Jacek Matulewski
http://www.phys.uni.torun.pl/~jacek/

# Tworzenie prostych aplikacji obsługi lokalnych baz danych w środowisku Borland Delphi/C++ Builder

# Ćwiczenia

Toruń, 23 października 2002

Najnowsza wersja tego dokumentu znajduje się pod adresem
http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad2.pdf

Źródła opisanych w tym dokumencie programów znajdują się pod adresem
http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad2.zip

# I. Spis treści

# II. Aplikacje bazodanowe (lokalne)

Borland Delphi oraz C++ Builder posiadają zbiór komponentów służących do programowej obsługi baz danych. Dzięki temu możliwe jest łączenie lokalne i zdalne z bazami bazami danych różnego typu (od Paradox do Access i Excel), a następnie edycja ich struktury i zawartych w nich danych.

Szczególnym ułatwieniem jest fakt, że formy aplikacji bazodanowych można w aplikacjach Borlanda tworzyć za pomocą kreatora (Menu główne: Database\Form Wizard...). Jednak, żeby dobrze zrozumieć mechanizm działania tego typu aplikacji pierwszy przykład zaprojektujemy „ręcznie".

**Uwaga!** Jeżeli w menu głównym Delphi 3 nie ma pozycji Database, a wśród palet komponentów nie ma Data Access i Data Controls to znaczy, że Delphi zostało zainstalowane w okrojonej postaci lub, co bardziej prawdopodobne trzeba „podłączyć" odpowiednie komponenty (Components\Install Packages..., Design Packages, Delphi Database Components) z pliku Delphi 3\BIN\dcldb30.dpl. Możliwe jest wreszcie, że zainstalowana wersja (zapewne Standard o numerze większym niż 3) nie obejmuje obsługi baz danych.

## 1. Tworzenie aplikacji bazodanowej

Dane w bazach danych zgrupowane są w tabelach. Słowo tabela nie jest tu użyte na oznaczenie graficznej siatki kolumn i wierszy, ale jako coś bardziej abstrakcyjnego – zbioru danych. To z pozoru nieistotne rozróżnienie między tabelą a jej reprezentacją graficzną okaże się istotne w trakcie projektowania aplikacji, ponieważ inny komponent będzie reprezentował tabelę z bazy danych, a inny tabelę umieszczoną w formie.

Można sobie wyobrażać, że komponent TTable z palety Data Access jest reprezentantem wybranej tabeli bazy danych w naszej aplikacji. Do obiektu tej klasy (np. Table1) inne obiekty/komponenty mogą odnosić się za pośrednictwem komponentu TDataSource, będącym łączem między zbiorami danych a komponentami służącymi do wyświetlania i modyfikowania zawartości tabel (np. TDBGrid, który jest graficznym obrazem całej tabeli na naszej formie lub DBEdit – typowe okienko edycyjne).

**Dostęp do bazy danych:**
Aby nasza aplikacja mogła czytać dane np. z tabeli ANIMALS.DBF zawartej w bazie danych DBDEMOS[1] (dostarczanej z C++ Builder i Delphi) należy więc na czystej formie

1) **umieścić komponent TTable** (paleta Data Access) i **ustawić jego własność DatabaseName** na DBDEMOS (tj. wybrać bazę danych DBDEMOS, z której chcemy czytać dane) oraz **ustawić TableName** na ANIMALS.DBF (w tej kolejności, gdyż nazwy tabeli można wybrać z rozwijalnego menu dopiero po wskazaniu bazy danych). W ten sposób obiekt TTable1 reprezentuję tabelę ANIMALS.DBF z bazy danych DBDEMOS i zapewnia odpowiednie powiązanie z plikiem zawierającym potrzebne nam informacje oraz umożliwi ich poprawne czytanie.
2) Table1, nawet jeżeli została prawidłowo skonfigurowana, nie czyta danych z pliku dopóki nie zostanie wywołana **metoda Table1.Open** lub równoważne temu polecenie ustawienia własności **Table1.Active** na True (to ostatnie można zrobić za pomocą inspektora obiektów w trakcie projektowania aplikacji).
3) **umieścić na formie komponent TDataSource** (paleta Data Access) służący do przekazywania danych bazodanowym komponentom wizualnym. Większość komponentów z palety Data Controls odnosi się do tego komponentu. Rzadko kiedy programista odnosi się bezpośrednio do TTable. Aby połączyć DataSource1 z naszą tabelą należy **ustawić własność DataSet** na Table1.

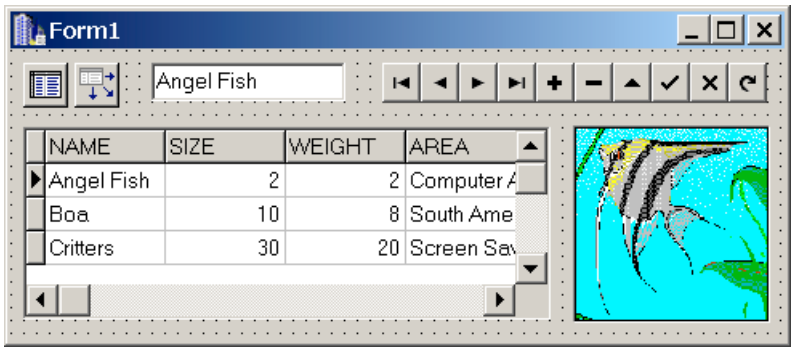
Te trzy kroki ustalają połączenie naszej aplikacji z bazą danych i powodują jej odczytanie. Kolejnym zadaniem jest prezentacja danych i umożliwienie ich modyfikacji[2].

**Kontrola danych:**

4) Przejdźmy do palety Data Controls i **umieśćmy na formie komponenty TDBGrid, TDBNavigator, TDBImage i TDBEdit**. Zaznaczmy wszystkie (klikając na kolejne komponenty na formie przy wciśniętym klawiszu Ctrl) i **ustawmy ich własność DataSource** na DataSource1.
   Jeżeli już w trakcie projektowania Table1 została aktywowana, to w chwili przypisania własności DataSource komponent TDBGrid powinien pokazać dane znajdujące się w wybranej przez nas tabeli bazy danych.

[1] Zbiór przykładowych tabel DBDEMOS może być również występować pod nazwą BCDEMOS.
[2] Użytkownik systemów Windows NT/2000/XP może nie mieć uprawnień do modyfikacji bazy danych. Wówczas możliwość ich edycji jest zablokowana.

5) Komponent TDBGrid prezentuje dane z wszystkich pól (kolumn) tabeli. Jednak większość komponentów (np. TDBListBox lub TDBComboBox) umie odnosić się do jednego pola lub jednej komórki tabeli (np. okno edycyjne TDBEdit lub obraz TDBImage). W tej sytuacji należy **ustalić także własność DataField** we wszystkich komponentach umieszczonych na formie poza TDBGrid i TDBNavigator (niech to będzie BMP w DBImage1 i AREA w DBEdit1).
Komponenty obsługujące jedno pole tabeli (np. TDBImage lub TDBEdit) pokazują wartość danego pola w aktywnym rekordzie. Aktywny rekord jest zaznaczony w DBGrid za pomocą czarnego trójkąta na początku wiersza. Domyślny rekord (wiersz) po aktywowaniu tabeli to rekord pierwszy.




**Działanie aplikacji:**

6) Po uruchomieniu możemy oglądać i edytować dane za pomocą DBGrid1, a pole AREA za pomocą DBEdit1 (zgodnie z przypisaną wartością DataField). Zmiany zostaną zapisane do pliku bazy danych (przy domyślnych ustawieniach) gdy zmieni się aktywny rekord lub połączenie jest zamykane. Użytkownik może zmieniać aktywny rekord przez wybranie odpowiedniego wiersza w DBGrid1 lub korzystając z przycisków DBNavigator1. Można zapobiec modyfikacjom tabeli ustawiając atrybut **DBGrid1.ReadOnly:=True** i **DBEdit.ReadOnly:=True**. Przyciski nawigatora kolejno to skok do pierwszego rekordu, skok do poprzedniego, skok do następnego, skok do ostatniego, wstaw rekord, usuń rekord, edycja rekordu, potwierdź zmiany wprowadzone w trakcie edycji, anuluj zmiany, odśwież dane.

Na zakończenie kilka dodatkowych informacji:

- Add 1) Nie musimy importować całej tabeli. Aby wybrać pola (kolumny), które chcemy udostępnić aplikacji używamy **Fields Editora** - klikamy dwukrotnie na Table1 i z kontekstowego menu (prawy przycisk myszy) wybieramy pozycję Add Fields
- Add 4) Nie wszystkie udostępnione pola (kolumny) muszą być pokazane w DBGrid1. Aby wybrać kolumny, które nas interesują (wskazane jest pominięcie pola BMP) należy z menu kontekstowego wybrać **Columns Editor** (lub dwukrotnie kliknąć myszką), kliknąć przycisk Add All Fields i skasować pole BMP. W tym miejscu można również zmienić kolejność kolumn.
- TDBImage, podobnie jak TImage, posiada własność Stretch, która powoduje dopasowanie wyświetlanego obrazka do wielkości komponentu.

Zachowajmy projekt w osobnym katalogu.

**Zadanie:**
Zmienić czytaną tabelę na BIOLIFE.DB. Pociągnie to za sobą konieczność zmiany DataField w niektórych komponentach.

## 2. Database Form Wizard

Rozumiejąc mechanizm dostępu aplikacji do bazy danych możemy posługiwać się kreatorem form (menu Database\Form Wizard). Ponownie skorzystamy z przykładowej bazy danych DBDEMOS.
Należy uruchomić kreatora.

1) W pierwszym oknie wybieramy *Create a simple form* oraz dostęp oparty na komponencie TTable. O pozostałych możliwościach będzie w następnych paragrafach.
2) Następnie określamy bazę danych (Drive or Alias Name: DBDEMOS) i tabelę (Table Name: COUNTRY.DB)

3) Teraz wybieramy pola udostępniane przez TTable (odpowiednik Field Editora w projektowaniu „ręcznym"). Wybierzmy wszystkie pola.
4) Na następnej stronie możemy wybrać jeden z trzech schematów prezentacji danych na formie (prezentacja pojedynczego rekordu w pionie i w poziomie oraz cała tabela). Wybierzmy *Vertically* (pozioma)
5) Etykiety mogą być z lewej strony lub nad TDBEdit. Jest to wybór czysto estetyczny.
6) Na kolejnej stronie akceptujemy wartości domyślne i kończymy pracę kreatora naciskając *Finish*

W tej chwili możemy przeglądać kolejne rekordy bazy.

**Drobne modyfikacje**
Możemy zastąpić DBNavigatora za pomocą własnych przycisków. Wystarczy do umieszczonych na formie klawiszy podczepić metody TTable.FindNext (następny rekord), TTable.FindPrior (poprzedni), FindFirst i FindLast. Poza prostym dublowaniem funkcji DBNavigatora możemy również dodać nowe klawisze. Np. przesuń o *n* pozycji za pomocą metody **TTable.MoveBy(*n*)** (n może być ujemne).
Można również posłużyć się komponentami ComboBox lub ListBox do wyboru rekordu z listy.

# 3. Bazy dane typu „master-detail"

Idea relacyjnych baz danych polega na umieszczaniu szczegółowych informacji o obiektach różnego typu w osobnych indeksowanych tabelach (*detail*), co daje możliwość odwoływania się do pełnych informacji tam zawartych na podstawie wyróżnionego pola-indeksu, grupowanie informacji w tabelach zbiorczych (*master*) i tworzenie relacji między tabelami. Warunkiem jest tworzenie indeksu tj. pola o niepowtarzających się wartościach.

Dla przykładu możemy sobie wyobrazić tabelę VENUES (z ang. miejsca) zawierającą informacje o obiektach sportowych (nazwa, położenie, plan itd.) oraz tabelę wydarzeń sportowych EVENTS, w której zamiast nazw obiektów, na których miały one miejsce znajduje się pole zawierające wartość indeksu z tabeli VENUES. Wówczas automatycznie mamy dostęp do szczegółowych informacji o tym obiekcie (gdyby dostępna była tylko nazwa, odpowiednie informacje musielibyśmy odnaleźć przeszukując bazę danych).

Do stworzenia formy korzystającej z relacji wykorzystamy **Database\Form Wizard**:
1) W pierwszym kroku wybieramy tym razem *Create a master/detail* form i pozostawiamy domyślną pozycję korzystającą z TTable.
2) Następnie, aby wybrać tablicę master wskazujemy na alias DBDEMOS i tablicę EVENTS.DB, uwzględniamy wszystkie pola (istotne, żeby widoczne dla programu były pola EventsNo indeksujący tę tablicę i VenueNo, który wskazuję na rekord w tablicy detail zawierający informację o obiekcie, na którym miało miejsce owo zdarzenie sportowe; ewentualnego odrzucenia niektórych pół/kolumn dokonamy na poziomie Column Editora komponentu DBGrid) i prezentujemy dane na siatce (grid).
3) Tablica detail to DBDEMOS, VENUES.DB. Także udostępniamy wszystkie pola, lecz tym razem wybieramy *Vertical* jako formę prezentacji dancyh (z etykietami z lewej strony, żeby zająć mniej miejsca w pionie).
4) Teraz musimy wskazać na relację. Indeksem bazy detail (VENUES.DB) jest pole VenueNo. Musimy wskazać pole tablicy master (EVENTS.DB), którego wartości odnoszą się do tego indeksu identyfikując obiekt sportowy. Zazwyczaj jest to pole o tej samej nazwie (tu VenueNo), lecz w tej tablicy nie jest ono oczywiście indeksem. Zaznaczamy więc VenueNo w listach Detail Fields i Master Fields i klikamy Add. W ten sposób określiliśmy relację w sposób analogiczny, w jaki robi się to np. w MS Access.
5) Naciskamy przycisk Finish kończąc tym samym pracę kreatora.

W efekcie uzyskaliśmy formę (kreator uczynił ją formą główną) podzieloną na dwie części. Górna prezentuje na siatce dane o zdarzeniach sportowych, a w dolnej znajdują się szczegółowe informacje nt. obiektu na którym zdarzenie (opisane w aktywnym rekordzie tabeli master) miało miejsce.
Połączenie Table2 (detail) z Table1 (master) odbywa się za pomocą własności Table2.MasterSource:=DataSource1 (związanej z Table1) oraz Table2.MasterField, które wskazuje pole Table1, w którym zapisana jest informacja o tym, który rekord tablicy detail (który obiekt sportowy) pokazać (w naszym przypadku jest to pole VenueNo) zgodnie z wartościami pola indeksu (wskazanym przez Table2.IndexFieldName, co jest niezbędne, jeżeli jest więcej niż jedno indeksowane pole w tabeli). Do własności MasterField przypisany jest edytor pozwalający zmieniać relacje w bazie danych (ustala wartości zarówno MasterField, jak i IndexFieldName definiując w ten sposób relację).

Zachowaj projekt na dysku.

**Zadanie**
Dodaj do formy, w górnej części (obok DBGrid1 – uwaga na ustawienie DBGrid1.Align) komponent TDBImage, który będzie prezentował zawartość pola Events_Photo z tabeli master oraz usuń to pole spośród prezentowanych w DBGrid1. Usuń Form1 z projektu.

# 4. Dodawanie kolumn z obliczoną wartością do TTable[3]

TTable reprezentuje tabelę bazy danych. Za pomocą Fields Editora można wybrać kolumny, które mają być dostępne. Można również stworzyć kolumny „wirtualne" – nieobecne w rzeczywistej tabeli, a zawierające dane obliczone na podstawie danych uzyskanych z innych pól lub pochodzące z innej tabeli (wypełnienie tabeli danymi jest zadaniem dla programisty).
Korzystanie z Fields Editora oznacza dodanie do projektu obiektów odpowiadających poszczególnym polom (do wyboru jest tyle klas ile dopuszczalnych zawartości pól, m.in. TIntegerField, TStringField, TGraphicString, TCurrencyField i wiele innych; wszystkie są pochodnymi względem klasy TField) i za ich pomocą reprezentowanie wybranych danych. Za pomocą Fields Editora (po dodaniu wszystkich pól obecnych w tabeli) **dodamy jedno dodatkowe pole, które nazwiemy bilet_cena** (pozycja New Field... w menu kontekstowym). Nie oznacza to dodania kolumny w pliku bazy danych, a tylko stworzenie „wirtualnego" pola. Automatycznie pojawi się nazwa komponentu (w C++ Builderze wskaźnika do zmiennej dynamicznej) związana z nazwą pola Table1bilet_cena. Podobnie jak w przypadku pozycji menu, pola tabeli są bezpośrednio własnościami klasy TForm1, choć nazwa sugeruje ich logiczny związek z komponentem Table1. Takie rozwiązanie, choć może nie najbardziej eleganckie, pozwala środowisku Delphi/C++ Builder ograniczyć automatyczne modyfikacje jedynie do własności i metod obiektu formy TForm1 (pochodnej względem TForm), bez tworzenia pochodnych obiektów innych typów (w tym przypadku TTable), co dopiero pozwalałoby na ich modyfikacje.
**Określmy typ stworzonego pola jako Currency** (co oznacza stworzenie komponentu Table1bilet_cena typu TCurrencyField – jego własności są dostępne w Object Inspectorze) oraz ustawmy *Calculated*, ponieważ będzie to pole, którego wartość będzie obliczana na podstawie innych danych.
Aby obliczyć wartość danych w polu bilety_cena musimy wykonać następujące działanie na obiektach reprezentujących pola tabeli:

```
//C++ Builder
Table1bilet_cena->Value=3.97*Table1Ticket_price->Value;

//Delphi
Table1bilet_cena.Value:=3.97*Table1Ticket_price.Value;
```

(na szczęście można wykonywać operacje na całych kolumnach danych!). Aby zostało ono wykonane zawsze, gdy odświeżana jest zawartość tabeli należy je umieścić w metodzie związanej ze zdarzeniem Table1.OnCalcField. Teraz, dla poprawienia estetyki aplikacji, można usunąć z DBGrid1 kolumnę Ticket_price – w programie zastąpiliśmy ją poprawioną kolumną bilet_cena.

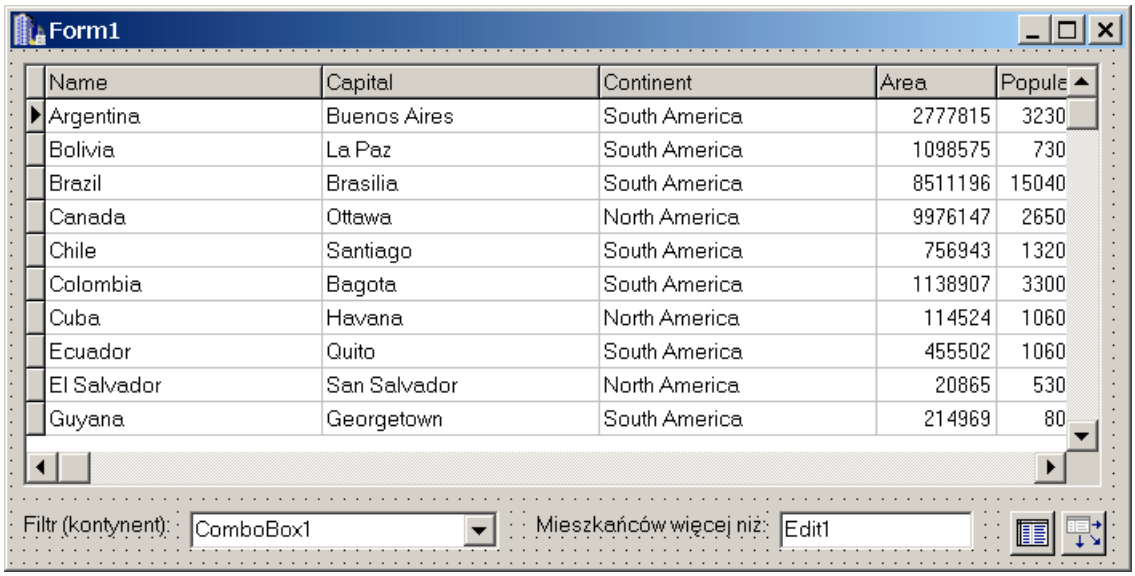
# 5. Filtrowanie

Przygotujmy prostą aplikację bazodanową wg wzoru na rysunku.

Umieszczamy na formie następujące komponenty i ustawiamy ich własności za pomocą Object Inspectora:

**Table1:**
DatabaseName: DBDEMOS
TableName: COUNTRY.DB
Active: True
**DataSource1:**
DataSet: Table1
**DBGrid:**
DataSource: DataSource1

[3] W tym paragrafie rozwijamy projekt stworzony w §3. Dodawanie wirtualnych pól możliwe jest jedynie, gdy użytkownik posiada uprawnienia do edycji tabeli.




Za pomocą edytora własności umieść w ComboBox1->Items dwie linie „South America" i „North America" (muszą być identyczne z nazwami kontynentów w tabeli).

**a) TTable.Filter**
Filtrowanie danych przekazywanych programowi odbywa się na poziomie komponentu TTable. Służy do tego własność **TTable.Filter**, której wartość jest łańcuchem (w C++ Builder AnsiString) zawierającym logiczne sformułowanie filtru np. **`Continent='North America'`**[4]. Ten tekst można wpisać w Object Inspectorze w trakcie projektowania aplikacji. Wówczas, po zmianie własności **TTable.Filtered** na True, efekt zobaczymy natychmiast.
Możemy również zmieniać jego wartość w trakcie działania programu. Do metody FormCreate za dodanymi przez kreatora formy poleceniami otwierającymi pliki baz danych dodajmy polecenia przypisania łańcucha filtru oraz aktywowania filtru::

```
//C++ Builder
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Table1->Filter="Continent='South America'";
Table1->Filtered=true;
}
```

Aby uczynić aplikację czułą na zmianę wartości w ComboBox1 do jej metody OnChange wpiszmy:

```
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
Table1->Filter="Continent='"+ComboBox1->Text+"'";
}
```

**Uwaga!** Do filtrowania nie mogą być wykorzystane pola dodane spoza tabeli bazy danych, np. pole bilet_cena. Opcje filtrowania znajdują się we własności Table1.FilterOptions. Pozwalają przede wszystkim na zaniedbanie wielkości liter porównywanych łańcuchów.

**Uwaga!** W niektórych przypadkach wygodniej jest korzystać z SQLa o czym poniżej.

**b) Table.OnFilterRecord**
Pod pewnymi względami elastyczniejszym sposobem filtrowania tabeli jest wykorzystanie zdarzenia OnFilterRecord. Pozwala ono w dowolny sposób konstruować wyrażenia filtrujące

[4] Treść własności Filter jest zapisana w formacie SQL, więc w C++ Builderze nie można korzystać z charakterystycznego dla C++ operatora porównania „==".

Wykorzystamy formę z poprzedniego przykładu – należy jedynie usunąć zdefiniowane tam filtrowania (tj. usunąć łańcuch w własności `Table1->Filter` oraz usunąć zawartość metod ComboBox1Change i FormCreate – po usunięciu wartości kompilator sam zadba o usunięcie szkieletu funkcji i ich deklaracji).

Zajmiemy się teraz oprogramowaniem zdarzenia `Table1->OnFilterRecord`. Naszym celem jest skrzyżowanie filtru znanego z poprzedniego przykładu (wybór kontynentu) z filtrem wybierającym państwa o liczbie mieszkańców większej niż zadana przez użytkownika wielkość.

Kliknijmy dwukrotnie na `OnFilterRecord` w zakładce Events Object Inspectora (F11). Zauważmy, że metoda Table1FilterRecord() w drugim argumencie zwraca przez referencję wartość logiczną Accept, która decyduje, czy dany rekord jest przekazywany do DataSource. Na początku można dla próby umieścić w tej metodzie tylko linię ustalającą wartość Accept na false: `Accept=false;`. W efekcie uzyskamy pustą tabelę.

**Uwaga!** Filtrowanie zadziała tylko, jeżeli **TTable1.Filtered** jest ustawione na True.

Aby odtworzyć filtr wybierający kontynent pobieramy wartość dla pola Continent metodą `TTable->FieldByName()->Value` (alternatywny sposób wykorzystujący tablicę Fields jest mniej wygodny, a ponadto uniemożliwia zmianę kolejności pól w tabeli bez modyfikacji programu). FieldByName posiada własność Value typu Variant (co można rozumieć jako „dowolny typ" lub „typ nieokreślony") i dlatego niezbędna jest konwersja na AnsiString.

```
AnsiString kontynent=(AnsiString)Table1->FieldByName("Continent")->Value;
```

Podobnie pobieramy wartości z pola Population i w efekcie metoda filtrująca powinna wyglądać następująco:

```
void __fastcall TForm1::Table1FilterRecord(TDataSet *DataSet, bool &Accept)
{
//nastepna linia ilustruje najprostszy dostep do pol TTable
AnsiString kontynent=(AnsiString)Table1->FieldByName("Continent")->Value;
int populacja=(int)Table1->FieldByName("Population")->Value;

Accept=(kontynent==ComboBox1->Text && populacja>min_populacja);
}
```

Wykorzystano w niej zmienną `min_populacja` (najlepiej zadeklarować ją w sekcji Private Form1). Wartość tej zmiennej będzie określana przez użytkownika za pomocą okienka edycyjnego. Dlatego ze zdarzeniem `Edit1->OnChange` wiążemy metodę:

```
void __fastcall TForm1::Edit1Change(TObject *Sender)
{
try
      {
      min_populacja=Edit1->Text.ToInt();
      }
catch(...)
      {
      min_populacja=0;
      }

Table1->Refresh();
}
```

Konstrukcja **try** – **catch** ma wyłapać błąd polegający na wpisaniu do Edit1 łańcucha nie dającego się przekonwertować na liczbę (z tego powodu warto za pomocą Object Inspectora ustalić wartość `Edit1->Text` na „0"). W przypadku tego błędu granica filtrowania min_populacja ustalana jest na 0.

Aby filtr został zastosowany po każdej zmianie wartości w Edit1.Text na końcu metody Edit1Change musi znajdować się polecenie `Table1->Refresh();`. To polecenie musi być również związane z `ComboBox1->OnChange` (można je łatwo związać korzystając z zakładki Events Object Inspectora).

# 6. Odczytywanie nazw dostępnych baz danych, tabel i pól

Możliwe jest również bardziej elastyczne konfigurowanie połączenia aplikacji z bazą danych (np. jeżeli nie znamy własności bazy danych i pozwalamy na konfigurację połączenia użytkownikowi).
Zaprojektujmy bardzo prostą przeglądarkę baz danych.

Na formę zrzućmy TTable, TSession i TDataSource. Ich własności wskazujące na konkretną tabelę będziemy ustalać w trakcie działania aplikacji. Korzystając z komponentu TSession musimy jego własności SessionName przypisać unikalną nazwę identyfikującą połączenie z bazą danych. Identyczny łańcuch (znajduje się w rozwijalnej liście) należy przypisać własnościom SessionName pozostałych komponentów.

Na formę zrzućmy także trzy listboxy (można ustawić opcje Align i użyć splittera[5]. W listboxach umieścimy odczytane w trakcie działania programu dostępne aliasy, tablice w wybranym aliasie i pola w wybranej tabeli.

W tym celu w zdarzeniu **Form1Create** umieśćmy:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Session1.ConfigMode:=[cfmPersistent]; //Tylko stałe aliasy BDE
Session1.GetAliasNames(ListBox1.Items);
ListBox1.ItemIndex:=0;
end;
```

ListBox1 (lista aliasów) nie będzie odświeżana w trakcie działania programu. Zawartość pozostałych list zależeć będzie od tego, którą pozycję na pierwszej liście wybierze użytkownik. Zatem czytanie nazw tabeli należy powiązać ze zdarzeniem **ListBox1.OnClick** uwzględniając wybraną pozycję w tym listboxie:

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
Session1.GetTableNames(ListBox1.Items[ListBox1.ItemIndex],'',true,false,
                                                          ListBox2.Items);
ListBox2.ItemIndex:=0;
end;
```

I podobnie odczytujemy (w reakcji na kliknięcie **ListBox2**) listę pól w wybranej tabeli. Są one udostępniane metodą TTable.GetFieldNames, ale warunkiem jest ustalenie odpowiednich własności komponentu reprezentującego tabelę, a więc:

```
procedure TForm1.ListBox2Click(Sender: TObject);
begin
Table1.DatabaseName:=ListBox1.Items[ListBox1.ItemIndex];
Table1.TableName:=ListBox2.Items[ListBox2.ItemIndex];
Table1.GetFieldNames(ListBox3.Items);
ListBox3.ItemIndex:=0;
end;
```

Do przeczytania pól tabeli nie trzeba jej uaktywniać (tj. zbędne, a nawet potencjalnie szkodliwe byłoby wywoływanie metody Table1.Open).

Kliknięcie na listę aliasów spowoduje odświeżenie listy tabel, ale nie spowoduje odświeżenie listy pól (niestety nie ma zdarzenia ListBox1.OnChange, które byłoby czułe nie tylko na akcje użytkownika, ale i na programową zmianę zawartości). W konsekwencji musimy o wywołanie odpowiednich metod zadbać sami: **do FormCreate dodajmy wywołanie ListBox1Click(Self) i konsekwętnie do ListBox1Click dodajmy wywołanie ListBox2Click**.

**Zadanie 1**
Można się oczywiście posunąć dalej i obejrzeć listę wartości we wskazanym polu dla wybranej bazy danych i tabeli. Analogicznie do poprzednich kroków należy napisać metodę zdarzeniową związaną z ListBox3.OnClick. Ponieważ

[5] Splitter dołączony jest do VCL dopiero od wersji bodajże 3; znajduje się na pasku Additional.

możliwa była sytuacja, że użytkownik nie kliknął na poprzednie listy musimy wywołać wpierw odpowiednie skonfigurowanie TTable i co najważniejsze wywołać metodę Table1.Open().

**Zadanie 2**
Dodać do formy TDBGrid oraz klawisz przyłączający i odłączający bazę danych (`Table1.Active:=Not Table1.Active;` lub `Table1->Active=!Table1->Active;`). TDBGrid połączyć z TDataSource i dalej z TTable tak, by po połączeniu pokazywana była zawartość wybranej tabeli. Należy pamiętać, aby uniemożliwić zmianę tabeli jeżeli jest ona otwarta (przełączyć własność Enabled w ListBox1 i ListBox2).

# 7. Quick Raport

Quick Raport to bogaty zbiór komponentów (paleta QReport) służący do budowy raportów w aplikacjach stworzonych za pomocą Delphi i C++ Builder. Tutaj dokonamy jedynie ich pobieżnej prezentacji.

**Przykładowy raport**
1) Otwórz zachowany uprzednio projekt (z punktu pierwszego, korzystający z tablicy DBDEMOS/ANIMALS.DBF)
2) Dodaj nową formę do projektu (ikona *New Form*) i z menu *File\Use Unit* dodaj Unit1 do plików widzianych przez nowy formularz. Zadbaj, aby dodatkowa forma nie była widoczna (Form2.Visible:=False)
3) **Umieść na niej komponent QuickRep** i ustal **DataSet:=Form1.Table1** oraz **Bands.HasDetails:=True**.
4) W sekcji Detail wstaw **QRLabel** (Caption:=”Nazwa zwierza:”), **QRDBText** (**DataSet:=Form1.Table1**, **DataField:=NAME**) oraz **QRDBImage** (**DataSet:=Form1.Table1**, **DataField:=BMP**)
5) Już w tej chwili, jeżeli Form1.Table1.Active jest ustawiony na True, można klikając na QuickRep i wybierając z menu kontekstowego Preview obejrzeć wygląd raportu. Jest to samodzielna forma, która posiada możliwość przeglądania zawartości naszego raportu w kilku trybach, nagrania raportu na dysk i przede wszystkim wydruku.
6) Przed kompilacją i uruchomieniem warto zajrzeć do QuickRep.PrinterSettings. Można tu wybrać zakres stron do druku, ilość kopii itp.
7) W trakcie działania aplikacji wywołamy podgląd raportu za pomocą metody **QuickRep.Preview**. W tym celu do formy głównej (Form1) dodajmy przycisk (Caption=”Podgląd”) i dwukrotnie go kliknijmy, aby stworzyć metodę związaną ze zdarzeniem domyślnym OnClick. Tam wpiszmy poleceniee Form2.QuickRep1.Preview. (Oczywiście kompilator zapyta nas o to, czy chcemy, aby Form2 było widoczne z Form1, co jest warunkiem uruchomienia metody. Jak dotąd udostępniliśmy jedynie formę główną w Form2). Efekt naciśnięcia przycisku Pogląd będzie podobny jak podgląd w trakcie projektowania.

**Uwaga!** Zbiór komponentów QReport posiada kilka błędów. Zdarza się, że lista DataField nie zawiera informacji o polach bazy danych pomimo, że DataSet został prawidłowo wybrany. W QRDBImage ustawienie Stretch:=True powoduje, że niektóre formaty obrazków nie są wyświetlane. Itp.

**Zadanie**
Raport uzupełnić o nagłówek i podsumowanie (Bands.HasTitle, Bands.HasSummary) oraz prosty nagłówek i stopkę na każdej stronie (HasPageHeader, HasPageFooter).

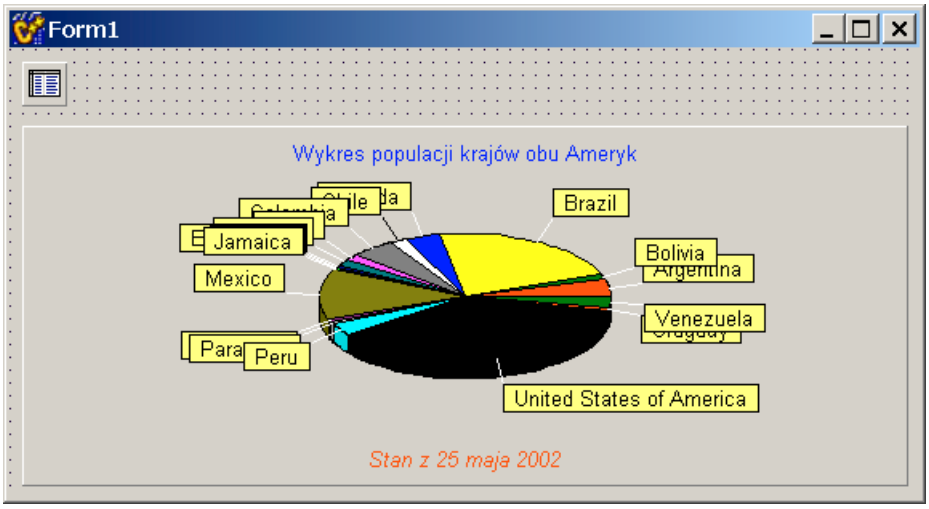
# 8. **DBChart** (C++ Builder 5 Enterprise)

W aplikacjach bazodanowych często niezbędne jest pokazanie danych w bardziej przejrzystej formie niż rzędy cyferek. Służyć do tego może komponent DBChart pozwalający na przygotowywanie wykresów. Podobnie jak QReport nie jest on dziełem Borlanda, ale jest dostarczany razem z jego produktami w wersji Enterprise. Można go również kupić od autora na stronie http://www.teechart.com (w dawniejszych wersjach TeeChart było grupą komonentów)

Obsługa jest dość prosta. Podobnie jak QReport nie korzysta z pośrednictwa DataSource.

1) Skonfigurować połączenie z tabelą zawierającą dane liczbowe (np. ANIMALS.DBF w DBDEMOS). Wystarczy tylko komponent Table1.
2) Na formie umieścić DBChart.
3) Pominę tu wszystkie aspekty estetyczne (ułożenie, kolory, itp.), zajmiemy się jedynie sporządzeniem kilku prostych wykresów.
4) Tworzenie wykresów umożliwia edytor własności **`DBChart->SeriesList`**. Należy w zakładce **Chart, Series** przycisnąć **Add ...** . Domyślnym jest wykres kołowy.

5) Następnie w zakładce **Series, Data Source** w rozwijalnej liście należy wybrać **Dataset** i jeszcze poniżej wybrać Table1.
6) W wykresie kołowym można z polami tabeli związać wyświetlane wartości (POPULATION) i etykiety (NAME)
7) Wykres stanie się bardziej czytelny, jeżeli na karcie **Chart, Legend** wyłączymy legendę.




Analogicznie postępując tworzymy wykres słupkowy obrazujący obszar poszczególnych krajów (pole AREA). Wykresy nakładają się (można ustalić ich wzajemne położenie), co umożliwia np. porównywanie danych. Można również wykorzystać wiele komponentów TDBChart.

**Export i wydruk**
Ciekawą możliwością jest Export wykresu do pliku (zarówno bitmapy jak i wektorowego WMF) – **Chart, General, Export**. Na tej samej zakładce można również wydrukować wykres.

# III. Aliasy BDE oraz łączenie z bazami danych Microsoft Access

## 1. Łączenie z MS Access i MS Excel za pośrednictwem ODBC i BDE

Zanim zaczniemy pisać aplikację korzystającą z połączenia z bazami danych Access lub danymi zawartymi w arkuszach kalkulacyjnych Excel należy najpierw przygotować połączenie za pomocą administratorów ODBC i BDE.

**Słownik:**
**ODBC** – program obsługi zapewniający dostęp do baz danych firmy Microsoft zgodnie ze specyfikacją tejże firmy *Open Database Conncectivity*. Jest konieczny przy próbie połączenia z MS Excel i FoxPro (także wykupiony przez Microsoft); umożliwia również czytanie danych z plików tekstowych.
**BDE** – zestaw metod zapisanych w bibliotekach DLL umożliwiający programom Borlanda kontakt z bazami danych. Bezpośredni kontakt przez BDE możliwy jest jedynie z bazami firmy Borland (Paradox i dBase), pozostałe muszą kontaktować się za pomocą dodatkowych narzędzi (ODBC w przypadku baz Microsoftu lub SQL Links w przypadku serwerów sieciowych baz danych Sybase, Microsoft SQL Server, Oracle, InterBase, itd.) [6]

**Uwaga!** Bazy danych MS Access są jedynymi bazami nie będącymi własnością Borlanda, z którymi można połączyć się za pomocą BDE z pominięciem ODBC. Odbywa się to za pomocą sterownika **MSACCESS**, ale wymaga zainstalowanej aplikacji MS Access w systemie.

**ODBC – tworzenie DSN** (przykład dla bazy MS Access)
1) W Panelu Sterowania uruchamiamy kontrolkę ***Źródła danych ODBC (32 bity)***. Przechodzimy na zakładkę **DSN Użytkownika**[7] Klikamy przycisk **Dodaj...** i wybieramy **Microsoft Access Driver (*.mdb)**.
2) W nowym okienku, w polu Data Source Name wpisujemy nazwę nie zawierającą nawiasów, + itp. (wybierzmy **Studenci2000 Access**) Warto w nazwie podać typ bazy.
3) W polu Description możemy wpisać dowolny łańcuch tekstu opisującego naszą bazę danych.
4) Za pomocą przycisku **Select...** wskazujemy na plik bazy danych (wybierzmy **D:\temp\Studenci2000.mdb**).
5) Przycisk Advanced pozwala na podanie użytkownika i hasła, który może czytać dane z tej bazy.
6) W ten sposób stworzyliśmy tzw. DSN („protokół" kontaktu w tym przypadku z bazą MS Access)

**BDE – tworzenie aliasu**
Ponieważ Borland nie potrafi kontaktować się za pomocą specyfikacji ODBC, konieczny jest tłumacz. Rolę tę pełni BDE, w którym stworzymy symboliczną nazwę naszej bazy danych wskazującą na konkretny DSN. Można to zrobić zarówno z poziomu Delphi lub C++ Buildera (menu Database\Explore) jak i z Panelu Sterowania *BDE Administrator*. Wybierzmy tę drugą metodę (prawidłowo skonfigurowane aliasy pojawią się w Database Explorerze, ale dopiero po zamknięciu BDE Administratora i ponownym uruchomieniu Database Explorera).
1) W Panelu Sterowania uruchamiamy kontrolkę **BDE Administrator**. W przeciwieństwie do Database Explorer w BDE Administratorze widoczne jest stworzone przez nas połączenie DSN. Trzeba je jeszcze poprawnie skonfigurować.
2) Przechodzimy na odpowiedni alias (w naszym przypadku jest to Studenci2000 Access). Ważne jest, aby pole **ODBC DSN** wskazywało na odpowiedni DSN. Poza tym jedyną zmianą jest (a informację tą BDE Administrator mógłby przecież pobrać z naszego DSN) wpisanie w polu **DATABASE NAME** ścieżki pliku naszej bazy danych (tj. **D:\temp\Studenci2000.mdb**). W momencie edycji przy nazwie aliasu z pojawi się zielony trójkąt co oznacza przejście w tryb edycji. Aby zaakceptować zmiany należy z menu kontekstowego wybrać polecenie **Apply**. Aby sprawdzić poprawność kontaktu z ODBC powinniśmy wybrać polecenie **Open**, wpisać użytkownika i hasło jeżeli ustawiliśmy te opcję i nacisnąć OK. Aliast zostanie obwiedzony zieloną linią, co oznacza, że baza danych jest otwarta.
3) Otwieramy **Database Explorera**, aby stwierdzić poprawność naszych dotychczasowych poczynań. Wybieramy polecenie **Open**. Mimo podobnego wyglądu ten program potrafi znacznie więcej od BDE Administratora. Po

[6] Wyjaśnienie pozostałych akronimów związanych z bazami danych znajduje się np. w http://www.vbfaq.pl/ pyt. 13.7
[7] Możliwe jest tworzenie DSN (*Data Source Name*) o różnych zakresach: **DSN Systemu** może być używane przez wszystkich użytkowników, **DSN użytkownika** jest przypisane tylko do jednego użytkownika, **DSN pliku** stosowane jest z przenośnymi źródłami danych, przydatne jest również w aplikacjach sieciowych.

otwarciu bazy danych możemy np. przeglądać tabele i zawarte w nich dane. Jeżeli wszystko działa właściwie – zamykamy program.

Identycznie postępujemy w przypadku skoroszytu MS Excel (który podobnie jak baza danych jest zbiorem tabel – arkuszy).

**Uwaga!** Polskie litery w bazach danych reprezentowanych przez aliasy BDE wymagają ustawienia własności LANGDRIVER na ANSI Paradox Polish.
**Uwaga!** Wśród kilku warunków, pod którymi arkusze Excela mogą być w pełni traktowane jako tabele baz danych jednym z mniej oczywistych jest, że nazwy kolumn nie mogą być dłuższe niż 64 litery.

**Przykładowa aplikacja współpracująca z bazą danych MS Access lub skoroszytem MS Excel**

1) Na formie umieszczamy komponent TTable (w rozwijalnym menu Object Inspector\Database Name powinien być widoczny nasz alias; czasem, aby zobaczyć aktualną listę trzeba, podobnie jak Database Explorera, Buildera/Delphi uruchomić ponownie. Zdarza się, że niezbędne jest ponowne uruchomienie systemu). Wybieramy DatabaseName (Studenci2000 Access). A następnie wybieramy TableName (2 sem; o ile baza nie jest otwarta, zostaniemy zapytani o hasło).
2) Umieszczamy jeszcze dwa komponenty TTable i łączymy je z pozostałymi tablicami bazy danych (Grupy PUK i Pracownie elektroniczne; pomijamy stworzoną przez nas kwerendę).
3) Umieszczamy komponent typu TDataSource i ustawiamy DataSet na Table1. Tylko jeden DataSource pozwoli na proste przełączanie oglądanej w wizualnych komponentach tablicy – będziemy po prostu zmieniali własność DataSet.

Bardzo często spotykanym problemem związanym z wyświetlaniem tabel MS Access w TDBGrid jest zbyt duża szerokość kolumn związanych z polami zawierającymi łańcuchy. Można temu łatwo zaradzić wykonując polecenie:

```
//C++ Builder
for(int i=0;i<DBGrid1->Columns->Count;i++)
            DBGrid1->Columns->Items[i]->Width=150;
```

## 2. Prywatne alisy BDE (TDatabase)

Aby skonfigurować prywatny alias BDE tzn. alias widziany jedynie podczas działania aplikacji musimy wykorzystać komponent TDatabase. W czasie działania aplikacji alias widziany jest w Database Explorerze, ale jeżeli z jego poziomu próbujemy go zamknąć – zniknie z listy dostępnych aliasów BDE.

Kładziemy na formę i konfigurujemy następujące komponenty:

1. **TDatabase**
   a) **DatabaseName = Bibliografia** (w przeciwieństwie nazwy aliasu publicznego przechowywanego we własności AliasName, alias prywatny przechowywany jest we własności DatabaseName)
   b) **DriverName = MSACCESS** (sterownik BDE pomijający ODBC)
   c) Istotą skonfigurowania połączenia prywatnego jest edycja łańcucha **Params**. W najprostszym przypadku może on składać z jednej linii
   `DATABASE NAME=D:\BIBLIO.MDB`[8]
   Ważne, żeby nie pominąć spacji w nazwie klucza „DATABASE NAME" i nie dodać żadnej spacji przy znaku „=". Nazwa pliku musi być podana z pełną ścieżką dostępu. Wzory możliwych nazw pozycji dla danego sterownika można podejrzeć w Database Explorerze dla aliasu tego samego typu.
   d) Jeżeli baza danych nie jest chroniona hasłem lub login i hasło zostało umieszczone we własności Params można ustawić **LoginPrompt = False**.
2. **TTable**
   a) **DatabaseName = Bibliografia** (nazwa pojawiła się w rozwijalnej liście)
   b) TableName = Titles (lub każdy inny)
   c) Active = True
3. TDataSource
   a) DataSet = Table1
4. TDBGrid
   a) DataSource = DataSource1

[8] Wykorzystany w tym przykładzie plik bazy danych jest dostarczany z MS Visual Studio 6.0. Może go jednak zastąpić dowolna baza danych – nie koniecznie MS Access.

Jeżeli wszystko poszło dobrze w siatce pojawią się rekordy bazy danych BIBLIO.MDB.

Stworzony w ten sposób alias prywatny, pomimo, że jest widoczny w Database Explorerze, nie jest zapisywany do plików konfiguracyjnych BDE. Można się więc nim posługiwać nawet wówczas, gdy użytkownik nie ma uprawnień do modyfikacji tych plików lub w ogóle do edycji katalogu Windows.

# 3. Łączenie z bazami danych MS Access za pomocą ADO

Nowszym sposobem łączenia aplikacji z bazami danych (przede wszystkim Microsoft Access) umożliwia mechanizm **ADO** (ang. *ActiveX Data Objects*) dostępny w produktach Borlanda od wersji 5. Wykorzystuje sterowniki systemowe (od Windows 95 SE), co zwalnia programistę od dystrybucji obszernej biblioteki BDE. ADO może być wykorzystywany zarówno do lokalnych baz danych, jak i aplikacji typu Klient/Serwer.

1) Na formie umieszczamy komponent **TADODataSet**. Najpierw musimy skonfigurować połączenie z bazą danych łańcuchem **ConnectionString** obecnym w większości komponentów na palecie ADO. Edytor własności umożliwia wybór parametrów połączenia (klawisz Build ...): na zakładce **Dostawca** wybieramy dostawcę (providera): **Microsoft Jet 4.0 OLE DB Provider**, a na zakładce **Połączenie** wskazujemy na plik bazy danych.
2) W przypadku łączenia ADO poprzez ODBC jako dostawcę należy wybrać **Microsoft OLE DB Provider for ODBC Drivers** i dalej postępować jak w przypadku wykorzystywania BDE.
3) TADODataSet wymaga polecenia SQL, które jest odpowiedzialne za pobranie danych z tabeli. Odpowiedni edytor umożliwia zbudowanie polecenia typu SELECT FROM bez znajomości tego języka: w tym celu wywołujemy **edytor własności CommandText**, w którym wybieramy jedną tabelę bazy danych i dowolną ilość pól (wszystkie tabele reprezentowane są przez *) wykorzystując klawisze Add ... to SQL. W przypadku nazw tabeli zawierających spację należy ręcznie otoczyć nazwę nawiasami prostokątnymi np. SELECT * FROM **[**Pracownie elektroniczne**]**
4) Następnie na formie umieszczamy „zwykły” **DataSource1** z karty DataAccess i ustalamy własność **DataSet** na **ADODataSet1**.
5) W końcu na formie umieszczamy komponenty z zakładki DataControls (np. DBGrid) i ustalamy ich własność DataSource na DataSource1.
6) Sprawdzianem poprawności połączenia z bazą danych jest ustawienie własności **`ADODataSet1->Active`** na True.

**Uwaga!** Grupa komponentów ADO oprócz najbardziej podstawowego komponentu reprezentującego tabelę (TADODataSet) zawiera odpowiedniki klasycznych TADOTable i TADOQuery ułatwiające zmianę mechanizmu łączenia z bazami danych z BDE na ADO.

# IV. Wykorzystanie SQL z poziomu aplikacji

Komponent TQuery umożliwia wykorzystanie języka SQL (ang. *Structured Query Language*). Ze względu na specyfikę tego języka możliwe są jego dwa zastosowania:

1) Po pierwsze TQuery umożliwia dostęp do bazy danej w sposób całkowicie analogiczny jak TTable z wykorzystaniem polecenia SELECT
2) Wykonywanie operacji na tabelach, tzn. wykorzystanie pozostałych poleceń SQL: CREATE, ALTER, DROP, DELETE, INSERT, UPDATE itd.

## 1. TQuery vs TTable

TTable i TQuery są alternatywnymi komponentami służącymi (z pomocą TDataSource) do udostępniania baz danych. Różnica polega na tym, że TQuery umożliwia użytkownikowi podanie komendy SQL (zazwyczaj SELECT). Porównamy wykorzystanie komponentów TTable i TQuery. W tym celu na formie należy umieścić po jednym z każdego typu komponentów i powiązać je z dwoma komponentami TDataSource i dalej z dwoma komponentami TDBGrid.

1) Zaznaczmy **Table1** i **Query1** i ustawmy ich własności **DatabaseName** na DBDEMOS.
2) Ponadto w **Table1 ustawmy TableName** na ANIMALS.DBF, a w Query1 własność SQL na **`SELECT * FROM „Animals.dbf”`**
3) Powiążmy DataSource1 z Table1 oraz DataSource2 z Query1 (własności DataSet), a następnie DBGrid1 z DataSource1 i DBGrid2 z DataSource2.
4) Aby zobaczyć wynik naszych zabiegów należy zaznaczyć Table1 oraz Query1 i ustawić własność **Active** na true.

Efekt powinien być identyczny (zbędne jest nawet kompilowanie programu) – powinniśmy zobaczyć całą tablicę ANIMALS. W obu komponentach można ustawić filtry (należy pamiętać o włączeniu własności Filtered), ale możliwości SQLa sa znacznie większe. Zastąpmy podane w drugim punkcie polecenie, po chwilowym wyłączeniu Query1, przez **`SELECT Name, Area FROM "Animals.dbf" WHERE WEIGHT<10`**. W wyniku otrzymujemy tylko pola „Name” i „Area”, i tylko te rekordy, w których pole „Weight” ma wartość mniejszą niż 10.

## 2. Local SQL (Delphi)

Przygotujemy aplikację – terminal SQLa – pomocną w nauce SQLa[9]. BDE umożliwia wykorzystanie okrojonej wersji SQLa do dostępu do lokalnych baz Paradox, DBase i FoxPro. Opiszę dwie wersje tej aplikacji – podstawową wersję dla Delphi i bardziej funkcjonalną – zaprojektowaną z pomocą C++ Buildera.

Przygotujmy formę według poniższej ilustracji:


Dla ułatwienia korzystania z tej aplikacji umieścimy zestaw przykładowych poleceń SQL w łańcuchu `ComboBox1->Items`:

```
CREATE TABLE "Studenci" (id SMALLINT, Nazwisko CHAR(20), Imie CHAR(10), Drugie_Imie CHAR(10))
ALTER TABLE "Studenci" ADD Oceny NUMERIC
ALTER TABLE "Studenci" DROP Drugie_Imie
ALTER TABLE "Studenci" DROP Oceny, ADD Ocena_konc NUMEIC
INSERT INTO "Studenci" VALUES (0, "Mouse", "Miki",3)
INSERT INTO "Studenci" (id, Nazwisko, Imie) VALUES (1, "Duck", "Donald")
INSERT INTO "Studenci" (id, Imie) VALUES (2, "Goofy")
UPDATE "Studenci" SET Ocena_konc=4.5 WHERE id=1
UPDATE "Studenci" SET Ocena_konc=Ocena_konc+1 WHERE Ocena_konc<4
DELETE FROM "Studenci" WHERE id=2
DELETE FROM "Studenci" WHERE Ocena_konc<3
```

[9] Warunkiem jej udanej dystrybucji na dowolny komputer jest dołączenie BDE.

```
DROP TABLE "Studenci"
CREATE TABLE "Studenci_kopia" (id SMALLINT, Nazwisko CHAR(20), Imie CHAR(10), Oceny_konc NUMERIC)
INSERT INTO "Studenci_kopia" SELECT * FROM "Studenci"
INSERT INTO "Studenci_kopia" SELECT * FROM "Studenci" WHERE Ocena_konc>2
INSERT INTO "Studenci_kopia" SELECT MAX(Ocena_konc) FROM "Studenci"
```

Następnie należy skonfigurować połączenie TQuery z bazą danych (zbędne jest uaktywnianie TQuery – własność Active może pozostawać w trakcie działania cały czas false).
Niech baza danych typu **Paradox** będzie bieżącym katalogiem. Żeby tak było nie musimy robić absolutnie nic – są to domyślne ustawienia w Delphi. Pewnym utrudnieniem może być ograniczenie nałożone na długość łańcucha ścieżki dostępu do katalogu/bazy danych w Paradox 5.0 – jeżeli nazwa bieżącego katalogu jest zbyt długa możemy możemy określić inny katalog bazy danej wpisując do **Query1.DatabaseName** nazwę katalogu np. C:\TMP

Aby wykonać polecenie zapisane w ComboBox1.Text po naciśnięciu przycisku „Exec SQL" należy związać ze zdarzeniem Button1.OnClick następującą procedurę:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Query1.SQL.Clear();
Query1.SQL.Add(ComboBox1.Text);
Query1.ExecSQL();
end;
```

TQuery posiada własność SQL typu TString (my będziemy korzystali zawsze tylko z jednej linii). Czyścimy go poleceniem Clear i wczytujemy za pomocą metody Add zawartość ComboBox1.Text. Wywołanie polecenia Query1.ExecSQL powoduje wykonanie na naszej bazie danych zawartych w Query1.SQL poleceń.

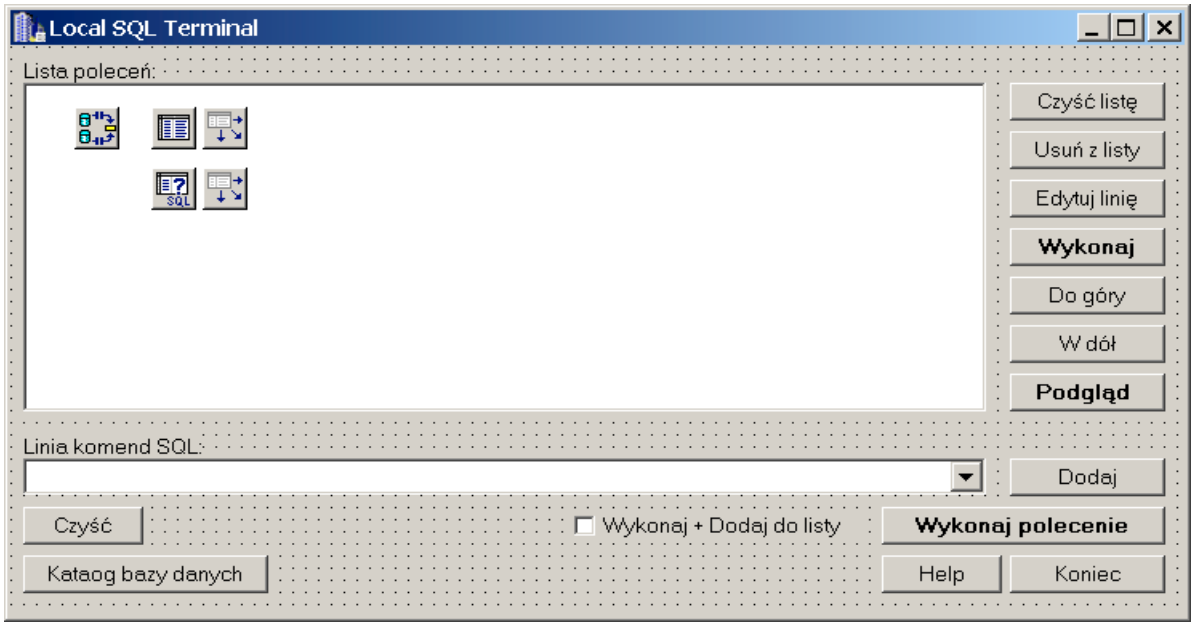
Podgląd zmian dokonanych w strukturze tabel w bazie danych można obserwować podglądając pliki w katalogu bazy danych, natomiast zawartość tabeli można obejrzeć z pomocą Database Explorera (wywołanie z menu Delphi Database, Explorer).

**Zadanie**
Dodać do projektu podgląd wybranej tabeli w bieżącym katalogu w komponencie TDBGrid umożliwiający śledzenie zmian w modyfikowanej poleceniami SQL tabeli.

# 3. Wersja rozbudowana „Local SQL" (C++ Builder)

Wersja C++ Builder jest bardziej rozbudowana. Poniżej znajduje się pełny listing trzech form:

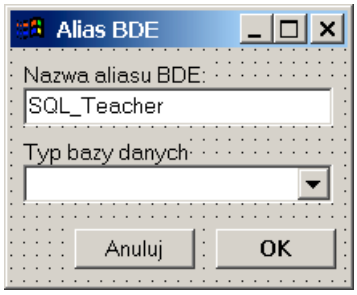
**Form1:**



W ComboBox1 umieszczamy listę poleceń SQLa identyczną jak w wersji Delphi uzupełnioną jedynie o przykłady poleceń SELECT, których skutki ta wersja aplikacji będzie również potrafiła pokazać:

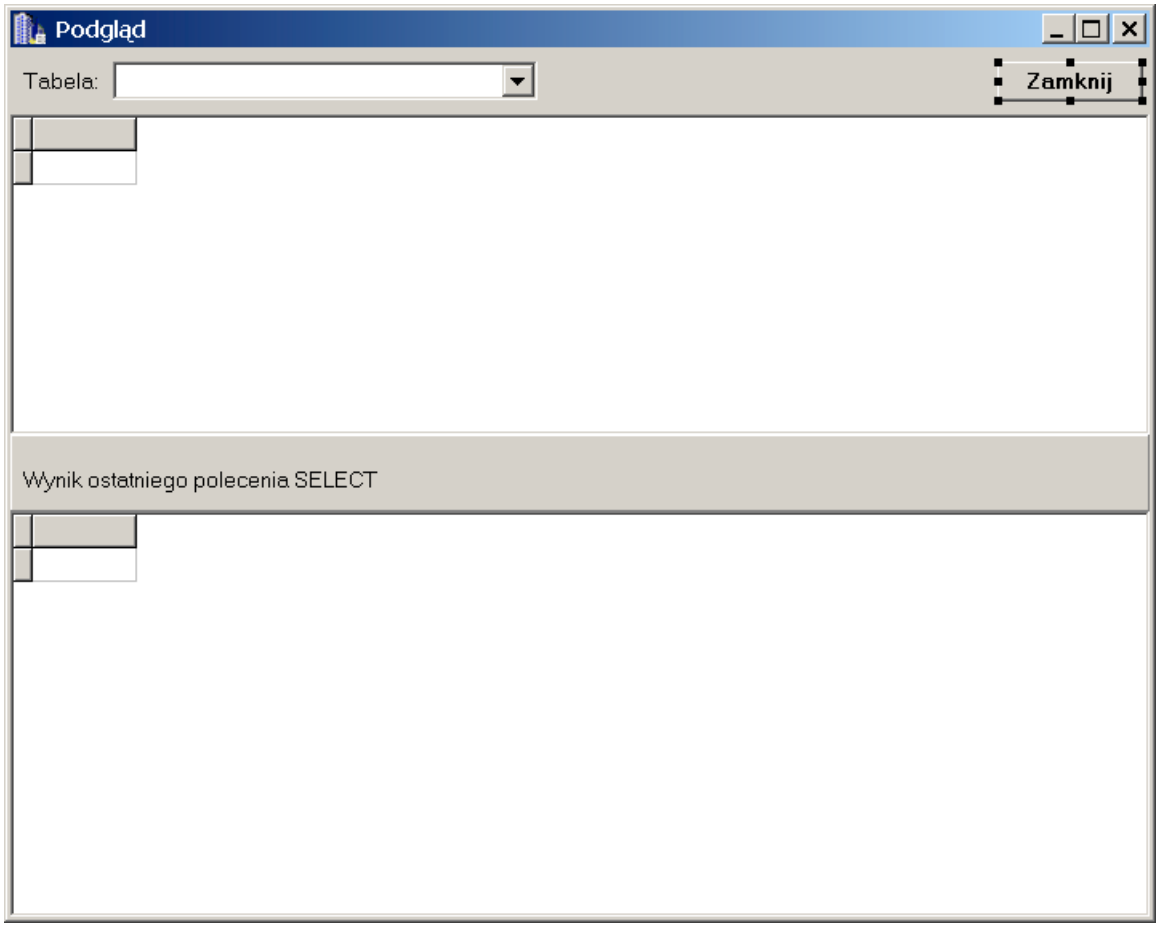
```
SELECT * FROM "Studenci" WHERE Ocena_konc<4.5
```

**Form2:**




W `ComboBox1->Items` umieszczamy pozycje PARADOX i DBASE. Aby ograniczyć bazy danych do tych dwóch typów ustalamy własność `ComboBox1->Style` na csDropDownList. Własność formy BorderStyle zmieniamy na fsDialog.

**Form3:**




W Form3 umieszczamy dwa komponenty TDBGrid – na ilustracji wykorzystano także dwa panele i opcje Align umożliwiające automatyczne dopasowywanie komponentów do rozmiaru formy.

**Alias BDE**
Aby jako pierwsze pojawiało się okienko formy drugiej – okno wyboru własności aliasu należy zablokować pokazywanie Form1 (formy głównej) poleceniem dodanym do pliku aplikacji (Project1.cpp) przed tworzeniem formy: `Application->ShowMainForm=false;` i jednocześnie ustalić, najłatwiej za pomocą inspektora obiektów, własność Form2->Visible na true.

Tworzenie aliasu BDE będzie następowało po naciśnieciu klawisza „OK". Zamknięcie okna lub naciśnięcie „Anuluj" zakończy działanie aplikacji. Do sekcji Private Form2 dodajemy zmienną typu logicznego IsOK, której wartość ustalamy w konstruktorze na false.

```
//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
      : TForm(Owner)
{
IsOK=false;
ComboBox1->ItemIndex=0;
}
//---------------------------------------------------------------------------
void __fastcall TForm2::Button1Click(TObject *Sender)
{
IsOK=true;
Form1->CreateBDEAlias();
Form1->Show();
Close();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::Button2Click(TObject *Sender)
{
Close();
}
//---------------------------------------------------------------------------
void __fastcall TForm2::FormClose(TObject *Sender, TCloseAction &Action)
{
if (!IsOK) Application->Terminate();
}
//---------------------------------------------------------------------------
```

Jeżeli wybrany zostanie klawisz „OK" (Button1) wywołana zostanie metoda formy głównej Form1.CreateBDEAlias(), która stworzy odpowiedni alias BDE i pokazana zostanie forma główna. W każdym przypadku Form2 zostanie zamknięte.

Wywołana metoda tworząca alias (wymaga oczywiście odpowiedniej deklaracji w sekcji public – musi być dostępna dla Form2) jest następująca:

```
void __fastcall TForm1::CreateBDEAlias()
{
AliasName=Form2->Edit1->Text;
DatabaseType=Form2->ComboBox1->Text;

char tmpdir[MAX_PATH];
GetCurrentDirectory(MAX_PATH,tmpdir);
DatabasePath=(AnsiString)tmpdir;

ShowMessage("Tworzę tymczasowy alias BDE do bazy danych ...\nNazwa aliasu:"
+AliasName+"\nTyp bazy danych: "+DatabaseType+"\nŚcieżka: "+DatabasePath);

/* TWORZENIE ALIASU */
if (Session1->IsAlias(AliasName))
    {
    ShowMessage("Alias "+AliasName+" już istnieje");
    return;
```

```
    };

TStringList* Opcje=new TStringList;
Opcje->Add("LANGDRIVER=Pdox ANSI Polish");
Opcje->Add("DATABASE NAME="+AliasName);
Opcje->Add("PATH="+DatabasePath);
Session1->AddAlias(AliasName,DatabaseType,Opcje);
delete Opcje;

Session1->SaveConfigFile();
/* KONIEC TWORZENIA ALIASU */
//Alias jest usowany w Form1Close

Form1->Caption=" "+AliasName+", "+DatabasePath;
}
```

Należy równocześnie w sekcji private zadeklarować trzy zmienne łańcuchowe:

```
private:    // User declarations
      AnsiString AliasName;
      AnsiString DatabasePath;
      AnsiString DatabaseType;
```

Część z nich będzie potrzebna także w innych metodach Form1.

**Uwaga!** Tworzony tutaj alias jest aliasem stałym, co oznacza, że informacja o nim jest zapisywana do plików konfiguracyjnych BDE. Ponieważ jest on kasowany przy zamknięciu aplikacji, to znacznie rozsądniejsze byłoby korzystanie z aliasu prywatnego stworzonego przez TDatabse analogicznie do paragrafu „Prywatne aliasy BDE (TDatabase)".

Stworzenie aliasu BDE umożliwiają metody komponentu TSession. Najpierw należy, najprościej za pomocą Object Inspectora, **nadać sesji unikalną nazwę** np. „LocalSQL". W pierwszych liniach metody z okienek edycyjnych Form2 pobieramy typ i nazwę dla aliasu BDE i umieszczamy je w zmiennych globalnych. Podobnie do zmiennej globalnej zapisujemy nazwę bieżącego katalogu, w którym chcemy umieścić bazę danych. Następnie sprawdzamy, czy alias już nie istnieje. Jeżeli nie tworzymy łańcucha typu TStringList, do którego zapisujemy opcje aliasu (polski zestaw znaków, nazwa i położenie katalogu) i tworzymy alias poleceniem `Session1->AddAlias` i zapisujemy go do pliku BDE za pomocą `Session1->SaveConfigFile()`.
Po uruchomieniu aplikacji (po stworzeniu aliasu) można uruchomić Database Explorer, aby zobaczyć efekt działania tej metody.

Aby alias był usunięty przy zamknięciu aplikacji należy umieścić w Form.OnClose metodę `Session1->DeleteAlias` sprawdzając wcześniej dla pewności, czy odpowiedni alias istnieje:

```
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
ShowMessage("Kasowanie tymczasowego aliasu ...");
if (!Session1->IsAlias(AliasName))
    {
    ShowMessage("Alias "+AliasName+" już nie istniał!");
    return;
    };

Session1->DeleteAlias(AliasName);
Session1->SaveConfigFile();
```

Meotda wykonująca polecenie SQL związana jest z typem tego polecenia. Jeżeli jest to SELECT uaktywniamy Query1 metodą Open() (należy ustalić w inspektorze obiektów `Query1->SessionName` na nazwę naszej sesji) i łączymy przez DataSource2 z `Form3->DBGrid2`, w pozostałych przypadkach wykonujemy ExecuteSQL. Aby sprawdzić, który typ polecenia wpisany jest w comboboxie monitorujemy jego własność tekst korzystając ze zdarzenia OnChange:

```
void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
```

```
if (ComboBox1->Text.AnsiPos("SELECT")==1)
      {
    ComboBox1->Font->Color=clNavy;
    IsSELECT=true;
    }
    else
    {
    ComboBox1->Font->Color=clWindowText;
    IsSELECT=false;
    }
}
```

Polecenie SELECT akcentujemu niebieskim kolorem tekstu; zmienna IsSELECT musi być zadeklarowana jako zmienna dostępna dla całej formy (najlepiej w sekcji private).

Metoda wykonująca polecenie zawarte w `ComboBox1->Text` uwzględnia wartość zmiennej IsSELECT:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (ComboBox1->Text.IsEmpty()) return;

Query1->Close();
Query1->SQL->Clear();
Query1->SQL->Add(ComboBox1->Text);
if (IsSELECT)
    Query1->Open();
    else
    {
    Table1->Close();
    Query1->ExecSQL();
    }
}
```

Tabele naszej bazy danych możemy zobaczyć w `Form3->DBGrid1`, która jest podłączana do Table1 przez DataSource1 przy każdym pokazaniu Form3 (zawartość bazy danych może być zmieniana):

```
void __fastcall TForm3::FormShow(TObject *Sender)
{
Form1->Session1->GetTableNames("","",false,true,ComboBox1->Items);
ComboBox1->ItemIndex=0;

Form1->Table1->TableName=ComboBox1->Text;
if (!Form1->Table1->TableName.IsEmpty()) Form1->Table1->Open();
}
```

W pierwszej linii pobierane są aktualne tabele bazy danych i wczytywane do ComboBoxu. Nazwa tabeli pobierana jest z pierwszego elementu na liście `ComboBox1->Items`. Aby zobaczyć inne tabele, należy wykorzystać własność `ComboBox1->OnChange` pamiętając, że zmienić własność `Form1->Table1->TableName` można tylko przy nieaktywnej tabeli:

```
void __fastcall TForm3::ComboBox1Change(TObject *Sender)
{
Form1->Table1->Close();
Form1->Table1->TableName=ComboBox1->Text;
if (!Form1->Table1->TableName.IsEmpty()) Form1->Table1->Open();
}
```

**Zadanie**
Dodać obsługę ListBox1. Może funkcjonować jako historia poleceń i/lub jak listing „programu” SQL.