Praktyczna aplikacja bazodanowa : Program do fakturowania - Cz. I

STRUKTURA BAZY DANYCH

Przed rozpoczęciem tworzenia formularza aplikacji musimy najpierw zdefiniować strukturę danych w których będą gromadzone informacje przetwarzane przez program. Dane w bazach są przechowywane w tabelach. Każda tabela składa się z rekordów, które z kolei zwierają pola określonego typu. W rekordach (wierszach) znajdują się dane opisujące określony element rzeczywistości . Poszczególne pola tworzą kolumny tabeli, stąd też czasami obydwu pojęć używa się zamiennie. Każde z pól [kolumn] w tabeli bazy danych musi być określonego typu - jest to jeden z mechanizmów zapewnienia integralności danych. W środowisku Delphi narzędziem do tworzenia tabel jest program Database Desktop (DBD). Pozwala on między innymi na definiowanie tabel w formacie Paradox, z którego będziemy korzystać. Format Paradox jest zalecany w przypadku tworzenia aplikacji jednowarstwowych (nie typu klient-serwer). Tabele w tym formacie są najlepiej obsługiwane przez komponenty dostępu do danych Delphi i kontrolki z palety BDE.

W tabelach Paradox można stosować pola następujących typów :

Typ : Alpha

Symbol : A

Rozmiar : 1 - 125

Opis : Wartości alfanumeryczne (łańcuchy tekstowe). Mogą to być litery, liczby, symbole specjalne i inne drukowalne znaki kodu ASCII. Wartość określającą rozmiar deklaruje się w momencie tworzenia tabeli

Typ : Number

Symbol : N

Opis : Zmiennoprzecinkowa wartość liczbowa

Typ : Money

Symbol : $

Opis : Pole tego typu , podobnie jak pole Number, może zawierać tylko liczby. Różnica między nimi jest taka ,że to ostatnie jest domyślnie formatowane w zapisie walutowym (zgodnie z ustawieniami formatu waluty w systemie Windows)

Typ : Short

Symbol : S

Opis : Pole do przechowywania krótkich wartości całkowitych (z przedziału od -32 767 do 32 767)

Typ : LongInteger

Symbol : I

Opis : Pole do przechowywania długich wartości całkowitych (z przedziału od -2 147 483 do 2 147 483 647)

Typ : BCD

Symbol : #

Rozmiar : 0-32

Opis : Pola liczb dziesiętnych kodowanych binarnie BCD (Binary Coded Decimal). Stosowany między innymi w obliczeniach o wysokim poziomie dokładności

Typ : Date

Symbol : D

Opis : Pole daty. Wpisując datę w takim polu, nie musimy sprawdzać jej poprawności

Typ : Time

Symbol : T

Opis : Pole godziny; ma analigiczne właściwości jak pole daty

Typ : Timestamp

Symbol : D

Opis : Pole "datownika" ; połączenie daty i godziny

Typ : Memo

Symbol : M

Rozmiar : 1-240

Opis : To pole służy do przechowywania dużych informacji tekstwowych (które nie mieszczą się w 255 bajtach pola typu Alpha). Teoretycznie, można w nim przechowywać dane dowolnej długości - jedynym ograniczeniem jest ilość wolnego miejsca na dysku. Mimo to dla pola Memo określa się rozmiar - są to bajty przechowywane wraz z tabelą, w któ®j zdefiniowano to pole. Pozostałe bajty są przechowywane w pliku o rozszerzeniu .MB i nazwie takiej samej jak nazwa tabeli

Typ : Formated Memo

Symbol : F

Rozmiar : 0-240

Opis : Przypomina pole typu Memo z tą różnicą, że zapisany w nim tekst może być sformatowany.

Typ : Graphic

Symbol : G

Rozmiar : 0 -240

Opis : Pole do przechowywania obrazów w formatach .BMP, .PCX, .TIF, .GIF, oraz .EPS

Typ : OLE

Symbol : O

Rozmiar : 0 -240

Opis : To pole umożliwia przechowywanie danych różnych typów, takich jak obrazy, dźwięki, dokumenty tekstowe itp. Nie trzeba określać rozmiaru tego pola, ponieważ jego zawartość jest przechowywana w oddzielnych plikach

Typ : Logical

Symbol : L

Opis : Pole logiczne. Domyślnie może przybierać wartość True lub False

Typ : AutoIncrement

Symbol : +

Opis : Pole, którego wartość jest automatycznie zwiększana po wstawieniu po wstawieniu kolejnego rekordu. Wartość ta nie może być edytowana. Pole to pozwala uzyskać niepowtarzalność rekordów - w danej tabeli nie znajdą się dwa rekordy o takiej samej wartości pola Autoincrement

Typ : Binary

Symbol : B

Rozmiar : 0 - 240

Opis : Pole binarne, służące do przechowywania danych, których nie jest w stanie zinterpretować program Database Desktop. Nie trzeba określać rozmiaru tego pola, ponieważ jego zawartość jest przechowywana w oddzielnych plikach

Typ : Bytes

Symbol : Y

Rozmiar : 1 -255

Opis : Pole bajtowe, służące do przechowywania danych, których nie jest w stanie zinterpretować program Database Desktop

Rozmiar pola Memo, Formated Memo, Grpahic, OLE oraz Binary określa liczbę bajtów przechowywanych w tabeli. Jeżeli np. rozmiar pola Memo wynosi 100, wówczas 100 pierwszych bajtów tego pola będzie przechowywanych w tabeli wraz z innymi danymi, natomiast reszta zawartości tego pola zostanie zapisana jako plik o rozszerzeniu .MB. Jeżeli przewidujesz ,że znacząca większość wartości w tym polu będzie zabierała mniej niż 240 bajtów, lub planujesz wykorzystywać początkowe bajty pola (np. do sortowania), określ odpowiedni rozmiar - w ten sposób prawdopodobnie przyspieszysz działanie aplikacji

Plik bazy danych w formacie Paradox

Baza danych w formacie Paradox składa się z wielu plików. Każda utworzona tabela jest zapisywana w oddzielnym pliku. Osobno są również zapisywane poszczególne indeksy i plik z warunkami poprawności. Rozwiązanie takie ma swoje zalety i wady .Dzięki podziałowi bazy danych na pliki mamy do nich większy dostęp z poziomu systemu operacyjnego i możemy w wygodny sposób kontrolować ich lokalizację oraz wykonywać różnego rodzaju operacje (usuwanie zawartości pojedynczej tabeli itp.) Z drugiej strony , baza danych jest w znacznym stopniu uzależniona od działania systemu operacyjnego, co oznacza ,że aparat bazy danych ma mniejsze szanse na zastosowanie własnych mechanizmów zarządzania i optymalizacji , które odgrywają coraz większą rolę w nowoczesnych

systemach zarządzania relacyjnymi bazami danych. Poniżej mamy opisane rozszerzenia plików powstających podczas tworzenia tabel w formacie Paradox:

DB : plik z danymi

MB : plik z danymi typu Memo, Formated Memo lub Graphic

PX : plik z indeksem (kluczem) podstawowym

Xnn, Ynn : Pliki z prostymi indeksami dodatkowymi

XGn, YGn : Pliki ze złożonymi indeksami dodatkowymi

VAL : Plik z regułami poprawności oraz integralności referencyjnej

Tworzymy pierwszą tabelę

Po wstępie teoretycznym możemy przystąpić do realizacji pierwszego zadania praktycznego. Spróbujemy utworzyć tabele ,na które będzie operować nasza aplikacja. Rozpoczniemy od najprostszej z nich, przeznaczonej do gromadzenia danych kontrahentów.

Tabela kontrahentów

1.W menu File, New wybierz Table. Pojawi się okno dialogowe Create Table

2.Upewnij się ,że polu Table Type zaznaczona jest opcja Paradox 7, a następnie kliknij przycisk OK. Na ekranie pojawi się okno dialogowe Create Paradox 7 Table. Możemy teraz przystąpić do definiowania pól tabeli. Program Database Desktop umożliwia również definiowanie tabel w innych formatach, na przykład dBAse IV, Oracle, Informix czy MSSQL. Ponieważ tworzymy prostą aplikację jednowarstwową, format Pardox wystarcza całkowicie. Ponadto użycie tego formatu zapewnia największą swobodę w korzystaniu z komponentów Delphi.

3.Gdy zaznaczone jest pierwsze pole w kolumnie Field Name, wpisz IDKontrah i naciśnij klawisz [Tab]. Zostanie zaznaczone pole Type

4.W polu Type wpisz A i naciśnij klawisz [Tab]. Identyfikator kontrahenta (pole IDKontrah) będzie typu znakowego, umożliwi to przypisywanie kontrahentom zarówno symboli liczbowych , jak i nazw symbolicznych (skróconych)

5.W pole Size wpisz 12 i naciśnij [Tab]. Wartość ta oznacza ,że identyfikator klienta nie może być dłuższy niż 12 znaków

6. W pole Key wpisz * . Gwiazdka (*) oznacza ,że to pole należy do indeksu podstawowego. Indeks podstawowy decyduje o uporządkowaniu rekordów w tabeli. Wynika z tego ,że tworzona tabela kontrahentów będzie porządkowana według ich identyfikatorów. Podzbiór pól wchodzących w skład klucza podstawowego musi być niepowtarzalny. W naszym przypadku, ponieważ do indeksu podstawowego tej tabeli będzie należeć tylko pole IDKontrah, wszystkie jego wartości w tej tabeli muszą być unikatowe. Na szczęście zadba o to sam aparat bazy danych - jeżeli w trakcie działania programu użytkownik spróbuje wprowadzić kontrahent o kodzie, który jest już używany w innym rekordzie , pojawi się komunikat o błędzie i nowy rekord nie zostanie zapisany w bazie. Zadaniem programisty pozostaje jedynie odpowiednie obsłużenie takiego błędu - użytkownik powinien zostać poinformowany o zaistniałej sytuacji, by mógł na nią właściwie zareagować.

7.Naciśnij klawisz [Tab]. Do tabeli zostanie wstawiony nowy wiersz i zaznaczona będzie kolumna Field Name

8.Posługując się poniższa tabelą , zdefiniuj pozostałe pola tabeli:

Nazwa pola : Nazwa

Typ : A

Rozmiar : 80

Nazwa pola : NIP

Typ : A

Rozmiar : 13

Nazwa pola : Miasto

Typ : A

Rozmiar : 80

Nazwa pola : Ulica

Typ : A

Rozmiar : 80

Nazwa pola : Bank

Typ : A

Rozmiar : 40

Nazwa pola : Konto

Typ : A

Rozmiar : 40

Nazwa pola : Telefon

Typ : A

Rozmiar : 40

Nazwa pola : Telefon_kom

Typ : A

Rozmiar : 20

Nazwa pola : Kontakt

Typ : A

Rozmiar : 80

Nazwa pola : Odbier_fakture

Typ : A

Rozmiar : 80

Definiując nazwy pól, staraj się unikać stosowania polskich znaków oraz odstępów w nazwach. Chociaż Database Desktop i Delphi akceptują takie ustawienia, możesz mieć wiele kłopotów, gdy w przyszłości zajdzie potrzeba połączenia twoje bazy danych z jakimś innym systemem. Z tego samego względu staraj się również tworzyć stosunkowo krótkie nazwy pól - niektóre starsze systemy baz danych mają limitowane długości nazw. Oczywiście skracanie nazwy pola nie powinno się odbywać kosztem jej czytelności - żaden programista nie ucieszy się , gdy będzie musiał napisać aplikację operująca na tabeli, w której pola nazywają się np. P1, P2, W itp. Jeśli chcesz używać nazwy wieloczłonowej, poszczególne jej elementy połącz znakami podkreślenia

9.Kliknij przycisk Save As. Pojawi się okno dialogowe Save Table As

10.W polu Nazwa pliku wpisz Kontrah, a następnie kliknij przycisk Zapisz

W przypadku nazw tabel zaleca się stosowanie podobnych zasad ,jak przy definiowaniu nazw pól tabeli - należy unikać stosowania znaków diakrytycznych i odstępów. Dodatkowo warto stosować nazwy nie dłuższe niż 8 znaków. Ułatwi to zarządzanie tabelami w aplikacjach i ich integrację z innymi systemami baz danych

Definiowanie warunków poprawności pól tabeli

Zanim zapiszemy zdefiniowaną tabelę Kontrah, wprowadzimy jeszcze kilka warunków poprawności (validity checks) .Pozwalają one dodatkowo zawęzić przedział danych, które można wprowadzać w poszczególnych polach. Warunki te wprowadzimy w polu IDKontrah - chcemy ,aby wpisywane litery były zawsze wielkie.

1.W menu Tools, Utilities wybierz polecenie Restructure

2.W oknie dialogowym Select File zaznacz plik Kotrah i kliknij przycisk Otwórz. Na ekranie pojawi się okno dialogowe Restructure Paradox 4 Table, przedstawiające strukturę tabeli Kontrah. Podczas zapisywania struktury tabeli program Database Desktop analizuje zastosowane typu pol i wybiera jak najprostszy format zapisu. W przypadku tabeli Kontrah wystarczył zapis w formacie Paradox 4. Mechanizm ten pozwala na uzyskanie zgodności z aplikacjami starszego typu , przy jednoczesnym zachowaniu pełnej funkcjonalności tabel (wcześniejsze wersje formatu Paradox są zgodne z nowszymi)

3.Upewnij się ,że zaznaczone jest pole IDKontrah oraz, że na liście rozwijanej Table Properties wybrana jest opcja Validity Chekcs, a następnie zaznacz pole wyboru Required Field. Zaznaczenie tej opcji powoduje ,że wpisanie wartości we wskazanym polu będzie wymagane.

4.Kliknij w polu tekstowym Picture i wpisz w nim *!. Wykrzynik oznacza ,że wszystkie wpisane w tym polu znaki będą przekształcane na wielkie litery. Gwiazdka oznacza ,że umieszczony za nią znak może wystąpić wielokrotnie. W rezultacie uzyskujemy maskę, która pozwala na wprowadzenie dowolnej liczby znaków, przy czym znaki alfabetu będą zapisywane wielkimi literami. Pole tekstowe Picture oferuje o wiele więcej możliwości. Jeżeli chcesz je przetestować, kliknij znajdujący się poniżej przycisk Assist. Pojawi się wówczas okno dialogowe Picture Assistance. Możesz skorzystać z przykładowych masek zdefiniowanych na liście rozwijanej Sample pictures lub utworzyć nową maskę w polu Picture. Omawiane okno dialogowe pozwala również na natychmiastowe przetestowanie utworzonej maski. Wystarczy przejść do pola Sample value i wpisać jakąś wartość. Jeżeli jest ona niezgodna z maską, zostanie wygenerowany sygnał dźwiękowy i pojawi się komunikat Picture does not accept value. Poniższa tabela przedstawia pełny wykaz symboli stosowanych w maskach :

Symbol : #

Opis : Cyfra

Symbol : ?

Opis : Dowolna litera (mała klub wielka)

Symbol : &

Opis : Dowolna litra zamieniana na wielką

Symbol : ~

Opis : Dowolna litera zamieniana na małą

Symbol : @

Opis : Dowolny znak

Symbol : !

Opis : Dowolny znak, litery zamieniane na wielkie

Symbol : ;

Opis : (średnik) Następny znak będzie traktowany dosłownie, a nie jako symbol maski

Symbol : *

Opis : Dowolna liczba powtórzeń następnego znaku (najczęściej jednego z symboli maski)

Symbol : [abc]

Opis : Opcjonale znaki : a,b lub c]

Symbol : [a,b,c]

Opis : Opcjonalne znaki : a,b lub c

Na przykład maska do wprowadzania kodu pocztowego mogłaby wyglądać następująco :

####

Masek nie należy jednak nadużywać. Zbytnie ograniczenie zakresu danych , które może wprowadzić użytkownik, czasami staje się dla niego kłopotliwe. Rozważmy przedstawiony przed chwilą przykład kodu pocztowego. Jeżeli w tabeli z takim polem zapisujemy tylko adresy krajowe, wszystko będzie w porządku. Co jednak zrobić, jeżeli użytkownik zechce wprowadzić kod pocztowy kraju, w którym stosowane są oznaczenia 6 cyfrowe lub oprócz cyfr występują również litery? Na tym etapie projektowania bazy danych pojawia się wiele podobnych pytań : czy zastosowana maska nie jest zbyt restrykcyjna, czy długość pola jest wystarczająca? Szukając na nie odpowiedzi ,trzeba wykazać się dużą dalekowzrocznością. Jeżeli mamy wątpliwości, warto zakładać na wyrost. Na przykład skoro nie ma przeciwwskazań, by w tabeli były gromadzone również adresy zagraniczne, dlaczego ograniczać format kodu pocztowego? Nad każdym z takich pytań trzeba się dobrze zastanowić, gdyż poprawki, wprowadzane w dalszych fazach pracy, będą o wiele bardziej kosztowne. Maskę wprowadzania można również zdefiniować w kontrolkach tyupu TDBEit, umieszczonych na formularzach. Jeżeli to jednak możliwe, odpowiednie ograniczenia należy wprowadzić już na poziomie struktur danych, a nie dopiero w aplikacji klienckiej. Sprzyja to zmniejszeniu rozmiarów kod, co upraszcza logikę aplikacji i zmniejsza ryzyko popełnienia ewentualnych błędów. Oprócz określenia maski można również

definiować dodatkowe warunki poprawności pól. Różnią się one w zależności od typu pola. Na przykład dla pola tekstowego (typ A) można określić wartość minimalną(Minimum value), maksymalną (Maximum value) oraz domyślną (Default value) . Opcje te są dostępne z prawej strony okna dialogowego Restructure/Create Table , gdy na liście rozwijanej Table Properties aktywna jest opcja Validity Checks

5.Kliknij przycisk Save.Pojawi się okno dialogowego Restructur Warning, Okno to jest wyświetlane, gdy zmiany wprowadzone w strukturze tabeli mogą wpłynąć na zawarte w niej dane. W naszym przypadku wprowadziliśmy warunek poprawności wymuszający stosowanie wielkich liter w identyfikatorach klientów. W związku z tym musimy zadecydować czy dotychczas zgromadzone w tabeli dane (oczywiście w tej chwili jest ona jeszcze pusta) będą zmodyfikowane zgodnie z wprowadzonym ograniczeniem. Jeżeli chcesz uzyskać wspomniany efekt zaznacz opcję Yes

6.Kliknij przycisk OK. Zmiany wprowadzone w strukturze tabeli Kontrah zostaną zapisane

Tworzymy pozostałe tabel

Po utworzeniu pierwszej tabeli z pewnością nie będziesz miał trudności z wprowadzeniem następnej definicji Oto wykaz tabel, które musisz zdefiniować, zanim przystąpisz do tworzenia aplikacji. Postępuj zgodnie z opisem definiowania nowej tabeli , podanym przy "Tabeli kontrahentów"

Tabela towarów i usług

W tej tabeli będzie przechowywany wykaz towarów i usług sprzedawanych przez naszą firmę

1.Zdefiniuj tabelę TowUslug zgodnie z poniższym opisem

Nazwa pola : IDTowUslug

Typ : A

Rozmiar : 12

Klucz : *

Nazwa pola : Nazwa

Typ : A

Rozmiar : 80

Klucz :

Nazwa pola : KWiU

Typ : A

Rozmiar : 30

Klucz :

Nazwa pola : J_miary

Typ : A

Rozmiar : 10

Klucz :

Nazwa pola : Stan

Typ : N

Rozmiar :

Klucz :

Nazwa pola : Stan_min

Typ : N

Rozmiar :

Klucz :

Nazwa pola : Cena_zak

Typ : $

Rozmiar :

Klucz :

Nazwa pola : Cena_netto

Typ : $

Rozmiar :

Klucz :

Nazwa pola : St_VAT

Typ : A

Rozmiar : 2

Klucz :

W omawianej tabeli wykorzystujemy , oprócz pola znakowego także dwa inne typu pól : N - pole numeryczne oraz $ - pole walutowe .W zasadzie obydwa pola służą do przechowywania wartości zmiennoprzecinkowych. Podstawowa różnica między nimi polega na tym ,że pole typu Currency ($) jest wyświetlane w formacie walutowym - z odstępem co trzy pozycje dziesiętne, dwoma miejscami po przecinku i symbolem waluty. Domyślnie do wyświetlania stosowany jest format waluty zdefiniowany w systemie Windows

2.W polu IDTowUslug definiuj maskę w postaci *!

3.Kliknij przycisk SaveAs , w polu Nazwa pliku wpisz TowUslug i kliknij przycisk Zapisz

Tabela nagłówka transakcji

Tabele Kontrah i TowUslug utworzone w powyżej, chociaż bardzo ważne, mają charakter pomocniczy. Kluczowe informacje związane ze sprzedażą będą jednak przechowywane w dwóch innych tabelach, które utworzymy za chwilę. Pierwsza z nich będzie zawierać dane nagłówka faktury, takie jak jej numer, data wystawienia, identyfikator kontrahenta itp. W drugiej znajdują się rekordy związane z poszczególnymi sprzedanym towarami lub usługami. Rozpoczniemy od utworzenia tabeli nagłówka faktury.

Zdefiniuj tabelę TrangNagl, zgodnie z powyższym opisem:

Nazwa pola : IDTrangNagl

Typ : +

Rozmiar :

Klucz : *

Nazwa pola : Typ

Typ : A

Rozmiar : 5

Klucz :

Nazwa pola : Seria

Typ : A

Rozmiar : 12

Klucz :

Nazwa pola : Numer

Typ : I

Rozmiar :

Klucz :

Nazwa pola : Dopisek

Typ : A

Rozmiar : 12

Klucz :

Nazwa pola : Data_wystaw

Typ : D

Rozmiar :

Klucz :

Nazwa pola : Data_trans

Typ : D

Rozmiar :

Klucz :

Nazwa pola : Term_platn

Typ : D

Rozmiar :

Klucz :

Nazwa pola : Forma_platn

Typ : A

Rozmiar : 12

Klucz :

Nazwa pola : IDKontrah

Typ : A

Rozmiar : 12

Klucz :

Nazwa pola : Uwagi

Typ : M

Rozmiar : 80

Klucz :

Tej tabeli musimy poświęcić nieco więcej uwagi. Pole IDTranNagl, jak można się domyślić, będzie zawierało niepowtarzalny identyfikator każdego rekordu nagłówka transakcji. Jest ono typu AutoIncrement (o czym informuje znak "+"), a więc o automatyczne zwiększanie jego zawartości zadba sam aparat bazy danych . Polu temu jest przypisywany następny wolny numer wyznaczany na podstawie wewnętrznego licznika tabeli. Numer raz wykorzystany nigdy nie jest używany powtórnie. Jeżeli na przykład do tabeli TranNagl wprowadzimy pięć rekordów i w polu IDTranNagl uzyskują one kolejno wartości 1,2,3,4 i 5, a następnie usuniemy rekordy 4 i 5, to po utworzeniu nowego rekordu uzyska on wartość 6. Pole Typ definiujemy nieco na wyrost, gdyż będzie ono przeznaczone do określania typu transakcji. W naszym programie co prawda ograniczymy się do transakcji będących dokumentami sprzedaży, czyli fakturami. Nic nie stoi jednak na przeszkodzie, aby w przyszłości tych samych tabel użyć do gromadzenia dokumentów zakupu, dokumentów wydania

(WZ) itp. Rodzaj transakcji, którego dotyczy określony rekord, będzie wskazywany właśnie przez pole Typ. Pola Seria, Numer i Dopisek w obrębie danego typu transakcji powinny tworzyć niepowtarzalne oznaczenie transakcji, na przykład "FA/31/2012" lub "FA/124/C" .Więcej o tych polach napiszemy później. Kolejne trzy pola nie powinny przysporzyć problemu .Są one typu Date i będą w nich przechowywane odpowiednio : data wystawienia dokumentu (Data_wystaw), data sprzedaży (Data_trans) oraz termin płatności (Term_platn). W polu IDKontrah znajdzie się identyfikator kontrahenta, którego dotyczy transakcja. Zwróć uwagę ,że nazwa tego pola jest taka sama ,jak nazwa indeksu podstawowego w tabeli Kontrah. Nie jest to żadna reguła, lecz dobra praktyka programistyczna - w ten sposób sygnalizujemy ,że obydwa pola są ze sobą powiązane relacją. Oczywiście ,ze zbieżności nazw w praktyce nie nie wynika - konieczne jest jeszcze zdefiniowanie odpowiedniej zależności (relacji) między tabelami. Ostatnie pole, Uwagi, jest przeznaczone do przechowywania informacji niezwiązanych bezpośrednio ze sprzedawanym towarem. Może się w nim znaleźć na przykład numer zamówienia, dodatkowe informacje dotyczące dostawy lub nota o naliczaniu odsetek w przypadku zwłoki z płatnością. Pole to jest typu Memo, co oznacza ,że ilość zawartych w nim danych jest praktycznie nieograniczona.

Definiowanie indeksu dodatkowego w tabeli nagłówka transakcji

Aby uzyskać pewność ,że w tabeli nagłówków transakcji nie pojawią się dwa rekordy o takim samym typie, serii, numerze i dopisku, zdefiniujemy indeks dodatkowy (secondary index) bez powtórzeń .(Indeksy dodatkowe, w przeciwieństwie do indeksów podstawowych, mogą być z powtórzeniami lub bez)

1.W menu Tools, Utilities wybierz polecenie Restructure

2.Zaznacz plik TranNagl i kliknij przycisk Otwórz. Pojawi się okno dialogowe Restructure Table przedstawiające definicję tabeli TranNagl

3.Na liście Table properties z prawej strony wybierz opcję Secondary Indexes

4.Kliknij przycisk Define. Na ekranie pojawi się okno dialogowe Define Secondary Index

5.Na liście z lewej strony dwukrotnie kliknij pole Typ. Pole to zostanie przeniesione na listę Indexed fields

6.W ten sam sposób przenieś na listę pól indeksu kolejno pola Seria, Dopisek i Numer

7.Zaznacz opcję Unique .Zaznaczenie tej opcji spowoduje powstanie indeksu bez powtórzeń

8.Kliknij przycisk OK. Pojawi się okno dialogowe Save Index As

9.W polu Index name wpisz dokument, a następnie kliknij przycisk OK. Ponownie aktywne stanie się okno Restructure Table. Na liście z prawej strony widoczna będzie nazwa zdefiniowanego indeksu.

10.Kliknij przycisk Save

Utworzony przed chwilą indeks da nam również inną korzyść, mianowicie w prosty sposób będziemy mogli sortować i filtrować dokumenty określonych typów. Z tego względu istotna jest kolejność, w jakie te pola zostały umieszczone w definicji indeksu. Dotyczy to zwłaszcza pola Typ. Dzięki temu , że jest ono na pierwszym miejscu, znacznie uprości się filtrowanie danych. Jeżeli na przykład wszystkie dokumenty sprzedaży będą miały w tym polu wpisaną wartość SPRZ a dokumentu zakupu wartość ZAK, odfiltrowanie poszczególnych kategorii dokumentów nie przysporzy problemu .Dzięki temu użytkownik będzie widział tylko dokumenty sprzedaży lub tylko dokumenty zakupu. Gdyby to pole znalazło się na dalszej pozycji , konieczne byłoby utworzenie dla niego indeksu dodatkowego (secondary index). W przeciwnym razie w trakcie filtrowania tworzony byłby indeks doraźny lub następowałoby skanowanie całej tabeli - gdy w tabeli będzie już wiele danych, mogłoby to powodować znaczne spowolnienie programu

Tabela pozycji transakcji

Pozostało nam jeszcze zdefiniowane w tabeli TranSzcz, w której gromadzone będą rekordy dotyczące poszczególnych pozycji transakcji. Zdefiniuj tabelę TranSzcz zgodnie z poniższym opisem :

Nazwa pola : IDTranNagl

Typ : I

Rozmiar :

Klucz : *<br

Nazwa pola : IDTranSzcz

Typ : +

Rozmiar :

Klucz : *<br

Nazwa pola : IDTowUslug

Typ : A

Rozmiar : 12

Klucz : <br

Nazwa pola : Nazwa

Typ : A

Rozmiar : 80

Klucz : <br

Nazwa pola : KWiU

Typ : A

Rozmiar : 30

Klucz : <br

Nazwa pola : J_miary

Typ : A

Rozmiar : 10

Klucz : <br

Nazwa pola : Ilosc

Typ : N

Rozmiar :

Klucz : <br

Nazwa pola : Cena_jedn

Typ : $

Rozmiar :

Klucz : <br

Nazwa pola : St_VAT

Typ : A

Rozmiar : 2

Klucz :

Indeksem podstawowym tej tabeli są pola IDTranNagl i IDTranSzcz. Pierwsze z nich wskazuje nagłówek transakcji, do której należą poszczególne pozycje. Razem z pola te muszą tworzyć unikatową parę - wymaga tego ograniczenie klucza podstawowego. Ponieważ pole IDTranSzcz jest typu AutoIncrement , już samo to pole zapewnia niepowtarzalność wartości klucza. Kolejne pole to IDTowUslug. Jak można się domyślić, znajdziesz w nim identyfikator towaru lub usługi pobrany z Tabeli TowUslug. Uważny czytelnik dostrzeże tutaj pozorną niekosekwencję. Podczas definiowana tabeli TranNagl wystąpiła analogiczna sytuacja - w nagłówku transakcji pojawił się identyfikator kontrahenta, lecz nie było w nim już innych pól związanych z kontrahentem. W tym przypadku jednak, oprócz identyfikatora występuje również nazwa, symbol KWiU (Klasyfikacja Wyrobów i Usług) oraz jednostka miary. Zgadza się , w tej tabeli pojawia się pewna nadmiarowość, jest ona jednak zamierzona .Dzięki niej użytkownik będzie mógł umieszczać na fakturze pozycje pobrane z tabeli TowUslug oraz wpisywać inne pozycje (dotyczy to zwłaszcza usług), które nie zostały zdefiniowane w tej tabeli. Zwróćmy również uwagę na fakt ,że w powyższej definicji brakuje takich pól jak cena brutto oraz kwota podatku VAT. Nie musimy zapamiętywać wartości tych pól w tabeli, ponieważ można je w każdej chwili wyliczyć. Dzięki temu unikniemy przechowywania nadmiarowych danych w tabeli. Brak wspomnianych pól ułatwia również zachowanie integralności danych - wyobraźmy sobie sytuację ,w której istnieją pola Cena_jedn, St_VAT, Ilosc i Cena_brutto. Na skutek błędu w programie mogłoby dojść do sytuacji , że cena brutto nie zostanie zaktualizowana wraz ze zmianą jednej z trzech poprzednich wartości - innymi słowy doszłoby do utraty integralności danych w bazie. Likwidowanie nadmiarowości danych nazywane jest normalizacją bazy danych. Jest to dosyć szerokie zagadnienie. Jednak, każdy "szanujący się" projektant baz danych powinien się z nim zapoznać i potrafić doprowadzić bazę danych przynajmniej do trzeciej postaci normalnej (3NF) .Czasami się zdarza ,że celowo unikamy normalizacji bazy danych a nawet denoromalizujemy ją (z postaci znormalizowanej wracamy do postaci nieznormalizowanej). Ten odwrotny proces stosuje się ,gdy bardziej niż na jakości struktur danych zależy nam na czasie dostępu do nich - niestety im bardziej znormalizowana jest baza, tym więcej czasu zabiera przetwarzania danych, gdyż w zapytaniach trzeba wykonywać wiele operacji złączeń i obliczeń. Prostszym od normalizacji sposobem uzyskiwania poprawnych struktur baz danych jest modelowanie jednostka - związek (entity - relation), nazywane również modelowaniem ER

Tabela stawek podatku VAT

Z poprzedniego podrozdziału wynika ,że wystarczy w tabeli TranSzcz przechowywać odpowiedni procent podatku , a następnie używać go do obliczania kwoty podatku oraz wartości brutto. Stwierdzenie to byłoby prawdziwe, gdyby wszystkie stawki podatku VAT były wyrażone liczbowo. Niestety istniej kilka wariantów stawek zerowych , rozliczanych na podstawie różnych kryteriów. Nie będziemy się wgłębiać w rachunkowość, wystarczy ,że zauważymy następujący problem - jeżeli stawka podatku jest wyrażona liczbowo, wystarczy dokonać konwersji tej liczby (pamiętajmy ,że pole St_VAT z tabeli TransSzcz jest typu tekstowego) i przeprowadzić stosowne obliczenia. Z drugiej strony, jeżeli stawki podatku nie można konwertować na liczbę, wówczas jako procent należy przyjąć 0. Zaznaczam , że problem ten dałoby się rozwiązać bez tworzenia dodatkowej tabeli, jednak z przyczyn opisanych w dalszych rozdziałach, zaproponowane tutaj rozwiązanie będzie dla nas najwygodniejsze.

1.Zdefiniuj tabelę St_vat, zgodnie z poniższym opisem:

Nazwa pola : IDSt_VAT

Typ : A

Rozmiar : 2

Klucz : *

Nazwa pola : Pozycja

Typ : S

Rozmiar :

Klucz :

Nazwa pola : Procent

Typ : N

Rozmiar :

Klucz :

Nazwa pola : Opis

Typ : A

Klucz :

2.Pozostając w programie Database Desktop, otwórz tabelę St_vat (polecenie File, Open, Table)

3.Naciśnij klawisz [F9], aby przejść do trybu edycji

4.Wypełnij tabelę :

St_vat : 1,2,3,4

IDst_VAT : 0 ,22, 7 ,zw

Pozycja : 0 ,23,7 -1

Procent : 0,00 ; 23,00 ; 7,00 ; 0,00

Opis : stawka 0% , stawka 23%, stawka 7% , zwolniona

Zwróć uwagę na kolumnę Pozycja. Określa ona kolejność ,w jakiej będą występowały poszczególne stawki na różnego rodzaju zestawieniach i wydrukach. Oczywiście można by tutaj zastosować ciągłą numeracją np. 1,2,3,4 ale przyjęte rozwiązanie ułatwia nam dodanie ewentualnych nowych stawek - gdyby zaszła potrzeba wstawienia stawki np. 12%, otrzyma ona pozycję 12 i nie trzeba będzie zmieniać kolejności w pozostałych rekordach .Przypisanie wartości -1 stawce "zw" oznacza ,że będzie ona drukowana jako pierwsza (przed pozycją 0, która jest przypisywana stawce 0%). Podsumowując, dzięki zastosowaniu dodatkowej tabeli uzyskamy następujące korzyści:

-Nie trzeba będzie tworzyć skomplikowanych zapytań SQL , konwertujących ciągi znaków na liczby

-Podczas wypełniania faktury użytkownik będzie mógł wprowadzić tylko jedną ze zdefiniowanych stawek podatku (w tym celu trzeba będzie zdefiniować relację)

-Będziemy mieli kontrolę nad kolejnością poszczególnych stawek VATY w wydrukach

Ujemnymi stronami przyjętego rozwiązania jest nieznaczne zwiększenie złożoności programu i konieczność stosowania dodatkowej operacji złączenia w zapytaniach operujących na stawkach podatku VAT.

Określanie zależności między tabelami

W tej chwili nasza baza danych składa się z pięciu tabel : Kontrah, TowUslug, TranNagl i St_vat. Trudną ją jednak nazwać relacyjną, ponieważ zdefiniowane tabele nie są ze sobą jeszcze w żaden sposób powiązane. W tym podrozdziale wyjaśnimy , na czym polega definiowanie relacji i jakie to ma praktyczne znaczenie dla funkcjonowania baz danych oraz zarządzających nimi aplikacji. Istnieją następujące typy relacji : jeden - do - jednego, jeden - do - wielu i wiele - do - wielu . Najczęściej występuje drugi typ. Relacja jeden - do -wielu oznacza ,że z jednym rekordem w tabeli nadrzędnej jest powiązanych wiele rekordów w tabeli podrzędnej .Spróbujmy to wyjaśnić na przykładzie tabel TranNagl i Kontrah. Danemu kontrahentowi może być przypisanych wiele nagłówków transakcji (faktury), natomiast każdemu nagłówkowi transakcji jest przypisany tylko jeden kontrahent. Mamy więc do czynienia z relacją typu jeden - do - wielu . Zależność taką często przedstawia się w formie graficznej. Identyfikator kontrahenta (pole IDKontrah) w tabeli TranNagl nazywany jest kluczem obcym lub kluczem zewnętrznym (foreign key), ponieważ pochodzi z innej, zewnętrznej tabeli. Jednym z najważniejszych zadań systemu zadań systemu zarządzania relacyjnymi bazy danych (SZRBD) jest kontrola poprawności powiązań między tabelami, czyli dbanie o zachowanie integralności referencyjnej .W naszym przykładzie możemy powiedzieć , że baza danych zachowuje integralność referencyjną, jeżeli dla każdego pola IDKontrah w tabeli TranNagl istnieje odpowiedni powiązany rekord w tabeli Kontrah, innymi słowy - dla każdego identyfikatora kontrahenta w tabeli nagłówków transakcji istnieje odpowiedni zapis w tabeli kontrahentów. Łatwo się domyślić,co stałoby się , gdyby ten warunek nie był spełniony - najlepszym razie zostałaby wydrukowana faktura bez odbiorcy. Ponieważ wiemy do czego zmierzamy, pozostało nam zakodować potrzebne informacje w strukturze bazy danych .Do tego celu użyjemy oczywiście programu narzędziowego Database Desktop

1.W menu Tools, Utilities wybierz polecenie Restructure

2.W oknie dialogowym SelectFile zaznacz tabelę TranNagl, a następnie kliknij przycisk Otwórz. Pojawi się okno dialogowe Restructure Table, przedstawiające strukturę tabeli TranNagl

3.Na liście Table Properties z prawej strony okna dialogowego wybierz opcję Referential Integrity. Zawartość okna pod listą zmieni się.

4.Kliknij przycisk Define. Pojawi się okno dialogowego Referential Integrity . Z lewej strony tego okna znajduje się lista pól tabeli podrzędnej (TranNagl), z prawej widoczne są wszystkie zdefiniowane nazwy tabel

5.Na liście Fields zaznacz pole IDKontrah i kliknij przycisk strzałki w prawo .Wskazane pole znajdzie się na liście Child fields

6.Na liście Tables zaznacz plik tabeli Kontrah i klkinij przyciski strzałki w lewo. Na liście Parent′s key zostanie umieszczony indeks podstawowy tabeli Kontrah. Jak pamiętamy , został on utworzony w oparciu o pole IDKontrah. W grupie opcji Update rule mamy do wyboru dwie możliwości : Cascade oraz Prohibit. Gdy zaznaczona jest pierwsza , wszystkie zmiany dokonane w indeksie głównym tabeli nadrzędnej będą kaskadowo (stąd nazwa) wprowadzane w tabelach podrzędnych. W naszym przykładzie oznacza to ,że jeżeli użytkownik zmieni identyfikator kontrahenta, aparat bazy danych spróbuje zmienić wszystkie identyfikatory tego kontrahenta występujące w rekordach tabeli TranNagl. Aby przeprowadzić ten proces, tabela podrzędna (tutaj TranNagl) jest blokowana na wyłączność - w trakcie modyfikacji kaskadowej ,żaden inny użytkownik nie będzie mógł wprowadzić zmian w tabeli. Z drugiej strony, jeżeli jakiś użytkownik spróbuje zmienić identyfikator klienta w tabeli Kontrah, a równocześnie zawartość tabeli TranNagl jest edytowana

przez kogoś innego, próba utworzenia blokady wyłączonej tabeli nie powiedzie się, co z kolie uniemożliwi dokonanie zmiany identyfikatora. Druga opcja, Prohibit, po prostu uniemożliwia zmianę klucza podstawowego (indeksu głównego w tabeli nadrzędnej) ,jeżeli w tabeli podrzędnej istnieją jakieś rekordy powiązane z tym kluczem. Wracając do naszego przykładu - gdy ta opcja będzie zaznaczona, nie będzie można usunąć ani zmienić identyfikatora kontrahenta w tabeli Kontrah, jeżeli istnieją już dla niego jakieś transakcje. Jak widać , pierwsza opcja zapewnia większą elastyczność i należy ją stosować jeżeli to tylko możliwe (ograniczenia stosowania tej opcji występuje w przypadku niektórych formatów przechowywania danych, innych niż Paradox) .Zaznaczona domyślna opcja Strict referential integrity powoduje ,że tabela będzie chroniona przed uszkodzeniem przez wcześniejsze wersje programu Database Desktop. Niestety sprawia to , że tabela ta stanie się niedostępna z poziomu bazy Paradox dla systemu DOS. Oczywiście ma to znaczenie tylko wtedy jeżeli przewidujesz wykorzystanie tabel w aplikacjach starszego typu

7.Kliknij przycisk OK. Pojawi się okno dialogowe Save Referential Integrity As

8.Wpisz KontrahRI i kliknij OK. Ponownie aktywne stanie się okno dialogowe Restructure Table. Na liście z prawej strony okna będzie widoczny zdefiniowany przed chwilą warunek integralności. W ten sposób zdefiniowaliśmy reguły integralności referencyjnej między tabelami TrangNagl i Kontrah. Kluczem obcym w warunku integralności w tabelach typu Paradox może być tylko indeks podstawowy

W analogiczny sposób zdefiniujemy teraz warunki integralności między tabelami TranNagl i TranSzcz

1.W menu Tools, Utilities wybierz polecenie Restructure

2.W oknie dialogowym SelectFile zaznacz plik TranSzcz i kliknij przycisk Otwórz. Pojawi się okno dialogowe Restructure Table przedstawiające strukturę tabeli TranSzcz

3.Na liście rozwijanej Table properties zaznacz opcję Referential Integrity

4.Kliknij przycisk Define .Pojawi się okno Referential Integrity

5.Na liście z lewej strony dwukrotnie kliknij pole IDTranNagl. Na liście z prawej strony dwukrotnie kliknij tabelę TranNagl

6.Upewnij się ,że zaznaczone są opcje Cascade i Strict referential integrity, a następnie kliknij przycisk OK

7.W oknie dialogowym Save Referentia;l Integrity As wpisz TranNaglRI i kliknij przycisk OK

Kolejny warunek integralności zdefiniujemy dla pola IDTowUslug, będącego identyfikatorem towaru lub usługi w tabeli szczegółów transakcji

1.Ponownie kliknij Define

2.W oknie dialogowym Referential Integrity z listy po lewej stronie wybierz pole IDTowUSlug, natomiast z listy po prawej wybierz TowUslug. Dzięki tej relacji , użytkownik nie będzie mógł wprowadzić w pozycji transakcji identyfikator towaru, który nie został zdefiniowany w tabeli TowUslug

3.Upewnij się ,że zaznaczone są opcje Cascade i Strict referential integrity, a następnie kliknij przycisk OK

4.W oknie dialogowym Save Referential Integrity As wpisz TowUSlugRI i kliknij przycisk OK

Ostatni warunek integralności w tej tabeli będzie dotyczył pola ze stawką podatku VAT. Będziesz postępować jak w poprzednim ćwiczeniu.

1.Ponownie kliknij przycisk Define

2.W oknie dialogowym Referential Integrity z listy po lewej stronie wybierz pols St-VAT, natomiast z listy po prawej wybierz tabelę St_vat. Dzięki tej relacji, użytkownik nie będzie mógł wprowadzić w pozycji transakcji stawki podatku VAT, która nie została zdefiniowana w tabeli St_vat

3.Upewnij się ,że zaznaczone są opcje Cascade i Strict referential integrity , a następnie kliknij przyciks OK

4.W oknie dialogowym Save Referential Integrity As wpisz St_VATRI i kliknij przycisk OK.

5.W oknie dialogowym Restructure Table kliknij przycisk Save

W tej chwili wszystkie niezbędne relacje między tabelami naszej bazy danych są już zdefiniowane. Warto tutaj nadmienić ,że wspomniane zależność można definiować również w samym programie klienckim, który w tym przypadku będzie aplikacja do wystawiania faktur. Oznacza to ,że w praktyce etap definiowana relacji na poziomie aparatu bazy danych można by w ogóle pominąć. Takie postępowanie jest jednak niewskazane. Projektując bazy danych, wszystkie zależności i ograniczenia należy definiować na możliwie jak najniższym poziomie .Sprzyja to zwiększeniu niezawodności systemu i odciąża aplikacje użytkowe - relacje między tabelami są definiowane w aparacie bazy danych tylko raz

Tabela pomocnicza Liczniki

W tabeli Liczniki będą zapisywane informacje o ostatnich wykorzystanych numerach faktur. Ponieważ dla każdego typu transakcji, serii i dopisku mogą być prowadzone oddzielne numeracje, oprócz ostatniego numeru faktury w tej tabeli musimy również przechowywać informacje o wartościach z pól Typ, Seria i Dopisek z tabeli TranNagl

Zdefiniuj tabelę Liczniki zgodnie z poniższym opisem:

Nazwa pola : Licznik

Typ : A

Rozmiar : 40

Klucz : *

Nazwa pola : OstatniNumer

Typ : I

Rozmiar :

Klucz :

Tabela ta ma tylko dwa pola. W pierwszym polu będzie przechowywana nazwa licznika (np. nazwa serii faktury),natomiast w drugim jego wartość (np. ostatni numer faktury dla określonej serii).

Podsumowanie

W tej części zaprojektowałeś kompletną strukturę bazy danych. W ramach tego procesu utworzyłeś sześć tabel : Kontrah, Liczniki, St_vat, TowUslug, TranNagl i TranSzcz. Ponadto zdefiniowałeś warunki

integralności, zapewniając zachowanie prawidłowych powiązań między tabelami. Gdy baza danych jest już zdefiniowana, możemy przystąpić do tworzenia szkieletu aplikacji użytkowej, za pomocą której będziemy wprowadzać i edytować dane oraz sporządzać wydruki (…)

Praktyczna aplikacja bazodanowa : Program do fakturowania cz. II

SZKIELET APLIKACJI

Interfejs użytkownika

Nadszedł czas byśmy wreszcie uruchomili zintegrowane środowisko programowania Delphi. Za jego pomocą przygotujemy wszystkie niezbędne formularz i raporty oraz wprowadzimy odpowiedni kod sterujący wykonywanymi operacjami. Rozpoczniemy od utworzenia formularza głównego, uruchamianego zaraz po starcie aplikacji. Będzie on pełnił rolę "centrum dowodzenia", które udostępnia różne funkcje programu. Jednak zanim przystąpimy do kodowania, musimy na chwilę cofnąć się do etapu projektu i przyjrzeć się ogólnej koncepcji naszej aplikacji pod kątem jej wewnętrznej struktury i właściwości interfejsu użytkownika

Konieczne jest rozważenie takich właściwości interfejsu jak :

- egronomiczność
- intuicyjność
- atrakcyjność
- ajednorodność

Co rozumiemy przez ergonomiczność interfejsu? Oczywiście chodzi tu o łatwy dostęp do najważniejszych funkcji. W każdej aplikacji są opcje bardzo często wykorzystywane (u nas będzie to moduł odpowiedzialny za wystawianie faktur) i na nich powinniśmy skoncentrować uwagę. Należy je tak zaprojektować, aby korzystanie z nich było jak najwygodniejsze. Musimy pamiętać na przykład, że użytkownik będzie wystawiał ponad sto faktur dziennie, co oznacza ,że każde niepotrzebne kliknięcie przycisku lub zbędny ruch myszą będą przez niego traktowane jako uciążliwość. Warto również zadbać o to, by dla każdej operacji dostępnie za pomocą myszy istniał również odpowiednik klawiaturowy. Praca w aplikacjach bazodanowych z reguły polega na wpisywaniu tekstu, co oznacza ,że każde oderwanie rąk od klawiatury wymaga dodatkowego czasu i przeniesienie uwagi na mysz czy inne urządzenie. Intuicyjność interfejsu w pewnym sensie decyduje o ergonomiczności. Wyróżniamy ją jednak tutaj , aby podkreślić szczególne znaczenie tego czynnika dla użytkownika dopiero poznającego aplikację. Ergonomiczność , tak jak ją opisaliśmy wcześniej, jest istotna szczególnie z punktu widzenia doświadczonego użytkownika aplikacji, który dąży do maksymalnej optymalizacji wykonywanych przez siebie czynności. Z kolei mówimy ,że aplikacja ma intuicyjny interfejs, jeśli "dana funkcja programu jest tam, gdzie się jej można najbardziej spodziewać". Rozważmy ten problem na przykładzie. Gdy wyświetlana jest lista kontrahentów, warto umieścić obok niej przycisk Drukuj, który umożliwi wydruk list ogólnych, list zobowiązań, obrotów itp. Natomiast analogiczny przycisk obok listy towarów powinien uruchamiać wydruk stanów magazynowych, wysokości sprzedaży itp. "Nieintuicyjny" program mógłby wyglądać następująco - lista kontrahentów umożliwiałaby co prawda przejście do kartoteki określonego klienta, ale wydruk dostawców wymagałby powrotu do menu głównego, wejścia do menu Wydruki i wybranie opcji Dostawcy. Atrakcyjność interfejsu nie wymaga chyba specjalnego definiowania. Zależy nam na tym, aby formularze były czytelne, przejrzyste i "przyjemne dla oka", a nie odstręczające. W tym miejscu warto jednak ostrzec przed "wodotryskami" - nadmiar kolorów, animacji i efektów specjalnych może utrudnić korzystanie z programu. Przerost formy nad treścią jest równie niewskazany , jak zupełny brak formy. Ostatni element związany z interfejsem to jego jednorodność. Jeżeli na przykład na jednym formularzu znajduje się przycisk Zapisz, to na innym formularzu powinien on mieć takie same rozmiary, taką samą ikonę i znajdować się w takim samym miejscu. Ułatwi to użytkownikowi naukę korzystania z programu i jego dalsze stosowanie

Wewnętrzna struktura aplikacji

Wiele elementów aplikacji , choć ukrytych przed użytkownikiem, ma kluczowe znaczenie dla jej poprawnego działania i dalszej rozbudowy. Aplikacja kliencka powinna spełniać dwa podstawowe kryteria:

- działać poprawnie
- cechować się skalowalnością

Poprawność działania aplikacji to warunek minimum jej użyteczności. Jeśli program bazodanowy nieprawidłowo analizuje zgromadzone dane lub ,co gorsza, powoduje zapis błędny bądź niespójnych informacji w tabelach , dyskwalifikuje go to całkowicie. Warto tutaj przypomnieć ,że choć aplikacja kliencka może (a wręcz powinna) weryfikować pewne aspekty poprawności danych, to generalnie mechanizmy te należy definiować na możliwe najniższym poziomie. Jeżeli na przykład między tabelami Kontrah i TranNagl ma zachodzić relacja typu jeden - do - wielu, warto ją zdefiniować na poziomie struktury bazy danych, a nie dopiero w aplikacji klienckiej. Takie podejście ma same zalety:

- podnosi bezpieczeństwo danych
- zmniejsza ryzyko utraty integralności referencyjnej
- upraszcza kod aplikacji, co jednocześnie poprawi jej niezawodność

Jednak pewne elementy można sprawdzić tylko na poziomie aplikacji klienckiej, co oznacza ,że w kodzie opracowywanych modułów formularzy muszą się znaleźć odpowiednie mechanizmy weryfikujące. Skalowalność aplikacji ma dwa aspekty : ilościowy i jakościowy .W pierwszym przypadku - skalowalności ilościowej chodzi przede wszystkim o to, aby zwiększająca się objętość danych nie powodowała drastycznego spowolnienia działania aplikacji użytkowych. Odpowiednie decyzje związane ze skalowalnością ilościową powinny być podejmowane już w fazie projektowanie. Jeżeli przewidujemy, że nasza aplikacja będzie wykorzystywana tylko przez kilku użytkowników i znajdzie się w niej maksymalnie kilkaset tysięcy rekordów, wystarczy utworzenie systemu jednowarstwowego, w którym większość zadań związanych z przetwarzaniem danych wykonuje aplikacja kliencka. Gdy aplikacja jest przeznaczona dla wielu użytkowników i chcemy w niej gromadzić miliony rekordów, konieczne jest zastosowanie rozwiązania dwuwarstwowego (aplikacja klient - serwer) lub nawet trójwarstwowego (aplikacja kliencka - serwer aplikacji - serwer baz danych). Skalowalność jakościowa polega na takim zaprojektowaniu struktury bazy danych i aplikacji, aby łatwo je było rozbudowywać. O skalowalności bazy danych zadbaliśmy , na przykład definiując tabelę TranNagl. Wstawiliśmy do niej dodatkowe pole, które umożliwi podział transakcji na sprzedaż , zakup i inne dokumenty. Skalowalność aplikacji wiąże się również z tym by dodawanie kolejnych elementów nie powodowało zaburzenia działania dotychczasowych funkcji i nie wymagało przebudowy całego programu. Podczas projektowania kolejnych elementów programu przez cały czas musimy mieć na uwadze powyższe zalecenia

Aplikacja Faktury - schemat działania

Podczas projektowania interfejsu użytkownika warto zaplanować sobie wcześniej, w jaki sposób będą na siebie oddziaływać poszczególne moduły programu. Taki schemat ułatwi w przyszłości tworzenie kodu. Aplikacja będzie rozpoczynała działanie od wyświetlenia formularza z głównym menu. Menu to pozwoli na bezpośrednie wyświetlenie formularzy z listami kontrahentów, towarów i faktur. Dopiero po wyświetleniu jednej z tych list użytkownik będzie mógł wybrać rekord konkretnego kontrahenta (towaru lub faktury) i przejść do formularza, w którym będą dostępne informacje szczegółowe. Takie rozwiązanie daje istotne korzyści: wyświetlanie rekordów w postaci tabeli ułatwia ich przeglądanie i wyszukiwanie; z drugiej strony , zastosowanie formularza prezentującego pojedynczy rekord pozwala na jednoczesne uzyskanie wszystkich zapamiętanych w nim informacji. Formularz główny będzie

również pozwalał na wyświetlanie formularzy z opcjami konfiguracji. Wydruki zbiorcze związane z fakturami (np. sprzedaż za dany okres, obroty z kontrahentem) są dostępne z formularza, w którym widzimy listę faktur. Z kolei w formularza, w którym edytujemy pojedyncze faktury, możemy uruchomić ich wydruk. Dzięki takiemu rozwiązaniu realizujemy postulat intuicyjności interfejsu. Ponieważ formularz główny będzie punktem wyjścia do wszystkich funkcji programu, tworzenie przykładowej aplikacji rozpoczniemy właśnie od niego

Formularz główny

Jeżeli myślisz o komercyjnym potraktowaniu tworzonej aplikacji, właściwie zaprojektowanie formularza głównego jest bardzo istotna. Nie na próżno mówi się ,że najważniejsze jest pierwsze wrażenie. Musisz zadbać przede wszystkim o wygodny dostęp do najczęściej wykorzystywanych opcji programu, ale nie zapomnieć o estetyce. Pierwszy ekran , który pojawi się przed oczami użytkownika musi go zachęcać do dalszej pracy z programem. Często aplikację rozpoczyna ekran powitalny (splash screen). Ma on dwojakie zastosowanie - pozwala zareklamować się twórcom programu i jednocześnie wypełnić jakoś czas potrzebny na wczytanie modułów startowych aplikacji. Tworzenie takiego ekranu jest bardzo proste (najczęściej jest to formularz bez żadnych aktywnych kontrolek), pominiemy więc ten etap i od razu przystąpimy do tworzenia formularza z głównym menu programu

Nowa aplikacja

W tej części rozpoczniemy tworzenie nowej aplikacji. Zadanie to jest bardzo proste - wystarczy uruchomić środowisko Delphi

1.Uruchom program Delphi. Na ekranie pojawi się pusty formularz główny nowego projektu .Z lewej strony ekranu powinny być widoczne okna Object TreeView oraz Object Inspector. Ponad pustym formularzem aplikacji znajduje się główne menu programu Delphi oraz paski narzędzi i przybornik z komponentami

2.W okienku Object Inspector przejdź do zakładki Properties, zaznacza właściwość Caption i w miejsce dotychczasowego tytułu formularza wpisz Moje faktury

Menu główne

Na przygotowanym formularzu możemy wprowadzić opcje menu i podmenu, które posłużą do uaktywniania różnych funkcji programu. Utworzymy trzy menu z menu z poleceniami : menu Transakcje z podmenu Sprzedaż, menu Słowniki z dwoma podmenu Kontrahenci i Słownik, wreszcie - menu Narzędzia zawierające polecenia Konfiguracja. Wykonaj poniższe kroki:

1.Upewnij się ,że na palecie komponentów aktywna jest zakładka Standard, kliknij ikonę komponentu MainMenu, a następnie kliknij w lewym górnym rogu formularza. Na formularzu pojawi się komponent MainMenu1

2.Dwukrotnie kliknij komponent MainMenu1. Pojawi się okno dialogowe edycji menu głównego

3.Gdy aktywne jest okno dialogowe Form1.MainMenu1, na karcie Properties okna dialogowego Object Inspector przejdź do właściwości Caption, wpisz Transakcje i naciśnij klawisz Enter. Do menu zostanie dodana nowa pozycja Transakcje, poniżej widoczne będzie zaznaczenie umożliwiające wprowadzenia polecenia menu

Po wstawieniu nowego menu jego właściwość AutoHotKeys ma wartość maAutomatic. Oznacza to ,że kompilator automatycznie dobierze klawisz skrótu dla danej opcji menu. Jako litera skrótu dobierana jest zawsze pierwsza wolna litera z nazwy opcji - w przypadku pozycji Transakcje będzie to litera T (po

uruchomieniu programu zostanie ona wyróżniona podkreśleniem; aby uaktywnić ją za pomocą klawiatury należy nacisnąć klawisze [lewy Alt + T]. Jeżeli na tym samym poziomie menu pojawi się kolejna opcja na literę T, na przykład Towary, kompilator przypisze klawisz skrótu do litery) itd. Jeżeli sami chcemy wybierać klawisze skrótu lub z jakichś względów chcemy z nich całkowicie zrezygnować, musimy zmienić właściwość AutoHotKeys na maManual. Aby zdefiniować literę skrótu, podczas wpisywania nazwy opcji menu we właściwości Caption należy poprzedzić odpowiednią literą znakiem &

4.Kliknij pozycję menu Transakcje , a następnie zaznacza wolną pozycję podmenu. Naciśnij Enter, aby przejść do właściwości Caption kolejnego polecenia menu, wpisz Sprzedaż i naciśnij Enter.

5.Naciśnij strzałkę w prawo. Menu Transakcje zostanie zwinięte, a na prawo od niego pojawi się niebieskie zaznaczenie i ramka

6.Naciśnij klawisz Enter, aby przejść do właściwości Caption polecenia menu i wpisz Słowniki

Podczas tworzenia menu, Delphi automatycznie generuje definicje pozycji menu typu TMenuItem. Każda z tych pozycji uzyskuje nazwę zgodną z tekstem wpisanym w jej właściwości Caption, zakończoną kolejną liczbą. Z uwagi na wymogi kompilatora z nazwy tej są usuwane niektóre niedozwolone znaki; dotyczy to między innymi spacji oraz polskich znaków diakrytycznych. I tak np. opcja menu o nazwie Środki trwałe będzie reprezentowana przez zmienną rodkitrwae1. Dla zachowania czytelności kodu w takich przypadkach warto skorygować właściwość Name tworzonej pozycji menu, wpisując na przykład SrodkiTrwale1

Jeśli zauważysz ,że w czasie wpisywania niektórych polskich znaków w polu Caption zamiast nich uaktywniają się różne opcje Delphi. Aby to usunąć, należy wprowadzić odpowiedni klucz w rejestrze ([HKEY_CURRENT_USER\Software\Delphi\x.0\Editor\Options] "NoCtrlAltKeys" = "1") .

7.Kliknij pozycję menu Słowniki a następnie kliknij wolną pozycję menu, która pojawia się poniżej

8.Przejdź do właściwości Caption zaznaczonego elementu i wpisz w niej Kontrahenci

9.Kliknij następną wolną pozycję menu i w jej właściwości Caption wpisz Towary i usługi

10.W analogiczny sposób utwórz menu Narzędzia ,a w nim polecenia Konfiguracja

11.Zamknij okno Form1.MainMenu1

Można powiedzieć ,że wstępna wersja formularza głównego jest już gotowa. Każda z utworzonych pozycji menu będzie powodowała uruchomienie jakiejś opcji programu ,na przykład wyświetlanie listy towarów lub kontrahentów. Zapiszmy naszą pracę na dysk

1.Zaznacz formularz i w Object Inspector zmień jego właściwość Name na MenuG1Form

2.Kliknij przyciks Save All na standardowym pasku narzędzi. Pojawi się okno dialogowe Save As. Należy na nim określić nazwę modułu w którym będzie przechowywany kod związany z formularzem głównym

3.W polu Nazwa pliku wpisz MenuG1

4.Odszukaj folder w którym chcesz przechowywać projekt aplikacji i kliknij Zapisz. Pojawi się kolejne okno dialogowe Save As, w którym należy podać nazwę całego projektu. Wprowadzoną tutaj nazwę uzyska również plik wykonywalny, który powstanie po skompilowanym kodu źródłowego

5.W polu Nazwa pliku wpisz Faktury i kliknij przycisk Zapisz

Informacje związane z formularzem są zapisywane w dwóch plikach. Kod języka ObjectPascal, obejmujący definicję klasę formularza, procedury obsługi, zmienne itp., jest zapisywany w pliku o rozszerzeniu .PAS, natomiast właściwości formularza i umieszczonych na nim komponentów są zapisywane w pliku o rozszerzeniu .DFM

Paski narzędzi

Pasek narzędzi nie jest niezbędnym elementem aplikacji, jednak jego zastosowanie ułatwia realizację postulatu ergonomiczności. Np. jeśli na pasku narzędzi umieścimy ikonę wywołującą formularz do wyświetlania faktur, użytkownik uzyska go za pomocą jednego kliknięcia - korzystając z menu musiałoby wykonać dwie operacje. Spróbujemy teraz utworzyć pasek narzędzi, którego przyciski będą wyświetlały następujące formularze:

- wystawianie faktury
- lista kontrahentów
- lista towarów i usług
- konfiguracja

Wykonaj następujące czynności:

1.Na palecie Win32 wybierz komponent ToolBar i umieść go na formularzu .W górnej części formularza pojawi się pusty pasek narzędzi o nazwie ToolBar1

2.Prawym przyciskiem myszy kliknij pasek narzędzi i wybierz polecenie New Button. Na pasku narzędzi pojawi się pusty przycisk.

3.Prawym przyciskiem myszy klikaj pasek narzędzi i wybieraj kolejno polecenia : New Separator, New Button, New Button, New Separator i New Button. Aby na utworzonym pasku narzędzi pojawiły się ikony przedstawiające poszczególne opcje, na formularzu musi się znaleźć komponent typu TImageList

4.Na palecie Win32 wybierz komponent ImageList i umieść go na formularzu (najlepiej w pobliżu komponentu MainMenu1)

5.Dwukrotnie kliknij komponent ImageList1. Na ekranie pojawi się okno dialogowe

6.Kliknij przycisk Add. Aby wypełnić pola przycisków, skorzystamy z przygotowanej uprzednio listy obrazów

7.W oknie dialogowym Add Images przejdź do folderu gdzie trzymasz ikony

8.Kliknij OK aby wczytać je do obiektu ImageList1. Do tworzenia ikon możesz użyć programu Image Editor dostarczanego wraz z Delphi. Możesz od razu utworzyć listę wszystkich ikon lub zdefiniować ikony pojedynczo, a następnie za pomocą przycisku Add dodawać je kolejno do obiektu ImageList

9.Zaznacz komponent ToolBar1

10.Przejdź do Object Inspectora i we właściwości Images ustaw wartość ImageList1

11.Kliknij Save As aby zapisać wprowadzone modyfikacje

Alias bazy danych

Aliasa bazy danych to umowan nazwa ,zastępująca pełną ścieżkę dostępu do plików. Jeśli chcemy zapewnić sobie swobodę w lokalizowaniu naszej aplikacji nie możemy na sztywno zakodować ścieżek dostępu, ponieważ w przypadku zainstalowania programu w innym katalogu nie można będzie

odnaleźć niezbędnych plików. Z drugiej strony zmuszanie użytkownika do zainstalowania programu w ściśle określonej lokalizacji jest rozwiązaniem co najmniej nieeleganckim. Aby uniknąć takich problemów stosuje się aliasy. Alias definiuje się tylko raz - w programie Database Desktop lub bezpośrednio w aplikacji użytkowej. Wszystkie następne odwołania do bazy danych nie podają jej lokalizacji, lecz posługują się aliasem. Jak można się domyślić, jeżeli z jakichś powodów lokalizacja bazy danych musi być zmieniona, wystarczy zmodyfikować alias, a wszystkie odwołania do bazy w dalszym ciągu będą działały prawidłowo. Aliasy można definiować bezpośrednio w programie Database Desktop lub w samej aplikacji użytkowej. Każde z tych rozwiązań ma swoje zalety i wady .Alias utworzony w programie Database Desktop jest przechowywany w pliku konfiguracyjnym IDAPI.CFG i mają do niego dostęp wszystkie aplikacje korzystające z aparatu BDE (Borland Database Enviroment), co jest niewątpliwie jego zaletą. Z drugiej strony aliasy BDE są dosyć kłopotliwe podczas przygotowania wersji instalacyjnej programu. Alias generowany w aplikacji użytkowej ma charakter doraźny - jest tworzony przez aplikację i przestaje istnieć natychmiast po jej zakończeniu (lub wcześniej, na skutek pewnych operacji w programie). Ma do niego dostęp tylko aplikacja, w której zostanie utworzony. Funkcjonalnie obydwa typy aliasów daję ten sam efekt - w komponentach operujących na tabelach nie musimy odwoływać się bezpośrednio do ścieżek dostępu, lecz stosujemy "wyrażenie zastępcze" - słowem ,alias. Aby utworzyć alias:

1.Przejdź do palety komponentów BDE

2.Przenieś na formularzu główny aplikacji komponent TDatabase .W tym celu kliknij go na palecie komponentów, a następnie kliknij formularz

3.Prawym przyciskiem myszy kliknij umieszczony na formularzu komponent Database1 i z menu podręcznego wybierz polecenie Database Editor. Pojawi się okno dialogowe MenuGlForm.Database1Database

4.W polu Name wpisz BAZAFAKT. Wpisana wartość będzie aliasem bazy danych używanym w tej aplikacji

5.Na liście DriveName odszukaj pozycję STANDARD. Sterowników standardowych należy używać , jeżeli baza danych korzysta z formatu Paradox lub dBaseIV

6. W polu Parameter overrides wpisz PATH = .\Dane. Wspomniane pole służy do wpisania wartości przesłaniających ustawienia domyślne. W tym przypadku, chcemy przesłonić domyślną wartość parametru PATH (pusty ciąg znaków), przypisując mu ścieżkę dostępu do plików naszej bazy danych. Kropka oznacza bieżący katalog. Wyrażenie ".\Dane" oznacza podkatalog Dane katalogu bieżącego. Wyjaśnijmy znaczenie pozostałych opcji widocznych w oknie MenuGLForm.Database1 Database .W polu Alias name można wyświetlić nazwę aliasu zdefiniowanego w aparacie BDE .W ten sposób nowo tworzony alias powstanie w oparciu o alias istniejący już w bazie danych. Jeżeli zaznaczone jest pole wyboru Login prompt w momencie łączenia się z bazą danych wymagającą zalogowania się, pojawia się okno dialogowe umożliwiające podanie nazwy użytkownika i hasła. Gdy ta opcja zostanie wyłączona , programista sam musi zapewnić przekazanie tych wartości (na przykład kodując je w programie) .Zaznaczone pole wyboru Keep inactive connection oznacza ,że połączenie bazą danych będzie utrzymywane, nawet jeżeli nie będzie ono korzystało z żadnego zestawu danych w bazie. Zapis PATH = .\Dane może stosować ,gdy w fazie projektowej chcemy mieć dostęp do bazy danych. Jednak w kodzie programu warto umieścić jawne przypisanie , deklarujące ,że katalog danymi jest podkatalogiem katalogu z plikiem wykonywalnym aplikacji. Nazwę katalogu , w której znajduje się plik wykonywalny, można uzyskać za pomocą wywołania ExtractFilePath(Application.ExeName)

7.Kliknij przycisk OK

8.Zaznacza formularza, przejdź do karty Event Object Inspector i dwukrotnie kliknij w polu obok zdarzenia OnCreate. Ścieżkę dostępu do danych należy ustalić zaraz po rozpoczęciu aplikacji

9.Uzupełnij procedurę zdarzenia , aby uzyskała postać :

```
procedure TMenuGlForm.FormCreate(Sender: TObject);

begin

with DataBase1 do begin

Connected := False;

Params.Clear;

Params.Add('Path=' + ExtractFilePath(Application.ExeName) + 'dane');

Connected := True;

end; end;
```

Powyższa procedura kolejno : rozłącza component DataBase1 ze źródłowoą bazą danych, czyści listę parametrów, wstawia nowy parametr PATH i ponownie łączy z bazą danych

10.Kliknij przycisk Save All aby zapisać wprowadzone modyfikacje

Formularz do edycji konfiguracji programu

Jak na razie nasze programowanie, rozumiane w tradycyjnym znaczeniu tego słowa jako pisanie kodu, ograniczyło się do wstawienia kilku wierszy. Jedną z zalet środowiska Delphi jest automatyzacja wielu procesów. Między innymi w trakcie tworzenia formularza głównego powstał obiekt dziedziczący po klasie TForm. Następnie w trakcie definiowania elementów menu, zostały do niej dołączone definicje komponentów typu TMainMenu i TMenuItem. Jeżeli chcesz zobaczyć definicję modułu MainGl, wciśnij klawisz F12 lub kliknij przycisk Toggle Form/Unit. Niestety (a może na szczęście) kodowania nie da się uniknąć. Kolejne linie kodu wpiszemy, tworzy moduł do edycji ustawień konfiguracyjnych naszego programu. Rozpoczniemy od wygenerowania nowego formularza, przeznaczonego do przechowywania informacji konfiguracyjnych. Każda większa aplikacja ma pewien zbiór ustawień. Z reguły są one dostępne dla użytkownika, który może je dowolnie modyfikować. Jednocześnie większość tych ustawień jest zapamiętywana, co pozwala na ponowne uruchamiania programu z taką samą konfiguracją .Wspomniane ustawienia mogą być zapisane w wielu miejscach. W tym celu wykorzystuje się najczęściej : rejestr systemu operacyjnego , plik bazy danych lub zewnętrzny plik tekstowy .W naszym programie zastosujemy ostatnie z wymienionych rozwiązań. Informacje konfiguracyjne będą zapisywane w pliku tekstowym o rozszerzeniu .INI .Delphi posiada wbudowane mechanizmy do pracy tego typu plikami, dzięki czemu pobieranie z nich wartości oraz modyfikowanie ich jest bardzo proste. Co powinno się znaleźć w pliku konfiguracyjnym? Ogólnie rzecz biorąc można stwierdzić ,że im więcej ustawień jest dostępnych dla użytkownika, tym lepiej. Jednak w naszej aplikacji ograniczymy się tylko do kilku niezbędnych elementów. W ustawieniach konfiguracyjnych będziemy przechowywali informacje umieszczane w nagłówku faktury, takie jak : domyślna seria i dopisek do numeru faktury ,dane adresowe sprzedawcy oraz miejsce wystawienia faktur. Pierwsze dwie wartości ułatwią użytkownikowi wystawianie faktur - jeżeli stosowana jest na przykład numeracja FA/1/2012, FA/2/2012 itd. warto zautomatyzować wpisywanie skrajnych wartości. Podobne znaczenie mają dane firmy - będą one automatycznie wstawianie do każdej drukowanej faktury

1.W menu File -> New, wybierz polecenie Others. Pojawi się okno dialogowe New Items

2.Przejdź do zakładki Dialogs, upewnij się , czy zaznaczona jest opcja Copy i dwukrotnie kliknij ikonę Standard Dialog (Horizontal). Na ekranie pojawi się nowy formularz. Będie na nim umieszczona ramka typu TBevel oraz dwa przyciski TButton. Właściwość BorderStyle tego okna ma wartość bsDialog, co oznacza ,że będzie ono wyświetlane w formie okna dialogowego (bez możliwości minimalizacji oraz maksymalizacji)

3.We właściwości Caption przycisku CancelButton wpisz Anuluj

4. We właściwości Caption nowego formularza wpisz Konfiguracja programu, a jego właściwości Name wpisz KonfigForm

5.Zapisz utworzony moduł formularza pod nazwą Konfig

6.Na formularzu wstaw cztery komponenty TLabel oraz siedem komponentów TEdit (z panelu Standard)

7.Zmień właściwości Caption kolejnych komponentów TLabel na:

Domyślna seria faktur: ; Domyślny dopisek dop numeru: ; Dane firmy do faktury : ; Miejsce wystawienia : ;

8.Rozmieść etykiety i pola tekstowe. W razie potrzeby zwiększ rozmiary formularza i ramki oraz przesuń przyciski OK i Anuluj

Tekstowy plik konfiguracyjny

Spróbujemy teraz utworzyć plik typu INI, w którym umieścimy ustawienia konfiguracyjne. Budowa takiego pliku jest bardzo prosta. Aby przypisać wartości poszczególnym ustawieniom, stosuje się znak równości, np.

dane1 = Serwis Komputerowy "KOMKOM"

W celu uproszczenia organizacji danych dzieli się je na sekcje; poszczególne sekcje są ciągami znaków umieszczonymi w nawiasach kwadratowych np.

[Faktury]

Jak wynika z utworzonego przed chwilą formularza, w projektowanym pliku powinno się znaleźć siedem pozycji. Do utworzenia pliku tekstowego użyjemy edytora Delphi, choć równie dobrze możemy posłużyć się np. Notatnikiem. Pliku INI nie musisz tworzyć w opisany tutaj sposób. Możesz pod razu wprowadzić kod procedur opisanych w dalszych częściach, skompilować i uruchomić program, przejść do formularza konfiguracyjnego, wypełnić odpowiednie pola i kliknąć OK. Jeżeli plik INI nie istnieje zostanie utworzony przez metodę Create. Z kolei metoda WriteString wstawi do nowego pliku odpowiednie klucze

1.W menu File -> New, wybierz polecenie Other. Pojawi się okno dialogowe New Items

2. Na karcie New odszukaj ikonę Text File i kliknij ja dwukrotnie .W oknie edytora Delphi pojawi się nowa zakładka File1.txt

3.W nowym pliku tekstowym wpisz poniższy tekst. Zwróć uwagę ,że przed i po znaku równości nie powinno być spacji:

[Faktury]

seria = FA

dopisek = 2012

[Nazwa]

dane1 = Serwis Komputerowy "KOMKOM"

dane2 = 01-124 Brutów, ul. Błotna 21a

dane3 = NIP : 666-555-44-33

dane4 = konto : BGG o/Brutów 1010999-123456789

[Inne]

miejsce-wystaw = Brutów

Jak widać, plik zawiera dwie sekcje, w pierwszej znajdują się dane firmy umieszczane na fakturze, w drugiej - informacje dotyczące numeru faktury. Oczywiście przedstawione tutaj opcje stanowią jedynie niezbędne minimum dla omawianej aplikacji, jednak przyjęcie tego rozwiązania nie ogranicza dalszej rozbudowy pliku INI i obsługującego go formularza (Jeżeli liczba opcji obsługiwanych przez ten formularza ma być duża , warto zastanowić się nad zastosowaniem komponentu typu TTabControl i rozmieszczenie opcji na różnych zakładkach)

4.Kliknij przycisk Sabe i zapisz plik w katalogu systemowym (np. C:\Windows) pod nazwą Faktury.ini.

Pliki INI domyślnie są przechowywane w katalogu systemowym. Jeżeli chcesz zmienić lokalizację pliku INI, musisz podać pełną ścieżkę dostępu

Odczyt danych z pliku INI

Kolejnym zadaniem jest odczyt z pliku INI i wyświetlenie ich w formularzu. Pierwsze pytanie, które musimy sobie postawić, brzmi : w jakim momencie odczytywać dane w pliku: odpowiednie działanie można wykonać w momencie tworzenia formularza (zdarzenie ONCreate), w momencie jego wyświetlenia (OnShow) lub w momencie uaktywnienia (OnActivate). Wybierzmy tą ostatnią opcję, gdyż daje nam ona gwarancję ,że w chwili przejścia do formularza konfiguracyjnego zawsze zobaczymy aktualną zawartość pliku INI. Oznacza to ,że musimy przygotować odpowiednią procedurę dla zdarzenia OnActivate formularza KonfigForm

1.Uaktywnij formularz KonfigForm

2.W Object Inspectorze przejdź do karty Events i odszukaj właściwość OnActivate, a następnie dwukrotnie kliknij pole wolne obok tej właściwości

3.Delphi wygeneruje powłokę procedury obsługi zdarzenia :

Procedurę TKonfigForm.FormActivate(Sender : TObject);

begin

end;

4.Uzupełnij treść procedury, by miała ona postać taka jak na poniższym wydruku:

procedure TKonfigForm.FormActivate(Sender ; TObject);

begin

KonfigINI := TINIFileCreate('Faktury.ini'); {Otwarcie pliku}

{Wczytywanie serii i dopisku do numeru faktury}

Edit1.Text := KonfigINI.ReadString('Faktury','seria','');

Edit2.Text := KonfigINI.ReadString('Faktury','dopisek','');

Edit3.Text := KonfigINI.ReadString('Nazwa','dane1','');

Edit4.Text := KonfigINI.ReadString('Nazwa','dane2','');

Edit5.Text := KonfigINI.ReadString('Nazwa','dane3','');

Edit6.Text := KonfigINI.ReadString('Nazwa','dane4','');

Edit7.Text := KonfigINI.ReadString('Inne','miejsce_wystaw','');

Pierwsza instrukcja procedury powoduje otwarcie lub utworzenie pliku (jeżeli nie istnieje)/ Następnie instrukcje przyporządkowują właściwościom Text odpowiednich pól TEdit wartości odczytanych z pliku INI. Do odczytu służy metoda ReadString należąca do kalsy TIniFile. Metoda ta przyjmuje trzy parametry: nazwę sekcji, nazwę klucza oraz wartość domyślną, która zostanie użyta w przypadku braku odpowiedniego klucza w pliku. W przedstawionym przykładzie wartość domyślna jest proponowana tylko dla serii faktur. Klasa TIniFile jest zdefiniowana w module IniFiles. Moduł ten nie jest domyślnie dołączany do nowych formularzy, musimy więc zadeklarować jego użycie w klauzuli uses. Ponieważ w naszym przypadku operacja na pliku INI ma charakter lokalny, deklaracja może zostać umieszczona w sekcji implementation klasy TKonfigForm.

5.Umieść kursor na końcu listy klauzuli uses, znajdujący się na początku pliku Konfig.pas i dołącz do niej moduł IniFiles. Musimy jeszcze zdefiniować atrybut KonfigINI. Zmienna ta będzie wykorzystywana przez dwie procedury, lecz nie ma potrzeby jej udostępniania poza modułem Konfig, wystarczy więc, jeżeli zadeklarujemy ją w sekcji private

6.Utwórz wolny wiersz poniżej komentarza {private declarations} i wpisze :

KonfigINI : TIniFile;

7.Kliknij przycisk Save, aby zapisać wprowadzone zmiany

Zapis danych do pliku INI

Chcąc uzyskać w pełni funkcjonalny formularz, musimy zadbać o to ,aby po wprowadzeniu zmian w wartościach konfiguracyjnych zostały one zapisane w pliku INI. Odpowiednie czynności będą wykonywane po kliknięciu przycisku OK opracowywanego formularza

1.Przejdć ponownie do widoku formularza i dwukrotnie kliknij przycisk OK. Dwukrotne kliknięcie kontrolki typu TBitBtn oznacza przejście do edycji zdarzenia OnClick (twórcy Delphi słusznie założyli, że jest to najczęściej wykorzystania zdarzenie)

2.Uzupełnij kod procedury, aby wyglądała ona jak poniżej:

procedure TKonfigForm.OKBtnClick (Sender : TObject);

begin

with KonfigINI do begin

WriteString('Faktury', 'seria', Edit1.Text);

WriteString('Faktury', 'dopisek', Edit2.Text);

WriteString('Nazwa', 'dane1', Edit3.Text);

WriteString('Nazwa', 'dane2', Edit4.Text);

WriteString('Nazwa', 'dane3', Edit5.Text);

WriteString('Nazwa', 'dane4', Edit6.Text);

WriteString('Inne', 'miejsce_wystaw', Edit7.Text);

end; end;

Procedura WriteString przyjmuje trzy parametry : nazwę sekcji i klucza pliku INI oraz zapamiętywaną wartość. Zwróć uwagę na zastosowanie w powyższym kodzie konstrukcji with ... do. Dzięki niej unikamy wielokrotnego wpisywania nazwy klasy, z której pochodzi metoda WriteString (trzeba by pisać KonfigINI.WriteString)

3.Kliknij Save , aby zapisać wprowadzone zmiany

Wywołanie formularza konfiguracyjnego

Moduł Konfig.pas jest już gotowy, pozostało jeszcze napisać odpowiednie instrukcje, które będą wyświetlały formularz. Ponieważ wywołanie formularza KonfigForm będzie następowało z poziomu formularza głównego, musimy do niego powrócić

1.Na pasku narzędzi kliknij prycisk View Form i dwukrotnie kliknij pozycję MenuGlForm

2.Na formularzu kliknij menu Narzędzia i wybierz Polecenia Konfiguracja. Pojawi się powłoka procedury obsługi zdarzenia OnClick dla polecenia menu Konfiguracja1.

3.Uzupełnij tę procedurę , aby miała ona postać:

procedure TMenuGlForm.Konfiguracja1Click (Sender: TObject);

begin

WyswietlKonfiguracje;

end;

Oczywiście w tym miejscu można by od razu umieścić polecenia KonfigForm.ShowModal, ale ponieważ to samo działanie będzie realizować przycisk na pasku narzędzi, dobry styl programowania nakazuje utworzenie oddzielnej procedury. Procedura, której nagłówek nie jest generowany automatycznie, (jak to ma miejsce w przypadku wszystkich procedur obsługi zdarzenia), musi być w całości wpisana przez użytkownika. Należy również pamiętać o umieszczeniu jej deklaracji w definicji klasy

4.Kliknij przycisk Toggle Form/Unit i dwukrotnie kliknij ostatni przycisk paska narzędzi formularza (przedstawiający koło zębaty). Zostanie utworzona powłoka procedury zdarzenia OnClick dla przycisku paska narzędzi. Uzupełnij tę procedurę ,aby wyglądała następująco:

procedure TMenuGlForm.ToolButton6Click(Sender : TObject)

begin

WyswietlKonfiguracje;

end;

5.Ustaw kursor w wolnej linii przed słowem kluczowym end. na końcu modułu MenuGl.pas i wpisz poniższą procedurę:

procedure TMenuGlForm.WyswietlKonfiguracje(Sender : TObject)

begin

KonfigForm.ShowModal;

end;

6.Odszukaj definicję klasy TMenuGlForm i umieść nagłówek procedury przed słowem kluczowym private. Ten fragment kodu będzie wyglądał następująco:

procedure Konfiguracja1Click(Sender : TObject);

procedure WyswietlKonfigurcje(Sender : TObject);

private;

7.Kliknij przycisk Save, aby zapisać wprowadzone zmiany

8.Kliknij przycisk Run ,aby skompilować i uruchomić aplikację. Na ekranie pojawi się komunikat informujący ,że formularz MenuGlForm odwołuje się do formularza KonfigForm, lecz w tym pierwszym barak odpowiedniej deklaracji

9.Kliknij przycisk Yes, aby dołączyć brakujące deklarację

10.Ponownie kliknij przycisk Run

Jeżeli w kodzie nie ma błędów, program zostanie skompilowany i uruchomiony, Po przejściu do formularza z konfiguracją programu w poszczególnych polach, zobaczysz wartości pobrane z pliku INI

Podsumowanie

W ramach tego zadania opracowałeś menu główne oraz pasek narzędzi programu. Przygotowałeś również formularz służący do przeglądania i edycji danych konfiguracyjnych programu. Poznałeś też podstawowe zasady korzystania z plików w formacie INI. Właśnie w takim pliku są przechowywane dane wyświetlane w formularzu konfiguracyjnym. W kolejnej części rozpoczniemy tworzenie formularzy dla danych zgromadzonych w bazie(…)

FORMULARZE Z POJEDYNCZYM ŹRÓDŁEM DANYCH

Zajmiemy się teraz przygotowaniem formularzy do wyświetlenia listy faktur, kontrahentów i towarów/usług. Chociaż tak naprawdę utworzymy tylko jeden formularz. Dlaczego? Jeśli uważniej przeanalizujemy zadania naszej aplikacji, szybko zauważymy, że formularze wyświetlające wspomniane listy są bardzo podobne. Różnica sprowadza się jedynie do danych wyświetlanych na poszczególnych listach oraz do formularzy, które będą wywoływane, gdy użytkownik zażąda edycji pojedynczego rekordu. Oczywiście nic nie stoi na przeszkodzie, aby opracować oddzielne formularze jeden wyświetlający listę kontrahentów, drugi listę towarów a trzeci listę faktur. Pytanie jednak po co?. Rozwiązanie które zastosujemy daje dwie istotne korzyści:

* W naturalny sposób uzyskujemy jednorodność interfejsu aplikacji

* Zmniejszają się rozmiary kodu źródłowego

Tworzenie formularza za pomocą kreatora

Do utworzenia formularza z listą rekordów użyjemy bardzo wygodnego narzędzia, jakim jest kreator formularzy - Form Wizard. Generuje on formularz ze wszystkimi komponentami niezbędnymi do przeglądania i edycji rekordów pojedynczej tabeli lub zapytania. Następnie dokonamy niezbędnych modyfikacji , aby formularz ten spełniał wymogi naszej aplikacji .Zanim jednak przystąpimy do realizacji konkretnych zadań, wyjaśnijmy jak Delphi uzyskuje dostęp do danych zgromadzonych w bazie. Jak pamiętamy, korzystamy z tabel w standardzie Paradox, obsługiwanych przez aparat bazy danych BDE (Borland Database Engine). Dostęp do BDE z poziomu Delphi uzyskujemy przez komponenty wywodzące się z klasy TDataSet, są to TTable lub TQuery. W przypadku operacji na pojedynczych tabelach z reguły stosuje się pierwszy z tych komponentów - Ttable. Do jego właściwości należą m.in. : DatabaseName, określająca nazwę bazy danych (w przypadku tabel Paradox może to być ścieżka do katalogu lub alias) , a także TableName, przechowująca nazwę tabeli obsługiwanej przez ten komponent. Pomiędzy obiektami TTable a kontrolkami wyświetlanymi na formularzu istnieje jeszcze jeden element pośredniczący - komponent TDataSource. Służy od do odseparowania źródła danych od kontrolek danych, co znacznie zwiększa elastyczność aplikacji. Za pomocą kreatora utwórzmy teraz formularz, w którym będzie wyświetlana lista kontrahentów:

1.Uruchamiamy środowisko Delphi

2.W menu File wybieramy polecenie Open, odszukujemy katalog z naszymi plikami, otwieramy plik projektu Faktury. Na ekranie pojawi się formularz startowy projektu MenuGlForm

3. W menu Database wybieramy polecenie Form Wizard. Pojawi się pierwsze okno dialogowe kreatora.

4.Upewnij się ,że zaznaczone są opcje Create a simple form (tworzenie formularza z jednym źródłem danych) i Create form using TTable objects (tworzenie formularza opartego na tabeli), a następnie kliknij Next

5.W polu Directories otwórz folder Dane

6.Na liście Table name zaznacz plik tabeli Kontrah, a następnie kliknij Next

7.Za pomocą przycisku oznaczonego symbolem >> przenieś wszystkie pola tej tabeli na listę Ordered Selected Fields

8.Kliknij Next

9.Zaznacz opcję In a grid i kliknij przycisk Next .Wybór tej opcji oznacza ,że dane będą prezentowane w kontrolce typu TDBGrid, a więc w postaci siatki, w której każdemu wierszowi odpowiada jeden rekord tabeli, a każdej kolumnie - pojedyncze pole rekordu

10.Upewnij się ,że opcja Generate a main form jest nieaktywna, zaznacz pcję Form and data module, a następnie kliknij Finish

Zostanie wygenerowany formularz zawierający obiekty typu TDBGrid oraz TDBNavigator. Obiekt TDBNavigator służy do wykonywania podstawowych operacji na zbiorze danych, takich jak przewijanie (cztery pierwsze przyciski), wstawianie, usuwanie, rozpoczynanie edycji, zatwierdzanie, anulowanie zmian i odświeżanie rekordów (kolejnych pięć przycisków) W razie potrzeby można ukryć niektóre przyciski, ustawiając odpowiednie wartości właściwości VisibleButtons obiektu TDBNavigator na False. Oprócz formularza na ekranie pojawi się moduł danych. Zaznaczenie pola wyboru Generate a main form w ostatnim oknie kreatora spowodowałoby utworzenie formularza głównego. Z kolei wybór opcji Form only pozwala na utworzenie pojedynczego modułu, w którym jest umieszczany zarówno interfejs formularza, jak i obiekty dostępu do danych. Choć takie rozwiązanie z technicznego punktu widzenia jest zupełnie prawidłowe, zaleca się tworzenie oddzielnego modułu danych (data module) co pozwala na odseparowanie interfejsu formularza od przetwarzanych danych. Takie podejście ma dwie zalety. Po pierwsze poprawia skalowalność aplikacji, a po drugie - pozwala nam stworzyć uniwersalny formularz, który po umieszczeniu w repozytorium może być wykorzystywany w innych aplikacjach. Wybór opcji Form and data module spowodował powstanie dodatkowego modułu danych (o domyślnej nazwie DataModule1), w którym znalazły się komponenty Table1 i DataSource1. W tej chwili w utworzonym formularzu nie widzimy żadnych rekordów, ponieważ właściwość Active komponentu Table1 ma wartość False. Jeśli w fazie projektowej chcesz wyświetlić rekordy tabeli, z którą są powiązane kontrolki danych, ustaw tę właściwość na True. W trakcie działania aplikacji , tabela będzie automatycznie otwierana w chwili tworzenia modułu danych - zadbał o to kreator, wstawiając odpowiednią procedurę w kodzie modułu danych :

Procedurę TDataModule1.DataModuleCreate(Sender : TObject)

begin

Table1.Open;

end;

Wywołanie metody Open obiektu TTable jest równoznaczne z przypisaniem właściwości Active wartości True.

11.Zaznacz komponent Table1 w module danych DataModule1, przejdź do Object Inspectora i zmodyfikuj właściwość DatabaseName na BAZYFAKT. Niestety kreator nie radzi sobie ze ścieżkami względnymi (np. "\dane") dlatego w trakcie jego działania jako źródło podaliśmy bezwzględną lokalizację pliku. Oczywiście aplikacja będzie bardziej elastyczna, jeżeli w tym miejscu zastosujemy alias.

12.Przejdź do właściwości Name nowego formularza i wpisz ListaForm

13.Przejdź do właściwości Name nowego modułu danych i wpisz KontrahDModule

14.Kliknij przycisk Save All. Zapisz nowy formularz jako plik Lista.pas (zapis formularza będzie proponowany jako pierwszy), natomiast modułu danych jako plik KontrahD.pas

>

Wstawianie formularza do repozytorium

Utworzony przed chwilą formularz chcemy wykorzystać również do wyświetlania list towarów oraz faktur. Najprostszym rozwiązaniem wydaje się skopiowanie odpowiednich plików *.dfm i *.pas a następnie zmodyfikowanie właściwości TableName w obiektach TTable powielonych modułów danych. W ten sposób uzyskaliśmy trzy całkowicie niezależne formularze. Jednak droga ta może prowadzić na manowce. Rozważmy najprostszy przykład. Po pewnym czasie doszliśmy do wniosku ,że pasek nawigatora powinien się znajdować nieco bardziej na prawo. Jeżeli zastosujemy powyższe rozwiązanie będzie to oznaczało konieczność dokonania zmian we wszystkich trzech formularzach. Co więcej, nie będziemy mieli pewności, że w każdym z tych formularzy dokonamy takiego samego przesunięcia. Pół biedy, gdy takie modyfikacje dotyczą tylko wyglądu formularzy - w ostateczności uzyskamy niejednorodny interfejs, który co najwyżej będzie irytował użytkownika. Może się jednak zdarzyć ,że zajdzie potrzeba wprowadzenia jakichś zmian do uniwersalnego kodu tego formularza, a wtedy już tylko krok do popełnienia błędu lub przeoczenia. Problemów tych możemy uniknąć w prosty sposób, wystarczy zapisać formularz jako szablon. W praktyce oznacza to zapamiętanie go w repozytorium obiektów. W repozytorium umieszczamy obiekty wielokrotnie wykorzystywane dla różnych aplikacji. Na przykład możemy dojść do wniosku ,że jakiś formularz jest na tyle doskonały ,że warto umieścić go w repozytorium. Obiekty wstawione do repozytorium powinny mieć charakter uniwersalny. Aby za taki można było uznać nasz formularz listy, musimy najpierw usunąć odwołania kontrolek danych do źródła danych w module KontrahDModule (każdy z formularzy potomnych będzie korzystał z innego modułu danych)

1.Zaznacz kontrolki DBNavigator i DBGrid , przejdź do Object Inspectora, a następnie usuń zawartość właściwości DataSource

2.Przejdź do kodu formularza ListaForm i usuń deklarację uses Unit1 z sekcji implementation. Po tych operacjach formularz przestanie być powiązany z konkretnym źródłem danych

3.Zapisz wprowadzone zmiany<br

4.Prawym przyciskiem myszki kliknij w dowolnym miejscu formularza ListaForm.

5.W menu podręcznym wybierz polecenie Add To Repository. Na ekranie pojawi się okno dialogowe o takiej samej nazwie.

6.Wypełnij go : Title - Lista; Description - Lista rekordów ; Page - Forms Author - ????. Repozytorium jest standardowo podzielony na kilka stron (można tworzyć własne), na których są pogrupowane obiekty różnych kategorii. W naszym przykładzie zastosowaliśmy stronę Forms

7.Kliknij przycisk OK. Formularz zostanie umieszczony w repozytorium obiektów. Oznacza to, że będzie można go wielokrotnie wykorzystywać w tej i innych aplikacjach. Sprawdźmy teraz, czy operacja umieszczenia w repozytorium powiodła się .

8.W menu Tools wybierz polecenie Repository

9.Na liście Pages zaznacz pozycję Forms. Na liście Objects pojawią się wszystkie elementy zapamiętane na stronie Forms repozytorium, a wśród nich szablon formularza ListaForm. Okno dialogowe umożliwia reorganizowanie repozytorium (tworzenie i usuwanie stron, przemieszczanie obiektów między stronami itp.)

10.Kliknij Save All aby zapisać wprowadzone zmiany. Jak sama nazwa wskazuje formularz ListaForm jest teraz szablonem - nie będziemy więc go bezpośrednio wykorzystywać w aplikacji, lecz użyjemy do

stworzenia trzech formularzy pochodnych, w których znajdą się oddzielne listy kontrahentów, towarów i faktur.

Pobieranie formularza z repozytorium

Istnieją trzy sposoby korzystania z formularzy umieszczonych w repozytorium:

* Kopiowanie formularzy

* Dziedziczenie po formularzach

* Operowanie bezpośrednio na formularzach

Pierwsza z wymienionych metod nie różni się niczym od skopiowania plików DFM i PAS formularza, jest tylko nieco wygodniejsza, gdy wielokrotnie powtarzamy ten proces. Otrzymany w ten sposób formularz jest całkowicie niezależną kopią swojego pierwowzoru. Druga metoda jest o wiele bardziej elastyczna, gdyż prowadzi do powstania formularza dziedziczącego po formularzu-rodzicu. Co to oznacza w praktyce? Wszelkie zmiany wprowadzane w klasie macierzystej będą automatycznie odzwierciedlone w klasach potomnych. Jeżeli na przykład w formularzu - rodzicu przesuniemy pasek nawigatora, o taką sama wartość przesunie się on w formularzach potomnych. Jeżeli we wzorcowym formularzu wstawimy nowy przycisk, pojawi się on natychmiast w jego kopiach. Czyż nie o to nam chodziło? Dzięki dziedziczeniu w prosty sposób uzyskamy jednorodny interfejs. Trzecia metoda polega ba wprowadzaniu zmian bezpośrednio w formularzu umieszczonym w repozytorium. Zmiany będą odzwierciedlone we wszystkich projektach wykorzystujących ten formularz .Zanim wstawimy nowy formularz listy, musimy nieco "oszukać" środowisko Delphi. Niestety nie radzi ono sobie dobrze w sytuacji, gdy w projekcie umieszczamy kilka formularzy dziedziczących po jednym formularzu z repozytorium. Problem polega na tym ,że wstawienie formularza w trybie dziedziczenia oznacza również konieczność wstawienia formularza nadrzędnego - w naszym przypadku jest to ListaForm. Gdy spróbujemy wstawić formularz dziedziczący, pojawi się komunikat "The project already contains a form or module named Lista" (projekt zawiera już formularz lub moduł o nazwie Lista). Oto jeden ze sposobów poradzenia sobie z tym problemem.

1.Kliknij przycisk View Unit, a następnie na liście wybierz plik projektu Faktury. Na ekranie pojawi się główny plik projektu

2.W edytorze usuń wiersz w klauzuli uses związany z modułem Lista.pas. Usuń również wywołanie Applcation.CreateForm(TListaForm,ListaForm), aby moduł projektu Faktury.dpr uzyskał postać :

```
program Faktury;

uses

Forms,

MenuGl in 'MenuGl.pas' {MenuGlForm},

Konfig in 'Konfig.pas' KonfigForm),

KontrahD in 'KontrahD.pas' {KontrahDModule : TDataModule};

{$R *.res}

begin

Application.Initialize;
```

Application.CreateForm(TMenuGlForm, MenuGlForm);

Application.CreateForm(TKonfigForm, KonfigForm);

Application.CreateForm(TKontrahDModule, KontrahDModule);

Application.Run;

end.

Zwróć uwagę aby klauzula uses kończyła się średnikiem! Spróbujmy teraz utworzyć formularz listy oparty na formularzu z repozytorium

3.W menu File, New wybierz polecenie Other. Pojawi się okno dialogowe NewItems

4.Przejdź do karty Forms i zaznacz ikonę Lista. Zawartość tego okna w rzeczywistości jest zawartością repozytorium, widzimy więc tutaj ikonę symbolizującą formularz Lista

5.Zaznacz opcję Inherit, a następnie kliknij przycisk OK. Wybór tej opcji spowoduje utworzenie nowego formularza dziedziczącego po klasie TListaForm. Możemy się o tym przekonać , sprawdzając moduł formularza.

6.Przejdź do widoku kodu modułu formularza i odszukaj definicję jego klasy. Powinna ona wyglądać następująco

type

TListaForm1 = class(TListaForm)

private

{Private declarations}

public

{Public declarations}

end;

Gdybyśmy pozostawili opcję Copy, powstałaby klasa formularza dziedzicząca bezpośrednio po klasie TForm i zbudowana identycznie jak klasa TListaForm, zapamiętana w repozytorium:

type

TListaForm1 = class(TListaForm)

DBGrid1 : TDBGrid;

DBNavigator : TDBNavigator;

Panel1 : TPanel;

Panel2: TPanel;

private

{Private declarations}

public

{Public declarations}

end;

Przypisanie źródła danych formularzowi pochodnemu

Kolejnym zadaniem do wykonania jest ponowne powiązanie formularza z modułem danych. Jak pamiętasz ,ze względu na uniwersalny charakter formularza ListaForm, usunęliśmy powiązania jego komponentów bazodanowych ze źródłem danych. Oznacza to ,że w formularzach pochodnych musimy jeszcze raz zdefiniować te wartości.

1.Naciśnij klawisz F12, aby przenieść się do widoku kodu formularza i w sekcji implementation (za dyrektywą {$R *.DFM}) umieść deklarację :

uses KontrahD;

2.W Object Inspectorze odszukaj obiekt formularza KontrahLForme, przejdź do karty Events, a następnie dwukrotnie kliknij w polu procedury zdarzenia OnCreate dla tego formularza

3.Uzupełnij powstałą procedurę obsługi zgodnie z poniższym:

procedure TKontrahLForm.FormCreate(Sender : TObject);

begin

inherited;

DBNavigator.DataSource : = KontrahDModule.DataSource1;

DBGrid1.DataSource := KontrahDModule.DataSource1;

end;

Wartości te moglibyśmy przypisać w fazie projektwej , korzystając z Obiekt Inspectora. Niestety pojawiają się tu pewne niekorzystne konsekwencje zastosowania dziedziczenia formularzy. Jeżeli w jakiś sposób zmieniasz formularz macierzysty (ListForm), jego właściwości zostaną odwzorowane w formularzach pochodnych. Ponieważ w formularzu macierzystym usunęliśmy wartości DataSource z kontrolki nawigatora i siatki, może to spowodować to usunięcie tych właściwości w kontrolkach formularzy pochodnych. Ponadto opcja KontrahDModule.DataSource1 w polu właściwości DataSource komponentów formularza będzie niedostępna, jeżeli uprzednio nie otworzysz pliku, w którym znajduje się wskazywane źródło danych. W powyższym przykładzie, jeśli plik KontrahD.pas nie jest otwarty (nie widać jego zakładki w edytorze kodu), kliknij przycisk View Unit i wybierz moduł KontrahD. Aby uniknąć tych niedogodności , przypisywanie źródła danych będzie się odbywało podczas każdego tworzenia listy, w procedurze obsługi zdarzenia OnCreate. Pozostało nam jeszcze do wykonania kilka prostych czynności porządkowych, takich jak nazwanie nowego formularza, nadaniu mu etykiety i zapisanie zmian na dysku.

4.Przejdź do widoku View Form nowego formularza

5.W Object Inspectorze odszukaj właściwość Name z zmień ja na KontrahLForm

6.Odszukaj właściwość Caption i zmień ją na Lista kontrahentów

7.Kliknij przycisk Save i zapisz moduł formularza jako plik KontrahL

Utworzenie dodatkowych źródeł danych

Zanim w analogiczny sposób utworzysz formularze wyświetlające listy towarów/usług i faktur, musimy najpierw zdefiniować odpowiednie moduły danych. Tym razem nie skorzystamy z pomocy kreatora

1.W menu File, New, wybierz polecenie Data mdule

2.Do nowego modułu wstaw komponent TTable (z palety BDE) oraz TDateSource (z palety Data Access)

3.W Object Inspectorze zmień właściwości obiektu Table1:

* DatabaseName na BAZAFAKT

* TableName na TowUslug

4.Dwukrotnie kliknij ikonę komponentu Table1. Na ekranie pojawi się okno z listą pól. W tej chwili powinno być ono puste.

5.Prawym przyciskiem myszy kliknij w pustym oknie i z menu podręcznego wybierz polecenie Add all fields. Na liście pojawią się wszystkie pola tabeli TowUslug

6.Zamknij okno z listą pól

7.W Object Inspectorze wybierz właściwość Active obiektu Table1 i ustaw ją na True. Ustawienie tej właściwości na True spowoduje ,że tabela będzie dostępna już w fazie prjektowej. Równoważna dla tej właściwości jest metoda Open, obiektu TTable; z reguły wywołuje się ją w procedurze zdarzenia Create formularza. W kodzie programu można by więc zapisać:<br.>

Table1.Active := true; lub po prostu Table1.Open

8.Zaznacz obiekt DataSource1 i ustaw jego właściwość DataSet na Table1

9.Zaznacz okno modułu danych i zmień jego nazwę (właściwość Name) na TowarDModule

10.Kliknij przycisk Save i zapisz utworzony moduł jako plik TowarD.pas

11.Zamknij okno modułu danych

Powtórz czynności z punktów 1- 11 ,aby zdefiniować moduł danych dla tabeli z nagłówkami faktur. W punkcie 3 jako nazwę tabeli (właściwość TableName) podaj TranNagl. Zmień nazwę modułu na FaktDModule i zapisz go jako plik FaktD.pas

Tworzenie pozostałych formularzy

Czas na nieco samodzielnej pracy. W analogiczny sposób jak w poprzedniej części musisz wstawić do aplikacji kolejne dwa formularze - do wyświetlenia list towarów oraz faktur. Ponieważ należy wykonywać identyczne czynności jak powyżej, nie będziemy powielali ich opisu. Zmienią się jedynie nazwy utworzonych obiektów i niektóre ich właściwości. Nie zapomnij od opcji Inherit podczas wstawiania formularza dziedziczącego. Pamiętaj też aby przed każdą operacją wstawienia takiego formularza usuną wiersz Lista in 'LISTA.pas' z klauzuli uses modułu projektu. Po utworzeniu formularza dla listy towarów zmień jego właściwości:

* Name na TowarLForm

* Caption na Lista towarów i usług

W kodzie modułu dołącz klauzulę uses TowarD Utwórz procedurę zdarzenia OnCreate dla formularza TowarLForm i na jej końcu wstaw polecenia:

DBNavigator.DataSource := TowarDModule.DataSource1;

DBGrid1.DataSource := TowarDModule.DataSource1;

Zapisz formularz jako plik TowarL. Po utworzeniu formularza dla listy faktur zmień jego właściwości:

* Name na FaktLForm

* Caption na Lista faktur

W kodzie modułu dołącz klauzulę uses FaktD. Utwórz procedurę zdarzenia OnCreate dla formularza TowarLForm i na jej końcu wstaw polecenia:

DBNavigator.DataSource := FaktDModule.DataSource1;

DBGrid.DataSource := FaktDModule.DataSource1;

Zapisz formularz jako plik FaktL. Nasza aplikacja wzbogaciła się o kolejne trzy moduły danych, Jak widać, jej złożoność wzrosła w stosunku do stanu z poprzedniej części. Jednak dzięki zachowaniu spójnej konwencji nazewniczej, struktura modułów i formularzy jest przejrzysta. "Sztuczka" z usuwaniem modułu Lista może spowodować nieco zamieszania w klauzuli uses modułu projektu. Sprawdź czy ma on prawidłowa budowę:

1.W oknie edytora kodu przejdź do zakładki Faktury. Tekst pliku projektu Faktury.dpr powinien mieć postać:

program Faktury;

uses

Forms, MenuGl in 'MenuGl.pas' {MenuGlForm},

Konfig in 'Konfig.pas' {KonfigForm},

Lista in 'LISTA.pas' {ListaForm},

KontrahD in 'KontrahD.pas' {KontrahDModule : TDataModule},

KontrahL   in 'KontrahL.pas' {KontrahLForm},

TowarL in 'TowarL.pas' {TowarLForm},

TowarD   in 'TowarD.pas' {TowarDModule : TDataModule},

FaktD in 'FaktD.pas' {FaktDModule : TDataModule},

FaktL in 'FaktL.pas' {FaktLForm},

{$R *.res}

begin

Application.Initialize;

Application.CreateForm(TMenuGlForm, MenuGlForm);

Application.CreateForm(TKonfigForm, KonfigForm);

Application.CreateForm(TKontrahDModule, KontrahDModule);

Application.CreateForm(TKontrahLForm, KOntrahLForm);

Application.CreateForm(TTowarDModule, TowarDModule);

Application.CreateForm(TTowarLForm, TowarLForm);

Application.CreateForm(TFaktDmodule, FaktDModule);

Application.CreateForm(TFaktLForm, FaktLForm);

Application.Run;

end.

Kolejność modułów może być dowolna z dwoma wyjątkami:

* Pierwsza metoda CreateForm powinna tworzyć formularz startowy aplikacji (w naszym przykładzie MenuGlForm)

* Tworzenie modułów danych powinno poprzedzać tworzenie formularzy, z którymi te moduły są związane. W przeciwnym razie podczas tworzenia formularza może dojść do wywołania metody Open obiektu TTable, gdy nie został on jeszcze utworzony

2.Wprowadź ewentualne zmiany i kliknij przycisk Save All

3.Skompiluj zmodyfikowany kod, aby sprawdzić czy nie zawiera błędów

Powiązanie nowych formularzy z formularzem głównym

Utworzone przed chwilą formularze powinny być wywoływane z poziomu formularza głównego MenuGlForm poprzez wybranie polecenia menu lub kliknięcie przycisku na pasku narzędzi. Musimy więc zdefiniować odpowiednie procedury w formularzu głównym aplikacji. Będziemy postępować tak samo jak w przypadku formularza KonfigForm. Wspólnie zdefiniujemy odpowiednie mechanizmy dla formularza KontrahForm. Powiązanie pozostałych formularzy z formularzem głównym pozostawimy do samodzielnej realizacji

1.Na pasku narzędzi kliknij przycisk VewForm i wybierz formularz MenuGlForm.

2.Dwukrotnie kliknij drugi przycisk paska narzędzi (przedstawiający segregatory), uzupełnij procedurę obsługi zdarzenia, aby wyglądała następująco:

procedure TMenuGlForm.ToolButton3Click(Sender : TObject);

begin

WyswietlKontrah;

end;

3.Powróć do formularza, klikając przycisk Toggle Form/Unit , w menu Słwniki wybierz polecenie Kontrahenci, a następnie uzupełnij procedurę obsługi zdarzenia aby wyglądała następująco:

procedure TMenuGlForm.Kontrahenci1Click(Sender : TObject);

begin

WyswietlKontrah;

end;

4.Poniżej utworzonej przed chwilą procedury wpisz:

procedure TMenuGlForm.WyswietlKontrah;

begin

KontrahLForm.ShowModal;

end;

5.Przejdź do definicji klasy TMenuGlForm i wstaw w niej deklarację procedury (przed słowem kluczowym private):

procdure WyswietlKontrah;

6.W klauzuli uses w sekcji implementation dołącz moduł KontrahL. Klauzula ta powinna mieć teraz postać :

uses Konfig, KontrahL;

7.Powtórz czynności z punktów 1-6 dla formularzy TowarLForm i FaktLForm. Tworząc procedury w punkcie 4 nazwij je odpowiednio WyswietlTowar i WyswietlFakt

8.Kliknij przycisk Save, aby zapisać wprowadzone zmiany

9.Skompiluj i uruchom aplikację

Sprawdź czy wszystkie formularze działają prawidłowo. Do ich wyświetlenia użyj opcji menu oraz przycisków paska narzędzi. Spróbuj wpisywać przykładowe dane w listach towarów i kontrahentów. Obserwuj jak zmienia się stan przycisków nawigatora.

Modyfikacja właściwości pól

Jeżeli zgodnie z zaleceniami poświęciłeś nieco czasu na zapoznanie się z działaniem nowego formularza, z pewnością dostrzegłeś kilka elementów, które warto by zmienić. Przede wszystkim nagłówki poszczególnych kolumn siatki, choć zgadzają się z nazwami pól tabel, z pewnością nie wyglądają zbyt ładnie w formie skrótów, w dodatku pozbawionych polskich znaków diakrytycznych. Ponadto niektóre kolumny mogłyby być nieco węższe, co pozwoliłoby wyświetlić więcej informacji na ekranie, a inne - całkowicie ukryte przed użytkownikiem, na przykład kolumna IDTranNagl z tabeli TranNagl (wartości tej kolumny będą wykorzystywane wewnętrznie przez aplikację i nie ma potrzeby kłopotać nimi użytkownika) Odpowiednie zmiany uzyskamy, modyfikując pola obiektu TTable

1.Przejdź do widoku modułu KontrahDmodule

2.Dwukrotnie kliknij obiekt Table1. Na ekranie pojawi się okno Field Editor z listą pól zdefiniowanych w obiekcie Table1. Powinny być to wszystkie pola tabeli Kontrah. Jeżeli na liście brak któregoś pola lub chcesz usunąć jakieś pole z listy, kliknij ją prawym przyciskiem myszki i wybierz odpowiednie polecenie (Add fields lub Delete)

3.Kliknij pole IDKontrah, przejdź do Object Inspector i zmień właściwość DisplayLabel na Kod

4.Kliknij pole nazwa, przejdź do Object Inspectora i zmień właściwość DisplayWidth na 40. Właściwość DisplayWidth określa liczbę znaków wyświetlanych jednocześnie na ekranie. Nie wpływa ona na liczbę znaków przechowywanych w tabeli ani też na liczbę znaków, które można wpisać w kontrolce ekranowej

5.W analogiczny sposób zmodyfikuj szerokość pól Miasto i Ulica

6.Kliknij pole Telefon_kom, a następnie przejdź do Object Inspector i zmień właściwość DisplayLabel na Tel.komórkowy. Ta wartość będzie wyświetlana w nagłówku kolumny w komponencie siatki

7.Zaznacz jednocześnie pole Kontakt i Odbier_fakture i Object Inspectorze ustaw ich właściwość Visible na False. To ustawienie spowoduje, że obydwa pola nie będą wyświetlane w komponencie siatki

8.Kliknij przycisk Save ,aby zapisać wprowadzone zmiany

Odwoływanie się do pól obiektu typu DataSet

Jeżeli pola tabeli lub zapytania zostaną dodane do listy pól, jest to równoznaczne ze zdefiniowaniem na poziomie klasy odpowiednich obiektów TFielld (w rzeczywistości będą obiekty typów przypisanych do poszczególnych typów danych w zestawie. Np. poniżej mamy kilka definicji pól dla tabeli TranNagl:

Table1Seria : TStringField;

Table1Numer : TIntegerField;

Table1Dopisek: TStringField;

Table1Data_wystaw: TDateField;

Jeżeli np. chcemy przypisać polu Seria wartość "FA", wystarczy użyć konstrukcji:

Tabel1Seria.AsString := "FA"

Takie rozwiązanie jest wygodniejsze , jeżeli zależy nam na możliwości edytowania właściwości tych pól w fazie projektowej. Czasami jednak, gdy chcemy zachować większą elastyczność kodu, lub gdy korzystamy tylko z niewielu pól zestawu danych, rezygnujemy z definiowania obiektów przedstawiających te pola. Lista pól jest wtedy pusta. Mimo to pola tabeli lub zapytania są dostępne - zwykle odwołujemy się do nich stosując metodę FiledByName zdefiniowaną w klasie TDataSet. Na przykład

Table1.FieldByName('Seria'),AsString := "FA"

Czynności opisane w punktach 1-8 wykonaj dla pozostałych komponentów TTable umieszczonych w modułach danych TowarDmodule i FaktDModule. Modyfikując właściwości, skorzystaj z poniższych wartości:

Pole : IDTowUslug

Właściwość : DisplayLabel

Nowa wartość : Kod

Pole : J_miary

Właściwość : DisplayLabel

Nowa wartość : J.m

Pole : Stan_min

Właściwość : DisplayLabel

Nowa wartość : Stan min

Pole : Cena_zak

Właściwość : DisplayLabel

Nowa wartość : Cena zakupu

Pole : Cena_sprz

Właściwość : DisplayLabel

Nowa wartość : Cena sprzed.

Pole : St_VAT

Właściwość : DisplayLabel

Nowa wartość : St. VAT

Pole : Data_wystaw

Właściwość : DisplayLabel

Nowa wartość : Data wystaw

Pole : Data_trans

Właściwość : DisplayLabel

Nowa wartość : Data sprzed.

Pole : Term_plat

Właściwość : DisplayLabel

Nowa wartość : Termin płatn.

Pole : Forma_platn

Właściwość : DisplayLabel

Nowa wartość : Forma płatn.

Pole : IDKOntrah

Właściwość : DisplayLabel

Nowa wartość : Forma płatn.

Pole : IDTranNagl

Właściwość : Visible

Nowa wartość : False

Zwróć uwagę ,że w tabeli TranNagl (moduł FaktDModule) ukrywamy pole z identyfikatorem transakcji. Pole to służy jedynie do powiązania nagłówka transakcji z pozycjami transakcji - dla użytkownika nie niesie ono żadnych informacji a więc nie powinno być wyświetlane. Na zakończenie skompiluj i uruchom aplikacje .Sprawdź efekt wprowadzonych modyfikacji

Definiowanie maski edycji

Jak już wspomnieliśmy, chcemy aby identyfikatory kontrahentów i towarów/usług były wyświetlane wielkimi literami. W tym celu podczas opracowywania struktury tabeli zdefiniowaliśmy dla odpowiednich pól właściwość Picture, umożliwiającą formatowanie wprowadzonego tekstu. Niestety mechanizm ten jest dostępny tylko z poziomu programu Database Desktop. Jeżeli chcemy, aby takie same reguły obowiązywały również podczas wypełniania rekordów z poziomu formularzy , musimy je ponownie zdefiniować w aplikacji. Umożliwia to właściwość EditMask

1.Przejdź do widoku modułu KontrahDModule

2.Dwukrotnie kliknij obiekt Table1

3.Zaznacz pole IDKontrah

4.Przejdź do Object Inspectora i we właściwości EditMask wpisz następujący ciąg znaków : >aaaaaaaaaaaa;1;_

Jest to tzw. maska edycji. Składa się ona z trzech części rozdzielonych średnikami. Pierwsza część określa samą maskę. W powyższym przykładzie symbol większości oznacza ,że wszystkie wpisane znaki alfabetu będą przekształcane na wielkie litery. Dwanaście małych liter "a oznacza ,że w polu edycji można wpisać co najwyżej tyle znaków. W ten sposób formułujemy warunek ,że kod kontrahenta może się składać maksymalnie z dwunastu znaków. Wartość 1 za średnikiem oznacza ,że w odpowiednim polu bazy danych będą wpisywane tylko wartości wprowadzane przez użytkownika. Np. jeżeli maska edycji ma postać 00-000;1;_, to po wpisaniu wartości 44-310 zostanie ona zapamiętana w polu bazy danych wraz z elementem stałym (w tym przypadku jest nim łącznik), a więc 44-310. Gdyby maska miała postać 00-000;0;_, wtedy w polu bazy danych znalazłaby się wartość 44310. Ostatni element maski określa znak jaki będzie się pojawiał w jeszcze niewypełnionych pozycjach maski. Jeżeli na przykład wprowadzimy kod kontrahenta ADA, to w polu tekstowym będzie on widoczny jako ADA_____. Nie zawsze to dobrze wygląda, dlatego zamiast znaku podkreślenia można użyć spacji

5.Kliknij przycisk Save, aby zapisać wprowadzone zmiany

6.Skompiluj u uruchom aplikację, Przejdź do formularza z listą towarów/usług. Spróbuj wpisać dowolny kod towaru. Jak widać , tekst jest wpisywany wielkimi literami. Spróbuj samodzielnie zdefiniować taką samą maskę edycji dla kodu towaru (obiekt Table1 w module danych TowarDModule)

Tworzenie formularza do edycji pojedynczego rekordu

Dane mogą być wyświetlane w formie listy lub pojedynczych rekordów. Praca z listą nie zawsze jest wygodna. Z jednej strony dzięki niej widzimy zawsze więcej rekordów, z drugiej jednak, jeżeli zawierają one zbyt wiele pól, nie wszystkie mieszczą się na ekranie, Nie można definitywnie stwierdzić ,że któraś w wymienionych form prezentacji danych jest lepsza, Ich przydatność wynika z potrzeb użytkownika, dlatego dobra aplikacja bazodanowa powinna oferować obydwie możliwości. Opracujemy teraz formularz do edycji pojedynczego rekordu z tabeli Kontrah

1.W menu Database wybierz polecenie FormWizard. Pojawi się pierwsze okno kreatora

2.Upewnij się ,że zaznaczone są opcje Create simple form i Create a form using TTable objects i kliknij przycisk Next<br

3.W kolejnym oknie kreatora na liście Directories przejdź do folderu Dane, na liście Table name zaznacza tabelę Kontrah i kliknij przycisk Next

4.W kolejnym oknie kreatora kliknij przycisk >>, aby przenieść wszystkie pola tabeli na listę Ordered Selected Fields. Kliknij przycisk Next

5.W następnym oknie kreatora upewnij się,że zaznaczona jest opcja Horizontally i kliknij pzycisk Next. Pola na tworzonym formularzu zostaną umieszczone w wierszach jedno obk drugiego Etykiety opisujące przeznaczenie poszczególnych pól zostaną umieszczone ponad nimi. Druga opcja (Verticaly) powoduje umieszczanie pól jedno pod drugim

6.W ostatnim oknie kreatora upewnij się ,że pole wyboru Generate a main form nie jest zaznaczone oraz ,że jest aktywna opcja Form only. Na ekranie pojawi się gotowy formularz d edycji pojedynczego rekordu tabeli Kontrah. Oprócz kontrolek typu TDBEdit widać na nim pasek nawigacyjny oraz komponenty TTable i TDataSource. W tym przypadku zrezygnowaliśmy z tworzenia oddzielnego modułu danych - ten formularz będzie przeznaczony wyłącznie do edycji danych kontrahentów, a więc odseparowanie źródła danych od interfejsu nie przyniosłoby na istotnych korzyści. Oczywiście utworzony w ten sposób formularz warto jeszcze nieco doszlifować , dobierając odpowiednie wielkości czcionek, rozmieszczając komponenty itp.

7.Gdy zaznaczony jest nowo utworzony formularz, przejdź do karty Properties Object Inspectora i zmień właściwość Caption na Dane kontrahenta oraz właściwość Name na KontrahForm.

8.Kliknij przycisk Save i zapisz moduł nowego formularza jako plik Kontrah.pas.

Wykonując czynności opisane w punktach1-8, w analogiczny sposób przygotuj formularz do edycji pojedynczych rekordów z tabeli TowUslug. Po utworzeniu formularza zmień jego nazwę na TowarForm. Zapisz go jak Towar.pas.

Powiązanie formularza do edycji pojedynczego rekordu z formularzem-listą

Formularze do edycji pojedynczych rekordów z tabel kontrahentów i towarów są już gotowe, musimy je tylko ze sobą powiązać. Dobrze jest, gdy aplikacja oferuje kilka dróg osiągnięcia tego samego celu. Na przykład w wielu popularnych edytorach tekstu zmianę czcionki można uzyskać za pomocą menu, paska narzędzi, menu podręcznego lub klawisza skrót. Taka wszechstronność pozwala na dużą elastyczność w użytkowaniu aplikacji i jest ceniona zwłaszcza przez zaawansowanych użytkowników, dążących zwykle do optymalizacji wykonywanych czynności. W naszej aplikacji wprowadzimy dwa sposoby przejścia do edycji pojedynczego rekordu: poprzez kliknięcie odpowiedniego przycisku w formularzu listy oraz poprzez podwójne kliknięcie wybranego wiersza na liście (to drugie rozwiązanie prawdopodobnie będzie preferowane przez zaawansowanych użytkowników aplikacji ponieważ pozwoli im zmniejszyć liczbę wykonywanych czynności). Najpierw jednak musimy umieścić na formularzach odpowiedni przycisk. Przy okazji będziemy mieli możliwość docenienie zalet korzystania z formularzy dziedziczonych z repozytorium - przycisk wstawiamy tylko w formularzu - rodzicu, co spowoduje jego automatyczne pojawienie się we wszystkich formularzach potomnych.

1.Na pasku narzędzi kliknij przycisk View Form

2.Dwukrotnie kliknij pozycję ListaForm. Na ekranie pojawi się formularz macierzysty dla wszystkich formularzy list używanych w aplikacji.

3.Na palecie Additional wybierz komponent TBitBtn i umieść go na formularzu obok paska nawigatora

4.W Object Inspectorze zmień właściwość Name przycisku na EdytujRekord-BitBtn oraz właściwość Caption na &Edytuj rekord

5.Kliknij przycisk Save.

Musimy teraz wprowadzić kod wyświetlający odpowiednie formularze. Ponieważ w zależności od typu listy będziemy wyświetlać różne dane, odpowiedni kod zdefiniujemy już nie w formularzu wzorcowym , lecz w poszczególnych formularzach list

1.Wyświetl na ekranie formularz KontrahLForm

2.Dwukrotnie kliknij przycisk EdytujRekordBitBtn

3.W utworzonej procedurze , poniżej słowa kluczowego Inherit, wpisz poniższy kod:

WyswietlKontrah;

Zwróć uwagę, aby wpisywany kod znalazł się pod słowem kluczowym Inherit. Zostało ono tutaj wstawione automatycznie, aby wykonać ewentualne operacje przypisane analogicznej procedurze zdarzenia na formularzu - rodzicu. Ostatecznie definiowana procedura powinna mieć postać:

procedure TKontrahLForm.EdytujRekrdBitBtnClick(Sender : TObject);

begin

inherited

WyswietlKontraj;

end;

4.Naciśnij klawisz F12, aby powrócić do widoku formularza i zaznacz komponent DBGrid1

5.W Object Inspectorze wybierz zakładkę Events i dwukrotnie kliknij w polu tekstowym obok zdarzenia OnDblClick

6.W utworzonej procedurze wpisz poniższy kod:

WyswietlKontrah;

Dzięki temu podwójne kliknięcie dowolnego wiersza siatki danych będzie powodowało wyświetlenie formularza z pojedynczym rekordem kontrahenta

7.Na końcu modułu formularza wpisz procedurę:

procedure TKontrahLForm.WyswietlKontrah;

begin

KontrahForm.ShowModal

end;

8.Uzupełnij deklarację procedury w sekcji interfejsu modułu (powyżej słowa kluczowego private), wpisując:

procedure WyswietlKontrah;

Wykonaj analogiczne czynności z punktów 1-8 dla formularza TowarLForm. Utwórz w nim procedurę WyswietlTowar, w której będzie wykonywana metoda TowarForm.ShowModal

9.Kliknij przycisk Save All ,aby zapisać zmiany

10.Przetestuj działanie aplikacji

Synchronizacja danych między formularzami

Czy wprowadziłeś przynajmniej kilka rekordów do tabeli Kontrah? Jeżeli nie, zrób to.Gdy będziesz w widoku listy, dwukrotnie kliknij któryś z dalszych rekordów (nie pierwszy). Jak się można było spodziewać, nastąpi przejście do formularza KontrahForm. Jednak czeka nas przykra niespodzianka. Zamiast rekordu który klikaliśmy, zobaczymy pierwszy rekord z tabeli Kontrah. Nic dziwnego - komponent TDataSource na tym formularzu korzysta z innego obiektu TTable niż siatka na formularzu KontarhLForm. Krótko mówiąc, operacje wykonywane w siatce (np. zmiana bieżącego rekordu) nie są synchronizowane z formularzem KontrahForm. Z tym problemem można sobie poradzić na kilka sposobów, mu jednak zdecydujemy się na najprostsze rozwiązanie. Aby uzyskać synchronizację danych między formularzami, wystarczy aby komponenty TDataSource, do których odwołują się kontrolki tych formularzy, korzystały z tych samych komponentów TTable

1.Przejdź do widoku formularza KontrahForm

2.Usuń komponent Table1

3.Zaznacz komponent DataSource1

4.Przejdź do zakładki Properties w oknie Object Inspectora i w polu właściwości DataSet wpisz KontrahDModule.Table1

5.Przejdź do widoku kodu modułu formularza, odszukaj procedurę obsługi zdarzenia FormCreate i usuń w niej wiersz

Table1.Open Procedura ta jest tworzona automatycznie przez kreator formularzy. Ponieważ obiekt Table1 w tym formularzu przestał istnieć, pozostawienie powyższej instrukcji spowodowałoby błąd kompilacji. Czynności z punktów 1-5 wykonaj w formularzu TowarForm, Jako właściwość DataSet wpisz TowarDModule.Table1

6.Kliknij przycisk Save All, aby zapisać wprowadzone zmiany

7.Skompiluj i uruchom aplikację. Przetestuj wprowadzone modyfikacje.

Synchronizacja między formularzami została osiągnięta.

Podsumowanie

W tej części utworzyłeś formularz ListaForm, który posłużył jako formularz macierzysty do zdefiniowania formularzy wyświetlających listy faktur, towarów i kontrahentów. Formularze te powiązałeś z menu głównym aplikacji, dzięki czemu są dostępne po jej uruchomieniu. Przy okazji nauczyłeś się umieszczać formularze w repozytorium i używać go z wykorzystaniem mechanizmu dziedziczenia. Dodatkowo utworzyłeś formularze do edycji pojedynczych rekordów tabel TowUslug i Kontrah,. Wszystkie zadania , które wykonałeś w tej i poprzednich częściach miały na celu przygotowanie aplikacji do wprowadzenia najważniejszego formularza, jakim jest formularz do wprowadzania faktur. Jego tworzenie zaczniemy w kolejnej części

FORMULARZE Z WIELOMA ŹRÓDŁAMI DANYCH

Większość aplikacji ma swój "punkt węzłowy", na którym koncentruje się znaczna część działań użytkownika i w którym są zgromadzone kluczowe elementy logiki aplikacji .Wszystkie pozostałe fragmenty kodu mają wobec wspomnianego punktu węzłowego charakter służebny - ich zadaniem jest zabezpieczenie integralności i poprawności danych oraz zapewnienie maksymalnej wygody

użytkownika. W naszej aplikacji kluczowym elementem jest formularz do sporządzania faktur. Stwierdzenie to nie wymaga chyba specjalnego uzasadnienia. Wszystkie listy towarów i kontrahentów tworzymy by móc wypełnić fakturę, a wydruki służą do graficznego zaprezentowania jednej lub wielu faktur. Formularz ten będzie najbardziej złożonym w opracowywanej aplikacji - połączymy w nim dane z kilku źródeł (kontrahenci, towary itp.) i jednocześnie poddamy wielorakiej weryfikacji. Oczywiście w bardziej złożonych aplikacjach (czy pakietach aplikacji) punktów węzłowych może być kilka, na przykład obok formularza do sporządzania faktur może się nim okazać formularz do przygotowywania listy wynagrodzeń

Tworzenie formularza do edycji dwóch tabel powiązanych relacją

Do przygotowania formularza faktur użyjemy znanego nam już kreatora formularzy. Jednak sposób postępowania będzie nieco inny niż w poprzednich przykładach. Do tej pory tworzyliśmy formularze wyświetlające zawartość jednej tabeli. Planowany formularz będzie wyświetlał zawartość dwóch tabel. Pierwszą z nich jest tabela TranNagl, w której jak pamiętamy, są gromadzone informacje z nagłówka faktury, takiej jak jej numer, data wystawienia, identyfikator kontrahenta itp. Druga tabela to TranSzcz, w której znajdują się poszczególne pozycje faktury wraz z wyszczególnieniem ceny i ilości. Pierwsza tabela będzie nadrzędna (master) w stosunku do drugiej (detail). Stąd te z tego typu formularze określa się jako master/detail. Aby tworzony formularz wyświetlał zawartość dwóch tabel, wykonaj następujące kroki:

1.Uruchom Delphi

2.W menu Database, wybierz polecenie Form Wizard

3.W pierwszym oknie kreatora zaznacz opcję Create a master/detail form i pozostaw zaznaczoną opcję Create form using TTable objects, a następnie kliknij Next. Pojawi się drugie okno kreatora, w którym należy podać nazwę tabeli głównej (nadrzędnej)

4.W polu Directories przejdź do folderu Dane, a następnie na liście Table Name zaznacz tabele TranNagl i kliknij Next

5.Za pomocą przycisku >> przenieś wszystkie pola tabeli TranNagl na listę Ordered Selected Fields i kliknij Next

6.Zaznacz opcję Horizontally i kliknij Next. W tworzonym formularzu w danym momencie będziemy wyświetlać tylko jedną fakturę, a więc wybór opcji In a grid nie miałby sensu. W kolejnych oknach kreatora będziemy odpowiadali na podobne pytania lecz dotyczące tabeli szczegółowej

7.Zaznacz tabelę TranSzcz i kliknij Next

8.Kliknij przycisk >> aby przenieść wszystkie pola tabeli TranSzcz na listę Ordered Selected Fields, a następnie kliknij Next

9.Upewnij się, że jest zaznaczona opcja In a grid i kliknij Next. Na ekranie pojawi się okno kreatora, z którym dotychczas nie mieliśmy do czynienia. Musimy w nim określić pola łączące tabelę główną z tabelą szczegółową. Jeśli dobrze zaprojektowaliśmy strukturę bazy danych ta operacja nie powinna nastręczyć trudności. Wyjaśnimy w kilku słowach budowę tego okna. Po pierwsze w polu Available Index widnieje tekst Primary Oznacza to ,że do złączenia dwóch źródeł bazy danych posłuży indeks pdstawowy tabeli TranNagl. W skład tego indeksu wchodzi jedno pole, IDTranNagl. Na liście Detail Fields są wyświetlane wszystkie elementy indeksu podstawowego tabeli TranSzcz, a więc pola IDTranNagl i IDTranSzcz. Przycisk Add służy do określenia pól złączonych (joined fields), to znaczy takich, których wartości w obu polach powinny być równe

10.Na liście Detail Fields zaznacz pole IDTranNagl

11.Pole o takiej samej nazwie zaznacz na liście Master Fields, a następnie kliknij Add .W polu Joined Fields pojawiło się wyrażenie : IDTranNagl → IDTranNagl. Zapsi ten oznacza ,że w formularzu pojawią się tylko te rekordy z tabeli TranSzcz których pole IDTranNagl ma taką samą wartość ,jak pole IDTranNagl w bieżącym rekordzie tabeli TranNagl. Jeśli do połączenie używamy indeksu złożonego (składającego się z więcej niż jednego pla), par pól na liście Joined Fields może być więcej

12.Kliknij przycisk Next

13.Upewnij się, że w ostatnim oknie kreatora zaznaczona jest opcja Form Only oraz ,że pole wyboru Genrate main form jest zaznaczone, a następnie kliknij przycisk Finish. Formularz został utworzony.

14.Przejdź do Object Inspectora i zmień właściwość Name formularza na FakturaForm

15.Przejdź do właściwości Caption i wpisz Faktura

16.Zaznacz komponent Table2 i zmień jego właściwości Database na BAZAFAKT. Komponentu Table1 nie modyfikujemy ponieważ za chwilę zostanie on usunięty (w celu uzyskania synchronizacji formularzy)

17.Kliknij przycisk Save i zapisz moduł nowego formularza pod nazwą Faktura

Aby nasz formularz stał się dostępny z poziomu formularza z listą faktur, musimy jeszcze zdefiniować odpowiednie powiązania. Ponieważ czynności te wykonywaliśmy już trzykrotnie, nie powinieneś mieć problemów z ich powtórzeniem. A tej chwili musisz utworzyć odpowiednie procedury obsługi zdarzeń w module FaktL.pas, wywołujące formularz FakturaForm. Informacje znajdziesz w sekcji "Powiązanie formularza do edycji pojedynczego rekordu z formularzem-listą". Aby uzyskać synchronizację danych wyświetlanych na liście faktur i w formularzu jednej faktury, musisz usunąć z tego ostatniego komponent Table1, a następnie właściwości DataSource1.DataSet przypisać komponent typu TTable umieszczony w module danych FaktDModule (nie zapomnij wstawić klauzuli uses FaktD; w sekcji implementation modułu Faktura.pas)

Określanie zależności między tabelą nadrzędną a podrzędną

Przeanalizujemy teraz budowę formularza FakturaForm. Interesuje nas zwłaszcza powiązanie miedzy wyświetlanymi na nim dwiema tabelami. Gdybyśmy teraz wprowadzili kilka faktur, zauważylibyśmy następujący fakt - przy przejściu do innej faktury (zmianie bieżącego rekordu w tabeli TranNagl), w siatce pojawiają się tylko pozycje faktury (rekordy z tabeli TranSzcz), które dotycza tylko tej właśnie faktury. By lepiej zrozumieć ten mechanizm, musimy na chwilę cofnąć się do struktury bazy danych. Pamiętamy, że między tabelami TranNagl i TranSzcz zachodzi relacja jeden-do-wielu. Relację tę definiują pola IDTranNagl zdefiniowane w obydwu tabelach. W przypadku tabeli TranSzcz pole IDTranNagl jest jednym z dwóch składników klucza podstawowego (należy do niego również pole Pozycja). Gdy na utworzonym przed chwilą formularzu wyświetlany jest jakiś rekord z tabeli TranNagl, np. o numerze 5, wówczas na obiekcie Table2, reprezentującym tabelę TranSzcz, zakładany jest filtr, który udostępnia rekordy z tej tabeli o tym samym identyfikatorze, zgodnym z identyfikatorem tabeli nadrzędnej. Zwróćmy uwagę ,że mechanizm ten wykorzystuje powiązanie zdefiniowane w trakcie działania kreatora , a nie poziomie struktury bazy danych. Powiązania tego typu definiujemy często bez pomocy kreatora, warto więc dowiedzieć się w jaki sposób są one zapisane na formularzu:

1.Przejdź do widoku formularza FakturaForm. Na formularzu widoczne są dwa komponenty typu TDataSource i jedne komponent TTable

2.Zaznacz komponent Table2 , a następnie w Object Inspectorze wyświetl kartę Properties. W arkuszu właściwości zwróć uwagę na dwa pola : MasterSource i MasterFields. Pierwsze z nich wskazuje źródło danych, z którym powiązany jest ten obiekt TTable. Jak widać jest nim obiekt DataSource1, powiązany z tabelą TranNagl (poprzez komponent FaktDModule.Table1). Z kolei w polu MasterFields znajdują się nazwy pól z tabeli nadrzędnej, wchodzące w relacje z kluczem podstawowym tabeli podrzędnej - u nsa jest to pole IDTranNagl

Wiedzę tę wykorzystamy do zdefiniowana kolejnego typu zależności na formularzu faktury

Stosowanie pól kombi i pól memo

Niektóre informacje wprowadzane przez kreator są niepotrzebne, część kontrolek trzeba nieco zmodyfikować, zmienić ich położenie oraz inne właściwości itp. Tu skoncentrujemy się na takich "edytorskich" operacjach. Pokażemy jak zastąpić standardowe kontrolki edycyjne zaproponowane przez kreator bardziej zaawansowanymi narzędziami. Po pierwsze założymy ,że liczba możliwych form płatności jest ograniczona do co najwyżej kilku. Jeżeli chcemy zawęzić zbiór wartości, którymi powinien dysponować użytkownik, przy czym zbiór ma charakter stały, dobrym rozwiązaniem jest zastosowanie pola listy lub pola kombi. Ponieważ nasz formularz będzie wyświetlał wiele różnych informacji, a jego powierzchni jest ograniczona zdecydujemy się na tę drugą kontrolkę

1.Zaznacz kontrolkę EditForma_platn, a następnie usuń ją naciskając klawisz Del

2.Na palecie Data Controls zaznacz kontrolę TDBComboBoxm a następnie umieść ja w miejscu z którego zostało usunięte pole tekstowe. W razie potrzeby przesuń nieco pozostałe kontrolki, aby uzyskać miejsce dla wstawionego obiektu

Uwaga : Zanim wstawisz nowa kontrolkę , zaznacz komponent typu TScrollBox, na który zostały umieszczone pozostałe komponenty związane z nagłówkiem faktury. TScrollBox jest tak zwanym komponentem posiadającym, co oznacza ,że może być właścicielem innych komponentów (ich właściwości Owner można przypisać komponentowi TScrollBx). Jeżeli nie postąpisz zgodnie z tym zaleceniem, wstawiana kontrolka może zostać przejęta przez inny komponent posiadający ,co utrudni późniejszą pracę z formularzem - w szczególności nie będzie można prawidłowo ustawić kolejności poruszania się za pomocą klawisza Tab. Po prawidłowym wstawieniu kontrolka powinna być widoczna w okienku Object TreeView jako odgałęzienie obiektu TScrollBox. Komponentem posiadającym oprócz komponentu typu TScrollBox jest m.in. formularz, kontrolki typu TPanel, TradioGroup lub TGroupBox itp. Hierarchię posiadania najlepiej obserwować w okienku Object TreeView, umieszczonym nad okienkiem Object Inspectora. Jak widać, kontrolki danych związane z tabelą TranNagl są w posiadaniu komponentu TScrollBox, który z kolei jest zarządzany przez komponent Panel2, będący własnością formularza FakturaForm

Na formularzu pojawi się nowa kontrolka o nazwie DBComboBox. Kontrolka ta nie jest jeszcze powiązana ze źródłem danych

3.W Object Inspectorze odszukaj właściwość Name i wpisz Forma_platnDBCombo

4.Przejdź do właściwości DataSource i z listy wybierz źródło danych DataSource1 .W tym momencie nowa kontrolka została powiązana ze źródłem danych DataSource1.

5.Przejdź do właściwości DataField i z listy wybierz pole Forma_platn .W kolejnym kroku musimy zdefiniować możliwe formy płaności

6.Przejdź do właściwości Items i kliknij przycisk wielokropka. Ponieważ elementy pola Items są typu TStrings, na ekranie pojawi się okno edytora łańcuchów znaków

7.W oknie dialogowym String List Editor wpisz jedno pod drugim następujące słowa:

gotówka

przelew

czek

8.Kliknij przycisk OK

W tej chwili użytkownik będzie mógł zarówno wybierać z listy jaki i wpisywać swoje własne propozycje z klawiatury. Jeżeli chcesz aby w danym polu użytkownik mógł wybierać tylko wartości proponowane na liście, ustaw właściwość Style pola kombi na csDropDownList

Kolejny problem z jakim musimy sobie poradzić to pole Uwagi. Kreator utworzył dla niego kontrolkę typu TDBEdit. Jednak jak pamiętamy pole to zostało zdefiniowane w bazie danych jako pole typu Memo. Gdybyś teraz skompilował aplikację i spróbował coś wpisać w polu EditUwagi na formularzu FakturaForm, nie będzie to możliwe. Musimy więc zastosować kontrolkę odpowiedniego typu

1.Zaznacz pole EditUwagi i usuń je, naciskając klawisz Del

2.Na palecie DataControls zaznacz kontrolkę typu TDBMemo, a następnie umieść ją na formularzu. Na formularzu pojawi się nowa kontrolka DBMemo1

3.Dopasuj rozmiary wstawionej kontrolki aby była w całości widoczna i nie przesłaniała innych obiektów

4.w Object Inspectorze odszukaj właściwość DataSource i z listy wybierz źródło danych DataSource1

5.Przejdź do właściwości DataField i z listy wybierz pole Uwagi. Kontrolka zostanie powiązana z polem Uwagi w tabeli TranNagl

6.Przejdź do właściwości Name i wpisz UawgiDBMemo

Zmiana kolejności klawisza Tab

Do kolejnych kontrolek ekranu, których właściwość TabStop jest ustawiona na True, można przechodzić za pomocą klawisza Tab .W aplikacjach bazodanowych jest to bardzo dużym ułatwieniem, gdyż użytkownik w trakcie wprowadzania danych z klawiatury nie musi co chwila sięgać po myszkę. Podczas tworzenia formularza kreator domyślnie tak ustawia kolejność klawisza Tab, aby jego naciśnięcie powodowało przechodzenie do sąsiadujących kontrolek. Często jednak zdarza się tak ,że dokonujemy modyfikacji formularza, które polegają na wstawianiu nowych kontrolek edycyjnych lub zmianie uporządkowania już istniejących. Nowe kontrolki są zawsze umieszczane na końcu listy klawisza Tab. W takiej sytuacji programista sam musi zadbać ,aby poszczególne kontrolki były dostępne we właściwej kolejności. W ty celu trzeba kliknąć prawym klawiszem myszki w dowolnym miejscu komponentu posiadającego, a następnie wybrać polecenie Tab Order i w oknie dialogowym ustawić kontrolki we właściwej kolejności. Pracując nas swoimi aplikacjami ,zwróć szczególną uwagę na ten aspekt. Nic tak nie deprymuje użytkownika jak kursor przemieszczający się po ekranie w nieprzewidziany sposób

Ukrywanie wybranych pól

Chociaż kreator bardzo się starał, wygląd formularza wciąż jest daleki od naszych oczekiwań Przede wszystkim, dwa pola z tabeli TranNagl (IDTranNagl i Typ) są używane wewnętrznie przez aplikację, a więc użytkownik nie musi ich widzieć

1.Na formularzu FakturaForm zaznacz kontrolki EditIDTranNagl, EditTyp orz towarzyszące im etykiety

2.Naciśnij klawisz Delete

Zaznaczone kontrolki zostaną usunięte z ekranu i z definicji klasy. Nie oznacza to jednak ,że tracimy do nich dostęp - w dalszym ciągu możemy na ich operować programowo przez komponent TTable1, zdefiniowany w module danych FaktModule

Zmiana tekstu etykiety na formularzu

Kreator tworząc formularz wygenerował również etykiety dla każdej kontrolki TDBEdit. Ich właściwości Caption zostały zdefiniowane na podstawie nazwy pól obsługiwanych przez odpowiednie kontrolki TDBEdit. Ponieważ uzyskane nazwy nie zawsze są wystarczając czytelne, warto zmodyfikować nieco treść etykiet

1.Zaznacz etykietę z tekstem Data_wystaw, przejdź do Object Inspectora i zmień jej właściwość Caption na Data wystawienia

2.W analogiczny sposób zmień właściwości Caption pozostałych etykiet:

Data_trans : Data sprzedaży

Term_platn : Termin płatności

Forma_platn : Forma płatności

IDKontrah : Kontrahent

Rozmieszczanie kontrolek na formularzu

Czynności, które zamierzamy teraz wykonać, choć bardzo ważne z punktu widzenia ergonomiczności i estetyki interfejsu użytkownika, mają charakter manualny. Z tego względu nie będziemy się tym zajmować szczegółowo.

1.Rozmieść kontrolki związane z nagłówkiem faktury. W razie potrzeby zwiększ nieco rozmiary komponentu Panel2 (w pionie) i samego formularza (szerokość)

2.Gdy zakończysz modyfikowanie formularza zapisz go

Modyfikacja właściwości pól wyświetlanych w komponencie TDBGrid

Analogiczne czynności wykonamy teraz na nagłówkach siatki danych z szczegółami faktury. W ich przypadku musimy zastosować takie same zasady postępowania , jak podczas zmiany nagłówków kolumn w formularzach list

1.Dwukrotnie kliknij komponent Table2, aby wyświetlić listę jego pól

2.Na liście zaznacza pole IDTranNagl

3.W Object Inspectorze odzsukać właściwość Visible, a następnie ustaw ją na False

4.Zmdyfikuj inne właściwości pól obiektu Table2. Skorzystaj z tego:

Pole : IDTranSzcz

Właściwość : Visible

Nowa wartość : False

Pole : IDTowUslug

Właściwość : DisplayLabel

Nowa wartość: Kod

Pole : Nazwa

Właściwość : DisplayWidth

Nowa wartość : 40

Pole : KWiU

Właściwość : DisplayWidth

Nowa wartość 12

Pole : J_miary

Właściwość : DisplayLabel

Nowa wartość: j.m

Pole : J_miary

Właściwość : DisplayWidth

Nowa wartość : 4

Pole : Ilosc

Właściwość : DisplayLabel

Nowa wartość Ilość

Pole : Cena_jedn

Właściwość : DisplayLabel

Nowa wartość : Cena jedn.

Pole : St_VAT

Właściwość : DisplayLabel

Nowa wartość : St. VAT

Podobnie jak w przypadku nagłówka faktury, również pole IDTranSzcz z tabeli TranSzcz nie powinno być wyświetlane w siatce danych - służy ono wyłącznie do zdefiniowania relacji z tabelą TranNagl

6.Kliknij przycisk Save All, aby zapisać wprowadzone zmiany

7.Skompiluj i uruchom aplikację

8.Przejdź do formularza FakturaForm i spróbuj wpisać fikcyjną fakturę (nie podawaj kodu kontrahenta). Wypełnij pozycje faktury

Defniowanie podrzędnego źródła danych

W obecnej postaci na formularzu wyświetlany jest tylko identyfikator kontrahenta. Takie rozwiązanie jest nie do przyjęcia - nie możemy zmuszać użytkownika aby pamiętał identyfikatory wszystkich klientów. Wynika z tego ,że na ekranie powinny się również pojawić takie podstawowe informacje o kliencie, jak nazwa oraz adres. Aby uzyskać dane ,zdefiniujemy kolejne podrzędne źródło danych

1.Na palecie BDE zaznacz komponent TTable i umieść go na formularzu obok komponentu DataSource2. Na formularzu pojawi się nowy komponent TTable1. Otrzymał taką nazwę ponieważ usunęliśmy poprzedni komponent TTable1 wygenerowany przez kreator. Ponieważ źródło danych , które za chwilę wstawimy otrzyma nazwę DataSource3,zmienimy teraz nazwę komponentu Table1 na Table3 co ułatwi nam kojarzenie tych dwóch obiektów

2.Gdy zaznaczony jest nowy komponent TTable przejdź do Object Inspectora i zmień właściwość Name na Table3

3.Zmodyfikuj dalsze właściwości komponentu Table3 zgodnie z tym:

DatabaseName : BAZAFAKT

TableName : Kontrah

MasterSource : FaktDModule.DataSource1

ReadOnly : True

Pierwsze dwie właściwości .DatabaseName i TableName, służą do wskazania tabeli w bazie danych, z którą zostanie powiązany komponent Table3. Za pomocą właściwości MasterSource definiujemy nadrzędne źródło danych, którym w tym przypadku jest tabelaTranNagl (wskazywana przez źródło danych FaktDModule.DataSource1). Dane kontrahenta wyświetlane na tym formularzu nie powinny być dostępne do edycji. W takiej sytuacji warto ustawić właściwość ReadOnly na True. Jeśli zestaw danych jest tylko do odczytu, zaleca się ustawienie tej właściwości na True, gdyż zwiększa to wydajność aplikacji i oszczędza zasoby systemowe.

4.Przejdź do właściwości MasterField i kliknij przycisk wielokropka. Pojawi się okno dialogowe Field Link Desginer

5.Na liście Detail Fields zaznacz pole IDKontrah (z tabeli Kontrah), natomiast na liście Master Fields odszukaj pole o takiej samej nazwie (lecz należące do tabeli TranNagl)

6.Kliknij przycisk Add,, a następni OK. Pole łączące budiwe tabele zostało zdefiniowane. Utworzenie takiej zależności jest możliwe, ponieważ pole IDKontrah jest kluczem obcym w tabeli TranNagl i kluczem podstawowym w tabeli Kontrah. Dzięki temu w zestawie danych Table3 będzie dostępny tyko ten rekord z tabeli Kontrah, którego wartość IDKontrah jest równa wartości z pola IDKontrah, ale w tabeli TranNagl

7.Z palety Data Access pobierz komponent TDataSource i umieść obok komponentu Table3. Pojawi się nowy komponent DataSource3

8.Przejdź do Object Inspectora i ustaw właściwość DataSet komponentu DataSource3 na Table3. Możemy przystąpić do wstawiania komórek danych w których będą wykorzystywane informacje o kontrahencie. Do ich wyświetlenia wykorzystamy pola typu TDBEdit

9.W pobliżu pola EditIDKontrah umieść cztery pola TDBEdit oraz jedno pole TLabel, i rozmieść je na formularzu

10.Zaznacz wszystkie nowe pola TDBEdit (skorzystaj z klawisza Shift) i zmdyfikuj ich wspólne właściwości :

DataSource : DataSource3

Enabled : False

Color : clSkyBlue

Zmiana właściwości Enabled na False powoduje ,że w polu tym nie będzie można ustawić kursora. Zmiana właściwości Color oznacza ,że tło pola będzie wyświetlane w innym kolorze - to również jeden ze sposobów poinformowania użytkownika, że wartości w tym polu nie podlegają edycji

11.Poszczególnym komponentom TDBEdit przypisz właściwości Name i DataField :

TDBEdit1 : Name - EditNazwa :DataField - Nazwa

TDBEdit2 : Name - EditMiasto :DataField - Miasto

TDBEdit3 : Name - EditUlica :DataField - Ulica

TDBEdit1 : Name - EditNIP :DataField - NIP

12.Zmień właściwość Caption nowej etykiety na NIP

Operacje związane z otwieraniem wstawionej tabeli

Musimy jeszcze zadbać o to, aby komponent Table2 otwierał się w czasie tworzenia formularza

1.Naciśnij F12 aby przejść do widoku kodu, odszukaj procedurę zdarzenia FormCreate i uzupełnij ją o wiersz kodu:

Table3.Open

2.Kliknij przycisk Save, aby zapisać wprowadzone zmiany

3.Przejdź do formularza FakturaForm, w polu Kontrahent wpisz identyfikator NOWAK i naciśnij klawisz TAB aby przejść do następnej kontrolki. Jeżeli podany identyfikator istnieje w tabeli Kontrah, w nowych kontrolkach TDBEdit pojawią się dane kontrahenta

4.Korzystając z paska nawigatora, przejdź do innego rekordu tabeli TranNagl

5.Powróć do poprzedniego rekordu

Jak widać, dzięki powiązaniu zdefiniowanemu w komponencie Table3 (właściwość MasterSource i MasterFields), wartości w kontrolkach związanych z danymi kontrahenta są natychmiast odświeżane. Oczywiście nie możemy oczekiwać, że użytkownik będzie pamiętał identyfikatory wszystkich kontrahentów lub przed wystawieniem każdego dokumentu sprzedaży wychodził z formularza FakturaForm, wracał do menu i wyświetlał sobie listę kontrahentów, a następnie wracał znowu do wystawienia faktur. Wniosek jest oczywisty - na formularzu FakturaForm powinna być możliwość podglądu zdefiniowanych kodów kontrahentów

Stosowanie pól TDBLookupCombo do wyświetlania listy dopuszczalnych wartości

Zwróć uwagę ,że jedyną informacją o kontrahencie przechowywaną w tabeli TranNagl jest jego identyfikator zaś pozostałe jego dane są uzyskiwane poprzez relację z tabelą Kontrah, której kluczem jest właśnie pole IDKontrah. Z tego spostrzeżenia wynika fakt ,że nie będzie można wystawić faktury

dla kontrahenta, którego danych nie wprowadzono do tabeli Kontrah. Spróbujmy wykorzystać te wnioski do zmodyfikowania formularza FakturaForm. Jeżeli wiemy ,że w jakimś polu może się pojawić tylko wartość, dal której istnieje odpowiedni zapis w tabeli podrzędnej (tabela Kontrah), najwygodniej jest użyć komponentu TDBLookupCombo. W polu tego typu są wyświetlane wartości z tabeli podrzędnej, nazywanej również tabelą odnośników (lookup table)

1.Usuń z formularza komponent EditIDKontrah

2.Zaznacz komponent TDBLookupCombo na palecie Data Controls i umieść go w tym samym miejscu , w którym znajdowało się usunięte pole tekstowe. Musimy jeszcze zdefiniować źródło danych dla nowej kontrolki. Potrzebne są do tego komponenty TTable i TDataSource<nr>

3.Z palety BDE pobierz komponent TTable i umieść na formularzu. Zmień jego właściwości :

Name : Table4 DatabaseName: BAZAFAKT TableName : Kontrah ReadOnly : True Active : True

Chociaż na formularzu jest już jeden zestaw danych odwołujący się do tabeli Kontrah, nie możemy z niego skorzystać ponieważ został powiązany relacją master / detail z tabelą TranNagl - po rozwinięciu listy w polu kombi pojawiłby się tylko identyfikator kontrahenta zgodny z wartością przechowywaną w polu IDKontrah w tabeli TranNagl

4.Obok komponentu Table4 umieść komponent TDataSource z palety Data Access. Na formularzu pojawi się komponent DataSource4

5.Przejdź do Object Insepctora i ustaw właściwość DataSet nowego komponentu na Table4. Po zdefiniowaniu źródła danych musimy jeszcze określić sposób działania pola kombi.

6.Zaznacz komponent DBLookupComboBox1 i zmień jego właściwości:

DataSource : DataSource1 DataField : IDKontrah ListSource : DataSource4 KeyField : IDKontrah ListField : IDKontrah; Nazwa DropDownWidth : 200 NullValueKey : Del Name : IDKontrahDBLookup

Właściwości DataSource i DataField , podobnie jak w innych kontrolkach danych, wskazują ple tabeli, w którym będzie przechowywana wartość wybrana w tym polu kombi. Z kolei właściwości ListSource i KeyField wskzują źródło danych i pole, którego wartość powinna być zgodna z wartością przechowywaną w polu wskazanym we właściwości DataField. We właściwości ListField podajemy pole lub pola, które będą wyświetlane po rozwinięciu listy. W naszym przypadku chcemy aby obok identyfikatora kontrahenta wyświetlała się również jego nazwa. Ponieważ wyświetlane informacje zajmą dużo miejsca, musimy odpowiednio zmodyfikować wartość DropDownWidth (domyślna wartość 0 oznacza ,że rozwinięta lista ma taką samą szerokość jak pole kombi). I wreszcie NullValueKey. Pole kombi ma tą cechę, że podczas wpisywania w nim kolejnych znaków, automatycznie proponowane są najbardziej pasujące wartości klucza. Może się jednak zdarzyć ,że po wypełnieniu pola IDKontarhenta z jakichś względów będziemy chcieli usunąć wyświetlaną po nim wartość. Nie jest to jednak zwykłe pole tekstowe, a wiec tradycyjne metody nie zadziałają. Musimy wtedy skorzystać z właściwości NullValueKey. Przypisujemy jej nazwę klawisza (lub kombinacji klawiszy), który będzie odpowiedzialny za czyszczenie tego pola W naszym przykładzie wybraliśmy Del

7.Przejdź do widoku kodu i uzupełnij procedurę zdarzenia FormCreate o poniższy wiersz:

Table4.Open;

8.Kliknij przycisk Save All , aby zapisać wprowadzone zmiany

9.Skompiluj i uruchom aplikację.

10.Przejdź do formularza Faktura i rozwiń listę w polu kombi

11.Z listy kontrahentów wybierz jeden z kodów. Po wybraniu kodu powinny się zaktualizować pozostałe pola dotyczące kontrahent na formularzu Faktura

Podsumowanie

W tej części poznałeś zasady tworzenia formularzy korzystających z kliku powiązanych ze sobą zestawów danych. Formularz FakturaForm korzysta obecnie z jednego nadrzędnego zestawu danych (tabela TranNagl) oraz trzech zestawów podrzędnych (jeden dla tabeli TranSzcz i dwóch dla tabeli Kontrah). Dzięki temu, w trakcie edycji pojedynczej faktury będą wyświetlane wszystkie rekordy z tabeli TranSzcz o takim samym identyfikatorze transakcji oraz dane kontrahenta o identyfikatorze zgodnym z identyfikatorem zapamiętanym w nagłówku transakcji. Dowiedziałeś się również ,jak używać pola TDBLookupCombo do wyświetlania listy dopuszczalnych wartości utworzonej tabeli odnośników.

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

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