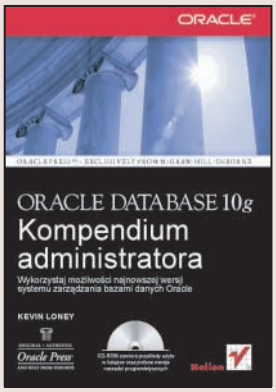
Oracle Database 10g. Kompendium administratora



Autor: Kevin Loney
Tłumaczenie: Zbigniew Banach, Sławomir Dzieniszewski, Paweł Gonera, Radosław Meryk
ISBN: 83-7361-750-7
Tytuł oryginału: Oracle Database 10g. The Complete Reference
Format: B5, stron: 1480

Baza danych Oracle od dawna cieszy się zasłużoną sławą. Jest wykorzystywana wszędzie tam, gdzie dba się o stabilność i bezpieczeństwo danych oraz szybkość dostępu do nich. Każda nowa wersja Oracle'a wnosi coś nowego i wytycza nowe standardy. Ogromne możliwości Oracle'a pociągają za sobą konieczność dołączania do niej tysięcy stron dokumentacji. Każdy z opasłych tomów instrukcji szczegółowo opisuje inne elementy systemu. Często jednak podczas pracy z bazą zachodzi konieczność szybkiego odnalezienia konkretnej informacji. W takich przypadkach przydatne okazuje się zestawienie najbardziej istotnych zagadnień, zebranych w jednej publikacji.

W książce „Oracle Database 10g. Kompendium administratora" zebrano wszystkie najważniejsze pojęcia dotyczące bazy danych Oracle. W jednym podręczniku zgromadzone są opisy poleceń, funkcji i właściwości oraz dokumentacja narzędzi dołączanych do Oracle'a. Każdy użytkownik, administrator i programista baz danych znajdzie tu coś, co przyda mu się w pracy. Jednych zainteresuje opis języka SQL, innych – opis instalacji, konfiguracji i strojenia bazy, a jeszcze inni docenią omówienie zasad tworzenia aplikacji współpracujących z Oracle'em.

- Instalacja bazy danych Oracle 10g
- Planowanie i projektowanie aplikacji bazodanowych
- Język SQL i narzędzie SQL*Plus
- Operacje na danych z wykorzystaniem języka SQL
- Budowanie złożonych zapytań
- Zarządzanie tabelami, perspektywami, indeksami i klastrami
- Mechanizmy bezpieczeństwa bazy danych
- Eksport danych i technologia Data Pump
- Zapytania flashback
- Dołączanie tabel zewnętrznych
- Tworzenie aplikacji w języku PL/SQL
- Strojenie aplikacji i optymalizacja zapytań

Dodatkową pomocą dla użytkowników Oracle'a jest przewodnik po wszystkich jej funkcjach, potencjalnych zastosowaniach i zestawienie poleceń wraz z opcjami i parametrami.

Ta książka powinna znaleźć się na biurku każdego, kto wykorzystuje w pracy bazę Oracle 10g

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 4.

Planowanie aplikacji systemu Oracle — sposoby, standardy i zagrożenia

Aby stworzyć aplikację systemu Oracle i szybko oraz efektywnie z niej korzystać, użytkownicy i programiści muszą posługiwać się wspólnym językiem, a także posiadać głęboką wiedzę zarówno na temat aplikacji biznesowych, jak i narzędzi systemu Oracle. W poprzednich rozdziałach zaprezentowano ogólny opis systemu Oracle oraz sposoby jego instalacji i aktualizacji. Teraz, po zainstalowaniu oprogramowania możemy przystąpić do tworzenia aplikacji. Kluczowym elementem w tym przypadku jest ścisła współpraca menedżerów i personelu technicznego. Obie grupy pracowników powinny orientować się w zadaniach firmy oraz wiedzieć, jakie dane są przetwarzane w codziennym działaniu.

Dawniej analitycy systemowi szczegółowo badali wymagania klienta, a następnie programiści tworzyli aplikacje, które spełniały te wymagania. Klient dostarczał jedynie opis procesu, który aplikacja miała usprawnić, oraz testował jej działanie. Dzięki najnowszym narzędziom systemu Oracle można tworzyć aplikacje znacznie lepiej odpowiadające potrzebom i przyzwyczajeniom użytkowników. Jest to jednak możliwe tylko w przypadku właściwego rozumienia zagadnień biznesowych.

Zarówno użytkownicy, jak i programiści powinni zmierzać do maksymalnego wykorzystania możliwości systemu Oracle. Użytkownik aplikacji ma wiedzę na temat zagadnień merytorycznych, której nie posiada programista. Programista rozumie działanie wewnętrznych funkcji i własności systemu Oracle i środowiska komputerów, które są zbyt skomplikowane dla użytkownika. Ale takie obszary wyłączności wiedzy nie są liczne. Podczas korzystania z systemu Oracle użytkownicy i programiści zwykle poruszają się w obrębie zagadnień znanych obu stronom.

Nie jest tajemnicą, że pracownicy „merytoryczni” i „techniczni” od lat nie darzą się szczególną sympatią. Przyczyną tego stanu są różnice w posiadanej wiedzy, zainteresowaniach i zwyczajach, a także inne cele. Nie bez znaczenia jest także poczucie odrębności,

jakie powstaje w wyniku fizycznego oddzielenia obu grup. Mówiąc szczerze, te zjawiska nie są wyłącznie domeną osób zajmujących się przetwarzaniem danych. Podobne problemy dotyczą na przykład pracowników działu księgowości, którzy często pracują na różnych piętrach, w oddzielnych budynkach, a nawet w innych miastach. Relacje pomiędzy członkami fizycznie odizolowanych grup stają się formalne, sztywne i dalekie od normalności. Powstają sztuczne bariery i procedury, które jeszcze bardziej potęgują syndrom izolacji.

Można by powiedzieć, że to, co zostało napisane powyżej, jest interesujące dla socjologów. Dlaczego więc przypominamy te informacje przy okazji systemu Oracle? *Ponieważ wdrożenie tego systemu fundamentalnie zmienia naturę związków zachodzących pomiędzy pracownikami merytorycznymi a technicznymi*. W systemie Oracle nie używa się specyficznego języka, który rozumieją tylko profesjonaliści. System ten może opanować każdy i każdy może go używać. Informacje, wcześniej więzione w systemach komputerowych pod czujnym okiem ich administratorów, są teraz dostępne dla menedżerów, którzy muszą jedynie wpisać odpowiednie zapytanie. Ta sytuacja znacząco zmienia obowiązujące reguły gry.

Od momentu wdrożenia systemu Oracle obydwa obozy znacznie lepiej się rozumieją, normalizując zachodzące pomiędzy nimi relacje. Dzięki temu powstają lepsze aplikacje.

Już pierwsze wydanie systemu Oracle bazowało na zrozumiałym modelu relacyjnym (który wkrótce zostanie szczegółowo omówiony). Osoby, które nie są programistami, nie mają problemów ze zrozumieniem zadań wykonywanych przez system Oracle. Dzięki temu jest on dostępny w stopniu praktycznie nieograniczonym.

Niektóre osoby nie rozumieją, jak ważną rzeczą jest, aby runęły przestarzałe i sztuczne bariery pomiędzy użytkownikami i systemowcami. Z pewnością jednak metoda kooperacyjna korzystnie wpływa na jakość i użyteczność tworzonych aplikacji.

Jednak wielu doświadczonych projektantów wpada w pułapkę: pracując z systemem Oracle, usiłują stosować metody sprawdzone w systemach poprzedniej generacji. O wielu z nich powinni zapomnieć, gdyż będą nieskuteczne.

Niektóre techniki (i ograniczenia), które były stałym elementem systemów poprzedniej generacji, teraz nie tylko są zbyteczne, ale nawet mają ujemny wpływ na działanie aplikacji. W procesie poznawania systemu Oracle należy pozbyć się większości starych nawyków i bezużytecznych metod. Od teraz są dostępne nowe odświeżające możliwości.

Założeniem tej książki jest prezentacja systemu Oracle w sposób jasny i prosty — z wykorzystaniem pojęć, które są zrozumiałe zarówno dla użytkowników, jak i programistów. Omawiając system, wskazano przestarzałe i niewłaściwe techniki projektowania i zarządzania oraz przedstawiono alternatywne rozwiązania.

Podejście kooperacyjne

Oracle jest obiektowo-relacyjnym systemem baz danych. Relacyjna baza danych to niezwykle prosty sposób przedstawiania i zarządzania danymi wykorzystywanymi w biznesie. Model relacyjny to nic innego, jak kolekcja tabel danych. Z tabelami wszyscy

spotykamy się na co dzień, czytając na przykład raporty o pogodzie lub wyniki sportowe. Wszystko to są tabele z wyraźnie zaznaczonymi nagłówkami kolumn i wierszy. Pomimo swojej prostoty model relacyjny wystarcza do prezentowania nawet bardzo złożonych zagadnień. Obiektowo-relacyjna baza danych charakteryzuje się wszystkimi własnościami relacyjnej bazy danych, a jednocześnie ma cechy modelu obiektowego. Oracle można wykorzystać zarówno jako relacyjny system zarządzania bazą danych (RDBMS), jak też skorzystać z jego własności obiektowych.

Niestety jedyni ludzie, którym przydaje się relacyjna baza danych — użytkownicy biznesowi — z reguły najmniej ją rozumieją. Projektanci aplikacji tworzący systemy dla użytkowników biznesowych często nie potrafią objaśnić pojęć modelu relacyjnego w prosty sposób. Aby była możliwa współpraca, potrzebny jest wspólny język.

Za chwilę wyjaśnimy, czym są relacyjne bazy danych i w jaki sposób wykorzystuje się je w biznesie. Może się wydawać, że materiał ten zainteresuje wyłącznie „użytkowników". Doświadczony projektant aplikacji relacyjnych odczuje zapewne pokusę pominięcia tych fragmentów, a książkę wykorzysta jako dokumentację systemu Oracle. Chociaż większość materiału z pierwszych rozdziałów to zagadnienia elementarne, to jednak projektanci, którzy poświęcą im czas, poznają jasną, spójną i funkcjonalną terminologię, ułatwiającą komunikację z użytkownikami oraz precyzyjniejsze ustalenie ich wymagań.

Ważne jest również pozbycie się niepotrzebnych i prawdopodobnie nieświadomie stosowanych przyzwyczajeń projektowych. Wiele z nich zidentyfikujemy podczas prezentacji modelu relacyjnego. Trzeba sobie uświadomić, że nawet możliwości tak rozbudowanego systemu, jak Oracle można zniweczyć, stosując metody właściwe dla projektów nierelacyjnych.

Gdy użytkownik docelowy rozumie podstawowe pojęcia obiektowo-relacyjnych baz danych, może w spójny sposób przedstawić wymagania projektantom aplikacji. Pracując w systemie Oracle, jest w stanie przetwarzać informacje, kontrolować raporty i dane oraz niczym prawdziwy ekspert zarządzać własnościami tworzenia aplikacji i zapytań. Świadomy użytkownik, który rozumie funkcjonowanie aplikacji, z łatwością *zorientuje się*, czy osiągnęła ona maksymalną wydajność.

System Oracle uwalnia także programistów od najmniej lubianego przez nich obowiązku: tworzenia raportów. W dużych firmach niemal 95 % wszystkich prac programistycznych to żądania tworzenia nowych raportów. Ponieważ dzięki systemowi Oracle użytkownicy tworzą raport w kilka minut, a nie w kilka miesięcy, spełnianie takiego obowiązku staje się przyjemnością.

Dane są wszędzie

W bibliotece znajdują się informacje o czytelnikach, książkach i karach za nieterminowy zwrot. Właściciel kolekcji kart baseballowych zbiera informacje o graczach, notuje daty i wyniki meczów, interesuje się wartością kart. W każdej firmie muszą być przechowywane rejestry z informacjami o klientach, produktach czy cenach. Informacje te określa się jako *dane*.

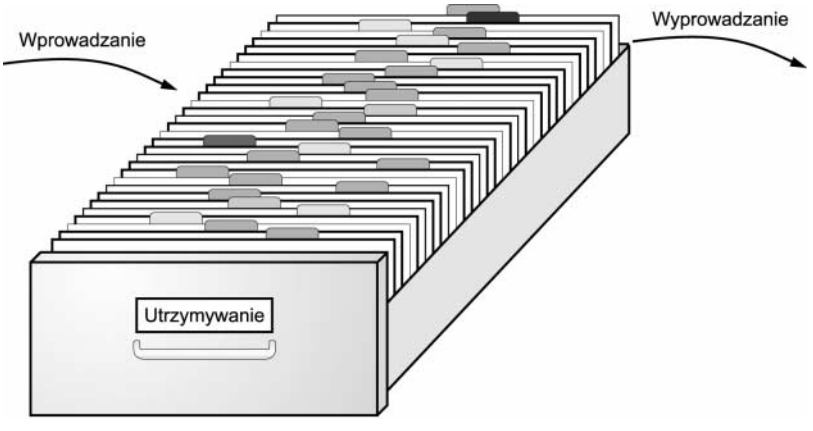
Teoretycy często mówią, że dane pozostają danymi, dopóki nie zostaną odpowiednio zorganizowane. Wtedy stają się informacjami. Jeśli przyjąć taką tezę, system Oracle śmiało można nazwać mechanizmem wytwarzania informacji, gdyż bazując na surowych danych, potrafi na przykład wykonywać podsumowania lub pomaga identyfikować trendy handlowe. Jest to wiedza, z istnienia której nie zawsze zdajemy sobie sprawę. W niniejszej książce wyjaśnimy, jak ją uzyskiwać.

Po opanowaniu podstaw obsługi systemu Oracle można wykonywać obliczenia z danymi, przenosić je z miejsca na miejsce i modyfikować. Takie działania nazywa się *przetwarzaniem* danych. Rzecz jasna, przetwarzać dane można również, wykorzystując ołówek, kartkę papieru i kalkulator, ale w miarę jak rosną zbiory danych, trzeba sięgnąć po komputery.

System zarządzania relacyjnymi bazami danych (ang. *Relational Database Management System* — RDBMS) taki, jak Oracle umożliwia wykonywanie zadań w sposób zrozumiały dla użytkownika i stosunkowo nieskomplikowany. Mówiąc w uproszczeniu, system Oracle pozwala na:

- wprowadzanie danych do systemu,
- utrzymywanie (przechowywanie) danych,
- wyprowadzanie danych i posługiwanie się nimi.

Schemat tego procesu pokazano na rysunku 4.1.



Rysunek 4.1. *Dane w systemie Oracle*

W systemie Oracle sposób postępowania z danymi można opisać schematem wprowadzanie-utrzymywanie-wyprowadzanie. System dostarcza inteligentnych narzędzi pozwalających na stosowanie wyrafinowanych metod pobierania, edycji, modyfikacji i wprowadzania danych, zapewnia ich bezpieczne przechowywanie, a także wyprowadzanie, na przykład w celu tworzenia raportów.

Język systemu Oracle

Informacje zapisane w systemie Oracle są przechowywane w tabelach. W podobny sposób są prezentowane w gazetach na przykład informacje o pogodzie (rysunek 4.2).

Rysunek 4.2.
Dane w gazetach często podawane są w tabelach

POGODA

Miasto	Temperatura	Wilgotność	Warunki
Ateny.....	36	89	Słonecznie
Chicago.....	19	88	Deszcz
Lima.....	7	79	Deszcz
Manchester...	19	98	Mgła
Paryż.....	27	62	Pochmurno
Sparta.....	23	63	Pochmurno
Sydney.....	21	99	Słonecznie

Tabela pokazana na rysunku składa się czterech kolumn: *Miasto*, *Temperatura*, *Wilgotność* i *Warunki*. Zawiera także wiersze dla poszczególnych miast — od Aten do Sydney — oraz nazwę: *POGODA*. Kolumny, wiersze i nazwa to trzy główne cechy drukowanych tabel. Podobnie jest w przypadku tabel z relacyjnych baz danych. Wszyscy z łatwością rozumieją pojęcia używane do opisu tabeli w bazie danych, ponieważ takie same pojęcia stosuje się w życiu codziennym. Nie kryje się w nich żadne specjalne, niezwykłe czy tajemnicze znaczenie. To, co widzimy, jest tym, na co wygląda.

Tabele

W systemie Oracle informacje są zapisywane w tabelach. Każda tabela składa się z jednej lub większej liczby kolumn. Odpowiedni przykład pokazano na rysunku 4.3. Nagłówki: *Miasto*, *Temperatura*, *Wilgotność* i *Warunki* wskazują, jaki rodzaj informacji przechowuje się w kolumnach. Informacje są zapisywane w wierszach (miasto po mieście). Każdy niepowtarzalny zestaw danych, na przykład temperatura, wilgotność i warunki dla miasta Manchester, jest zapisywany w osobnym wierszu.

Rysunek 4.3.
Tabela POGODA w systemie Oracle

Nazwa tabeli

Kolumna

POGODA

Miasto	Temperatura	Wilgotność	Warunki
ATENY	36	89	SŁONECZNIE
CHICAGO	19	88	DESZCZ
LIMA	7	79	DESZCZ ← Wiersz
MANCHESTER	19	98	MGŁA
PARYŻ	27	62	POCHMURNO
SPARTA	23	63	POCHMURNO
SYDNEY	21	99	SŁONECZNIE

Aby produkt był bardziej dostępny, firma Oracle unika stosowania specjalistycznej, akademickiej terminologii. W artykułach o relacyjnych bazach danych kolumny czasami określa się jako „atrybuty", wiersze jako „krotki", a tabele jako „encje". Takie pojęcia są jednak mylące dla użytkownika. Często terminy te są tylko niepotrzebnymi zamiennikami dla powszechnie zrozumiałych nazw z języka ogólnego. Firma Oracle stosuje ogólny język, a zatem mogą go również stosować programiści. Trzeba pamiętać, że niepotrzebne stosowanie technicznego żargonu stwarza barierę braku zaufania i niezrozumienia. W tej książce, podobnie jak to uczyniono w dokumentacji systemu Oracle, konsekwentnie posługujemy się pojęciami „tabele", „kolumny" i „wiersze".

Strukturalny język zapytań

Firma Oracle jako pierwsza zaczęła stosować *strukturalny język zapytań* (ang. *Structured Query Language* — SQL). Język ten pozwala użytkownikom na samodzielne wydobywanie informacji z bazy danych. Nie muszą sięgać po pomoc fachowców, aby sporządzić choćby najmniejszy raport.

Język zapytań systemu Oracle ma swoją strukturę, podobnie jak język angielski i dowolny inny język naturalny. Ma również reguły gramatyczne i składnię, ale są to zasady bardzo proste i ich zrozumienie nie powinno przysparzać większych trudności.

Język SQL, którego nazwę wymawia się jako *sequel* lub *es-ku-el*, oferuje, jak wkrótce się przekonamy, niezwykłe wręcz możliwości. Korzystanie z niego nie wymaga żadnego doświadczenia w programowaniu.

Oto przykład, jak można wykorzystywać SQL. Gdyby ktoś poprosił nas, abyśmy w tabeli POGODA wskazali miasto, gdzie wilgotność wynosi 89 %, szybko wymienilibyśmy Ateny. Gdyby ktoś poprosił nas o wskazanie miast, gdzie temperatura wyniosła 19°C, wymienilibyśmy Chicago i Manchester.

System Oracle jest w stanie odpowiadać na takie pytania niemal równie łatwo. Wystarczy sformułować proste zapytania. Słowa kluczowe wykorzystywane w zapytaniach to `select` (wybierz), `from` (z), `where` (gdzie) oraz `order by` (uporządkuj według). Są to wskazówki dla systemu Oracle, które ułatwiają mu zrozumienie żądań i udzielenie poprawnej odpowiedzi.

Proste zapytanie w systemie Oracle

Gdyby w bazie danych Oracle była zapisana przykładowa tabela `POGODA`, pierwsze zapytanie przyjęłoby pokazaną poniżej postać (średnik jest informacją dla systemu, że należy wykonać zapytanie):

```
select Miasto from POGODA where Wilgotnosc = 89;
```

System Oracle odpowiedziałby na nie w taki sposób:

```
Miasto
------------
ATENY
```

Drugie zapytanie przyjęłoby następującą postać:

```
select Miasto from POGODA where Temperatura = 19;
```

W przypadku tego zapytania system Oracle odpowiedziałby tak:

```
Miasto
-----------
MANCHESTER
CHICAGO
```

Jak łatwo zauważyć, w każdym z tych zapytań wykorzystano słowa kluczowe `select`, `from` oraz `where`. A co z klauzulą `order by`? Przypuśćmy, że interesują nas wszystkie miasta uporządkowane według temperatury. Wystarczy, że wprowadzimy takie oto zapytanie:

```
select Miasto, Temperatura from POGODA
  order by Temperatura;
```

— a system Oracle natychmiast odpowie w następujący sposób:

```
Miasto     Temperatura
---------- -----------
LIMA                7
MANCHESTER         19
CHICAGO            19
SYDNEY             21
SPARTA             23
PARYŻ              27
ATENY              36
```

System Oracle szybko uporządkował tabelę od najniższej do najwyższej temperatury. W jednym z kolejnych rozdziałów dowiemy się, jak określić, czy najpierw mają być wyświetlane wyższe, czy niższe wartości.

Zaprezentowane powyżej przykłady pokazują, jak łatwo uzyskuje się potrzebne informacje z bazy danych Oracle, w postaci najbardziej przydatnej dla użytkownika. Można tworzyć skomplikowane żądania o różne dane, ale stosowane metody zawsze będą zrozumiałe. Można na przykład połączyć dwa elementarne słowa kluczowe `where` i `order by` — po to, by wybrać tylko te miasta, w których temperatura przekracza 26 stopni, oraz wyświetlić je uporządkowane według rosnącej temperatury. W tym celu należy skorzystać z następującego zapytania:

```
select Miasto, Temperatura from POGODA
  where Temperatura > 26
  order by Temperatura;
```

System Oracle natychmiast wyświetli następującą odpowiedź:

```
Miasto     Temperatura
---------- -----------
PARYŻ              27
ATENY              36
```

Można stworzyć jeszcze dokładniejsze zapytanie i poprosić o wyświetlenie miast, w których temperatura jest wyższa niż 26 stopni, a wilgotność mniejsza niż 70 %:

```
select Miasto, Temperatura, Wilgotnosc from POGODA
  where Temperatura > 26
  and Wilgotnosc < 70
  order by Temperatura;
```

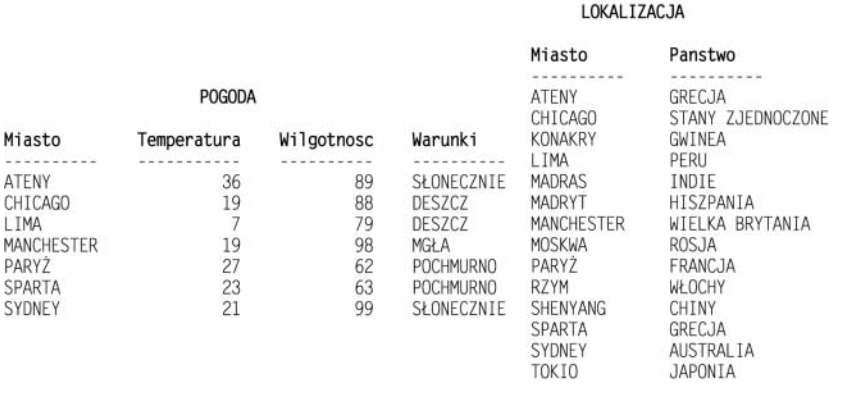
— a system Oracle natychmiast odpowie w następujący sposób:

```
Miasto        Temperatura  Wilgotnosc
------------- ------------ ----------
PARYŻ                  27         62
```

Dlaczego system baz danych nazywa się „relacyjnym”?

Zwróćmy uwagę, że w tabeli POGODA wyświetlają się miasta z kilku państw, przy czym w przypadku niektórych państw w tabeli znajduje się więcej niż jedno miasto. Przypuśćmy, że interesuje nas, w którym państwie leży określone miasto. W tym celu można utworzyć oddzielną tabelę LOKALIZACJA zawierającą informacje o zlokalizowaniu miast (rysunek 4.4).

Rysunek 4.4. *Tabele POGODA i LOKALIZACJA*



POGODA

Miasto	Temperatura	Wilgotnosc	Warunki
ATENY	36	89	SŁONECZNIE
CHICAGO	19	88	DESZCZ
LIMA	7	79	DESZCZ
MANCHESTER	19	98	MGŁA
PARYŻ	27	62	POCHMURNO
SPARTA	23	63	POCHMURNO
SYDNEY	21	99	SŁONECZNIE

LOKALIZACJA

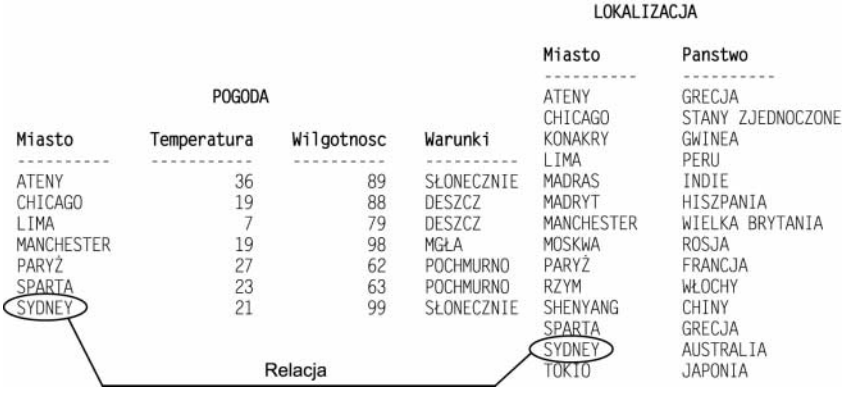
Miasto	Panstwo
ATENY	GRECJA
CHICAGO	STANY ZJEDNOCZONE
KONAKRY	GWINEA
LIMA	PERU
MADRAS	INDIE
MADRYT	HISZPANIA
MANCHESTER	WIELKA BRYTANIA
MOSKWA	ROSJA
PARYŻ	FRANCJA
RZYM	WŁOCHY
SHENYANG	CHINY
SPARTA	GRECJA
SYDNEY	AUSTRALIA
TOKIO	JAPONIA

Każde miasto z tabeli POGODA znajduje się również w tabeli LOKALIZACJA. Wystarczy odnaleźć interesującą nas nazwę w kolumnie Miasto, a wówczas w kolumnie Panstwo w tym samym wierszu znajdziemy nazwę państwa.

Są to całkowicie odrębne i niezależne od siebie tabele. Każda z nich zawiera własne informacje zestawione w kolumnach i wierszach. Tabele mają część wspólną: kolumnę Miasto. Dla każdej nazwy miasta w tabeli POGODA istnieje identyczna nazwa miasta w tabeli LOKALIZACJA.

Spróbujmy na przykład dowiedzieć się, jakie temperatury, wilgotność i warunki atmosferyczne panują w miastach australijskich. Spójrzmy na obie tabele i odczytajmy z nich te informacje.

Rysunek 4.5. *Relacje pomiędzy tabelami POGODA i LOKALIZACJA*



W jaki sposób to zrobiliśmy? W tabeli LOKALIZACJA znajduje się tylko jeden zapis z wartością AUSTRALIA w kolumnie Panstwo. Obok niego, w tym samym wierszu w kolumnie Miasto figuruje nazwa miasta SYDNEY. Po odnalezieniu nazwy SYDNEY w kolumnie Miasto tabeli POGODA przesunęliśmy się wzdłuż wiersza i znaleźliśmy wartości pól Temperatura, Wilgotnosc i Warunki. Były to odpowiednio: 21, 99 i SŁONECZNIE.

Nawet jeśli tabele są niezależne, z łatwością można spostrzec, że są z sobą powiązane. Nazwa miasta w jednej tabeli jest *powiązana* (pozostaje w *relacji*) z nazwą miasta w drugiej (patrz: rysunek 4.5 powyżej). Relacje pomiędzy tabelami tworzą podstawę nazwy *relacyjna baza danych* (czasami mówi się też o *relacyjnym modelu danych*).

Dane zapisuje się w tabelach, na które składają się kolumny, wiersze i nazwy. Tabele mogą być z sobą powiązane, jeśli w każdej z nich znajduje się kolumna o takim samym typie informacji. To wszystko. Nie ma w tym nic skomplikowanego.

Proste przykłady

Kiedy zrozumiemy podstawowe zasady rządzące relacyjnymi bazami danych, wszędzie zaczynamy widzieć tabele, wiersze i kolumny. Nie oznacza to, że wcześniej ich nie widzieliśmy, ale prawdopodobnie nie myśleliśmy o nich w taki sposób. W systemie Oracle można zapisać wiele tabel i wykorzystać je do szybkiego uzyskania odpowiedzi na pytania.

Typowy raport kursów akcji w postaci papierowej może wyglądać tak, jak ten, który zaprezentowano na rysunku 4.6. Jest to fragment wydrukowanego gęstą czcionką alfabetycznego zestawienia wypełniającego szereg wąskich kolumn na kilku stronach w gazecie. Jakich akcji sprzedano najwięcej? Które akcje odnotowały najwyższy procentowy wzrost, a które największy spadek? W systemie Oracle odpowiedzi na te pytania można uzyskać za pomocą prostych zapytań. Jest to działanie o wiele szybsze niż przeszukiwanie kolumn cyfr w gazecie.

Na rysunku 4.7 pokazano indeks gazety. Co można znaleźć w sekcji F? W jakim porządku będziemy czytać artykuły, jeśli wertujemy gazetę od początku do końca? W systemie Oracle odpowiedzi na te pytania można uzyskać z pomocą prostych zapytań. Nauczymy się, jak formułować takie zapytania, a nawet tworzyć tabele do zapisywania informacji.

W przykładach użytych w tej książce wykorzystano dane i obiekty, z którymi często spotykamy się na co dzień — w pracy i w domu. Dane do wykorzystania w ćwiczeniach można znaleźć wszędzie. Na kolejnych stronach pokażemy, w jaki sposób wprowadza się i pobiera dane. Przykłady bazują na spotykanych na co dzień źródłach danych.

Podobnie jak w przypadku każdej nowej technologii, nie wolno dostrzegać tylko jej zalet, trzeba również widzieć zagrożenia. Jeśli skorzystamy z relacyjnej bazy danych oraz szeregu rozbudowanych i łatwych w użyciu narzędzi dostarczanych z systemem Oracle, niebezpieczeństwo, że padniemy ofiarą tej prostoty, stanie się bardzo realne. Dodajmy do tego właściwości obiektowe i łatwość wykorzystania w internecie, a zagrożenia staną się jeszcze większe. W kolejnych punktach przedstawimy niektóre zagrożenia związane z korzystaniem z relacyjnych baz danych, o których powinni pamiętać zarówno użytkownicy, jak i projektanci.

Firma	Wczorajsza cena zamknięcia	Dzisiejsza cena zamknięcia	Sprzedano akcji
Auto alarmy SA	31.75	31.75	18,333,876
ABC	33.75	36.50	25,787,229
ATLETA SA	46.75	48.00	11,398,323
Aukcje online	15.00	15.00	12,221,711
Bramy i ogrodzenia	32.75	33.50	25,789,769
Geodezja SA	64.25	66.00	7,598,562
Gongi i dzwonki	22.75	27.25	22,533,944
Harmonie i akordeony	104.25	106.00	3,358,561
IDK	95.00	95.25	9,443,523
Indyjskie Kosmetyki	30.75	30.75	8,134,878
INTERPROMOCJA	13.25	13.75	22,112,171
KDK	80.00	85.25	7,481,566
Kosmetyki Lidia	18.25	19.50	6,636,863
Lubelskie Fabryki Wag	21.50	22.00	3,341,542
Lokalne Oprogramowanie	26.75	27.25	2,596,934
Matylda i spółka	8.25	8.00	2,836,893
MPK	43.25	41.00	10,022,980
Maszyny rolnicze SA	15.50	14.25	4,557,992
Malborska Firma Handlowa	77.00	76.50	25,205,667
Nowotex SA	13.50	14.25	14,222,692
Nadmorska fabryka sieci	26.75	28.00	1,348,323
Opolskie Przedsiębiorstwo Maszynowe	21.50	22.00	7,052,990
Oskar Kamiński i spółka	87.00	88.50	25,798,992
Roman Jankowski i bracia	23.25	24.00	19,032,481
Starachowicka Fabryka Grzebieni	16.25	16.75	22,574,879
Wagony kolejowe	5.00	5.00	2,553,712

Rysunek 4.6. *Tabela kursów akcji*

Rysunek 4.7. *Tabela danych o sekcjach w gazecie*

Rubryka	Sekcja	Strona
Biznes	E	1
Brydż	B	2
Doktor radzi	F	6
Filmy	B	4
Komiks	C	4
Narodziny	F	7
Nekrologi	F	6
Nowoczesny styl	B	1
Od redakcji	A	12
Ogłoszenia drobne	F	8
Pogoda	C	2
Sport	D	1
Telewizja	B	7
Wiadomości z kraju	A	1

Zagrożenia

Podstawowym zagrożeniem podczas projektowania aplikacji wykorzystujących relacyjne bazy danych jest... prostota tego zadania. Zrozumienie, czym są tabele, kolumny i wiersze nie sprawia kłopotów. Związki pomiędzy dwoma tabelami są łatwe do uchwycenia. Nawet *normalizacji* — procesu analizy wewnętrznych „normalnych" relacji pomiędzy poszczególnymi danymi — można z łatwością się nauczyć.

W związku z tym często pojawiają się „eksperci", którzy są bardzo pewni siebie, ale mają niewielkie doświadczenie w tworzeniu rzeczywistych aplikacji o wysokiej jakości. Gdy w grę wchodzi niewielka baza z danymi marketingowymi lub domowy indeks, nie ma to wielkiego znaczenia. Popełnione błędy ujawnią się po pewnym czasie. Będzie to dobra lekcja pozwalająca na uniknięcie błędów w przyszłości. Jednak w przypadku ważnej aplikacji droga ta w prostej linii prowadzi do katastrofy. Brak doświadczenia twórców aplikacji jest często podawany w artykułach prasowych jako przyczyna niepowodzeń ważnych projektów.

Starsze metody projektowania generalnie są wolniejsze, ponieważ zadania takie, jak kodowanie, kompilacja, konsolidacja i testowanie zajmują więcej czasu. Cykl powstawania aplikacji, w szczególności dla komputerów typu mainframe, często jest tak żmudny, że aby uniknąć opóźnień związanych z powtarzaniem pełnego cyklu z powodu błędu w kodzie, programiści dużą część czasu poświęcają na sprawdzanie kodu na papierze.

Narzędzia czwartej generacji kuszą projektantów, aby natychmiast przekazywać kod do eksploatacji. Modyfikacje można implementować tak szybko, że testowanie bardzo często zajmuje niewiele czasu. Niemal całkowite wyeliminowanie etapu sprawdzania przy biurku stwarza problem. Kiedy zniknął negatywny bodziec (długotrwały cykl), który zachęcał do sprawdzania przy biurku, zniknęło także samo sprawdzanie. Wielu programistów wyraża następujący pogląd: „Jeśli aplikacja nie działa właściwie, błąd szybko się poprawi. Jeśli dane ulegną uszkodzeniu, można je szybko zaktualizować. Jeśli metoda okaże się niedostatecznie szybka, dostroi się ją w locie. Zrealizujmy kolejny punkt harmonogramu i pokażmy to, co zrobiliśmy".

Testowanie ważnego projektu Oracle powinno trwać dłużej niż testowanie projektu tradycyjnego, niezależnie od tego, czy zarządza nim doświadczony, czy początkujący menedżer. Musi być także dokładniejsze. Sprawdzić należy przede wszystkim:

- poprawność formularzy wykorzystywanych do wprowadzania danych,
- tworzenie raportów,
- ładowanie danych i uaktualnień,
- integralność i współbieżność danych,
- rozmiary transakcji i pamięci masowej w warunkach maksymalnych obciążeń.

Ponieważ tworzenie aplikacji za pomocą narzędzi systemu Oracle jest niezwykle proste, aplikacje powstają bardzo szybko. Siłą rzeczy czas przeznaczony na testowanie w fazie projektowania skraca się. Aby zachować równowagę i zapewnić odpowiednią jakość produktu, proces planowanego testowania należy świadomie wydłużyć. Choć zazwyczaj problemu tego nie dostrzegają programiści, którzy rozpoczynają dopiero swoją przygodę z systemem Oracle lub z narzędziami czwartej generacji, w planie projektu należy przewidzieć czas i pieniądze na dokładne przetestowanie projektu.

Znaczenie nowego podejścia

Wielu z nas z niecierpliwością oczekuje dnia, kiedy będzie można napisać zapytanie w języku naturalnym i w ciągu kilku sekund uzyskać na ekranie odpowiedź. Jesteśmy o wiele bliżsi osiągnięcia tego celu, niż wielu z nas sobie wyobraża. Czynnikiem ograniczającym nie jest już technika, ale raczej schematy stosowane w projektach aplikacji. Za pomocą systemu Oracle można w prosty sposób tworzyć aplikacje pisane w języku zbliżonym do naturalnego języka angielskiego, które z łatwością eksploatują niezbyt zaawansowani technicznie użytkownicy. W bazie danych Oracle i skojarzonych z nią narzędziach tkwi potencjał, choć jeszcze stosunkowo niewielu programistów wykorzystuje go w pełni.

Podstawowym celem projektanta Oracle powinno być stworzenie aplikacji zrozumiałej i łatwej w obsłudze tak, aby użytkownicy bez doświadczenia programistycznego potrafili pozyskiwać informacje, stawiając proste pytania w języku zbliżonym do naturalnego. Czasami oznacza to intensywniejsze wykorzystanie procesora lub większe zużycie miejsca na dysku. Nie można jednak przesadzać. Jeśli stworzymy aplikację niezwykle łatwą w użyciu, ale tak skomplikowaną programistycznie, że jej pielęgnacja lub usprawnianie okażą się niemożliwe, popełnimy równie poważny błąd. Pamiętajmy także, że nigdy nie wolno tworzyć inteligentnych programów kosztem wygody użytkownika.

Zmiana środowisk

Koszty użytkowania komputerów liczone jako cena miliona instrukcji na sekundę (MIPS) stale spadają w tempie 20 % rocznie. Z kolei koszty siły roboczej ciągle wzrastają. Oznacza to, że wszędzie tam, gdzie ludzi można zastąpić komputerami, uzyskuje się oszczędności finansowe.

W jaki sposób cechę tę uwzględniono w projektowaniu aplikacji? Odpowiedź brzmi: „W jakiś", choć na pewno nie jest to sposób zadowalający. Prawdziwy postęp przyniosło na przykład opracowanie środowiska graficznego przez instytut PARC firmy Xerox, a następnie wykorzystanie go w komputerach Macintosh, w przeglądarkach internetowych oraz w innych systemach bazujących na ikonach. Środowisko graficzne jest znacznie przyjaźniejsze w obsłudze niż starsze środowiska znakowe. Ludzie, którzy z niego korzystają, potrafią stworzyć w ciągu kilku minut to, co wcześniej zajmowało im kilka dni. Postęp w niektórych przypadkach jest tak wielki, że całkowicie utraciliśmy obraz tego, jak trudne były kiedyś pewne zadania.

Niestety wielu projektantów aplikacji nie przyzwyczaiło się do przyjaznych środowisk. Nawet jeśli ich używają, powielają niewłaściwe nawyki.

Kody, skróty i standardy nazw

Problem starych przyzwyczajeń programistycznych staje się najbardziej widoczny, gdy projektant musi przeanalizować kody, skróty i standardy nazewnictwa. Zwykle uwzględnia wówczas jedynie potrzeby i konwencje stosowane przez programistów, zapominając o użytkownikach. Może się wydawać, że jest to suchy i niezbyt interesujący

problem, któremu nie warto poświęcać czasu, ale zajęcie się nim oznacza różnicę pomiędzy wielkim sukcesem a rozwiązaniem takim sobie; pomiędzy poprawą wydajności o rząd wielkości a marginalnym zyskiem; pomiędzy użytkownikami zainteresowanymi i zadowolonymi a zrzędami nękającymi programistów nowymi żądaniami.

Oto, co często się zdarza. Dane biznesowe są rejestrowane w księgach i rejestrach. Wszystkie zdarzenia lub transakcje są zapisywane wiersz po wierszu w języku naturalnym. W miarę opracowywania aplikacji, zamiast czytelnych wartości wprowadza się kody (np. `01` zamiast `Konta przychodowe`, `02` zamiast `Konta rozchodowe` itd). Pracownicy muszą znać te kody i wpisywać je w odpowiednio oznaczonych polach formularzy ekranowych. Jest to skrajny przypadek, ale takie rozwiązania stosuje się w tysiącach aplikacji, przez co trudno się ich nauczyć.

Problem ten najwyraźniej występuje podczas projektowania aplikacji w dużych, konwencjonalnych systemach typu mainframe. Ponieważ do tej grupy należą również relacyjne bazy danych, wykorzystuje się je jako zamienniki starych metod wprowadzania-wyprowadzania danych takich, jak metoda VSAM (ang. *Virtual Storage Access Method*) oraz systemy IMS (ang. *Information Management System*). Wielkie możliwości relacyjnych baz danych są niemal całkowicie marnotrawione, jeśli są wykorzystywane w taki sposób.

Dlaczego zamiast języka naturalnego stosuje się kody?

Po co w ogóle stosować kody? Z reguły podaje się dwa uzasadnienia:

- kategoria zawiera tak wiele elementów, że nie można ich sensownie przedstawić lub zapamiętać w języku naturalnym,
- dla zaoszczędzenia miejsca w komputerze.

Pierwszy powód można uznać za istotny, a poza tym częściej występuje. Wprowadzanie kodów numerycznych zamiast ciągów tekstowych (np. tytułów książek) zwykle jest mniej pracochłonne (o ile oczywiście pracownicy są odpowiednio przeszkoleni), a co za tym idzie tańsze. Ale w systemie Oracle stosowanie kodów oznacza po prostu niepełne wykorzystanie jego możliwości. System Oracle potrafi pobrać kilka pierwszych znaków tytułu i automatycznie uzupełnić pozostałą jego część. To samo potrafi zrobić z nazwami produktów, transakcji (litera „z" może być automatycznie zastąpiona wartością „zakup", a litera „s" wartością „sprzedaż") i innymi danymi w aplikacji. Jest to możliwe dzięki bardzo rozbudowanemu mechanizmowi dopasowywania wzorców.

Drugi powód to już anachronizm. Pamięć operacyjna i masowa były niegdyś tak drogie, a procesory tak wolne (ich moc obliczeniowa nie dorównywała współczesnym nowoczesnym kalkulatorom), że programiści musieli starać się zapisywać dane o jak najmniejszej objętości. Liczby przechowywane w postaci numerycznej zajmują w pamięci o połowę mniej miejsca niż liczby w postaci znakowej, a kody jeszcze bardziej zmniejszają wymagania w stosunku do komputerów.

Ponieważ komputery były kiedyś drogie, programiści *wszędzie* musieli stosować kody. Takie techniczne rozwiązanie problemów ekonomicznych stanowiło prawdziwą udrękę dla użytkowników. Komputery były zbyt wolne i za drogie, aby sprostać wymaganiom ludzi, a zatem szkolono ludzi, aby umieli sprostać wymaganiom komputerów. To dziwactwo było koniecznością.

Ekonomiczne uzasadnienie stosowania kodów znikło wiele lat temu. Komputery są teraz wystarczająco dobre, aby można je było przystosować do sposobu pracy ludzi, a zwłaszcza do używania języka naturalnego. Najwyższy czas, aby tak się stało. A jednak projektanci i programiści, nie zastanawiając się nad tym zbytnio, w dalszym ciągu używają kodów.

Korzyści z wprowadzania czytelnych danych

Stosowanie języka naturalnego przynosi jeszcze jedną korzyść: niemal całkowicie znikają błędy w kluczach, ponieważ użytkownicy natychmiast widzą wprowadzone przez siebie informacje. Cyfry nie są przekształcane, nie trzeba wpisywać żadnych kodów, zatem nie istnieje tu możliwość popełnienia błędu. Zmniejsza się również ryzyko, że użytkownicy aplikacji finansowych stracą pieniądze z powodu błędnie wprowadzonych danych. Aplikacje stają się także bardziej zrozumiałe. Ekrany i raporty zyskują czytelny format, który zastępuje ciągi tajemniczych liczb i kodów.

Odejście od kodów na rzecz języka naturalnego ma olbrzymi i ożywczy wpływ na firmę i jej pracowników. Dla użytkowników, którzy musieli się uczyć kodów, aplikacja bazująca na języku naturalnym oznacza chwilę wytchnienia.

Jak zmniejszyć zamieszanie?

Zwolennicy kodów argumentują czasami, że istnieje zbyt duża liczba produktów, klientów, typów transakcji, aby można było każdej pozycji nadać odrębną nazwę. Dowodzą również, ile kłopotów sprawiają pozycje identyczne bądź bardzo podobne (np. trzydziestu klientów nazywających się „Jan Kowalski"). Owszem, zdarza się, że kategoria zawiera za dużo pozycji, aby można je było łatwo zapamiętać lub rozróżnić, ale znacznie częściej jest to dowód nieodpowiedniego podziału informacji na kategorie: zbyt wiele niepodobnych do siebie obiektów umieszcza się w zbyt obszernej kategorii.

Tworzenie aplikacji zorientowanej na język naturalny wymaga czasu, w którym użytkownicy muszą komunikować się z programistami — trzeba zidentyfikować informacje biznesowe, zrozumieć naturalne relacje i kategorie, a następnie uważnie skonstruować bazę danych i schemat nazw, które w prosty i dokładny sposób odzwierciedlą rzeczywistość. Są trzy podstawowe etapy wykonywania tych działań:

- normalizacja danych,
- wybór nazw dla tabel i kolumn w języku naturalnym,
- wybór nazw danych.

Każdy z tych etapów zostanie omówiony w dalszej części rozdziału. Naszym celem jest projektowanie sensownie zorganizowanych aplikacji, zapisanych w tabelach i kolumnach, których nazwy brzmią znajomo dla użytkownika, gdyż są zaczerpnięte z codzienności.

Normalizacja

Bieżące relacje pomiędzy krajami, wydziałami w firmie albo pomiędzy użytkownikami i projektantami zazwyczaj są wynikiem historycznych uwarunkowań. Ponieważ okoliczności historyczne dawno minęły, w efekcie czasami powstają relacje których nie można

uważać za normalne. Mówiąc inaczej, są one *niefunkcjonalne*. Zaszłości historyczne równie często wywierają wpływ na sposób zbierania, organizowania i raportowania danych, co oznacza, że dane także mogą być nienormalne lub niefunkcjonalne.

Normalizacja jest procesem tworzenia prawidłowego, normalnego stanu. Pojęcie pochodzi od łacińskiego słowa *norma* oznaczającego przyrząd używany przez stolarzy do zapewnienia kąta prostego. W geometrii normalna jest linią prostą przebiegającą pod kątem prostym do innej linii prostej. W relacyjnych bazach danych norma także ma specyficzne znaczenie, które dotyczy przydzielania różnych danych (np. nazwisk, adresów lub umiejętności) do *niezależnych grup* i definiowania dla nich normalnych lub inaczej mówiąc „prawidłowych" relacji.

W normalizacji wyróżnia się kilka postaci: najpopularniejsze są pierwsza, druga i trzecia postać normalna, przy czym trzecia reprezentuje stan najbardziej znormalizowany. Istnieją także postacie czwarta i piąta, ale wykraczają one poza zakres przedstawionego tu wykładu.

Za chwilę zostaną omówione podstawowe zasady normalizacji, dzięki czemu użytkownicy będą mogli uczestniczyć w procesie projektowania aplikacji lub lepiej zrozumieją aplikację, która została stworzona wcześniej. Błędem byłoby jednak myślenie, że proces normalizacji dotyczy jedynie baz danych lub aplikacji komputerowych. Normalizacja to głęboka analiza informacji wykorzystywanych w biznesie oraz relacji zachodzących pomiędzy elementami tych informacji. Umiejętność ta przydaje się w różnych dziedzinach.

Model logiczny

Jedną z pierwszych czynności w procesie analizy jest utworzenie modelu logicznego, czyli po prostu znormalizowanego diagramu danych. Wiedza na temat zasad klasyfikowania danych jest niezbędna do zrozumienia modelu, a model jest niezbędny do stworzenia funkcjonalnej aplikacji.

Rozważmy przypadek z biblioteką. Każda książka ma tytuł, wydawcę, autora lub autorów oraz wiele innych charakterystycznych cech. Przyjmijmy, że dane te posłużą do zaprojektowania dziesięciokolumnowej tabeli w systemie Oracle. Tabelę nazwaliśmy `BIBLIOTECZKA`, a kolumnom nadaliśmy nazwy `Tytul`, `Wydawca`, `Autor1`, `Autor2`, `Autor3` oraz `Kategoria1`, `Kategoria2`, `Kategoria3`, `Ocena` i `OpisOceny`. Użytkownicy tej tabeli już mają problem: jednej książce można przypisać tylko trzech autorów lub tylko trzy kategorie.

Co się stanie, kiedy zmieni się lista dopuszczalnych kategorii? Ktoś będzie musiał przejrzeć wszystkie wiersze tabeli `BIBLIOTECZKA` i poprawić stare wartości. A co się stanie, jeśli jeden z autorów zmieni nazwisko? Ponownie będzie trzeba zmodyfikować wszystkie powiązane rekordy. A co zrobić, jeśli książka ma czterech autorów?

Każdy projektant poważnej bazy danych staje przed problemem sensownego i logicznego zorganizowania informacji. Nie jest to problem techniczny sensu stricto, więc właściwie nie powinniśmy się nim zajmować, ale przecież projektant bazy danych musi się z nim uporać — musi znormalizować dane. Normalizację wykonuje się poprzez stopniową reorganizację danych, tworząc grupy podobnych danych, eliminując niefunkcjonalne relacje i budując relacje normalne.

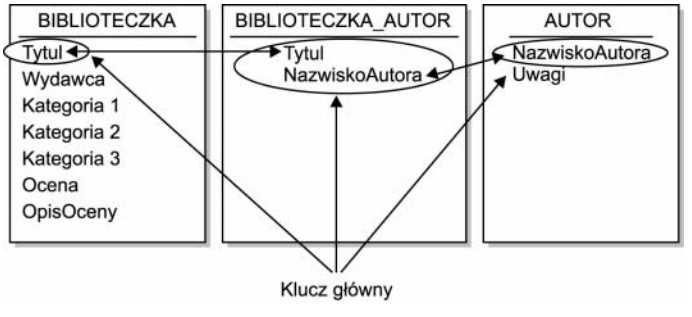
Normalizacja danych

Pierwszym etapem reorganizacji jest sprowadzenie danych do pierwszej postaci normalnej. Polega to na umieszczeniu danych podobnego typu w oddzielnych tabelach i wyznaczeniu w każdej z nich klucza głównego — niepowtarzalnej etykiety lub identyfikatora. Proces ten eliminuje powtarzające się grupy danych takie, jak autorzy książek w tabeli `BIBLIOTECZKA`.

Zamiast definiować tylko trzech autorów książki, dane każdego autora są przenoszone do tabeli, w której każdemu autorowi odpowiada jeden wiersz (imię, nazwisko i opis). Dzięki temu w tabeli `BIBLIOTECZKA` nie trzeba definiować kolumn dla kilku autorów. Jest to lepsze rozwiązanie projektowe, ponieważ umożliwia przypisanie nieograniczonej liczby autorów do książki.

Następną czynnością jest zdefiniowanie w każdej tabeli *klucza głównego* — informacji, która w niepowtarzalny sposób identyfikuje dane, uniemożliwia dublowanie się grup danych i pozwala na wybranie interesującego nas wiersza. Dla uproszczenia załóżmy, że tytuły książek i nazwiska autorów są niepowtarzalne, a zatem kluczem głównym w tabeli `AUTOR` jest kolumna `NazwiskoAutora`.

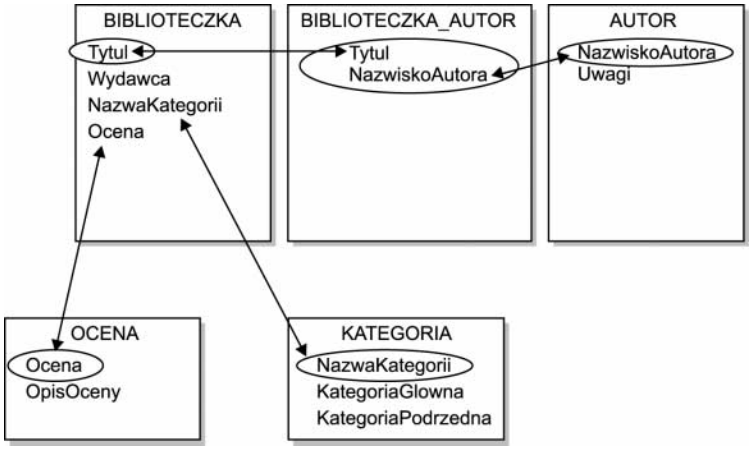
Podzieliliśmy już tabelę `BIBLIOTECZKA` na dwie tabele: `AUTOR` zawierającą kolumny `NazwiskoAutora` (klucz główny) i `Uwagi` oraz `BIBLIOTECZKA`, z kluczem głównym `Tytul` i kolumnami `Wydawca`, `Kategoria1`, `Kategoria2`, `Kategoria3`, `Ocena` i `OpisOceny`. Trzecia tabela `BIBLIOTECZKA_AUTOR` definiuje powiązania. Jednej książce można przypisać wielu autorów, a jeden autor może napisać wiele książek — jest to zatem relacja *wiele do wielu*. Relacje i klucze główne zaprezentowano na rysunku 4.8



Rysunek 4.8. *Tabele BIBLIOTECZKA, AUTOR i BIBLIOTECZKA_AUTOR*

Następna czynność w procesie normalizacji — doprowadzenie danych do drugiej postaci normalnej — obejmuje wydzielenie danych, które zależą tylko od części klucza. Jeśli istnieją atrybuty, które nie zależą od całego klucza, należy je przenieść do nowej tabeli. W tym przypadku kolumna `OpisOceny` w istocie nie zależy od kolumny `Tytul`, zależy natomiast od kolumny `Ocena` i dlatego należy ją przenieść do oddzielnej tabeli.

Aby osiągnąć trzecią postać normalną, należy pozbyć się z tabeli wszystkich atrybutów, które nie zależą wyłącznie od klucza głównego. W zaprezentowanym przykładzie pomiędzy kategoriami zachodzą relacje wzajemne. Nie można wymienić książki jednocześnie w kategorii *Fikcja* i *Fakty*, a w kategoriach *Dorośli* i *Dzieci* można wymienić kilka podkategorii. Z tego powodu informacje o kategoriach należy przenieść do oddzielnej tabeli. Tabele w trzeciej postaci normalnej przedstawia rysunek 4.9.



Rysunek 4.9. *Tabela BIBLIOTECZKA i tabele powiązane*

Dane, które osiągnęły trzecią postać normalną, z definicji znajdują się także w postaciach pierwszej i drugiej. W związku z tym nie trzeba żmudnie przekształcać jednej postaci w drugą. Wystarczy tak zorganizować dane, aby wszystkie kolumny w każdej tabeli (oprócz klucza głównego) zależały wyłącznie od *całego klucza głównego*. Trzecią postać normalną czasami opisuje się frazą „klucz, cały klucz i nic innego, tylko klucz".

Poruszanie się w obrębie danych

Baza danych `BIBLIOTECZKA` jest teraz w trzeciej postaci normalnej. Przykładową zawartość tabel pokazano na rysunku 4.10. Z łatwością można zauważyć relacje pomiędzy tabelami. Aby uzyskać informacje o poszczególnych autorach, korzystamy z kluczy. Klucz główny w każdej tabeli w sposób niepowtarzalny identyfikuje pojedynczy wiersz. Wybierzmy na przykład autora o nazwisku Stephen Jay Gould, a natychmiast znajdziemy odpowiadający mu zapis w tabeli `AUTOR`, ponieważ pole `NazwiskoAutora` jest kluczem głównym.

Odszukajmy autorkę *Harper Lee* w kolumnie `NazwiskoAutora` w tabeli `BIBLIOTECZKA_AUTOR`, a zobaczymy, że napisała ona powieść *Zabić drozda*. Następnie w tabeli `BIBLIOTECZKA` możemy sprawdzić wydawcę, kategorię i ocenę tej książki, a w tabeli `OCENA` — opis oceny.

Jeśli szukamy tytułu *Zabić drozda* w tabeli `BIBLIOTECZKA`, skorzystamy z klucza głównego w tabeli. Aby znaleźć autora książki, można odwrócić wcześniejszą ścieżkę wyszukiwania, poszukując w tabeli `BIBLIOTECZKA_AUTOR` rekordów, które w kolumnie `Tytul` mają szukaną wartość — kolumna `Tytul` jest kluczem obcym w tabeli `BIBLIOTECZKA_AUTOR`. Jeśli klucz główny tabeli `BIBLIOTECZKA` znajdzie się w innej tabeli tak, jak to ma miejsce w przypadku tabeli `BIBLIOTECZKA_AUTOR`, nazywa się go kluczem obcym tej tabeli.

Ponieważ dane zorganizowano w sposób logiczny, można przygotować kategorie i oceny, do których nie przypisano jeszcze żadnych książek. Jest to sensowny i logiczny sposób organizowania informacji nawet wtedy, gdy „tabele" są zapisane w książce magazynowej lub na fiszkach przechowywanych w pudełkach. Oczywiście ciągle czeka nas sporo pracy, aby dane zorganizowane w ten sposób przekształcić w prawdziwą bazę danych. Na przykład pole `NazwiskoAutora` można podzielić na `Imie` i `Nazwisko`. Przydałaby się też możliwość wyświetlenia informacji o tym, kto jest głównym autorem książki, a kto na przykład recenzentem.

Rysunek 4.10. *Przykładowe dane z bazy danych BIBLIOTECZKA*

AUTOR

NazwiskoAutora	Uwagi
DIETRICH BONHOEFFER	TEOLOG NIEMIECKI, ZABITY W OBOZIE JENIECKIM
ROBERT BRETALL	WYDAWCA PISM KIERKEGAARDA
HELENA BECHLEROWA	AUTORKA KSIĄŻEK DLA DZIECI
STEPHEN JAY GOULD	AUTOR ARTYKUŁÓW NAUKOWYCH, PROFESOR HARVARDU
SOREN KIERKEGAARD	FILOZOF I TEOLOG DUŃSKI
HARPER LEE	POWIEŚCIOPISARZ AMERYKAŃSKI, WYDAŁ TYLKO JEDNĄ POWIEŚĆ
LUCY MAUD MONTGOMERY	POWIEŚCIOPISARKA KANADYJSKA
JOHN ALLEN PAULOS	PROFESOR MATEMATYKI
J. RODALE	EKSPERT OGRODNICTWA

OCENA

Ocena	OpisOceny
1	ROZRYWKA
2	INFORMACJE PODSTAWOWE
3	POLECANE
4	GORĄCO POLECANE
5	TO TRZEBA PRZECZYTAĆ

KATEGORIA

NazwaKategorii	KategoriaGlowna	KategoriaPodrzedna
DOROŚLIKOMENT	DOROŚLI	KOMENTARZE
DOROŚLIFIKCJA	DOROŚLI	FIKCJA
DOROŚLIFAKTY	DOROŚLI	INNE NIŻ FIKCJA
DZIECIOBRAZKI	DZIECI	KSIĄŻKI Z OBRAZKAMI
DZIECIFIKCJA	DZIECI	FIKCJA
DZIECIPOPNAUK	DZIECI	POPULARNONAUKOWE

BIBLIOTECZKA_AUTOR

Tytul	NazwiskoAutora
ZABIĆ DROZDA	HARPER LEE
WSPANIAŁE ŻYCIE	STEPHEN JAY GOULD
ANALFABETYZM MATEMATYCZNY	JOHN ALLEN PAULOS
ANTOLOGIA KIERKEGAARDA	ROBERT BRETALL
ANTOLOGIA KIERKEGAARDA	SOREN KIERKEGAARD
ANIA Z ZIELONEGO WZGÓRZA	LUCY MAUD MONTGOMERRY
CZTERY MISIE I TEN PIĄTY	HELENA BECHLEROWA
LISTY I NOTATKI Z WIĘZIENIA	DIETRICH BONHOEFFER

BIBLIOTECZKA

Tytul	Wydawca	NazwaKategorii	Ocena
ZABIĆ DROZDA	KSIĄŻKA I WIEDZA	DOROŚLIFIKCJA	5
WSPANIAŁE ŻYCIE	W.W. NORTON & CO	DOROŚLIFAKTY	5
ANALFABETYZM MATEMATYCZNY	GDAŃSKIE WYDAWNICTWO OŚWIATOWE	DOROŚLIFAKTY	4
ANTOLOGIA KIERKEGAARDA	PRINCETON UNIV PR	DOROŚLIKOMENT	3
ANIA Z ZIELONEGO WZGÓRZA	NASZA KSIĘGARNIA	DZIECIFIKCJA	3
CZTERY MISIE I TEN PIĄTY	NASZA KSIĘGARNIA	DZIECIOBRAZKI	1
LISTY I NOTATKI Z WIĘZIENIA	SCRIBNER	DOROŚLIFAKTY	4

Cały ten proces nazywa się normalizacją. Naprawdę nie ma w tym nic specjalnego. Chociaż dobry projekt aplikacji bazodanowej uwzględnia kilka dodatkowych zagadnień, podstawowe zasady analizowania „normalnych" relacji pomiędzy różnymi danymi są tak proste, jak właśnie opisaliśmy.

Trzeba jednak pamiętać, że normalizacja jest częścią procesu analizy, a nie projektu. Uznanie, że znormalizowane tabele modelu logicznego są projektem rzeczywistej bazy danych to istotny błąd. Mylenie procesów analizy i projektowania jest podstawową przyczyną niepowodzeń poważnych aplikacji wykorzystujących relacyjne bazy danych,

o których później można przeczytać w prasie. Zagadnienia te — z którymi dokładnie powinni zapoznać się programiści — zostaną opisane bardziej szczegółowo w dalszej części rozdziału.

Opisowe nazwy tabel i kolumn

Po zidentyfikowaniu relacji zachodzących pomiędzy różnymi elementami w bazie danych i odpowiednim posegregowaniu danych, należy poświęcić sporo czasu na wybranie nazw dla tabel i kolumn, w których zostaną umieszczone dane. Zagadnieniu temu często poświęca się zbyt mało uwagi. Lekceważą je nawet ci, którzy powinni zdawać sobie sprawę z jego doniosłości. Nazwy tabel i kolumn często są wybierane bez konsultacji oraz odpowiedniej analizy. Nieodpowiedni dobór nazw tabel i kolumn ma poważne następstwa podczas korzystania z aplikacji.

Dla przykładu rozważmy tabele pokazane na rysunku 4.10. Nazwy mówią tu same za siebie. Nawet użytkownik, dla którego problematyka relacyjnych baz danych jest nowa, nie będzie miał trudności ze zrozumieniem zapytania następującej postaci[1]:

```
select Tytul, Wydawca
  from BIBLIOTECZKA
  order by Wydawca;
```

Użytkownicy rozumieją zapytanie, ponieważ wszystkie słowa brzmią znajomo. Nie są to jakieś tajemnicze zaklęcia. Kiedy trzeba zdefiniować tabele zawierające znacznie więcej kolumn, dobranie odpowiednich nazw jest trudniejsze. W takiej sytuacji wystarczy jednak konsekwentnie stosować kilka reguł. Przeanalizujmy kilka problemów, które często powstają w przypadku braku odpowiedniej konwencji nazw. Zastanówmy się, co by się stało, gdybyśmy zastosowali takie oto nazwy:

```
BIBLIOTECZKA   B_A       AUT           KATEGORIE
-------------- -----     ------------  ----------
tytul          tytul     naz_a         kat
wyd            naz_a     koment        p_kat
kat                                    s_kat
oc
```

Nazwy zastosowane w tej tabeli, mimo że dziwne, są niestety dość powszechne. Zostały dobrane zgodnie z konwencją (lub brakiem konwencji) wykorzystywaną przez znanych producentów oprogramowania i programistów. Poniżej wymieniono kilka bardziej znanych problemów występujących podczas dobierania nazw.

- *Stosowanie skrótów bez powodu*. W ten sposób zapamiętanie pisowni nazw staje się niemal niemożliwe. Nazwy mogłyby równie dobrze być kodami, ponieważ użytkownicy i tak muszą ich szukać.
- *Niespójne stosowanie skrótów*.

[1] Mowa tu oczywiście o użytkownikach znających podstawy języka angielskiego. Jednak nawet ci, którzy nie znają tego języka, zrozumieją zapytanie, jeśli poznają znaczenie kilku słów: *select* — wybierz, *from* — z, *where* — gdzie, *order by* — uporządkuj według — *przyp. tłum*.

- *Na podstawie nazwy nie można w sposób jednoznaczny określić przeznaczenia tabeli lub kolumny*. Skróty nie tylko powodują, że zapamiętanie pisowni nazw staje się trudne, ale również zaciemniają naturę danych zapisanych w kolumnie lub tabeli. Co to jest `p_kat` lub `koment`?
- *Niespójne stosowanie znaków podkreślenia*. Znaków podkreślenia czasami używa się do oddzielania słów w nazwie, ale innym razem nie używa się ich w ogóle. W jaki sposób zapamiętać gdzie wstawiono znaki podkreślenia, a gdzie nie?
- *Niespójne stosowanie liczby mnogiej*. `KATEGORIA` czy `KATEGORIE`? `Komentarz` czy `Komentarze`?
- *Stosowane reguły mają ograniczenia*. Jeśli nazwa kolumny rozpoczyna się od pierwszej litery nazwy tabeli (np. kolumna `Anaz` w tabeli, której nazwa rozpoczyna się na literę *A*), to co należy zrobić, kiedy innej tabeli trzeba nadać nazwę na literę *A*? Czy kolumnę w tej tabeli również nazwiemy `Anaz`? Jeśli tak, to dlaczego nie nadać obu kolumnom nazwy `Nazwisko`?

Użytkownicy, którzy nadają tabelom i kolumnom niewłaściwe nazwy, nie są w stanie w prosty sposób formułować zapytań. Zapytania nie mają intuicyjnego charakteru i znajomego wyglądu tak, jak w przypadku zapytania do tabeli `BIBLIOTECZKA`, co w znacznym stopniu zmniejsza użyteczność aplikacji.

Dawniej od programistów wymagano tworzenia nazw składających się maksymalnie z ośmiu znaków. W rezultacie nazwy były mylącym zlepkiem liter, cyfr i niewiele mówiących skrótów. Obecnie podobne ograniczenia, wymuszane przez starą technologię, nie mają już zastosowania. W systemie Oracle nazwy kolumn i tabel mogą mieć rozmiar do 30 znaków. Projektanci zyskali dzięki temu możliwość tworzenia nazw pełnych, jednoznacznych i opisowych.

Pamiętajmy zatem, aby w nazwach wystrzegać się skrótów, liczby mnogiej i nie stosować znaków podkreślenia (lub stosować je konsekwentnie). Unikniemy wówczas mylących nazw, które obecnie zdarzają się nader często. Jednocześnie konwencje nazw muszą być proste, zrozumiałe i łatwe do zapamiętania. W pewnym sensie potrzebna jest zatem normalizacja nazw. W podobny sposób, w jaki analizujemy dane, segregujemy je według przeznaczenia i w ten sposób normalizujemy, powinniśmy przeanalizować standardy nazewnictwa. Bez tego zadanie utworzenia aplikacji zostanie wykonane niewłaściwie.

Dane w języku naturalnym

Po przeanalizowaniu konwencji nazw dla tabel i kolumn, trzeba poświęcić uwagę samym danym. Przecież od czytelności danych będzie zależeć czytelność drukowanego raportu. W przykładzie z bazą danych `BIBLIOTECZKA`, wartości w kolumnie `Ocena` są wyrażone za pomocą kodu, a `Kategoria` to połączenie kilku wartości. Czy to dobrze? Jeśli zapytamy kogoś o książkę, czy chcielibyśmy usłyszeć, że uzyskała ona ocenę *4* w kategorii *Dorośli-Fakty*? Dlaczego mielibyśmy pozwalać maszynie, aby wyrażała się nie dość jasno?

Dzięki opisowym informacjom formułowanie zapytań staje się łatwiejsze. Zapytanie powinno w maksymalny sposób przypominać zdanie z języka naturalnego:

```
select Tutul, NazwiskoAutora
  from BIBLIOTECZKA_AUTOR;
```

Stosowanie wielkich liter w nazwach i danych

Nazwy tabel i kolumn są zapisywane w wewnętrznym słowniku danych systemu Oracle za pomocą wielkich liter. Gdy w zapytaniu nazwy wpiszemy małymi literami, system natychmiast zmieni ich wielkość, a następnie przeszuka słownik danych. W niektórych systemach relacyjnych baz danych wielkość liter ma znaczenie. Jeśli użytkownik wpisze nazwę kolumny jako `Zdolnosc`, a w bazie danych kolumna ta występuje jako `zdolnosc` lub `ZDOLNOSC` (w zależności od tego, w jaki sposób zdefiniowano kolumnę podczas tworzenia tabeli), system nie zrozumie zapytania.

Użytkownik może wymusić na systemie Oracle obsługę nazw o mieszanej wielkości liter, ale wówczas tworzenie zapytań i praca z danymi będą trudniejsze. Z tego powodu najlepiej wykorzystywać domyślną właściwość — stosowanie wielkich liter.

Rozróżnianie wielkości liter jest czasami promowane jako zaleta baz danych, dzięki której programiści mogą tworzyć różne tabele o podobnych nazwach — jak choćby `pracownik`, `Pracownik` i `pRAcownik`. Ale w jaki sposób zapamiętać różnice? Taka właściwość jest w istocie wadą, a nie zaletą. Firma Oracle była wystarczająco rozsądna, aby nie wpaść w tę pułapkę.

Podobnie rzecz się ma w przypadku danych zapisanych w bazie. Skoro istnieje możliwość interakcji z systemem Oracle w taki sposób, że wielkość liter w zapytaniach nie ma znaczenia, to istnieją również sposoby wyszukiwania danych, niezależnie od tego, czy zostały zapisane wielkimi, czy małymi literami. Ale po co wykonywać niepotrzebne działania? Poza kilkoma wyjątkami, takimi jak teksty prawnicze lub formularze, o wiele łatwiej zapisywać dane za pomocą wielkich liter i tworzyć aplikacje, w których wybrano spójną wielkość liter dla danych. Dzięki temu formułowanie zapytań staje się łatwiejsze, a dane otrzymują spójniejszy wygląd w raportach. Jeśli niektóre dane trzeba zapisać mieszanymi literami (np. nazwisko i adres na kopercie), można wywołać funkcję systemu Oracle, która dokona odpowiedniej konwersji.

Uważni czytelnicy dostrzegli zapewne, że autor książki nie przestrzegał dotychczas tej zasady. Jej stosowanie opóźniano do czasu odpowiedniego wprowadzenia i umieszczenia jej we właściwym kontekście. Od tej pory, poza kilkoma uzasadnionymi wyjątkami, dane w bazie danych będą zapisywane wielkimi literami.

Normalizacja nazw

Na rynku pojawiło się kilka narzędzi umożliwiających stosowanie w zapytaniach słów języka naturalnego zamiast dziwnych zestawień. Działanie tych produktów polega na utworzeniu logicznej mapy ze słów języka naturalnego, trudnych do zapamiętania kodów oraz nazw kolumn i tabel. Przygotowanie takiego odwzorowania wymaga szczegółowej analizy, ale po pomyślnym wykonaniu tego zadania interakcja z aplikacją staje się o wiele łatwiejsza. Jednak dlaczegóż by nie zwrócić uwagi na ten problem od początku? Po co tworzyć dodatkowy produkt i wykonywać dodatkową pracę, skoro można uniknąć większości zamieszania, od razu nadając odpowiednie nazwy?

Aby zapewnić odpowiednią wydajność aplikacji, czasami niektóre dane są zapisywane w bazie w postaci zakodowanej. Kody te nie powinny być ujawniane użytkownikom ani podczas wprowadzania danych, ani podczas ich pobierania. System Oracle umożliwia ich łatwe ukrywanie.

Jeśli przy wprowadzaniu danych trzeba zastosować kody, natychmiast wzrasta liczba błędów literowych. Jeśli kody występują w raportach zamiast powszechnie używanych słów, pojawiają się błędy interpretacji. A kiedy użytkownicy chcą utworzyć nowy raport, ich zdolność do szybkiego i dokładnego wykonania tej czynności jest znacznie ograniczona zarówno za sprawą kodów, jak i z powodu trudności w zapamiętaniu dziwnych nazw kolumn i tabel.

System Oracle daje użytkownikom możliwość posługiwania się językiem naturalnym w całej aplikacji. Ignorowanie tej sposobności jest marnotrawieniem możliwości systemu Oracle. Jeśli z niej nie skorzystamy, bezsprzecznie powstanie mniej zrozumiała i mało wydajna aplikacja. Programiści mają obowiązek skorzystania z okazji. Użytkownicy powinni się tego domagać. Obie grupy z całą pewnością na tym skorzystają.

Czynnik ludzki

Użytkownicy, którzy dopiero rozpoczynają przygodę z systemem Oracle, być może poczuli już ochotę, aby przejść do konkretnych działań. Jednak sposoby pracy z użyciem języka SQL zostaną opisane dopiero w następnym rozdziale. Teraz zajmiemy się inną problematyką: spróbujemy przeanalizować projekt programistyczny, w którym zamiast skupiać się na danych, wzięto pod uwagę rzeczywiste zadania biznesowe wykonywane przez użytkowników docelowych.

Technologie normalizacji danych oraz wspomagana komputerowo inżynieria oprogramowania (ang. *Computer Aided Software Engineering* — CASE) zyskały tak wielkie znaczenie podczas projektowania aplikacji relacyjnych, że koncentracja na danych, zagadnieniach referencyjnej integralności, kluczach i diagramach tabel stała się prawie obsesją. Zagadnienia te często są mylone z właściwym projektem. Przypomnienie, iż jest to tylko analiza, dla wielu stanowi spore zaskoczenie.

Normalizacja jest analizą, a nie projektowaniem. A właściwie jest to jedynie część analizy, niezbędna do zrozumienia zasad funkcjonowania danej firmy i stworzenia użytecznej aplikacji. Celem aplikacji jest przede wszystkim wspomaganie działań przedsiębiorstwa poprzez szybsze i wydajniejsze wykonywanie zadań biznesowych oraz usprawnienie środowiska pracy. Jeśli pracownikom damy kontrolę nad informacjami oraz zapewnimy do nich prosty i intuicyjny dostęp, odpowiedzą wzrostem wydajności. Jeśli kontrolę powierzymy obcej grupie, ukryjemy informacje za wrogimi interfejsami — pracownicy będą niezadowoleni, a przez to mniej wydajni.

Naszym celem jest zaprezentowanie filozofii działania, która prowadzi do powstania przyjaznych i wygodnych aplikacji. Do projektowania struktur danych oraz przepływu sterowania wystarczą narzędzia, które znamy i których używamy w codziennej pracy.

Zadania aplikacji i dane aplikacji

Twórcy oprogramowania często nie poświęcają należytej uwagi identyfikacji zadań, których wykonywanie chcą uprościć. Jedną z przyczyn tego stanu rzeczy jest wysoki poziom złożoności niektórych projektów, drugą — znacznie częściej występującą — skupienie się na danych.

Podczas analizy programiści najpierw pytają o rodzaj pobieranych danych oraz sposób ich przetwarzania i przedstawiania w raportach. Później zadają szereg pytań pomocniczych, poruszając takie zagadnienia, jak formularze do wprowadzania danych, kody, projekty ekranów, obliczenia, komunikaty, poprawki, raport audytu, objętość pamięci masowej, cykl przetwarzania danych, formatowanie raportów, dystrybucja i pielęgnacja. Są to bardzo ważne zagadnienia. Problem polega na tym, że wszystkie koncentrują się wyłącznie na danych.

Profesjonaliści korzystają z danych, ale wykonują zadania. Ich wiedza i umiejętności są należycie zagospodarowane. Z kolei szeregowi urzędnicy często jeszcze zajmują się jedynie wpisywaniem danych do formularza wejściowego. Zatrudnianie ludzi tylko po to, by wprowadzali dane, w szczególności dane o dużej objętości, spójne co do formatu (tak, jak w przypadku formularzy) oraz z ograniczoną różnorodnością, jest drogie, przestarzałe, a przede wszystkim antyhumanitarne. Czasy takiego myślenia już przeminęły, podobnie jak czasy kodów minimalizujących ograniczenia maszyn.

Pamiętajmy ponadto, że rzadko kiedy pracownik, rozpocząwszy realizację zadania, zajmuje się nim tak długo, aż je ukończy. Zwykle wykonuje różne dodatkowe czynności, mniej lub bardziej ważne, ale w jakiś sposób powiązane z zadaniem głównym. Bywa, że realizuje je równolegle — wszystkie naraz.

Wszystko to ma swoje konsekwencje praktyczne. Projektanci, którzy pracują w nowoczesny sposób, nie koncentrują się na danych tak, jak to było w przeszłości. Dla nich najważniejsze są zadania, które z pomocą aplikacji będzie wykonywać użytkownik. Dlaczego środowiska okienkowe odniosły taki sukces? Ponieważ pozwalają użytkownikowi na szybką zmianę realizowanego zadania, a kilka zadań może oczekiwać na swoją kolej w stanie aktywności. Takiej lekcji nie można zmarnować. Należy wyciągnąć z niej prawidłowe wnioski.

Identyfikacja zadań aplikacji to przedsięwzięcie znacznie ambitniejsze niż prosta identyfikacja i normalizacja danych lub zwykłe zaprojektowanie ekranów, programów przetwarzających i narzędzi raportujących. Oznacza ona zrozumienie rzeczywistych potrzeb użytkownika i w efekcie opracowanie aplikacji, która interpretuje zadania, a nie tylko pobiera skojarzone z nimi dane. Jeśli projektant skoncentruje się na danych, aplikacja zniekształci zadania użytkowników. Największym problemem jest więc uświadomienie sobie, że skoncentrowanie się na zadaniach jest koniecznością.

Rozpoczynając proces analizy, najpierw musimy określić, do jakich zadań użytkownicy docelowi wykorzystują komputery. Jaką usługę lub produkt chcą wytworzyć? Jest to podstawowe i może nawet nieco uproszczone pytanie, ale jak zaraz zobaczymy, zaskakująco wielu ludzi nie potrafi udzielić na nie przekonującej odpowiedzi. Przedstawiciele licznych branż, począwszy od ochrony zdrowia, a na bankach, firmach wysyłkowych i fabrykach skończywszy, uważają, że istotą ich pracy jest przetwarzanie danych. Przecież dane są wprowadzane do komputerów i przetwarzane, po czym tworzy się raporty — czyż nie?

Z powodu tej deformacji, którą należy uznać za jeszcze jeden przejaw orientacji na dane w projektowaniu systemów, mnóstwo firm podjęło próbę wprowadzenia na rynek wyimaginowanego produktu — przetwarzania danych — z katastrofalnymi, rzecz jasna, skutkami.

Dlatego tak ważna jest wiedza na temat przeznaczenia aplikacji. Aby projektant zrozumiał, czym naprawdę zajmuje się dana dziedzina, do czego służy wytworzony produkt i jakie zadania są wykonywane w procesie produkcji, musi mieć otwartą głowę i często intuicyjnie wyczuwać przedmiot biznesu. Równie ważne jest, aby użytkownicy aplikacji znali język SQL i orientowali się w założeniach modelu relacyjnego. Zespół składający się z projektantów, którzy rozumieją potrzeby użytkowników i doceniają wartość zorientowanego na zadania, czytelnego środowiska aplikacji, oraz z użytkowników mających odpowiednie przygotowanie techniczne (np. dzięki przeczytaniu niniejszej książki) z pewnością stworzy bardzo dobry system. Członkowie takiego zespołu wzajemnie sprawdzają się, mobilizują i pomagają w realizacji indywidualnych zadań.

Punktem wyjścia w pracach nad przygotowaniem profesjonalnej aplikacji będzie więc opracowanie dwóch zbieżnych z sobą dokumentów: jednego z opisem zadań, drugiego z opisem danych. Przygotowując opis zadań, uświadamiamy sobie sens aplikacji. Opis danych pomaga w implementacji wizji i zapewnia uwzględnienie wszystkich szczegółów i obowiązujących zasad.

Identyfikacja zadań

Dokument, który opisuje zadania związane z prowadzoną działalnością, powstaje w wyniku wspólnej pracy użytkowników docelowych i projektantów aplikacji. Rozpoczyna się od zwięzłego opisu tej działalności. Powinno to być jedno proste zdanie, składające się z nie więcej niż 10 słów, pisane w stronie czynnej, bez przecinków i z minimalną liczbą przymiotników, na przykład:

> Zajmujemy się sprzedażą ubezpieczeń.

A nie:

> Firma *Amalgamated Diversified* jest wiodącym międzynarodowym dostawcą produktów finansowych w dziedzinie ubezpieczeń zdrowotnych i komunikacyjnych, prowadząc w tym zakresie również szkolenia i świadcząc usługi doradcze.

Istnieje pokusa, aby w pierwszym zdaniu umieścić wszelkie szczegóły dotyczące prowadzonej działalności. Nie należy tego robić. Skrócenie rozwlekłego opisu do prostego zdania umożliwia skoncentrowanie się na temacie. Jeśli ktoś nie potrafi się w ten sposób streścić, oznacza to, że jeszcze nie zrozumiał istoty prowadzonej działalności.

Ułożenie tego zdania, które inicjuje proces tworzenia dokumentacji, jest wspólnym obowiązkiem projektantów i użytkowników. Współpracując, łatwiej znajdą odpowiedź na pytania, czym zajmuje się przedsiębiorstwo i w jaki sposób wykonywane są jego zadania. Jest to cenna wiedza dla firmy, gdyż w procesie poszukiwania odpowiedzi zidentyfikowanych zostaje wiele zadań podrzędnych, jak również procedur i reguł, które nie mają znaczenia lub są używane marginalnie. Zazwyczaj są to albo odpryski problemu, który już rozwiązano, albo postulaty menedżerów, które dawno zostały załatwione.

Przekorni twierdzą, że aby poradzić sobie z nadmiarem raportów, należy zaprzestać ich tworzenia i sprawdzić, czy ktokolwiek to zauważy. W tym przesadnym stwierdzeniu tkwi ziarno prawdy — przecież w podobny sposób rozwiązano problem roku dwutysięcznego w starszych komputerach! Okazało się, że wiele programów nie wymagało wprowadzania poprawek z tego prostego powodu, że zaprzestano ich używania!

Programista aplikacji, dokumentując zadania firmy, ma prawo zadawać dociekliwe pytania i wskazywać marginalne obszary jej działalności. Należy jednak zdawać sobie sprawę, że projektant nigdy nie będzie tak dobrze rozumiał zadań firmy, jak użytkownik docelowy. Istnieje różnica pomiędzy wykorzystaniem prac projektowych do zracjonalizowania wykonywanych zadań i zrozumienia ich znaczenia a obrażaniem użytkowników poprzez próbę wmówienia im, że znamy firmę lepiej niż oni.

Należy poprosić użytkownika o w miarę szczegółowe przedstawienie celów firmy, a także zadań, które zmierzają do ich realizacji. Jeśli cele są niezbyt dobrze umotywowane (np. „zawsze to robimy w ten sposób" lub „myślę, że wykorzystują to do czegoś"), powinna zapalić się nam czerwona lampka. Powinniśmy powiedzieć, że nie rozumiemy, i poprosić o ponowne wyjaśnienia. Jeśli odpowiedź w dalszym ciągu nie jest zadowalająca, należy umieścić takie zadanie na oddzielnej liście w celu późniejszej dokładnej analizy. Na niektóre z takich pytań odpowiedzi udzielą użytkownicy lepiej znający temat, z innymi będzie trzeba się zwrócić do kierownictwa, a wiele zadań wyeliminujemy, ponieważ okażą się niepotrzebne. Jednym z dowodów na dobrze przeprowadzony proces analizy jest usprawnienie istniejących procedur. Jest to niezależne od nowej aplikacji komputerowej i generalnie powinno nastąpić na długo przed jej zaimplementowaniem.

Ogólny schemat dokumentu opisującego zadania

Dokument opisujący zadania składa się z trzech sekcji:

- zdanie charakteryzujące prowadzoną działalność (trzy do dziesięciu słów),
- kilka krótkich zdań opisujących w prostych słowach główne zadania firmy,
- opis zadań szczegółowych.

Po każdej sekcji można umieścić krótki, opisowy akapit, jeśli jest taka potrzeba, co oczywiście nie usprawiedliwia lenistwa w dążeniu do jasności i zwięzłości. Główne zadania zazwyczaj numeruje się jako 1.0, 2.0, 3.0 itd. i opisuje jako *zadania poziomu zero*. Dla niższych poziomów wykorzystuje się dodatkowe kropki, na przykład 3.1 oraz 3.1.14. Każde zadanie główne rozpisuje się na szereg *zadań niepodzielnych* — takich, które albo trzeba wykonać do końca, nie przerywając ich realizacji, albo całkowicie anulować.

Wypisanie czeku jest zadaniem niepodzielnym, ale wpisanie samej kwoty już nie. Odebranie telefonu w imieniu działu obsługi klienta nie jest zadaniem niepodzielnym. Odebranie telefonu i spełnienie żądań klienta jest zadaniem niepodzielnym. Zadania niepodzielne muszą mieć znaczenie, a ich wykonanie musi przynosić jakieś rezultaty.

Poziom, na którym zadanie staje się niepodzielne, zależy od treści zadania głównego. Zadanie oznaczone jako 3.1.14 może być niepodzielne, ale równie dobrze może zawierać kilka dodatkowych poziomów podrzędnych. Niepodzielne może być zadanie 3.2, ale także 3.1.16.4. Ważny nie jest sam schemat numerowania (który jest po prostu metodą opisu hierarchii zadań), ale dekompozycja zadań do poziomu niepodzielnego.

Dwa zadania w dalszym ciągu mogą być niepodzielne, nawet jeśli okaże się, że jedno zależy od drugiego, ale tylko wtedy, kiedy jedno może zakończyć się niezależnie od drugiego. Jeśli dwa zadania zawsze zależą od siebie, nie są to zadania niepodzielne. Prawdziwe zadanie niepodzielne obejmuje oba te zadania.

W przypadku większości przedsięwzięć szybko się zorientujemy, że wiele zadań podrzędnych można przyporządkować do dwóch lub większej liczby zadań z poziomu zero, co niemal zawsze dowodzi, że niewłaściwie zdefiniowano zadania główne lub dokonano nieodpowiedniego ich podziału. Celem naszych działań jest przekształcenie każdego zadania w pojęciowy „obiekt", z dobrze zdefiniowanymi zadaniami i jasno określonymi zasobami niezbędnymi do osiągnięcia celu.

Wnioski wynikające z dokumentu opisującego zadania

Wniosków jest kilka. Po pierwsze, ponieważ dokument ten jest bardziej zorientowany na zadania niż na dane, istnieje prawdopodobieństwo, że pod wpływem zawartych w nim zapisów zmieni się sposób projektowania ekranów użytkownika. Dokument ma także wpływ na zasady pobierania i prezentacji danych oraz na implementację systemu pomocy. Gwarantuje, że przełączanie pomiędzy zadaniami nie będzie wymagało od użytkownika zbytniego wysiłku.

Po drugie, w miarę odkrywania konfliktów pomiędzy zadaniami podrzędnymi może zmienić się opis zadań głównych. To z kolei wpływa na sposób rozumienia zadań przedsiębiorstwa zarówno przez użytkowników, jak i projektantów aplikacji.

Po trzecie, nawet akapity kończące sekcje prawdopodobnie ulegną zmianie. Racjonalizacja, polegająca w tym przypadku na definiowaniu niepodzielnych zadań i budowaniu wspomnianych przed chwilą pojęciowych „obiektów", wymusza usunięcie niedomówień i zależności, które niepotrzebnie wywierają wpływ na działalność firmy.

Tworzenie tego dokumentu nie jest procesem bezproblemowym — trzeba szukać odpowiedzi na niewygodne pytania, ponownie definiować niewłaściwe założenia, żmudnie korygować błędy. Ostatecznie jednak jego opracowanie przynosi duże korzyści. Lepiej rozumiemy sens działalności firmy, potrafimy określić obowiązujące procedury, możemy zaplanować automatyzację zadań.

Identyfikacja danych

Definiując każde zadanie, określamy zasoby potrzebne do jego realizacji, w tym przede wszystkim dane. Wymagania w zakresie danych uwzględniamy w dokumencie opisu danych. Nie chodzi tu o opis pól w formularzach i zawartych w nich elementów. Chodzi o rejestr koniecznych danych. Ponieważ o tym, które dane są konieczne, decyduje treść zadania, a nie istniejące formularze, niezbędna staje się dokładna analiza tej treści oraz określenie rzeczywistych wymagań w zakresie danych. Jeśli osoba wykonująca zadanie nie zna zastosowania pola, w które wprowadza dane, takie pole należy umieścić na liście problemów wymagających dokładniejszej analizy. Pracując w ten sposób, usuwamy mnóstwo niepotrzebnych danych.

Po zidentyfikowaniu danych należy je uważnie przeanalizować. Szczególną uwagę powinniśmy zwrócić na kody numeryczne i literowe, które ukrywają rzeczywiste informacje za barierą nieintuicyjnych, niewiele mówiących symboli. Ponieważ zawsze budzą podejrzenia, należy do nich podchodzić z dystansem. W każdym przypadku trzeba zadać sobie pytanie, czy określony element powinien być kodem, czy nie. Czasami użycie kodów jest poręczne — choćby dlatego, że ułatwia zapamiętywanie. Dla ich stosowania można znaleźć również inne racjonalne argumenty i sensowne powody. Jednak w gotowym projekcie takie przypadki nie mogą zdarzać się często, a znaczenie kodów powinno być oczywiste. W przeciwnym razie łatwo się pogubimy.

Proces konwersji kodów na wartości opisowe to zadanie dość proste, ale stanowi dodatkową pracę. W dokumencie opisującym dane kody należy podać wraz z ich znaczeniem uzgodnionym i zatwierdzonym wspólnie przez użytkowników i projektantów.

Projektanci i użytkownicy powinni również wybrać nazwy grup danych. Nazwy te zostaną użyte jako nazwy kolumn w bazie danych i będą regularnie stosowane w zapytaniach, a zatem powinny to być nazwy opisowe (w miarę możliwości bez skrótów, poza powszechnie stosowanymi) i wyrażone w liczbie pojedynczej. Ze względu na bliski związek nazwy kolumny z zapisanymi w niej danymi, oba te elementy należy określić jednocześnie. Przemyślane nazwy kolumn znacznie upraszczają identyfikację charakteru danych.

Dane, które nie są kodami, także muszą zostać dokładnie przeanalizowane. W tym momencie możemy przyjąć, że wszystkie zidentyfikowane elementy danych są niezbędne do wykonania zadań biznesowych, choć niekoniecznie są one dobrze zorganizowane. To, co wygląda na jeden element danych w istniejącym zadaniu, może w istocie być kilkoma elementami, które wymagają rozdzielenia. Dane osobowe, adresy, numery telefonów to popularne przykłady takich danych.

Na przykład w tabeli `AUTOR` imiona były połączone z nazwiskami. Kolumna `NazwiskoAutora` zawierała zarówno imiona, jak i nazwiska, pomimo że tabela została doprowadzona do trzeciej postaci normalnej. Byłby to niezwykle nieudolny sposób zaimplementowania aplikacji, choć z technicznego punktu widzenia reguły normalizacji zostały zachowane. Aby aplikacja pozwalała na formułowanie prostych zapytań, kolumnę `NazwiskoAutora` należy rozdzielić na co najmniej dwie nowe kolumny: `Imie` i `Nazwisko`. Proces wydzielania nowych kategorii, który prowadzi do poprawy czytelności i większej funkcjonalności bazy danych, bardzo często jest niezależny od normalizacji.

Stopień dekompozycji zależy od przewidywanego sposobu wykorzystania danych. Istnieje możliwość, że posuniemy się za daleko i wprowadzimy kategorie, które choć składają się z oddzielnych elementów, w rzeczywistości nie tworzą dodatkowej wartości w systemie. Sposób wykonywania dekompozycji zależy od aplikacji i rodzaju danych. Nowym grupom danych, które staną się kolumnami, należy dobrać odpowiednie nazwy. Trzeba uważnie przeanalizować dane, które zostaną w tych kolumnach zapisane. Nazwy należy także przypisać danym tekstowym, dla których można wyróżnić skończoną liczbę wartości. Nazwy tych kolumn, ich wartości oraz same kody można uznać za tymczasowe.

Modele danych niepodzielnych

Od tego momentu rozpoczyna się proces normalizacji, a razem z nim rysowanie modeli danych niepodzielnych. Istnieje wiele dobrych artykułów na ten temat oraz mnóstwo narzędzi analitycznych i projektowych, które pozwalają przyspieszyć proces normalizacji, a zatem w tej książce nie proponujemy jednej metody. Taka propozycja mogłaby raczej zaszkodzić, niż pomóc.

W modelu należy uwzględnić każdą niepodzielną transakcję i oznaczyć numerem zadanie, którego ta transakcja dotyczy. Należy uwzględnić nazwy tabel, klucze główne i obce oraz najważniejsze kolumny. Każdej znormalizowanej relacji należy nadać opisową nazwę oraz oszacować liczbę wierszy i transakcji. Każdemu modelowi towarzyszy dodatkowy arkusz, w którym zapisano nazwy kolumn z typami danych, zakresem wartości oraz tymczasowymi nazwami tabel, kolumn oraz danych tekstowych w kolumnach.

Model biznesowy

Dokument opisujący dane jest teraz połączony z dokumentem opisującym zadania, tworząc model biznesowy. Projektanci aplikacji i użytkownicy muszą wspólnie go przeanalizować w celu sprawdzenia jego dokładności i kompletności. W tym momencie powinni posiadać już jasną wizję przedsięwzięcia, realizowanych w nim zadań i używanych danych.

Po wprowadzeniu poprawek oraz zatwierdzeniu modelu biznesowego rozpoczyna się proces syntezy zadań i danych. W tej fazie następuje sortowanie danych, przydzielanie ich do zadań, ostateczne zakończenie normalizacji oraz przypisanie nazw wszystkim elementom, które ich wymagają.

W przypadku rozbudowanych aplikacji zwykle powstaje dość duży rysunek. Dołącza się do niego dokumentację pomocniczą uwzględniającą zadania, modele danych (z poprawionymi nazwami utworzonymi na podstawie kompletnego modelu), listę tabel, nazw kolumn, typów danych i zawartości. Jedną z ostatnich czynności jest prześledzenie ścieżek dostępu do danych dla każdej transakcji w pełnym modelu biznesowym w celu sprawdzenia, czy wszystkie dane wymagane przez transakcje można pobierać lub wprowadzać oraz czy nie przetrwały zadania wprowadzania danych z brakującymi elementami, co mogłoby uniemożliwić zachowanie integralności referencyjnej modelu.

Z wyjątkiem nadawania nazw poszczególnym tabelom, kolumnom i danym, wszystkie czynności aż do tego momentu były elementami analizy, a nie fazami projektowania. Celem tego etapu było właściwe zrozumienie biznesu i jego komponentów.

Wprowadzanie danych

Model biznesowy koncentruje się raczej na zadaniach, a nie na tabelach danych. Ekrany aplikacji projektowanej w oparciu o model biznesowy wspomagają tę orientację, umożliwiając sprawną obsługę zadań, a w razie potrzeby przełączanie się między nimi. Ekrany

te wyglądają i działają nieco inaczej niż ekrany typowe, odwołujące się do tabel i występujące w wielu aplikacjach, ale przyczyniają się do wzrostu skuteczności użytkowników oraz podnoszą jakość ich pracy.

W praktyce oznacza to możliwość formułowania zapytań o wartości umieszczone w tabeli głównej dla danego zadania oraz w innych tabelach, aktualizowanych w momencie, kiedy jest wykonywany dostęp do tabeli głównej. Bywa jednak i tak, że gdy nie określono tabeli głównej, dane można pobrać z kilku powiązanych z sobą tabel. Wszystkie one zwracają (a w razie potrzeby przyjmują) dane w czasie wykonywania zadania.

Interakcja pomiędzy użytkownikiem a komputerem ma znaczenie kluczowe. Ekrany do wprowadzania danych i tworzenia zapytań powinny być zorientowane na zadania i pisane w języku naturalnym. Ważną rolę odgrywają także ikony i interfejs graficzny. Ekrany muszą być zorganizowane w taki sposób, aby komunikacja odbywała się językiem, do jakiego są przyzwyczajeni użytkownicy.

Zapytania i tworzenie raportów

Jedną z podstawowych cech, odróżniających aplikacje relacyjne i język SQL od aplikacji tradycyjnych, jest możliwość łatwego formułowania indywidualnych zapytań do obiektów bazy danych oraz tworzenia własnych raportów. Pozwala na to narzędzie SQL*Plus. Zapytania wpisywane z wiersza poleceń i otrzymywane tą drogą raporty nie wchodzą w skład podstawowego zestawu opracowanego przez programistę w kodzie aplikacji. Dzięki temu użytkownicy i programiści systemu Oracle zyskują niezwykłą kontrolę nad danymi. Użytkownicy mogą analizować informacje, modyfikować zapytania i wykonywać je w ciągu kilku minut oraz tworzyć raporty. Programiści są zwolnieni z niemiłego obowiązku tworzenia nowych raportów.

Użytkownicy mają możliwość wglądu w dane, przeprowadzenia szczegółowej ich analizy i reagowania z szybkością i dokładnością, jakie jeszcze kilka lat temu były nieosiągalne. Wydajność aplikacji znacznie wzrasta, jeśli nazwy tabel, kolumn i wartości danych są opisowe, spada natomiast, jeśli w projekcie przyjęto złą konwencję nazw oraz użyto kodów i skrótów. Czas poświęcony na spójne, opisowe nazwanie obiektów w fazie projektowania opłaci się. Z pewnością użytkownicy szybko odczują korzyści z tego powodu.

Niektórzy projektanci, szczególnie ci niemający doświadczenia w tworzeniu dużych aplikacji z wykorzystaniem relacyjnych baz danych, obawiają się, że kiepsko przygotowani użytkownicy będą pisać nieefektywne zapytania zużywające olbrzymią liczbę cykli procesora, co spowoduje spowolnienie pracy komputera. Doświadczenie pokazuje, że z reguły nie jest to prawda. Użytkownicy szybko się uczą, jakie zapytania są obsługiwane sprawnie, a jakie nie. Co więcej, większość dostępnych dziś narzędzi, służących do wyszukiwania danych i tworzenia raportów, pozwala oszacować ilość czasu, jaką zajmie wykonanie zapytania, oraz ograniczyć dostęp (o określonej porze dnia lub gdy na komputerze zaloguje się określony użytkownik) do tych zapytań, które spowodowałyby zużycie nieproporcjonalnie dużej ilości zasobów. W praktyce żądania, które użytkownicy wpisują do wiersza poleceń, tylko czasami wymykają się spod kontroli, ale korzyści, jakie z nich wynikają, znacznie przekraczają koszty przetwarzania. Niemal za każdym razem, kiedy scedujemy na komputery zadania wykonywane dotychczas przez ludzi, osiągamy oszczędności finansowe.

Rzeczywistym celem projektowania jest sprecyzowanie i spełnienie wymagań użytkowników biznesowych. Zawsze należy starać się, aby aplikacja była jak najłatwiejsza do zrozumienia i wykorzystania, nawet kosztem zużycia procesora lub miejsca na dysku. Nie można jednak dopuścić do tego, aby wewnętrzna złożoność aplikacji była tak duża, że jej pielęgnacja i modyfikacja staną się trudne i czasochłonne.

Normalizacja nazw obiektów

Najlepsze nazwy to nazwy przyjazne w użytkowaniu: opisowe, czytelne, łatwe do zapamiętania, objaśnienia i zastosowania, bez skrótów i kodów. Jeśli decydujemy się w nazwach na znaki podkreślenia, stosujmy je w sposób spójny i przemyślany. W dużych aplikacjach nazwy tabel, kolumn i danych często składają się z wielu słów, na przykład `OdwroconeKontoZawieszone` lub `Data_Ostatniego_Zamkniecia_KG`. Na kilku następnych stronach zostanie zaprezentowane nieco bardziej rygorystyczne podejście do nazewnictwa, którego celem jest opracowanie formalnego procesu normalizacji nazw obiektów.

Integralność poziom-nazwa

W systemie relacyjnej bazy danych obiekty takie, jak bazy danych, właściciele tabel, tabele, kolumny i wartości danych, są uporządkowane hierarchicznie. W przypadku bardzo dużych systemów różne bazy danych mogą zostać rozmieszczone w różnych lokalizacjach. Dla potrzeb zwięzłości, wyższe poziomy zostaną na razie zignorowane, ale to, co tutaj powiemy, ich także dotyczy.

Każdy poziom w hierarchii jest zdefiniowany w ramach poziomu znajdującego się powyżej. Obiektom należy przypisywać nazwy odpowiednie dla danego poziomu i unikać stosowania nazw z innych poziomów. Na przykład w tabeli nie może być dwóch kolumn o nazwie `Nazwisko`, a konto o nazwie `Jerzy` nie może zawierać dwóch tabel o nazwie `AUTOR`.

Nie ma obowiązku, aby wszystkie tabele konta `Jerzy` miały nazwy niepowtarzalne w całej bazie danych. Inni właściciele także mogą posiadać tabele `AUTOR`. Nawet jeśli `Jerzy` ma do nich dostęp, nie ma możliwości pomyłki, ponieważ tabelę można zidentyfikować w sposób niepowtarzalny, poprzedzając nazwę tabeli prefiksem w postaci imienia jej właściciela, na przykład `Dariusz.AUTOR`. Nie byłoby spójności logicznej, gdyby częścią nazw wszystkich tabel należących do Jerzego było jego imię, jak na przykład `JERZYAUTOR`, `JERZYBIBLIOTECZKA` itd. Stosowanie takich nazw jest mylące, a poza tym umieszczenie w nazwie części nazwy nadrzędnej niepotrzebnie komplikuje nazewnictwo tabel i w efekcie narusza integralność poziom-nazwa.

Zwięzłości nigdy nie należy przedkładać nad czytelność. Włączanie fragmentów nazw tabel do nazw kolumn jest złą techniką, gdyż narusza logiczną strukturę poziomów oraz wymaganą integralność poziom-nazwa. Jest to także mylące, ponieważ wymaga od użytkowników wyszukiwania nazw kolumn niemal za każdym razem, kiedy piszą zapytania. Nazwy obiektów muszą być niepowtarzalne w obrębie poziomu nadrzędnego, ale używanie nazw spoza własnego poziomu obiektu nie powinno być dozwolone.

Obsługa abstrakcyjnych typów danych w systemie Oracle wzmacnia możliwości tworzenia spójnych nazw atrybutów. Na przykład typ danych o nazwie ADRES_TY będzie charakteryzował się takimi samymi atrybutami za każdym razem, kiedy zostanie użyty. Każdy atrybut ma zdefiniowaną nazwę, typ danych i rozmiar, co powoduje, że implementacja aplikacji w całym przedsiębiorstwie stanie się spójniejsza. Jednak wykorzystanie abstrakcyjnych typów danych w ten sposób wymaga:

- właściwego zdefiniowania typów danych na początku procesu projektowania, aby uniknąć konieczności późniejszego ich modyfikowania,
- spełnienia wymagań składniowych obowiązujących dla abstrakcyjnych typów danych.

Klucze obce

Jedną z przeszkód stojących na drodze do stosowania zwięzłych nazw kolumn jest występowanie obcych kluczy w tabeli. Czasem się zdarza, że kolumna w tabeli, w której występuje klucz obcy, ma tę samą nazwę, co kolumna klucza obcego w swojej tabeli macierzystej. Nie byłoby problemu, gdyby można było użyć pełnej nazwy klucza obcego wraz z nazwą tabeli macierzystej (np. BIBLIOTECZKA.Tytul).

Aby rozwiązać problem z takimi samymi nazwami, trzeba wykonać jedno z następujących działań:

- opracować nazwę, która obejmuje nazwę tabeli źródłowej klucza obcego, bez wykorzystywania kropki (np. z użyciem znaku podkreślenia),
- opracować nazwę, która obejmuje skrót nazwy tabeli źródłowej klucza obcego,
- opracować nazwę, która różni się od nazwy w tabeli źródłowej,
- zmienić nazwę kolumny powodującej konflikt.

Żadne z tych działań nie jest szczególnie atrakcyjne, ale jeśli zetkniemy się z tego typu dylematem dotyczącym nazwy, należy wybrać jedno z nich.

Nazwy w liczbie pojedynczej

Obszarem niespójności powodującym sporo zamieszania jest liczba gramatyczna nazw nadawanych obiektom. Czy tabela powinna nosić nazwę AUTOR, czy AUTORZY? Kolumna ma mieć nazwę Nazwisko czy Nazwiska?

Najpierw przeanalizujmy kolumny, które występują niemal w każdej bazie danych: Nazwisko, Adres, Miasto, Wojewodztwo i KodPocztowy. Czy — nie licząc pierwszej kolumny — kiedykolwiek zauważyliśmy, aby ktoś używał tych nazw w liczbie mnogiej? Jest niemal oczywiste, że nazwy te opisują zawartość pojedynczego wiersza — rekordu. Pomimo że relacyjne bazy danych przetwarzają zbiory, podstawową jednostką zbioru jest wiersz, a zawartość wiersza dobrze opisują nazwy kolumn w liczbie pojedynczej. Czy formularz do wprowadzania danych osobowych powinien mieć taką postać, jak poniżej?

```
Nazwiska: _____________________________________________________
Adresy: _____________________________________________________
Miasta: _____________________ Województwa:_____ KodyPocztowe:__-___
```

Czy też nazwy wyświetlane na ekranie powinny mieć liczbę pojedynczą, ponieważ w określonym czasie pobieramy jedno nazwisko i jeden adres, ale podczas pisania zapytań trzeba poinformować użytkowników, aby przekształcili te nazwy na liczbę mnogą? Konsekwentne stosowanie nazw kolumn w liczbie pojedynczej jest po prostu bardziej intuicyjne.

Jeśli nazwy wszystkich obiektów mają tę samą liczbę, ani programista, ani użytkownik nie muszą zapamiętywać dodatkowych reguł. Korzyści są oczywiste. Załóżmy, że zdecydowaliśmy o tym, że wszystkim obiektom nadamy nazwy w liczbie mnogiej. W tym przypadku pojawią się różne końcówki w nazwie niemal każdego obiektu. Z kolei w nazwie wielowyrazowej poszczególne słowa otrzymają różne końcówki. Jakie korzyści mielibyśmy osiągnąć z ciągłego wpisywania tych dodatkowych liter? Czyżby tak było łatwiej? Czy takie nazwy są bardziej zrozumiałe? Czy takie nazwy łatwiej zapamiętać? Oczywiście nie.

Z tego powodu najlepiej stosować następującą zasadę: dla wszystkich nazw obiektów zawsze należy używać liczby pojedynczej. Wyjątkami mogą być terminy, które są powszechnie stosowane w biznesie, takie jak na przykład `Aktywa`.

Zwięzłość

Jak wspomniano wcześniej, zwięzłości nigdy nie należy przedkładać nad czytelność, ale w przypadku dwóch jednakowo treściwych, równie łatwych do zapamiętania i opisowych nazw zawsze należy wybrać krótszą. W czasie projektowania aplikacji warto zaproponować alternatywne nazwy kolumn i tabel grupie użytkowników i programistów i wykorzystać ich rady w celu wybrania tych propozycji, które są czytelniejsze. W jaki sposób tworzyć listę alternatyw? Można skorzystać z tezaurusa i słownika. W zespole projektowym, zajmującym się tworzeniem różnych aplikacji, każdego członka zespołu należy wyposażyć w tezaurus i słownik, a następnie przypominać im o konieczności uważnego nadawania nazw obiektom.

Obiekt o nazwie tezaurus

Relacyjne bazy danych powinny zawierać obiekt o nazwie tezaurus, podobnie jak zawierają słownik danych. Tezaurus wymusza korzystanie z firmowych standardów nazewniczych i zapewnia spójne stosowanie nazw i skrótów (jeśli są używane). Takie standardy czasami wymagają używania (również spójnego i konsekwentnego!) znaków podkreślenia, co ułatwia identyfikację poszczególnych elementów złożonej nazwy.

W agencjach rządowych lub dużych firmach wprowadzono standardy nazewnictwa obiektów. Standardy obowiązujące w dużych organizacjach w ciągu lat przeniknęły do pozostałej części komercyjnego rynku i może się zdarzyć, że tworzą podstawę standardów nazewnictwa naszej firmy. Mogą one na przykład zawierać wskazówki dotyczące stosowania terminów *Korporacja* lub *Firma*. Jeśli nie zaadaptowaliśmy takich standardów nazewnictwa, w celu zachowania spójności należy opracować własne normy, które uwzględniają zarówno standardy bazowe, jak i wskazówki nakreślone w tym rozdziale.

Inteligentne klucze i wartości kolumn

Nazwa *inteligentne klucze* jest niezwykle myląca, ponieważ sugeruje, że mamy do czynienia z czymś pozytywnym lub godnym uwagi. Trafniejszym określeniem mógłby być termin *klucze przeciążone*. Do tej kategorii często zalicza się kody Księgi Głównej oraz kody produktów (dotyczą ich wszystkie trudności charakterystyczne dla innych kodów). Trudności typowe dla kluczy przeciążonych dotyczą także kolumn niekluczowych, które zawierają więcej niż jeden element danych.

Typowy przykład przeciążonego klucza opisano w następującym fragmencie: „Pierwszy znak jest kodem regionu. Kolejne cztery znaki są numerem katalogowym. Ostatnia cyfra to kod centrum kosztów, o ile nie mamy do czynienia z częścią importowaną — w takiej sytuacji na końcu liczby umieszcza się znacznik /. Jeśli nie jest to element występujący w dużej ilości, wtedy dla numeru katalogowego są wykorzystywane tylko trzy cyfry, a kod regionu oznacza się symbolem HD".

W dobrym projekcie relacyjnym wyeliminowanie przeciążonych kluczy i wartości kolumn ma zasadnicze znaczenie. Zachowanie przeciążonej struktury stwarza zagrożenie dla relacji tworzonych na jej podstawie (zazwyczaj dotyczy to obcych kluczy w innych tabelach). Niestety w wielu firmach przeciążone klucze są wykorzystywane od lat i na trwałe wrosły w jej zadania. Niektóre zostały wdrożone we wcześniejszych etapach automatyzacji, kiedy używano baz danych nieobsługujących kluczy wielokolumnowych. Inne mają podłoże historyczne — stosowano je po to, aby krótkiemu kodowi nadać szersze znaczenie i objąć nim większą liczbę przypadków, niż pierwotnie planowano. Z wyeliminowaniem przeciążonych kluczy mogą być trudności natury praktycznej, które uniemożliwiają natychmiastowe wykonanie tego zadania. Dlatego tworzenie nowej aplikacji relacyjnej staje się wówczas trudniejsze.

Rozwiązaniem problemu jest utworzenie nowego zestawu kluczy — zarówno głównych, jak i obcych, które w prawidłowy sposób normalizują dane, a następnie upewnienie się, że dostęp do tabel jest możliwy wyłącznie za pomocą tych nowych kluczy. Przeciążone klucze są wówczas utrzymywane jako dodatkowe, niepowtarzalne kolumny tabeli. Dostęp do danych za pomocą historycznych metod jest zachowany (np. poprzez wyszukanie przeciążonego klucza w zapytaniu), ale jednocześnie promuje się klucze o poprawnej strukturze jako preferowaną metodę dostępu. Z czasem, przy odpowiednim szkoleniu, użytkownicy przekonają się do nowych kluczy. Na koniec przeciążone klucze (lub inne przeciążone wartości kolumn) można po prostu zamienić na wartości `NULL` lub usunąć z tabeli.

Jeśli nie uda się wyeliminować przeciążonych kluczy i wartości z bazy danych, kontrola poprawności danych, zapewnienie integralności danych oraz modyfikowanie struktury staną się bardzo trudne i kosztowne.

Przykazania

Omówiliśmy wszystkie najważniejsze zagadnienia wydajnego projektowania. Warto teraz je podsumować w jednym miejscu, stąd tytuł *Przykazania* (choć może lepszy byłby tytuł *Wskazówki*). Znając te zalecenia, użytkownik może dokonać racjonalnej oceny projektu i skorzystać z doświadczeń innych osób rozwiązujących podobne problemy.

Celem tego podrozdziału nie jest opisanie cyklu tworzenia oprogramowania, który prawdopodobnie wszyscy dobrze znają, ale raczej udowodnienie twierdzenia, że projektowanie z odpowiednią orientacją powoduje radykalną zmianę wyglądu i sposobu korzystania z aplikacji. Uważne postępowanie zgodnie z podanymi wskazówkami pozwala na znaczącą poprawę wydajności i poziomu zadowolenia użytkowników aplikacji.

Oto dziesięć przykazań właściwego projektu:

1. Nie zapominajmy o użytkownikach. Włączajmy ich do zespołu projektowego i uczmy modelu relacyjnego i języka SQL.
2. Nazwy tabel, kolumn, kluczy i danych nadawajmy wspólnie z użytkownikami. Opracujmy tezaurus aplikacji w celu zapewnienia spójności nazw.
3. Stosujmy opisowe nazwy w liczbie pojedynczej, które mają znaczenie, są łatwe do zapamiętania i krótkie. Wykorzystujmy znaki podkreślenia konsekwentnie lub wcale.
4. Nie mieszajmy poziomów nazw.
5. Unikajmy kodów i skrótów.
6. Wszędzie tam, gdzie to możliwe, używajmy kluczy mających znaczenie.
7. Przeprowadźmy dekompozycję kluczy przeciążonych.
8. Podczas analizy i projektowania miejmy na uwadze zadania, a nie tylko dane. Pamiętajmy, że normalizacja nie jest częścią projektu.
9. Jak najwięcej zadań zlecajmy komputerom. Opłaca się poświęcić cykle procesora i miejsce w pamięci, aby zyskać łatwość użytkowania.
10. Nie ulegajmy pokusie szybkiego projektowania. Poświęćmy odpowiednią ilość czasu na analizę, projekt, testowanie i dostrajanie.

Ten rozdział celowo poprzedza rozdziały opisujące polecenia i funkcje — jeśli projekt jest zły, aplikacja również będzie działała źle, niezależnie od tego, jakich poleceń użyjemy. Funkcjonalność, wydajność, możliwości odtwarzania, bezpieczeństwo oraz dostępność trzeba odpowiednio zaplanować. Dobry plan to gwarancja sukcesu.