ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1999
Seria: INFORMATYKA z. 36 Nr kol. 1414

Stanisław WOŁEK
Politechnika Rzeszowska, Zakład Informatyki

PROJEKTOWANIE SIECIOWYCH BAZ DANYCH W ŚRODOWISKU *DELPHI*

Streszczenie. W artykule przedstawiono podstawowe cechy środowiska programowania Delphi. Przedstawiono narzędzia przeznaczone do tworzenia oprogramowania obsługi baz danych, dotyczące przeglądania i modyfikacji danych oraz wyszukiwania informacji za pomocą zapytań w języku SQL. Szczególną uwagę zwrócono na operacje obsługi transakcji realizowanych z komputerów terminalnych w komputerze serwerze sieciowej rozproszonej bazy danych.

DESIGNING DISTRIBUTED DATABASES WITHIN DELPHI ENVIRONMENT

Summary. The properties of the *Delphi* programming environment are described. The engines for creating the database manipulation programming, data manipulations and information searches with *SQL* are considered. A particular emphasis has been put on the transaction servicing being processed on the distributed database server through PC-terminals.

1. Wprowadzenie

W projektowaniu systemów baz danych można wyróżnić dwa rodzaje zagadnień, których realizacja może być niezależna zarówno od w odniesieniu do zastosowanego systemu zarządzania bazą danych (SZBD), używanego środowiska programistycznego, jak i systemu operacyjnego, z którym wymienione elementy współpracują.

Jednym zagadnieniem jest projektowanie interfejsu użytkownika systemu bazy danych, obsługującego bazę. Obsługa baz danych dotyczy ogólnie ich przeglądania (manipulowania da-

nymi), wprowadzania lub modyfikowania danych, przeszukiwania danych oraz generowania raportów.

Drugim zagadnieniem jest bezpośrednia realizacja dostępu do danych przechowywanych w bazie. Dane zapisywane są fizycznie w pliku lub plikach, obsługiwanych przez system zarządzania bazą danych. System umożliwia realizację prostych operacji przeglądania danych, ich dodawania lub zmieniania oraz bardziej skomplikowane operacje, dotyczące wyszukiwania danych na podstawie zadanych pytań oraz zarządzania danymi, np. w zakresie ich spójności, poprawności oraz ochrony dostępu do nich.

System *Delphi* firmy *Borland* daje możliwość projektowania systemów baz danych z graficznym interfejsem obsługi danych oraz z możliwością współpracy systemu baz danych z danymi obsługiwanymi przez różne systemy zarządzania bazami danych. *Delphi* jest środowiskiem programowania, pochodnym środowiska języka *Pascal*, przeznaczonym do tworzenia programów-aplikacji współpracujących z systemem operacyjnym *Windows* oraz obsługiwanych przez użytkownika za pośrednictwem typowego dla *Windows* interfejsu graficznego.

Delphi stanowi narzędzie typu **CASE** (*Computer Aided Software Engineering*). W skład systemu Delphi wchodzi:

- mechanizm szybkiego projektowania aplikacji (**RAD** - *Rapid Application Development*), służący do tworzenia aplikacji za pomocą wygodnego interfejsu oraz wielu narzędzi, które ułatwiają automatyzację tworzenia aplikacji, takich jak biblioteka komponentów-obiektów graficznych oraz procedury (metody) obsługi typowych zdarzeń generowanych dla tych komponentów,
- zintegrowane środowisko tworzenia aplikacji (**IDE** – *Integrated Development Environment*), które zawiera takie składowe jak pasek komponentów interfejsu graficznego, kontroler obiektów-komponentów (*Object Inspector*), biblioteka procedur obsługi zdarzeń, edytor procedur, kompilator, debugger,
- język **ObjectPascal**, zawierający oprócz wszystkich elementów języka *Pascal* możliwości definiowania i stosowania obiektów (elementów typu klasa) oraz poszerzony zakres instrukcji, między innymi do wykrywania i obsługi sytuacji wyjątkowych (np. błędów wykonania programu).

Delphi wraz z językiem *ObjectPascal* stanowi system **programowania obiektowego**, w którym można przede wszystkim interaktywnie projektować graficzne elementy interfejsu użytkownika (np. formularz, okienka, przyciski) oraz korzystać z bogatej biblioteki zawierającej opis typowych obiektów wraz z metodami obsługi zdarzeń ich dotyczących. Można też definiować własne obiekty typu klasa.

Programy tworzone w środowisku Delphi przyjmują postać przystosowaną do wykonywania **sterowanego zdarzeniami** (ang. *event driven*). Główna metoda programu wynikowego, o nazwie *Application.Run*, służy głównie do:

- wykrywania zdarzeń, generowanych przez użytkownika zewnętrznego (np. za pomocą klawiatury, myszki) lub oznaczających wewnętrzne sytuacje wyjątkowe,
- aktywizowania procedur obsługi tych zdarzeń, oznaczających standardową (systemową) ich obsługę, działanie wynikające z procedur napisanych przez projektanta w języku *ObjectPascal*.

Program utworzony w środowisku *Delphi* jest więc zbiorem niezależnych procedur prezentacji obiektów graficznego interfejsu użytkownika oraz procedur obsługi zdarzeń generowanych dla składowych tych obiektów. Aplikacja wynikowa przyjmuje postać kompletnego programu binarnego (wykonywalnego), zapisanego w pliku typu *.exe*.

2. Narzędzia systemu *Delphi* do obsługi baz danych

Delphi zawiera zintegrowany zespół narzędzi przeznaczonych do obsługi baz danych. Definiowanie połączenia z dowolnym systemem zarządzania bazą danych a projektowaną aplikacją jest procesem dwuetapowym.

W pierwszym etapie korzysta się z **ODBC** (*Open DataBase Connectivity*), stanowiącego metody pośrednictwa i wymiany danych z różnymi systemami zarządzania bazami danych.

Przy definiowaniu nowej bazy ODBC określa się:

- USDN (*User Data Sources Name*) będący pseudonimem bazy danych (zwanym też aliasem, alternatywną, zastępczą lub roboczą nazwą), która może być przyporządkowywana różnym bazom fizycznym,
- aktualne przyporządkowanie bazy fizycznej, polegające głównie na podaniu jej typu, określanego nazwą sterownika obsługi bazy, sprecyzowaniu położenia pliku (-ów) bazy, jej składu itp.

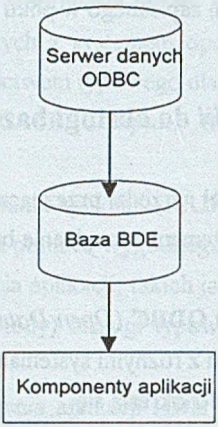
W dalszym ciągu projektowania używa się już tylko umownej nazwy bazy danych.

W drugim etapie definiowania połączenia z jednym z systemów zarządzania bazami danych, określonych pseudonimem w pierwszym etapie, realizuje się poprzez narzędzie **BDE** (*Borland DataBase Engine*) zwane też **IDAPI** (*Integrated DataBase Application Programming Interface*). Za pomocą BDE definiuje się źródła danych dla projektowanej aplikacji. Źródłu danych BDE także nadaje się pseudonim (alias) oraz określa typ bazy danych, jako standardowy dla *Delphi* (tj. bazę *Paradox*) lub jako inną bazę określaną pseudonimem ODBC. Poprawne skonfigurowanie BDE pozwala na dostęp z pojedynczej aplikacji do danych wcho-

dzących w skład różnych systemów zarządzania bazami danych, np. na przemian *Paradox*, *Acces* czy *Oracle*. W ramach jednego komputera można skonfigurować jeden zestaw ODBC oraz wiele zestawów BDE.

Stosowanie pseudonimów (rys. 1) pozwala nie tylko na skrócenie zapisu rzeczywistego położenia plików bazy danych, ale także na:

- uniezależnienie i odseparowanie treści procedur aplikacyjnych od faktycznej nazwy baz danych,
- ułatwienie przejścia między obsługą lokalnej bazy danych a bazą dostępną w sieci w trybie klient/serwer,
- umożliwienie prostej zmiany obsługiwanej bazy danych, przez zmianę przyporządkowania pseudonimu BDE pseudonimowi ODBC.



Rys. 1. Ilustracja powiązań różnych poziomów definiowania baz danych
Fig. 1. Illustration of interdepentencies of different levels of database definition

3. Projektowanie interfejsu systemu baz danych

Do obsługi baz danych z poziomu aplikacji tworzonych w środowisku Delphi służą komponenty, które można ustawiać w oknach służących do realizacji dialogu użytkownika aplikacji z programem. Komponenty te zebrane są w dwa zestawy:

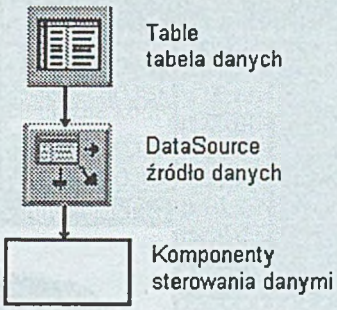
- **Data Access** – zawierający niewidzialne w oknie komponenty, służące do uzyskiwania dostępu do bazy lub zadawania pytań określających warunki wyszukiwania danych,
- **Data Controls** - zawierający komponenty widzialne, służące do sterowania danymi, ich wyświetlania lub edycji.

Najważniejsze komponenty do uzyskiwania dostępu do bazy (rys. 2) to:

- Tabela (**Table**) – o trzech głównych właściwościach:
  - DataBaseName – *pseudonim (nazwa zastępcza) bazy danych,*
  - TableName – *nazwa tablicy w bazie danych,*
  - Active – *stan otwarcia lub nieaktywności tabeli.*

  Poprzez kontroler obiektów w polu o nazwie *DataBase Name* udostępniana jest lista wszystkich pseudonimów zdefiniowanych w ODBC. Dla komponentu *Table* można rozwijać menu podręczne, w którym najważniejszą operację stanowi edytor pól, umożliwiający określanie tych pól tabeli danych, które będą dostępne poprzez komponent *Table*. Dodawanie lub usuwanie pól z listy pól dostępnych realizuje się operacjami wywoływanymi z kolejnego menu podręcznego.

  Do repertuaru pól związanych z komponentem *Table*, a więc dostępnych także dla pozostałych komponentów zależnych od danych, można dodawać pola wirtualne (ang. *calculated field*). Mogą one zawierać informacje, które nie są zapamiętywane w tabeli, lecz są na bieżąco wyznaczane na podstawie zawartości innych pól. Sposób wyznaczania wartości pól wirtualnych zapisywany jest jako procedura związana ze zdarzeniem *onCalcFields*, dotyczącym komponentu *Table*.
- Źródło danych (**DataSource**) – łącznik między tabelą a komponentami sterowania danymi, którego własność *DataSet* oznacza tabelę dostarczającą dane.



Rys. 2. Ilustracja połączeń komponentów obsługi baz danych
Fig. 2. Illustration of connections for database management

Do najważniejszych komponentów-obiektów zależnych od danych (ang. *data-aware control*), a związanych ze sterowaniem, wyświetlaniem lub ich edycją, należą:

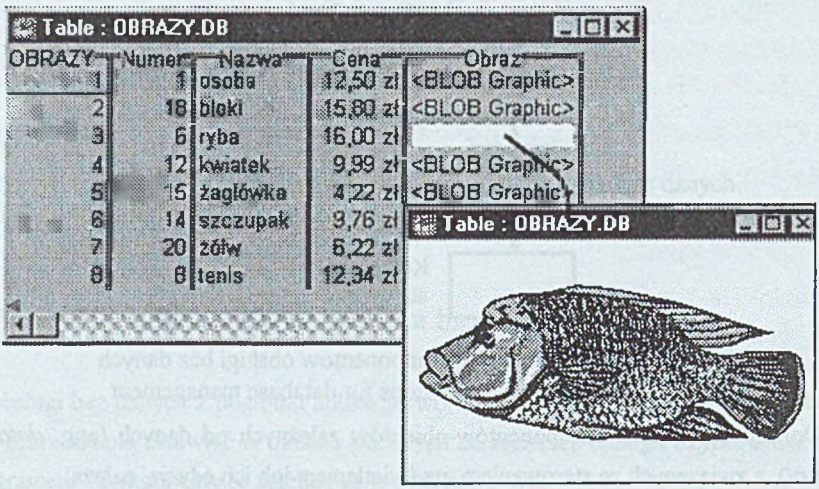
- Siatka z danymi (**DBGrid**) – służąca do prezentacji i edycji danych, przedstawianych w układzie kolumnowym.

  Można dobierać rozmiar komponentu i jego położenie na formularzu. Dostępne jest menu podręczne, w którym najważniejszą operację stanowi edytor kolumn, umożliwiający określanie tych kolumn tabeli danych, które będą prezentowane w obiekcie. Można zmieniać szerokość kolumn z danymi.

- Nawigator (**DBNavigator**) – obiekt przeznaczony do sterowania poruszaniem się w tabeli danych (rekordach i polach) oraz do wskazywania rekordu lub pola aktywnego, dla których można inicjować takie operacje, jak:
  - edycja istniejących danych,
  - wstawienie nowego rekordu,
  - usunięcie istniejącego rekordu.
- Pole wartości (**DBText**) – jest przeznaczone do wyświetlenia wartości pola (z bieżącego rekordu), wskazanego przez cechę *DataField* komponentu,
- Pole edycji (**DBEdit**) – służy do wyświetlenia oraz edycji wartości pola (z bieżącego rekordu), wskazanego przez cechę *DataField* komponentu.

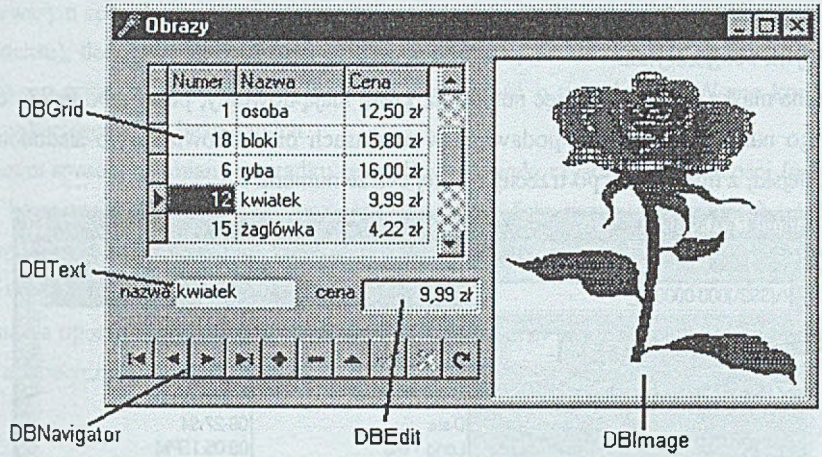
  Za pomocą tego komponentu można projektować wizualizację danych jednego rekordu w dowolnym układzie graficznym.
- Pole grafiki (**DBImage**) – jest przeznaczone do wyświetlenia wartości pola typu graficznego (z bieżącego rekordu), wskazanego przez cechę *DataField* komponentu.

Ilustracja komponentów przedstawiona jest dla przykładowej aplikacji współpracującej z bazą danych typu *Paradox*. Została zdefiniowana tabela o nazwie *obrazy.db*, której struktura i zawartość pokazane są na rys. 4. Ostatnia kolumna zawiera informacje typu graficznego, które można oglądać w osobnym oknie, po przejściu do podglądu zawartości pola.



Rys. 3. Struktura i zawartość przykładowej tabeli bazy danych
Fig. 3. Structure and contents of a database table

W formularzu dialogu dla przykładowej aplikacji (rys. 4) wykorzystano pięć podstawowych komponentów związanych z danymi przykładowej tabeli.



Rys. 4. Okno dialogu z przykładową aplikacją
Fig. 4. Form for interface with example application

Istotnym problemem obsługi wprowadzania nowych lub edycji danych istniejących w bazie jest sprawdzanie poprawności danych w zakresie ich formy i wartości. Można korzystać z dwóch typów zabezpieczeń:

- na poziomie aplikacji, w ramach dialogu realizowanego w Delphi,
- na poziomie systemu zarządzania bazą danych.

Najprostszą techniką wymuszania poprawnej postaci danych wejściowych jest stosowanie maski wprowadzania, zarówno w odniesieniu do dowolnego pola rekordu, jak i osobnego pola edycji jednowierszowej. W masce można kodować ograniczenia dotyczące każdego wprowadzanego znaku, w przykładowym zakresie:

- dopuszczalna tylko obligatoryjna cyfra,
- tylko opcjonalna cyfra,
- tylko cyfra, znak plus lub minus,
- tylko litera, obligatoryjna lub opcjonalna,
- litera lub cyfra.

W ramach maski można także wstawiać znaki, które automatycznie pojawiają się w polu jako stała ich zawartość, nie wymagająca ich pisania Do ustawiania maski w *Delphi* służy osobne okno dialogowe (rys. 5).

Klasycznym przykładem maski jest jej treść odpowiadająca poprawnej postaci numeru telefonicznego, przyjmująca w *Delphi* postać:

\(999\)000-00-00;1;_

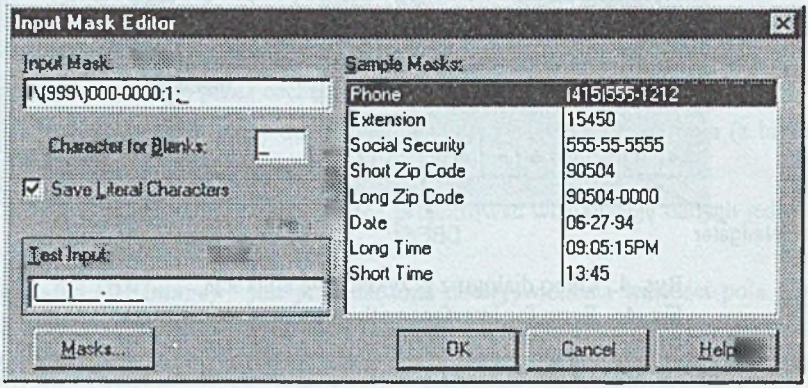
w której:

\(, \) oraz - – stałe znaki numeru telefonicznego,

9 – opcjonalna cyfra,

0 – cyfra obligatoryjna.

Podana maska odpowiada więc numerom, które mają dowolny, pusty albo jedno- do trzycyfrowego numer kierunkowy, podawany w nawiasach oraz obowiązkowo siedmiocyfrowy numer miejski, z myślnikiem po trzeciej i piątej cyfrze numeru.



Rys. 5. Okno ustawiania maski danych wejściowych
Fig. 5. Dialogue box of input mask editor

Nieprawidłowe znaki w sensie ich znaczenia nie są wprowadzane, natomiast zakończenie wprowadzania z brakiem znaków obligatoryjnych powoduje wygenerowanie zgłoszenia wyjątku (błędu), którego programowa obsługa może polegać na pokazaniu komunikatu o odpowiedniej treści i przejściu do ponownego wprowadzania danej.

Na poziomie systemu zarządzania bazą danych dostępne są wewnętrzne mechanizmy określania poprawnych wartości pól, indywidualne dla różnych systemów. Najczęściej odnoszą się one do:

– obowiązku wypełnienia pola (np. stanowiącego klucz główny bazy),

– typu wartości (numeryczna całkowita lub rzeczywista, tekstowa, data, czas itp.),

– wartości maksymalnej i minimalnej,

– domyślnej wartości początkowej.

Delphi daje możliwość programowego sprawdzania poprawności nowego lub modyfikowanego rekordu i automatycznego kasowania rekordu nowego lub wycofania się ze zmian istniejącego rekordu, w przypadku gdy wprowadzone dane nie są poprawne. Do tego celu służy operacja (metoda) o nazwie *Cancel*.

Dane prezentowane w oknie dialogu użytkownika (na przykład w polach typu *DBGrid*, *DBEdit* lub *DBText*) mogą być pokazywane w różnej kolejności, czyli z różnym uporządkowaniem rekordów tabeli.

Pierwszym sposobem jest porządkowanie rekordów według dowolnego istniejącego klucza (indeksu), definiowanego na poziomie systemu zarządzania bazą danych. Określenia klucza nowego porządku dokonuje się poprzez właściwość o nazwie *IndexName* komponentu *Table*, wskazującego na bazę danych.

Drugim sposobem zmiany porządku rekordów jest wykorzystanie właściwości *IndexFieldNames* komponentu *Table*, której wartością może być lista nazw pól (kolumn) tabeli, traktowanych jako klucz główny i klucze dodatkowe uporządkowania. Przykładowa instrukcja:

```
Obrazy.indexFieldNames := 'cena, nazwa'
```

Oznacza uporządkowanie rekordów według ceny, a przy jednakowych cenach według porządku alfabetycznego nazw obrazów.

4. Wyszukiwanie danych

W realizacji wyszukiwania danych uczestniczy osobny komponent, typu *DataAccess*. Umożliwia on dostęp do bazy przez zadanie pytań określających warunki wyszukiwania danych.

Najważniejsze komponenty do realizacji zapytań do bazy to:

- zapytanie (**Query**) – oznaczające tabelę stanowiącą wynik wyszukiwania, a odpowiadające obiektowi *Table* w bezpośrednim dostępie do danych.

  Główne właściwości obiektu *Query* to:

  - *DataBaseName* – nazwa (pseudonim) bazy, w której wykonuje się wyszukiwanie,
  - *SQL* – treść zapytania, podawana w języku SQL,
  - *Filtered* – ustawienie, czy ma być realizowane filtrowanie,
  - *Filter* – określenie filtru przez podanie warunku selekcji rekordów.

Na rys. 6 przedstawione jest połączenie obiektu typu *Query*, poprzez komponent *Datasource* z obiektem typu siatka danych. Dla obiektu *Query* ustawiona jest następująca postać zapytania SQL:

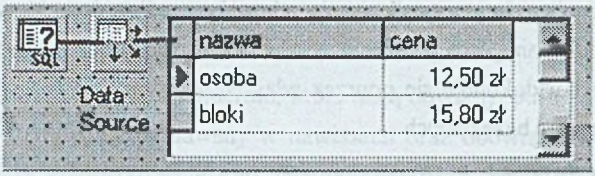
```
select nazwa, cena from "obrazy.db" where numer <10
```

oznaczającego wyświetlanie wyłącznie wartości kolumn *nazwa* i *cena*, z tabeli *obrazy.db*, z selekcją rekordów spełniających warunek *numer <10*.

Własność *filter* obiektu została zapisana następująco:

```
cena >= 12
```

oznaczając dodatkowe filtrowanie rekordów spełniających podany warunek.



Rys. 6. Schemat realizacji operacji wyszukiwania danych
Fig. 6. Schema of data searches operation

5. Rozproszony system baz danych w Delphi

W systemie *Delphi* można projektować aplikacje wielowęzłowych (ang. *multi-tiered*) rozproszonych baz danych typu klient/serwer. W systemach takich dane są dzielone na jednostki logiczne, z których każde są w połączeniu ze sobą obsługiwane przez osobne komputery, komunikujące się między sobą w sieci lokalnej lub przez Internet. Architektura tego typu systemu rozproszonego składa się z trzech typów składowych:

- aplikacji użytkownika końcowego, obsługującej jego dialog z systemem, realizowany w komputerze terminalu,
- serwera aplikacji, działającego w serwerze sieci lokalnej i realizującego dostęp wielu aplikacji użytkowników końcowych do wspólnych rozproszonych danych,
- zdalnego serwera baz danych, realizującego funkcje systemu zarządzania relacyjnymi bazami danych.

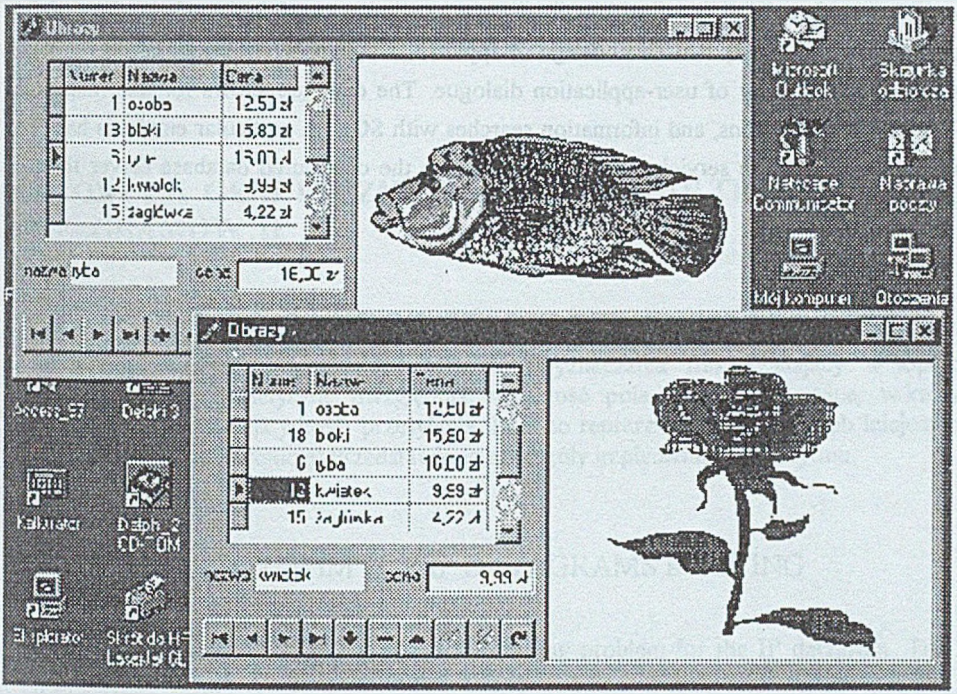
W tym trójskładnikowym modelu systemu serwer aplikacji zarządza wymianą danych między użytkownikami i zdalnym serwerem baz danych.

W realizacji aplikacji użytkownika końcowego korzysta się z komponentu typu *TClientDataSet*, pośredniczącego w dostępie do rozproszonych danych. Komponent typu *TRemoteServer* określa zdalny system zarządzania bazą danych.

Dostępne są mechanizmy obsługi transakcji. Procedura o nazwie *StartTransaction* służy do rozpoczęcia w serwerze baz danych nowej transakcji. Przed rozpoczęciem nowej transakcji należy sprawdzić stan dostępu do bazy. Gdyby serwer obsługiwał już transakcję (właściwość *InTransaction*), przed rozpoczęciem nowej należy poprzednią zakończyć (operacja *Commit*) lub wycofać się z niej (operacja *Rollback*). Gdyby tak się nie stało, wszystkie dalsze operacje uaktualniania, wstawiania lub usuwania danych będą wstrzymane aż do zakończenia poprzedniej transakcji.

Przy realizacji wielodostępu do baz danych i działania korzysta się z wbudowanych mechanizmów ochrony dostępu do danych, np. obowiązujące jako domyślne blokowanie dostępu do rekordu będącego w trakcie modyfikowania przez innego użytkownika.

Na rys. 7. przedstawiony jest ekran komputera, w którym równocześnie uaktywniono dwa procesy wykonywania tej samej aplikacji przykładowej. Obie aplikacje mogę pracować niezależnie, do momentu, gdy spróbuje się przejść do stanu edycji tego samego rekordu. Próba uaktywnienia drugiej edycji spowoduje jej zablokowanie, sygnalizowane odpowiednim komunikatem.



Rys. 7. Ilustracja działania aplikacji przykładowej w trybie wieloużytkownikowym
Fig. 7. Illustration of running a multi-user mode of application

LITERATURA

1. Osier D., Grobman S., Batson S.: Teach Yourself DELPHI 3 in 14 days. Sams Publishing, Indianapolis 1997.
2. Osier D., Grobman S., Batson S.: Delphi 3. Helion, Gliwice 1997.

Recenzent: Dr inż. Katarzyna Stąpor

Wpłynęło do Redakcji 9 kwietnia 1999 r.

Abstract

The basic properties of the Delphi programming environment are described. The engines for creating the database manipulation programming are considered in particular. Among them ODBC (open database connectivity) enabling communications with any DBMS through a common interface, BDE (Borland DataBase Engine) allowing to define configurations of data sources. The last facility is used in design of application user-program graphical interface, establishing a possibility of user-application dialogue. The dialogue allows for data manipulations, report generations, and information searches with SQL. A particular emphasis has been put on the transaction servicing being processed on the distributed database server through PC-terminals.