Zaawansowane mechanizmy raportów

Wydruki generowane przez bazę danych generalnie określane są jako raporty. Nazwy tej będziemy więc używali również do tworzenia formularza odpowiedzialnego za wydruk faktury, choć początkowo może to nie odpowiadać naszemu potocznemu rozumieniu słowa "raport". Przygotowanie wydruku faktury jest procesem o wiele bardziej skomplikowanym od analogicznego zadania dla listy klientów, które wykonaliśmy w poprzedniej części. Możliwości kreatora raportów są zbyt skromne, aby je zrealizować. W związku z tym cały proces tworzenia wydruku faktury wykonamy ręcznie. W nagłówku każdej faktury znajdują się takie informacje, jak data i miejsce wystawienia , numer faktury, dane sprzedawcy i nabywcy, data sprzedaży, termin płatności i forma płatności. W sekcji szczegółów będą umieszczone informacje o poszczególnych towarach / usługach. Na każdą pozycję tej sekcji będą się składały takie informacje , jak nazwa towaru, symbol KWiU, cena jednostkowa, ilość ,jednostka miary, wartość netto, stawka i kwota podatku VAT oraz wartość brutto. Najważniejszym elementem stopki wydruku jest łączna kwota faktury, wyrażona cyfrowo i słownie. Nie możemy zapomnieć o miejscu na "stosowne" pieczęcie i podpisy. Do tego momentu układ raportu nie przedstawia się szczególnie skomplikowanie - mamy trzy sekcje : nagłówków, szczegółów i stopki. Wystarczy przygotować odpowiednie zapytanie, które zgromadzi potrzebne dane, a następnie we właściwych miejscach umieścić etykiety i komponenty TQRDBRText. Niestety pod sekcją szczegółów pojawia się dodatkowe zestawienie prezentujące podsumowanie faktury według stawek VAT, Każda stawka VAT ma swój oddzielny wiersz. Oznacza to ,że w tworzonym raporcie wystąpią dwie sekcje szczegółów : jedna z danymi pozycji transakcji i druga z sumą kwot dla wybranej stawki VAT. Problemu tego można uniknąć, jeśli "na sztywno" zdefiniujemy typy stawek w sekcji stopki raportu. Jednak takie rozwiązanie oznaczałoby znaczące ograniczenie skalowalności aplikacji, gdyż w przypadku pojawienia się nowej stawki VAT, trzeba by modyfikować kod formularza. W pojedynczym raporcie sekcja szczegółów może wystąpić tylko raz. Aby obejść ten problem, stosuje się zwykle raport złożony (composite report). Mechanizm, ten jest realizowany za pomocą komponentu TQRCompositeReport z palety QReport. Pewna niedogodność tego rozwiązania polega na tym ,że musimy używać dwóch (lub więcej komponentów) raportu, a efekt złożenia możemy zobaczyć dopiero na podglądzie wydruku. W naszej aplikacji zastosujemy inne, bardziej oryginalne rozwiązanie, mianowicie użyjemy dwóch komponentow TQRSubDetail, jednocześnie rezygnując z sekcji szczegółów

Tworzenie nowego raportu od podstaw

Pusty raport można utworzyć na dwa sposoby:
• korzystając z opcji New Report w zakładce New w konie dialogowym NewItems
• umieszczając na formularzu komponent TQuickRep w palety QReport
W pierwszym przypadku tworzony raport nie jest związany z żadnym formularzem i istnieje jako samodzielny obiekt dostępny z poziomu aplikacji. W drugim - raport jest elementem formularza. Właśnie to rozwiązanie zastosujemy, tworząc nowy formularz
1. W menu File, New wybierz polecenie Form. Na ekranie pojawi się pusty formularz
2. W Object Inspectorze przejdź do właściwości Name formularza i wpisz FakturaQForm
3. Zapisz nowy formularz jako plik FakturaQ.pas
4. Przejdź do zakładki QReport i umieść na formularzu komponent TQuickRep i umieść na formularzu komponent TQuickRep
Na formularzu pojawi się szablon raportu. Przerywane linie oznaczają bieżące marginesy. Wartości widoczne wzdłuż pionowej i poziomej krawędzi raportu wskazują odległość odpowiednio do lewego i górnego brzegu kartki wydruku

Definiowanie sekcji raportu

Jak wynika z naszych rozważań z początku tej sekcji, opracowywany raport będzie się składał z następujących sekcji:

• Title - informacje związane z nagłówkiem faktury
• ColumnHeader - nagłówki pozycji faktury
• SubDetail 1 - pozycje faktury
• SubDetail 2 - sumy dla poszczególnych stawek podatku VAT
• Footer - wartość faktury , miejsce na podpisy i pieczątki
Zwróć uwagę, że w powyższym opisie pojawiła się dodatkowa sekcja nagłówka kolumny (ColumnHeader). Gdybyśmy mieli pewność (a raczej jej nie mamy), że wydruk wszystkich pozycji faktury zmieści się na jednej stronie, nagłówki poszczególnych jej pozycji moglibyśmy umieścić u dołu sekcji Title. Jednak, jak pamiętamy, sekcja ta pojawia się tylko na pierwszej stronie wydruku. Gdyby lista pozycji faktury przeszła na następną stronę, byłaby ona pozbawiona wiersza nagłówkowego. (Nie możemy w tym miejscu użyć sekcji PageHeader, ponieważ jest ona drukowana przed sekcją tytułową). Zdefiniujmy wymienione sekcje
1. Przejdź do palety QReport, odszukaj komponent TQRBand i wstaw go do raportu. Została utworzona nowa sekcja raportu. Domyślnie wstawiany komponent TQRBand jest zawsze sekcją typu Title (typ sekcji definiuje jej właściwość BandType)
2. Wstaw następny komponent TQRBand. Przejdź do Object Inspectora i zmodyfikuj jego właściwość BandType na rbColumnHeader. W raporcie jednokolumnowym sekcje tego typu są drukowane na początku strony, przed sekcją szczegółów, lecz za sekcją tytułową.
3. Odszukaj komponent TQRSubDetail i dwukrotnie wstaw go do raportu. W raporcie pojawią się dwa komponenty typu TQRSubDetail. W pierwszym z nich będą wyświetlane szczegóły faktury, natomiast w drugim - podsumowanie według stawek podatku VAT. Ponieważ nad podsumowaniem chcemy umieścić nagłówek, wstawimy dodatkową sekcję nagłówkową
4. Zaznacz obydwa komponenty TQRSubDetail, przejdź do Object Inspectora i sprawdź , czy ich właściwość Master jest ustawiona na QuickRep1. Właściwość Master określa poziom nadrzędny, któremu jest podporządkowana sekcja typu TQRSubDetail. W naszym przypadku sekcje te są powiązane bezpośrednio z samym raportem. W typowych zastosowaniach sekcje SubSetail są podporządkowane innym sekcjom
5. Z palety QReport wstaw kolejny komponent TQRBand. Pojawi się kolejna sekcja tytułowa. Tym razem jednak potrzebujemy nagłówka dla drugiego komponentu TQRSubDetail. Będzie w nim wyświetlany krótki tekst informujący o charakterze poniższego podsumowania.
6. Zaznacz komponent QRSubDetail2, przejdź do Object Inspectora i zmień właściwość HeaderBand na QRBand3 (jeżeli sekcja nagłówka ma inny numer, podaj odpowiednią wartość). Sekcja QRBand3 zostanie automatycznie przeniesiona przed sekcję QRSubDetail2. Jednocześnie jej właściwość BandType zmieni się na rbGroupHeader
7. Z palety QReport wstaw kolejny komponent TQRBand
8. Przejdź do Object Inspectora i zmień właściwość BandType nowej sekcji na rbSummary. Po zmiane typu sekcja automatycznie przejdzie na koniec raportu. Wszystkie potrzebne sekcje zostały zdefiniowane
9. Kliknij przycisk Save , aby zapisać wprowadzone zmiany
Zwróć uwagę ,że struktura tego raportu jest nietypowa - brak w nim sekcji Detail, są za to dwie sekcje typu SubDetail. Zabieg ten stosujemy ,aby uzyskać dwa niezależne zestawienia szczegółowe - jedno z poszczególnymi pozycjami faktury drugie z podsumowaniem poszczególnych stawek podatku

Definiowanie zestawów danych raportu
Zanim przystąpimy di rozmieszczania kontrolek w raporcie, musimy określić źródło (lub źródła) danych , z którymi będą powiązane te kontrolki. Źródłem danych może być tabela lub zapytanie. Ponieważ żadna z tabel nie zawiera wszystkich danych, które chcemy umieścić na wydruku, posłużymy się zapytaniem
1. Umieść w raporcie komponent TQuery z palety BDE. Wstawiony komponent otrzyma nazwę Query1
2. Przejdź do Object Inspectora i w polu właściwości DatabaseName wpisz BAZAFAKT
3.Przejdź do właściwości SQL, uruchom edytor (kliknij przycisk oznaczony wielokropkiem) i wpisz tekst zapytania:
SELECT

TN.IDTranNagl, TN.Seria, TN.Numer, TN.Dopisek, TN.Uwagi, TN.Data_wystwa,
TN.Data_trans, TN.Term_platn , TN.Forma_platn, TS.*, TS.Cena_jedn * TS.Ilosc AS Netto,
TS.Cenat_jedn * TS.Ilosc * SV.Procent / 100 AS VAT
K.Nazwa AS Nazwa_kontr, K.Miasto , K.Ulica , K.NIP, SV.Procent
FROM
TranNagl TN, TranSzcz TS, Kontrah K, St_VAT SV
WHERE
(TN.IDTranNagl = : ID)
AND (TN.IDTranNagl = TS.IDTranNagl)
AND (TN.IDKontrah = K.IDKontrah)
AND (TS.St_VAT = SV.IDSt_VAT)
Powyższe zapytanie pobiera wszystkie pola potrzebne do wypełnienia zarówno nagłówka , jak i pierwszej sekcji SubDetail, w której znajdują się pozycje faktury. Wynik zapytania zawiera dane zawiera dane z jednego rekordu tabeli TranNagl (którego identyfikator równa się parametrowi zapytania ID) oraz ze wszystkich rekordów tabeli TranSzcz o tym samym identyfikatorze (pozycje faktury). Pobierane są również informacje o o kliencie potrzebne dla wydruku oraz procenty podatku VAT, niezbędne dla wyznaczenia wartości netto, kwoty podatku oraz kwoty brutto poszczególnych pozycji faktur. , Zwróć uwagę na słowo kluczowe AS występujące z polem K.Nazwa. Służy ono do zmiany nazwy pola w zestawie wynikowym. Używamy go w zapytaniu , gdyż zwraca ono już jedno pole o takiej nazwie z tabeli TranSzcz. (Jeżeli zapytanie zwraca pola o takich samych nazwach, są one automatycznie mdyfikowane - w naszym przykładzie, gdybyśmy nie zmienili nazwy pola K.Nazwa , w zestawie wynikowym pojawiłoby się ono jako Nazwa_1). Jak widać w zapytaniu obliczamy również wartość netto sprzedanego towaru oraz kwotę podatku VAT. Co prawda wartości te można by obliczyć również za pomocą komponentu TQRExpr , ale wyznaczenie ich w zapytaniu ułatwi późniejsze podsumowania. W zapytaniu pojawia się również parametr (: ID). Ponieważ okrśla on identyfikator drukowanej faktury, jego wartość będzie ustalana dopiero przed wydrukiem. Aby można było uruchomić kwerendę, należy określić typ danych jakie będą przypisywane temu parametrowi
4. W okienku Object TreeView rozwiń gałąź Query1 → Params i zaznacza parametr 0-ID. (Alternatywnie , możesz w okienku Object Inspectora odszukać komponent Query1, przejść do właściwości Params, po czym dwukrotnie kliknąć widoczny z prawej strony przycisku wielokropka). W Object Inspectorze pojawią się właściwości wskazanego parametru
5. Przejdź w Object Inspectorze do właściwości DataType i wybierz wartość ftInteger. Identyfikatory faktur są wartościami wskazanego parametru. Do uzyskania podsumowania według stawek podatku VAT musimy użyć innego zapytania, potrzebny więc będzie dodatkowy komponent TQuery
6. Umieść w raporcie komponent TQUery z palety BDE. Wstawiony komponent otrzyma nazwę Query2
7. Przejdź do Object Inspectora i w polu właściwości DatabaseName wpisz BAZAFAKT
8. Przejdź do właściwości SQL, uruchom edytor (kliknij przycisk oznaczony wielokropkiem) i wpisz tekst zapytania:
SELECT
SV.Pozycja, SV.Procent, TS.St_vat
SUM (TS.Cena_jedn * TS.Ilosc) as Netto,
SUM (TS.Cena_jedn * TS.Ilosc * SV.Procent / 100) as VAT
FROM
TranSzcz TS, St_VAT, SV
WHERE
TS.St_VAT = SV.IDSt_VAT
AND IDTranNagl = : ID
GROUP BY
SV.Pozycja, SV.Procent, TS.ST_VAT
ORDERE BY
SV.Pozycja
W powyższym zapytaniu pobieramy pola Pozycja i Procent z tabeli St_vat oraz pole St_VAT z

tabeli TranSzcz. Zastosowano również funkcję agregującą SUM, której zadaniem jest podsumowanie wszystkich cen netto (cena jednostkowa * ilość) towarów w poszczególnych stawkach oraz samego podatku VAT. Pole Pozycja nie pojawia się na wydruku - występuje ono w klazuli SELECT tylko dlatego , ponieważ chcemy go użyć do posortowania wyników (klauzula ORDER BY). W powyższym zapytaniu dostrzegamy praktyczne korzyści wynikające z zastosowania tabeli St_vat. Dzięki polu Pozycja możemy dowolnie ustalać kolejność poszczególnych stawek na wydruku (bez tego pola sortowanie według satwek byłoby znacznie utrudnione, ze względu na występowanie oprócz stawek procentowych stawek opisanych słownie, np. "zw") W zapytaniu tym można by od razu wyznaczyć kwotę brutto, lecz znając kwotę netto oraz podatek, wystarczy dokonać prostego sumowania i umieści je w komponencie QRExpr
9. W okienku Objetc TreeView rozwiń gałąź Query2 → Params i zaznacz parametr 0 - ID
10. Zaznacz komponent raportu ,a następnie ustaw jego właściwość DataSet na Query1. W ten sposób określamy podstawowy zestaw danych,z którym jest związany raport. Pominięcie tej operacji może spowodować , że niektóre sekcje raportu nie będą drukowane (W omawianej aplikacji , gdy właściwość DataSet raportu nie była ustawiona, na wydruku brakowałoby sekcji ColumnHeader)
11. W Object Inspectorze przejdź do właściwości DataType i wybierz wartość ftInteger

Powiązanie sekcji SubDetail z zestawami danych

Aby wydruk przebiegał zgodnie z naszymi oczekiwaniami, musimy skojarzyć każdą z sekcji SubDetail z odpowiednim zestawem danych
1. Zaznacz sekcję SubDetail
2. W Object Inspectorze odszukaj właściwość DataSet i wybierz wartość Query1
3. Zaznacz opcję SubDetail2
4. W Object Inspectorze odszukaj właściwość DataSet i wybierz wartość Query2
5. Kliknij przycisk Save All,a by zapisać wprowadzone zmiany

Sekcja tytułowa

Kolejnym, ważnym etapem pracy jest rozmieszczenie odpowiednich komórek w raporcie. Chociaż zadanie to jest często czasochłonne, nie można powiedzieć by było szczególnie twórcze. Z tego względu przedstawimy jedynie najważniejsze wskazówki dotyczące umieszczania obiektów w raporcie pozostawiając większość czynności do samodzielnego wykonania. W naszym przykładzie w nagłówku strony powinny się znaleźć następujące elementy:
• data i miejsce wystawienia dokumentu
• seria, numer i dopisek faktury
• pełne dane sprzedawcy i nabywcy, wraz z numerami NIP
• data sprzedaży
• termin płatności
• forma płatności

Wszystkie powyższe dane będą drukowane za pomocą kontrolek TQRLabel, TQRDBText oraz TQRExpr. Za pomocą etykiet będziemy prezentować informacje , które nie pochodzą z plików bazy danych (np. miejsce wystawienia faktury i dane sprzedawcy), oraz teksty stałe (niezmienne dla każdego raportu). Kontrolek TQRDBText użyjemy gdy drukowany element będzie po prostu reprezentować wartość jakiegoś pola zapytania. Kontrolkę TQRExpr zastosujemy w przypadkach, gdy wyrażenie wymaga dodatkowego formatowania lub powstaje z połączenia wartości kilku pól

Stosowanie komponentów TQRLabel i TQRDBText

Definiowanie elementów sekcji tytułowej rozpoczniemy od umieszczenia w niej kilku etykiet, reprezentowanych przez komponenty TQRLabel. Wykonaj poniższe czynności:
1. Zaznacz sekcję Title i przeciagnij jej dolny uchwyt wymiarowania, aby uzyskała wysokość 8 cm. (Możesz też kliknąć raport prawym klawiszem myszki, wybrać polecenie Report Settings i wpisać odpowiednią wartość w polu Length dla sekcji Title)
2. Wyświetl paletę QReport
3. Odszukaj na palecie komponent TQRLabel i umieść go w lewym górnym rogu sekcji Title (około 5 cm od prawego marginesu). W tym polu będzie wyświetlane miejsce wystawienia faktury. Wartość tę pobierzemy z pliku Faktury.ini. Ponieważ plik INI jest niedostępny w fazie projektowej, odpowiednie wyrażenie będziemy definiowali dopiero w momencie generowania dokumentu.

4. Gdy zaznaczony jest nowy komponent TQRLabel, przejdź do Object Inscpetora i we właściwości Name wpisz MiejsceQRLabel. Wprowadzając adekwatną nazwę komponentu, zwiększamy czytelność kodu, w którym będzie modyfikowana jego wartość
5. Ustaw właściwość Alignment komponentu etykiety na taRightAlignment. Ponieważ właściwość AutoSize etykiety ma wartość True i jednocześnie wybrano wyrównywanie prawostronne, prawa krawędź etykiety pozostanie nieruchoma, natomiast lewa będzie się dostosowywać do aktualnej długości drukowanego tekstu. Kolejnym komponentem, który umieścimy w raporcie , będzie komponent TQRDBText. Jak już mówiliśmy, pozwala on prezentować informacje tekstowe i numeryczne z bazy danych. W tym przypadku wykorzystamy go wydrukowania daty sporządzenia faktury
1. Z prawej strony komponentu MiejsceQRLabel wstaw komponent typu TQRDBText. Komponent ten posłuży do wyświetlenia daty wystawienia dokumentu.
2. Gdy zaznaczony jest nowy komponent, przejdź do Object Inspectora

Stosowanie komponentów TQRExpr

Zajmiemy się definiowaniem komponentu, który posłuży do wyświetlania numeru faktury. Najprostszym rozwiązaniem w tym przypadku byłoby użycie trzech kontrolek TQRDBText, przy czym każda z nich przedstawiałaby odpowiednią serię, numer i dopisek faktury. Takie rozwiązanie ma jednak podstawową wadę - na każdą z tych wartości musielibyśmy z góry zaplanować wystarczająco dużo miejsca. Oznacza to z kolei, że jeżeli użytkownik zdecyduje się na krótkie oznaczenie serii i dopisku lub jeżeli będzie operował na niskich numerach faktur, elementy te znajdowałyby się w pewnym oddaleniu od siebie. Efekt z pewnością nie byłby zbyt elegancki. Aby temu zaradzić, wszystkie trzy wartości (serię, numer i dopisek) będziemy wyświetlali za pomocą jednego komponentu QRExpr. Wykonaj poniższe czynności:
1. Z lewej strony sekcji Title, około 5 cm od brzegu strony, wstaw kolejny komponent TQRLabel
2. Przejdź do Object Inspectora i we właściwości Caption wpisz Faktura VAT. Zmodyfikowany tekst pojawi się również w raporcie. Jak widać, etykieta TQRLabel ma podobne właściwości jak zwykły komponent TLabel - nic dziwnego , gdyż TQRLabel, dziedziczy po klasie TLabel
3. Z prawej strony wstawionej etykiety umieść komponent TQRExpr
4. Przejdź do Object Inspector i we właściwości Name wpisz QRSeriaNumerDopisek. Jak wynika z podanej nazwy, w tym komponencie zakodujemy identyfikator faktury, zapisany w polach Seria, Numer i Dopisek
5. Przejdź do właściwości Expression wstawionego komponentu i wpisz wyrażenie :
Query.Seria + '/' + Query1.Numer + '/' + Query1.Dopisek
Jak widać ,składnia używana do definiowania wyrażeń nie różni się dpecjalnie od stosowanej w języku Pascal. Po zdefiniowaniu powyższego wyrażenia, w polu TQRExpres będą pojawiały się numery faktur np. FAKT/12/2012

Definiowanie obszaru z danymi sprzedawcy

Na razie nasza faktura drukuje jedynie datę wystawienia numer. Kolejnym elementem , który musimy zdefiniować, są dane sprzedawcy. Ponieważ dane te nie zmieniają się, zrezygnowaliśmy z umieszczenia ich w bazie danych i zapisaliśmy je w pliku INI. Oznacza to ,że do prezentowania tych wartości nie możemy użyć standardowych komponentów TQRDBText. W związku z tym znowu posłużymy się etykietami
1. Poniżej numery faktury umieść kolejną etykietę typu TQRLabel i zmodyfikuj jej właściwości Caption na oryginał/kopia. Ten tekst nie będzie się zmienił ale musimy jednak przewidzieć dla niego miejsce
2. Wstaw kolejną etykietę i zmodyfikuj jej właściwość Caption na SPRZEDAWCA. Umieść ją we właściwym miejscu
3. Obok wstawionej przed chwilą etykiety umieść kolejny komponent TQRLabel. Docelowo w polu tym pojawi się nazwa sprzedawcy. Nie stosujemy tutaj komponentu TQRDBText, ponieważ informacje o sprzedawcy zostaną pobranie nie z bazy danych, lecz z pliku INI
4. Przejdź do Object Inspectora i zmień właściwość Name nowego komponentu na Sprzedawca_dane1
5. Powtrarzając czynności z 3 i 4 , wstaw kolejne trzy komponenty TQRLabel i nazwij je odpowiednio : Sprzedawca_dane2, Sprzedawca_dane3 , Sprzedawca_dane4

Definiowanie obszaru z danymi nabywcy

Dane nabywcy pochodzą z bazy danych i w żaden sposób nie będą przetwarzane na wydruku faktury. Do ich zaprezentowania możemy więc użyć zwykłych komponentów TQRDBText. Wykonaj poniższe czynności:
1.Poniżej pola QRSprzedawca_dane4 wstaw etykietę TQRLabel i zmień jej właściwość na NABYWCA :
2. Z prawej strony obok wstawionej przed chwilą etykiety umieść komponent TQRDBText
3.W Object Inspectorze zaznacz właściwość DataSet wstawionego przed chwilą komponentu i z listy wybierz wartość Query1. Źródłem danych dla tego pola będzie zapytanie Query1
4. We właściwości DataField wybierz wartość Nazwa_kontr. Powtarzając czynności z 2 - 4 , umieść w sekcji Title pola TQRDBText. Ustaw ich właściwość DataSet na Query1 oraz właściwość DataField odpowiednio na : Miasto, Ulica, NIP. W analogiczny sposób wprowadź również pola Data_trans, Forma_platnosci i Termin_platnosci. Umieść je na jednej wysokości, u dołu sekcji Title. na lewo od każdego z tych pól umieść stosowaną etykietę. W razie potrzeby dostosuj wysokość sekcji
5. Kliknij przycisk Save aby zapisać wprowadzone zmiany

Pobieranie wartości z pliku INI

Jak już wspomnieliśmy w poprzedniej sekcji, część informacji umieszczonych na fakturze ma charakter stały. Informacje te są zapisane w pliku konfiugracyjnym Faktury.ini. Oznacza to ,że przed wydrukiem musimy otworzyć ten plik, pobrać z niego stosowne informacje , a następnie go zamknąć
1. Zaznacz obiekt raportu
2. Przejdź do Object Inspectora, otwórz karte Events i dwukrotnie kliknij w polu BeforePrint
3. Zmodyfikuj procedurę obsługi zdarzenia, aby uzyskała postać:

```
procedure TFakturaQForm.QuickRep1FBeforePrint(Sender : TCustomQuckRep; var PrintReport :
Boolean);
var
KonfigIni : TIniFile;
begin
{Pobranie wartości stałych}
try
KonfigINI := TIniFile.Create('Faktury.ini'); (Otwarcie pliku)
with KonfigINI do begin
Sprzedawca_dane1.Caption := ReadString('Nazwa', 'dane1', '') ;
Sprzedawca_dane2.Caption := ReadString('Nazwa', 'dane2', '') ;
Sprzedawca_dane3.Caption := ReadString('Nazwa', 'dane3', '') ;
Sprzedawca_dane4.Caption := ReadString('Nazwa', 'dane4', '') ;
MiejsceQRLabel.Caption := KonfigINI.ReadString('Inne', 'miejsce_wystaw', '');
finally
KonfigIni.Free;
end;
end;
```

4. Za słowem kluczowym implementation umieść odwołanie do biblioteki IniFiles

```
uses
IniFiles;
```

5. Zapisz wprowadzone zmiany i skompiluj aplikację

Nagłówki kolumn

Korzystając z komponentów TQRLabel, umieść w sekcji Column Header dziesięć etykiet i zmodyfikuj ich właściwości Caption na : Lp., Nazwa, Cena jedn ,KWiU, Ilość, J.m, art netto, %VAT, VAT, Wart brutto

Pierwsza sekca SubDetail

W pierwszej sekcji SubDetail znajdą się rekordy poszczególnych pozycji faktury. Odpowiednie pola zapytania Query1 będą reprezentowane na wydruku przez komponent TQRDBText. Dodatkowo musimy tutaj obliczyć i drukować cenę brutto poszczególnych towarów oraz numerowanie pozycji faktury. Do tego zadania użyjemy kompnentów TQRExpr

Drukowanie pozycji faktury

Aby w sekcji SubDetail pojawiły się wartości pozycji faktury, wykonaj poniższe czynności:
1. W sekcji QRSubDetail umieść osiem komponentów TQRDBText
2. Wyrównaj je w pionie zgodnie z nagłówkami następujących kolumn: nazwa towaru lub usług, Cena jedn., KWiU, Ilość, J.m. , Wart. netto, %VAT, i VAT
3. Zaznacz wszystkie wstawione komponenty TQRDBText i zmodyfikuj ich właściwość DataSet a Query1
4. Gdy wstawione komponenty są zaznaczone, zmodyfikuj ich właściwość AutoSize na False. W wierszu szczegółów jest dosyć mało miejsca, więc sami będziemy musieli dobrać maksymalną szerokość wszystkich pól. Gdyby właściwość AutoSize miała wartość True, poszczególne wartości na wydruku mogłyby się wzajemnie przesłaniać
5. Zmodyfikuj właściwości DataField komponentów TQRDBText wybierając kolejno nazwy pól : Nazwa, Cena_jedn, KWiU, Ilosc, J)miary, Netto, St_VAT i VAT
6. Zaznacz komponenty pól Cena_jedn, Ilosc, Netto, St_VAT oraz VAT, na następnie ustaw ich właściwość Alignment na taRightJustify. Ponieważ te komponenty prezentują wartości liczbowe warto je wyrównać do prawej
7. Dostosuj szerokości wszystkich wstawianych komponentów, aby każdy z nich zajmował maksymalny dostępny dla niego obszar Najlepiej gdy poszczególne komponenty TQRDBText bedą się ze sobą stykać

Wydruk liczby porządkowej z wykorzystaniem etykiety

Następnym elementem , który zdefiniujemy w wydruku będzie liczba porządkowa
1. Poniżej komponentu L.p . wstaw komponent TQRLabel
2. W Object Inspectorze zmień właściwość Name nowego komponentu na PozycjaQRLabel. Ta etykieta posłuży do wyświetlenia numeru bieżącego rekordu. Jej wartość będziemy modyfikować programowo przed wydrukiem
3. Ustaw właściwość AutoSize na False i Alignment na taRightJustify. Pole to będzie miało stały rozmiar, a wyświetlane w nim wartości będą wyrównywane do prawej
4. Dostosuj szerokość pola, aby nie zachodziło na pole z nazwą towaru

Obliczanie kwoty brutto za pomocą komponentu TQRExpr

Jako następny wstawimy do raportu komponent TQRExpr, który posłuży do wyznaczana kwoty brutto poszczególnych pozycji faktury. W tym celu wykonaj poniższe czynności
1. W sekcji QRSubDetail wstaw komponent TQRExpr. Umieść go pod nagłówkiem kolumny Wart. brutto
2. Dla nowo wstawionego komponentu ustaw właściwość AutoSize na False i Alignment na taRightJustify
3. Zmień właściwość Expression nowego komponentu na Query1.Netto + Query1.VAT
4. Dostosuj szerokość komponentu , aby zajmował maksymalny dostępny obszar

Rozciąganie zawartości pola na kilka wierszy - właściwość AutoStrech

Szerokości pól liczbowych są na tyle duże, aby bez problemu pomieścić kwoty w wysokości kilkuset tysięcy złotych, a więc w typowych zastosowaniach wystarczające. Nie możemy tego jednak powiedzieć o kolumnie przeznaczonej na nazwy. Jak pamiętamy, w bazie danych pole to ma szerokość 80 znaków, natomiast na wydruku zmieści się ich co najwyżej 30. Niestety rozszerzenie tego pola spowodowałoby konieczność zwężenia pozostałych elementów na wydruku. Kompromisowym rozwiązaniem w takiej sytuacji jest ustawienie właściwości AutoStrech komponentu TQRDBText na True. Spowoduje to automatyczne przeniesienie tekstu do następnej linii i w razie potrzeby wydłużenie całej sekcji, aby mogła pomieścić rozciągnięte pole. Z właściwości AutoStrech należ y korzystać ostrożnie. Można ją ustawić dla większej liczby komponentów w sekcji lecz muszą one znajdować się jeden obok drugiego a nie jeden nad drugim. Właściwość AutoStrech jest również bardzo przydatna przy wydruku nieformatowanych pól typu Memo (do wydruku formatowanych pól Memo stosuje się komponent TQRDBRichText), ponieważ wystarczy w odpowiednim miejscu umieścić komponent TQRDBText, przypisać mu odpowiednie pole Memo z tabeli lub zapytania i ustawić jego właściwość AutoStrech na True. Pole zostanie prawidłowo wydrukowane, nawet jeżeli będzie się to wiązało z rozciągnięciem go na kilku stronach. Aby uzyskać ten efekt, wykonaj poniższe czynności:
1. Zaznacz komponent TQRDBText wyświetlający pole Nazwa
2. W Object Inspectorze zmodyfikuj właściwość AutoStrech komponentu na True

3. Zredukuj wysokość sekcji SubDetail, aby zlikwidować niewykorzystane miejsce
4. Kliknij Save , aby zapisać wprowadzone zmiany

Druga sekcja SubDetail

Nagłówek

Ta sekcja nie przysporzy nam żadnych problemów. Jedynym jej celem jest wydruk krótkiego tekstu informującego o charakterze zestawienia. W sekcji Group Header (nazwa wyświetlana w raporcie) umieść komponent etykiety i zmodyfikuj jej tytuł na Podsumowanie według stawek podatku VAT. Wyrównaj etykietę i zmniejsz rozmiar sekcji nagłówkowej

Podsumowanie według stawek podatku VAT

W tej sekcji SubDetail będą wyświetlane podsumowania pogrupowane według stawek. Źródłem danych tej sekcji jest zapytanie Query2. Do wyświetlenia odpowiednich wartości będziemy potrzebowali trzech komponentów TQRDBText (kwota netto, podatek VAT i stawka VAT) oraz jednego TQRExpr (kwota brutto)

1. W sekcji SubDetail2 umieść trzy komponenty TQRDBText
2. Zaznacz wszystkie wstawione komponenty i zmodyfikuj ich właściwości DataSet na Query2, Alignment na taRightJustify i AutoSize na False
3. W kolejnych polach typu TQRDBText we właściwośc DataField wpisz odpowiednio : Netto, St_VAT i VAT
4. Wstaw komponent TQRExpr
5. Ustaw jego właściwość Alignment na taRightJustify i AutoSize na False
6. We właściwości Expression wpisz następujące wyrażenie Query2.Netto + Query2.VAT
7. Wyrównaj wstawione komponenty i dostosuj wysokość sekcji . Postaraj się, aby poszczególne pola znalazły się dokładnie pod odpowiadającymi im polami z pierwszej sekcji SubDetail

Sekcja podsumowania wydruku

Jak powszechnie wiadomo. diabeł tkwi w szczegółach, a ściślej mówiąc w groszach. Wydawać by się mogło ,że nic prostszego, jak podsumować wszystkie wartości brutto raportu by uzyskać całkowitą sumę faktury. Niestety sprawa jest nieco bardziej skomplikowana. Rozważmy następujący przykład ;dla trzech różnych towarów uzyskaliśmy niżej wymienione kwoty podatku VAT:

12,6666
7,6666
3,6666

Tysięczne części złotego mogą się pojawić, ponieważ , po pierwsze, ilość towaru może być ułamkowa, po drugie, przemnożenie przez stawkę podatku VAT również może spowodować powstanie setnych części grosza. Jednak na wydruku faktury wartości kwotowe zostaną przedstawione tylko z dwoma miejscami po przecinku. Co więcej w środowisku Delphi podczas formatowania następuje zaokrąglenie wartości, a więc na wydruku zobaczymy liczby:

12,67
7,67
3,67

. Z prostego rachunku wynika ,że suma wydrukowanych liczb wynosi 23,01. Jednak w podsumowaniach nie jest brany pod uwagę format drukowania, lecz faktycznie pamiętane wartości. Oznacza to ,że gdybyśmy polecili komputerowi obliczyć sumę powyższych liczb, wynik wyniósłby 23,00. Mamy więc różnicę jednego grosza. Jak łatwo się domyślić, gdy faktura składa się z wielu pozycji, błąd może urosnąć do kilkunastu groszy - a t już na pewno nie spodoba się w pewnym urzędzie. Aby uniknąć tego problemu musimy więc dokonywać obliczeń na wartościach zaokrąglanych do 1 grosza. Zastosujemy mechanizm programowego obliczania sum z wykorzystaniem funkcji RoundTo z biblioteki Math

1. W sekcji Summary umieść cztery komponenty typu TQRLabel
2. Zmień właściwość Caption pierwszego komponentu na RAZEM :
3. Zmień nazwy pozostałych trzech etykiet kolejno : NettoRazemQRLabel, VATRazemQRLabel i BruttoRazemQRLabel. Ustaw ich właściwości Alignment na taRightJustify oraz AutoSize na False
4. Rozmieść etykiety na formularzu. Jak z pewnością się domyślasz, sumy będą wyświetlane za pomocą etykiet. Zwykle w tym celu posługujemy się komponentami TQRExprm w których można skorzystać z funkcji SUM. Na przykład aby uzyskać sumę wartości netto można by użyć

komponentuTQRExpr z wyrażeniem SUM(Query1,Cena_jedn * Query1.Ilosc). Jednak aby komponent TQRExpr prawidłowo zliczył sumę, w raporcie musi się pojawić sekcja Detail. Jak pamiętamy, świadomie zrezygnowaliśmy z jej użycia, co zmusza nas do samodzielnego wykonania obliczeń. W tym celu zdefiniujemy dodatkowe zmienne. Wartości tych zmiennych będą zwiększane podczas wydruku każdej kolejnej sekcji SubDetail. W rezultacie, po wydrukowaniu wszystkich rekordów uzyskanych w zapytaniu Query1, zmienne te będą zawierały odpowiednie sumy

5. Zaznacz komponent raportu (kliknij w pustym miejscu w pobliżu marginesu), przejdź do Object Inspectora i na karcie Events dwukrotnie kliknij w polu zdarzenia BeforePrint. W tej procedurze będziemy zerowali sumowane elementy. W ten sposób zyskamy pewność ,że przed rozpoczęciem wydruku ,zmienne odpowiedzialne za naliczenie sum będą wyzerowane

6. W obszarze private sekcji interfejsu modułu umieść deklaracje zmiennych:

```
private
{Private declarations}
RazemNetto : double;
RazemVAT : double;
```

7. Uzupełnij procedurę obsługi zdarzenia o dwa poniższe polecenia

```
RazemNetto := 0.0;
RazemVAT := 0.0;
```

8. Wróć do widoku raportu, zaznacz sekcję QRSubDetail1 i w Object Inspectorze dwukrotnie kliknij w polu zdarzenia BeforePrint

9. Uzupełnij procedurę obsługi zdarzenia, aby uzyskała postać:

```
procedure TFakturaQForm.QRSubDetail1BeforePrint(Sender : TQRCustomBandl var PrintBand :
Boolean);
begin
{narastujące obliczenia sumy}
RazemNetto := RazemNetto + RoundTo(Query1.FieldByName('Netto').Value , -2);
RazemVAT := RazemVAT + RoundTo(Query1.FieldByName('VAT').Value , -2);
{wyznaczenie liczby porządkowej}
PozycjaQRLabel.Caption := inttoStr(Query1.Recno)
```

Jak wynika z powyższego kodu, przed wydrukowaniem każdej sekcji szczegółów, będzie następowało zwiększenie wartości RazemNetto iRazemVAT o bieżące wartości pól Netto i VAT z zapytania Query1. Funkcja RoundTo powoduje zaokrąglenie tych liczb do dwóch miejsc po przecinku. Zwróć uwagę ,że w procedurze tej załatwiamy od razu problem numerowania pozycji faktury (kolumna L.p.). Po prostu właściwości Caption etykiety PozycjaQRLabel przypisujemy numer bieżącego rekordu (właściwość RecNo)

10. W klauzuli uses sekcji implementation wstaw odwołanie do modułu Math

```
uses
IniFiles, Math;
```

11. Wróć do widoku raportu, zaznacz sekcję Summary i w Object Inspectorze dwukrotnie kliknij w polu zdarzenia BeforePrint

12. Uzupełnij procedurę obsługi zdarzenia, aby uzyskała postać:

```
procedure TFakturaQForm.QRBand4BeforePrint(Sender : TQRCustomBand; var PrintBand :
Boolean);
begin
NettoRazemQLabel,Caption := FloatToStr(RazemNetto, ffNumber, 15,2);
VATRazemQRLabel.Caption := FloatToStr(RazemVAT, ffNumber, 15,2);
BruttoRazemQRLabel.Caption := FloatToStr(RazemNetto + RazemVAT, ffNumber, 15,2);
end;
```

Funkcja FloatToStr konwertuje wartości zmiennoprzecinkowe na formatowane ciągi znaków. Parametr ffNumber oznacza ,że liczba zostanie sformatowana z użyciem systemowego symbolu grupowania (zwykle jest to spacja pojawiająca się co trzy pozycje). Trzeci parametr (liczba 15) określa precyzję - ilość pozycji znaczących. W przypadku wartości typu double precyzja nie powinna być większa niż 15. Ostatni parametr wskazuje liczbę miejsc dziesiętnych. Suma faktury brutto jest wyznaczana przez proste zsumowanie kwoty netto i podatku VAT

Powiązanie wydruku faktury z formularzem FakturaForm

Chociaż wydrukowi faktury daleko jeszcze do doskonałości, warto jednak zdefiniować mechanizm pozwalający na jego uaktywnienie z poziomu działającej aplikacji. Podgląd wydruku oferowany w fazie projektowej nie pozwala uzyskać wszystkich informacji - nie widzimy na nim wartości pól wyznaczonych w kodzie programu. W tym momencie jak zwykle pojawia się pytanie, z którego miejsca aplikacji będzie uaktywniany wydruk. Łatwo się domyślić , że wydruk faktury jest zwykle inicjowany zaraz po jej wypełnieniu, dlatego też najbardziej naturalnym miejscem, w którym powinniśmy udostępnić tę opcję , jest formularz FakturaForm

1. Korzystając z przycisku View Form wyświetl formularz FakturaForm
2. Przejdź do palety Additional, zaznacz na niej kompnent TBitBtn i umieść go z prawej strony , u góry formularza
3. Przejdź do Object Inspectora i zmień właśściwość Name wstawionego przycisku na DrukujBitrBtn oraz właściwość Caption na &Drukuj
4. Dwukrotnie kliknij przycisk DrukujBitBtn
5. Uzupełnij procedurę zdarzeia OnCick zgodnie z poniższym wydrukiem:

```
procedure TFakturForm.DrukujBitBtn (Sender : TObject);
begin
{zatwierdznie ewentualnych zmian w tabeli nagłówka …}
if FaktDModule.Table1.State in [dsEdit, dsInsert] then
FatDModule.Table1.Post;
{… i szczegółów faktury}
if Table2.State in [dsEdit, dsInsert] then
Table2.Post;
with FakturaQForm do begin
{przygotowanie zapytań genrujących dane raportu}
if Query1.Active then Query1.Close;
if Query2.Active then Query2.Close;
Query1.ParamByName('ID').Value := FaktDModule.Table1IDtranNagl.Value;
Query2.ParamByName('ID').Value := FaktDModule.Table1IDtranNagl.Value; try
Query1.Open;
Query2.Open;
{Wydruk raportu}
QuickRep1.Preview;
finally
Query2.Close;
Query1.Close;
end;
end;
end;
```

Zwróć uwagę ,że w powyższym wydruku użyto metody Preview, a nie Print. Oznacza to ,że raport zostanie wyświetlony w okienku poglądu. W końcu do uzyskana końcowej formy raportu wciąż mamy sporo pracy, a nie będziemy tracili paczek papieru i cennych minut , by drukować każdą poprawioną jego wersję. Dopiero gdy raport będzie już gotowy, metodę Preview należy zastąpić metodą Print

6. Naciśnij kombinację klawiszy [Ctrl + F9] aby rozpocząć kompilację
7. Gdy kompilator zaproponuje dołączenie modułu raportu FakturaQReport do klauzuli uses , kliknij przycisk Yes
8. Zapisz wprowadzone zmiany; skompiluj i uruchom aplikację
9. Wyświetl dowolną fakturę i kliknij Drukuj

Ostatnie "pociągnięcia pędzlem"

Możemy powiedzieć ,że surowa wersja faktury jest już gotowa. Brakuje tylko słownego przedstawienia jej wartości (ale tym zajmiemy się w ostatniej części). Pozostała część pracy ma charakter estetyczny.

Korzystanie z podglądu wydruku w fazie projektowej

Jak już wspomnieliśmy, w fazie projektowej jest również dostępny podgląd wydruku. Mechanizm

ten jest bardzo przydatny , gdyż od razu możemy obserwować efekt wprowadzanych zmian. Jednak , aby podgląd wydruku działał prawidłowo, zestawy danych , na których jest oparty wydruk, powinny być otwarte. Problem pojawia się , gdy jak w naszym przypadku, zestawem danych jest zapytanie sprametryzowane. W takiej sytuacji czas projektowania można wprowadzić roboczą wartość parametru, a następnie uaktywnić zapytanie. W tym celu:
1. Zaznacz komponent zapytania i w Object Inspectorze kliknij przycisk wileokropka w polu właściwości Params. Pojawi się okno dialogowe Edititng Query.Params
2. Zaznacz parametr
3. W Object Inspectorze wybierz odpowiedni typ w polu DataType (w omawianym przykładzie ftInteger) oraz wpisz wartość domyślną parametru w polu Value (w naszym przykładzie należy wybrać jedną z istniejących wartości identyfikatora transakcji - IDTranNagl). Aby uzyskać jakąś wartość identyfikatora transakcji ,wyświetl tabelę TranNagl w programie Database Desktop
4. Ponownie zaznacz komponent zapytania i ustaw jego właściwość Active na True. Jeżeli w zapytaniu są błędy, operacja się nie powiedzie

Jednorodne formatowanie pól wyświetlających wartości liczbowe

Część pól w utworzonym raporcie wyświetla wartości kwotowe w formacie walutowym (są to pola typu TQRDBText), natomiast część jest w ogóle niesformatowana. Aby ujednolicić format wyświetlania tych pól, zaznacz je wszystkie , a następnie we właściwości Mask wpisz:
• # ##0.00 - jeżeli chcesz drukować liczby z dwoma miejscami dziesiętnymi i z odstępem co trzy pozycje całkowite
• # ##0.00 zł - jeżeli dodatkowo chcesz drukować symbol waluty
Z uwagi na dużą ilość informacji drukowanych w wierszu pozycji faktury, bezpieczniej będzie zastosować pierwszy format, dzięki czemu uzyskamy więcej miejsca dla samych wartości. Zwróć uwagę ,że wartości wyliczane w polach TQREpr nie są interpretowane jako kwoty , a więc nie stosuje się w nich domyślnego formatowania dla tego typuliczb. W związku z tym w każdym z pól TQREpr musisz zdefiniować maskę # ## 0.00 lub # ##0.00 zł. W wydruku faktury dotyczy to pól z wartością brutto w obydwu sekcjach SubDetail

Obramowanie elementów wydruku

W zestawieniach tabelarycznych (a takim jest również faktura) zwykle stosujemy różnego rodzaju obramowania. Gdy korzystamy z komponentów QuickReport, obramowania można uzyskać na dwa spsoby :
• Ustawiając odpowiednie wartości właściwości Frame (ma ją każdy komponent drukowalny, łącznie z samym komponentem raportu)
• stosując komponent TQRShape
Kwota do zapłaty jest etykietą, której tytuł wyznaczany w taki sam sposób ,jak tytuł etykiety BruttoRazemQRLabel. Dodatkowo do kwoty dołączony jest symbol waluty:
DoZaplatyQRLabel.Caption := FloatToStrF(RazemNetto + RazemVAT, ffNumber, 15,2) + ' zł' ;
Na samym dole raportu umieszczone zostało pole TQRDBText, w którym wyświetlana jest zawartość pola Uwagi z zapytania TQuery1

Końcowe udoskonalenia

Wyszukiwanie przyrostowe

Jednym z udogodnień, które warto wprowadzić jest tzw. wyszukiwanie przyrostowe. Dotyczy zwłaszcza tabel słownikowych, zawierających nazwy towarów oraz kontrahentów. Z czasem , gdy liczba danych zgromadzonych w aplikacji zacznie się zwiększać, dostrzeżesz potrzebę szybkiego wyszukiwania określonych rekordów. Na przykład gdy na liście kontrahentów będzie już kilkaset pozycji, odszukanie kontrahenta o symbolu rozpoczynającym się na literę "P" będzie wymagało przewinięcia kilku a nawet kilkunastu ekranów. Aby tego uniknąć, próbujemy wprowadzić mechanizm, który będzie wyszukiwał kontrahentów (oraz towary) na podstawie kilku pierwszych liter ich identyfikatora. w formularzu z listą kontrahentów wprowadzimy dodatkowe pole tekstowe, w którym będą wpisywane początkowe litery kodu kontrahenta. Jeżeli na przykład użytkownik wpisze literę "K" nastąpi przejście do pierwszego symbolu kontrahenta zaczynającego się od tej litery. Gdy użytkownik wpisze kolejną literę m na przykład "O", nastąpi odszukanie pierwszego kontrahenta , którego identyfikator zaczyna się od liter "KO" itd. Stąd właśnie wzięła

się nazwa wyszukiwanie przyrostowe (incremental search)

Przygotowanie formularza do wyszukiwania

Ponieważ mechanizm ten będzie działał w analogiczny sposób na liście kontrahentów i na liście towarów/usług, zdefiniujemy do formularzu wzorcowym ListaForm. Rozpoczniemy od wstawienia nowego panelu , na którym znajdzie się pole tekstowe ze znakami używanymi do wyszukiwania

1. Kliknij przycisk View Form i z listy wybierz formularz ListaForm
2. Zaznacz kontrolkę Panel2. Jest to panel , na którym jest wyświetlana siatka danych. Jeżeli będziesz miał problemy z zaznaczeniem tego panelu (prawie w całości jest on zasłonięty przez siatkę), skorzystaj z okna Object Tree View
3. Ustaw właściwość Align wybranego panelu na alNone. Umożliwi to ręczne zmniejszenie jego rozmiarów i umieszczenie na formularzu dodatkowego panelu
4. Przeciągnij w dół górną krawędź kontrolki Panel2 a następnie w zwolniony, obszarze umieść nowy panel
5. Usuń tekst z właściwości Caption kontrolki Panel3. Ustaw jej właściwość Align na alTop. Kontrolka zostanie wyrównana do góry formularza
6. Ustaw właściwość BevelOuter panelu na bvNone a właściwość Height na 32. Dzięki temu nowy panel będzie wyglądał identycznie ,jak powyższy na którym znajduje się pasek nawigatora oraz przyciski Edytuj reokrd i Drukuj
7. Ponownie zaznacz Panel2 i przywróć jego właściwości Align wartość alClient. Kontrolka Panel2 zajmie cały dostępny obszar a więc podobny do poprzedniego, lecz pomniejszonego o nowy panel
8. Z lewej strony nowego panelu umieść etykietę i zmień jej właściwość Caption na Wyszukaj
9. Obok etykiety umieść pole tekstowe. Zmień jego właściwość Name na WyszukajEdit oraz usuń zawartość właściwości Text

Oczywiście zastosowanie dodatkowego panelu nie jest konieczne jednak takie rozwiązanie jest dla nas znacznie wygodniejsze. Opcja wyszukiwania przyrostowego będzie dostępna tylko podczas wyświetlania listy kontrahentów i towarów/usług. Gdy wyświetlimy listę faktur, wyszukiwanie przyrostowe nie będzie dostępne. Można by wtedy wyłączyć odpowiednie kontrolki (ustawić właściwości Enabled etykiety i pola tekstowego na True, ale po co marnować miejsce - o wiele prostsze będzie całkowite ukrycie Panelu2

Proste wyszukiwanie

Kolejnym zadaniem jest oprogramowanie mechanizmu wyszukiwania przyrostowego. rekordy w zestawie danych można odnajdywać na wiele sposobów. Służą do tego między innymi metody FindNearest i Locate obiektu TDataSet. w naszym przykładzie zastosujemy jednak inne rozwiązanie, wykorzystując metodę GotoNearest. Umożliwia ona (podobnie jak FindNearest) oszukanie rekordu najbardziej zbliżonego do podanej wartości klucza. Jeżeli nie zostanie znaleziony rekord, który dokładnie odpowiada podanemu kluczowi, kursor tabeli jest ustawiony na pierwszym rekordzie, którego klucz przekracza wyszukiwaną wartość. Oznacza to ,że jeżeli będziemy szukali klienta o identyfikatorze KO, to jeśli taki identyfikator nie istnieje, kursor zostanie ustawiony w następnym rekordzie (np. o identyfikatorze KOWALSKI). Takie rozwiązanie zupełnie nas satysfakcjonuje. Musimy jeszcze określić zdarzenie, które będzie powodowało rozpoczęcie wyszukiwania - z pewnością najlepiej nadaje się do tego zdarzenie OnChange komponentu WyszukajEdit. W tym celu musisz wykonać poniższe czynności:

1. Na formularzu ListaForm zaznacz komponent WyszukajEdit, przejdź do Object Inspectora i dwukrotnie kliknij w polu zdarzenia OnChange
2. Uzupełnij procedurę obsługi , aby uzyskała postać:

```
procedure TListForm.WyszukajEditChange(Sender : TObject);
begin
with DBGrid1.DataSource.DataSet as TTable do begin
SetKey;
FieldByName(DBGrid1.Columns[0].FieldName).AsString := WyszukajEdit.Text;
GotoNearest;
end;
end;
```

Przed wywołaniem metody GotoNearest zestaw danych musi się znaleźć w stanie dsSetKey, aby

można było zdefiniować lub zmodyfikwać klucz wyszukiwania. W tym celu należy skorzystać z metody SetKey lub EditKey. Kolejne polecenie FiledByName, przypisuje pierwszemy polu siatki (jego nazwę uzyskujemy z właściwości FiledName zerowego elementu kolekcji Columns siatki) tekst wpisany przez użytkownika w polu WyszukajEdit. Następna instrukcja GotoNearest, powoduje wykonanie wyszukiwania. W rezultacie kursor ustawia się na rekordzie ,w którym wartość klucza jest najbardziej zbliżona do podanego łańcucha znaków. Zwróć uwagę na budowę instrukcji with - zastosowaliśmy w niej wyrażenie DBGrid1.DataSource.DataSet, co pozwala nam określić aktualny zestaw danych przypisany do siatki bez konieczności sprawdzania, która tabela jest aktualnie wyświetlana. Dzięki temu cała procedura będzie działać prawidłowo niezależnie od zestawu danych przypisanego siatce (KontrahDModule.Table1 lub TowarDModule.Table1). Następnie, za pomocą operatora as , wyrażenie to jest rzutowane na typ TTable, dzięki czemu uzyskujemy dostęp do metod komponentów tego typu (np. SetKey)
3. W sekcji uses interfejsu umieść odwołanie do modułu DBTables. Po modyfikacji sekcja ta będzie miała postać:
uses
Windows, Messages, Classes, SysUtils, Graphics, Controls, StdCtrls, Forms, dialogs, DBCtrls, DB, DBGrids, Grids, ExtCtrls, Buttons, QUickRpt, Printers, DBTables;
Jak zapewne pamiętasz, podczas tworzenia formularza listy, zdecydowaliśmy się na zastosowanie oddzielnego modułu danych. W reuzltacie w module formularza nie znalazło się odwołanie do modułu DBTables, w którym jest zdefiniowana klasa TTable. Jednakw procedurze opisanej w punkcie 2 stosujemy rzutowanie na tą klasę. Konieczne jest więc wprowadzenie tej deklaracji
4. Kliknij przycisk Save All, aby zapisać wprowadzone zmiany
Możesz teraz skompilować i uruchomića aplikację. Wyświetl listę kontrahentów lub towarów/usług i spróbuj wpisać początkowe litery jakiegoś identyfikatora. Wskaźnik bieżącego rekordu przesunie się do identyfikatora klienta (lub towaru), którego początkowe litery kodu są zgodne z wprowadzonymi znakami (lub do następnego, jeżeli brak zgodności)

Wyszukiwanie z autouzupełnianiem tekstu pola

Wyszukiwanie z autouzupełnianiem tekstu pola

Spróbujmy teraz nieco udoskonalić przedstawione rozwiązanie. Z pewnością działanie pola wyszukiwania stanie się przyjemniejsze dla oka, jeżeli wyświetlany w nim tekst będzie automatycznie uzupełniany o resztę identyfikatora klienta lub towaru. W podobny sposób działa na przykład kontrolka DBLookupCOmbo i wiele innych funkcji automatycznego uzupełniania, na które możemy natknąć się w środowisku Windows. Jednak aby wprowadzić ten mechanizm musimy dosyć znacznie rozbudować kod programu. Oto czynności które należy wykonać
• Zastąp dotychczasowy kod procedury WyszukajEditChange następującym:

```
procedure TListaForm. WyszukajEditChange(Sender : TObject);
var
Szukany, Znaleziony : string;
begin
{Jeżeli użytkownik usuwa znaki, następuje przerwanie obsługi zdarzenia}
if Kasowanie then begin
Kasowanie := false;
Exit;
end;
Szukany := WyszukajEdit.Text;
{Jeżeli pole tekstowe jest puste, nie trzeba niczego szukać}
if Length(Szukany) = 0 then Exit;
{Wyszukiwanie rekordu}
with DBGrid1.DataSource.DataSet as TTable do begin
SetKey;
FieldByName(DBGrid1.Columns[0].FieldName).AsString := Szukany;
GotoNearest;
Znaleziony := FieldByName(DBGrid1.Columns[0].FieldName).AsString ;
end;
{Zmiana tekstu na wielkie litery}
```

```
WyszukajEdit.Text := UpperCase(Znaleziony);
{Określenie początku obszaru zaznaczania}
WyszukajEdit.SelStart := Length(Szukany);
{Określenie końca obszaru zaznaczania}
WyszukajEdit.SelLength :- Length(Znaleziony) - Length(Szukany);
end;
```

W powyższym kodzie wprowadzono dwie zmienne lokalne : Szukany i Znaleziony. Pierwsza z nich zawiera tekst, który użytkownik wpisał w polu WyszukajEdit, natomiast druga tekst znalezionego klucza. Procedura zostanie przerwana , jeżeli użytkownik wykonuje operację usuwania tekstu z pola lub gdy pole jest puste - za zrealizowanie tych postulatów odpowiadają dwa pierwsze warunki (zmienna Kasowanie zostanie omówiona później). Kolejna grupa instrukcji wykonuje omówioną już wcześniej operację wyszukiwania. Zwróć uwagę ,że metoda FieldByName została tu użyta dwókrotnie, jednak w dwóch różnych kontekstach. Podczas jej pierwszego wystąpienia zestaw danych jest w trybie dsSetKey, co oznacza ,że wartość przypisana pierwszemu polu siatki jest w rzeczywistości wartością wyszukiwanego klucza. W drugim przypadku, ponieważ po wykonaniu metody GotoNearest zestaw danych powrócił do trybu dsBrowe, metoda FieldByName zwraca faktyczną zawartość pierwszego pola siatki. Następna instrukcja zmienia wielkość liter. Jak zapewne zauważyłeś , do tej pory tekst w polu WyszukajEdit był wpisywany małymi literami - jednak ponieważ założyliśmy ,że symbole kontrahentów i towarów są wpisywane wielkimi literami, dla wygody użytkownika dokonujemy tutaj odpowiedniej konwersji (funkcja UpperCase). Ostatnie dwie instrukcje wykorzystują metody SelStart i SelLengrh kontrolki WyszukajEdit. Pierwsza z nich określa pozycję początkową zaznaczonego obszaru, druga wyznacza jego długość. W rezultacie uzyskujemy efekt zaznaczenia wszystkich znaków, których nie wpisał samodzielnie użytkownik, lecz które zostały pobrane z wartości klucza. Dodatkowo, aby mechanizm automatycznego uzupełnienia tekstu działa prawidłowo, musimy zdefiniować zmienną Kasowanie. Jej zadaniem jest informowanie czy użytkownik nie kasuje tekstu w polu WyszukajEdit. Jak już mówiliśmy - w takim przypadku operacja wyszukiwania klucza nie powinna być realizowana

1. W sekcji private klasy formularza zdefiniuj zmienną Kasowanie :

```
Kasowanie : boolean;
```

2 . Na formularzu ListaForm zaznacz kontrolkę WyszukajEdit, przejdź do karty Events bject Inspectora, a następnie utwórz procedurę obsługi dla zdarzenia OnKeyDown. Zdarzenie to jest uaktywniane w momencie, gdy użytkownik naciśnie jakiś klawisz

3. Uzupełnij procedurę, aby uzyskała postać:

```
procedure TFormList.WyszukajEditKeyDown(Sender : TObject; var Key : Word; Shift :
TShift(State);
begin
if (Key = VK_DELETE) or (Key = VK_BACK) then
if Length(wyszukajEdit.Text)> 0 then
Kasowanie := true;
end;
```

Stałe VK_DELETE oraz VK_BACK reprezentują klawisze [Delete] i [Backsapce]. Jeżeli użytkownik naciśnie któryś z nich, zmienna Kasowanie jest ustawione na True, co w stosownym momencie spowoduje przerwanie działania procedury obsługi WyszukajEditChange.

Ostatnim zadaniem do wykonania jest ukrycie panelu z polem WyszukajEdit, gdy wyświetlany jest formularz z listą faktur. Ta czynność jest bardzo prosta

1. Kliknij przycisk View Unit i wyświetl moduł FaktL

2. Odszukaj procedurę obsługi zdarzenia FormShow i uzupełnij ją aby uzyskała postać:

```
procedure TFaktLForm.FormShow(Sender : TObject);
begin
' inherited;
Panel3.Visible := false;
FaktDModule.Table1.Last;
end;
```

3. Kliknij przycisk Save All, aby zapisać wprowadzone zmiany

4. Skompiluj i uruchom aplikację
Możesz teraz przetestować wprowadzone modyfikacje. Uzysakny efekt z pewnością będzie o wiele bardziej interesujący niż przedtem. Sprawdź też , czy w trakcie wyświetlania listy faktur pole wyszukiwania jest niewidoczne

Właściwość ActiveControl

Wyobraź sobie ,że użytkownik wystawia dziennie około 50 faktur. Tworząc nowy dokument ,za każdym razem klika przycisk na pasku nawigatora, a następnie przechodzi do pola z numerem faktury. Wymagający użytkownik od razu zapyta : "A czy nie można tak zrobić , że gdy tworzę nową fakturę, to kursor od razu ustawia się w polu Numer?". Oczywiści, że można. Wystarczy w odpowiednim momencie modyfikować właściwość ActiveControl. Aby zdefiniować właściwość ActiveControl dla formularza FakturaForm, wykonaj poniższe czynności:
1. Uruchom Delphi i otwórz projekt Faktury.dpr
2. Kliknij przycisk View Form i przejdź do modułu danych FaktDModule
3. Na końcu procedury obsługi zdarzenia Table1NewRecord wpisz:

```
if Assigned(FakturaForm) then
FakturaForm.ActiveControl := FakturaForm.EditSeria;
```

Głównym elementem powyższego fragemntu kodu jest metoda ActiveControl. Ustawia ona fokus (uaktywnienia) na kontrolke wskazaną z prawej strony instrukcji przypisania. Ustawienie fokusu będzie wykonywane za każdym razem, gdy do tabeli zostanie wstawiony nowy rekord. Polecenie to jest wykonywane pod warunkiem,że formularz FakturaForm istnieje (instrukcja Assigned(FakturaForm) zwróci wartość True). Warunek ten umieszczamy tutaj nieco na wyrost, przewidując ewentualne inne zachowania modułu danych FaktDModule, w sytuacjach, gdy formularz FakturaForm nie będzie dostępny
4. Za słowem kluczowym implementation umieść dyrektywę : uses Faktura;
5. Zapisz wprowadzone zmiany i skompiluj aplikację
6. Przetestuj wprowadzone zmiany. Ustawiaj kursor w różnych miejscach formularza FakturaForm i rozpoczynaj edycje nowego rekordu
Po każdym wstawieniu nowego rekordu fokus automatycznie jest ustawiany na polu tekstowym EditSeria .Analogiczne modyfikacje wprowadź w formularzu do edycji towarów / usług i kontrahentów. Po wyświetleniu tych formularzy powinno być aktywne pole odpowiednio z kodem towaru / usługi lub kodem kontrahenta
Równie praktyczne będzie ustawienie fokusu na komponencie siatki za każdym razem , gdy będzie otwierana lista faktur, kontrahentów lub towarów/usług. Ponieważ operacja ta będzie wykonywana dla każdej z wymienionych list, odpowiednią procedurę obsługi zdarzenia możemy zdefiniować w obiekcie macierzystym - formularzu ListaForm<
1. Kliknij przycisk View Form i wybierz z listy formularz ListForm
2. Gdy zaznaczony jest formularz przejdź do karty Events Object Inspectora i dwukrotnie kliknij w polu zdarzenia OnActivate. Zdarzenie OnActivate jest wzbudzane , gdy formularz uzsykuje fokus (staje się aktywnym formularzem)
3. Uzupełnij procedurę obsługi:

```
procedure TListaForm.FormActivate(Sender : TObject;)
begin
if Panel3.Visible then
ActiveControl := WyszukajEdit
else
ActiveControl := DVGrid1;
end;
```

Jak pamiętasz, kontrolka Panel3 , na której znajduje się pole WyszukiwanieEdit jest widoczna tylko podczas wyświetlania list kontrahentów i towarów/usług. W rezultacie próba uaktywnienia tego pola w momencie wyświetlenia listy faktur spowodowałby błąd.
4. Zapisz wprowadzone zmiany i skompiluj aplikację. Od tego momentu każde uaktywnienie formularza z listą faktur, kontrahentów lub towarów/usług będzie powodowało ustawienie fokusu w komponencie siatki

Programowe przemieszczanie kursora w zestawie danych

Jeśli prześledzimy działanie użytkownika na liście faktur, z pewnością zauważymy ,że bardzo

często przechodzi on na jej koniec, W ten sposób może szybko sprawdzić jaki jest ostatni wykorzystany numer oraz czy faktury zachowują ciągłość numeracji. Przejście na koniec listy jest naturalnym odruchem, zwłaszcza przed wystawieniem nowej faktury. Poza tym zwykle interesują nas bardziej faktury wystawione przed kilkoma dniami niż kilkoma miesiącami. Uwagi te skłaniają do zastosowania następującego rozwiązania : po wyświetleniu listy wskaźnik rekordu powinien się ustawić na ostatniej fakturze, Aby zrealizować ten cel wystarczy skorzystać z metody Last komponentu DataSet, reprezentującego tabelę lub zapytanie

1. Kliknij przycisk View Form i wybierz z listy formularz FaktLForm
2. Gdy zaznaczony jest formularz, przejdź do karty Events Object Inspectora i dwukrotnie kliknij w polu zdarzenia OnShow. Dzięki temu przejście na koniec listy (gdy zostanie wywołany on z formularza głównego) ale nie po powrocie z formularz FakturaForm, ponieważ formularz listy nie jest wtedy ponownie wyświetlany, a tylko uaktywniony (nie zachodzi więc zdarzenie OnShow)
3. Uzupełnij procedurę obsługi zdarzenia:

```
procedure TFaktLForm.FormShow(Sender : TObject):
begin;
inherited;
FaktDModule.Table1.Last;
end;
```

4. Zapisz wprowadzone zmiany , skompiluj i uruchom aplikację

Po wyświetlenia formularza z listą faktur, aktywny jest ostatni rekord zestawu danych. Oczywiście metoda Last uwzględnia bieżący zestaw danych, a nie całą zawartość tabeli. Ponieważ w tym widoku wyświetlamy tylko faktury sprzedaży, nastąpi przejście do ostatniego rekordu tej grupy transakcji

Ostrzeżenie o niezapisanych modyfikacjach

Bardzo często zdarza się , że użytkownik zamyka formularz z danymi bez ich uprzedniego zatwierdzenia. Nie jesteśmy jednak w stanie stwierdzić ,czy podjął taką decyzję, ponieważ rezygnuje z wprowadzonych modyfikacji , czy też na skutek zwykłego przeoczenia. Jeżeli na przykład użytkownik wprowadza rekordy kolejnych dziesięciu kontrahentów, wstawienie następnego nowego rekordu automatycznie oznacza zatwierdzenie poprzedniego i zapamiętanie zmian w bazie danych. Jeżeli jednak po wprowadzeniu lub zmodyfikowaniu ostatniego rekordu użytkownik nie zatwierdzi zmian (np. kliknięciem przycisku Post na pasku nawigatora) i zamknie formularz oraz związany z nim zestaw danych, modyfikacje zostaną utracone bez żadnego ostrzeżenia. Aby tego uniknąć, przed zamknięciem formularza należy sprawdzać ,czy edytowane zestawy danych nie są w stanie dsInsert lub dsEdit. Spróbujmy teraz zdefiniować odpowiedni mechanizm zabezpieczający dla formularza FakturaForm

1. Przejdź do formularza FakturaForm
2. W Object Inspectorze dwukrotnie kliknij w polu zdarzenia OnClose. Jest ono wzbudzane w chwili zamykania formularza
3. Uzupełnij procedurę obsługi zdarzenia zgodnie z poniższym listingiem:

```
procedure TFakturaForm.FormClose(Sender : TObject , var Action :TCloseAction);
var
ZT1, ZT2 : boolean; {Zmienne ZT1 i ZT2 przyjmują wartość True, jeżeli odpowiednio w tabeli nagłówków faktury lub szczegółów faktury są jakieś niezatwierdzone zmiany}
begin
ZT1 := (FaktDModule.Table1.State in [dsEdit, dsInsert]);
ZT2 := (Table2.State in [dsEdit, dsInsert]);
if ZT1 or ZT2 then
case Application.MessageBox('Zapisać wprowadzone zmiany ?', 'Ostrzeżenie' ,
MB_YESNOCANCEL) of
IDYES : begin
if ZT1 then FaktDModule.Table1.Post
if ZT2 then Table2.Post;
end;
IDNO : begin
if ZT1 then FaktDModule.Table1.Cancel
```

```
if ZT2 then Table2.Cancel;
end;
IDCANCEL : abort;
end;
end;
```
Zmienne ZT1 i ZT2 wskazują czy w obydwu zestawach daych (z nagłówkami i szczegółami transakcji) wprowadzono jakieś modyfikacje. Modyfikowany rekord może być w stanie edycji (dsEdit) lub wstawiania (dsInsert). Jeżeli w którymkolwiek zestawie danych zostanie wykryty jeden z wymienionych stanów, na ekranie pojawi się stosowny komunikat. Do wyświetlenia komunikatu stosujemy metodę MessageBox obiektu Application. Metoda ta przyjmuje trzy parametry : tekst wyświetlaneg komunikatu, tytuł okna z komunikatem oraz zbiór przycisków. W naszym przykładzie użyliśmy przycisków Tak, Nie i Anuluj, co jest reprezentowane przez jedną wartość MB_YESNOCANCEL. Funkcja MessageBox zwraca rezultat w postaci numeru naciśniętego przycisku. Odpowiednie numery są zdefiniowane przez stałe, np. IDYES odpowiada naciśnięciu przycisku Tak. Zwrócony przez MessageBox wynik jest analizowany w instrukcji Case . Jeżeli użytkownik kliknął Tak, następuje zapisanie zmian w odpowiednich zestawach. Jeżeli kliknął Nie - zmiany są anulowane. Po zapisie lub anulowaniu zmian , obsługa zdarzenia jest kontynuowana, a więc następuje zamknięcie formularza. Jeżeli użytkownik wybrał Anuluj, zgłaszany jest tzw. cihcy wątek, powodujący przerwanie bieżącej ścieżki wykonawczej, a więc w tym przypadku przerwania obsługi zdarzenia zamykania. W rezultacie formularz nie zostanie zamknięty, a użytkownik powróci do edycji rekordu. Analogiczny mechanizm wprowadź w formularzach TowarForm, KontrahForm oraz KonfigForm. Ponieważ w formularzu KonfigForm dane nie są zapamiętywane w bazie danych , lecz w pliku INI, zamiast metody Post musisz zastosować metodę SaveToFile

Definiowanie pól wymaganych

Zwróćmy uwagę ,że w obecnym stanie aplikacji użytkownik może wydrukować fakturę, nawet jeżeli najważniejsze jej pola, takie jak numer faktury, nabywca, data sprzedaży itp. są puste. Taka sytuacja jest niedopuszczalna z punktu widzenia prawnego. Aby wymusić konieczność wypełnienia pola, należy ustawić właściwość Required odpowiedniego komponentu TField na True

1. Przejdź do modułu danych faktDModule
2. Dwukrotnie kliknij komponent Table1. Na ekranie pojawi się lista pól zdefiniowanych w tym zestawie danych
3. Zaznacz pole Numer i ustaw jego właściwość Required na True
4. Taki sam warunek ustaw dla pól Data_wystaw, Termin_trans, Termin_platn
5. Przejdź do formularza FakturaForm
7. Zaznacz wszystkie pola za wyjątkiem : IDTowUslug, Symbol, KWiU i ustaw ich właściwości Required na True

Kwota słownie

Nawet niezbyt wymagający księgowy zauważy ,że na wydruku faktury brakuje tak istotnego elementu, jak słowne przedstawienie kwoty. W krajach anglojęzycznych, gdzie nie występuje odmiana przez przypadki, problem ten jest stosunkowo prosty. Niestety historia obdarzyła nas podobno jednym z najbardziej skomplikowanych języków świata, co dodatkowo "uatrakcyjnia" rozwiązanie tego problemu. Oczywiście można sobie poradzić tak , jak postępuje wielu autorów oprogramowania , stosując dopuszczalny format skrócony, na przykład wyświetlając kwotę 145,24 jako sto*czt*pie*zł*dwa*czt*gr, ale w końcu chcemy , aby nasza aplikacja generowała profesjonalne wydruki. Poniżej przedstawiamy rozwiązanie , zakodowane w postaci funkcji KwotaNaSlownie. Większość komentarzy jest umieszczona w samym listingu

```
function KwotaNaSlownie(Kwota: extended) : string:
const
MAX_CYFR = 12; {maksymalna liczba cyfr w kwocie}
{tablica pozycji tysięcznych}
{1..4 : miliard, milion, tysiąc, złoty}
{1..3 : odmiany przez przypadki:
1 złoty, 2-4 złote, 5 złotych}
```

```
PozTys : array [1..4, 1..3] of string [20]
=(('miliard', 'miliardy', 'miliardów'),
('milion', 'miliony', 'milionów'),
('tysiąc', 'tysiące', 'tysięcy'),
('złoty', 'złote','złotych'));
{tablica nazw liczb}
{1-9 : kolejne cyfry} {0-2 : pozycja cyfry w liczbie}
CyfrSlow : array [1..9, 0..2[ of string[20]
= (('sto', 'dziesięć', 'jeden'), ('dwieście', 'dwadzieścia' , 'dwa'), ('trzysta', 'trzydzieści', 'trzy'),
('czterysta', 'czterdzieści' , 'cztery'), ('pięćset', 'pięćdziesiąt', 'pięć'), ('sześćset', 'sześćdziesiąt' ,
'sześć'), ('siedemset', 'siedemdziesiąt', 'siedem'), ('osiemset', 'osiemdziesiąt' , 'osiem'),
('dziewięćset', 'dziewięćdziesiąt', 'dziewięć'));
{nazwy liczb z przedziału 11-19}
Nascie : array [1..9] of string[20];
=('jedenaście', 'dwanaście', 'trzynaście', 'czternaście', 'piętnaście', 'szesnaście', 'siedemnaście',
'osiemnaście', 'dziewiętnaście');
var
CalkowSlow : string[MAX_CYFR]; {calkowita część konwertowanej liczy}
PomocSlow : string; {zmienna pomocnicza, przechowująca budowany napis}
i : byte ;
CzyNascie : boolean; {czy liczba z przedziału 11-19}
CzyZnaczaca : boolean; {czy cyfra znacząca}
Grosze : string[15]; {groszowa część kwoty}
begin
{Zapamiętanie części całkowitej liczby w tablicy znaków}
Str(Int(Kwota) : MAX_CYFR:0, CalkowSlow);
{wypełnienie zerami pozycji nieznaczących}
for i := 1 to MAX_CYFR do
if CalkowSlow[i] = ' ' then CalkowSlow[i] := '0'
else Break;
{Generowanie tekstu}
PomocSlow "= '';
CzNascie := false;
CzyZnaczaca := false;
for i := 1 to MAX_CYFR do begin
if (CalkowSlow[i] < > '0') or CzyZnaczaca then begin
CzyZnaczaca := true;
if CzyNascie theb {Wstawienie słownie liczby 11-19}
PomocSlow := PomocSlow + Nascie[StrToInt(CalkoSlow[i])];
{Sprawdzenie , czy liczba z przedziału 11-19}
if ((i mod 3) = 2) { i wskazuje cyfrę na pozycji dziesiątek}
and (CalkoSlow[i]='1') {wskazywaną cyfrą jest 1}
and (CalkowSlow[i+] < > '0') {następna cyfra nie jest zerem} then
CzyNascie := true;
{Sprawdzenie czy liczba słownie powinna wystąpić} if (not CzyNascie) {dla liczb z przedziału 11-
19 pierwsza cyfra jest ignorowana, wstawienie nastąpi dopiero w następnym przejściu pętli}
and (CalkowSlow[i] < > '0') {zero nie zmienia się na słownie} then
{wstawienie następnego słowa do ciągu}
PomocSlow := PomocSlow + CyfraSlo[StrToInt(CakowSlow[i]), (i+2) mod 3];
{wprowadzenie rzędu wielkości po trzech kolejnych cyfrach}
if (( i mod 3 = 0) {bieżąca cyfra jest jednostką w rzędzie}
and ( i < > MAX_CYFR) {bieżąca cyfra nie jest ostatnią}
{ostatnie 3 przetwarzane cyfry nie są zerami}
and (copy(CalkowSlow, i-2,3) < > '000')
then begin
```

```
{"tysiąc, milion" - dla końcówek 1, za wyjątkiem 11} if (CalkowSlow[i] = '1') and (not CzyNascie)
then
PomocSlow := PomocSlow } PozTys[i div 3, 1]
{"tysiące, milion" - dla końcówek 2,3,4 za wyjątkiem liczb z przedziału 11-19"}
else if (CalkowSlow[i] in ['2' , '3', '4']) and (not CzyNascie) then
PomocSlow := PomocSlow + PozTys[i div 3,2]
{"tysięcy, milionów - dla pozostałych końcówek}
else begin
PomocSlow := PomocSlow + PozTys[i div 3,3];
end;
CzyNascie := false;
end;
end;
{if CzyZnaczaca}
end;
{Dodanie końcówki "złoty"}
{Szczególnym przypadkiem jest kwota 1 zł}
if (Kwota >= 1.0) and (Kwota < 2.0) then
PomocSlow := PomocSlow + PozTys[4,1] {"złoty"}
else begin
{Ostatnia cyfra kwoty to 2,3 lub 4 i nie jest to liczba z przedziału 11-19}
if (CalkowSlow[MAX_CYFR] in ['2', '3', '4'])
and (not CzyNascie) then
PomocSlow := PomocSlow + PozTys[4,2] {"złote"}
else
{w pozostałych przypadkach}
PomocSlow := PomocSlow + PozTys[4,3] {"złotych'}
end
{Dodanie groszy w postaci nn/100}
Str(Kwota:15:2, Grosze);
Grosze : Copy(Grosze, Length(Grosze) -1 , 2);
KwotaNaSlownie := PomocSlow + ' ' + Grosze + '/100';
end;
```

W pierwszej części funkcji definiowane są tablice stałych będących różnymi odmianami liczebników oraz nazw rzędów wielkości. W dodatkowej tablicy o nazwie Nascie zostały zdefiniowane słowne określenia liczb z przedziału 11-19. Zwróć uwagę ,że liczby te w przeciwieństwie do pozostałych, chociaż składają się z dwóch cyfr, są reprezentowane przez jdenoczłonowe słowo (np. "pięćdziesiątpięć" ma dwa człony : "pięćdziesiąt" i "pięć", natomiast "piętnaście" tylko jeden człon) Funkcja jako parametr przyjmuje liczbę rzeczywistą. Całkowita część tej liczby jest zapisywana w tablicy znaków, a pozycje niewykorzystane są wypełniane zerami. (Ponieważ części groszowe można przedstawić cyfrowo, funkcja nie realizuje ich konwersji). Np. jeśli jako parametr zostanie podana liczba 1230,45, w tablicy CalkowSlow znajdzie się ciąg znaków 000000001230. Dzięki temu zabiegowi mamy pewność, że na pierwszej pozycji tego ciągu mamy cyfrę tysięcy , na drugiej - dziesiątek, na trzeciej - jedności, na czwartej znowu tysięcy itd. Dalsza część funkcji analizuje kolejne znaki ciągu. Początkowe cyfry to z reguły zera nieznaczące (w powyższym przykładzie jest ich 8), a więc są pomijane. Dopiero po napotkaniu pierwszej cyfry różnej od 0 zmienna CzyZnaczaca przyjmuje wartość True. Wszystkie następne cyfry (w tym również zera) są już znaczące. Co trzy cyfry do ciągu znaków dołączane jest słowo określające rząd wielkości (milirad, milion lub tysiąc), oczywiście w odpowiednim przypadku i liczbie. Po wyjście pętli głównej następuje wybranie odmiany słowa "złoty" na podobnych zasadach , jak odmiana liczb) oraz wyznaczenie wartości groszy i dołączenie ich do kwoty złotych. Przedstawiony algorytm być może nie jest optymalny, ale działa prawidłowo i pozwoli nam uzyskać napis na wydruku faktury:

1. Kliknij przycisk Add file to project
2. Na liście odszukaj plik FunFin.pas i kliknij Otwórz.

3. Kliknij przycisk View Unit i wybierz moduł FakturaQ
4.W klauzuli uses w sekcji implementation dodaj deklarację :
uses
IniFiles, Math, FunFin;
5. Przejdź do widoku formularza (klawisz [F12])
6. Poniżej etykiety DO ZAPŁATY umieść dwie kolejne etykiety TQRLabel
7. Zmień właściwość Caption pierwszej etykiety na Kwota słownie
8. Zmień właściwość Name drugiej etykiety na SlownieQRLabel
9. Zaznacz sekcję Summary i dwukrotnie kliknij w polu zdarzenia BeforePrint
10. Na końcu procedury obsługi zdarzenia (przed zamykającym poleceniem End) wpisz wiersz:
SlownieQRLabel.Caption:= KwotaNaSlownie(RazemNetto + RazemVAT);
11. Kliknij przycisk Save ALl, aby zapisać wprowadzone zmiany
12. Skompiluj i uruchom aplikację
13. Wyświetl podgląd wydruku faktury

Globalne właściwości aplikacji

Istnieje kilka tzw. obiektów jednowystąpieniowych (singleton objects), wspólnych dla całej aplikacji. Najciekawszym z nich jest obiekt Application, będący wystąpieniem klasy TApplication. Klasa ta posiada kilkadziesiąt przydatnych właściwości i metod, związanych przede wszystkim z tworzeniem , uruchamianiem i zwalnianiem aplikacji widzianym z poziomu systemu operacyjnego Windows. Zadania obiektu Application obejmuje m.in. : przetwarzanie komunikatów Windows, obsługę kontekstowego systemu pomocy oraz obsługę wyjątków z poziomu aplikacji. Obiekt Application jest tworzony automatycznie, czyli nie musimy umieszczać w aplikacji żadnego odpowiadającego mu komponentu

Ikona i tytuł

Ostatnio korzystaliśmy z metody MessageBox, umożliwiającej wyświetlenie komunikatu systemowego z odpowiednimi przyciskami, W tej sekcji wyjaśnimy jak za pomocą innych właściwości i metod obiektu Application modyfikować ikonę oraz pojawiający się pod nią tytuł aplikacji. Właściwości te możemy modyfikować bezpośrednio w kodzie aplikacji lub korzystając z polecenia Project Options:
1. W menu Project wybierz polecenie Options
2. Przejdź do zakładki Application
3. W polu Title wpisz Moje faktury. Tytuł ten będzie się pojawiał pod ikoną aplikacji. Wykonanie powyższej operacji spowoduje również automatyczne wstawienie w pliku Faktury.dpr wiersza kodu:
Application.Title := 'Faktury';
4. Kliknij przycisk Load Icon. Wybierz plik z rozszerzeniem .ico a następnie kliknij przycisk Otwórz

Dynamiczne tworzenie formularzy

Gdy umieszczamy w aplikacji nowy formularz, w pliku projektu automatycznie jest wstawiane wywołanie metody Application.Create dla tego formularza. Oznacza to ,że formularz zostanie utworzony w chwili uruchomienia aplikacji. Ponieważ do tej pory akceptowaliśmy te domyślne ustawienia, w module głównym projektu Faktury.dpr są inicjalizowane wszystkie formularze używane w aplikacji:

```
begin
Application.Initialize;
Application.Titlle := 'Faktury';
Application.CreateForm(TMenuGlForm, MenuGlForm);
Application.CreateForm(TKonfigForm, KonfigForm);
Application.CreateForm(TTowarDModule, TowarDModule);
Application.CreateForm(TTowarLForm, TowarLForm);
Application.CreateForm(TKontrahDModule, KontrhDModule);
Application.CreateForm(TKontrahLForm, KontrahLForm);
Application.CreateForm(TFaktDModule, FaktDModule);
Application.CreateForm(TFaktLForm, FaktLForm);
Application.CreateForm(TKontrahForm, KontrahForm);
Application.CreateForm(TTowarForm, TowarForm);
```

```
Application.CreateForm(TFakturaForm, FakturaForm);
Application.CreateForm(TLicznDModule, LicznDModule);
Application.CreateForm(TKontrLQReport, KOntrLQReport);
Application.CreateForm(TFakturaQForm, FakturaQForm);
Application.Run;
end;
```

Tworzenie wystąpienia formularza od razu na początku aplikacji ma swoje wady i zalety. Niewątpliwą zaletą jest to ,że w chwili wywołania formularz znajduje się już w pamięci, co znacznie przyspiesza jego wyświetlanie. Z punktu widzenia użytkownika ma to istotne znaczenie zwłaszcza wtedy gdy formularz jest wielokrotnie otwierany i zamykany. Z drugiej strony, tworzenie wystąpienia formularza w chwili uruchomienia aplikacji spowalnia proces (zwłaszcza gdy jednocześnie jest alokowana pamięć dla kilkudziesięciu formularzy). Jednak jeszcze ważniejszym powodem, dla którego racjonalnie powinniśmy przydzielać pamięć dla formularzy, jest oszczędność w gospodarowaniu zasobami systemowymi. Chociaż obecnie komputery cechują się bardzo dużymi mocami obliczeniowymi, nie możemy "egoistycznie" zakładać ,że nasza aplikacja będzie jedyną i tą najważniejszą w systemie. Jak zwykle w takiej sytuacji , gdy dążymy do osiągnięcia sprzecznych celów, staramy się znaleźć rozwiązanie kompromisowe. W tym przypadku powinniśmy zdecydować, które z formularzy są używane najczęściej, a które najrzadziej .Formularze z pierwszej grupy powinny być tworzone w chwili uruchomienia aplikacji i pozostawać w pamięci przez cały czas jej działania, natomiast te należące do drugiej grupy będą tworzone ad hoc, dopiero w momencie gdy zajdzie potrzeba ich użycia, a po zamknięciu - natychmiast zwalniane. Do drugiej grupy z pewnością możemy zaliczyć formularz KonfigForm oraz KontrLQReport (oraz w przyszłości większość formularzy, na których zostały umieszczone raporty o charakterze statystycznym. Na przykład formularz KonfigForm , na którym zapisane są dane konfiguracyjne, z pewnością będzie uruchamiany bardzo rzadko i użytkownik bez przeszkód pogodzi się nawet z kilkusekundową zwłoką podczas jego uruchamiania

1. W menu Project wybierz polecenie Options a następnie przejdź do zakładki Forms.
2. Za pomocą przycisku > przenieś z listy Auto-create forms na listę Available forms pozycje KonfigForm i KontrLQReport
3. Przejdź do widoku formularza MenuGlForm i odszukaj procedurę WyswietlKonfiguracje, a następnie zmodyfikuj ją zgodnie z poniższym :

```
procedure TMenuGlForm.WyswietlKonfiguracje;
begin
KonfigForm := TKonfigForm.Create(Self);
try
KonfigForm.ShowModal;
finally
KonfigForm.Free;
end;
end;
```

Procedura powoduje teraz utworzenie formularza za pomocą jego konstruktora. Parametr Self wskazuje właściciela tworzonego obiektu, a więc formularz MenuGlForm. Po zamknięciu formularza jest on natychmiast usuwany z pamięci. Zwróć uwagę ,że zastosowaliśmy tutaj klauzulę try … finally. W ten sposób nawet jeżeli wyświetlenie formularza nie powiedzie się , wykorzystywane przez niego zasoby pamięci zostaną prawidłowo zwolnione. Analogiczny kod wprowadź przed i po wywołaniu formularza KontrLQReport. Zwróć uwagę ,aby wywołanie konstruktora znalazło się przed wszystkimi odwołaniami do formularza KontrLQReport

Niestandardowa obsługa błędów aparatu bazy danych

Błędy czasu wykonania, w zasadzie wyjątki (expection) są nieodłącznym elementem aplikacji. Zadaniem programisty jest obsłużenie tych wyjątków za pomocą odpowiednich procedur. Jeżeli odpowiednio zdefiniujemy ten mechanizm, wyjątki przestaną być błędami a staną się źródłem cennych informacji. Szczególnie często wyjątki są zgłaszane w trakcie operacji na rekordach bazy danych - wiele z nich pojawia się , ponieważ dochodzi do próby naruszenia zasad integralności. Z pewnością zauważyłeś już ,że jeśli użytkownik spróbuje zdefiniować kontrahenta o kodzie, który już jest wykorzystywany w tabeli Kontrah, wówczas na ekranie pojawi się komunikat "Key

violation". Ostrzeżenie pojawi się również wtedy, gdy użytkownik spróbuje usunąć rekord z danymi kontrahenta, przy czym dla tego kontrahenta utworzono już jakąś fakturę. Pojawi się wówczas komunikat "Master had detai records. Cannot delete or modify". Być może takie komunikaty są przynajmniej częściowo zrozumiałe dla osoby mówiącej po angielsku, lecz polskiego użytkownika mogą co najwyżej wprawić w zakłopotanie. Aby temu zapobiec, należy przechwycić wyjątek , a następnie obsłużyć go w kodzie programu, co zaowocuje wyświetleniem bardziej czytelnego komunikatu i pozwoli podjąć dodatkowe działania. Wszystkie wyjątki są potomkami klasy Exception. Wyjątki o których teraz mówimy ,należą do podklasy EDBEngineError utworzonej na bazie klasy EDatabaseError, która z kolei wywodzi się z klasy Exception. Obsługa wyjątków klasy EDBEngineError może być wykonywana na wielu poziomach - na poziomie aplikacji, formularza lub lub samego obiektu DataSet, który był przyczyną wyjątku. Im niższy poziom obsługi , tym bardziej szczegółowe mogą być komunikaty generowane dla użytkownika. Na przykład jeśli wyjątek "Key violation" obsługujemy na poziomie aplikacji, wyświetlony komunikat może mieć tekst : "Ten identyfikator jest już wykorzystywany w innym rekordzie". Nie wiemy jednak (korzystając ze standardowych właściwości klasy EDBEngineError), w jakiej tabeli powstał ten wyjątek. Jeżeli jednak ten sam błąd przechwycimy na przykład w obiekcie Table udostępniającym tabelę Kontrah, możemy wygenerować o wiele bardziej czytelny komunikat. Przestawione poniżej czynności pokazują , jak zaimplementować obsługę wyjątków na poziomie aplikacji

1. Kliknij przycisk View Unit i wyświetl moduł kodu MenuGl.pas
2. Przejdź na początek formularza i do klauzuli uses, dołącz odwołanie do modułu BDE :

```
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, Menus,
ImgList, ComCtrls, ToolWin, DB, DBTables, BDE.
```

W tym module są zdefiniowane stałe, odpowiadające numeromo obsługiwanych błędów

3. W sekcji interfejsu, za istniejącymi nagłówkami procedur (przed słowem private) , umieść kolejne dwa nagłówki:

```
procedure KomunikatOBledzie (S: PChar);
procedure ObslugaBledow (Sender : TObject; E : Exception);
```

4. Przejdź na koniec modułu interfejsu (przed instrukcję end z kropką) i wpisz kod procedur:

```
procedure TMenuGlForm.KomunikatOBledzie( S: PChar);
begin
MessageBeep(MB_ICINERROR);
Aplication.MessageBox(S, 'Błąd', MB_ICONERROR + MB_OK);
end;
procdure TMenuGlFom.ObslugaBledow(Sender : TObject; E : Eception);
begin
if E is EDBEngineError and (EDBEngineError(E).ErrorCount > 0) then
case EDBEngineError(E).Errors[0].ErrorCode od
DBIERRP_KEYVIOLS:
KomunikatOBledzie('Nie można modyfikować identyfiaktora lub usuwać rekordu nadrzędnego' +
#13#10 + 'jeżeli jest on wykorzystywany w tabeli podrzędnej.');
else
Application.ShowEception(E);
end
else
Application.ShowEception(E);
end;
```

Procdura KomunikatOBledzie jest wykorzystywana do wyświetlenie odpowiedniego tekstu błędzie. Wykonanie tej procedury powoduje wygenerowanie dźwięku skojarzonego z komunikatem o błędzie (procdura MessageBeep) oraz wyświetlenie okienka dialogowego MessageBox z komunikatem, tytułem 'Błąd', ikoną błędu (parametr MB_ICONERROR) i przyciskiem OK (parametr MB_OK). W ten sposób przesłonimy działanie standardowej procedury ShowError wyświetlające komunikaty w języku angielskim. Działanie drugiej procedury jest nieco bardziej skomplikowane. Po pierwsze sprawdzamy czy zgłoszony wyjątek należy do klasy EDBEngineError.

Jeżeli tak, za pomocą instrukcji CASE sprawdzamy czy wyjątek ten jest obsługiwany przez naszą procedurę .Zastosowane tutaj stałe (DBIERR_KEYVIOL i DBIERR_DETAILRECORDSEXIST) pochodzą z modułu BDE. Po odnalezieniu właściwego kodu, uaktywniana jest procedura KomunikatOBledzie z odpowiednim tekstem komunikatu. Jeżeli wyjątek o tym kodzie nie jest obsługiwany przez procedurę , metoda Applicatin.ShowException wyświetla standardowy komunikat .Standardowy komunikat jest również wyświetlany, jeżeli zgłoszony wyjątek nie należy do klasy EDBEngineError
5. Przejdź do procedury obsługi zdarzenia OnCreate formularza MenuGlForm i na jej końcu wpisz wiersz : Application.OnException := ObslugaBledow; . W ten sposób procedura ObslugaBledow została przypisana zdarzeniu OnException obiektu Application. Dzięki temu będzie ona uruchamiana po każdym wystąpieniu wyjątku w obszarze całej aplikacji
6. Skompiluj i uruchom aplikację
7. Przejdź do formularza z listą kontrahentów
8. Wstaw nowy rekord i wpisz identyfikator kontrahenta , który jest już zdefiniowany w bazie danych a następnie spróbuj zatwierdzić cały rekord. Na ekranie pojawi się komunikat o błędzie
9. Spróbuj teraz usunąć identyfikator kontrahenta, dla którego istnieje już faktura. Również w tym przypadku powinien pojawić się odpowiedni komunikat

Obsługa błędów na poziomie obiektu DataSet

Jeżeli chcesz przechwycić i obsłużyć błąd na poziomie tabeli , musisz utworzyć procedury obsługi dla zdarzeń OnDeleteError, OnEditError, OnPostError i OnUpdateError. Na przykład procedura obsługi błędu , który wystąpił podczas usuwania rekordu kontrahenta, mogłaby mieć ogólną postać :

```
procedure TKontrahDModule.Table1DeleteError (DataSet : TDataSet; E : EDatabaseError; var
Action : TDataAction);
begin
{Kod obsługi błędu}
Action := daAbort;
end;
```

W miejscu {Kod obsługi błędu} należy zastosować rozwiązanie analogiczne, jak podczas obsługi błędu na poziomie aplikacji. Ustawienie parametru Action na daAbort spowoduje ,że wyjątek nie będzie dalej przetwarzany

Końcowe Podsumowanie

Na tym etapie zakończymy tworzenie naszej aplikacji. Czy można więc stwierdzić , że jest już gotowa? Na pewno nie. Brakuje w niej wielu elementów, które mogłyby się znaleźć w programie do fakturowania. Jednak zrealizowaliśmy podstawowe założenia, które postawiliśmy sobie na początku. Stworzyliśmy program, który umożliwia wystawianie i wydruk faktur w oparciu o dane o towarach i kontrahentach przechowywanych w bazie danych. Jeżeli chcesz , możesz teraz przystąpić do samodzielnej rozbudowy aplikacji, wzbogacając ją o kolejne funkcje, które twoim zdaniem mogą być przydatne