ZAAWANSOWANE FUNKCJE FORMULARZY

Filtrowanie rekordów

Projektując strukturę bazy danych ,założyliśmy ,że tabele TranNagl i TranSzcz mogą w przyszłości posłużyć do przechowywania informacji o wystawionych fakturach, ale również do gromadzenia innych danych tego typu (faktur zakupu, dokumentów magazynowych itp.) Co prawda, na razie w tych tabelach będą umieszczane tylko dane opisujące wystawione faktury, ale warto z wyprzedzeniem zatroszczyć się o zdefiniowanie mechanizmu pozwalającego rozdzielić poszczególne grupy dokumentów. Aby ograniczyć liczbę wyświetlanych rekordów, można wykorzystać jedno z następujących narzędzi:

• właściwość Filter obiektu TTable
• komponent TQuery zamiast TTable i zdefiniować w nim zapytanie z odpowiednimi kryteriami selekcji
• metody SetRangeStart, SetRangeEnd i SetRangeApply
• metodę SetRange

W naszej aplikacji zastosujemy ostatnie z wymienionych rozwiązań. Metoda SetRange należy do komponentu TTable i ma następujący nagłówek:
procedure SetRange (const StartValues, EndValues : array of const);
W tabelach typu Paradox , metoda SetRange działa ty;lko na polach indeksowanych. Jak pamiętamy, w tabeli TranNagl zdefiniowaliśmy pole o nazwie Typ, w którym będzie zapisywany typ transakcji. Utworzyliśmy również indeks nazwie Dokument, w którego skład wchodzą pola Typ, Seria, Numer i Dopisek. Wykorzystamy go do definicji zakresu
1 1.Otwórz Delphi i uruchim plik projektu Faktury.dpr
2.Przejdź do widoku modułu danych FaktDModule
3.W sekcji interface modułu umieść deklarację nagłówka procedury : procedure TylkoSprzedaz;
4.W sekcji implementattion modułu wpisz poniższą procedurę:
procedurę TFaktDModule.TylkoSprzedaz;
begin;
Tabel1.SetRange([SPRZ],[SPRZ]);
end;
Metoda SetRAange przyjmuje jako parametry dwie tablice stałych. Pierwsza z nich określa początek uwzględnianego zakresu, druga jego koniec. W powyższym przykładzie obydwie tablice stałych zawierają po jednym ciągu znaków, co oznacza ,że przy ustalaniu zakresu zostanie uwzględnione tylko pierwsze pole z bieżącego indeksu. Ponadto, aby metoda SetRange zadziałała prawidłowo, musimy wybrać indeks, którego pierwszym jest pole Typ z tabeli TarnNagl
5.Zaznacz komponent Table1 i w Object Inspectorze odszukaj właściwość IndexName
6.Z listy wybierz indeks Dokument. Po wprowadzeniu powyższych modyfikacji wykonanie procedury TylkoSprzedaz spowoduje ,że zestaw danych Table1 będzie udostępniał tylko te rekordy, które w polu Typ mają wpisaną wartość "SPRZ". Oczywiście nie ma sensu zmuszać użytkownika, aby tworząc nagłówki kolejnych faktur, za każdym razem uważnie wypełniał pole Typ. Pole to jest używane wewnętrznie przez aplikację i została zaprojektowane "na wyrost"., z myślą o innych typach transakcji, które mogą być zaimplementowane w aplikacji. Najlepiej jeśli aplikacja sama je wypełnia odpowiednią wartością. Pozostało nam jeszcze wybrać miejsce, z którego będzie uaktywniana porcedura TylkoSprzedaz .Najbardziej stosowny wydaje się moment wyświetlania listy faktur
7.Przejdź do kodu formularza MenuGlForm
8.Odszukaj procedurę WyswietlFakt i zmodyfikuj ją aby uzyskała postać:

```
procedure TMenuGlForm.WyswietlFakt;
begin
FaktDModule.TylkoSprzedaz;
FaktLForm.ShowModal;
```

end;
9.Kliknij przycisk Save All ,aby zapisać wprowadzone zmiany

Stosowanie metody SetRange

Prototyp metody SetRange jest następujący:
procedure SetRange (const StartValues, EndValues : array of const);
Z zapisu tego można wywnioskować ,że obydwa argumenty są tablicami stałych. Co więcej, rozmiary tych tablic nie są z góry ustalone. Oznacza to ,że każdy z parametrów może być tablicą wieloelementową. Przyjęcie tego rozwiązania staje się oczywiste, gdy zdamy sobie sprawę ,że ograniczenie zakresu danych może być sformułowane na podstawie wartości kilku pól. Istnieje tu pewne ograniczenie. Pole użyte do określenia początkowej i końcowej zakresu muszą być kolejnymi polami bieżącego indeksu. W naszym przykładzie korzystamy z indeksu Dokument, w skład którego wchodzą kolejno pola : Typ, Seria, Dopisek i Numer, dzięki czemu powyższy przykład jest prawidłowy. Jeżeli chcesz zastosować inne pola definiujące zakres, musisz najpierw utworzyć odpowiedni indeks.

Zastosowanie stałej do określania zakresu

Zwróć uwagę ,że w procedurze TylkoSprzedaz wokół znaków SPRZ brak apostrofów - to nie pomyłka. Bezpośrednie odwoływanie się do ciągów znaków nie jest dobrą praktyką programistyczna. Zawsze należy dążyć do definiowania stałych. Uzasadnienie jest bardzo proste. Projektujące ten moduł pamiętamy, że faktury mają w polu Typ wpisaną wartość 'SPRZ'. Jednak pisząc kolejne fragmenty kodu możemy o tym zapomnieć i użyć ciągu 'Sprz' lub 'SPR', co zostanie zaakceptowane przez kompilator, ale niestety będzie powodować błędne działanie aplikacji. Zdefiniujmy teraz stała SPRZ. Ponieważ będziemy jej używać tylko w tym module, wystarczy ,jeżeli zdefiniujemy ją w sekcji implementacji
1.Wróć do kodu modułu danych FaktDModule
2.W sekcji implementation, za dyrektywą {$R 8.dfm}, umieść poniższy kod:
const
SPRZ = 'SPRZ'
3.Kliknij przycisk Save All, aby zapisać zmiany
4.Skompiluj i uruchom aplikację
Wystarczy teraz pamiętać, aby nie używać bezpośrednio ciągu znaków "SPRZ" lecz stałej o tej nazwie, a ryzyko popełnienia błędu znacznie zmaleje. Jeśli chcesz możesz teraz wykonać następujący eksperyment: wyświetl formularz z listą faktur i spróbuj wstawić nowy rekord, wpisując w polu Typ litery SPRZ (koniecznie wielkie) Zatwierdź wprowadzony rekord. Spróbuj teraz zmienić wartość w tym polu na dowolną inną i ponownie zatwierdź zmiany. Rekord zniknie z ekranu. Dzieje się tak, ponieważ wartość w polu Typ przestała spełniać kryteria filtru. Jednak w tabeli TranNagl rekord ten w dalszym ciągu istnieje. Stałe należy definiować tak aby ich zasięg był jak najmniejszy - to znaczy ograniczony do fragmentów kodu w których ta stała jest faktycznie używana. Jeżeli definiujemy stałą globalną, należy ją umieścić w sekcji interfejsu. Stała lokalna powinna się znaleźć w sekcji implementacji. Stała wykorzystywana tylko w jednej procedurze powinna być zdefiniowana w tej procedurze. Ta sama zasada dotyczy zmiennych

Wstawianie wartości domyślnych do nowego rekordu

Dobrze przygotowana aplikacja powinna jak najbardziej ułatwiać życie użytkowania .Do takich ułatwień zalicza się między innymi wartości domyślne proponowane w niektórych polach w momencie tworzenia nowego rekordu. W tabeli TranNagl możemy proponować następujące wartości domyślne:

Pole : Typ
Wartość : SPRZ
Pole : Seria
Wartość : klucz Seria z pliku Faktury.ini
Pole : Dopisek
Wartość : klucz Dopisek z pliku Faktury.ini
Pole : Data_wystaw
Wartość : bieżąca data systemowa
Pole : Data_trans
Wartość : bieżąca data systemowa
Pole : Termin_platn
Wartość : bieżąca data systemowa
Pole : Forma płatności
Wartość : gotówka
Jak widać, do wprowadzenia pozostał tylko numer faktury oraz kontrahent.

Pobranie wartości domyślnych z pliku INI
Domyślną serię i dopisek do numeru faktury będziemy pobierali z pliku Faktury.ini w momencie wyświetlania formularza FaktLForm. Utworzymy więc odpowiednią procedurę dla zdarzenia OnShow tego formularza.
1.Kliknij przycisk View Unit i przejdź do modułu FaktD
2.Poniżej dyrektywy {$R *.dfm} wpisz procedurę:

```
procedure TFaktDModule.PobierzINI;
var
KonfigINI ; TIniFile;
begin
try
KonfigINI := TIniFile.Create('Faktury.ini'); {Otwarcie pliku}{Wczytanie serii i dopisku faktury}
FSeria : KonfigINI.ReadString('Faktury','seria' , 'FA');
FDopisek : KonfigINI.ReadString('Faktury', 'dopisek', '');
finally
KonfigIni.Free;
end;
end;
```

3.Uzupełnij deklarację procedury w definicji interfejsu klasy (powyżej słowa kluczowego private):

```
procedure PobierzIni;
```

4.W klauzuli private wpisz deklaracje zmiennych:

```
private
{Private declarations }
FSeria : string;
FDopisek : string;
```

Literka F na początku nazw zmiennych nie ma nic wspólnego z fakturą. Jest to konwencjonalne oznaczenie pola (ang. field) należącego do jakiejś klasy
5.Do klauzuli uses na początku pliku dołącz odwołanie do biblioteki IniFiles:

```
uses
SysUtils, Classes, DB, DBTables, IniFiles;
```

Zastosowanie zdarzenia OnNewRecord
Wartości domyślne powinny być przyporządkowane nowemu rekordowi w procedurze obsługi zdarzenia OnNewRecord obiektu TTable. Zdarzenie to jest uaktywniane w chwili wstawienia bądź dołączenia nowego rekordu do zestawu danych. Do przypisywania wartości domyślnych polom

nowego rekordu należy używać tylko zdarzenia OnNewRecord. Nie należy w tym celu stosować zdarzenia AfterInsert
1.Kliknij przycisk Toggle From/Unit, aby powrócić do widoku modułu danych
2.Zaznacz komponent Table1
3.W Object Inspectorze dwukrotnie kliknij pole obok zdarzenia OnNewRecord
4.Uzupełnij procedurę zdarzenia, aby wyglądała następująco:

```
procedure TFaktDModule.Table1NewRecord(DataSet : TDataSet);
begin
TableTyp.Value := SPRZ;
PobierzIni;
Table1Seria.Value := FSeria;
Table1.Dopisek.Value := FDopisek;
Table1Data_wystaw.Value := date;
Table1.Data_trans.Value := date;
Table1.Term_platn.Value := date;
Table1.Forma_platn.Value := 'gotówka';
end;
```

Wyjaśnijmy w skrócie działanie tej procedury. Jest na uruchamiana zaraz po wstawieniu nowego rekordu do określonego zestawu danych. Najpierw pole Typ uzyskuje wartość przypisaną stałej SPRZ (a więc ciąg znakrów "SPRZ"). Następnie pola Seria i Dopisek uzyskują wartości zapamiętane odpowiednio w zmiennych FSeria i FDopisek Jak pamiętamy, wartości zapamiętane odpowiednio są ustalane przez wykonywaną wcześniej procedurę PobierzIni. Następnie trzem polom daty z nagłówka faktury są przypisywane bieżące daty systemowe. Data ta jest zwracana przez funkcję standardową date. W polu Forma_platn jest wstawiany ciąg znaków "gotówka" Pole Typ z tabeli TranNagl jest wypełniane automatycznie - w przypadku faktury sprzedaży będzie w nim zawsze widniała wartość 'SPRZ'. Informacja ta podobnie jak identyfikator transakcji (pole IDTranNagl), jest wykorzystywane wewnętrznie przez aplikację, a więc użytkownik nie musi nawet wiedzieć o jej istnieniu
5.Kliknij przycisk Toggle From/Unit, aby powrócić do widoku modułu danych
6.Dwukrotnie kliknij komponent Table1. Na ekranie pojawi się lista pól tabeli Table1
7.Zaznacz pole Typ
8.Przejdź do Object Inspectora i ustaw właściwość Visible na False
9.Kliknij przycisk Save All, aby zapisać wprowadzone zmiany
10.Skompiluj i uruchom aplikację.
Możesz teraz sprawdzić , czy program rzeczywiście wstawia wartości domyślne do nowych rekordów: wyświetl listę faktur, kliknij przycisk Edytuj rekord, a następnie przycisk + na pasku nawigatora w formularzu Faktura. Jeżeli powyższy kod został wprowadzony prawidłowo, pojawi się nowy nagłówek faktury z wartościami domyślnymi.

Wyznaczanie następnego numeru faktury

Ponieważ przyjęty przez nas mechanizm wyznaczania nowego numeru faktury jest dość rozbudowany, został omówiony na stronie [http://www.chacker.pl/apend/apend.php#XI] "Automatyczne nadawanie numerów z wykorzystaniem tabeli pomocniczej" .Są w nim opisane wszystkie procedury modułu LicznDModule, zarządzającego mechanizmem nadawania numerów. W tej chwili ograniczymy się jedynie do stwierdzenia ,że aby uzyskać numer dla nowo tworzonej faktury musimy wywołać funkcję ZwiekszNumer z parametrem w postaci ciągu znaków, który jest tworzony na podstawie typu, serii i dopisku do faktury za pomocą funkcji FakturaNaLicznik. Funkcja ta ma prototyp:

```
function ZwiekszNumer(L:String) : longint;
```

Funkcja ta zwraca następny wolny numer dla licznika, podanego w postaci parametru L. W przypadku błędu (nie można zwiększyć numeru lub utworzyć nowego licznika) funkcja zwraca wartość -1. Licznikiem powinna być wartość zwrócona przez funkcję FakturaNaLicznik. Faktura będzie otrzymywała nowy numer nie w chwili przypisania polom wartości domyślnych (w zdarzeniu OnNewRecord), lecz dopiero wtedym gdy użytkownik przejdzie do pola Numer w formularzu Faktura.

Przyjęcie takiego rozwiązania jest uzasadnione - zanim zostanie wyznaczony nowy numer, użytkownik będzie mógł zmienić serię faktury, co spowoduje ,że otrzyma ona zupełnie inny numer, niż gdyby zastosowano serię domyślną Pozwoli to w prosty sposób uzyskać niezależne ciagi numeracji dla różnych serii faktur

1.Na pasku narzędzi kliknij przycisk Add file to Project

2.W folderze aplikacji odszukaj plik LicznD.pas i kliknij przycisk Otwórz. Plik zostanie dołączony do projektu

3.Kliknij przycisk View Form i wyświetl formularz FakturaForm

4.Zaznacz kontrolkę EditNumber

5.W Object Inspector dwukrotnie kliknij w polu obok właściwości OnEnter

6.Uzupełnij kod procedury zdarzenia aby uzyskała postać :

```
procedure TFakturaForm.EditNumerEnter(Sender : TObject):
var
NowyNumer : longint;
begin
{sprawdzenie, czy nie nadano już wcześniej tego numeru
if FaktDModule.Table1Numer.IsNull then
with FaktDModule do begin
NowyNumer := LicznDModule.NadajNumer (
LicznDModule.FakturaNaLicznik(
Table1Typ.AsString;
Table1.Seria.AsString;
Table21Dopisek.AsString;
if (NowyNumer < > -1 ) then {nadano nową wartość}
Table1.Numer.Value := Nowy Numer;
end;
end;
```

Pierwsza instrukcja warunkowa sprowadza, czy w polu Numer nie ma już jakiejś wartości. Jeżeli to pole jest puste, następuje wywołanie funkcji NadajNumer. Parametrem tej funkcji jest wynik zwracany przez inną funkcję - FakturaNaLicznik (a więc odpowiednio sklejony ciąg znaków uzyskany z typu ,serii i dopisku do numeru faktury). Uzyskany numer jest przypisywane zmiennej lokalnej NowyNumer. Jeżeli jej wartość jest różna od -1 (nie wystąpił błąd), jest ona przypisywana polu Numer

7.W klauzuli uses sekcji implementation wstaw odwołanie do modułu kodu LicznD

```
uses
FaktD, LicznD
```

8.Kliknij przycisk Save, aby zapisać wprowadzone zmiany

9.Skompiluj i uruchom aplikację

10.Przejdź do formularza Faktura

11.Na pasku nawigatora kliknij przycisk + ,aby rozpocząć edycję nowego rekordu. W odpowiednich polach pojawią się wartości domyślne

12.Kliknij w polu Numer. Pojawi się wartość 1

13.Zatwierdź edytowany rekord i powtórz czynności z punktów 11 i 12. Kolejny rekord otrzyma wartość

Jeśli chcesz możesz teraz za pomocą programu Database Desktop sprawdzić zawartość tabelki Liczniki. Pojawił się w niej nowy rekord, którego pole Licznik składa się z trzech elementów jednoznacznie identyfikujących fakturę :Serii, numeru i dopisku. Zawartość tego pola została wyznaczona za pomocą funkcji FakturaNaLicznik. Oczywiście da każdej z wspomnianych trzech wartości można by zdefiniować oddzielne pole w tabeli Liczniki. Jednak ponieważ informacje. Jednak, ponieważ informacje te są potrzebne tylko do wyznaczenia nowego numeru, zaproponowane rozwiązanie jest o wiele prostsze. Ponadto tabela Liczniki zyskuje na uniwersalności - można w niej przechowywać również inne liczniki, a nie tylko kolejne numery faktur

Definiowanie pól obliczeniowych

Gdyby na formularzu, w obecnej postaci, spoczęło uważne oko księgowego, od razu zadał by pytanie: "A gdzie wartości i kwoty podatku VAT poszczególnych pozycji?" .Rzeczywiście , mamy cenę jednostkową, ilość i stawkę podatku VAT, ale nie mamy kwoty brutto i kwoty podatku. Przeoczenie? Nie. Jeżeli pamiętasz, projektując strukturę bazy danych, mówiliśmy o unikaniu nadmiarowości w kontekście zachowania spójności danych. Z tego względu świadomie zrezygnowaliśmy z tych pól, ponieważ znając cenę jednostkową, ilość i stawkę podatku, można je w każdej chwili wyliczyć. W tym celu należy utworzyć pole obliczeniowe (ang. calculated), którego wartość jest wyznaczana na podstawie innych pól określonego rekordu. Utworzymy pole obliczeniowe dla wartości podatku VAT oraz kwotę brutto pozycji faktury. Dodatkowo zdefiniujemy tutaj pole, które będzie odpowiedzialne za numerowanie pozycji faktury wyświetlanych na ekranie

1.Dwukrotnie kliknij komponent Table2 na formularzu FakturaForm

2.Na ekranie pojawi się lista pól tabeli TranSzcz

3.Kliknij listę prawym przyciskiem myszki i z menu podręcznego wybierz polecenie New Field. Pojawi się okno dialogowe New Field. Za jego pomocą można definiować nowe pola w zestawie danych. Mogą to być pola jednego z trzech typów :danych (opcja Data), obliczeniowe (opcja Calclated) oraz odnośników (opcja Lookup). Pole to jest zapamiętywane w odpowiednim zestawie danych (w naszym przypadku jest nim tabela Table2) pod nazwą Name. Ponadto tworzony jest reprezentujący je komponent odpowiedniego podtypu TField - jego nazwa jest zapisana w polu Component. Dodatkowo jest określany typ (Type) oraz rozmiar (Size) tworzonego pola

4.W polu Name wpisz VAT. W polu Component automatycznie pojawi się nazwa Tabela2VAT

5.W polu Type wybierz typ Currency. Definiowane pole będzie prezentowało wartości typu walutowego.

6.Upewnij się ,że jest zaznaczona opcja Calculated i kliknij przycisk OK. Pole obliczeniowe zostało utworzone. Stosowanie pól obliczeniowych ma jeszcze jedną zaletę - ich wartość jest wyznaczana natychmiast po zmianie zawartości pól, które są elementami wyrażenia składającego się na pole obliczeniowe. Jeżeli na przykład zmienimy wartość w polu , Ilosc, pozycje VAT i Wartość zostaną automatycznie przeliczone zanim jeszcze zostanie zatwierdzony nowy rekord

7.Zamknij okno z lista pól i przejdź do widoku kodu formularza. W deklaracji klasy formularza TFakturaForm pojawiły się trzy nowe obiekty pól:

```
Table2VAT : TCurrencyField;
Table2Wartosc : TCurrencyField;
Table2Pozycja : TIntegerField;
```

W tej chwili pole Pozycja znajduje się na końcu listy w zestawie pól tabeli Table2. Zwykle jednak numeracja wszelkich list jest umieszczana w ich pierwszej kolumnie. Aby zmienić kolejność wyświetlania pól w komponencie TDBGrid, wystarczy zmienić ich kolejność w edytorze pól

8.Przesuń pole Pozycja z końca na początek listy edytora pól tabeli Table2. Po zdefiniowaniu dodatkowych pól komponent siatki może być zbyt wąski aby wyświetlić je jednocześnie. W takim przypadku zwiększ odpowiednio szerokość całego formularza. Warto również zredukować nieco szerokość kolumny Pozycja

9.Dwukrotnie kliknij komponent Table2 na formularzu FakturaForm, aby wyświetlić edytor pól

10.Zaznacz pole Pozycja i zmień jego właściwość DisplayWidth na 2 oraz właściwość DisplayLabel na Poz

11.Wróć do widoku formularza FakturaForm i odpowiednio dostosuj jego szerokość

Aby w fazie projektowej zobaczyć poszczególne kolumny siatki danych, ustaw właściwość Active komponentu Table2 na True

Obsługa zadarzenia OnCalcFields

Odpowiednie pola są już zdefiniowane. Niestety , nasza aplikacja nie wie jeszcze w jaki sposób wyznaczyć ich wartości. W tym celu musimy wprowadzić odpowiedni kod do procedury obsługi

zdarzenia OnCalcFields komponentu Table2. Wartości pól obliczeniowych powinny być wyznaczane wyłącznie w procedurze obsługi zdarzenia OnCalcFields. Procedura ta jest uruchamiana za każdym razem gdy:

• zostanie otwarty zestaw danych
• zestaw danych znajdzie się w stanie dsEdit
• z bazy danych zostanie pobrany rekord

Ponadto ,jeżeli właściwość AutoCalcFields komponentu typu TTable ma wartość True, procedura ta będzie uruchamiana również wtedy ,jeśli doszło do modyfikacji rekordu i nastąpiło przejście do innego wizualnego komponentu bazodanowego (lub do innej komórki siatki). Właściwość AutoCalcFields domyślnie ma wartość True. Jeżeli właściwość AutoCalcFields ma wartość True, procedura obsługi zdarzenia OnCalcFields nie może modyfikować zestawu danych (zmieniać wartości zwykłych pól) W przeciwnym razie będzie dochodziło do nieskończonego rekurencyjnego wywołania tej procedury. Wprowadźmy odpowiedni kod , aby pola obliczeniowe wyliczały wartości

1.Zaznacz komponent Table2
2.W Object Inspectorze wyświetl zakładkę Events i dwukrotnie kliknij w wolnym polu obok zdarzenia OnCalcFields.
3.Wpisz kod obsługi zdarzenia, aby uzyskać następującą procedurę:

```
procedure TFakturaForm.Table2CalcFields(DataSet : TDataSet);
var
Stawka : integer;
Wart_netto : double;
begin
{wyznaczenie pozycji faktury} if Table2.State < > dsInsert then
(dla istniejącego rekordu wystarczy pobrać jego numer)
Table2.Pozycja.Value := Table2.RecNo
else
{jeżeli jest to nowy record , jego numer wyznaczamy zwiększając o 1
łączną liczbę rekordów}
Table2Pozycja.Value :=Table2.RecordCount + 1;
{wyznaczenie stawki podatku VAT}
Wart_netto : Table2.Cena_jedn.Value * Table2Ilosc.Value;
{wyznaczanie stawki podatku VAT}
try
Stawka : Table2St_Vat.AsInteger;
except
on E : EConvertError do
{Wpisanej wartości nie da się skonwertować na liczbę Integer}
Stawka := 0;
end;
{wyznaczenie kwoty podatku VAT}
Table2VAT.Value : Wart_netto * Stawka/100;
{wyznaczenie wartości brutto pozycji faktury}
Table2Wartosc.Value := Table2VAT.Value + Wart_netto;
end;
```

Jako numer pozycji przyjmowany jest numer bieżącego rekordu w zestawie danych, uzyskiwany z właściwości RecNo, W naszym przypadku zestawem danych jest tabela TransSzcz, a w zasadzie jej podzbiór, ponieważ w formularzu FakturaForm wyświetlane są tylko te rekordy z tabeli TranSzcz, których wartość pola IDTranNagl jest identyczna z wartością pola IDTranNagl w bieżącym rekordzie tabeli TranNagl. Właściwość ta podaje prawidłową wartość we wszystkich stanach zestawu danych, za wyjątkiem stanu dsInsert, w którym wartość RecNo wynosi -1. Aby uniknąć wyświetlania takiej liczby na ekranie, zamiast właściwości RecNo stosujemy wtedy właściwość RecordCount, która podaje aktualną liczbę rekordów w zestawie. Ponieważ RecordCount nie uwzględnia rekordu w stanie

dsInsert, zwiększamy tę wartość o 1. Słowo na tematy wyznaczania stawki podatku VAT. Przyjęliśmy ,że stawka VAT będzie polem typu znakowego aby użytkownik mógł w nim wpisać litery "zw" (stawka zwolniona). Do obliczeń potrzebujemy jednak wartości liczbowej. Moglibyśmy tutaj skorzystać ze zdefiniowanej tabeli St_Vat, ale zastosujemy prostsze rozwiązanie. Odpowiedniej konwersji dokonuje metoda AsInteger. Zadziała ona prawidłowo, jeżeli wartość wpisana w polu St_Vat jest liczbą. W przeciwnym razie zostanie zgłoszony wyjątek EConvertError. Stanie się tak, gdy użytkownik wpisze w tym polu litery "zw" lub pozostawi je puste. Do obsługi tego typu przypadków najlepiej nadaje się klauzula try except. Jej działanie można opisać w skrócie tak : jeżeli jakaś operacja po słowie kluczowym try spowoduje zgłoszenie wyjątku, sterowanie jego jest przekazywane do części except, w której jest realizowany kod obsługi tego wyjątku.

Stany zestawu danych

DsInactive : Zestaw danych jest nieaktywny
DsBrowse : Zestaw danych jest w trybie przeglądania - jest otwarty, lecz nie można modyfikować jego wartości
DsEdit : Rozpoczęto modyfikacje bieżącego rekordu
DsInsert : Bieżący rekord jest nowym rekordem - nie był jeszcze zapisywany

Uzyskiwanie wartości przez pole typu Autoincrement

Zanim uruchomimy i przetestujemy wszystkie wprowadzone zmiany musimy rozwiązać jeszcze jeden problem. Jak zapewne pamiętasz tabele TranNagl i TranSzcz są powiązane relacją poprzez pole IDTranNagl. W tabeli TranNagl pole to jest typu AutoIncrement, co oznacza ,że jego wartość jest nadawana automatycznie przez bazę danych. I w tym miejscu pojawia się kłopot - wartość pola Autoincrement jest nadawana dopiero w momencie zatwierdzania rekordu. Jakie konsekwencje ma dal nas ta właściwość? Otóż jeżeli użytkownik rozpocznie wprowadzanie nowych rekordów szczegółów transakcji a rekord nagłówka transakcji nie został jeszcze zapisany, wówczas nie będzie można określić wartości pola IDTranNagl w rekordach szczegółów. Efekt będzie następujący - jeżeli utworzysz nowy rekord w tabeli TranNagl i od razu (bez zatwierdzania tego rekordu) rozpoczniesz wypełnianie pozycji faktury, a dopiero potem zatwierdzisz nagłówek, otrzyma on wartość w polu IDTranNagl, co będzie równoznaczne z utratą powiązania z rekordami w tabeli TranSzcz (których wartości pól IDTranNagl są puste, a więc nie spełniają warunku relacji). Rozwiązanie tego problemu jest stosunkowo proste. Wystarczy przed wstawieniem pierwszego rekordu do tabeli TranSzcz zatwierdzić nowy rekord w tabeli TranNagl. Wykonaj poniższe czynności:

1.Gdy w Object Inspectorze zaznaczony jest obiekt Table2, przejdź do karty Events i dwukrotnie kliknij w polu zdarzenia BeforeInsert. Zdarzenie to jest wzbudzane przed wstawieniem nowego rekordu, jest więc idealnym miejscem do sprawdzenia, czy rekord nagłówka transakcji został już zatwierdzony

2.Uzupełnij procedurę zdarzenia, aby wyglądała następująco:

```
procedure TFakturaForm.Table2BeforeInsert(DataSet : TDataSet);
begin
if FaktDModule .Table1.State = dsInsert then
FaktDModuel.Table1.Post;
end;
```

3.Kliknij przycisk Save ,aby zapisać wprowadzone zmiany

4.Skompiluj i uruchom aplikację ,aby sprawdzić czy kod nie zawiera błędów

Pobieranie opisów towarów/usług podczas wypełniania faktury

Do tej pory wypełniając pozycje faktury, wszystkie wartości wpisywaliśmy z klawiatury, a pole IDTowUslug pozostawało puste. Gdybyśmy spróbowali wpisać w nim jakiś identyfikator, który nie został uprzednio zdefiniowany w tabeli TowUslug, nastąpiło by zgłoszenie wyjątku "Master record missing". Informuje on nas ,że brakuje rekordu nadrzędnego. Innymi słowy: podany klucz obcy nie

ma odpowiednika w postaci klucza podstawowego w tabeli TowUslug. Dzieje się tak, ponieważ zdefiniowaliśmy relację między polami IDTowUslug z tabeli TranNagl i IDTowUslug z tabeli TranSzcz. Podobnie jak w przypadku identyfikatora kontrahenta, nie możemy wymagać aby użytkownik pamiętał wszystkie identyfikatory towarów i usług, Musimy więc stworzyć mechanizm "podpowiedzi" podczas wypełniania pozycji faktury. W terminologii baz danych mechanizm ten nazywa się odnośnikami. Oprócz identyfikatora towaru/usługi, do pozycji faktury powinny zostać przeniesione także właściwości pól : Nazwa, Cena_sprz, KWiU, J_miary i St_Vat. Dzięki temu użytkownik będzie musiał wypełnić tylko kolumnę Ilość. Oczywiście w dalszym ciągu będzie możliwe wpisywanie pozycji spoza zestawu zdefiniowanego w tabeli TowUslug - w takim przypadku kolumna ID pozostanie pusta. Niestety do zaimplementowania tego mechanizmu nie możemy użyć pola TBLookupCombo (jak zrobiliśmy to w przypadku identyfikatora kontrahenta), ponieważ dane są wyświetlane w siatce, a nie w polu tekstowym. Na szczęście po drobnych modyfikacjach również w siatce można zaimplementować mechanizm odnośników .Aby skorzystać z tego rozwiązania , musimy najpierw zdefiniować pole odnośnika

1.Uzupełnij klauzulę uses sekcji implementation aby uzyskała postać

```
uses
FaktD, LicznD, TowarD;
```

Lista odnośników musi odwoływać się do jakiegoś zestawu danych. Aby nie tworzyć kolejnego zestawu w formularzu FakturaForm, skorzystamy z zestawu zdefiniowanego w module danych TowarDModule

2.Wyświetl okno kodu modułu TowarD. Ta operacja jest potrzebna aby źródło danych TowarDModule.DataSource1 było dostępne podczas definiowania pola odnośnika.

3.Przejdź do formularza FakturaForm i dwukrotnie kliknij komponent Table2, aby wyświetlić edytor pól. Zanim zaczniemy definiowanie nowego pola, upewnij się ,czy zestaw danych jest zamknięty (właściwość Active komponentu Table2 powinna mieć wartość False)

4.Prawym przyciskiem myszy kliknij listę pól w edytorze i wybierz polecenie New Field. Na ekranie pojawi się okno dialogowe New Field

5.Wypełnij okno dialogowe. Tworzone pole otrzyma nazwę Symbol. W kodzie formularza będzie je reprezentować komponent Table2Symbol. Pole to jest typu string(12), a wiec takiego samego jak pole IDTowUslug w tabeli TowUslug .Po zaznaczeniu opcji Lookup udostępnione są funkcje z obszaru Lookup definitione. Ich znaczenie jest następujące:

• KeyFields . Pole (pola) kluczowe w modyfikowanym zestawie danych, któremu musi odpowiadać pole (pola) w zestawie danych odnośników
• Lookup Keys . Pole (pola) kluczowe w zestawie danych odnośników
• Dataset. Zestaw danych zawierających odnośniki
• Result Field. Pole w tabeli odnośników, którego wartość zostanie przypisana polu w modyfikowanym zestawie danych

6.W edytorze pól przesuń nowo utworzone pole za pole IDTowUslugi. Po wstawieniu pola odnośników z listą towarów, wyświetlanie pola IDTowUslug przestaje mieć sens, gdyż informacja na ekranie dublowałyby się

7.Na liście zaznacz pole IDTowUslug i zmień jego właściwość Visible na False

8.Kliknij przycisk Save All aby zapisać wyprowadzone zmiany

9.Skompiluj i uruchom aplikację. Przejdź do formularza Faktura. Trzykrotnie kliknij na kolumnie Symbol. Pierwsze kliknięcie powoduje uaktywnienie pola, następne przejście do trybu edycji, a trzecie rozwinięcie listy odnośników

10.Za pomocą suwaków odszukaj właściwy symbol towaru/usługi i kliknij go

Analogiczne czynności można wykonać za pomocą klawiatury. Po umieszczeniu kursora w kolumnie Symbol należy nacisnąć Enter, co spowoduje przejście do trybu edycji. Aby rozwinąć listę odnośników, należy nacisnąć kombinację klawiszy Alt+strzałka w dół. Następnie można wyszukiwać obiekty na liście używając klawiszy strzałek lub wpisując pierwsze litery szukanego kodu.

Przedstawiony mechanizm jest najprostszym sposobem uzyskania listy odnośników .Niestety ma wiele wad:

• aby uzyskać listę, trzeba aż trzykrotnie kliknąć
• na liście można wyświetlić tylko wartości pól odnośnika (np. nie można wyświetlić pełnej nazwy towaru)
• ponieważ pole odnośnika jest tylko do odczytu, nie można stosować mechanizmu przyrostowego uzupełniania ,jaki oferuje np. komponent TDBLookupComboBox itp.
Aby uzyskać bardziej zaawansowanego rozwiązania, trzeba stosować różnego rodzaju sztuczki lub tworzyć własne komponenty. Prawdopodobnie właśnie dlatego na rynku jest dostępnych tak wiele różnych komponentów będących alternatywą dla standardowej siatki TDBGrid.

Wypełnienie pól nowego rekordu wartościami z tabeli odnośników
Chcąc w pełni wykorzystać listę odnośników, nie możemy zapominać o tym ,że z każdym towarem/usługą są związane takie informacje, jak cena jednostkowa, jednostka miary, stawka podatku VAT itp. Chcielibyśmy, aby w momencie wybrania odpowiedniego identyfikatora, program automatycznie wypełniania te pola w pozycji faktury. Stosownym momentem do wykonania tej operacji jest chwila, w której dochodzi do zmiany wartości w polu tabeli. Występuje wtedy zdarzenie OnChange
1.Wyświetl listę pól obiektu Table2 z formularza FakturaForm
2.Zaznacz pole IDTowUslug
3.W Object Inspectorze przejdź do zakładki Events i dwukrotnie kliknij w polu zdarzenia OnChange
4.Uzupełnij kod procedury obsługi zdarzenia, aby wyglądała następująco:
procedure TFakturaForm.Table2IDTowUslugChange (Sender : TField);
begin
{Pobranie danych towary/usługi z tabeli TowUslug} Table2Nazwa.Value :=
TowarDModule.Table1Nazwa.Value;
Table2KWiU.Value : = TowarDModule.Table1KWiU.Value;
Table2J_miary.Value := TowarDModule.Table1J_miary.Value;
Table2Cena_jedn.Value; := TowarDModule.Table1Cena_sprz.Value;
Table2St_VAT.Value := TowarDModule.Table1St_VAT.Value;
end;
5.Kliknij przycisk Save All, aby zapisać wprowadzone zmiany
6.Skompiluj i uruchom aplikację
7.Przejdź do formularza Faktura i wybierz dowolną wartość na liście odnośników w kolumnie Symbl, siatki. Pola : Nazwa, KWiU, J_Miary, Cena_jedn i St_VAT w wybranym rekordzie pozycji faktury zostaną automatycznie wypełnione
8.Wpisz dowolną wartość w polu Ilosc. Nastąpi przeliczenie kwoty podatku VAT oraz wartości pozycji faktury.
Stosują takie rozwiązania, pozornie dopuszczamy do wystąpienia nadmiarowości - po co w fakturze powtarzać jednostkę miary, stawkę VAT itp., skoro informacje te są już zapisane w innej tabeli? Jednak zaproponowany mechanizm ma bardzo ważną zaletę - podczas sporządzania faktury użytkownik może skorzystać z listy towarów/usług i automatycznie wprowadzić wszystkie niezbędne informacje lub używać pola odnośnika i wypełnić pola pozycji faktury ręcznie. Cecha ta jest szczególnie przydatna w firmach o charakterze usługowym - nie trzeba wtedy definiować odpowiednich rekordów tabeli TowUslug, gdy jest wystawiana faktura na jaką jednorazową usługę

Wyznaczanie podsumowania za pomocą zapytania sparametryzowanego Można śmiało powiedzieć ,że bez zapytań SQL trudno napisać dobrą aplikację bazodanową. Zapytania są przydatne w wielu sytuacjach - do pobierania danych wyświetlanych w kontrolkach formularza, podczas generowania raportów, do masowych modyfikacji danych oraz do obliczeń agregujących. Język zapytań SQL jest bardzo rozbudowanym mechanizmem W tworzonej aplikacji również w kliku miejscach zastosujemy zapytania. Po pierwsze użyjemy ich do wyznaczania sumy wszystkich pól Wartosc dla bieżącej faktury

wyświetlanej w formularzu FakturaForm. Przykładowe zapytania realizujące to zadania mogłoby mieć postać :

```
SELECT SUM (Wartosc)
FROM TranSzcz
WHERE IDTranNagl = Identyfikator_faktury
```

Niestety, nasze zadanie jest utrudnione , ponieważ w tabeli TranSzcz nie ma pola Wartosc. Przypomnijmy, zrezygnowaliśmy z niego, gdyż chcieliśmy uniknąć przechowywania nadmiarowych danych, Kwotę tę musimy więc wyznaczyć ponownie na podstawi pól Cena_jedn, Ilosc, St_VAT. Koejne przybliżenie instrukcji SELECT będzie miało postać:

```
SELECT SUM (cena_jedn * ilość * (1 + st_vat/100) AS SumaFakt
FROM TranSzcz
WHERE IDTranNagl = Identyfikator_faktury
```

W instrukcji tej pojawia się nowy element - klauzula AS. Za jej pomocą nadajemy nazwę plu z wynikiem - w zestawie wynikowym będzie się ono nazywał SumaFakt. Gdybyśmy nie użyli klauzuli AS SumaFakt, pole to uzyskałoby dosyć kłopotliwą nazwę SUM OF cena_jedn * ilość

Operacja załączenia w zapytaniu

Wyrażenie arytmetyczne cena_jedn * ilość * (1+st_vat/100(byłoby zupełnie prawidłowo, gdyby nie to ,że pole St_VAT jest typu tekstowego. Aby przekonwertować znaki na liczby, można zastosować funkcje SQL CAST, jednak nie zadziała ona prawidłowo w przypadku stawki zwolnionej (jak pamiętamy jest ona oznaczona literami "zw"). Na szczęście, przewidując te trudności zdefiniowaliśmy tabeę St_vat w której poszczególne opisy stawek ("0", "7" ,"23", "zw") zostały skojarzone z odpowiednimi wartościami procentowymi .By móc odwołać się do tych wartości, konieczne jest powiązanie tabeli TranSzczz z tabelą St_VAT. Odpowiednia kwerenda będzie miała postać:

```
SELECT
SUM (ts.cena_jedn * ts.ilosc * (1 + sv.procent/100) AS SumaFakt
FROM
TranSzcz ts, St_VAT sv
WHERE
ts.st_vat = sv.IDSt_VAT AND
ts.IDTranNagl = Identyfikator_faktury
```

Zwróć uwagę ,że w powyższym zapytaniu dodatkowo zastosowano tzw. aliasy tabel ('ts', 'sv'). Dzięki nim w zapytaniach operujących na wielu tabelach nie musimy kwalifikować nazw poszczególnych pól pełnymi nazwami tabel ,lecz tylko ich skrótami. Kwalifikowanie nazw pól (poprzedzanie nawy pola nazwą tabeli, z której to pole pochodzi) jest konieczne, gdy w tabelach występujących w zapytaniu znajdują się pola o takich samych nazwach. W powyższym przykładzie problem ten nie występuje, jednak zastosowanie kwalifikowania znacznie poprawia czytelność zapytania (między innymi nie musimy się domyślać z których tabel pochodzą poszczególne pola)

Zapytanie z parametrem

Do uzyskania pełnej formuły zapytania brakuje tyko poprawnego zdefiniowania Identyfikator_faktury. W zależności od wyświetlanego rekordu identyfikatora ten będzie za każdym razem inny. Co więcej, jeżeli użytkownik będzie się przemieszczał między dokumentami sprzedaży w formularzu FakturaForm, zapytanie powinno być automatycznie aktualizowane by uzyskać sumę bieżącej faktury. Oznacza to ,że w jakiś sposób musimy powiązać zapytanie z źródłem danych reprezentującym tabelę TranNagl

1.Na palecie BDE odszukaj komponent TQuery i umieść go na formularzu FakturaForm. Na formularzu pojawi się nowy komponent Query1

2.Przejdź do Object Inspectora , odzszukaj właściwość DatabaseName i wybierz z listy wartość BAZAFAKT

3.Przejdź do właściwości DataSource i wpisz wartość DataSource1. W ten sposób zdefiniowaliśmy nadrzędne źródło danych, względem którego będzie synchronizowane to zapytanie. Jak pamiętamy , komponent DataSource1 jest powiązany z komponentem Table1 w module danych FaktDModule, a więc faktycznie reprezentuje tabelę TranNagl
4.Przejdź do właściwości SQL i kliknij widoczny z prawej strony przycisk wielokropka. Na ekranie pojawi się okno dialogowe String List Editor,w którym należy wpisać poniższy tekst zapytania:

```
SELECT
SUM (ts.cena_jedn * ts.ilosc * (1 + sv.procent/100) AS SumaFakt
FROM
TranSzcz ts, St_VAT sv
WHERE
ts.st_vat = sv.IDSt_VAT AND
ts.IDTranNagl = IDTranNagl
```

Jedyną nowością w przedstawionym zapytaniu jest element :IDTranNagl. Dwukropek poprzedzający identyfikator oznacza ,że jest on parametrem. Jego nazwa jest nieprzypadkowa - brzmi dokładnie tak samo, jak nazwa pola w tabeli TranNagl. Ponieważ wcześniej , poprzez właściwość DataSource, powiązaliśmy komponent Query1 ze źródłem danych DataSource1, aplikacja będzie wiedziała ,że w miejsce tego parametru powinna być zawsze podstawiana bieżąca wartość pola IDTranNagl z tabeli TranNagl
5.Przejdź do właściwości Active i ustaw jąna True

Wstawianie kontrolki wyświetlającej podsumowanie
Oczywiście samo uaktywnienie zapytania nie wystarczy. Musimy jeszcze zdefiniować kontrolkę ,w której będzie wyświetlany jego wynik oraz komponent typu TDataSOurce, łączący kontrolkę z zestawem danych Query1
1.Z palety DataAccess pobierz komponent TDataSource i umieść go na formularzu obok komponentu Query1, który wstawiłeś w poprzednim podrozdziale. Na formularzu pojawi się komponent DataSource5
2.Gdyu zaznaczony jest nowy komponent , w Object Inspectorze przejdź do właściwości DataSet i wybierz z listy wartość Query1. Komponent DataSource5 zostanie powiązany z zestawem danych Query1
3.Z palety Standard pobierz kontrolkę TPanel i umieść ją w dowolnej części formularza. Na ekranie pojawi się komponent Panel4
4.W Object Inspectorze odszukaj właściwość Aign i zmień jej wartość na alBottom. Panel zostanie dosunięty do dolnej krawędzi formularza, jednocześnie zajmując całą jego szerokość
5.Usuń tekst z pola właściwości Caption. Zniknie nazwa wyświetlana na panelu
6.Zaznacza obiekt Panel4, a następnie umieść na nim kontrolki TLabel (z palety Standard) i TDEdit (z palety Data Controls). Zwróć uwagę ,że komponenty Query1 i DataSource5 znalazły się w pobliżu pola, w którym będzie wyświetlana informacja uzyskiwana przy ich użyciu. Choć z punktu widzenia działania aplikacji nie ma to żadnego znaczenia, jest to jednak bardzo praktyczny sposób postępowania. Gdy za jakiś czas postanowisz wprowadzić dalsze modyfikacje w tym formularzu, szybko odnajdziesz źródło danych związane z reprezentującym je komponentem wizualnym.
7.Gdy zaznaczony jest nowy komponent etykiety, przejdź do Object Inspectora i zmień jego właściwość Caption nowej etykiety na RAZEM :
8.Zaznacza kontrolkę DBEdit1 i zmień jej właściwości zgodnie z poniższym:
Color : clSkBlues
DataField : SumaFakt
DataSource : DataSource5
Enabled : False
Name : EditSumaFakt
Właściwość DataSource i DataField wskazują odpowiedni źródło oraz pole danych z zestawu

wynikowego zapytania. Pole to będzie tylko do odczytu., co najprościej można uzyskać, ustawiając jego właściwość Enabled na False (w ten sposób jednocześnie uniemożliwiamy ustawienie w nim kursora) .Zmieniając kolor tła dostosowujemy się do konwencji, jaką zastosowaliśmy w przypadku pól opisujących kontrahenta

Odświeżanie zapytania sparametryzowanego

Ostatnią czynnością ,jaką musimy wykonać , jest wymuszenie odświeżania wyniku zapytania po każdej modyfikacji danych. By lepiej zrozumieć sens tej operacji, przeanalizujemy, w jaki sposób użytkownik wypełnia fakturę. Zwykle najpierw wprowadzana jest nazwa towaru, następnie jego ilość, cena jednostkowa i stawka podatku VAT. Po podaniu tych trzech ostatnich składników następuje zaktualizowanie wartości w polach obliczeniowych VAT i Wartosc. Gdy użytkownik skończy edytować rekord pozycji faktury (zatwierdzi go lub przejdzie do następnego rekordu), powinno nastąpić automatyczne zaktualizowanie sumy faktury, aby w każdej chwili było wiadomo, jaka jest jej aktualna wartość. Można by więc stwierdzić ,że wystarczy napisać procedurę obsługi zdarzenia OnPost nie zachodzi, a mimo to suma u dołu formularza powinna być uaktualniana. W tym momencie najlepszym wyjściem będzie obsłużenie zdarzenia OnDataChange komponentu DataSource2

1.Zaznacz komponent DataSource2

2.W Object Inspectorze przejdź do karty Events i dwukrotnie kliknij w polu obok zdarzenia OnDataChange

3.Uzupełnij powłokę procedury zgodnie z poniższym wydrukiem:

```
procedure TFakturaForm.DataSource2DataChange(Sender : TObject; Field : TField);
begin
if Query1.Active then begin
Query1.Close;
Query1.Open;
end;
end;
```

Procedura ta będzie uruchamiana po każdej zmianie danych w rekordzie wskazywanym przez component DataSource2, niezależnie od tego czy nastąpiła zmiana zawartości bieżącego rekordu, czy też przejście kursora do innego rekordu

4.Kliknij przycisk Save, aby zapisać wprowadzone zmiany

5.Skompiluj i uruchom aplikację

Ustawianie siatki danych w tryb tylko do odczytu

O ile edycja danych kontrahentów i towarów w siatce danych typu TDBGrid czasami jest wygodniejsza, to w przypadku faktur użytkownik powinien raczej używać formularza FakturaForm. Wynika to z prostego faktu ,że na liście faktur brak wielu istotnych informacji, takich jak wartość faktury czy wykaz jej pozycji. Lista ta powinna więc służyć jedynie do wyszukiwania istniejących faktur i umożliwiać łatwe przechodzenie do ich edycji lub do tworzenia nowych. Aby uniemożliwić edycję danych w siatce , należy ustawić jej właściwość ReadOnly na True

1.Kliknij przycisk View Form, a następnie wyświetl formularz FaktLForm

2.Zaznacz komponent DBGrid

3.W Object Inspectorze ustaw właściwość ReadOnly na True

Dodatkowo ,aby uniknąć efektu zaznaczania zawartości komórki, można ustawić właściwość Options.dgEditing na False. Jeżeli chcesz, aby zaznaczenie obejmowało cały wiersz, ustaw właściwość Options.dgRowSelect na True

Niestandardowa obsługa przycisków nawogatora

Od tego momentu w siatce danych tego formularza nie będzie można modyfikować rekordów. Jeżeli

użytkownik zechce edytować rekord faktury, będzie musiał go odszukać , a następnie kliknąć dwukrotnie (lub jednokrotnie przycisk EdytujRekord). Analogicznie wstawienie nowego rekordu wymagałoby kliknięcie przycisku +, a następnie kliknięcie dwukrotnie nowego wiersza, który pojawi się w siatce. Taki sposób działania z pewnością jest mało ergonomiczny warto więc go zmienić, definiując odpowiednią procedurę zdarzenia dla paska nawigatora:
1.Dwukrotnie kliknij pasek nawigatora DBNavigator
2.Uzupełnij procedurę zdarzenia, aby uzyskała postać:

```
procedure TFaktLForm.DBNavigatorClick (Sender : TObject ; Button : TNavigateBtn);
begin
inherited
case Button of
nbInsert : WyswietlFakt;
nbEdit : WyswietlFakt;
end;
end;
```

Parametr Button przekazywany do procedury DBNavigatorClick wskazuje, który przycisk został uaktywniony. Parametr ten jest typu TNavigateButton. Kod procedury jest bardzo prosty. Jeżeli kliknięto przycisk wstawiania (nbInsert) lub edycji (nbEdit) , następuje wywołanie procedury wyświetlającej formularz FakturaForm. Modyfikacja działania przycisków nawigatora nie powoduj przesłonięcia ich głównych funkcji - jednocześnie z przejściem do formularza faktury, zestaw danych z nagłówkami transakcji znajdzie się w trybie wstawiania (dsInsert) lub edycji (dsEdit). Często rezygnujemy z niektórych przycisków nawigatora. Na przykład, aby zabezpieczyć się przed przypadkowym usunięciem faktury z listy faktur, możemy wyłączyć przycisk usuwania na pasku nawigatora. W tym celu należy zaznaczyć komponent nawigatora . W tym celu należy zaznaczyć komponent nawigatora i w Object Inspectorze ustawić odpowiedni element jego właściwości VisibleButtons na False (W tym przypadku wartość False należy ustawić dla elementu nbDelete)
3.Kliknij przycisk Save All, aby zapisać wprowadzone zmiany
4.Skompiluj i uruchom aplikację

Podsumowanie

W tej części wprowadziłeś końcowe udoskonalenia formularza do wystawiania faktur. Oczywiście można by tutaj wymyślić szereg innych udogodnień i zabezpieczeń, które znacznie zwiększyłyby komfort edycji dokumentów, jednak na potrzeby naszej aplikacji możliwości formularza FakturaForm są już zupełnie wystarczające. Zwróć uwagę ,że prawie wszystkie zagadnienia , które do tej pory omawialiśmy były związane z wprowadzaniem danych. Dobrem mechanizmy wprowadzania danych to dopiero połowa sukcesu aplikacji bazodanowej. Równie ważne są odpowiednie narzędzia umożliwiające analizę i drukowanie informacji pieczołowicie gromadzonych w systemie bazy danych.

RAPORTY

Trudno sobie wyobrazić aplikację bazodanową bez możliwości wydruku zawartych w niej danych. Uwaga ta dotyczy zwłaszcza aplikacji o charakterze finansowym, w przypadku których przepisy wymuszają generowanie dokumentów w "trwałej postaci", czyli na papierze. Jednym z takich dokumentów jest oczywiście faktura potwierdzająca operację sprzedaży. Być może w przyszłości całkowicie zniknie potrzeba przelewania informacji na papier (jakby nie patrzeć, jest to już działanie nieco "atawistyczne"), jednak póki co, zdolność do generowania wydruków jest jedną z nieodłącznych cech aplikacji bazodanowych.

Sekcje raportu

Każdy raport składa się z sekcji odpowiedzialnych za poszczególne elementy wydruku. W typowym raporcie mogą się pojawić następujące sekcje (wszystkie są obiektami typu TQRBand):

• tytył (właściwość BandType sekcji raportu ma wartość rbTitle)
• nagłówek strony (rbPageHeader)
• szczegóły (rbDetail)
• stopka strony (rbPageFooter)
• podsumowanie (rbSummary)
Wyjaśnijmy w kilku słowach znaczenie poszczególnych sekcji, Tytuł pojawia się tylko na pierwszej stronie raportu, nagłówki i stopki występują odpowiednio na początku i końcu każdej strony raportu (chyba ,że właściwości raportu Options.FirstPageHeader i Options.LastPageFooter zostaną ustawione na False) . W sekcji nagłówka można na przykład powtatrzać tytuł wydruku. Z kolei sekcja stopki jest dobrym miejscem na umieszczenie numerów stron lub sum cząstkowych. Sekcja szczegółów przedstawia dane poszczególnych rekordów tabeli lub zapytania - pola każdego rekordu są drukowane w oddzielnej sekcji szczegółów. Podsumowanie pojawia się tylko raz na końcu wydruku. Oczywiście każda z tych sekcji jest opcjonaną (na przykład można zdefiniować bez stopek i nagłówków). Oprócz tych podstawowych sekcji, Delphi oferuje również inne :
• sekcja podszczegółów (rbSubDetail) - opcja wykorzystywana tylko w przypadku sekcji typu TQRSubDetail; przydatna w raportach typu master/detail (tzn. występują w nim podraporty); zastosujemy ją podczas definiowania wydruk faktury
• sekcja podrzędna (rbChild) - zarezerwowana dla komponentów typu TQRChild; sekcja tego typu przylega do innej , nadrzędnej sekcji (wskazuje ją właściwość LinkBand); jest przydatna , jeżeli chcemy przemieszczać w dół niektóre kontrolki, w zależności od pionowego rozwinięcia innych
• sekcja nagłówki / stopki grupy (rbGroupHeader / rbGroupFooter) - stosowana w połączeniu z komponentami TQRGroup lub TQRSubDetai; dzięki niej można rozdzielać różne grupy rekordów spełniających pewne kryteria
• sekcja nagłówk kolumny (rbColumnHeader) - stosowana głównie w raportach wielokolumnowych; pojawia się na początku każdej kolumny wydruku

Tworzenie raportu za pomocą kreatora

W programie Delphi jest dostępny kreator raportów. Pozwala on tworzyć proste raporty w układzie tabelarycznym. Skorzystanie z pomocy tego kreatora jest dobrym punktem wyjścia do opracowania nowego raportu. Za pomocą poniższych czynności wygenerujesz prosty raport w układzie tabelarycznym
1.Uruchom Delphi i otwórz projekt Faktury.dpr
2.W menu File wybierz polecenie New, Other i przejdź do zakładki Business
3.Dwukrotnie kliknij ikonę QuickReport Wizard. Nastąpi uruchomienie kreatora raportów
4.Na liście Select Report wybierz opcję List report a następnie kliknij przycisk Start Wizard. Pojawi się okienko, w którym możesz podać nazwę użytkownika oraz hasło dostępu do bazy danych, która nie jest zabezpieczona hasłem, a więc te opcje są niepotrzebne
5.Pozostaw puste pola z nazwą oraz hasłem użytkownika i kliknij przycisk OK. Pojawi się pierwsze okno kreatora New Report Wizard
6.W polu listy Alias or Directory wybierz alias BAZAFAKT
7.W polu listy Table name wybierz tabelę Kontrah
8.Korzystając z przycisku > umieszczonego między listami Available fields i Selected fields, przenieś pola : IDKontrah, Nazwa, Miasto, Ulica na drugą z wymienionych list
9.Kliknij przycisk Finish. Zostanie utworzony nowy formularz raportu. Najważniejszym elementem wygenerowanym przez kreator jest sam komponent raportu, przedstawia on biały, wyskalowany prostokąt, symbolizujący kartkę papieru, na której docelowo pojawia się wydruk. Niebieska przerywana linia wskazuje bieżące ustawienie marginesów. Ponadto kreator umieścił na formularzu komponent Table1, który zostanie wykorzystany jako zestaw danych do wydruku (Jeżeli sprawdzisz właściwość DataSet komponentu QuickReport, zauważysz ,że ma ona właściwość Table1)
10.Gdy zaznaczony jest komponent formularza, na którym umieszczono raport, przejdź do Object

Inspectora i zmień właściwość Name na KontrLQReport
11.Zapisz utworzony raport jako plik KontrLQ

Sekcje raportu wygenerowane przez kreator

W raporcie wygenerowanym przed chwilą przez kreator występują trzy sekcje :

- sekcja nagłówków kolumn (ColumnHeaderBand1)
- sekcja szczegółów (DetailBand1)
- sekcja stopki (PageFooterBand1)

W sekcji nagłówków kolumn znajduje się etykieta z nazwami poszczególnych kolumn tabeli. Etykiety w raporcie są reprezentowane za pomocą komponentów TQRLabel. Sekcja nagłówków pojawia się na początku każdej strony lub na początku każdej kolumny (jeżeli raport jest wielokolumnowy). W sekcji szczegółów są umieszczone kontrolki typu TQRExpr. Służą one przede wszystkim do generowania wyrażeń opartych na innych polach tabeli lub zapytania. Wynik przypomina wartość pola obliczeniowego gdyż zmienia się w zależności od bieżącej wartości rekordu. Wartość wyrażenia wyświetlanego przez ten komponent jest zdefiniowana we właściwości Expression. Wyrażenia zastosowane przez kreator są bardzo proste - w odpowiednich miejscach podstawił on po prostu nazwy pól tabeli. Jeżeli samodzielnie tworzymy raport, w którym bezpośrednio korzystamy z wartości pól (nie przetwarzamy ich) zamiast pól typu TQRExpr wystarczy zastosować pola typu TQRDBText. Sekcja stopki strony zawiera tylko jedno pole typu TQRExpr, wyświetlające numer strony. Właściwość Expression tej kontrolki ma wartość : 'Page ' + PageNumber. Element 'Page ' jest stałym tekstem wyświetlanym w wyrażeniu natomiast PageNumber jest funkcją zwracającą bieżący numer strony wydruku.

Zmiana głównych ustawień raportu

Zmiana ustawień raprotu może odbywać się w czasie wykonania (poprzez odwoływanie się do właściwości obiektu typu TQuickRep) oraz w fazie projektowej. Oto zasady modyfikowania właściwości raportu w czasie projektowania:

1.Prawym przyciskiem myszy kliknij komponent QuickRep1 (Kliknij poza obszarem zajmowanym przez sekcje - najlepiej na marginesie raportu)
2.Z menu podręcznego wybierz polecenie Report Setting. Pojawi się okno dialogowe o takiej samej nazwie. W oknie tym można modyfikować rozmiar i układ strony (opcje z grupy Paper size), marginesy (grupa Margins), czcionki i jednostki miary (grupa Other) , styl i kolor obramowania strony (grupa Page frame) oraz długość i rodzaj sekcji występujących w raporcie (grupa Bands)
3.W sekcji Paper size rozwiń listę z prawej strony i wybierz opcję Portrait. Układ strony raportu zostanie zmieniony z poziomego na pionowy

Spróbujmy za pomocą okna dialogowego Report Settings nieco wzbogacić raport wygenerowany przez kreator, dodając do niego sekcję tytułu

1.Zaznacz pole wyboru Title. W sąsiednim polu tekstowym wpisz 15.W ten sposób zadeklarowaliśmy utworzenie sekcji tytułu o wysokości 15 milimetrów
2Kliknij przycisk OK. Okno dialogowe zostanie zamknięte ,a na początku raportu pojawi się nowa sekcja Title
3.Na palecie QReport wybierz kontrolkę TQRLabel i umieść ją w sekcji Title
4.Gdy zaznaczona jest nowo wstawiona kontrolka, przejdź do Object Inspectora i we właściwości Caption wpisz Lista kontrahentów.
5.Za pomocą właściwości Font.Size zmień rozmiar czcionki na 18

Stosowanie komponentów QRDBText

Komponenty TQRDBText służą do prezentowania danych tekstowych związanych z określonym polem zestawu danych. Umieszczenie takiego komponentu w sekcji szczegółów i przypisanie mu

odpowiedniego pola tabeli lub zapytania pozwala na uzyskanie raportu w którym w kolejnych sekcjach znajdują się wszystkie wartości tego pola pobrane z poszczególnych rekordów. Jeżeli przyjrzymy się raportowi utworzonemu przez kreator zauważymy ,że brak na nim pól Miasto i Ulica, chociaż zadeklarowaliśmy je podczas definiowania raportu. Stało się tak , ponieważ na zastosowanym formacie papieru mieszczą się w całości tylko dwa pierwsze pola : IDKontrahenta i Nazwa . Spróbujmy teraz poprawić raport

1.Zmniejsz szerokość komponentu TQRDBExpr wyświetlającego nazwę kontrahenta oraz towarzyszącej etykiety, aby kończyły się mniej więcej w połowie szerokości raportu
2.W sekcji nagłówka kolumny wstaw dwie etykiety TQRLabel
3.W sekcji szczegółów wstaw dwa komponenty TQRDBRext
4.Zaznacz komponenty QRDBText1 i QRDBText2 , a następnie ustaw ich właściwość DataSet na Table1. W ten sposób nowe komponenty zostaną powiązane z zestawem danych raportu
5.Gdy wspomniane komponenty są wciąż zaznaczone, zmodyfikuj ich właściwość AutoSize na False. Dzięki temu można będzie ręcznie ustawić maksymalną szerokość tych komponentów. Jeżeli właściwość AutoSize ma wartość True, komponent dostosowuje swoją szerokość do aktualnie prezentowanej zawartości, co grozi nachodzeniem na siebie tekstu
6.Zaznacz pierwszy z komponentów TQRDBText, a następnie rozciągnij go ,aby wykorzystał cały dostępny dla niego obszar. Tę samą operacje wykonaj dla drugiego komponentu.
7.Ustaw właściwość DataField komponentów QRDBText1 i QRDBText2 odpowiednio na Miasto i Ulica. Komponenty zostały powiązane z odpowiednimi polami zestawu danych Table1
8.Zmodyfikuj właściwość Caption komponentów QRLabel4 i QRLabel5 odpowiednio na Miasto i Ulica. W wielu przypadkach może się okazać, że szerokość poszczególnych pól tekstowych jest zbyt mała, aby pomieścić reprezentowane przez nie pola bazy danych . W rezultacie na wydruku uzyskalibyśmy obcięte nazwy lub adresy kontrahentów. Najwygodniejszym rozwiązaniem w takiej sytuacji jest ustawienie właściwości AutoStrech komponentu TQRDBText lub TQRExpr na wartość True, to gdy wyświetlana wartość zajmuje więcej miejsca , niż wynika to z szerokości pola tekstowego , nastąpi wydłużenie tego pola i przeniesienie pozostałego tekstu do następnego wiersza. Oczywiście może to spowodować wydłużenie całej sekcji.
9.Zaznacza komponenty prezentujące wartości pól Nazwa, Miasto i Ulica a następnie zmodyfikuj ich właściwość AutoStrech na True
10.Kliknij przycisk Save aby zapisać wprowadzone zmiany

Wyświetlanie podglądu wydruku

Uzyskanie podglądu wydruku raportu jest bardzo proste. Wystarczy klikną komponent raportu (w wolnym miejscu gdzie nie zostały umieszczone jakieś kontrolki oraz sekcje) i wybrać polecenie Preview. Na ekranie pojawi się podgląd wydruku raportu. W kodzie programu sprawa jest równie prosta. Analogiczne okienko podglądu będzie się pojawiać , gdy wywołamy metodę Preview komponentu TQuickRep. Wydruk raportu uzyskujemy poprzez wywołanie metody Print. Podgląd wydruku nie pojawi się jeżeli źródło danych (tabela lub zapytanie) raportu będzie nieaktywne (zamknięte) Nie wolno również zapominać o przypisaniu właściwości DataSet komponentu TQuickrep do odpowiedniego zestawu danych

Wywoływanie raportu z aplikacji

Podobnie jak w przypadku formularzy, które dołączaliśmy do aplikacji, również dla raportu musimy wskazać miejsce, z którego będzie on uaktywniany. Starając się realizować zasadę intuincyjności, najlepszą lokalizacją do wywołania raportu z listą kontrahentów jest na pewno formularz z listą kontrahentów. W tym celu wystarczy umieścić na formularzu przycisk , odpowiednio go nazwać, a następnie zdefiniować procedurę zdarzenia OnClick odpowiedzialną za wykonanie raportu. Zanim jednak zabierzemy się do modyfikowania formularza KontrahLForm, zwróćmy uwagę, że podobne wydruki prawdopodobnie będziemy chcieli uzyskiwać dla list towarów/usług oraz dla list transakcji.

Wniosek z tego jest jednoznaczny - przycisk umożliwiający wydruk raportu powinien zostać umieszczony na formularzu ListaForm, dzięki czemu zostanie on odziedziczony przez wszystkie formularze pochodne
1.Kliknij przycisk ViewForm (Shift+ F12) i wyświetl formualrz ListaForm
2.Na palecie Additional zaznacz przycisk TBitBtn i umieść go obok przycisku EdytujRekordBitBtn
3.Gdy nowy przycisk jest zaznaczony, przejdź do Object Inspectora i zmień jego właściwość Name na DrukujBitBtn oraz właściwość Caption na &Drukuj
4.Kliknij Save aby zapisać wprowadzone zmiany. Konkretny raport , który będzie drukowany, zależy już od typu wyświetlanej listy. Oznacza to ,że odpowiednie wywołanie musimy zdefiniować już na poziomie formularza pochodnego
5.Kliknij przycisk View Form i wyświetl formularz KontrahLForm. Jak widać ,dzięki mechanizmowi dziedziczenia znajduje się na nim przycisk DrukujBitBtn
6.Dwukrotnie kliknij przycisk DrukujBitBtn, a następnie uzupełnij kod procedury obsługi zdarzenia aby uzyskała postać :

```
procedure TKontrahLForm.DrukujBitBtnClick (Sender : TObject);
begin
inherited;
KontrLQReport.QuickRep1.Preview;
end;
```

Ponieważ aplikacja jest w dosyć wczesnej fazie powstawania, nie warto marnować papieru na testowanie wydruków - w zupełności wystarczy skorzystanie w podglądu
7.W klauzuli uses sekcji implementation dołącz odwołanie do modułu KontrLQ

```
uses
Kontrah, KontrahD, KontrLQ;
```

8.Kliknij przycis Save ,aby zapisać zmiany
9.Skompiluj i uruchom aplikację. Przejdź do formularza z listą kontrahentów a następnie kliknij przycisk Drukuj
W naszym przykładzie zastosowaliśmy najprostsze rozwiązanie , którym kliknięcie przycisku Drukuj powoduje natychmiastowe wygenerowanie raportu. W praktyce oczekiwania użytkownika są z reguły o wiele większe. Wydruki związane z listami kontrahentów służą przede wszystkim do różnego rodzaju analiz. Dlatego warto wśród nich umieścić na przykład zestawienia obrotów z kontrahentami w wybranym okresie, zestawienia związane z należnościami i zobowiązaniami, raporty porządkujące kontrahentów według wybranych kryteriów. Oznacza to, że wybranie funkcji wydruku powinno powodować wyświetlenie kolejnego formularza, na którym użytkownik będzie mógł określić rodzaj sporządzanego raportu oraz zdefiniować jego dodatkowe parametry. Parametry te są zwykle przekazywane do odpowiedniego zapytania, którego rezultat jest wykorzystywany jako zestaw danych dla raportu

Wybór drukarki i parametrów wydruku

Jeżeli zastosowaną w powyżej metodę Preview zastąpimy metodą Print, wszystkie strony raportu zostaną skierowane na domyślną drukarkę . Jednak użytkownik nie zawsze chce drukować całość raportu. Czasami chce skierować wydruk na inną drukarkę lub sprecyzować liczbę jego egzemplarzy. Odpowiednie opcje wydruku możemy ustawić w utworzonym samodzielnie formularzu, a następnie przypisywać je właściwościom komponentu QuickReo. Jednak o wiele prostsze rozwiązanie polega na użyciu standardowego komponentu PrintDialog. Wyświetla on standardowe okno dialogowe wydruku ,zdefiniowane w systemie Windows
1.Wróć do formualrza ListaForm
2.Otwórz paletę Dialogs, zaznacz na niej komponent TPrintDialog i umieść go na formularzu, najlepiej w pobliżu przycisku DrukujBitBtn
3.Przejdź do Object Inspectora i ustaw właściwość Options.poPageNums naowego komponentu na True

4.Odszukaj właściwość MaxPage i wpisz 9999. Dzięki dwóm ostatnim czynnościom w oknie dialogowym Drukuj będą dostępne przyciski umożliwiające wydruk wybranych stron. Jeśli chcesz udostępnić funkcję wydruku do pliku, uaktywnij właściwości Options.poPrintToFile
5.W sekcji interfejsu modułu Lista umieść nagłówek funkcji:
function UstawWydruk (var Raport : TQuickRep) :boolean;
6.W sekcji implementacji umieść definicję funkcji:

```
function TListForm.UstawWydruk (var Raport : TQuickRep) : boolean;
begin
result : PrintDialog1.Execute;
if result then
with Raport.PrinterSettings do begin
FirstPage := PrintDialog1.FromPage;
LastPage := PrintDialog1.ToPage;
Copies :=PrintDialog1.Copies;
PrinterIndex := Printers.Printer.PrinterIndex;
end
end;
```

Zadaniem utworzonej funkcji będzie ustawianie właściwości wydruku zgodnie z wartościami wybranymi w oknie dialogowym Drukuj. Do funkcji jako parametr przekazywany jest raport. Funkcja modyfikuje jego ustawienia, a następnie zwraca wartość boole'owską (True - raport będzie drukowany, False - anulowanie raportu). Okno dialogowe Drukuj jest wyświetlane za pomocą metody Execute komponentu PrintDialog1. Okno jest modalne .Jeżeli użytkownik kliknie przycisk OK, metoda Execute zwraca wartość True, w przeciwnym razie - wartość False. W analogiczny sposób, przypisując wynik uzyskany po wykonaniu metody Execute wynikowi (result) zwracanemu przez funkcję UstawWydruk, będziemy mogli decydować o rozpoczęciu wydruku. Jeżeli ustawienia w oknie dialogowym Drukuj zostały zatwierdzone (result = True), następuje ich przepisanie do ustawień samego raportu Operacje te są wykonywane w bloku with □ do. Pierwsze trzy instrukcje przypisania definiują pierwszą i ostatnią stronę wydruku oraz liczbę jego egzemplarzy. Ostatnia instrukcja przypisania umożliwia wybór drukarki, na którą zostanie skierowany wydruk. Informację tę uzyskujemy poprzez właściwość PrinterIndex obiektu Printer. Parametr funkcji jest typu TQuickRep zdefiniowanego w module QReport. Z kolei definicja obiektu Printer znajduje się w module Printers. Oznacza to ,że kod edytowanego formularza musimy uzupełnić o odpowiednie odwołania
7.W klauzuli uses sekcji interface wstaw wywołanie modułów QReport i Printers

```
uses
Windows, Messages. Classes, SysUtils, Graphics, Controls, StdCtrls, Forms, Dialogs, DBCtrls, DB,
DBGrids, Grids. ExtCtrls, Buttonsl QuickRpt, Printers;
```

8.Kliknij przycisk Save aby zapisać wprowadzone zmiany
9.Skompiluj i uruchom program. Przejdź do formularza z listą kontrahentów i kliknij przycisk Drukuj. Na ekranie pojawi się okno dialogowe Drukuj

Podsumowanie

W tej części poznałeś podstawowe zasady generowania raportów bazodanowych w środowisku Delphi. Wykorzystałeś w tym celu komponenty z palety QReport. Wiesz już jak utworzyć prosty raport za pomocą kreatora i jak wywołać go z poziomu aplikacji. Poznałeś metodę PrintPreview, umożliwiającą wygenerowanie podglądu wydruku oraz zapoznałeś sięz komponentem PrintDialog, służącym do zdefiniowania właściwości wydruku