprogramowanie motoru baz danych i sama fizyczna baza danych, jak również poprzez unowocześnianie oprogramowania systemów operacyjnych i motorów baz danych. Te ostatnie czynności są szczególnie trudne do realizacji.

Kolejnym elementem, który należy uwzględnić, jest **żądanie nowych aplikacji** użytkowych opartych na posiadanej już bazie danych. Ta cecha prowokowana przez użytkowników wynika z natury baz danych, gdyż z założenia powinny one ułatwiać i przyspieszać dostęp do informacji. Użytkownicy na początku nie potrafią precyzyjnie określić swoich potrzeb, stąd w miarę wzrostu kultury obcowania z informacją zawartą w bazie danych ich oczekiwania w stosunku do aplikacji rosną. Z reguły tylko nowe wersje pozwalają zaspokajać te rosnące potrzeby.

Rozpatrując to zagadnienie, należy zaznaczyć, że PostgreSQL pozwala na dość swobodny rozwój aplikacji. Możliwość ta wynika z wielości dostępnych interfejsów do łączności z bazą danych. Dobrze stworzony model funkcjonowania przedsiębiorstwa pozwala na wyodrębnienie szeregu oddzielnych zagadnień, które stosunkowo łatwo przekształcić w moduły jednej aplikacji bazodanowej z możliwie ujednoliconą obsługą. Dodatkowo, stosunkowo łatwo jest przenieść modele zbudowane za pomocą narzędzi CASE z innych platform bazodanowych na PostgreSQL.

Następnym elementem jest **konieczność wymiany sprzętu i oprogramowania** klienckiego. Zazwyczaj model i zbudowana na nim aplikacja nie wymuszają takich zmian w sprzęcie i oprogramowaniu. Jednakże szereg pomocniczych aplikacji funkcjonujących w firmie może skutecznie wymuszać cykliczne zmiany w oprogramowaniu i w sprzęcie stosowanym również do prac z aplikacją obsługi przedsiębiorstwa. Posłużenie się tutaj mechanizmami Intranetu pozwala na ograniczenie takich działań. Z kolei oparcie się na klastrze Linuxa i PostgreSQL pozwala na dynamiczne generowanie niezbędnej wydajności systemu.

Zastosowanie do budowy aplikacji na podstawie PostgreSQL musi mieścić się w przyjętym modelu rozwoju przedsiębiorstwa, możliwych do poniesienia nakładach, jak również przenoszalności bazy danych na inną platformę systemową i sprzętową, gdy istniejąca infrastruktura ulegnie uszkodzeniu lub producent wycofa swoje wsparcie dla przyjętego wcześniej przez nas rozwiązania i technologii. PostgreSQL jest z punktu widzenia platformy klienta odporny na takie działania. Wynika to z faktu, że kilka interfejsów dostępu do bazy danych PostgreSQL jest uniwersalnych na tyle, że mogą funkcjonować niezależnie od środowiska klienta. Oczywiście, najbardziej perspektywicznie zapowiada się tutaj technologia oparta na języku Java i JDBC, ale również dostęp poprzez serwer web z użyciem PHP.

Ostatnim, ale równie ważnym zagadnieniem, jakie należy rozpatrzyć, jest **zmiana uwarunkowań prawnych**, które wpływać mogą i zazwyczaj wpływają na funkcjonowanie systemu informatycznego i aplikacji bazodanowych w przedsiębiorstwie. Model funkcjonowania przedsiębiorstwa powinien uwzględniać nie tylko zastaną sytuację, ale również być na tyle elastyczny, aby można było wprowadzić pożądane zmiany i modyfikacje w bazie danych. Zmiana struktury baz danych pociąga za sobą konieczność zmian w większości modułów tworzących zintegrowany system obsługi przedsiębiorstwa.

PostgreSQL jest w pełni relacyjną bazą danych, zaś relacje nie występują w bazie danych jako fizyczne obiekty. Dobrze przygotowany model można z łatwością przenieść na strukturę bazy danych w PostgreSQL. Można by rzec, że dopóki założenia modelu dobrze korelują z relacyjnością bazy danych, to przeniesienie modelu na strukturę bazy danych i zapytania jest możliwe, a zmiany prawne nie powinny w istotny sposób wpływać na konieczność modyfikacji już istniejących i poprawnie funkcjonujących modułów.

Rozpatrując problem modelowania zagadnień gospodarczych i przenoszenia ich na grunt bazy danych PostgreSQL, należy pamiętać, że skonstruowanie modelu i następnie jego poprawne przeniesienie na bazę danych i aplikacje obsługi baz danych nie wystarczają do zagwarantowania rzeczywistego sukcesu aplikacji bazodanowej.

Istnieje cały szereg uwarunkowań, które sprawiają, że aplikacja bazodanowa zbudowana na podstawie modelu uwzględniającego granicę zasobów, konieczność modernizacji po stronie klienta, zmianę przepisów prawa i rodzące się w miarę rozwoju systemu żądania nowych aplikacji może być traktowana jako udana. Takie aplikacje odznaczają się następującymi cechami:

- Wykazują dużą i rzeczywistą użyteczność w pracy na poszczególnych stanowiskach. Im bardziej aplikacja okazuje się spełniać oczekiwania użytkownika, tym jest lepsza. Należy jednakże pamiętać, że wraz z użytkowaniem systemów informatycznych rosną wymagania wobec posiadanej aplikacji i w stosunku do posiadanej bazy danych.
- Aplikacja musi wspomagać działania organizacji (szczególnie gospodarczych) w osiąganiu ich celów. Stąd też wymagania dla bazy danych firmy wysyłkowej będą różne od wymagań firmy produkcyjnej, chociaż część założeń modelu bazy danych i aplikacji może być wspólna.

- Baza danych i zbudowana na niej aplikacja powinny umożliwiać bez większych modyfikacji obsługę innych niż obecnie zadań, w miarę jak zmienia się profil organizacji. Prawdopodobne kierunki zmian w działaniu firm powinny być uwzględnione już na etapie tworzenia modelu bazy danych.
- Powstanie i wdrożenie aplikacji bazodanowej powinno być akceptowane przez użytkowników. Ten cel najtrudniej jest osiągnąć. Wynika to z faktu, że bardzo często firmy i organizacje posiadają ukształtowaną historycznie strukturę i hierarchię podległości pracowniczej, od której trudno jest odstąpić, a która okazuje się przeszkodą w sprawnym przepływie informacji. Dobrze zaprojektowana baza danych zakłada sprawny przepływ informacji pomiędzy stanowiskami oraz pewną synchronizację w czasie operacji na bazie danych. Dotychczasowe, niezsynchronizowane działania wielu pracowników nie mogą już mieć miejsca, jeżeli aplikacja ma obejmować i wspomagać swoją pracą wiele stanowisk. W czasie projektowania aplikacji i tworzenia modelu bazy danych niezbędne jest uświadomienie użytkownikom zmian związanych z wprowadzeniem systemu informatycznego i uzyskanie od nich akceptacji. Bez tego nawet najlepsza aplikacja nie zafunkcjonuje prawidłowo i nie spełni pokładanych w niej nadziei.
- Tworzenie aplikacji i wdrożenie jej do działania musi być akceptowane przez zarząd organizacji. Dotyczy to zarówno akceptacji niezbędnych do poniesienia finansowych nakładów początkowych, jak i akceptacji kosztów eksploatacji systemów. Konieczne jest również zaaprobowanie przez kierownictwo niezbędnych zmian organizacyjnych, w tym wzrostu roli kadry informatycznej w przedsiębiorstwie czy w administracji.

Podsumowując, można rzec, że dzięki zrozumieniu rzeczywistych potrzeb organizacji i tworzących ją użytkowników możliwe jest opracowanie prawidłowego modelu bazy danych, na której ma funkcjonować aplikacja bazodanowa. Z kolei zrozumienie i akceptacja przez użytkowników przygotowanego i opracowanego modelu pozwala na pełne wykorzystanie możliwości tkwiących w nowoczesnym systemie informatycznym zbudowanym na relacyjnej bazie danych.

Bardzo często brak dobrego modelu bazy danych lub brak akceptacji wdrażanego systemu informatycznego skutkuje jedynie ogromnymi kosztami i znikomą przydatnością aplikacji dla organizmów gospodarczych.

Przewaga PostgreSQL w stosunku do innych baz danych wynika z prostoty obsługi tej bazy danych, stosunkowo dużej wydajności i niesamowitej mnogości interfejsów do bazy danych. Nie sposób też nie dostrzec, że wszystko to odbywa się na zasadach licencyjnych niewymagających opłat. Ponadto na jednym serwerze może być umieszczonych wiele niezależnych od siebie baz danych. Zwiększa to stopień elastyczności i pozwala na etapowe tworzenie i wdrażanie aplikacji bazodanowych.

Model działania firmy

Zasadniczym elementem modelu danych jest określenie funkcji firmy lub organizacji i sporządzenie na podstawie opisu modelu danych. Model danych powinien określać, które dane są niezbędne dla całej organizacji, a które są jedynie danymi pomocniczymi. Pozwala to na skupienie się na najważniejszych celach i funkcjach firmy i na dołączenie potem funkcji pobocznych do zasadniczego modelu jej działania.

Z opisów zasadniczych funkcji i celów firmy powinna wynikać struktura organizacyjna. Na podstawie formularzy można określić podstawowe wymagania dotyczące przetwarzanych na stanowisku danych, jednakże cele i funkcje organizacji muszą być określone przez kierownictwo.

W przypadku definicji funkcji przedsiębiorstwa należy opracować:

- Zakres świadczonych usług lub towarów będących przedmiotem zainteresowania firmy.
- Czynniki niezbędne do osiągnięcia celów stawianych sobie przez organizację. Chodzi tutaj przede wszystkim o czynnik ludzki, gdyż posiadane pożądane cechy załogi stanowić będą o sukcesie lub porażce przedsięwzięcia i będą wpływać na postrzeganie systemu informatycznego przez użytkowników. Innym elementem będą niezbędne do działania maszyny i urządzenia oraz sposoby rozliczania pracy tych urządzeń. Szczególnie ten ostatni czynnik musi znaleźć odzwierciedlenie w modelu działania firmy, a później w modelu bazy danych.
- Listę długo- i krótkoterminowych celów gospodarczych firmy.
- Priorytety poszczególnych celów gospodarczych. Będą one różne w zależności od stadium rozwoju firmy.
- Określenie zadań systemu informatycznego.
- Ocenę, które z istniejących już systemów informatycznych mogą być przydatne w dalszej działalności firmy. Dotyczy to również systemów informatycznych, z których w sposób zautomatyzowany można przenosić dane do nowo tworzonej aplikacji bazodanowej.

Na bazie zebranych informacji można zamodelować działanie firmy i określić kluczowe elementy w modelu działania. Model pozwala na określenie wszelkich nieefektywności w procedurach używanych dotychczas w firmie. Sztuką jest taka współpraca z kierownikami różnych szczebli, aby postrzegali zakłócenia i nieefektywności w działaniu swoich działów i pracowników na poszczególnych stanowiskach nie jako wytykanie błędów, a jako sposób na podniesienie wydajności. Pracownicy zaś powinni dostrzegać w modelu działania firmy sposób na wzrost płac lub chociażby na możliwość zdobywania nowych kwalifikacji i doświadczeń.

Na bazie modelu działania firmy można zbudować model bazy danych, a dalej aplikacje bazodanowe. Tak utworzony system powinien dawać typowe korzyści wynikające z zastosowania systemu informatycznego. Zasadniczą korzyścią jest zmniejszenie

nakładów pracy przy przetwarzaniu informacji. Dotychczasowe, często wielokrotne, wprowadzanie do systemów lub do formularzy jednakowej informacji może zostać wyeliminowane przez dobrze zaprojektowaną aplikację bazodanową. Baza danych, która stanowi podstawę systemów informatycznych, pozwala na wspólne posługiwanie się informacją zarówno przez dział księgowości czy kadr, jak i przez dział handlowy i marketingu.

Metodologia projektowania i wykonywania aplikacji bazodanowej

Podczas budowania aplikacji bazodanowej można natknąć się na dwa zasadnicze problemy. Jednym z nich jest prawidłowe zaprojektowanie aplikacji, zaś drugim jakość jej wykonania. Oba te zagadnienia są ze sobą ściśle związane.

Pierwsze podejście do zagadnienia stawia nas zwykle przed ogromną złożonością systemu, który ma obejmować firmę. Mechanizmem, który pozwala zmniejszyć wynikające z tego zagrożenie zagubieniem jakichś istotnych rzeczy w procesie projektowania, jest dzielenie zagadnienia na mniejsze części. Wszystkie późniejsze zagadnienia wiążą się z rozważaniem mniejszego i zamkniętego obszaru. Dopiero analiza strategiczna wiąże ze sobą poszczególne części w większą całość. Postaramy się teraz omówić zagadnienia związane z różnymi technikami tej analizy.

Na etapie analizy strategicznej przedsiębiorstwa można przeprowadzać rozeznanie zarówno wśród kadry zarządczej, jak i u średniego personelu technicznego i biurowego. Na bazie zebranych informacji można budować zarówno modele zbieżne, w których staramy się uchwycić elementy łączące poszczególne informacje w jedną wspólną całość, jak i rozbieżne, służące odnalezieniu zasadniczych różnic w sposobie postrzegania istoty działania przedsiębiorstwa. W efekcie powinniśmy uzyskać spójną informację, w wyniku której można zbudować stabilny model.

Podczas tworzenia modeli firmy ważne jest uchwycenie jej zasadniczych zadań. Nie od dziś przecież wiadomo, że przedsiębiorstwo inaczej jest postrzegane przez właściciela, inaczej przez zarząd, zaś jeszcze inaczej przez średni i niższy szczebel pracowniczy. Istotą **hierarchii funkcji** przedsiębiorstwa ma być określenie tego, co powinna robić firma. Wbrew obiegowym opiniom, często różni się to znacznie od tego, co firma i jej poszczególni pracownicy robią w rzeczywistości. Należy tutaj rozpatrzyć wszystkie funkcje przedsiębiorstwa, a nie tylko te, które w chwili obecnej mają być związane z informatyzacją czy automatyzacją pewnych procesów w firmie.

Gdy już zostaną sprecyzowane elementarne funkcje przedsiębiorstwa, należy przystąpić do **analizy zdarzeń**. Zdarzenia noszą cechę istotną z punktu widzenia przekazu informacji. Sprawiają one mianowicie, że uaktywniane są poszczególne elementarne funkcje przedsiębiorstwa. Z reguły też zdarzenia powodują zmianę lub aktualizację pewnych danych albo same wywołane są poprzez zmianę lub aktualizację danych.

W efekcie można wyodrębnić pojedyncze obszary funkcjonowania przedsiębiorstwa, zamknąć je w moduły (np. księgowość, kadry, produkcja itp.) oraz określić zakres współdziałania pomiędzy poszczególnymi modułami. Należy dążyć do tego, aby to, co zawarte jest w module, ściślej określało związki pomiędzy zdarzeniami i elementarnymi funkcjami niż to, co jest poza modułem. Wynika to z faktu, że informacje i dane spoza pewnego monolitycznego obszaru działania przedsiębiorstwa są trudniej dostępne niż dane wewnątrz takiego obszaru, a czasami wręcz niedostępne.

W ściśle określonym module można zbudować diagramy encji. Encję należy rozumieć jako cokolwiek, o czym chcemy przechowywać informację w naszej bazie danych. Dobrym przykładem diagramu encji jest związek towar – klient, w którym każdy towar jest powiązany z klientem, który ten towar zamówił lub kupił.

Na bazie diagramów encji można wyodrębnić poszczególne atrybuty, doskonale opisujące encję. Dla przykładu, każdy klient będzie miał swój adres, nazwisko lub nazwę, udzielony kredyt itp.

Tego typu diagramy łatwo przekładają się na tabele i ich kolumny. Teraz, jeszcze przed zbudowaniem tabel, należy określić związki relacyjne. W zasadzie są ich trzy rodzaje:

- **Jeden do jednego** — rzadko spotykany; z reguły jest to sygnał, iż być może diagram encji został niepoprawnie zbudowany.
- **Jeden do wielu** — najbardziej rozpowszechniony rodzaj związku; doskonale można go zobrazować tym, że jeden klient może mieć kilka adresów, pod które wysyłany jest towar z zamówienia.
- **Wiele do wielu** — często wskazuje to na potrzebę dalszej normalizacji encji, chociaż równie często złożoność systemu wymusza stosowanie tego typu związków. Dokładniejsza analiza obiegu dokumentów z reguły wystarcza do wyjaśnienia sprawy.

Podczas analizy związków encji możliwe staje się wyodrębnienie podtypów i nadtypów encji. Pozwala to na skupienie się tylko na niezbędnym nam zakresie informacji poprzez stworzenie nadtypów lub dalsze uszczegółowienie informacji poprzez wydzielenie podtypów.

Kolejny etap to porównanie przetwarzanej przez dany moduł informacji z informacją przetwarzaną w ramach innego, wyodrębnionego wcześniej modułu. Taka **analiza jednostek** w przedsiębiorstwie ma na celu określenie, czy nie zachodzi niepotrzebne dublowanie wprowadzania, analizowania lub przetwarzania informacji. Trzeba pamiętać o tym, że każde takie dublowanie się obciąża dodatkowymi kosztami przedsiębiorstwo i obniża jego zdolność do konkurencji. Czasem jednak może się również okazać, że nie sposób wyeliminować dublowania się pewnych czynności, szczególnie na rozproszonej bazie danych.

W efekcie podjętych działań powinniśmy otrzymać **model logiczny**, zaś na bazie diagramów encji możliwe jest **stworzenie diagramów przepływu danych**. Teraz pozostaje wybrać mechanizmy działań oraz opisać **procedury realizacji elementarnych funkcji**. Te elementy opisowe powinny pomóc nam podczas wdrażania aplikacji zbudowanej na bazie naszych analiz.

Etap projektowania powinien zakończyć się powtórną analizą. Brak kontroli na etapie projektowania często powoduje powstanie zasadniczych błędów, które są niemożliwe do wyeliminowania w przyszłości i mogą spowodować, że cały proces tworzenia i wdrażania aplikacji bazodanowej zakończy się niepowodzeniem.

Metodologia wykonania sprowadza się do wyboru odpowiedniej platformy systemowej, na której ma być osadzona aplikacja, wyboru narzędzi i bazy danych oraz zapewnienia standardów jakościowych.

Wybór platformy systemowej z reguły nie jest trudny. Platform znaczących jest obecnie niezbyt dużo i często wybór ten związany jest z wyborem narzędzi czy też bazy danych. PostgreSQL jako baza danych doskonale współpracuje z rodziną systemów uniksowych. Stąd też wybór tego rodzaju systemu wydaje się najbardziej celowy. Spośród systemów klasy Posix wyróżnia się, w sensie pozytywnym, Linux. Wybór tej platformy jest atrakcyjny zarówno z uwagi na nowoczesne technologie zastosowane podczas jej tworzenia, jak i ze względów cenowych.

Z kolei dla firm, gdzie systemy Posix nie występują, być może do przyjęcia będzie korzystanie z platformy Windows. Najnowsze wersje PostgreSQL są dostępne dla tej platformy.

Narzędzia do tworzenia aplikacji bazodanowej mogą zaś być różne, gdyż różne będą platformy klienckich stacji roboczych czy też zorganizowania się przedsiębiorstwa. Stąd też dla jednych podstawowy może być wybór MS Accessa jako warstwy obsługującej klienta aplikacji, zaś dla innych celowe będzie przeorientowanie się na serwis web, poprzez który będzie następowała interakcja z bazą danych.

Kolejnym elementem znaczącym dla procesu powstawania oprogramowania jest przyjęcie standardów zapewnienia jakości oraz modelu pozyskiwania informacji dla osób zarządzających poszczególnymi działami i całością firmy.

Praktyczna implementacja modelu

Załóżmy, że zostały już zbadane cele firmy oraz że model działania przedsiębiorstwa został pozytywnie zaopiniowany. Wydzielone bloki zagadnień pozwalają na zamknięcie danych w hermetyczne struktury, z których pobór informacji, uzupełnianie i wymiana następować będą poprzez ściśle zdefiniowane interfejsy.

Doskonałym przykładem takiej implementacji jest zagadnienie związane z obsługą magazynu hurtowni. Stałym elementem tej części firmy jest nieustanny obrót towarowy i dokumentowy wyrażony w pieniądzach, ilości towarów i asortymencie.

Zanim rozpoczniemy tworzenie aplikacji, warto przygotować i przetestować skrypt powłoki shell, gdyż przyda się on nam do zautomatyzowania procesu tworzenia bazy danych.

Skrypt mógłby wyglądać następująco:

```
#!/bin/sh
psql -d template1 <<!
/* usuniecie starej bazy danych */
drop database firma;
/* utworzenie nowej bazy danych */
create database firma;
/* podłaczenie sie do bazy danych */
\connect firma

/* utworzenie przykladowej tabeli */

DROP TABLE test;

CREATE TABLE test
(
id_test   serial not null primary key
test      char(25) NOT NULL,
opcja     boolean NOT NULL CHECK (tak = '1' OR nie='0') ,
data      date DEFAULT CURRENT_DATE
);

\q
!
```

Mechanizm działania skryptu jest stosunkowo prosty. Po podłączeniu się do wzorcowej bazy danych *template1* usuwana jest poprzednia wersja bazy danych (uwaga! — wraz z danymi), następnie generowana jest nowa baza danych *firma*, a po podłączeniu się do nowo utworzonej bazy danych wysyłany jest ciąg zapytań SQL, który tworzy strukturę bazy danych.

Takie rozwiązanie jest bardzo wygodne, gdyż w połączeniu z automatycznym eksportem pozwala to na bardzo szybkie odtworzenie stanu bazy danych sprzed awarii.

Rozważmy teraz zagadnienie obrotu magazynowego od strony asortymentu. W magazynie znajdują się towary, których stan może być zerowy, które były kiedyś już w sprzedaży i obecnie są wycofane z obrotu, oraz nowe towary, które będą pojawiać się w magazynie przedsiębiorstwa. Stąd też wszystkie elementy pozwalające wyróżnić towar powinny być zgrupowane w jednej zasadniczej tabeli.

Dane z takiej tabeli nie mogą być usuwane, gdyż może to spowodować problemy z dokumentami, na których przecież muszą widnieć nazwy towaru. Na tym drobnym przykładzie widać, że należy przyjąć ogólną *zasadę nieusuwania danych* z bazy danych. Każde dane, które usuniemy z bazy danych, są tracone bezpowrotnie. Bardzo często tuż po usunięciu danych niezbędne jest skorzystanie z nich. Stąd też, gdy usunięcia danych nie można uniknąć, należałoby z usuwanych danych sporządzić zestawienia zbiorcze i te przechowywać dłużej w bazie danych.

Magazyn towarów posługujący się semantyką asortymentową musi obejmować pewne elementy charakterystyki, które można przedstawić w postaci tabeli encji (tabela 10.1).

Tabela 10.1. *Tablica z charakterystyką asortymentu*

Lp.	Nazwa atrybutu	Charakterystyka
1.	Indeks	Ciąg znaków alfanumerycznych
2.	Nazwa asortymentu	Tekst nazwy
3.	Magazyn	Opis magazynu
4.	VAT	Stawka podatku VAT
5.	Akcyza	Stawka podatku akcyzowego
6.	Miara	Jednostka miary
7.	SWW	Symbol SWW
8.	Grupa towarowa	Rodzaj towaru lub grupy
9.	Dostawca towaru	Nazwa dostawcy towaru

Elementem encji, który pozwoli połączyć te zapisy w bazie danych z innymi, będzie unikatowy indeks lub nazwa towaru. Innym wyjściem jest ustanowienie zupełnie niezależnego klucza głównego dla zbioru asortymentu. Zazwyczaj używa się tutaj sekwencji, która może być w prosty sposób utworzona i wywołana podczas wstawiania wartości do tabeli.

```
drop sequence ID

create sequence ID increment 10 start 10;
```

Sekwencja jest pewnym szeregiem liczb całkowitych, stąd też unikniemy powtórzeń, gdy będziemy podawać kolejne wartości z sekwencji. Dobrym zwyczajem jest używanie tej samej sekwencji dla tej samej grupy zbiorów.

W tabeli encji pewne atrybuty można by było wyróżnić w postaci słowników, z których wartości byłyby wybierane do tabeli zasadniczej. Do takich atrybutów należą stawka VAT, stawka podatku akcyzowego, jednostka miary, symbol SWW oraz grupa towarowa i nazwa dostawcy.

Powyższą tabelę można przestawić w postaci zapytania SQL.

```
drop table asortyment;

create table asortyment
(
id_asortyment integer not null primary key default nextval('id'),
asortyment text not null,
id_magazyn integer references magazyn(id_magazyn),
id_VAT integer references VAT(id_VAT) not null,
id_akcyza integer references akcyza(id_akcyza),
id_miara intteger references miara(id_miara) not null,
index char(15) not null,
id_grupa integer references grupa(id_grupa) not null,
id_SWW integer references sww(id_sww),
id_kontrahent integer references kontrahent(id_kontrahent) not null,
opakowanie_zwrotne boolean not null default 'no',
stempel_czasowy timestamp not null default now()
);
```

Tablice słownikowe zostaną utworzone po wysłaniu do bazy danych zapytań, takich jak poniżej.

```
drop table magazyn;

create table magazyn
(
id_magazyn integer not null primary key default nextval('id'),
magazyn char(25) not null,
lokacja text not null
);

drop table vat;

create table vat
(
id_VAT integer not null primary key default nextval('id'),
vat integer not null default 22
);

drop table akcyza;

create table akcyza
(
id_akcyza integer not null primary key default nextval('id'),
akcyza integer not null
);


drop table miara;

create table miara
(
id_miara integer not null primary key default nextval('id'),
miara char(10) not null
);

drop table grupa;

create table grupa
(
id_grupa integer not null primary key default nextval('id'),
grupa text not null,
index_grp char(15) not null
);

drop table sww;

create table sww
(
id_SWW integer not null primary key default nextval('id'),
SWW char(25) not null,
nazwa text not null
);
```

Powiązania pomiędzy tablicami słownikowymi a tablicą zasadniczą wymuszane są poprzez polecenie `references`. Należy przy tym pamiętać o tym, że więzy integralności są

wymuszane dopiero w najnowszych wersjach bazy danych, oraz o *problemach z wartością null*. Te ostatnie można rozwiązać, stosując podzapytania do zapytania głównego. PostgreSQL na razie nie obsługuje powiązań zewnętrznych, ale nie jest to problem złożony i stosunkowo łatwo jest znaleźć rozwiązanie zadowalające programistów.

Oczywiste też jest to, że tablice słownikowe powinny być zawsze tworzone przed utworzeniem tabeli głównej, inaczej nie będzie można powołać się na referencje do obiektu bazy danych, który przecież jeszcze nie istnieje.

Skupmy się teraz na kontrahencie, który wobec firmy może pełnić rolę dostawcy lub odbiorcy towarów albo też obie te role łącznie. Możliwe jest również, że kontrahent pełni rolę usługową i nie ma bezpośredniego związku z asortymentem. Zatem tylko część kontrahentów i informacji o nich będzie związana z asortymentem, który stanowi przedmiot obrotu firmy handlowej lub produkcyjnej. Będą to oczywiście dostawcy towarów, bezpośrednio związani z konkretnym towarem, ale również odbiorcy towarów, których związki nie uwidocznią się w powiązaniach grup zbioru nazw towarów (asortyment) i zbioru kontrahentów (dostawców).

Przykładowa tabela encji może wyglądać następująco (tabela 10.2).

Tabela 10.2. *Tablica z charakterystyką dostawcy*

Lp.	Nazwa atrybutu	Charakterystyka
1.	Firma	Nazwa przedsiębiorstwa
2.	Nazwisko	Nazwisko właściciela lub osoby odpowiedzialnej
3.	Imię	Imię właściciela lub osoby odpowiedzialnej
4.	NIP	Numer identyfikacji podatkowej
5.	Regon	Numer statystyczny
6.	Miejscowość	Miejscowość będąca siedzibą firmy
7.	Ulica	Nazwa ulicy
8.	Numer domu	Numer domu
9.	Numer lokalu	Numer biura
10.	Poczta	Poczta
11.	Kod pocztowy	Kod pocztowy
12.	Region	Powiat lub województwo
13.	Państwo	Nazwa państwa siedziby kontrahenta
14.	Typ kontrahenta	Dostawca lub odbiorca
15.	Status	Określenie, czy kontrahent jest aktywny

Podobnie jak to miało miejsce poprzednio, z tablicy encji można wyodrębnić atrybuty, które z łatwością dadzą się przekształcić w tabele słownikowe. Stąd też zapytania tworzące grupę tabel związanych z kontrahentem mogłyby wyglądać następująco:

```
drop table kontrahent;

create table kontrahent
(
```

```
id_kontrahent integer not null primary key default nextval('id'),
nazwa text  not null,
nazwisko text,
imie text,
NIP char(10) not null,
REGON char(20),
miejscowosc text not null,
ulica text,
nr_domu char(5) not null,
nr_lokalu char(5),
poczta text not null,
kod_pocztowy char(6) not null,
id_powiat integer references powiat(id_powiat) not null default 0,
id_panstwo integer references panstwo(id_panstwo) not null default 0,
id_typ integer references typ(id_typ) not null default 0,
id_status integer references status(id_status) not null default 0,
stempel_czasowy timestamp not null default now()
);
```

Teraz spróbujmy utworzyć zapytania tworzące słowniki do tej grupy danych, które związane są z informacjami o kontrahentach. Pamiętajmy przy tym, że w rzeczywistej bazie danych najpierw tworzymy słownik, a dopiero potem tabelę powiązaną z tym słownikiem.

```
drop table panstwo;

create table panstwo
(
id_panstwo integer not null primary key default nextval('id'),
panstwo text not null
);

drop table powiat;

create table powiat
(
id_powiat integer not null primary key default nextval('id'),
wojewodztwo text not null,
powiat text not null
);

drop table typ;

create table typ
(
id_typ integer not null primary key default nextval('id'),
typ text
);
drop table status;

create table status
(
id_status integer not null primary key default nextval('id'),
status text
);
```

Na koniec przeanalizujmy dane zawierające rzeczywiste informacje o stanach magazynowych. Z danych tych będą przecież korzystać zarówno osoby przyjmujące towar do magazynu, jak i sprzedawcy oraz osoby odpowiedzialne za zamówienia, analizy przepływów i rotacji itp. Zasadnicze dane przechowywane w tej grupie tabel (tabela 10.3) to informacje o ilości danego towaru, cenie zakupu, cenie sprzedaży i, jeśli jest to szybko psujący się produkt, o okresie przydatności do użycia.

Tabela 10.3. *Tablica z charakterystyką magazynu w ujęciu ilościowym i cenowym*

Lp.	Nazwa atrybutu	Charakterystyka
1.	Nazwa asortymentu	Tekst nazwy
2.	Cena zakupu	Cena nabycia towaru
3.	Cena sprzedaży	Aktualna cena sprzedaży
4.	Ilość	Aktualna ilość towaru w magazynie

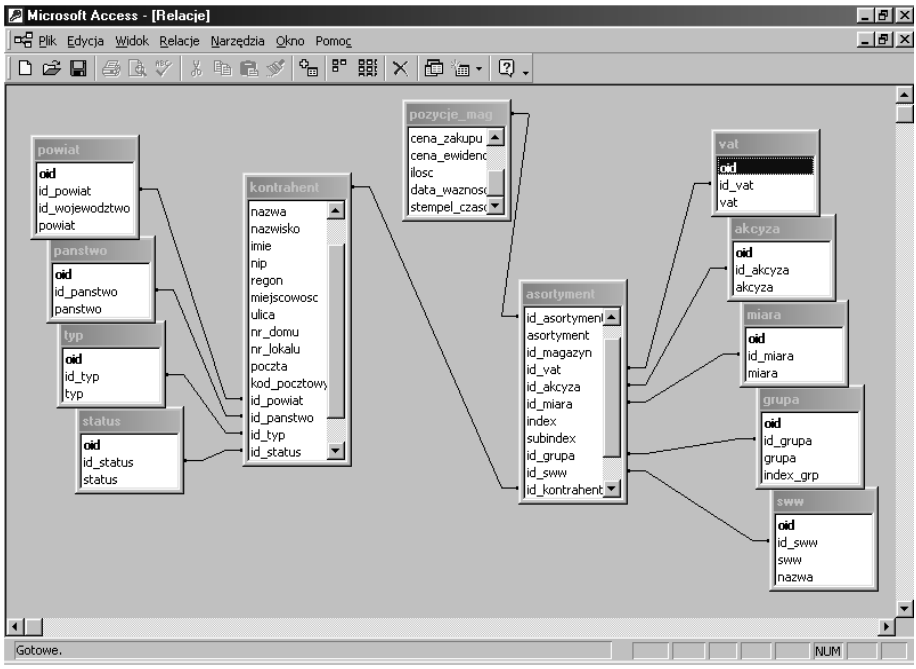
Z tablicy encji łatwo już jest utworzyć konkretne zapytanie tworzące tabelę. Ta ostatnia, w odróżnieniu od poprzednich tabel, nie będzie otoczona słownikami, gdyż są one zupełnie zbędne w przypadku tej grupy danych.

```
drop table pozycje_mag;

create table pozycje_mag
(
id_pozycje_mag integer not null primary key default nextval('id'),
id_asortyment integer not null references asortyment(id_asortyment),
cena_zakupu money not null,
cena_sprzedaży money not null,
ilosc real not null default 1.0,
data_waznosci date,
stempel_czasowy timestamp not null default now()
);
```

Poprzestańmy na tych przykładach i utwórzmy teraz obraz powiązań encji. Wizualizacja relacji pomiędzy danymi znakomicie ułatwi nam utworzenie zapytań SQL (rysunek 10.1) wstawiających dane do tabel, aktualizujących dane i usuwających je z tablic, najczęściej słownikowych. Jest to niezbędne, gdyż każda relacyjna baza danych ogranicza operacje z danymi wyłącznie do zapytań, zazwyczaj języka SQL, których rezultaty dopiero dalej mogą zostać obsłużone poprzez rozbudowane o elementy proceduralne języki programowania, takie jak PL/SQL, C++ i im podobne.

Tabele słownikowe nieco komplikują zapytania, ale znakomicie wpływają na wydajność bazy danych.

Na początek prześledźmy proces wpisywania danych do bazy danych. Jako pierwsze należałoby zapełnić tablice słownikowe. Zapytania tego typu są niezwykle proste i pozwalają na samym początku usystematyzować zakres informacji przechowywanych w bazie danych. Polecenie wprowadzania danych do słownika prześledźmy na przykładzie tabeli typów dostawców.



Rysunek 10.1. *Wizualizacja powiązań relacyjnych pomiędzy tabelami w bazie danych PostgreSQL. Access został tu użyty jako klient i posiada wyłącznie załączone tablice z bazy danych PostgreSQL*

```
begin work;

insert into typ
(typ)
values
('Dostawca');


insert into typ
(typ)
values
('Odbiorca');

commit work;
```

Do zgrupowania zapytań w transakcje warto użyć polecenia `begin work` otwierającego transakcje, a po zakończeniu procesu wstawiania informacji do bazy danych zatwierdzić transakcje za pomocą polecenia `commit work`. Pozwala to na zachowanie spójności relacyjnej w tych rzadkich przypadkach, gdy następuje przerwanie połączenia do bazy danych. Transakcje niezatwierdzone po pewnym czasie są automatycznie cofane z bazy. Z kolei dla transakcji niezadeklarowanych jawnie działa mechanizm automatycznego zatwierdzania transakcji, gdy tylko zapytanie zakończy się sukcesem.

Spróbujmy teraz usunąć dane z tabel słownikowych.

```
delete from typ
where typ='Odbiorca';
```

Jak widać, operacje na tabelach słownikowych są proste i mogą doskonale służyć jako wprawki do programowania w języku SQL. Z reguły zapytania uzupełniające tablice słownikowe podstawowymi danymi są traktowane jako część konstrukcji samej bazy danych.

Teraz, gdy tablice słownikowe są przygotowane, możemy przystąpić do wprowadzania danych do tablic podstawowych. Zapytania wpisujące dane do tabeli są nieco bardziej skomplikowane i z reguły wykorzystują klauzulę `select` do wybierania danych ze słowników. Zerknijmy na przykład wprowadzania danych do tabeli kontrahentów.

```
insert into kontrahent
(nazwa,
NIP,
miejscowosc,
nr_domu,
poczta,
kod_pocztowy,
id_powiat,
id_panstwo,
id_typ,
id_status)
select
'Fadena Ltd.',
'7181784060',
'Lomza',
'23',
'Łomża',
'18-400',
id_powiat,
id_panstwo,
id_typ,
id_status
from powiat, panstwo, typ, status
where
powiat='Lomza miasto'
and panstwo='Polska'
and typ='Dostawca'
and status='Aktywny';
```

Jak widać na powyższym przykładzie, zapytanie wybierające służy do pozyskania wartości z kluczy głównych i wstawienia tych wartości do kolumn zawierających klucze obce. Taka metoda nieco komplikuje zapytanie, jednakże zapewnia spójność relacyjną, niezbędną w każdej bazie danych. Powyższe zapytanie można potraktować jako wzorzec i z użyciem elementów języków proceduralnych konstruować elementy interfejsu użytkownika.

Jak już wspomniano wyżej, danych raczej nie należy usuwać z bazy danych. Stąd też rozpatrzmy teraz zapytanie aktualizujące dane. Dla przykładu chcielibyśmy zmienić adres siedziby firmy.

```
update kontrahent
set
miejscowosc='Kolno',
nr_domu='11',
poczta='Kolno',
```

```
kod_pocztowy='18-200',
where
nazwa='Fadena Ltd.';
```

Aktualizacja danych jest zazwyczaj prostsza niż ich wprowadzanie. Wynika to z faktu rzadszego sięgania do słowników i mniejszej komplikacji zapytania.

Niezwykle pożytecznym elementem bazy danych jest możliwość korzystania z widoków. Widoki bazy danych pozwalają ukryć złożoność wewnętrznej struktury bazy danych przed użytkownikiem oraz w znacznym stopniu ułatwiają operacje na danych.

Prześledźmy widok bazy danych złożony z tabeli podstawowej i tabel słownikowych. Do tworzenia widoków można z powodzeniem wykorzystać zapytania wybierające, gdyż sam widok nie zawiera danych, a do wyświetlania używa zapytań.

```
drop view v_kontrahent;

create view v_kontrahent as
select
nazwa,
NIP,
miejscowosc,
poczta,
kod_pocztowy,
powiat,
wojewodztwo,
panstwo,
typ,
status
from powiat, wojewodztwo, panstwo, typ, status
where
kontrahent.id_powiat=powiat.id_powiat
and kontrahent.id_wojewodztwo=wojewodztwo.id_wojewodztwo
and kontrahent.id_panstwo=panstwo.id_panstwo
and kontrahent.id_typ=typ.id_typ
and kontrahent.id_status=status.id_status;
```

Widoki z bazy danych mogą służyć za podstawę do sporządzania raportów. Ukryta za widokiem złożoność bazy danych pozwala uprościć zapytania, które stanowią podstawę do generowania raportów. Widoki z reguły scalają dane na różne sposoby, ułatwiając operowanie na bazie danych, i przygotowują materiał do wyciągów z bazy danych.

W części komercyjnych baz danych widoki pozwalają na aktualizację danych. W PostgreSQL, mimo że zapytanie aktualizujące wykona się poprawnie, w rzeczywistości aktualizacja poprzez widoki bazy danych nie jest możliwa. Trudność ta ma zostać naprawiona w nowszych wersjach PostgreSQL. Nie zmniejsza to jednak obszaru zastosowania widoków, zwłaszcza że w specyfikacji SQL92 ten element bazy danych nie występuje i pojawia się dopiero po roku 1995. Widoki pozwalają więc na pełną gamę operacji z widokami, mimo iż niezaimplementowana część nie wykona się w rzeczywistości.

Poniżej jeszcze jeden przykład widoku bazy danych, tym razem na grupę danych związanych z nazwami asortymentu w powiązaniu z grupą danych opisujących kontrahenta.

```
drop view v_asortyment;

create view v_asortyment as
select
asortyment,
index,
subindex,
opakowanie_zwrotne,
magazyn,
miara,
VAT,
akcyza,
grupa,
SWW,
kontrahent.nazwa,
NIP
from asortyment, magazyn, miara, vat, akcyza, grupa, sww, kontrahent
where asortyment.id_magazyn=magazyn.id_magazyn
and asortyment.id_miara=miara.id_miara
and asortyment.id_VAT=vat.id_vat
and asortyment.id_akcyza=akcyza.id_akcyza
and asortyment.id_grupa=grupa.id_grupa
and asortyment.id_SWW=sww.id_sww
and asortyment.id_kontrahent=kontrahent.id_kontrahent;
```

Relacyjna baza danych posiada własne mechanizmy zabezpieczeń dostępu do bazy danych. Jedynie użytkownik *postgres* posiada wszelkie przywileje do baz danych PostgreSQL. Zazwyczaj też administrator bazy danych wykonuje operacje tworzenia struktur bazodanowych, tablic, widoków, wprowadzenia podstawowych danych do słowników. Dlatego też ostatnim etapem tworzenia bazy danych jest nadanie uprawnień do obiektów bazy danych. Oczywiście, uprawnienia powinny być adekwatne do roli użytkownika w procesie przetwarzania informacji w bazie danych. Uprawnienia można przypisać do grupy, której członkiem będzie użytkownik bazy danych. Przywilejów jest kilka: prawo do wykonywania zapytań wybierających, prawo do wprowadzania informacji do tablic, prawo do aktualizacji danych, prawo do usuwania danych i prawa administracyjne. Przykładowe przywileje do obiektów PostgreSQL i sposób ich nadania mogą być wzięte z poniższego przykładu.

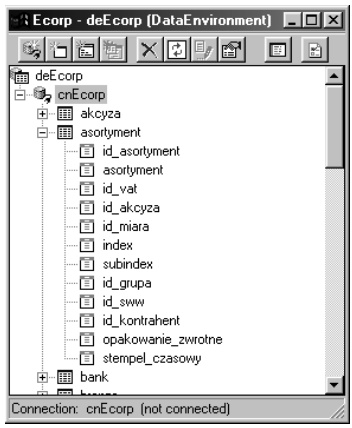
```
drop user fakt1;

create user fakt1 with password 'fadena' nocreatedb nocreateuser in group fakturzysta;

drop user kasa1;

create user kasa1 with password 'fadena' nocreatedb nocreateuser in group rachunkowosc;
grant select on
v_kontrahent,
v_asortyment,
v_pozycje_mag,
to group rachunkowosc;


grant select, insert, update on
kontrahent,
panstwo,
wojewodztwo,
```

```
powiat,
typ,
status,
magazyn,
vat,
akcyza,
miara,
grupa,
sww,
asortyment,
pozycje_mag,
to group fakturzysta;
```

W powyższym przykładzie większe uprawnienia do tablic baz danych posiada użytkownik będący członkiem grupy o identyfikatorze `fakturzysta`, gdyż to on będzie wprowadzał dane do bazy danych. Użytkownik ten nie ma prawa usuwania danych. Przywilej ten można by nadać kierownikowi zmiany lub osobie obsługującej system informatyczny firmy. Z kolei grupa `rachunkowość` i odpowiednio jej członkowie mają ograniczone przywileje jedynie do widoków z bazy danych. Pozwala to na zabezpieczenie przed błędną operacją na danych już na poziomie samej bazy danych. Taki sposób zabezpieczeń zdejmuje też szereg obowiązków z programistów tworzących interfejs aplikacji bazodanowej, gdyż sama baza danych chroni zawarte w niej informacje przed niesankcjonowanym dostępem do nich. Nadanie uprawnień jedynie do widoków stanowi kolejny stopień w procesie zabezpieczania danych. Pozwala to ukryć poszczególne kolumny, zaś z wielu tabel utworzyć jedną zbiorczą, zawierającą skondensowane lub tylko wybrane informacje.

Ostatnim elementem budowy aplikacji bazodanowej jest wytworzenie interfejsu użytkownika. Najczęściej z bazy danych korzysta się poprzez sieć lokalną, jednakże coraz częściej wymagany jest zdalny dostęp, na przykład poprzez stronę serwisu web. Dostęp zdalny był już prezentowany w poprzednich rozdziałach, teraz wspomnę jedynie o aplikacjach używających ODBC.

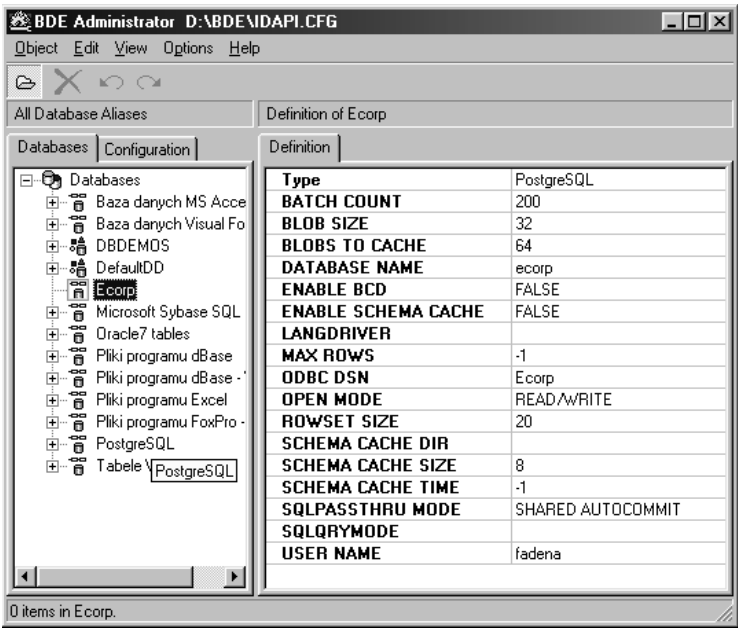
Coraz częściej aplikacje obsługujące interfejs do bazy danych budowane są z użyciem języków czwartej generacji, zawierających generatory aplikacji. Do takich języków należy Visual Basic (rysunek 10.2).



Rysunek 10.2. *Obiekt Data Environment języka Visual Basic z przyłączeniami do bazy danych PostgreSQL*

Szczególnie interesujący jest właśnie Visual Basic, gdyż posiada on cechy obiektowe, mechanizm ADO, jest dość szeroko znany, a przede wszystkim pozwala na zbudowanie interfejsu do bazy danych z użyciem gotowych już aplikacji, takich jak Excel, Word czy też Access. Arkusz kalkulacyjny lub edytor tekstu pełnić będą raczej rolę generatora raportów z bazy danych, natomiast za pomocą Accessa możliwe jest wytworzenie pełnowartościowego interaktywnego interfejsu do relacyjnej i dostępnej poprzez sieć bazy danych PostgreSQL.

Innym, równie ciekawym jak Visual Basic językiem tworzenia aplikacji bazodanowych jest Delphi. Język ten wykorzystuje mechanizm BDE (*Borland Database Engine*) jako jednolity dla wszystkich aplikacji interfejs dostępu do bazy danych (rysunek 10.3).

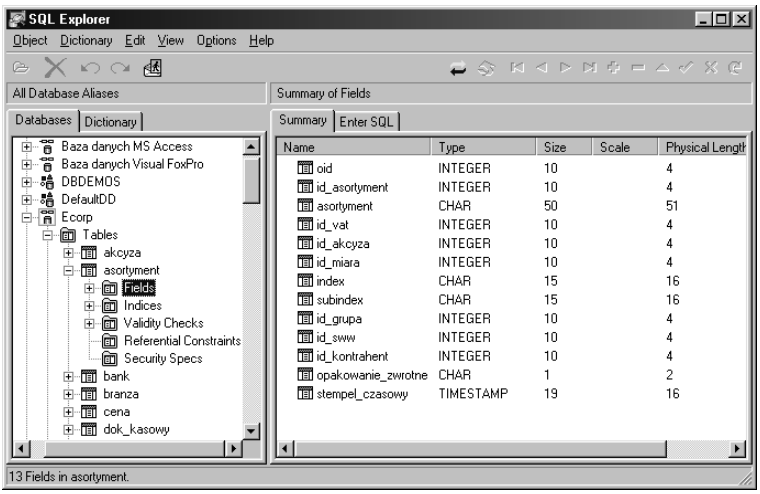


Rysunek 10.3. *Administrator BDE z definicją łącza do bazy danych używającego ODBC PostgreSQL*

Przedstawianie mechanizmów programowych w Delphi do łączenia się aplikacji z bazą danych wykracza poza zakres tej książki. Należy jedynie podkreślić, że za pomocą znanych programistom narzędzi można szybko i wydajnie tworzyć interfejsy użytkownika.

Przedstawione powyżej przykłady prezentują PostgreSQL w zupełnie innym świetle. Wysoko wydajna baza danych, prosta w obsłudze, ze zgodnym ze standardem językiem SQL oraz wykorzystująca do przekazywania danych mechanizmy DBI, ODBC, JDBC może stanowić solidną podstawę dla programów komercyjnych.

W trakcie pracy programistycznej nad interfejsem użytkownika często zachodzi potrzeba sprawdzenia wielu zależności w bazie danych. Można by powiedzieć, że im bardziej rozbudowana aplikacja, tym częściej używane są różnego typu monitory bazy danych w celu śledzenia zależności pomiędzy obiektami i zawartości tych obiektów. Delphi jako jeden z nielicznych języków programowania dostarczany jest wraz z uniwersalnym monitorem relacyjnych baz danych (rysunek 10.4).



Rysunek 10.4. *SQL Explorer pozwala na bieżąco kontrolować strukturę i zawartość bazy danych PostgreSQL*

Taki podgląd oraz wykorzystanie włączonych lub dołączonych do PostgreSQL języków programowania umożliwiają stworzenie procedur i przechowywanie ich wewnątrz bazy danych. Dodając do tego mechanizmy obsługi zdarzeń oraz wyzwalacze (trigery), można umieścić znaczną część kodu aplikacji bazodanowej wewnątrz samej bazy danych. Przyspiesza to działanie samej bazy danych i zmniejsza obciążenie łączy do bazy. W efekcie kod programów staje się prostszy i bardziej przejrzysty, łatwiej budować nowe wersje aplikacji, zaś sam program pracuje szybciej i jest zdecydowanie stabilniejszy.

Pomimo że taka metodologia programistyczna przy budowie aplikacji nie jest zbyt często stosowana, jednakże, jak pokazują doświadczenia wielu niezależnych programistów, zwiększa się w ten sposób produktywność pisania programów, co jest niezwykle ważne w świecie, gdzie użytkownicy i firmy żądają ciągle nowych lub ulepszonych aplikacji.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

Ś

T

U

V

W

X

Z

Ż

PostgreSQL Wydanie II

PostgreSQL to jeden z najpopularniejszych systemów zarządzania relacyjnymi bazami danych. To bezpłatne oprogramowanie jest rozwijane i wykorzystywane przez użytkowników na całym świecie. Cieszy się wsparciem dużych przedsiębiorstw tworzących rozwiązania bazodanowe i znajduje zastosowanie zarówno w przypadku niewielkich inicjatyw prywatnych, jak i poważnych środowisk komercyjnych. Doceniane za swoje możliwości, stabilność i wydajność działania oraz zgodność ze standardami, stanowi także podstawę nauczania teorii i praktyki w dziedzinie nowoczesnych systemów baz danych na wielu najbardziej prestiżowych uczelniach technicznych.

PostgreSQL jest niewątpliwie systemem, który musisz poznać, jeśli planujesz rozpocząć karierę administratora baz danych, wybierasz taką specjalizację na studiach informatycznych lub po prostu chcesz się zorientować, na czym polega praca z relacyjnymi bazami danych w praktyce na przykładzie stale rozwijanego, popularnego środowiska. Dobrze będzie skorzystać przy tym z książki *PostgreSQL. Wydanie II*. Prezentuje ona niezbędną teorię, stojącą za działaniem nowoczesnych baz danych, oraz strukturę i sposób korzystania z języka SQL. Znajdziesz tu także zagadnienia związane z tworzeniem i używaniem bazy opartej na systemie PostgreSQL oraz dotyczące budowania aplikacji bazodanowych.

- **Podstawy teoretyczne relacyjnego modelu bazy danych**
- **Struktura, możliwości i zastosowanie języka SQL**
- **Przegląd poleceń języka SQL**
- **Instalacja bazy danych PostgreSQL i zarządzanie nią**
- **Metody dostępu do danych przechowywanych w bazie**
- **Tworzenie aplikacji bazodanowych**
- **Korzystanie z języków proceduralnych**
- **Przechowywanie różnych rodzajów danych w bazie PostgreSQL**

Postaw na bezpłatne rozwiązania.
Postaw na PostgreSQL!

sięgnij po WIĘCEJ

KOD KORZYŚCI

helion.pl księgarnia internetowa

Nr katalogowy: 7728

Księgarnia internetowa: http://helion.pl

Zamówienia telefoniczne: 0 801 339900 0 601 339900

Helion

Sprawdź najnowsze promocje: http://helion.pl/promocje
Książki najchętniej czytane: http://helion.pl/bestsellery
Zamów informacje o nowościach: http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-246-3068-4

9 788324 630684

Cena 39,00 zł

Informatyka w najlepszym wydaniu