

Wykłady z informatyki

Janusz Górczyński, Paweł Górczyński

Projektowanie baz danych w MS Access, wykorzystanie VBA

*Wyższa Szkoła Zarządzania i Marketingu
Sochaczew 2005*

Zeszyt ten jest drugą pozycją w serii materiałów dydaktycznych
Wykłady z informatyki.

Dotychczas ukazała się pozycja „*Makropolecenia i aplikacje VBA w MS Excel*” w 2003 roku.

Wydanie I

Materiały do druku zostały w całości przygotowane przez Autorów
ISBN 83-88781-36-7

Wydawca: Wyższa Szkoła Zarządzania i Marketingu w Sochaczewie

Projekt okładki: Poligraphica, 91-231 Łódź, ul. Ratajska 1.

<http://www.CentrumPoligrafii.pl>

Druk cyfrowy: SOWA, Sp. z o.o., ul. Hrubieszowska 6a, 01-209 Warszawa,
<http://sowadruck.pl>

Arkuszy wydawniczych 20,0
Arkuszy drukarskich 20,0

Spis treści

OD AUTORÓW	6
1. WSTĘP.....	7
2. RELACYJNE BAZY DANYCH.....	7
2.1. BAZY KARTOTEKOWE.....	8
2.2. BAZY RELACYJNE	9
2.3. PROJEKTOWANIE RELACYJNYCH BAZ DANYCH.....	11
2.3.1. <i>Normalizacja</i>	11
2.3.2. <i>Relacje, więzy integralności</i>	17
2.3.4. <i>Kwerendy</i>	20
Kwerendy wybierające.....	20
Kwerendy krzyżowe	29
Kwerenda aktualizująca.....	30
Kwerenda usuwająca	33
Kwerenda dołączająca	34
2.3.5. <i>Formularze</i>	38
Formularze proste	38
Formularze złożone	46
Obliczenia w formularzach.....	54
2.3.6. <i>Raporty</i>	67
Raporty proste.....	67
Raporty złożone	72
Obliczenia w raportach	79
3. JĘZYK SQL	81
3.1. POJĘCIA PODSTAWOWE	81
3.2. POLECENIE SELECT	82
Klauzula Join	83
Klauzula As.....	85
Klauzula Where	86
Klauzula Order by.....	88
3.2. POLECENIE INSERT INTO	89
3.3. POLECENIE UPDATE	90
3.4. POLECENIE DELETE.....	90

4. VBA W MS ACCESS	91
4.1. POJĘCIA PODSTAWOWE	91
4.1.1. <i>Zmienne i stale</i>	92
4.1.2. <i>Procedury i funkcje</i>	96
4.1.3. <i>Klasy</i>	98
4.1.4. <i>Pętle</i>	104
4.1.5. <i>Struktury warunkowe</i>	107
4.2. PROGRAMOWANIE BAZ DANYCH, METODA ADO.....	108
4.2.1. <i>Obiekt Connection</i>	108
4.2.2. <i>Obiekt Recordset</i>	111
4.2.3 <i>Praca z rekordsetem</i>	113
5. PRZYKŁADY APLIKACJI – OBSŁUGA BIBLIOTEKI	121
5.1. FUNKCJONALNOŚĆ	121
5.2. TABELE I RELACJE	121
5.2.1. <i>Dane klientów</i>	121
5.2.2. <i>Dane zakupionych książek</i>	132
5.2.3. <i>Rejestracja wypożyczeń</i>	137
5.3. FORMULARZE I KWERENDY.....	139
5.3.1. <i>Kilka prostych formularzy</i>	139
5.3.2. <i>Formularze z kontrolką typu ComboBox</i>	142
5.3.3. <i>Formularze złożone</i>	150
5.4. MAKROPOLECENIA I PROCEDURY VBA	160
5.4.1. <i>Modyfikacja frmZakupy</i>	160
5.4.2. <i>Modyfikacja formularzy frmWydawnictwa i frmKategorie</i>	165
5.4.3. <i>Modyfikacja formularza frmAutorzy</i>	170
5.4.4. <i>Modyfikacja formularza frmKlienci</i>	173
5.5. OBSŁUGA WYPOŻYCZEŃ I ZWROTÓW KSIĄZEK	180
5.5.1. <i>Formularz frmWypozyczenia</i>	180
5.5.2. <i>Formularz frmRejestracjaZwrotu</i>	192
5.6. WYSZUKIWANIE INFORMACJI	199
5.6.1. <i>Według tytułów pozycji</i>	199
5.7. RAPORTY	208
5.7.1. <i>Raport zbiorczy zakupów</i>	208
5.7.2. <i>Dynamiczny raport zakupów</i>	212
5.8. WŁASNY PASEK NARZĘDZIOWY	215

6. PRZYKŁADY APLIKACJI – OBSŁUGA SKLEPU.....	219
6.1. FUNKCJONALNOŚĆ	219
6.2. REJESTRACJA Klientów	219
6.2.1. Formularze obsługujące klientów	220
6.3. OBSŁUGA ZAOPATRZENIA	221
6.3.1. Formularze obsługujące składanie zamówień	223
6.3.2. Przygotowanie wydruku zamówienia	236
6.3.3. Dodatkowe usprawnienia formularza frmZamowienie	241
6.3.4 Kontrola realizacji zamówień	257
6.4. OBSŁUGA SPRZEDAŻY	274
6.4.1 Formularze sprzedaży.....	275
6.4.2 Faktura sprzedaży.....	287
6.4.3 Kredyt kupiecki	291
6.4.4 Spłata kredytu	297
6.5. RAPORTY	301
6.5.1 Dzienny raport kasowy.....	301
6.5.2 Rozliczanie podatku VAT.....	307
6.5.3 Analiza wielkości obrotów wg różnych kryteriów	314
6.6. MENU UŻYTKOWNIKA	334
6.6.1 Projektowanie paska menu.....	334
6.6.2 Funkcje obsługujące polecenia menu	338
6.6.3 Modyfikacja ustawień startowych MS Access.....	344
7. BAZA DANYCH W SIECI LAN.....	347
7.1. PODZIAŁ BAZY DANYCH NA DANE I INTERFEJS	347
7.2. ZABEZPIECZENIE BAZY DANYCH	350
7.2.1. Kreator zabezpieczeń.....	351
7.2.2. Modyfikacja pliku grupy roboczej.....	358
7.2.3. Dodatkowe procedury	362
7.2.4. Utworzenie pliku MDA	366
7.2.5. Konfiguracja stacji roboczej klienta	368
8. LITERATURA.....	369
9. INDEKS	370

Od Autorów

Seria **Wykłady z Informatyki** została pomyślana jako miejsce publikowania bardziej zaawansowanych pozycji poświęconych informatyce stosowanej realizowanej na dwóch ostatnich semestrach specjalności *informatyka w zarządzaniu*. W 2003 roku ukazała się pozycja *Aplikacje i makropolecenia VBA w MS Excel*, obecna jest w pewnym sensie jej kontynuacją, ale w odniesieniu do relacyjnych baz danych i programu MS Access.

Pierwsze dwa rozdziały poświęcone są przypomnieniu i uporządkowaniu informacji o podstawach pracy z relacyjnymi bazami danych, ich projektowaniu w MS Access.

Rozdział trzeci poświęcony jest omówieniu języka SQL na takim poziomie, który jest niezbędny do sprawnego posługiwania się procedurami VBA w celu programowego dostępu i manipulowania danymi zapisanymi w tabelach bazy danych.

W czwartym rozdziale przedstawiliśmy podstawowe zasady programowania w języku Visual Basic for Applications (VBA), ze szczególnym zwróceniem uwagi na metodę ADO dostępu do danych.

W piątym i szóstym rozdziale zostały dość szczegółowo przedstawione dwie przykładowe aplikacje; od omówienia ich funkcjonalności, poprzez projekty tabel, kwerend, formularzy i raportów, wszystko to wsparcie procedurami VBA.

Ostatni, siódmy rozdział prezentowanej pozycji został poświęcony przygotowaniu bazy danych do pracy w lokalnej sieci komputerowej, w tym problemom związanym ze współużytkowaniem bazy danych i jej bezpieczeństwem.

Naszym zamiarem było przygotowanie takiej pozycji poświęconej relacyjnym bazom danych, która może być przydatna nie tylko dla studentów specjalizacji *informatyka w zarządzaniu*, ale także dla tych studentów innych specjalności, którzy pragną pogłębić swoją znajomość relacyjnych baz danych, przynajmniej na poziomie MS Access.

W książce tej przyjęliśmy zasadę, że wszystkie polecenia menu, okien dialogowych i nazwy właściwości obiektów będą wypisywane czcionką *pochyloną*.

Kody procedur i funkcji VBA oraz wartości właściwości, nazwy pól w tabelach będą wypisywane czcionką *Courier New*.

Zgodnie z zasadami zapisu instrukcji języka VBA wszelkie komentarze rozpoczęte są symbolem apostrofu, a dłuża instrukcja może być zapisana w kilku wierszach poprzez użycie tzw. symbolu kontynuacji linii, czyli kreski podkreślenia poprzedzonej spacją.

Janusz Górczyński

Paweł Górczyński

1. Wstęp

Jednym z istotnych składników pakietu Office firmy Microsoft jest MS Access, program typu SZRBD – system zarządzania relacyjną bazą danych. Inaczej mówiąc aplikacja ta przeznaczona jest do projektowania i zarządzania uporządkowanymi strukturami danych o dość znaczących możliwościach. Przy stosunkowo niewielkim koszcie aplikacji użytkownik otrzymuje efektywne i stosunkowo łatwe w użyciu narzędzie do tworzenia nawet dość dużych i skomplikowanych baz danych.

MS Access pozwala na tworzenie baz danych o wielkości do 2 gigabajtów, takiej też wielkości może być pojedyncza tabela. Baza danych stworzona w programie MS Access może być wykorzystywana przez pojedynczego użytkownika, tak z reguły będzie w małej firmie czy w przypadku bazy budowanej na użytek prywatny, ale może być także wykorzystywana przez wielu użytkowników jednocześnie.

MS Access jest, w pewnym sensie, kontynuatorem takich aplikacji SZRBD jak słynny dBBase, z tym, że po przeniesieniu do środowiska Windows stał się dostępny dla znacznie szerszego grona użytkowników. Włączenie do MS Access języka programowania VBA powoduje, że można w tym środowisku tworzyć naprawdę eleganckie i funkcjonalne aplikacje bazodanowe. Mamy nadzieję, że uda nam się przekonać naszych Czytelników do możliwości MS Access. Mamy zamiar poprowadzić kurs projektowania relacyjnych baz danych w oparciu o dwie przykładowe aplikacje. Mamy także zamiar nie wychodzić w tym podręczniku poza „standardowe” możliwości MS Access, bez nadmiernego wchodzenia w język VBA, choć w pewnych sytuacjach będzie to konieczne.

2. Relacyjne bazy danych

Pod pojęciem bazy danych z reguły rozumie się uporządkowaną strukturę danych, przy czym uporządkowanie jest takie, aby można było stosunkowo łatwo manipulować zgromadzonymi danymi.

Podstawowym jednostką każdej bazy danych jest **tabela**, wiersze tabeli opisują elementarną informację przechowywaną w danej tabeli. Przykładowo wiersze tabeli **Studenci** będą przechowywać informacje charakteryzujące pojedynczego studenta – nazwisko, imię, datę urodzenia itd. Student, w terminologii baz danych, jest **encją**, a tabela jest zbiorem encji.

Wiersze tabeli będziemy nazywać **rekordami**. Każdy wiersz (rekord) składa się z **pól**, które tworzą kolumny tabeli. Pola przechowują jednorodną, co do typu, informację, która opisuje atrybuty (właściwości) encji. W podanym wcześniej przykładzie pola **Nazwisko** i **Imię** przechowują informację typu tekstowego, a pole **DataUrodzenia** informację typu daty.

2.1. Bazy kartotekowe

W najprostszym przypadku można sobie wyobrazić taką bazę danych, w której wszystkie informacje przechowywane są w jednej tabeli. Bazy tego typu nazywamy bazami **kartotekowymi**, pojedynczy rekord takiej tabeli zawiera jawnie wszystkie informacje opisujące encje. Przykładem tego typu bazy danych mogą być choćby listy tworzone w MS Excel. Przykładem takiej bazy może być np. tabela przechowująca podstawowe informacje o studentach.

ID_Student	Nazwisko	Imię	Miasto	DataUrodzenia
1	Kowalski	Jan	Sochaczew	1981-02-13
2	Abacki	Adam	Teresin	1978-03-11
3	Pac	Adam	Sochaczew	1983-11-02
4	Kowalska	Anna	Sochaczew	1986-11-01

Pierwsza kolumna tej tabeli przechowuje informacje, które w sposób jednoznaczny identyfikują każdy rekord (encję). Pola tego typu będziemy nazywać **kluczami**, a szerzej do tej problematyki wróćmy w dalszej części tej książki.

Kartotekowe bazy danych, poza niewątpliwą zaletą w postaci czytelności informacji gromadzonych w bazie (tabeli), mają szereg istotnych wad. Jednym z pól w pokazanym przykładzie jest pole o nazwie **Miasto**, konieczność wpisywania pełnej nazwy miasta powoduje z jednej strony niepotrzebny wzrost wielkości pliku bazy danych, z drugiej stwarza możliwość popełnienia istotnych błędów. W przykładzie celowo w rekordzie 3 zrobiliśmy czeski błąd w zapisie słowa „Sochaczew”, jego konsekwencją będzie pomijanie tego rekordu przy np. wyszukiwaniu studentów zamieszkałych w Sochaczewie.

Zobaczmy jeszcze jeden przykład kartotekowej bazy danych, jej zadaniem jest przechowywanie ocen uzyskanych z poszczególnych przedmiotów przez studentów.

ID_Oceny	Nazwisko	Imię	Przedmiot	Ocena	Data
1	Kowalski	Jan	Matematyka	4,0	2003-04-12
2	Abacki	Adam	Matematyka	4,5	2003-04-12
3	Kowalski	Jan	Ekonomia	3,5	2003-05-22
4	Pac	Adam	Filozofia	5,0	2003-08-01
5	Kowalski	Jan	Bazy danych	4,0	2003-08-01
6	Abacki	Adam	Bazy danych	5,0	2003-09-23

Widzimy, że zaprojektowanie takiej bazy jako kartotekowej ma szereg istotnych wad:

- wielokrotne powtarzanie danych studenta dla rejestracji ocen z różnych przedmiotów ze wszelkimi konsekwencjami związanymi z ewentualnym błędym wprowadzeniem tych danych,
- wielokrotne powtarzanie nazw przedmiotów z potencjalnymi błędami.

2.2. Bazy relacyjne

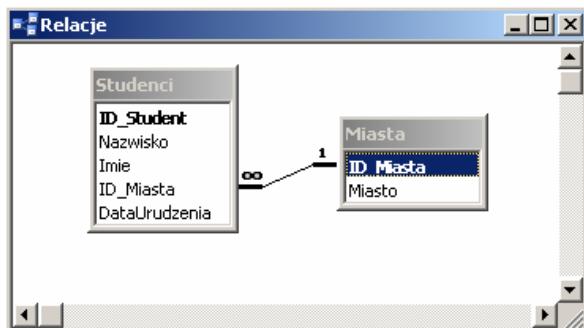
Można uniknąć pokazanych wyżej wad i potencjalnych zagrożeń rozdzielając informacje z pojedynczej tabeli na szereg tabel zawierających bardziej jednorodną информацию. W pierwszym z pokazanych przykładów wystarczy wprowadzić nową tabelę o nazwie np. Miasta:

ID_Miasta	Miasto
1	Sochaczew
2	Teresin

W konsekwencji trzeba przeprojektować dotychczasową tabelę Studenci do pokazanej niżej postaci:

ID_Student	Nazwisko	Imie	ID_Miasta	DataUrodzenia
1	Kowalski	Jan	1	1981-02-13
2	Abacki	Adam	2	1978-03-11
3	Pac	Adam	1	1983-11-02
4	Kowalska	Anna	1	1986-11-01

Obie tabele zawierają wspólne pole o nazwie `ID_Miasta`, wartości tego pola w tabeli `Studenci` pozwalają na uzyskanie jawnej nazwy miasta z tabeli `Miasta`. Powiemy, że obie tabele powiązane są **relacją** poprzez pole `ID_Miasta`.



Z pokazanego przykładu widać, że w porównaniu z bazą kartotekową mamy same zalety: z pewnością plik bazy danych będzie mniejszy, unikamy wielokrotnego powtarzania jawnych nazw miast w tabeli `Studenci`, nie ma możliwości popełnienia błędu typu literowego.

Przykład drugi jest bardziej skomplikowany, tym razem tabelę `Oceny` rozdzielimy na trzy tabele:

`Studenci` – przechowującą informację o studentach,
`Przedmioty` – przechowującą informacje o nazwach przedmiotów,

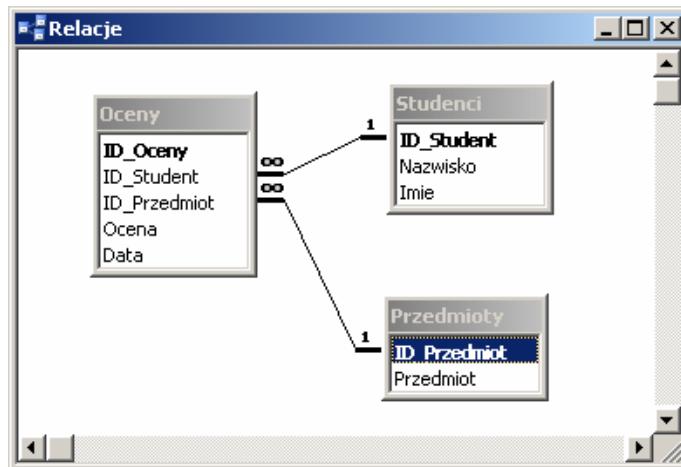
Studenci		
ID_Student	Nazwisko	Imie
1	Kowalski	Jan
2	Abacki	Adam
3	Pac	Adam

Przedmioty	
ID_Przedmiot	Przedmiot
1	Matematyka
2	Ekonomia
3	Filozofia
4	Bazy danych

Oceny – właściwą tabelę ocen powiązaną z dwoma wymienionymi wyżej tabelami.

ID_Oceny	ID_Student	ID_Przedmiot	Ocena	Data
1	1	1	4,0	2003-04-12
2	2	1	4,5	2003-04-12
3	1	2	3,5	2003-05-22
4	3	3	5,0	2003-08-01
5	1	4	4,0	2003-08-01
6	2	4	5,0	2003-09-23

Informacja przechowywana w tabeli Oceny nie jest bezpośrednio czytelna, zawiera bowiem zakodowaną informację o studencie oraz przedmiocie, ale poprzez relacje z tabelą Studenci oraz z tabelą Przedmioty istnieje możliwość zamiany informacji zakodowanej na jej jawnego odpowiednik.



2.3. Projektowanie relacyjnych baz danych

Proces projektowania relacyjnej bazy danych zaczyna się od ustalenia tego, czego oczekujemy od aplikacji, jakie zadania ma realizować projektowana baza. Jednym z ważniejszych etapów tego procesu jest ustalenie, jakie informacje będą gromadzone w bazie danych, w jakich tabelach, jaki będzie typ danych, ile i jakich pól będzie w każdej z tabel. Błędy, które mogą być popełnione na tym etapie projektowania mogą uniemożliwić lub utrudnić zrealizowanie zamierzonej funkcjonalności całej aplikacji. Istnieją pewne zasady projektowania tabel, ich przestrzeganie jest warunkiem dalszego sukcesu.

2.3.1. Normalizacja

Normalizacja tabel jest właśnie zbiorem reguł, które powinny być zastosowane na etapie projektowania tabel. Znanych jest kilka reguł normalizacji, bezsprzecznie najważniejsze z nich to pierwsze trzy zasady.

Mówimy, że tabela jest w **pierwszej postaci normalnej**, jeżeli jej pola (kolumny) zawierają informację elementarną. Przykładowo tabela, która w jednym z pól zawiera nazwisko i imię klienta nie spełnia warunków pierwszej postaci normalnej. Jeżeli jednak rozdzielimy tę informację między dwa pola, to tabela będzie już w pierwszej postaci normalnej.

Dla zilustrowania pozostałych postaci normalnych rozważmy klasyczny przykład gromadzenia danych niezbędnych do wystawienia faktury sprzedawy. Dokument tego typu zawiera informację o kupującym, czyli jego nazwę lub nazwisko, dane adresowe, numer NIP, nazwy zakupionych produktów, ich ilości i ceny jednostkowe brutto, wartość brutto, stawkę podatku VAT i kwotę tego podatku. Można wszystkie te dane zgromadzić w jednej tabeli, ale czy to będzie dobre rozwiązanie?

Po pierwsze informacja o wartości brutto zakupionego produktu nie jest informacją pierwotną, bez żadnego problemu można tę informację uzyskać mnożąc cenę jednostkową produktu przez jego ilość i stawkę VAT powiększoną o jeden, inaczej mówiąc jest to informacja **wyliczana**. Umieszczenie pola przechowującego wartość brutto narusza pierwszą postać normalną. Analogiczna uwaga dotyczy kwoty podatku VAT, to jest także informacja wyliczana i nie powinna być przechowywana w tabeli.

Po drugie w takiej tabeli każdy zakupiony produkt będzie zapisany w oddzielnym wierszu (rekordzie) i to jest poprawne, ale w takim razie w każdym rekordzie będzie się powtarzała informacja o kupującym – jego nazwa, dane adresowe, numer NIP. W rezultacie mamy tabelę, gdzie dane są wielokrotnie powtarzane, w języku baz danych powiemy, że występuje **redundancja** danych. Występowanie redundancji jest niekorzystne z wielu powodów, między innymi niepotrzebnie zwiększa rozmiar pliku bazy danych oraz zwiększa nakład pracy w momencie np. aktualizacji danych.

Jednym z celów normalizacji jest właśnie uniknięcie redundancji danych, a metodą, która do tego prowadzi jest podział tabeli (z redundancją) na kilka mniejszych tabel.

W naszym przypadku można natychmiast zaproponować podział przedstawionej wyżej tabeli na dwie tabele o nazwach odpowiednio np. *Faktura* i *DetaleFaktury*. W tabeli *Faktura* możemy wtedy przechowywać dane o kupującym – jego nazwę, dane adresowe, numer NIP, datę zakupu, a w tabeli *DetaleFaktury* takie informacje jak nazwa produktu, jego ilość, cena jednostkowa netto, stawka podatku VAT. Tabela *DetaleFaktury* musi zawierać także pole, które będzie łącznikiem z tabelą *Faktura*. Poniżej zamieszczony jest schemat obu tabel, możemy teraz zastanowić się, czy taka organizacja tabel dotyczących przechowywania danych opisujących transakcję sprzedaży jest poprawna?

Faktura

Id_f	Nazwa	Adres	DataZakupu

DetaleFaktury

Id_f	NazwaProduktu	CenaNetto	Ilosc	StawkaVAT

Nasze rozważania musimy zacząć od zdefiniowania w obu tabelach tzw. **klucza**, czyli jednego lub więcej pól, które w sposób jednoznaczny identyfikują rekordy tabeli. Jeżeli kluczem będzie pojedyncze pole, to taki klucz będziemy nazywać **kluczem prostym**, a w przypadku, gdy klucz będzie zbudowany z więcej niż jednego pola **kluczem złożonym**.

W przypadku tabeli *Faktura* wygodne będzie wprowadzenie specjalnego pola o wartościach liczbowych identyfikujących każdy rekord, w przypadku Accessa będzie to pole typu *autonumer*. Pole *autonumer* przechowuje informację typu *liczba całkowita dłuża*, przy czym silnik bazy danych dba o to, aby każda nowa wartość była automatycznie powiększana o jeden. Pole otrzymało nazwę *Id_f* i będzie także występować w tabeli *DetaleFaktury* jako tzw. **klucz obcy** wiążący obie tabele. W tabeli *DetaleFaktury* pole *id_f* **musi** być zdefiniowane jako *liczba całkowita dłuża*.¹

W przypadku tabeli *DetaleFaktury* można zdefiniować klucz prosty dodając jeszcze jedno pole typu *autonumer* albo zbudować klucz złożony. Rozważmy tę drugą możliwość. Przy założeniu, że cena netto produktu jest stała klucz złożony można zbudować z pól *Id_f* i *NazwaProduktu*. Gdyby cena netto była zmienna (np. w zależności od rodzaju opakowania, terminu przydatności itd.), to do klucza złożonego trzeba byłoby włączyć także pole *CenaNetto*.

¹ W tabeli może być tylko jedno pole typu *autonumer*

Jeżeli mamy już zdefiniowane klucze obu tabel, to spróbujmy sobie teraz wyobrazić te tabele wypełnione danymi w kontekście odpowiedzi na pytanie, czy wszystko jest dobrze zaplanowane.

Pierwsze obiekcie pojawiają się przy tabeli **Faktura**, jeżeli bowiem dany klient będzie wielokrotnie dokonywał zakupów, to w tak zaprojektowanej tabeli wielokrotnie będą się powtarzały dane klienta! Mamy więc redundancję danych, jedyne wyjście, to inne zaprojektowanie tabeli **Faktura**. Można np. zaproponować, aby dane klienta były przechowywane w oddzielnej tabeli, powiedzmy o nazwie **Klient**, a wtedy schemat obu tabel może być następujący (pogrubione są pola będące kluczami):

Klient			
Id_k	Nazwa	Adres	Nip

Faktura		
Id_f	Id_k	DataZakupu

Tabela **Faktura** będzie powiązana z tabelą **Klient** poprzez pole **Id_k**, w dalszej części tej pozycji poświęconej relacjom zdefiniujemy na tym przykładzie relację typu *jeden-do-wielu* między obu tabelami (jeden po stronie **Klient**, wiele po stronie **Faktura**).

Tak zaprojektowana tabela **Faktura** jest zarówno w pierwszej jak i drugiej postaci normalnej. W pierwszej, ponieważ zawiera tylko informacje elementarne (pierwotne), w drugiej dlatego, że nie ma powtarzania informacji (o nabywcy).

Tabela **Faktura** jest także w **trzeciej postaci normalnej**, ponieważ jej pola niebędące kluczem (tu **Id_f** i **DataZakupu**) zależą wyłącznie od klucza, a nie od jakiegoś innego pola niebędącego kluczem tej tabeli.

Spójrzmy teraz na tabelę **DetaleFaktury** w jej dotychczasowej postaci i przy założeniu, że kluczem jest kombinacja pól **Id_f** i **NazwaProduktu**, a więc przy założeniu, że cena netto jest stała (pola klucza są pogrubione).

Id_f	NazwaProduktu	CenaNetto	Ilosc	StawkaVAT

Natychmiast widać, że tak zaprojektowana tabela nie jest w drugiej postaci normalnej, ponieważ dopuszcza możliwość powtarzania informacji o nazwie produktu. Poza tym proszę zauważać, że pola **CenaNetto** i **StawkaVAT** nie zależą od całego klucza, a jedynie od jego części, czyli od nazwy produktu. Tym samym tak zaprojektowana tabela nie jest także w trzeciej postaci normalnej. W przypadku pola **Ilosc** ten warunek jest spełniony, ponieważ konkretna wartość tego pola jest zależna jednocześnie od całego klucza, jest funkcją jednocześnie pola **Id_f** i pola **NazwaProduktu**.

W sytuacji, gdyby klucz złożony był zbudowany także z pola CenaNetto, to pole StawkaVAT byłoby zależne jedynie od nazwy produktu, a nie od całego klucza, czyli w dalszym ciągu tabela nie byłaby w trzeciej postaci normalnej.

Proszę także zauważyc, że wprowadzenie klucza prostego także nie zmienia sytuacji.

FakturaDetale

Id_fd	Id_f	NazwaProduktu	CenaNetto	Ilosc	StawkaVAT

Takie pola jak CenaNetto czy StawkaVAT nie są zależne od klucza tej tabeli, a jedynie od nazwy produktu. Tym samym tabela znowu nie jest w trzeciej postaci normalnej.

Rozwiążemy nasz problem poprzez wprowadzenie tabeli Produkty, w niej będą przechowywane informacje o nazwach produktów, cenie jednostkowej netto i stawce podatku VAT.

Produkty

Id_p	NazwaProduktu	CenaNetto	StawkaVAT

W konsekwencji nowy projekt tabeli FakturaDetale może zawierać jedynie trzy pola:

FakturaDetale

Id_f	Id_p	Ilosc

W sumie informacje niezbędne do rejestrowania transakcji sprzedaży będą gromadzone w czterech wzajemnie powiązanych tabelach:

Klient

Id_k	Nazwa	Adres	Nip

Faktura

Id_f	Id_k	DataZakupu

Produkty

Id_p	NazwaProduktu	CenaNetto	StawkaVAT

FakturaDetale

Id_f	Id_p	Ilosc

Każda z tych tabel jest w pierwszej, drugiej i trzeciej postaci normalnej. Przeanalizujmy jeszcze związki, jakie występują między parami pól w tak zaprojektowanych tabelach. W tabeli Faktura każdej fakturze przyporządkowany jest jeden i tylko jeden odbiorca. Powiemy, że te dwa pola łączy zależność **funkcjonalna**.

Inny rodzaj zależności wiąże pola `Id_f` i `Id_p` w tabeli `FakturaDetale`, w tym przypadku każdej wartości pola `Id_f` może odpowiadać wiele wartości pola `Id_p`, powiemy, że pola te łączy zależność **wielowartościowa**.

Dotychczas mówiliśmy o pierwszych trzech postaciach normalnych, ale jest jeszcze jedna, czwarta postać normalna. Rozważmy ją na przykładzie uzupełnienia informacji o kliencie poprzez dodanie pola o numerze telefonu kontaktowego. Natychmiast powstaje problem jak to zrobić. Uzupełnienie projektu tabeli `Klient` o pole np. `NumerTelefonu` nie jest najlepszym rozwiązaniem z kilku co najmniej powodów. Jednym z nich jest problem liczby telefonów kontaktowych, konieczne byłoby dodanie kilku pól na numery telefonów (np. `NumerTelefonu1`, `NumerTelefonu2` itd.). Jeżeli dodamy dwa pola tego typu, to dla niektórych klientów będzie to zbyt dużo (bo mają tylko jeden telefon), dla innych z kolei zbyt mało (bo mają więcej niż dwa telefony). Jeszcze gorzej wygląda możliwość przechowania danych (nazwiska) osoby kontaktowej dostępnej pod danym numerem, zwłaszcza wtedy, gdy dana osoba może korzystać z kilku telefonów albo z jednego telefonu korzysta kilku rozmówców.

Jedynym rozwiązaniem jest wprowadzenie kolejnej tabeli, w niej będziemy przechowywać identyfikator klienta oraz numery telefonów i nazwiska osób kontaktowych. Schemat takiej tabeli pokazany jest poniżej.

`TelefonyKlienta`

<code>Id_k</code>	<code>NumerTelefonu</code>	<code>OsobaKontaktowa</code>

W tabeli tej można zdefiniować klucz złożony będący kombinacją wartości wszystkich trzech pól tej tabeli. Tak zaprojektowana tabela jest w trzeciej postaci normalnej (tym samym także w drugiej i pierwszej), niby wszystko jest poprawnie zaprojektowane, w praktyce jednak pojawi się problem z aktualizacją danych w tak zaprojektowanej tabeli.

Rozpatrzmy sytuację, gdy dany klient zmienił numer telefonu kontaktowego, a w naszej tabeli numer ten wystąpił np. dwukrotnie, bo dwie różne osoby kontaktowe mogły z tego numeru korzystać. Aktualizacja pojedynczej informacji będzie wymagała zmiany w dwóch wierszach tabeli `TelefonyKlienta`. Podobna sytuacja będzie wtedy, gdy klient poinformuje nas o dodatkowym numerze telefonu. W tabeli `TelefonyKlienta` musi się pojawić tyle rekordów, ile osób kontaktowych może korzystać z tego nowego numeru.

Taka sytuacja powstaje wtedy, gdy między parami pól (nietworzącymi klucza) istnieją zależności wielowartościowe. Niestety, taka sytuacja występuje w naszej tabeli. Pary `Id_k` i `NumerTelefonu`, `Id_k` i `OsobaKontaktowa` oraz `NumerTelefonu`

i OsobaKontaktowa powiązane są zależnościami wielowartościowymi. Polu `id_k` może odpowiadać więcej niż jeden numer telefonu, z identyfikatorem klienta może być związana więcej niż jedna osoba kontaktowa, a z jednego numeru telefonu może korzystać więcej niż jedna osoba.

Możemy uniknąć tych niedogodności poprzez doprowadzenie tej tabeli do **czwartej** postaci normalnej, a więc takiego jej zaprojektowania, aby między parami pól występowały zależności funkcjonalne. Zrobimy to dzieląc dotychczasową tabelę TelefonyKlienta na dwie nowe tabele.

TelefonyKlienta

Id_k	NumerTelefonu

TelefonyOsobyKontaktowe

Id_k	OsobaKontaktowa

Dzięki temu zależności wielowartościowe będą dotyczyć kluczy tych dwóch tabel, a obie tabele będą zgodne z czwartą postacią normalną.

Dla pewnego podsumowania jeszcze raz sformułujemy warunki, jakie musi spełnić projekt tabeli, aby tabela była w danej postaci normalnej.

Pierwsza postać normalna – tabela jest w pierwszej postaci normalnej wtedy, gdy jej pola przechowują informację elementarną. Metodą na doprowadzenie tabeli do tej postaci jest zapisanie informacji złożonej w oddzielnych polach oraz nie tworzenia pól dla informacji wyliczanych.

Druga postać normalna – tabela jest w drugiej postaci normalnej, jeżeli jest w pierwszej postaci normalnej oraz każde jej pole niewchodzące w skład klucza zależy od całego klucza, a nie od jego części. Tabela z kluczem prostym jest automatycznie w drugiej postaci normalnej. Metodą na doprowadzenie tabeli „anormalnej” do drugiej postaci normalnej jest jej podział na kilka wzajemnie powiązanych tabel.

Trzecia postać normalna – tabela jest w trzeciej postaci normalnej wtedy, gdy jest w drugiej postaci normalnej oraz jej pola nienależące do klucza głównego zależą jedynie od klucza tabeli. Warunek ten oznacza, że nie ma związku funkcyjnego między polem niewchodzącym w skład klucza a innym polem takiej tabeli również niebędącym składnikiem klucza. Metodą na doprowadzenie tabeli „anormalnej” do trzeciej postaci normalnej jest jej podział na kilka wzajemnie powiązanych tabel.

Czwarta postać normalna – tabela jest w czwartej postaci normalnej, jeżeli między jej polami nie występują zależności wielowartościowe. Warunek ten nie dotyczy klucza tabeli.

2.3.2. Relacje, więzy integralności

Po doprowadzeniu tabel do co najmniej drugiej postaci normalnej musimy zdefiniować związki między pokrewnymi tabelami. W omawianym w poprzednim podrozdziale przykładzie musimy powiązać z sobą cztery tabele (*FakturaDetaile*, *Faktura*, *Klient* i *Produkty*), a tak naprawdę to pięć tabel, ponieważ powinniśmy jeszcze przedefiniować tabelę *Klient* poprzez rozdzielnie pola *Adres* na co najmniej dwa pola opisujące miejscowości i pozostałą część adresu.

Będzie to wymagało utworzenia dodatkowej tabeli *Miejscowosc*, o strukturze pokazanej niżej:

Miejscowosc	
Id_m	Miejscowosc

a w konsekwencji zmianę projektu tabeli *Klient*.

Klient				
Id_k	Nazwa	Id_m	Adres	Nip

Korzystając z menu *Relacje* dodajemy do okna relacji te pięć tabel i kolejno będziemy definiować relacje (związki) między odpowiednimi polami dwóch tabel.

Generalnie będziemy rozróżniać trzy typy relacji:

- *jeden-do-jednego* – każdemu rekordowi z jednej tabeli odpowiada dokładnie jeden rekord z powiązanej tabeli,
- *jeden-do-wielu* – każdemu rekordowi z jednej tabeli odpowiada wiele rekordów z drugiej tabeli,
- *wiele-do-wiele* – w relacji tego typu każdemu rekordowi z jednej tabeli odpowiada wiele rekordów z drugiej tabeli i odwrotnie.

W zastosowaniach praktycznych podstawową relacją będzie relacja *jeden-do-wielu*, relacja *jeden-do-jeden* może mieć uzasadnienie np. w takiej sytuacji, gdy np. dane klienta chcemy rozdzielić na dwie tabele; jedną ogólnie dostępną i drugą zawierającą wyłącznie dane poufne. Można wtedy zastosować ograniczony dostęp do takich danych, a obie tabele będą wtedy połączone właśnie relacją *jeden-do-jednego*.

W praktyce relacja typu *wiele-do-wiele* nie znajduje zastosowania, zawsze bowiem można ją sprowadzić do relacji typu *jeden-do-wiele*.

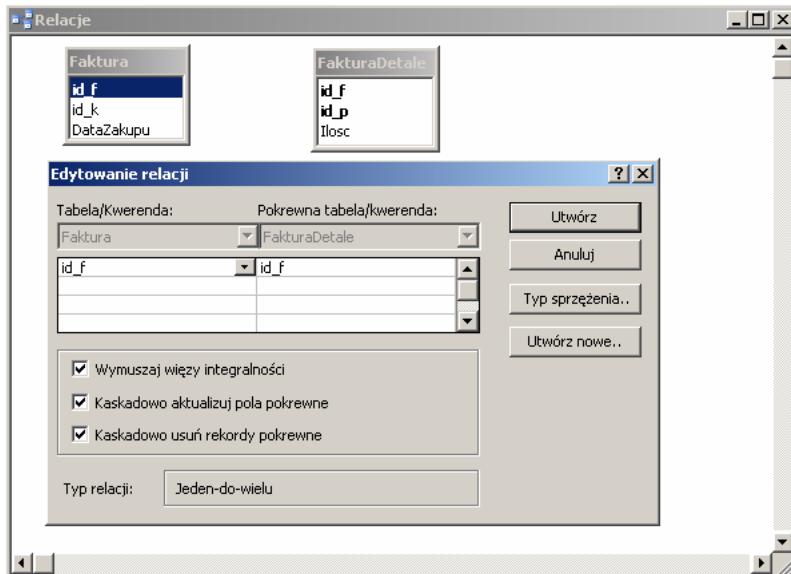
Dla utworzonej relacji zasadnicze znaczenie ma włączenie opcji *Wymuszaj więzy integralności*, dzięki czemu silnik bazy danych **nie dopuści** do takiej sytuacji, aby przykładowo w tabeli *Faktura* skasować rekord opisujący jakąś transakcję sprzedaży bez

wcześniejszego skasowania powiązanych z nią rekordów z tabeli FakturaDetale. Dzięki włączeniu tej opcji nie będzie także możliwe nadanie polu Id_f w tabeli FakturaDetale wartości, która nie istnieje w tabeli Faktury.

Włączenie opcji *Wymuszaj więzy integralności* wymaga spełnienia kilku warunków:

- definiowana relacja dotyczy dwu tabel przechowywanych w tej samej bazie danych (a nie np. tzw. tabel połączonych, które fizycznie istnieją w innej bazie danych),
- pole łączące jest dla tabeli głównej kluczem tej tabeli lub jest jednoznacznie indeksowane (bez duplikatów),
- odpowiadające sobie pola są tego samego typu (ale niekoniecznie mają tę samą nazwę). Wyjątkiem jest pole typu *autonumer*, któremu może odpowiadać pole typu *liczba całkowita dluza*.

W pokazanej sytuacji w oknie relacji widoczne są tabele FakturaDetale i Faktura, tabele te będą powiązane poprzez pole id_f (identyfikator faktury). Tabela Faktura jest tu tabelą główną, a tabela FakturaDetale tabelą powiązaną. Dla utworzenia relacji przeciągamy myszą (techniką „ciagnij i upuść”) pole id_f z tabeli głównej na odpowiadające mu pole w tabeli powiązanej, co skutkuje wyświetleniem okna *Edytowanie relacji*.



W oknie dialogowym *Edytowanie relacji* mamy pokazane pola wiążące w tabeli głównej i tabeli powiązanej, możemy dodatkowo zaznaczyć trzy pola wyboru:

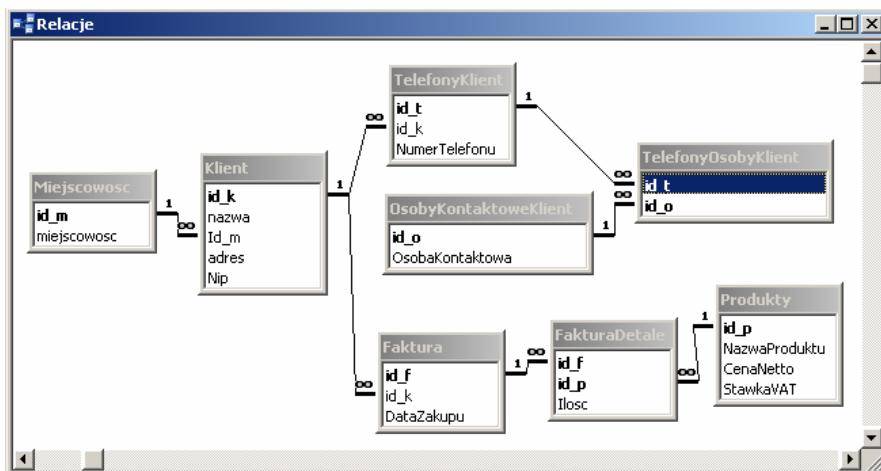
- Wymuszaj więzy integralności,*
- Kaskadowo aktualizuj pola pokrewne,*
- Kaskadowo usuń rekordy pokrewne.*

O znaczeniu opcji *Wymuszaj więzy integralności* już pisaliśmy wcześniej, można dodać jeszcze tylko jedną uwagę odnośnie tego, co się będzie działo w sytuacji, gdy to pole wyboru nie będzie wybrane (nie będzie aktywne). W takiej sytuacji zdefiniowana relacja będzie miała swoje znaczenie tylko w momencie definiowania kwerend, jednocześnie silnik bazy danych nie będzie pilnował związku między powiązanymi tabelami. Pozostałe dwie opcje będą dostępne tylko wtedy, gdy opcja *Wymuszaj więzy integralności* jest aktywna.

Opcja *Kaskadowo aktualizuj pola pokrewne* jest bardzo potrzebna w sytuacji, gdy po stronie tabeli głównej zostaną zmienione wartości pola łączącego. Jeżeli opcja jest aktywna, to zmiana taka spowoduje zmianę wartości pola łączącego w tabeli powiązanej.

Włączenie opcji *Kaskadowo usuń rekordy pokrewne* spowoduje, że będzie możliwe skasowanie rekordu z tabeli głównej bez wcześniejszego skasowania powiązanych rekordów z tabeli pokrewnej. Po prostu w momencie usuwania takiego rekordu np. z tabeli Faktura silnik bazy danych najpierw usunie wszystkie powiązane rekordy z tabeli pokrewej FakturaDetaile i dopiero usunie rekord z tabeli Faktura.

Poniżej pokazane jest okno wszystkich relacji w naszym przykładzie, z tym że tabele TelefonyKlient i OsobyKontaktowe zostały trochę przeprojektowane, pojawiła się jeszcze jedna tabela o nazwie TelefonyOsobyKlient jako wiążąca telefon i osobę kontaktową dla danego klienta.



2.3.4. Kwerendy

Informacje umieszczone w wielu powiązanych z sobą (poprzez relacje) tabelach mogą być wyszukiwane przy pomocy kwerend. Kwerendy w relacyjnych bazach danych potrafią nie tylko wyciągnąć z tabel dane pierwotne, ale potrafią je także odpowiednio przeliczyć, zagregatować, zrobić zestawienie krzyżowe czy także wykonać operacje modyfikujące dane w tabelach.

W MS Access rozróżniamy dwie podstawowe grupy zapytań (kwerend): kwerendy **wybierające i akcyjne**.

Zadaniem kwerend wybierających jest zwrócenie do dalszego wykorzystania wybranych pól z jednej czy większej liczby tabel, ewentualnie zwrócenie pola wyliczanego (np. wartość brutto jako iloczynu ilości produktu przez cenę jednostkową), pogrupowania danych przy użyciu funkcji agregatujących, posortowanie zwracanych rekordów czy ograniczenie ich liczby poprzez zdefiniowanie odpowiedniego kryterium.

Zadaniem kwerend akcyjnych jest wykonanie określonej operacji na danych zgromadzonych w tabelach, najczęściej będziemy je wykorzystywać do aktualizacji jednego czy kilku pól dla określonej grupy rekordów, do usuwania wskazanych rekordów, czy też do dołączenia do istniejącej tabeli nowych rekordów.

Kwerendy są budowane przy pomocy języka SQL, jednak w MS Access większość z nich może być zaprojektowana w graficznym interfejsie projektowania zapytań, gdzie większość prac jest wykonywana myszką komputerową. Są jednak pewne rodzaje zapytań, które mogą być wykonane jedynie przy pomocy języka SQL, takim przykładem jest kwerenda **składająca**, która łączy dane z kilku tabel o takiej samej strukturze. Oczywiście graficzny interfejs projektowania zapytań generuje odpowiednią kwerendę w SQL, tylko taka instrukcja jest zrozumiała dla silnika bazy danych i może być wykonana. Myślę, że dobrym rozwiązaniem jest budowanie kwerend w interfejsie graficznym, ale warto przeanalizować postać instrukcji (poleceń) SQL definiującego dane zapytanie. Znajomość języka SQL jest niezbędna, jeżeli będziemy chcieli wzbogacić projektowaną bazę danych o nowe możliwości przy pomocy procedur języka VBA. W dalszej części tej książki zarówno językowi SQL jak i programowaniu w VBA poświęcone są oddzielne rozdziały.

Do pokazania, jak w interfejsie graficznym można projektować kwerendy wykorzystamy bazę danych o nazwie `Wstep.mdb`, w bazie tej w poprzednim rozdziale zaprojektowaliśmy kilka tabel związanych z wystawianiem dokumentów sprzedaży.

Kwerendy wybierające.

Zaczniemy od przygotowania kwerendy zwracającej listę naszych klientów wraz z nazwą miejscowości i z przypisaniem wszystkich telefonów danego klienta i osób kontaktowych. Do zbudowania kwerendy wykorzystamy tabele `Klient`, `Miejscowosc`, `TelefonyKlient`, `TelefonyOsobyKlient` i `OsobyKontaktoweKlient`. Tabele te zostały wypełnione niewielką liczbą danych, tak jak to pokazano poniżej.

The screenshot shows four database tables displayed in separate windows:

- Klient : Tabela** (Client Table):

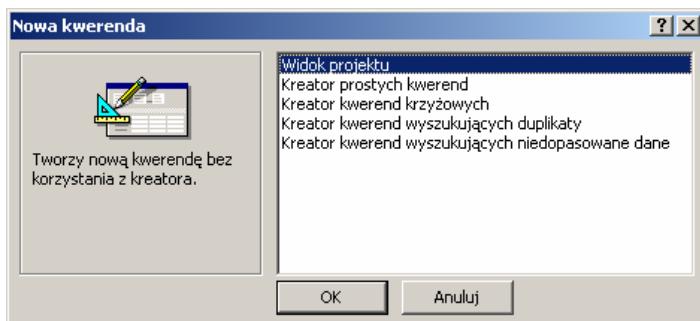
	id_k	nazwa	Id_m	adres	Nip
▶	1	Kowalski & Synowie, Sp. Z o.o	1	Jasna 23	234-456-45-56
▶	2	Abacki, Spółka Cywilna	2	Wiśniowa 123	123-345-45-45
- Miejscowosc : Tabela** (Location Table):

	id_m	miejscowosc
▶	1	Baranów
▶	2	Jamno
- TelefonyKlient : Tabela** (Client Phone Numbers Table):

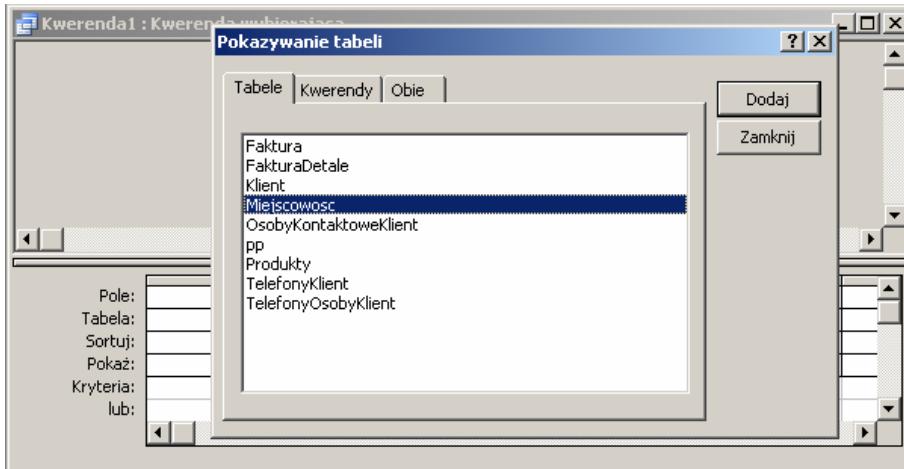
	id_t	id_k	NumerTelefonu
▶	1	1	1 234-45-34
▶	2	1	1 234-45-37
▶	3	2	2 (46) 846-45-67
- OsobyKontaktoweKlient : Tabela** (Client Contact Persons Table):

	id_o	OsobaKontaktowa
▶	1	Kowalski Adam
▶	2	Pacan Jan
▶	3	Abacki Adam
▶	4	Kowalski Jan
▶	5	Wróbel Zygmunt

Prace nad projektem kwerendy zaczynamy od wywołanie polecenia *Nowy* w obiekcie *Kwerendy* okna bazy danych.

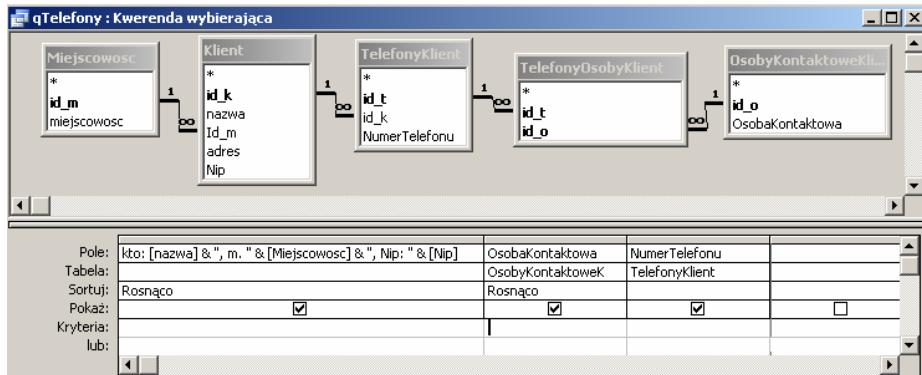


Spośród dostępnych opcji wybieramy pozycję *Widok projektu* z tego powodu, że chcemy zbudować kwerendę wybierającą zwracającą informacje z więcej niż jednej tabeli. Opcja *Kreator prostych kwerend* nie wchodzi w rachubę, ponieważ ograniczona jest do pól jednej tabeli. Po akceptacji przycisku *OK* przechodzimy już do okna projektowania kwerendy.



Projektowanie kwerendy zaczynamy od wstawienia do jej projektu wszystkich potrzebnych źródeł danych, czyli tabel lub wcześniej zdefiniowanych kwerend. W pokazanym przykładzie wskazana jest tabela **Miejscowosc**, poprzez przycisk *Dodaj* lub podwójny klik na nazwie źródła dodajemy je do projektu kwerendy.

Po dodaniu wszystkich źródeł danych i ewentualnym uzupełnieniu relacji między nimi przenosimy potrzebne pola z poszczególnych tabel (kwerend) do wiersza *Pole* w dolnej części okna kreatora zapytań.



W przypadku pól wyliczanych w wierszu *Pole* danej kolumny kwerendy musimy wpisać **wyrażenie** zwracające potrzebną informację. Pole wyliczane będzie widoczne w kwerendzie pod aliasem, który rozpoczyna wspomniane wyrażenie. Symbol dwukropka jest separatorem rozdzielającym alias pola wyliczanego od właściwego wyrażenia.

W pokazanym przykładzie pierwsza kolumna zapytania zwróci informację opisaną aliasem `kto`, a zdefiniowaną wg wyrażenia:

```
kto: [nazwa] & ", m. " & [Miejscowosc] & ", Nip: " & [Nip]
```

które łączy pola tekstowe z tabel `Miejscowosc` i `Klient`. Do „łączenia” informacji typu tekstowego wykorzystany jest operator `&` (w MS Access, w MS SQL będzie to symbol `+`).

Kolejność pól w projekcie kwerendy jest oczywiście podyktywana wymogami logicycznymi, przy czym możliwe jest zdefiniowanie pól wyliczanych, które korzystają z wcześniej zdefiniowanych pól wyliczanych w tym samym projekcie (w MS Access).

Informacje zwieracane przez zapytanie mogą być uporządkowane poprzez ich sortowanie wg jednego lub większej liczby pól, przy czym kolejność sortowania jest zgodna z wyborem klucza (w naszym przykładzie najpierw po polu `kto`, a następnie po polu `OsobaKontaktowa`). Sortowanie może być rosnące lub malejące.

Poniżej pokazany jest widok danych zwróconych przez tak zaprojektowaną kwerendę.

qTelefony : Kwerenda wybierająca			
	kto	OsobaKontaktowa	NumerTelefonu
▶	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Kowalski Jan	(46) 846-45-67
	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Wróbel Zygmunt	(46) 846-45-67
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Abacki Adam	234-45-37
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Kowalski Adam	234-45-37
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Kowalski Adam	234-45-34
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Kowalski Jan	234-45-34
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Pacan Jan	234-45-34
*			

Gdybyśmy byli zainteresowani uzyskaniem informacji wyłącznie o osobach kontaktowych danego klienta, to **nie wystarczy** w pokazanym wyżej projekcie usunięcie pola `NumerTelefonu`.

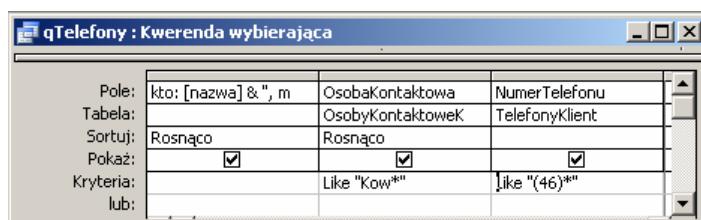
qOsobyKontaktowe : Kwerenda wybierająca		
	kto	OsobaKontaktowa
▶	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Wróbel Zygmunt
	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Kowalski Jan
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Kowalski Jan
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Kowalski Adam
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Abacki Adam
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Pacan Jan
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Kowalski Adam
*		

Łatwo zauważyc, że „Kowalski Adam” występuje dwukrotnie w ramach tego samego klienta „Kowalski & Synowie”. Można temu zaradzić ustawiając właściwość *Rekordy unikatowe* kwerendy na Tak.



qOsobyKontaktowe : Kwerenda wybierająca		
	kto	OsobaKontaktowa
▶	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Wróbel Zygmunt
	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Kowalski Jan
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Kowalski Jan
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Abacki Adam
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Pacan Jan
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56	Kowalski Adam
*		

Jak wspomnieliśmy wcześniej zwracane informacje mogą być ograniczone do tych rekordów, które spełniają odpowiednie kryteria. W pokazanym przykładzie wcześniej omawianej kwerendy qTelefony w wierszu *Kryteria* zdefiniowano dwa warunki ograniczające liczbę rekordów.



Dla pola OsobaKontaktowa wpisano warunek Like „Kow*”, jego zadaniem jest ograniczenie liczby rekordów do tych, dla których nazwisko osoby kontaktowej zaczyna się od znaków „Kow”.

Operator Like służy do sprawdzenia zgodności tekstu ze wzorcem, symbol gwiazdki oznacza dowolny dalszy tekst. Tym samym „Kow*” oznacza zarówno nazwisko „Kowalski” jak i „Kowalik” czy „Kowczyński”.

W tym samym wierszu kryteriów dla pola NumerTelefonu zdefiniowano warunek Like „(46)*”, jego zadaniem jest ograniczenie rekordów do tych, w których pole telefon zawiera numer kierunkowy 46. Oba warunki zostały umieszczone w tym samym wierszu kryteriów, co oznacza, że zostały połączone operatorem AND. Tym samym lista rekordów zostanie ograniczona wyłącznie do tych, dla których oba warunki będą spełnione **jednocześnie**. Wynik działania tak zdefiniowanej kwerendy pokazany jest poniżej.

qTelefony : Kwerenda wybierająca			
	kto	OsobaKontaktowa	NumerTelefonu
▶	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Kowalski Jan	(46) 846-45-67
Rekord:	◀ ◀ 2 ▶ ▶ * z 2		

Przeniesienie jednego z tych warunków do innego wiersza kryteriów jest równoważne połączeniu obu warunków za pomocą operatora OR, co oznacza, że zostaną zwrócone te rekordy, dla których będzie spełniony **co najmniej jeden** z tych dwóch warunków. Poniżej taka właśnie sytuacja.

qTelefony : Kwerenda wybierająca			
Pole:	kto: [nazwa] & ", m	OsobaKontaktowa	NumerTelefonu
Tabela:	OsobyKontaktoweK	TelefonyKlient	
Sortuj:	Rosnąco	Rosnąco	
Pokaż:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kryteria:	Like "Kow*" lub: Like "(46)*"		
	<input type="button" value="◀"/>	<input type="button" value="▶"/>	

qTelefony : Kwerenda wybierająca			
	kto	OsobaKontaktowa	NumerTelefonu
▶	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Kowalski Jan	(46) 846-45-67
	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Wróbel Zygmunt	(46) 846-45-67
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-45-37	Kowalski Adam	234-45-37
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-45-34	Kowalski Adam	234-45-34
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-45-34	Kowalski Jan	234-45-34
*			
Rekord:	◀ ◀ 1 ▶ ▶ * z 5		

Przed rozpatrzeniem kolejnych przykładów zapytań wprowadzimy kilka danych do tabel **Produkty**, **Faktura** i **FakturaDetale**.

Produkty : Tabela

	id_p	NazwaProduktu	CenaNetto	StawkaVAT
▶	1	Krążki CD	2,21 zł	22%
▶	2	Krążki DVD	12,45 zł	22%
▶	3	Mysz komputerowa	45,50 zł	22%
*	(Autonumerowanie)		0,00 zł	22%

Rekord: [◀ ▶] 1 [▶ ▶] [*] z 3

Faktura : Tabela

	id_f	id_k	DataZakupu
▶	1	1	2005-11-02
▶	2	2	2005-11-05
▶	3	1	2005-11-07
▶	4	1	2005-11-12
*	(nowanie)		2005-12-01

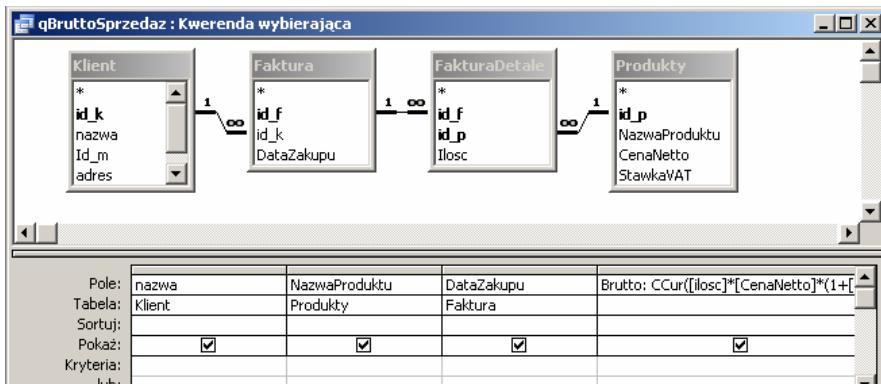
Rekord: [◀ ▶] 1 [▶ ▶] [*] z 4

FakturaDetale : Tabela

	id_f	id_p	Ilosc
▶	1	1	3
	1	2	12
	2	1	15
	2	2	15
	2	3	2
	3	1	55
	3	3	2
	4	2	250
	4	3	50
*			0

Rekord: [◀ ▶] 1 [▶ ▶] [*] z 9

Pokazana niżej kwerenda zwraca informacje o zakupionych produktach i dątach transakcji. Pole wyliczane **Brutto** zwraca wartość brutto zakupionego produktu, a funkcja konwertująca **CCur** została użyta po to, aby dokonać konwersji wyrażenia (tu: **Ilosc * CenaNetto * (1+StawkaVAT)**) do formatu waluty.



A tak wyglądają zwrócone przez kwerendę **qBruttoSprzedaz** informacje.

qBruttoSprzedaz : Kwerenda wybierająca

Nazwa klienta	NazwaProduktu	DataZakupu	Brutto
Kowalski & Synowie, Sp. Z o.o	Krążki CD	2005-11-02	8,09 zł
Kowalski & Synowie, Sp. Z o.o	Krążki DVD	2005-11-02	182,27 zł
Abacki, Spółka Cywilna	Krążki CD	2005-11-05	40,44 zł
Abacki, Spółka Cywilna	Krążki DVD	2005-11-05	227,84 zł
Abacki, Spółka Cywilna	Mysz komputerowa	2005-11-05	111,02 zł
Kowalski & Synowie, Sp. Z o.o	Krążki CD	2005-11-07	148,29 zł
Kowalski & Synowie, Sp. Z o.o	Mysz komputerowa	2005-11-07	111,02 zł
Kowalski & Synowie, Sp. Z o.o	Krążki DVD	2005-11-12	3 797,25 zł
Kowalski & Synowie, Sp. Z o.o	Mysz komputerowa	2005-11-12	2 775,50 zł

Rekord: [◀] [◀] [1] [▶] [▶] [*] z 9

Powiedzmy, że chcemy ograniczyć liczbę rekordów do tych jedynie, gdzie wartość brutto jest zawarta między pewnymi dwoma liczbami, np. 100 i 200 zł. Odpowiedni warunek można zapisać przy pomocy „zwykłych” operatorów relacji połączonych operatorem logicznym And:

qBruttoSprzedaz : Kwerenda wybierająca

Pole:	nazwa	NazwaProduktu	DataZakupu	Brutto: CCur([Ilosc]*[CenaNetto]*(1+[StawkaVAT]))
Tabela:	Klient	Produkty	Faktura	
Sortuj:				
Pokaż:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kryteria:				>=100 And <=200
lub:				

lub za pomocą operatora Between:

qBruttoSprzedaz : Kwerenda wybierająca

Pole:	nazwa	NazwaProduktu	DataZakupu	Brutto: CCur([Ilosc]*[CenaNetto]*(1+[StawkaVAT]))
Tabela:	Klient	Produkty	Faktura	
Sortuj:				
Pokaż:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kryteria:				Between 100 And 200
lub:				

Jako efekt działania tak zdefiniowanej kwerendy zwrócone zostaną 4 rekordy.

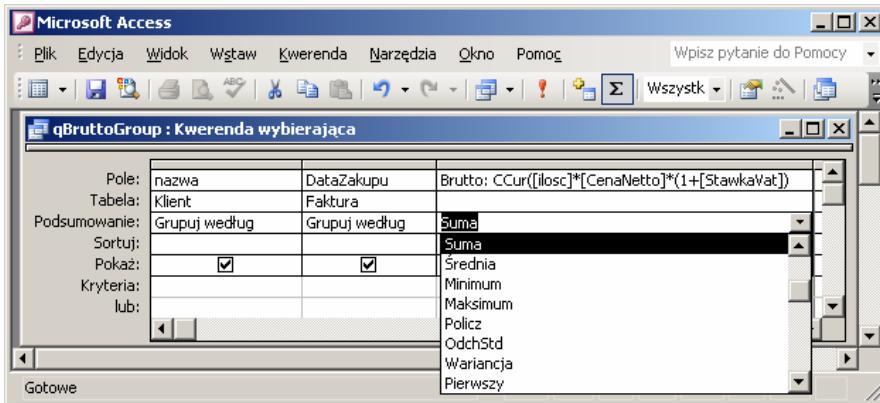
qBruttoSprzedaz : Kwerenda wybierająca

Nazwa klienta	NazwaProduktu	DataZakupu	Brutto
Kowalski & Synowie, Sp. Z o.o	Krążki DVD	2005-11-02	182,27 zł
Abacki, Spółka Cywilna	Mysz komputerowa	2005-11-05	111,02 zł
Kowalski & Synowie, Sp. Z o.o	Krążki CD	2005-11-07	148,29 zł
Kowalski & Synowie, Sp. Z o.o	Mysz komputerowa	2005-11-07	111,02 zł

Rekord: [◀] [◀] [1] [▶] [▶] [*] z 4

Kwerenda wybierająca może także grupować rekordy wg określonych pól, przykładem takiej kwerendy będzie modyfikacja kwerendy qBruttoRazem w taki sposób, aby została zwrócona łączna wartość transakcji opisanej daną fakturą.

Projekt takiej kwerendy pokazany jest poniżej, wyjątkowo w oknie Accessa po to, aby widoczny był włączony przycisk podsumowania w pasku narzędziowym.



Dzięki włączonemu podsumowaniu okno projektu zostało rozbudowane o wiersz *Podsumowanie*, w tym wierszu dla każdej kolumny kwerendy będziemy mogli określić jej rolę w zapytaniu poprzez wybór odpowiedniej opcji z rozwijanej listy.

W porównaniu z oryginalnym projektem kwerendy qBruttoRazem w tej kwerendzie usunięto pole *NazwaProduktu*, ponieważ nie interesuje nas wartość brutto produktu w ramach danej faktury, a jedynie łączna wartość zakupów opisana taką fakturą.

Każda faktura dotyczy określonego klienta, jest także wystawiona w określonym dniu, stąd dla obu pól w wierszu *Podsumowanie* wybrano opcję *Grupuj według*. Każdej fakturze odpowiada określona łączna wartość zakupów, stąd dla pola *Brutto* wybrano funkcję agregującą *Suma*.

Efekt działania tak zdefiniowanej kwerendy pokazany jest poniżej.

	Nazwa klienta	DataZakupu	Brutto
▶	Abacki, Spółka Cywilna	2005-11-05	379,30 zł
	Kowalski & Synowie, Sp. Z o.o	2005-11-02	190,36 zł
	Kowalski & Synowie, Sp. Z o.o	2005-11-07	259,31 zł
	Kowalski & Synowie, Sp. Z o.o	2005-11-12	6 572,75 zł

Kwerendy krzyżowe

Zadaniem tego typu kwerend jest zwrócenie wybranej informacji w formie, która umożliwia analizę danych. Kwerenda krzyżowa wykorzystuje jedno z pól źródła danych do pogrupowania informacji w kolumny oraz do trzech pól do pogrupowania informacji w wiersze. Na przecięciu wiersza i kolumny wyświetlana jest przetworzona zawartość tego pola, które jest przedmiotem analizy. Dla pól nienumerycznych możliwe jest zastosowanie jedynie funkcji **Policz**, dla pól numerycznych dostępnych jest wiele funkcji agregatujących (np. Suma, Min, Max).

Dla demonstracji na podstawie kwerendy **qBruttoGroup** zbudujemy kwerendę krzyżową, która w kolumnach zwróci nazwy klientów, w wierszach daty sprzedaży, a na przecięciu sumaryczną wartość transakcji danej firmy w danym dniu. Dodatkowo dodamy wiersz podsumowania zwracający wartość transakcji w danym dniu.

Poniżej pokazany jest projekt kwerendy krzyżowej wykorzystującej jako źródło danych kwerendę grupującą **qBruttoGroup**. Lista właściwości tej kwerendy została uzupełniona o wiersz *Krzyżowe*, to w tym wierszu będziemy określać rolę każdego pola kwerendy w strukturze kwerendy krzyżowej.



W naszym przypadku w wierszach zestawienia krzyżowego chcemy mieć daty transakcji, stąd w wierszu *Krzyżowe* wybrano opcję Nagłówek wiersza dla pola **DataZakupu**. W kolumnach chcemy mieć nazwy klientów, stąd wybranie opcji Nagłówek kolumny dla pola **nazwa**. Pole **Brutto** zawiera już łączną wartość zakupów klienta w danym dniu (tak zostało to pole zdefiniowane w **qBruttoGroup**), tym samym nie jest potrzebna żadna funkcja agregatująca – stąd opcja **Pierwszy** w wierszu *Podsumowanie*. W wierszu *Krzyżowe* dla tego pola wybrano opcję **Wartość**.

Listę pól projektowanej kwerendy uzupełnia pole wyliczane **Razem**, które odwołuje się do istniejącego pola **Brutto**, ale dla którego w wierszu *Podsumowanie* wybrano funkcję agregatującą **Suma**, Pole to ma zwracać sumaryczną wartość zakupów w danym dniu, stąd wybrana opcja Nagłówek wiersza w pozycji *Krzyżowe*.

W przypadku pola DataZakupu zostało jeszcze włączone sortowanie rosnące, a zwracane informacje przez tak zdefiniowaną kwerendę pokazane są poniżej.

	DataZakupu	Razem	Abacki, Spółka	Kowalski & Syno
▶	2005-11-02	190,36 zł		190,36 zł
	2005-11-05	379,30 zł	379,30 zł	
	2005-11-07	259,31 zł		259,31 zł
	2005-11-12	6 572,75 zł		6 572,75 zł

Kwerenda aktualizująca

Zadaniem tego typu kwerend jest modyfikacja istniejących danych w tabelach bazy danych polegająca na przypisaniu danemu polu nowej wartości. Projekt i funkcjonowanie takiej kwerendy pokażemy na przykładzie modyfikacji pola NumerTelefonu w tabeli TelefonyKlient. W tej tabeli telefony klienta o identyfikatorze 1 nie zawierają numeru kierunkowego.

	id_t	id_k	NumerTelefonu
▶	1		1 234-45-34
▶	2		1 234-45-37
▶	3		2 (46) 846-45-67
*	Autonumerowanie)		

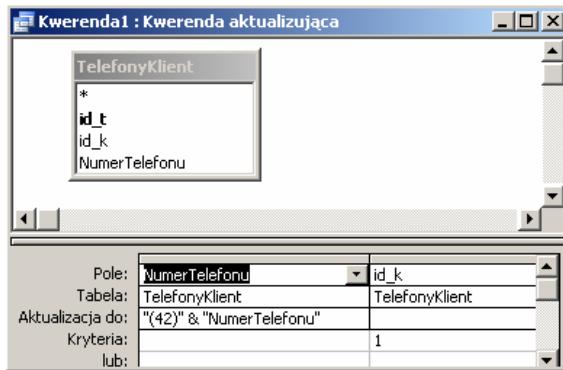
Powiedzmy, że ten numer kierunkowy to „(42)”, musimy więc dopisać ten numer na początku istniejącego już numeru klienta. Zrobimy to za pomocą kwerendy aktualizującej.

Zaczynamy od otwarcia nowego projektu kwerendy, do którego dodajemy tabelę TelefonyKlient. Korzystając z menu *Kwerenda* lub menu kontekstowego wybieramy jako typ kwerendy kwerendę dołączającą, co skutkuje dodaniem do właściwości projektu kwerendy wiersza *Aktualizacja do*.

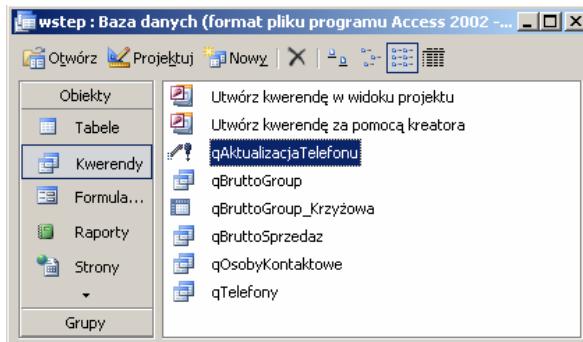
W projekcie musimy teraz umieścić pole, które ma być aktualizowane (u nas będzie to pole NumerTelefonu) oraz w wierszu *Aktualizuj do* określić jego nową wartość. Może to być wartość **statyczna** (niemająca żadnego związku z poprzednią wartością, w konsekwencji jednakowa dla wszystkich rekordów podlegających aktualizacji) lub wartość **dynamiczna** (nowa wartość tego pola jest jakąś funkcją poprzedniej wartości). W naszym przypadku będzie to zmiana dynamiczna polegająca na uzupełnieniu dotychczasowego numeru telefonu o numer kierunkowy „(42)”. Efekt taki osiągniemy poprzez przypisanie właściwości *Aktualizuj do* wyrażenia:

```
"(42)" & TelefonyKlient.NumerTelefonu
```

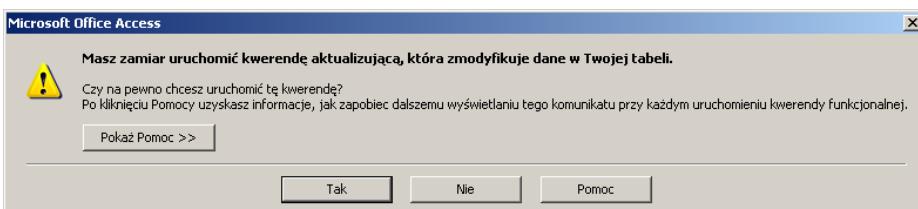
Projekt kwerendy musimy jeszcze uzupełnić o pole `id_k` identyfikujące klienta, pole to wykorzystamy do zdefiniowania warunku ograniczającego aktualizację jedynie do tych rekordów, które dotyczą klienta o `id_k` równym 1. Poniżej projekt tej kwerendy.



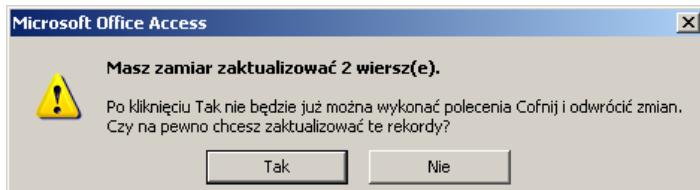
Kwerenda akcyjna jest wykonywana w momencie jej uruchomienia, generalnie silniki baz danych ostrzegają użytkownika przed uruchamianiem kwerend modyfikujących dane. W pokazanej sytuacji utworzona przed chwilą kwerenda akcyjna została zaznaczona i może być uruchomiona poleceniem *Otwórz* (lub poprzez podwójny klik).



Jej uruchomienie spowoduje wyświetlenie pokazanego niżej ostrzeżenia, wybór *Tak* uruchomi kwerendę, a wybór *Nie* anuluje operację otwarcia.



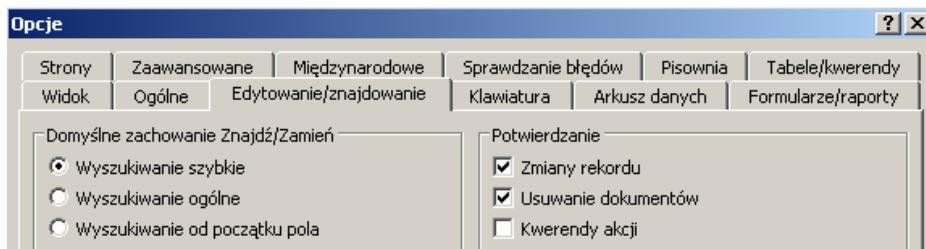
Po wyborze przycisku *Tak* system wyświetli kolejny komunikat informujący o liczbie rekordów, które zostaną zmodyfikowane i ponownie oczekujący na potwierdzenie ostatecznego wykonania aktualizacji.



Odpowiedź pozytywna na powyższy komunikat uruchamia kwerendę aktualizującą, a efekt jej działania pokazany jest poniżej.

TelefonyKlient : Tabela			
	id_t	id_k	NumerTelefonu
▶	1		1 (42) 234-45-34
▶	2		1 (42) 234-45-37
▶	3		2 (46) 846-45-67
*	Autonumerowanie)		

Komunikaty systemu ostrzegające o wykonaniu kwerendy akcyjnej są bardzo wygodne, jednak w niektórych przypadkach można je wyłączyć. Można tego dokonać poprzez modyfikację ustawień w menu *Narzędzia/Opcje* wyłączając pole wyboru *Kwerendy akcji* w grupie *Potwierdzenia*. Fragment tego okna dialogowego pokazany jest poniżej.



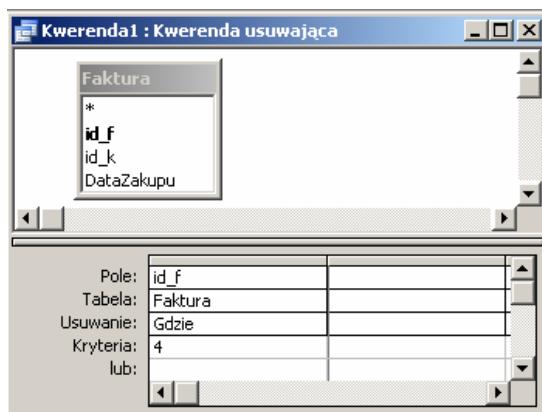
Kwerenda usuwająca

Zadaniem kwerendy tego typu jest usunięcie ze źródła danych wskazanych rekordów. Podobnie jak w przypadku kwerendy aktualizującej dokonane zmiany mają charakter trwały, stąd potrzebna jest rozproporcja przy stosowaniu takich kwerend.

Prześledzmy projektowanie i działanie kwerendy tego typu na przykładzie dotyczącym usunięcia z tabeli FakturaDetale i Faktura rekordów dotyczących faktury identyfikowanej id_f równym czterej.

Z uwagi na zdefiniowaną relację między tymi tabelami wystarczy usunięcie odpowiedniego rekordu z tabeli Faktura. Relacja między tymi dwoma tabelami miała aktywną opcję *Kaskadowo usuń rekordy pokrewne*, tym samym silnik bazy danych sam usunie powiązane rekordy z tabeli FakturaDetale.

Po otwarciu nowego projektu kwerendy dodajemy do niego tabelę Faktura i zmieniamy typ kwerendy na usuwającą, w efekcie w jej właściwościach pojawi się wiersz *Usuwanie*. Dodajemy do projektu pole id_f, a w kryteriach dla tego pola wpisujemy 4. Projekt takiej kwerendy pokazany jest poniżej, po jego zapisaniu (np. jako qUsun) można będzie wykonać tę kwerendę poprzez jej otwarcie.



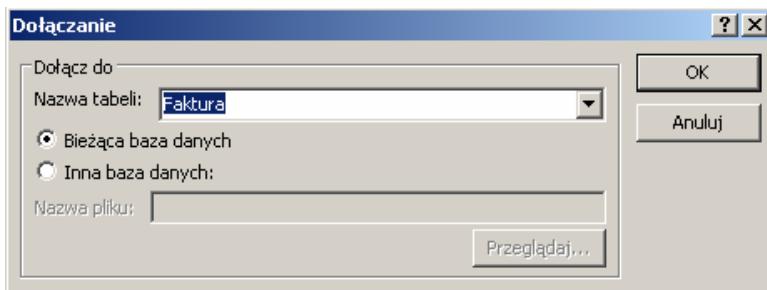
Efektem uruchomienia kwerendy qUsun będzie usunięcie z tabeli FakturaDetale tych rekordów, które dotyczyły faktury o identyfikatorze 4, a następnie z tabeli Faktura zostanie usunięty rekord opisujący tę fakturę.

Gdyby w relacji wiążącej obie tabele nie było aktywnej opcji *Kaskadowo usuń rekordy pokrewne*, to przy aktywnej opcji *Wymuszaj więzy integralności* uruchomienie kwerendy qUsun wywoła błąd wykonania programu. Poprawnie konieczne byłoby najpierw usunięcie rekordów z tabeli FakturaDetale (gdzie id_f = 4), a dopiero później usunięcie rekordu z tabeli Faktura (gdzie id_f = 4).

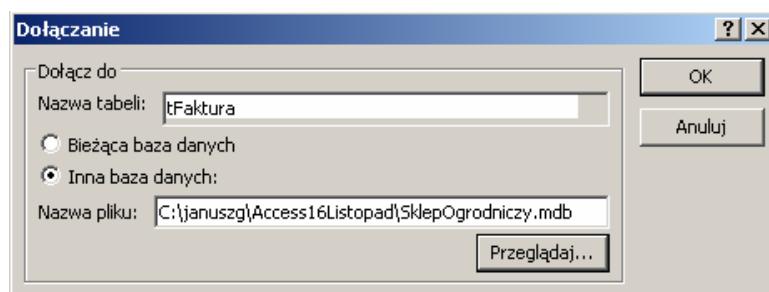
Kwerenda dołączająca

Zadaniem tego typu kwerend jest dołączenie do określonej tabeli nowego zestawu danych (rekordów). Dla zademonstrowania tego typu kwerend przygotujemy dwie kwerendy dołączające, ich zadaniem będzie „przywrócenie” stanu naszej bazy przed wywołaniem kwerendy qUsunFakture. Kwerenda ta usunęła z tabeli Faktura rekord odpowiadający fakturze o identyfikatorze 4, oraz odpowiadające im rekordy z tabeli FakturaDetaile. Przed uruchomieniem kwerendy usuwającej my jednak zrobiliśmy kopie obu tabel pod nazwami odpowiednio KopiaFaktura i KopiaFakturaDetaile. Wykorzystamy teraz te dwie tabele jako źródła danych dla przywrócenia usuniętych rekordów, z tym, że musimy to zrobić w dwóch krokach: najpierw przywróciemy usunięty rekord w tabeli Faktura, a następnie rekordy w tabeli FakturaDetaile.

Zaczynamy jak zwykle od otwarcia projektu nowej kwerendy i dodania do niej tabeli KopiaFaktury. Z menu *Kwerenda* (lub menu kontekstowego z prawego przycisku myszy) wybieramy jako typ kwerendę dołączającą. Efektem jest wyświetlenie pokazanego niżej okna dialogowego celem ustalenia nazwy tej tabeli, do której chcemy dołączyć dane. W przypadku, gdy tabela docelowa jest w bieżącej bazie danych, to jej nazwę wybieramy po prostu z rozwijanej listy, a taka sytuacja jest w naszym przykładzie.



Istnieje także możliwość dołączenia danych do tabeli, która znajduje się w innej niż bieżąca baza. Musimy wtedy podać pełną ścieżkę dostępu do pliku bazy danych, a nazwę tabeli musimy wpisać sami.



Po określaniu tabeli docelowej projekt kwerendy zostaje uzupełniony o wiersz *Dolaczanie do*, w tym miejscu będziemy określać nazwę pola, do którego mają być dołączone dane.

W pokazanym niżej projekcie ze źródła danych przenosimy do wiersza *Pole* kolejno pola *id_k*, *DataZakupu* i *id_f*. Z uwagi na zgodność nazw tych pól w źródle i tabeli docelowej Access automatycznie wstawia odpowiednie nazwy pól w wierszu *Dolaczane do*. W przypadku pierwszych dwóch pól jest to poprawne, ale pole *id_f* nie może być dołączone z dwóch powodów: po pierwsze nie jest to potrzebne, po drugie nie ma takiej możliwości, ponieważ typ tego pola to *autonumer*. Pole to jest nam potrzebne **jedynie** dla zdefiniowania kryterium określającego rekordy, które będą wykorzystane do dołączenia danych. Zaczynamy od usunięcia wpisu w wierszu *Dolaczanie do* tego pola, a następnie w wierszu *Kryteria* wpisujemy 4 (bo taki był numer usuniętej wcześniej faktury). Projekt tej kwerendy pokazany jest poniżej, możemy ją już zapisać w bazie danych, u nas otrzymuje ona nazwę *qDolaczFaktura*.



Po zapisaniu kwerendy w pliku bazy danych (po nadaniu jej nazwy) możemy ją otworzyć, co spowoduje jej wykonanie. W konsekwencji do tabeli *Faktura* zostanie dodany nowy rekord, jego pola *id_k* i *DataZakupu* będą miały dokładnie takie wartości, jak przed uruchomieniem kwerendy *qUsunFakture*. Pole *id_f* otrzymuje nową wartość, co jest konsekwencją tego, że jest zdefiniowane jako *autonumer*.

Poniżej widok tabeli *Faktura* po wykonaniu kwerendy *qUsunFakture*, czyli stan przed wykonaniem kwerendy *qDolaczFaktura* (nie ma rekordu identyfikowanego liczbą 4 w polu *id_f*) oraz widok tej samej tabeli po wykonaniu kwerendy *qDolaczFaktura*.

Po wykonaniu qUsunFakture.

	id_f	id_k	DataZakupu
▶	1	1	2005-11-02
▶	2	2	2005-11-05
▶	3	1	2005-11-07
*	(Autonumerowanie)		2005-11-25

Po wykonaniu qDolaczFakture.

	id_f	id_k	DataZakupu
▶	1	1	2005-11-02
▶	2	2	2005-11-05
▶	3	1	2005-11-07
▶	5	1	2005-11-12
*	(Autonumerowanie)		2005-11-25

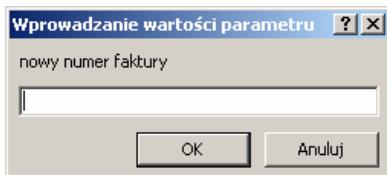
Rekord dołączony otrzymał jako identyfikator wartość 5, wykorzystamy teraz tę wartość dla przygotowania kolejnej kwerendy dołączającej, jej zadaniem będzie dołączenie danych opisujących detale faktury o pierwotnym numerze 4 (a obecnym 5). Projekt takiej kwerendy dołączającej dane do tabeli FakturaDetale pokazany jest poniżej.

qDolaczDetaleFaktury : Kwerenda dołączająca

KopiaFakturaDetale	
*	
id_f	
id_p	
Ilosc	
Pole:	id_p
Tabela:	KopiaFakturaDetale
Sortuj:	KopiaFakturaDetale
Dołączanie do:	id_p
Kryteria:	Ilosc
lub:	id_f

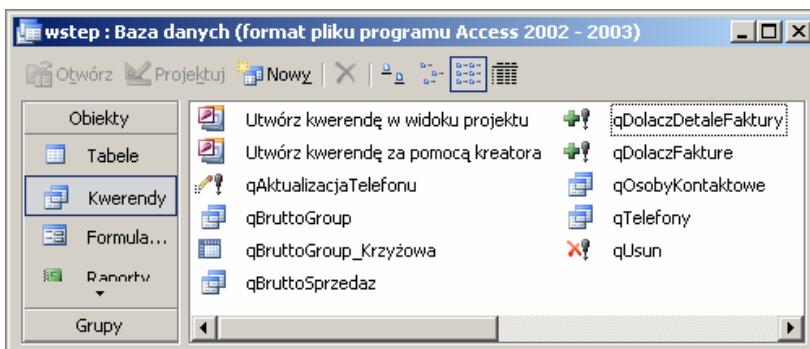
W projekcie tym pola **id_p** i **Ilosc** są dołączane do swoich odpowiedników w tabeli docelowej. Pole **id_f** jest wykorzystane do ograniczenia liczby rekordów do tych, które nas w tym momencie interesują. Problemem jest przekazanie nowej wartości pola

`id_f`, bowiem wartości tej nie mamy w źródle tej kwerendy. Na obecnym etapie „wtajemniczania” w MS Access w zasadzie nie mamy innego wyjścia jak określenie tej nowej wartości bezpośrednio w projekcie kwerendy stosując np. zapis `gg:5` lub zapytać o tę wartość użytkownika w momencie uruchamiania kwerendy. To ostatnie rozwiązanie zostało wykorzystane w pokazanym wyżej projekcie. Do listy pól kwerendy zostało dodane pole (wyliczane) o nazwie (aliasie) `gg`, której przypisana jest zmienna ujęta w nawiasy kwadratowe. Zmienna ta będzie pełnić rolę parametru dla pola `gg`, wartość tego parametru podamy w specjalnym oknie w momencie uruchomienia kwerendy.



Jeżeli podamy 5, to ta wartość zostanie przypisana do pola `gg`, a następnie dołączona do pola `id_f` tabeli docelowej. Wypróbowanie działania tak zaprojektowanej kwerendy pozostawiamy Czytelnikowi zastrzegając jednocześnie, że bardziej chodziło tu o pokazanie jak projektuje się kwerendę dołączającą, niż na zapewnieniu jej bezawaryjnej pracy.

Na zakończenie kwerend jeszcze widok okna bazy danych, każdy typ kwerendy ma swoją ikonę.



Poza wymienionymi typami kwerend w zastosowaniach praktycznych szczególnie istotną rolę odgrywają kwerendy wybierające z kryteriami zbudowanymi w sposób dynamiczny, z reguły będą to odwołania do obiektów (formantów) pewnego formularza. O takich kwerendach powiemy, że są to kwerendy parametryczne.

Wiele przykładów kwerend parametrycznych znajdzie Czytelnik w kolejnych rozdziałach poświęconych omówieniu dwóch kompletnych aplikacji.

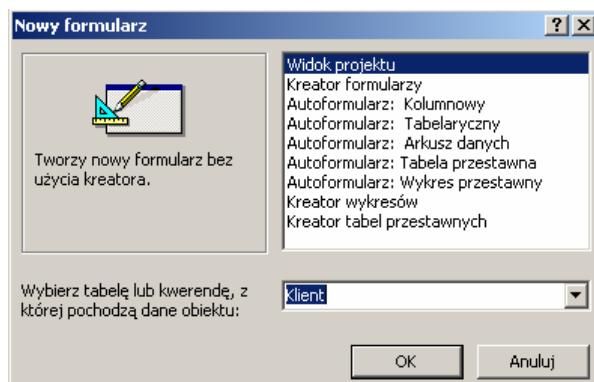
2.3.5. Formularze

Formularze to jeden z najważniejszych obiektów bazy danych w MS Access czy aplikacji bazodanowej. Bez formularzy relacyjne bazy danych w zasadzie nie istnieją! Odpowiednio zaprojektowane formularze ułatwiają przeglądanie, edycję i wprowadzanie nowych danych. W kolejnych przykładach przedstawimy sposoby projektowania formularzy, zarówno prostych jak i złożonych.

Formularze proste

W bazie *Wstęp.mdb* zaprojektowaliśmy kilka wzajemnie powiązanych tabel, w tym tabelle do przechowywania informacji o klientach, numerach ich telefonów i danych osób kontaktowych oraz tabelę przechowującą nazwy miejscowości. Przygotujemy teraz formularz przeznaczony do przeglądania, edycji i wprowadzania danych do tabeli *Klienci*.

Projektowanie tego formularza zaczynamy od przejścia do obiektu *Formularze* w oknie bazy danych, a następnie wywołujemy polecenie *Nowy*. W otwartym oknie dialogowym możemy wybrać rodzaj projektu formularza oraz wskazujemy na źródło danych skojarzonych z tym formularzem (tabelę lub kwerendę). W pokazanej niżej sytuacji wybrana jest opcja *Widok projektu*, a jako źródło danych tabela *Klient*.



MS Access ma stosunkowo dobrze rozbudowane kreatory do tworzenia formularzy, jednak wybrana opcja zapewnia największe możliwości. Po akceptacji przycisku OK zostanie otwarte okno projektu nowego formularza.

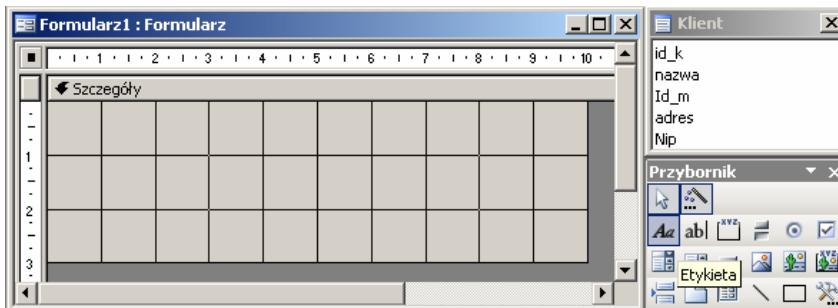
Formularz to nic innego jak okno, ma swój pasek tytułu z tytułem i przyciskami sterującymi zachowaniem się okna, ma krawędzie służące między innymi do zmiany rozmiaru okna, ma swoje paski poziomego i pionowego przewijania zawartości okna.

Wewnątrz okna jest przestrzeń do umieszczania różnego rodzaju kontrolek, takich jak etykiety, pola tekstowe, przyciski opcji, pola wyboru, ramki, listy rozwijane, listy czy

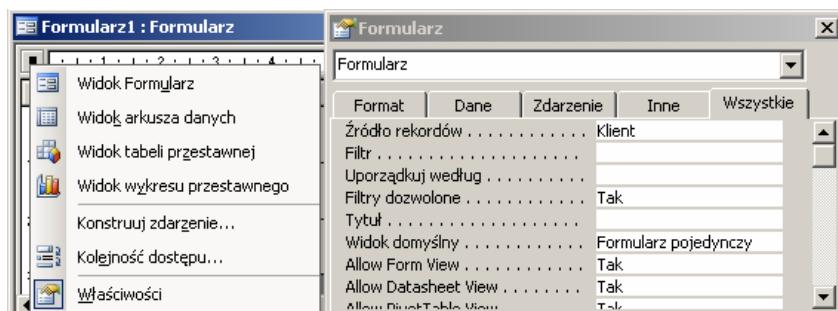
przyciski polecień. Kontrolki te będziemy nazywać **formantami**, część z nich będzie związana z polami źródła danych formularza, część nie.

Ta wewnętrzna część formularza może być jeszcze podzielona na części, na tzw. **sekcje**. Jedną z nich jest sekcja *Szczegóły*, to głównie w niej będziemy umieszczać formanty związane. W momencie otwarcia projektu nowego formularza zawiera on wyłącznie sekcję *Szczegóły*. Jeżeli zachodzi taka potrzeba, to wykorzystując menu *Widok* można włączyć do projektu także takie sekcje jak *Nagłówek/Stopka* formularza czy *Nagłówek/Stopka* strony.

Poniżej pokazany jest widok projektu naszego formularza, widoczna jest sekcja *Szczegóły*, pokazane jest okno z listą pól w źródle danych skojarzonym z projektem oraz okno, tzw. przybornik, ze standardową listą kontrolek. Jedna z nich jest w pokazanym momencie wybrana (etykieta).



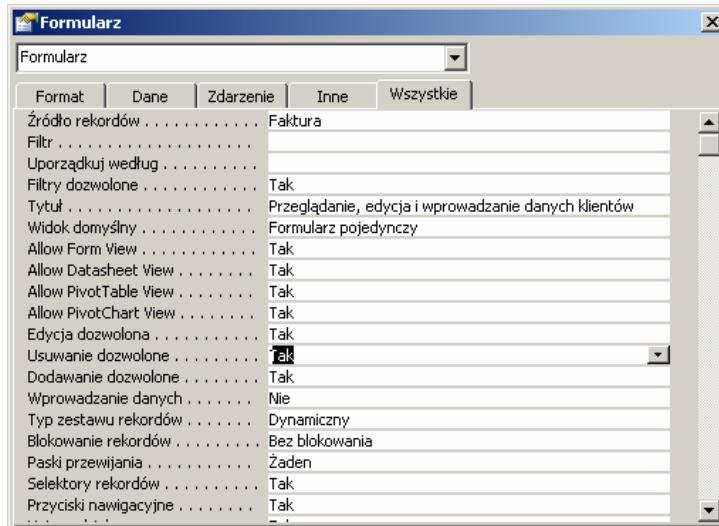
Formularz z punktu widzenia Windows jest obiektem, tym samym ma szereg właściwości opisujących ten formularz. Przycisk umieszczony na lewo od linijki poziomej służy do zaznaczenia (przydzielenia fokusu) formularzowi jako takiemu. Pozwala także na wywołanie menu kontekstowego dla formularza. W pokazanej niżej sytuacji menu kontekstowe jest rozwinięte, jedną z dostępnych opcji są *Właściwości*, okno właściwości formularza zostało także otwarte (precyzyjnie: dla potrzeb tej książki było odwrotnie, przy otwartym oknie właściwości otworzyliśmy menu kontekstowe).



Okno właściwości formularza i jego obiektów (sekcji, formantów) będzie bardzo przydatne w trakcie projektowania formularza. Okno właściwości możemy wywołać albo z menu kontekstowego obiektu, albo poprzez klik jego przycisku w pasku narzędziowym MS Access. Obojętnie jak je wywołamy, to zawsze będzie pokazywać właściwości zaznaczonego obiektu.

Zawartość okna właściwości jest zmienna, zależy po prostu od rodzaju obiektu, którego dotyczy. W większości przypadków lista dostępnych właściwości jest stosunkowo duża, stąd zastosowanie zakładek grupujących właściwości w oddzielne kategorie.

W pokazanej sytuacji w oknie właściwości formularza widoczna jest jedna z ważniejszych właściwości, *Źródło rekordów* (ang. *Record source*). Właściwość ta może być zdefiniowana w momencie projektowania formularza (ang. *design time*) jak i programowo w trakcie jego otwierania (ang. *run time*). Generalnie będziemy mieli takie właściwości, które mogą być ustawione wyłącznie w czasie projektowania, będą też takie, którym można będzie przypisać wartość jedynie programowo i będzie także grupa właściwości, które mogą być zdefiniowane w obu momentach życia formularza.



W pokazanym wyżej oknie właściwości formularza zdefiniowano jego tytuł przypisując właściwości *Tytuł* (ang. *Caption*) odpowiedni tekst. Każdy formularz może być wyświetlony w kilku widokach, domyślnym jest *Formularz pojedynczy*. W tym widoku formularz pokazuje pojedynczy rekord (wszystkie lub część pól). W przypadku widoku typu *Arkusz danych* formularz wyświetla jednocześnie więcej niż jeden rekord. Widok *Formularze ciągłe* jest w pewnym sensie połączeniem obu wcześniejszych widoków, widzimy wiele rekordów, ale w takim układzie jak w przypadku formularza pojedynczego.

Wersja MS Access 2003 dopuszcza jeszcze kilka innych widoków formularza, w tym bardzo przydatny widok typu Tabela przestawna.

Z uwagi na zamierzoną funkcjonalność formularza trzy ważne właściwości zostały ustawione na Tak, dotyczy to właściwości *Edycja dozwolona* (ang. *AllowEdits*), *Usuwanie dozwolone* (ang. *AllowDeletions*) i *Dodawanie dozwolone* (ang. *AllowAdditions*).

Właściwość *Wprowadzanie danych* (ang. *AllowEntry*) musi być ustawiona na Nie, jeżeli chcemy mieć możliwość przeglądania danych wcześniej wprowadzonych do tabeli. Inne właściwości formularza decydują o tym, czy np. mają być wyświetlane paski przewijania czy nie, ewentualnie który z nich. Możemy także decydować o tym, czy w prawym górnym narożniku będzie komplet przycisków sterujących oknem, czy tylko przycisk zamknij.

W przypadku formantów (kontrolek) do ważnych właściwości można zaliczyć te, które określają położenie formantu w danej sekcji, jego wysokość czy szerokość. Równie ważna jest możliwość pokazywania lub nie danego formantu, przy czym właściwość ta, jeżeli ma mieć sens, powinna być modyfikowana głównie programowo wg zasady: formant potrzebny jest pokazywany, formant niepotrzebny jest ukrywamy.

Oddzielna grupa właściwości to reakcja na zdarzenia, jakie mogą dotyczyć danego obiektu. W dalszych rozdziałach tej książki poświęconych omówieniu przykładowych aplikacji znajdziemy bardzo wiele zastosowań właściwości z tej grupy.

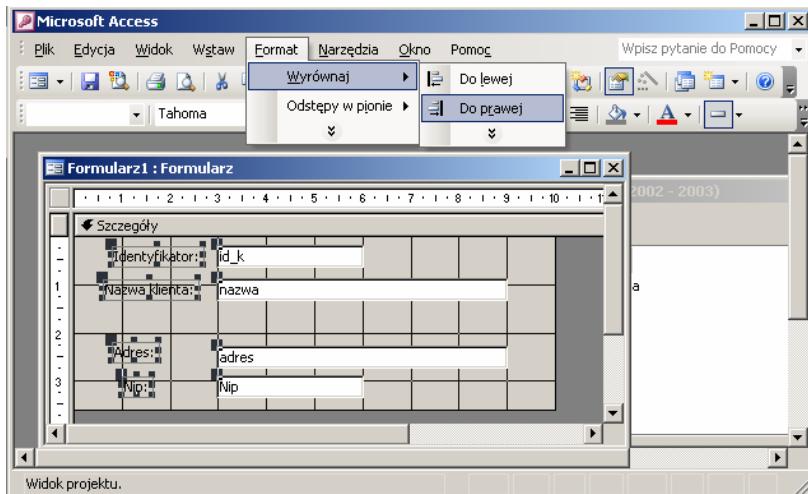
Kolejny krok projektowania formularza polega na umieszczeniu w określonych miejscach projektu formularza kontrolek odwołujących się do poszczególnych pól źródła danych. W naszym przypadku w źródle występują takie pola jak `id_k`, `nazwisko`, `id_m`, `adres` oraz `Nip`.

Pola te, z wyjątkiem `id_m` mają taki sam charakter w tym sensie, że informacja przechowywana w tych polach jest jednoznacznie czytelna (zrozumiała) dla użytkownika, tym samym do ich zaprezentowania w formularzu możemy wykorzystać kontrolki typu **pola tekstowego** (ang. *TextBox*).

W przypadku pola `id_m` jest zupełnie inna sytuacja, pole to zawiera identyfikator miejscowości z tabeli `Miejscowosc` i w postaci liczby jest absolutnie nieczytelne. Oznacza to, że do jego prezentacji w formularzu nie możemy wykorzystać pola tekstowego. Rozwiązaniem będzie wykorzystanie kontroli typu **lista rozwijana** (ang. *ComboBox*), to jest kontrolka dedykowana do rozwiązywania takiej właśnie sytuacji. Jej praca polega na powiązaniu (tłumaczeniu) zakodowanej informacji z pola `id_m` na jej zrozumiałą odpowiednik z tabeli `Miejscowosc`.

Pola tekstowe możemy umieścić w formularzu poprzez ich zwykłe przeciągnięcie z listy pól źródła danych i ustawienie w sekcji *Szczegóły* we właściwym miejscu. Musimy oczywiście skorygować - jeżeli zachodzi taka potrzeba - opisy tych pól w powiązanych z nimi kontrolkach typu **etykieta** (ang. *Label*).

W pokazanej niżej sytuacji do sekcji *Szczegóły* dodane zostały potrzebne pola tekstowe, wstępnie ustawione w miejscach przewidzianych dla nich, zostały zmienione tytuły etykiet związań z tymi polami, zmieniona została także szerokość niektórych pól tekstowych. Etykiety opisujące pola tekstowe zostały wspólnie zaznaczone w tym celu, aby korzystając z menu *Format/Wyrównaj/Do prawej* wyrównać je w sposób jednolity względem tej z nich, która jest maksymalnie wysunięta w prawo.

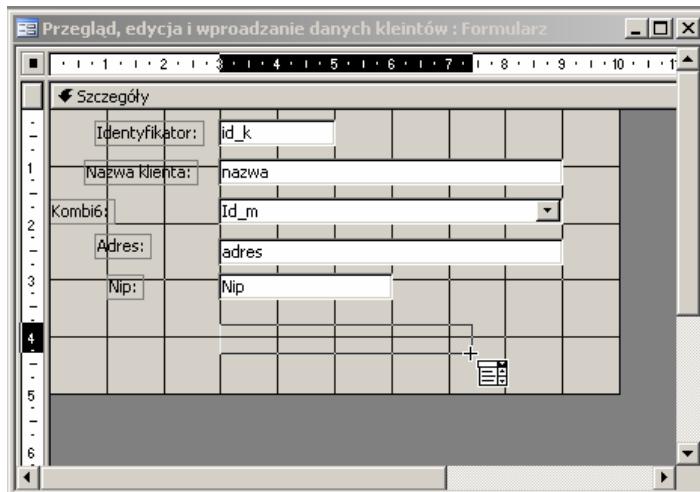


Dokładnie w podobny sposób będzie można wyrównać pola tekstowe wybierając tym razem opcję *Wyrównaj/Do Lewej*.

Pole *id_k* jest zdefiniowane w tabeli *Klient* jako *autonumer*, tym samym nie może być zmieniane przez użytkownika. Osiągniemy taką funkcjonalność tego pola ustawiając odpowiednio dwie jego właściwości: *Włączony* (ang. *Enabled*) na *Nie* oraz *Zablokowany* (ang. *Locked*) na *Tak*.

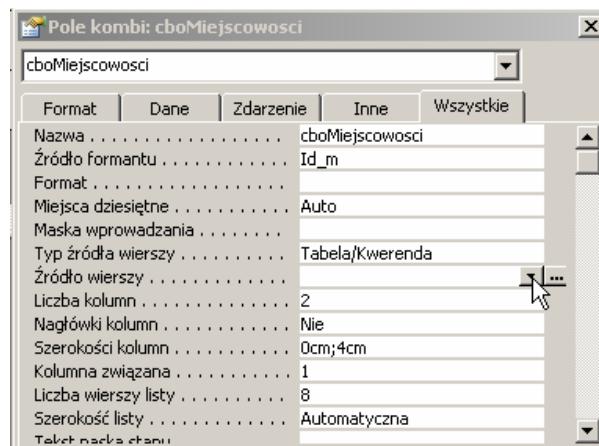


Miedzy polami tekstowymi *nazwa* i *adres* zostawiliśmy miejsce na kontrolkę typu rozwijanej listy. Selekcjonujemy tę kontrolkę w przyborniku i umieszczamy ją w tym wolnym miejscu poprzez po prostu jej narysowanie (nadając jej odpowiedni rozmiar). Poniżej widok projektu formularza w trakcie „rysowania” kontrolki typu pole kombi.



Po zwolnieniu myszy uruchamia się kreator pól kombi, który poprzez kolejne okna dialogowe pozwala na stosunkowo łatwe skonfigurowanie tej kontrolki. Myślę, że możemy Czytelnikowi pozostawić prześledzenie pracy tego kreatora jako ćwiczenie, a my skonfigurujemy tę kontrolkę samodzielnie, bez użycia kreatora.

Zaczynamy oczywiście od anulowania kreatora i otwarcia okna właściwości wstawionego pola kombi. Pierwszą właściwość, którą zmienimy to *Nazwa* (ang. *Name*), tu proponujemy wpisać nazwę *cboMiejscowosci*, gdzie prefix *cbo* wskazuje na typ kontrolki. Nazwa kontroli ma szczególne znaczenie wtedy, kiedy będziemy chcieli się do niej odwołać z poziomu kodu VBA.

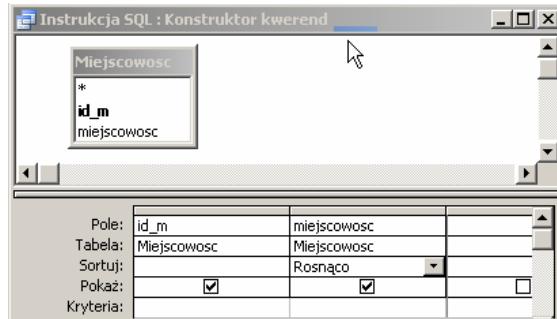


Musimy teraz określić właściwość *Źródło formantu* (ang. *ControlSource*), bezpośrednio po wstawieniu pola kombi do projektu formularza właściwość ta jest nieokreślona i musimy wskazać pole źródła danych, z którego kontrolka ta będzie pobierała dane czy też do którego będzie zwracać dane. W naszym przypadku jest to oczywiście pole `Id_m`.

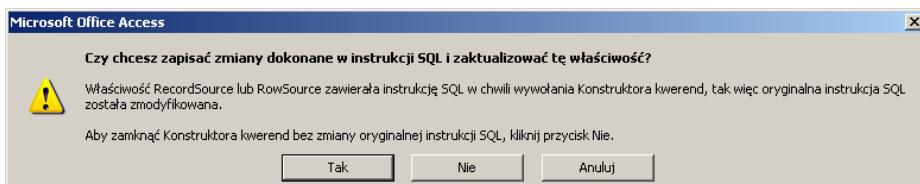
Kolejna ważna właściwość to *Typ źródła wierszy* (ang. *RowSourceType*), domyślnie jest tu ustawiana wartość *Tabela/Kwerenda* i tak ją pozostawimy.

Kolejny krok to określenie *źródła wierszy* (ang. *RowSource*) dla informacji wypełniającej listę pola kombi. Po ustawieniu kurSORA w polu tej właściwości możemy wybrać istniejące źródło danych (tabelę lub kwerendę), możemy także wpisać instrukcję zapytania w języku SQL. To ostatnie rozwiązanie jest zdecydowanie najlepsze, MS Access ułatwia budowanie instrukcji SQL poprzez udostępnienie graficznego kreatora, wystarczy w tym celu kliknąć przycisk polecenia oznaczony trzema kropkami.

Graficzny kreator instrukcji SQL pracuje praktycznie tak samo, jak kreator zapytań: zaczynamy od dodania potrzebnych tabel, a następnie wybieramy (lub budujemy) te pola, które chcemy wykorzystać w polu kombi. W naszym przypadku będą to pola `id_m` i `miejscowosc` z tabeli *Miejscowosc*. Dodatkowo sortujemy zwracane rekordy po nazwie miejscowości.



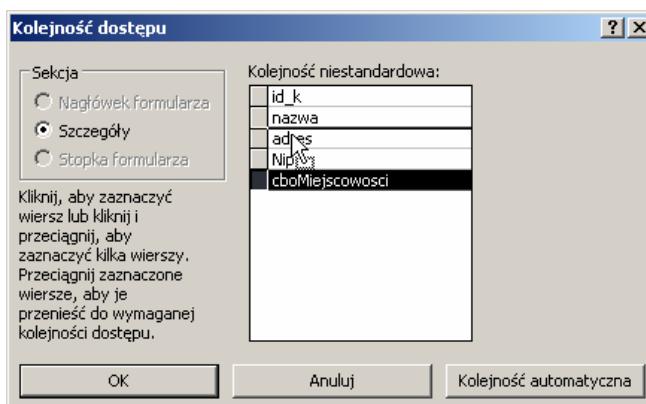
Po zamknięciu tego kreatora MS Access zaprośca, co ma zrobić z utworzoną instrukcją SQL wyświetlając pokazany niżej komunikat.



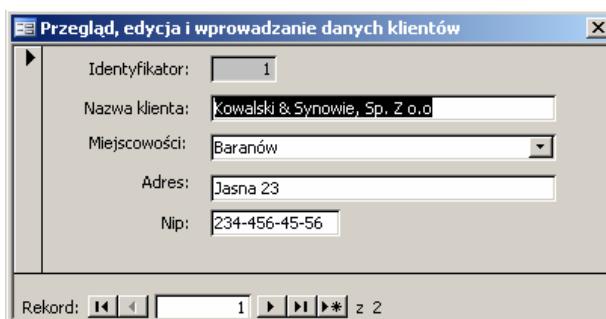
Odpowiedź pozytywna spowoduje przypisanie właściwości *Źródło wierszy* instrukcji SQL: `SELECT Miejscowosc.id_m, Miejscowosc.miejscowosc FROM Miejscowosc ORDER BY Miejscowosc.miejscowosc;`

Nasz formularz jest już w zasadzie gotowy, pozostało jeszcze sprawdzenie formowania kontrolek w aspekcie ich porządkowego ustawienia w formularzu, odpowiednich etykiet, szerokości i wysokości tych kontrolek. Możemy przykładowo zmienić tło pola tekstuowego `id_k` dla lepszego uwypuklenia, że pole to jest zablokowane. Powinniśmy także zweryfikować **kolejność dostępu** do kontrolek przy użyciu klawisza Tab. Standardowo kontrolki są dostępne wg kolejności ich umieszczania w projekcie, w naszym przypadku zasada ta spowoduje nienaturalną kolejność, bowiem pole kombi określające miejscowości będzie dostępne na końcu listy kontrolek, a powinno być po polu `nazwa`.

W menu *Widok*, także w menu kontekstowym formularza znajdziemy polecenie *Kolejność dostępu*, jego uruchomienie spowoduje wyświetlenie okna dialogowego pozwalającego na modyfikację kolejności dostępu. Poniżej widok tego okna w trakcie przesuwania pola `cboMiejscowosc` przed pole `adres`.



Po tej ostatniej modyfikacji formularz jest gotowy do pracy, ale wcześniej powinniśmy zapisać go w pliku bazy (u nas pod nazwą `frmKlient`, gdzie prefix `frm` będzie wskazywał na formularz). Otwarty formularz powinien wyglądać tak, jak pokazany niżej.



Tak przygotowany formularz pozwala na przeglądanie i edycję wprowadzonych do tabeli `Klient` informacji z pewnym ograniczeniem co do pola `id_m` opisującego miejscowości. To ograniczenie wynika z faktu, że z poziomu formularza `frmKlient` mamy dostęp **wyłącznie** do tych miejscowości, które wcześniej zostały wprowadzone do tabeli `Miejscowosc`. Eleganckim rozwiązaniem byłoby stworzenie takiej możliwości, aby z poziomu formularza `frmKlient` można było otworzyć formularz obsługujący tabelę `Miejscowosc`, dodać brakującą miejscowości, a po powrocie do formularza `frmKlient` znaleźć ją już na liście pola kombi `cboMiejscowosc`. Jest to oczywiście możliwe, ale wymaga zbudowania kliku procedur w języku VBA. W aplikacjach omówionych w dalszych rozdziałach jest kilka przykładów formularzy zbudowanych wg opisanej wyżej zasady.

Formularz `frmKlient` w obecnej postaci ma jeszcze jedną istotną wadę: nie pozwala na uzyskanie informacji o telefonicznych numerach kontaktowych ani informacji o osobach, które mogą odebrać podany telefon. Spróbujemy temu zaradzić poprzez zbudowanie takiego formularza, który da dostęp przynajmniej do numerów telefonów kontaktowych.

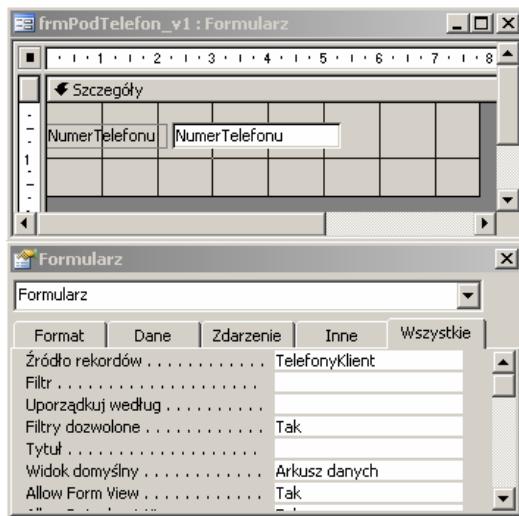
Formularze złożone

Generalna koncepcja budowy formularza złożonego polega na tym, aby w jednym formularzu pokazać informacje pochodzące z dwóch źródeł danych połączonych relacją *jeden-do-wielu*. Formularz główny pokazuje wtedy informacje po stronie „jeden” tej relacji, a podformularz informacje po stronie „wiele”. Oba formularza połączone są z sobą poprzez pole wiążące, dzięki czemu każda zmiana rekordu w formularzu głównym powoduje aktualizację danych w podformularzu. MS Access pozwala na zagnieżdżenie do dwóch poziomów podformularzy.

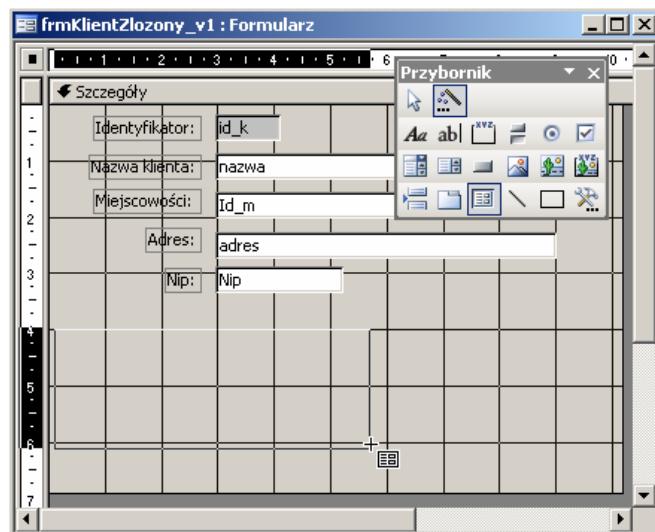
W naszym przykładzie zbudujemy nowy formularz obsługujący tabelę `Klient` w dwóch wersjach: pierwszy z nich będzie stosunkowo prostym formularzem złożonym dającym dostęp do numerów telefonów kontaktowych (jeden poziom zagnieżdżenia), drugi będzie znacznie bardziej skomplikowany z uwagi na relacje wiążące tabele `Klient`, `TelefonyKlient` i `OsobyKontaktoweKlient` (dwa poziomy zagnieżdżenia).

Budowanie pierwszego ze złożonych formularzy zaczniemy od utworzenia kopii formularza `frmKlient`, której nadamy nazwę np. `frmKlientZlozony_v1`.

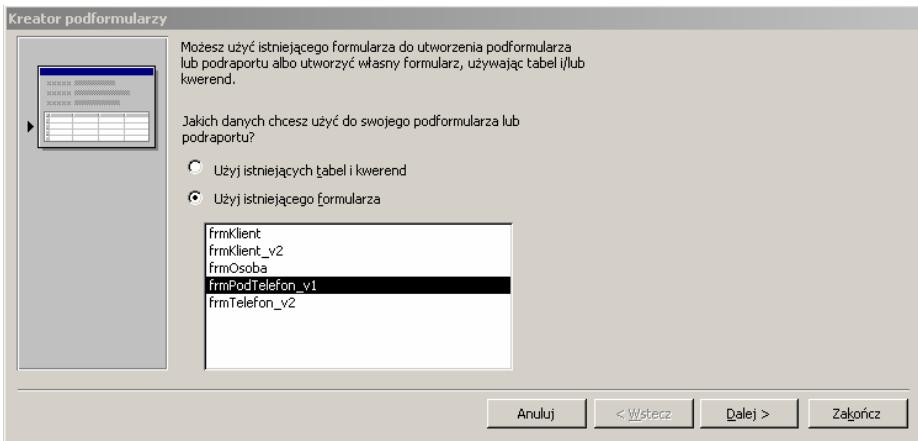
Do projektu tego ostatniego formularza dodamy formularz pokazujący dane z tabeli `TelefonyKlient`, formularz ten zaprojektujemy do pracy w widoku arkusza danych. Poniżej widok projektu tego formularza, w sekcji `Szczegóły` zostało umieszczone tylko pole `NumerTelefonu` (to wystarczy, ponieważ ten formularz ma pracować jako **podformularz**, informacja dla pola `id_k` będzie pobierana z formularza głównego). Formularz ten został zapisany w kolekcji formularzy pod nazwą `frmPodTelefon_v1`, dodatkowy prefiks `Pod` sygnalizuje podformularz, a `v1` oznacza wersję (będzie jeszcze wersja 2 dla drugiej wersji formularza złożonego obsługującego dane klientów).



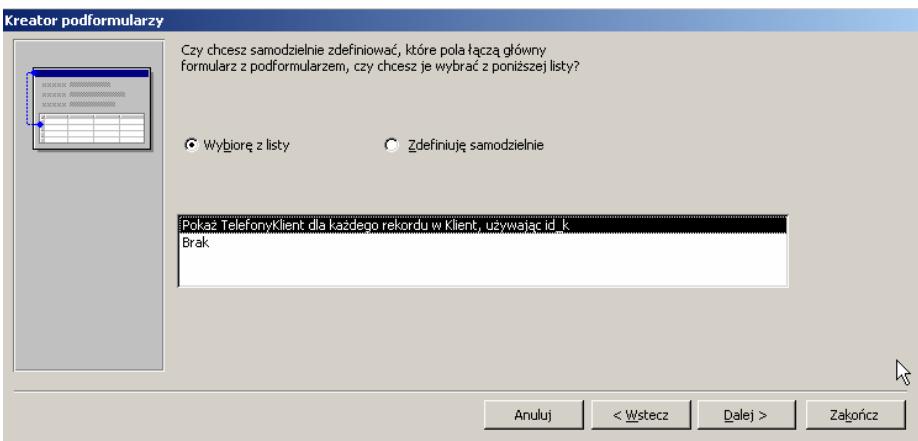
Mając gotowy podformularz otwieramy w widoku projektu formularz główny, czyli frmKlientZlozony_v1, po to, aby poniżej dotychczasowych kontrolek umieścić obiekt podformularza. W przyborniku znajdziemy odpowiednią kontrolę, aktywujemy ją, a następnie rysujemy podformularz w projekcie formularza głównego. Moment umieszczania podformularza pokazany jest poniżej.



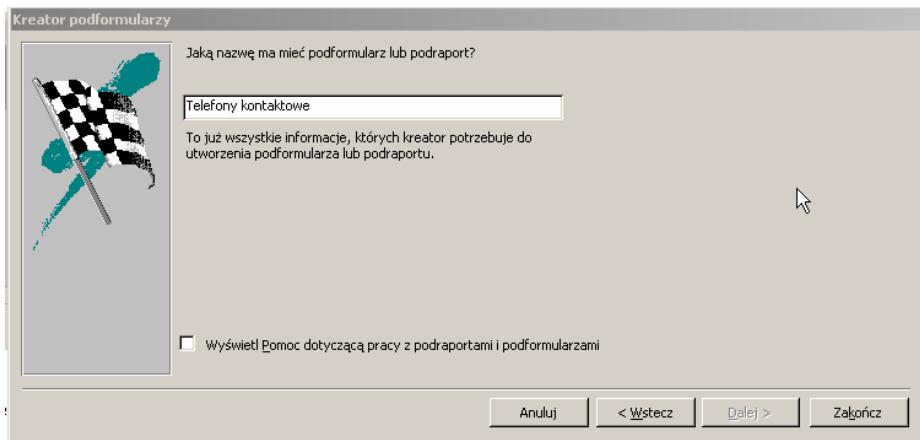
Po zwolnieniu myszy MS Access wyświetla okno kreatora formularzy, w kilku kolejnych oknach podformularz zostanie odpowiednio skonfigurowany. W poniżej sytuacji został wskazany formularz, który będzie pełnił rolę podformularza.



W kolejnym oknie kreator proponuje nam pole wiążące formularz główny z podformularzem.

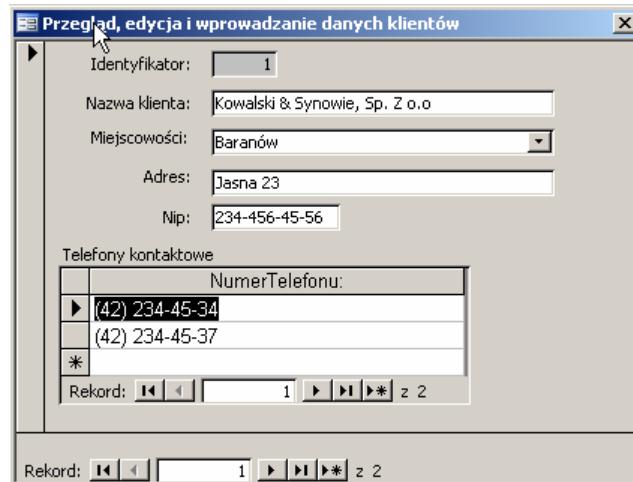


Przycisk polecenia *Dalej >* przenosi nas do ostatniego już okna dialogowego kreatora podformularzy, w oknie tym określamy etykietę dla tworzonego podformularza.



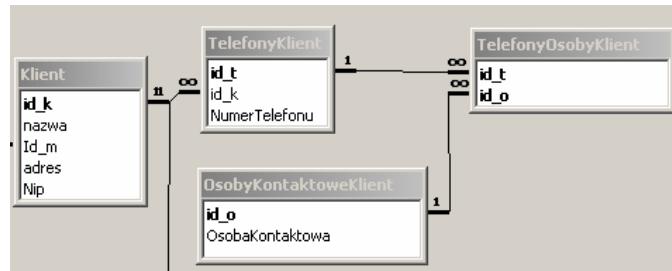
Po akceptacji przycisku *Zakończ* podformularz jest już w zasadzie gotowy, pozostało jeszcze nadać mu odpowiednie rozmiary i nasz pierwszy formularz złożony jest już gotowy do pracy.

Poniżej widok tak przygotowanego formularza złożonego, w porównaniu z formularzem frmKlient ma on znakomicie większą funkcjonalność, ponieważ daje dostęp nie tylko do informacji z tabeli Klient, ale także do tabeli TelefonyKlient włącznie z dodawaniem nowych rekordów do tej ostatniej tabeli.



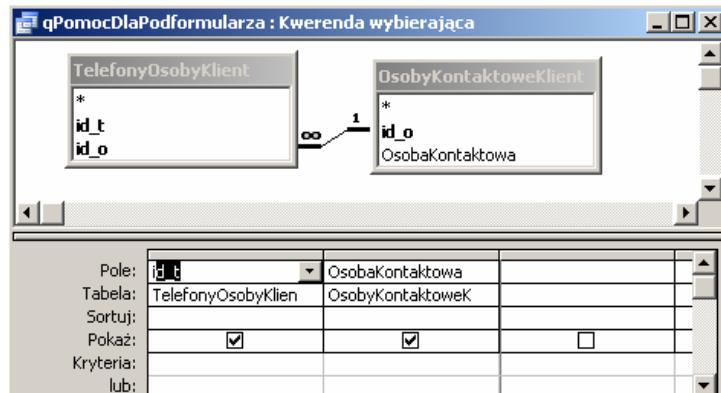
Jak wcześniej zwracaliśmy uwagę, że formularz złożony dający dostęp do danych klientów powinien umożliwić nie tylko dostęp do numerów telefonów kontaktowych, ale

także do informacji, kto będzie dostępny pod danym numerem telefonu. Zbudowanie takiego formularza złożonego w tym miejscu tej książki (przed wprowadzeniem programowania w VBA) jest stosunkowo skomplikowane. Komplikacja ta jest konsekwencją relacji wiążących potrzebne tabele, poniżej fragment okna relacji pokazujący te połączenia.

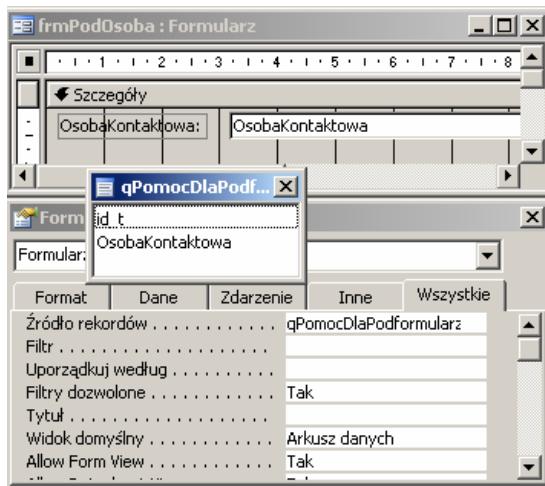


Jak widzimy tabela **OsobyKontaktoweKlient** nie jest bezpośrednio powiązana z tabelą **Klient** (tak jak **TelefonyKlient**), tym samym **nie można** umieścić w projekcie formularza **frmPodKlient_v1** jeszcze jednego podformularza pokazującego osoby kontaktowe, bo brak jest pola bezpośrednio wiążącego te dwie tabele.

Jedyna metoda, to powiązanie numeru telefonu z osobą kontaktową, czyli zbudowanie takiego formularza złożonego, gdzie po stronie „jeden” będzie występowała tabela **TelefonyKlient**, a po stronie „wiele” takie źródło, gdzie poza osobą kontaktową będzie także identyfikator telefonu. W tej chwili takiego źródła nie mamy, ale możemy je utworzyć jako kwerendę wybierającą wykorzystując tabele **TelefonyOsobyKlient** i **OsobyKontaktoweKlient**.

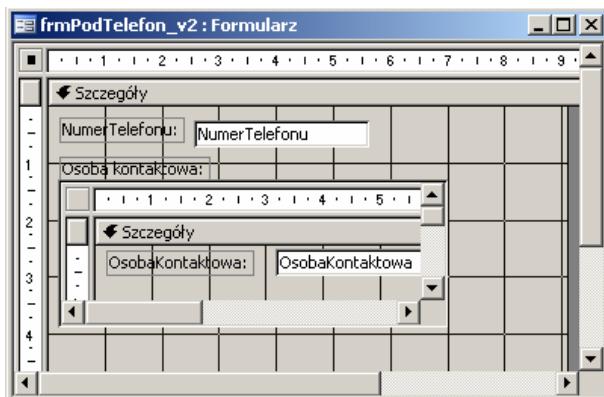


Kwerenda o nazwie **qPomocDlaPodformularza** zostanie teraz wykorzystana dla zbudowania formularza **frmPodOsoby**, formularz ten będzie pokazywał dane w widoku arkusza danych.

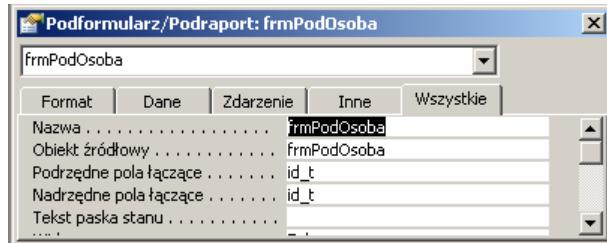


Kolejny krok to utworzenie nowego formularza dla zaprezentowania telefonów kierunkowych, tym razem formularz ten będzie zbudowany jako formularz złożony, w formularzu głównym będziemy pokazywać numery telefonów, a w podformularzu (opartym na zbudowanym przed chwilą frmPodOsoby) nazwiska osób kontaktowych dostępnych pod danym telefonom.

Poniżej widok projektu takiego formularza złożonego, dla potrzeb późniejszego jego wykorzystania został on zapisany pod nazwą frmPodTelefon_v2.



Z uwagi na konieczność umieszczenia w projekcie formularza podformularza jego widok domyślny musi być ustawiony na Formularz pojedynczy. Poniżej jeszcze okno właściwości podformularza wykorzystanego w tym projekcie.



Pozostało nam jeszcze zbudowanie nowej wersji formularza złożonego dla tabeli Klient, tym razem jako podformularz wykorzystamy frmPodTelefon_v2, który jak wiemy sam jest także formularzem złożonym.

Widok tak przygotowanego formularza pokazany jest poniżej. W porównaniu z wersją frmKlientZlozony_v1 mamy zwiększenie funkcjonalności poprzez pokazanie dla danego klienta wszystkich istotnych informacji z nim związanych.

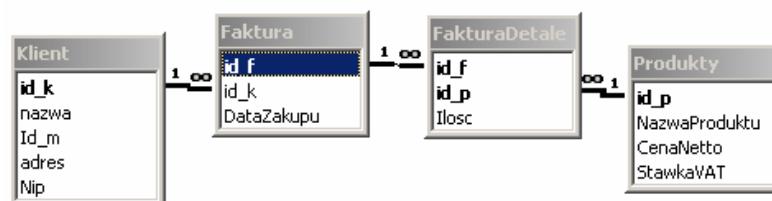
Formularz ten nie jest jednak pozbawiony jednej wady: z jego pomocą możemy wprowadzić nową osobę kontaktową, ale dodanie nowego rekordu z poziomu podformularza frmPodOsoba nie spowoduje dodania odpowiedniego rekordu do tabeli TelefonyOsobyKlient „kojarzącego” osobę z numerem telefonu. Bez użycia procedur napisanych w języku programowania VBA nie jest to po prostu możliwe.

Formularz frmKlientZlozony_v2, po usunięciu wspomnianego wyżej ograniczenia oraz po stworzeniu możliwości dodania brakującej miejscowości będzie w pełni funkcjonalny w zakresie przeglądania, edycji jak i dodawania kompletnych danych dla nowego klienta.

Obliczenia w formularzach

Jak wiadomo z rozdziału poświęconego normalizacji tabele nie powinny zawierać żadnych pól, których wartości mogą być wyznaczone na podstawie wartości innych pól zawierających informacje elementarne. Wiemy, że jedną z metod uzyskania takich przetworzonych informacji jest zbudowanie kwerendy zawierającej pola wyliczane. Inną metodą jest budowanie pól wyliczanych w formularzach. W tym rozdziale prześledzimy tę problematykę na przykładzie formularza złożonego, którego zadaniem będzie rejestracja nowej sprzedaży.

Na etapie projektowania tabel w naszej przykładowej bazie danych (`Wstep.mdb`) zostały zaprojektowane dwie tabele przeznaczone do rejestrowania sprzedaży: tabela `Faktura` odnotowuje numer faktury oraz datę sprzedaży i identyfikator klienta, z kolei tabela `FakturaDetale` rejestruje szczegóły transakcji obejmujące identyfikator faktury, identyfikator produktu i sprzedaną ilość produktu. Te dwie tabele współpracują z tabelami `Klient` i `Produkty`. Poniżej fragment okna relacji dla przypomnienia wzajemnych powiązań tych czterech tabel.



Mamy zamiar przygotować formularz złożony pozwalający w łatwy sposób na obsługę pojedynczej transakcji. Dobrze byłoby, aby w trakcie realizacji transakcji zarówno sprzedający jak i kupujący mieli wgląd w listę wybranych produktów, ich cenę jednostkową, stawkę podatku VAT, wartość brutto danego produktu jak i łączną wartość całej transakcji (inaczej mówiąc kwota do zapłaty).

Jak widzimy, w żadnej z pokazanych tabel nie ma kompletu wymienionych wyżej danych, będziemy je więc musieli dynamicznie wyliczać, przy czym ich znaczenie i czas trwania będą istotne wyłącznie w trakcie samej transakcji, po jej zawarciu (i wystawieniu stosownych dokumentów) większość tych danych przestanie istnieć.

W trakcie projektowania tego formularza, przy niemożności (jeszcze) korzystania z procedur VBA będziemy musieli skorzystać z innego obiektu bazy danych, jakim są **makropolecenia**. Konieczne będzie także przygotowanie kilku dedykowanych kwerend, w tym także kwerendy akcyjne.

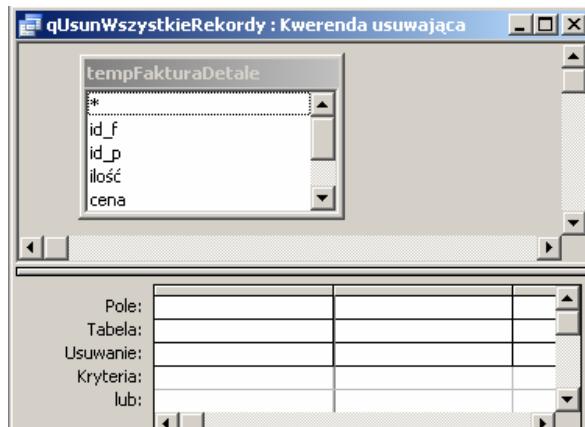
Pomysł zbudowania formularza do obsługi sprzedaży oprzemy na wykorzystaniu pomocniczej tabeli o nazwie `tempFakturaDetale`, tabela ta będzie zawierać wszystkie potrzebne dane elementarne jak i pola wyliczane dla potrzeb faktury. W momencie

zakończenia rejestracji sprzedaży dane z tabeli pomocniczej zostaną dołączone do tabeli FakturaDetale.

Nazwy pól tabeli tempFakturaDetale wraz z ich typami danych pokazane są poniżej jako fragment okna projektu tej tabeli. Warto zwrócić uwagę, że w tej tabeli nie został zdefiniowany klucz. Z uwagi na jej charakter tymczasowy nie było to po prostu konieczne.

Nazwa pola	Typ danych	Opis
id_f	Liczba	identyfikator faktury, liczba całkowita długa
id_p	Liczba	identyfikator produktu, liczba całkowita długa
ilość	Liczba	ilość produktu, liczba, pojedyncza precyzja
cena	Walutowy	cena jednostkowa, walutowy
stawkaVat	Liczba	% VAT, liczba, pojedyncza精度, format procentowy
kwotaVat	Walutowy	kwota podatku Vat, walutowy
brutto	Walutowy	wartość brutto danego produktu, walutowy

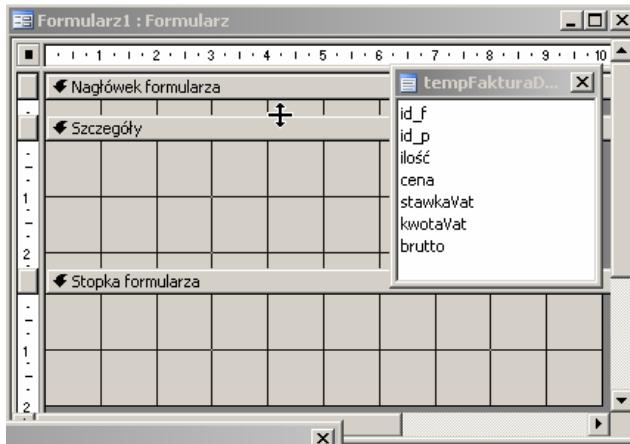
W momencie otwierania formularza obsługującego sprzedaż dane z tej tabeli muszą być usunięte, zrobimy to poprzez wywołanie kwerendy qUsunWszystkieRekordy. Jest to kwerenda usuwająca, a jej projekt pokazany poniżej jest wyjątkowo elegancki.



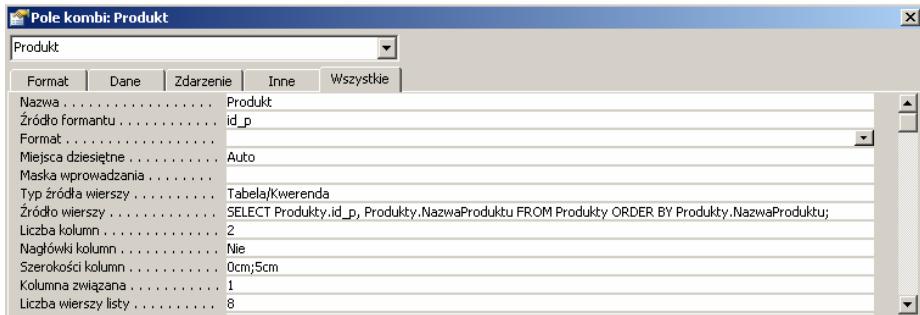
Przejdziemy teraz do zbudowania formularza o nazwie frmPodFakturaDetale, wykorzystamy go później jako podformularz w formularzu głównym.

Otwieramy nowy projekt formularza w oparciu o tabelę tempFakturaDetale, jako widok domyślny wybieramy arkusz danych. Z uwagi na zamiar sumowania wartości brutto pojedynczych produktów w celu wykorzystania tej informacji w formularzu głównymłączamy, korzystając z menu *Widok*, sekcję *Nagłówek/Stopka* formularza.

W oknie projektu tego nowego formularza zamykamy sekcję nagłówka minimalizując jego wysokość myszą, tak jak to pokazano niżej.

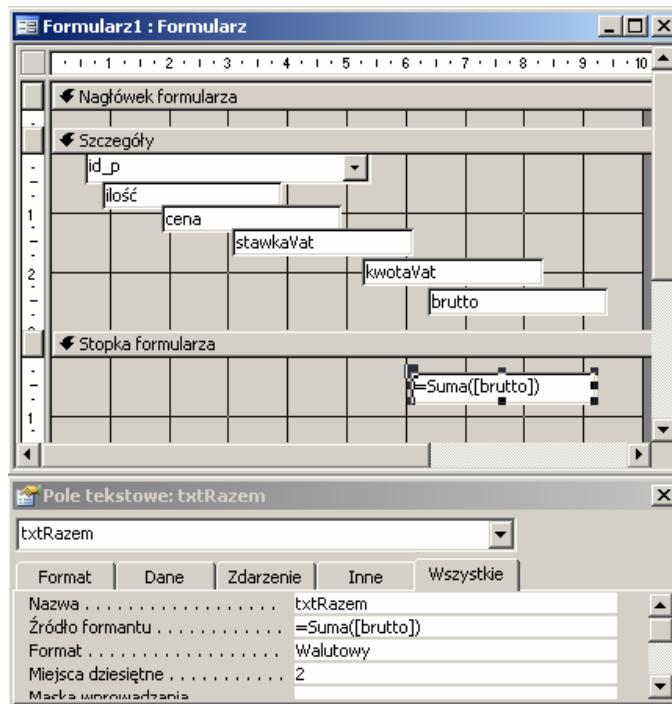


W sekcji *Szczegóły* musimy teraz umieścić pole kombi, które pozwoli na wybór nazwy produktu zwracając jednocześnie jego identyfikator. Pole to tworzymy w sposób omówiony wcześniej. Właściwości tego pola kombi pokazane są poniżej. Z uwagi na sposób pracy tego formularza (arkusz danych) nazwa tego pola (*Produkt*) jest taka, aby w sposób czytelny opisywała kolumnę danych w podformularzu.



Pozostałe pola przenosimy bezpośrednio z listy pól wyświetlonych w formularzu nie zwracając zbytniej uwagi na ich ustawienie w sekcji *Szczegóły*. Dla każdego z tych pól usuwamy etykietę zmieniając odpowiednio nazwę pola tekstowego (Ilość, Cena, Stawka Vat, Kwota Vat, Do zapłaty). Zmiany te wynikają z potrzeby opisania kolumn w podformularzu zgodnie z regułami języka polskiego.

W stopce formularza umieszczamy pole tekstowe o nazwie np. `txtRazem` wpisując jako źródło danych wyrażenie `=Suma([brutto])`.



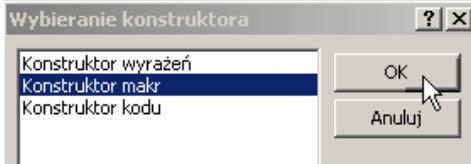
Pole `txtRazem` nie będzie wyświetlane w podformularzu, ale odwołamy się do tego pola w formularzu głównym wyświetlając łączną wartość faktury.

Kolejny problem, który musimy rozwiązać, to ustalenie ceny jednostkowej produktu i jego stawki podatku VAT. Te dwie informacje są właściwością produktu, dobrze więc byłoby, aby w momencie wyboru nazwy produktu w polu kombi `Produkt` automatycznie zostały przypisane odpowiednie wartości do pól `Cena` i `Stawka Vat`. Zadanie to byłoby stosunkowo łatwe do wykonania przy pomocy procedur języka VBA, z układu tej książki wynika jednak, że jeszcze nie możemy z tych rozwiązań skorzystać. Druga możliwość, to przygotowanie odpowiedniego makropolecenia (makra) i przypisanie go do zdarzenia *Po aktualizacji* pola kombi `Produkt`.

Proces tworzenia makra poprzedzimy jeszcze zapisaniem projektowanego formularza pod nazwą np. `frmPodFakturaDetale`, jest to konieczne z uwagi na potrzebę wskazywania kontrolek tego formularza z poziomu formularza głównego. Z tego samego powodu ustalimy tu, że formularz główny będzie miał nazwę `frmFaktura`.

Po przejściu do zakładki *Zdarzenia* w oknie właściwości pola kombi `Produkt` formularza `frmPodFakturaDetale` ustawiamy cursor w wierszu właściwości *Po*

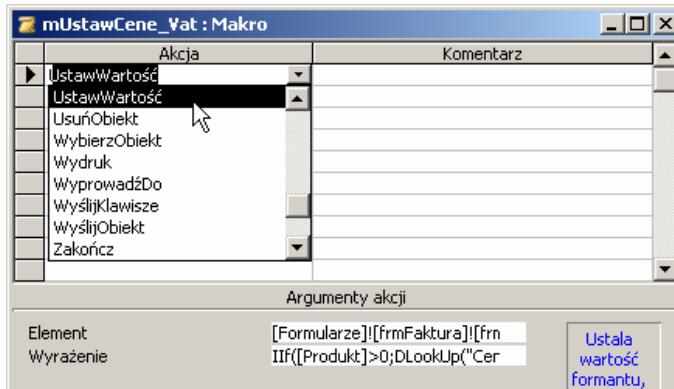
aktualizacji i po kliku przycisku z trzema kropeczkami otwieramy okno dialogowe *Wybieranie konstruktora*. W oknie tym spośród trzech dostępnych opcji wybieramy Konstruktor makr i akceptujemy wybór przyciskiem OK.



MS Access otwiera okno projektowania nowego makropolecenia wyświetlając jednocześnie okno dialogowe z pytaniem o nazwę tego nowego makra. W pokazanej niżej sytuacji makro to otrzyma nazwę `mUstalCene_Vat`.



Budowanie makra polega na wybraniu w kolumnie *Akcja* predefiniowanej czynności, w pokazanej sytuacji wybrana zostanie akcja *UstawWartość*, będziemy jeszcze musieli zdefiniować jej argumenty poprzez wskazanie kontrolki, której wartość chcemy ustawić oraz określenie tej nowej wartości.



W naszym przypadku pierwsze wywołanie akcji *UstawWartość* ma przypisać wartość do kontrolki Cena w podformularzu `frmPodFakturaDetale`, stąd w polu *Element* poniższy wpis.

```
[Formularze]![frmFaktura]![frmPodFakturaDetale].[Form]![cena]
```

W polu *Wyrażenie* musimy teraz wpisać taką instrukcję, która w momencie, gdy pole Produkt zwróci identyfikator produktu „sięgnie” do tabeli Produkty po wartość pola CenaBrutto. Funkcja IIf jest odpowiednikiem znanej np. z Excela funkcji logicznej. Jeżeli, w zależności od stanu warunku logicznego funkcja ta zwraca dwie wartości, odpowiednio zwrot_gdy_prawda oraz zwrot_gdy_fałsz. W naszym przypadku badany jest warunek, że pole Produkt zwraca informację większą od zera, jeżeli warunek jest prawdziwy, to uruchamiana jest funkcja Dlookup, jeżeli nie, to zwracana jest liczba zero.

Funkcja Dlookup jest wielce użyteczna w tych przypadkach, gdy chcemy zwrócić wartość wybranego pola z odpowiedniej tabeli, przy czym rekord jest określany poprzez warunek. Formalna składnia jest następująca:

```
Dlookup("NazwaPola"; "NazwaTabeli"; "WarunekWyszukaniaRekordu")
```

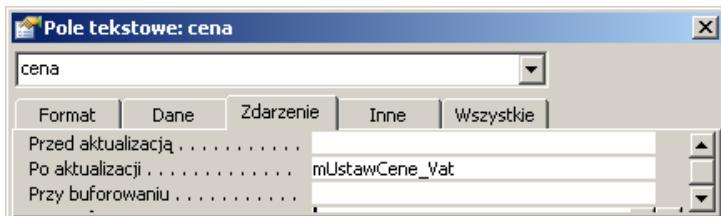
W przypadku naszego makra wyrażenie ma zwrócić wartość pola CenaNetto tabeli Produkty z tego rekordu, w którym pole id_p jest równe identyfikatorowi produktu wybranego w polu kombi Produkt.

```
IIf([Produkt]>0; DLookup("CenaNetto"; "Produkty";
    "id_p=" & [Produkt]); 0)
```

W analogiczny sposób zdefiniowano kolejną akcję *UstawWartość*, tym razem chodzi o przypisanie do kontrolki [Stawka Vat] wartości podatku VAT dla wybranego produktu. W polach *Element* i *Wyrażenie* wpisano odpowiednio:

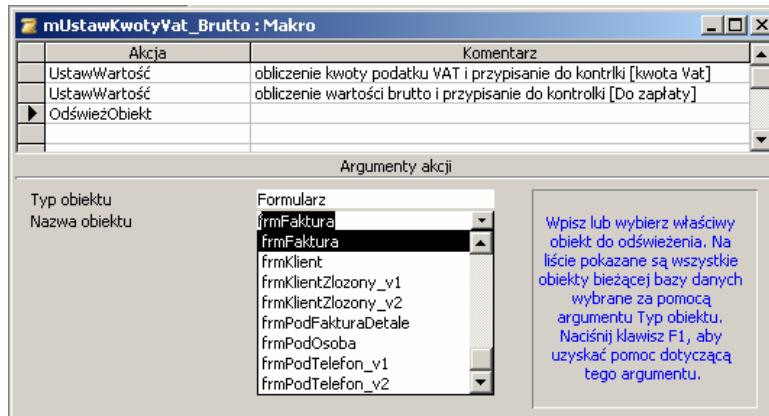
```
[Formularze]! [frmFaktura]! [frmPodFakturaDetaile].
[Form]! [Stawka Vat]
IIf([Produkt]>0; DLookup("StawkaVat"; "Produkty";
    "id_p=" & [Produkt]); 0)
```

Po zamknięciu makra (z zachowaniem zmian) w wierszu zdarzenia *Po aktualizacji* pola tekstowego Cena pojawi się nazwa utworzonego przed chwilą makra.



W bardzo podobny sposób utworzymy makro zdarzenia *Po aktualizacji* pola tekstowego Ilosc, jego zadaniem jest obliczenie i przypisanie do odpowiednich kontrolek kwoty podatku VAT oraz kwoty wartości brutto danego produktu. Do projektu tego makra

dodamy jeszcze akcję *OdświeżObiekt*, jej zadaniem będzie odświeżenie formularza głównego frmFaktura w celu pokazania aktualnej łącznej wartości faktury.



Argumenty pierwszego wywołania akcji *UstawWartość* zostały zdefiniowane następująco:

```
[Formularze]![frmFaktura]![frmPodFakturaDetaile].  
[Form]![kwota Vat]  
  
IIf([ilość]>0 And [Produkt]>0;  
    CCur([ilość]*[Cena]*[Stawka Vat]);0)
```

a dla drugiego wywołania następująco:

```
[Formularze]![frmFaktura]![frmPodFakturaDetaile].  
[Form]![Do zapłaty]  
  
IIf([ilość]>0 And [Produkt]>0;  
    CCur([ilość]*[Cena]*(1+[Stawka Vat]));0) .
```

W przypadku akcji *OdświeżObiekt* jako jej typ wskazano formularz, a nazwę formularza выбрано из списка объектов.

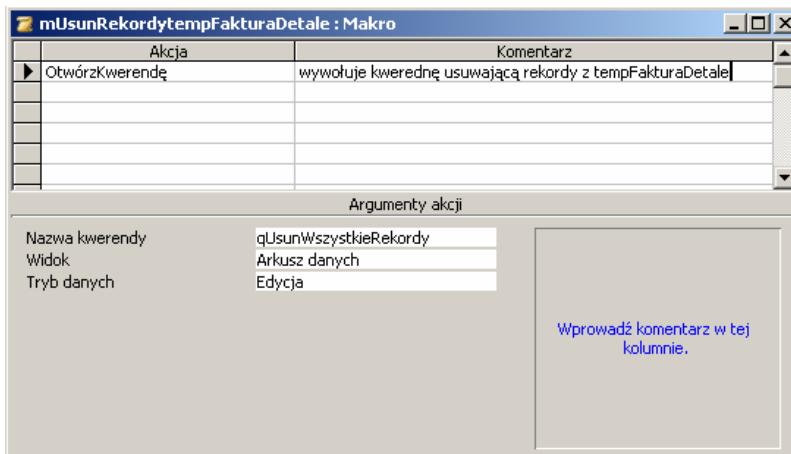
Po zapisaniu końcowej wersji formularza frmPodFakturaDetaile w bazie danych możemy przejść do zaprojektowania formularza głównego frmFaktura.

Otwieramy nowy projekt formularza wybierając jako źródło danych tabelę Faktura. Modyfikujemy odpowiednio tytuł formularza, wyłączamy przyciski Min i Max, także paski pionowego i poziomego przewijania. Znajdujemy właściwość Cykliczny i zmieniamy domyślną wartość Wszystkie rekordy na Bieżący rekord. Odszukujemy jeszcze właściwość Przyciski nawigacyjne i ustawiamy ją na wartość Nie. Zmiany dwóch ostatnich właściwości spowodowane są tym, że chcemy zablokować możliwość przejścia

do kolejnego rekordu za pomocą naciskania klawisza tabulatora (Tab). Taka możliwość byłaby wtedy, gdyby właściwości *Cykliczny* była przypisana opcja *Wszystkie rekordy*. Przejście do kolejnego rekordu byłoby także możliwe poprzez klik przycisku *następny* w zestawie przycisków nawigacyjnych, jego wyłączenie nie daje takiej możliwości. Obie zmiany zostały zrobione dlatego, że chcemy dokładnie kontrolować to, co się będzie działo w momencie zakończenia rejestracji nowej transakcji. Zrobimy to za pomocą specjalnego przycisku polecenia wyzwalającego odpowiednie makro, ale o tym trochę później.

Teraz zmienimy jeszcze jedną właściwość formularza, mianowicie właściwości *Wprowadzanie danych* przypiszemy opcję *Tak*. Dzięki temu formularz frmFaktura będzie się otwierał jako pusty formularz oczekujący na wprowadzenie nowych danych.

W momencie otwarcia formularza musimy wywołać kwerendę akcyjną usuwającą wszystkie dane z tabeli *tempFakturaDetaile*. Taka kwerenda została przez nas już stworzona na początku tego podrozdziału, teraz po prostu napiszemy makro uruchamiane w reakcji na zdarzenie *Przy otwarciu* projektowanego formularza.

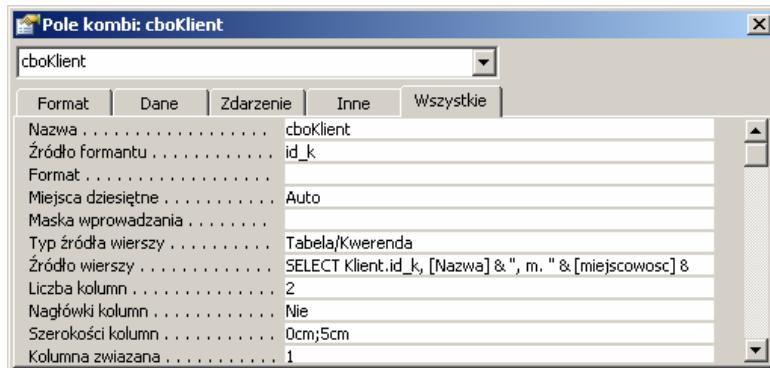


Możemy teraz przejść do umieszczenia kontrolek w sekcji *Szczegóły*, zaczniemy od utworzenia pola kombi, jego zadaniem będzie umożliwienie wyboru klienta firmy przy jednoczesnym zwrocie jego identyfikatora.

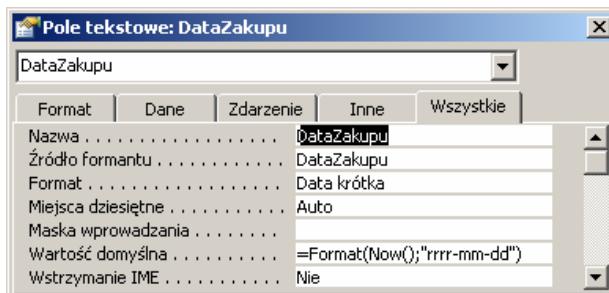
Pole kombi otrzymało nazwę *cboKlient*, jego źródłem danych jest pole *id_k*, typem danych dla listy pola kombi jest Tabela/Kwerenda, a źródło wierszy zostało zdefiniowane za pomocą zapytania SQL zbudowanego przy pomocy graficznego kreatora.

```
SELECT Klient.id_k, [Nazwa] & ", m. " & [miejscowosc] &
", Nip: " & [Nip] AS kto FROM Miejscowosc INNER JOIN Klient
ON Miejscowosc.id_m=Klient.Id_m ORDER BY Klient.nazwa;
```

Zapytanie to zwraca dwa pola, stąd właściwość *Liczba kolumn* ustawiona na dwa, pierwsza kolumna powinna być ukryta – stąd w wierszu *Szerokość kolumn* odpowiednio 0 cm i 5 cm. Ze źródłem formantu skojarzona jest informacja z pierwszego pola zwracanego przez zapytanie, stąd właściwość *Kolumna związana* ustawiona na 1.



Pod polem kombi ustawiamy pole *DataZakupu* wpisując w wierszu właściwości *Wartość domyślna* wyrażenie przypisujące temu polu datę bieżącą zgodnie z formatem „rrrr-mm-dd”.



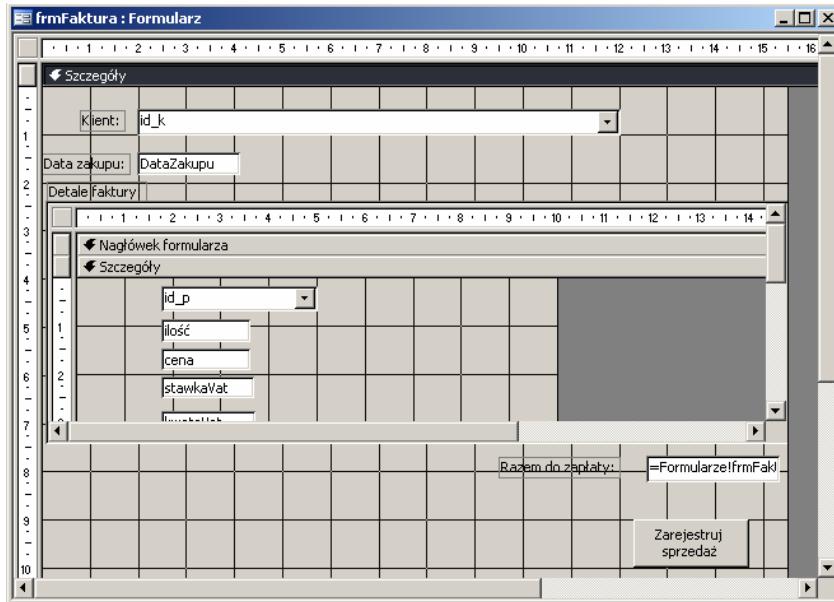
Poniżej pola daty umieszczamy kontrolkę podformularza rysując stosunkowo szeroki prostokąt, po zwolnieniu myszy konfigurujemy ten podformularz wskazując na wykorzystanie *frmPodFakturaDetale* z polem wiążącym *id_k*.

Pod obiektem podformularza umieszczamy teraz pole tekstowe, w etykiecie tego pola wpisujemy tekst „Razem do zapłaty”, a w źródle dajemy odwołanie do formantu *txtRazem*, który utworzyliśmy w stopce podformularza *frmPodFakturaDetale..*

```
=Formularze!frmFaktura!frmPodFakturaDetale.Form!txtRazem
```

Pole to odpowiednio ustawiamy (trocę metodą prób i błędów), tak aby było przedłużeniem kolumny „Do zapłaty” podformularza w (widoku danych).

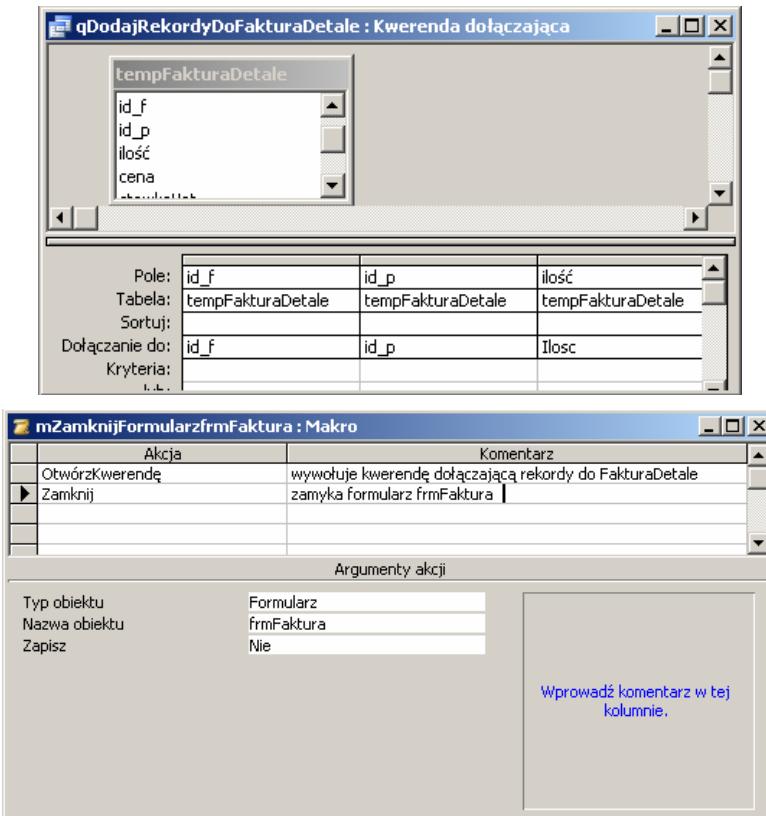
Pozostaje nam jeszcze umieszczenie na dole formularza kontroli typu przycisk polecenia (ang. *CommandButton*) i przypisanie mu odpowiedniego makra jako reakcji na zdarzenie *Przy kliknięciu*. Widok projektu tak przygotowanego formularza złożonego pokazany jest poniżej.



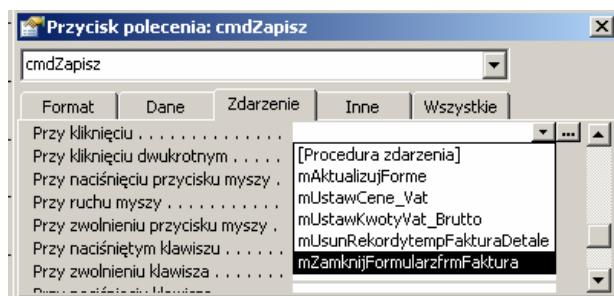
Pozostaje nam jeszcze przygotowanie działań, jakie mają być podjęte w momencie kliku przycisku polecenia „Zarejestruj sprzedaż”. Jak wiemy formularz frmFaktura został tak przygotowany, że szczegóły faktury nie trafiły do tabeli FaturaDetale, ale są tymczasowo przechowywane w tabeli tempFaturaDetale. W momencie zakończenia rejestracji sprzedaży dane z tabeli tymczasowej muszą być dołączone do tabeli FaturaDetale. Zrobimy to poprzez odpowiednio zaprojektowaną kwerendę dołączającą.

Projekt takiej kwerendy o nazwie qDodajRekordyDoFaturaDetale pokazany jest niżej, warto zwrócić uwagę, że ze źródła danych zostały wykorzystane jedynie pierwsze trzy pola opisujące identyfikator faktury, identyfikator produktu i jego ilość.

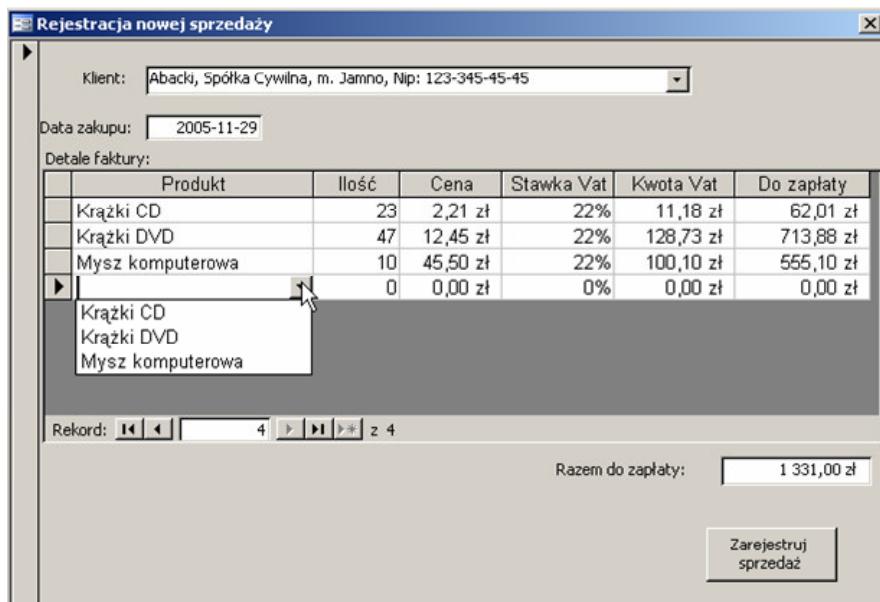
Pozostaje jeszcze przygotowanie makra uruchamiającego tę kwerendę, projekt takiego makra również pokazany jest poniżej. Poza akcją Otwórz kwerendę została dodana jeszcze jedna akcja Zamknij z zadaniem zamknięcia formularza frmFaktura.



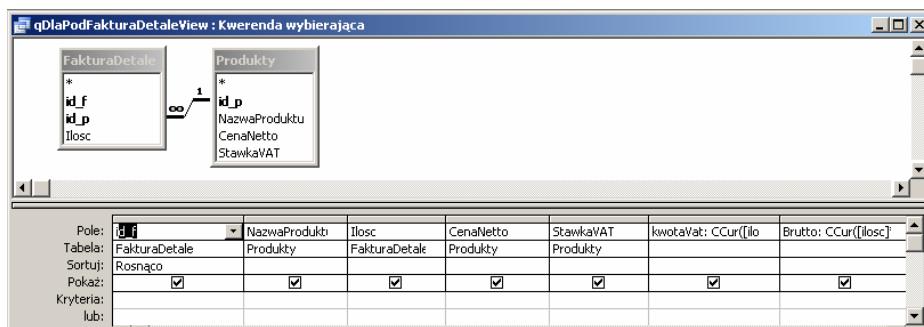
Pozostaje nam jeszcze przypisanie utworzonego makra do zdarzenia *Przy kliknięciu* przycisku polecenia cmdZapisz.



Na kolejnej stronie widok gotowego formularza frmFaktura w trakcie pracy.



Formularz frmFaktura został zaprojektowany **wyłącznie** dla potrzeb rejestracji nowej faktury. Gdybyśmy byli zainteresowani zbudowaniem formularza dla potrzeb przeglądania zrealizowanych zakupów, to jego projekt będzie bardzo podobny, z tym, że jako źródło danych dla podformularza posłuży kwerenda wybierająca. W kwerendzie tej wyznaczono kwotę Vat oraz wartość brutto dla każdego produktu.



Wykorzystując powyższą kwerendę jako źródło danych zaprojektowano nowy formularz frmPodFakturaDetaleView, także jako arkusz danych, z tym, że nie ma potrzeby stosowania makr, bo wszystkie potrzebne obliczenia zostały zrobione w kwerendzie.

Podobnie jak w poprzedniej wersji w sekcji *Stopka formularza* dodano pole tekstowe z zadaniem sumowania wartości faktury.

Jako formularz główny wykorzystano kopię formularza frmFaktura, której nadano nazwę frmFakturaView. Z uwagi na zbudowanie formularza **wyłącznie** do przeglądania dokonanych zakupów zmieniono kilka istotnych właściwości formularza. Takie właściwości jak *Edycja dozwolona*, *Usuwanie dozwolone* czy *Dodawanie dozwolone* ustawiono na wartość Nie. Analogiczną opcję ustawiono dla właściwości *Wprowadzanie danych*. Przycisk polecenia cmdZapisz nie jest potrzebny, został więc usunięty. W zamian zostały pokazane przyciski nawigacyjne.

Ponieważ korzystaliśmy z kopii formularza frmFaktura, to konieczne jest jeszcze usunięcie makra zdarzenia *Przy otwarciu* formularza (nie ma potrzeby usuwania rekordów z tabeli tempFakturaDetaile).

Musimy jeszcze uniemożliwić zmianę klienta w polu kombi cboKlient, wystarczy po prostu ustawić właściwość *Włączony* na Nie, a *Zablokowany* na Tak.

Konieczna jest aktualizacja nazwy podformularza oraz wpisu wyrażenia w polu tekstowym „razem do zapłaty” tak, aby się odwoływać do aktualnej nazwy podformularza i formularza głównego.

Poniżej widok formularza frmFakturaView w trakcie przeglądania dotyczących faktur i ich detali.

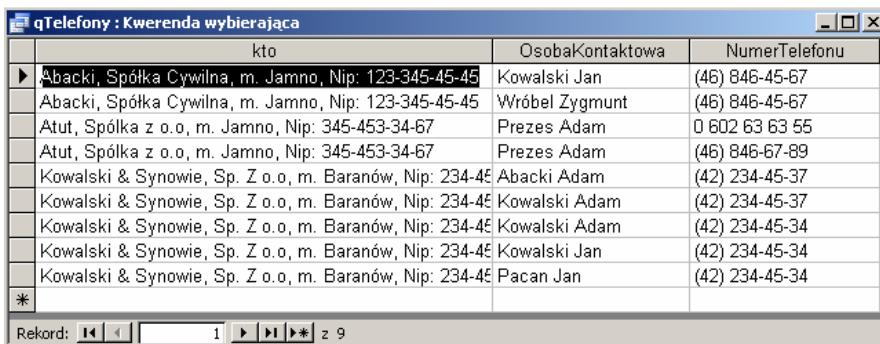
Produkt	Ilość	Cena	Stawka Vat	kwota Vat	Do zapłaty
Mysz komputerowa	2	45,50 zł	22%	20,02 zł	111,02 zł
Krążki DVD	15	12,45 zł	22%	41,09 zł	227,84 zł
Krążki CD	15	2,21 zł	22%	7,29 zł	40,44 zł

2.3.6. Raporty

Raporty w bazach danych to nic innego jak pisemne zestawienia informacji opisujące różnorodne operacje dokonywane na obiektach baz danych czy wyniki zestawień analitycznych. Raporty projektujemy w bardzo podobny sposób jak formularze; korzystając ze źródła danych będziemy umieszczać w różnych sekcjach raportu odpowiednie kontrolki, tak je rozmieszczając, aby otrzymać oczekiwany efekt. W kolejnych przykładach pokażemy projektowanie kilku ważnych typów raportów.

Raporty proste

W podroziale poświęconym kwerendom przygotowaliśmy kwerendę wybierającą o nazwie qTelefony, jej zadaniem było przyporządkowanie każdemu klientowi nazwiska osoby kontaktowej i numeru telefonu, pod którym dana osoba jest dostępna.



	kto	OsobaKontaktowa	NumerTelefonu
▶	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Kowalski Jan	(46) 846-45-67
	Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45	Wróbel Zygmunt	(46) 846-45-67
	Atut, Spółka z o.o, m. Jamno, Nip: 345-453-34-67	Prezes Adam	0 602 63 63 55
	Atut, Spółka z o.o, m. Jamno, Nip: 345-453-34-67	Prezes Adam	(46) 846-67-89
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-45	Abacki Adam	(42) 234-45-37
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-45	Kowalski Adam	(42) 234-45-37
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-45	Kowalski Adam	(42) 234-45-34
	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-45	Kowalski Jan	(42) 234-45-34
*	Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-45	Pacan Jan	(42) 234-45-34

Kwerendę tę wykorzystamy jako źródło danych do przygotowania raportu, w którym dla każdego klienta wprowadzimy wszystkie numery kontaktowe z wypisaniem nazwisk ludzi dostępnych pod danym telefonem.

Prace nad zbudowaniem takiego raportu zaczynamy od przejścia do obiektu *Raporty* w oknie bazy danych, a następnie pryciskiem *Nowy* otwieramy okno dialogowe nowego raportu. W oknie tym zaznaczamy opcję *Widok projektu*, a z listy źródła danych selekcjonujemy kwerendę qTelefony. Po akceptacji wyboru pryciskiem OK Access otwiera okno projektu nowego raportu, z reguły pokazywana jest także lista dostępnych pól oraz przybornik z kontrolkami.

Standardowo okno projektu otwierane jest w takim widoku, że poza sekcją *Szczegóły* wyświetlane są także sekcje *Nagłówek strony* oraz *Stopka strony*. W sekcji *Nagłówek strony* będziemy z reguły umieszczać opisy kolumn (pól) umieszczonych w sekcji *Szczegóły*, chodzi o to, aby przy dużej liczbie zwracanych rekordów każda strona raportu miała stosowny opis kolumn. W takiej sytuacji w sekcji *Stopka strony* z reguły umieszczana jest informacja o numerze strony raportu.

W naszym przypadku, korzystając z menu *Widok*,łączymy jeszcze widok *Nagłówka/Stopki raportu*, w nagłówku raportu umieścimy po prostu etykietę będącą tytułem raportu, także datę jego wydrukowania. Z kolei w stopce może pojawić się informacja o autorze raportu.

W założeniach do projektu raportu powiedzieliśmy, że chcemy mieć w raporcie informacje pogrupowane w ten sposób, że dla każdego klienta wyprowadzimy jego numery telefonów, a każdemu z nich przyporządkujemy osobę kontaktową.

Taką funkcjonalność zrealizujemy poprzez wywołanie z menu *Widok* polecenia *Sortowanie i grupowanie*. W oknie dialogowym tego polecenia wybieramy teraz pole *Kto* jako pierwsze pole grupujące wraz z utworzeniem nagłówka dla grupy. Podobnie postępujemy z polem *NumerTelefonu*.



Po zamknięciu okna *Sortowanie i grupowanie* okno projektu raportu zostanie rozbudowany poprzez dodanie dwóch nowych sekcji: *Nagłówka* dla pola *kto* oraz *nagłówka* dla pola *NumerTelefonu*.

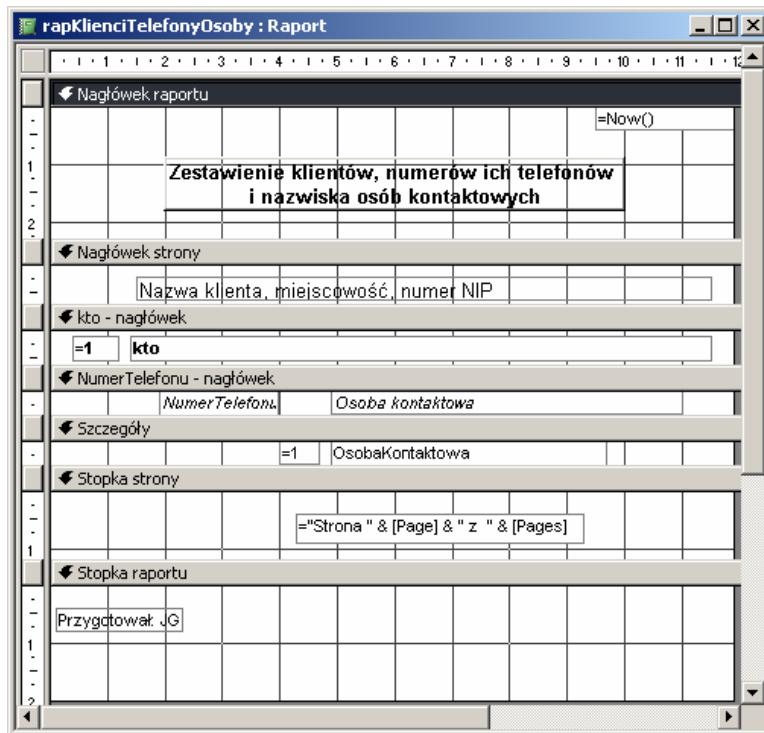
Mając otwarte wszystkie potrzebne sekcje projektu raportu możemy przystąpić do umieszczania kontrolek, zarówno związanych ze źródłem danych, jak i niezwiązanych, które wykorzystamy do opisu zwracanych informacji.

W sekcji *Nagłówek raportu* wstawiamy etykietę, której przypiszemy tekst np.”Zestawienie klientów, numerów ich telefonów i nazwiska osób kontaktowych”. Etykieta ta musi być w miarę porządnego sformatowana (rozmiar, położenie, czcionka, rozmiar czcionki, sposób jej wypisania). W tej samej sekcji wstawimy także niezwiązane pole tekstowe, pole to wykorzystamy do zwrócenia daty druku raportu, wystarczy, że właściwości *źródło formantu* przypiszemy wyrażenie `=Now()` a jako *format* tego pola wybierzemy opcję *Data krótka*.

W nagłówku strony wstawiamy etykietę, jej zadaniem będzie powtarzanie na każdej stronie raportu (jeżeli będziemy mieć raport wielostroanicowy) opisu informacji, które będą

pojawiać się niżej. W naszym przykładzie właściwości *Tytuł* tej etykiety przyporządkowano tekst „Nazwa klienta, miejscowości, numer NIP”. Podobnie jak w przypadku poprzedniej etykiety potrzebne jest w miarę porządne jej sformatowanie obejmujące te same elementy.

W sekcji *Nagłówek pola Kto* umieszczamy teraz pole *Kto* przeciągając je z listy pól źródła danych. Pozostawiamy tylko samo pole tekstowe usuwając jego etykietę (nie jest potrzebna, bo mamy opis tego pola w nagłówku strony). Przed polem tekstowym *Kto* możemy umieścić niezwiązane pole tekstowe z przypisaniem właściwości *Źródło formantu* wyrażeniem `=1`. Jeżeli właściwości *Suma bieżąca* tego pola przypiszemy opcję *Wszędzie*, to otrzymamy automatyczną numerację klientów w całym raporcie. Pola tekstowe wpisane w tej sekcji rozpoczynają grupę – stąd konieczne jest takie ich sformatowanie, aby każda grupa była wyraźnie zaznaczona. W pokazanym niżej projekcie tego formularza pola tej sekcji zostały wypisane czcionką pogrubioną. Dodatkowo poniżej obu pól umieszczono linię pojedynczą oddzielającą elementy grupy od jej opisu.



W kolejnej sekcji nagłówkowej umieszczono pole tekstowe *NumerTelefonu* oraz etykietę opisującą osoby kontaktowe. Znów dla zaznaczenia grupy użyto zmieniono krój

czcionki na pochylony, zastosowano także metodę wcięć (odsunięcia kontrolek od lewej krawędzi okna) dla lepszego zaznaczenia hierarchii informacji.

W sekcji *Szczegóły* umieszczono pole *OsobaKontaktowa* przeciągając je ze źródła danych, pole to zostało poprzedzone niezwiązanym polem tekstowym wykorzystywanym do wyprowadzenia numeracji osób kontaktowych skojarzonych z danym numerem telefonu. Efekt taki osiągnięto poprzez przypisanie właściwości *Suma bieżąca* opcji *W grupie*. Podobnie jak w przypadku wcześniejszej sekcji zastosowano wcięcia dla uwypuklenia hierarchii informacji.

Raport jest w zasadzie gotowy, poniżej jego widok w oknie podglądu wydruku.

The screenshot shows a window titled "rapKlienciTelefonyOsoby : Raport". The main content area displays a report titled "Zestawienie klientów, numerów ich telefonów i nazwiska osób kontaktowych". The report lists three companies with their contact details:

- 1 Abacki, Spółka Cywilna, m. Jamno, Nip: 123-345-45-45**
 - (46) 846-45-67 Osoba kontaktowa
 - 1 Wróbel Zygmuńt
 - 2 Kowalski Jan
- 2 Atut, Spółka z o.o, m. Jamno, Nip: 345-453-34-67**
 - (46) 846-67-89 Osoba kontaktowa
 - 1 Prezes Adam
 - 0 602 63 63 55 Osoba kontaktowa
 - 1 Prezes Adam
- 3 Kowalski & Synowie, Sp. Z o.o, m. Baranów, Nip: 234-456-45-56**
 - (42) 234-45-34 Osoba kontaktowa
 - 1 Kowalski Jan
 - 2 Pacan Jan
 - 3 Kowalski Adam
 - (42) 234-45-37 Osoba kontaktowa
 - 1 Abacki Adam
 - 2 Kowalski Adam

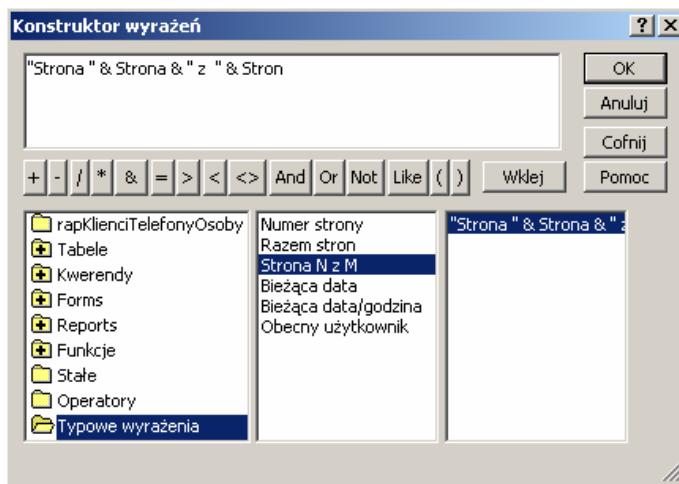
At the bottom left, it says "Przygotował: JG". At the bottom right, there is a page navigation bar labeled "Strona" with buttons for previous, next, and first/last pages.

Dla uzupełnienia dodamy jeszcze, że w sekcji *Stopka strony* zostało umieszczone niezwiązane pole tekstowe z zadaniem wyprowadzenia numeracji stron raportu w formacie „Strona x z N stron”.

Formalnie zadanie to zostało zrealizowane poprzez przypisanie właściwości *Źródło formantu* wyrażenia:

```
= "Strona " & [Page] & " z " & [Pages]
```

w którym pola *[Page]* i *[Pages]* są systemowymi polami MS Access zwracającymi numer strony bieżącej i liczbę wszystkich stron raportu. W tym miejscu konieczna jest jedna uwaga: wyrażenie to może być skomponowane poprzez wykorzystanie okna kreatora wyrażeń uruchamianego kliknięciem przycisku oznaczonego trzema kropkami w wierszu właściwości *Źródło formantu*.



Wklejenie jednak pokazanego wyżej wyrażenia powoduje, że jako źródło formantu zostanie wstawione wyrażenie:

```
= "Strona " & [Page] & " z " & [Stron]
```

które jest **niepoprawne** z powodu wykorzystania pola *[Stron]* zamiast poprawnej wersji *[Pages]*. Błąd ten występuje w polskiej wersji językowej MS Access i jest evidentnym błędem programistów, na całe szczęście możemy go łatwo poprawić zastępując pole *[Stron]* polem *[Pages]*.

Raporty złożone

W rozdziale poświeconym formularzom złożonym przygotowaliśmy formularz o nazwie frmFaktura, przy jego pomocy w łatwy sposób można było zarejestrować pojedynczą transakcję sprzedaży. Na zakończenie tej transakcji powinien zostać wydrukowany pisemny dowód jej przeprowadzenia, czyli faktura. Jej przygotowaniu w formie raportu złożonego poświęcony jest ten rozdział.

Raport faktury, podobnie jak formularz złożony, będzie zbudowany z raportu głównego opisującego stronę kupującą i sprzedającą oraz podraportu opisującego szczegóły zakupu (detale faktury). Przed rozpoczęciem prac projektowych nad tymi dwoma raportami musimy przygotować odpowiednie źródła danych.

Podraport musi zwracać te informacje, które są zapisywane w tabeli pomocniczej tempFakturaDetale z jednym wyjątkiem: w miejsce identyfikatora produktu musimy mieć jego pełną nazwę. Prostym rozwiązaniem będzie zbudowanie pokazanej niżej kwerendy wybierającej zwracającą wybrane informacje z tabel tempFakturaDetale i Produkty. Między tymi tabelami nie zostały zdefiniowane relacje, stąd po ich dodaniu do okna projektu kwerendy taka relacja została ustaloniona poprzez przeciągnięcie pola id_p z tabeli Produkty na pole id_p w tabeli tempFakturaDetale.

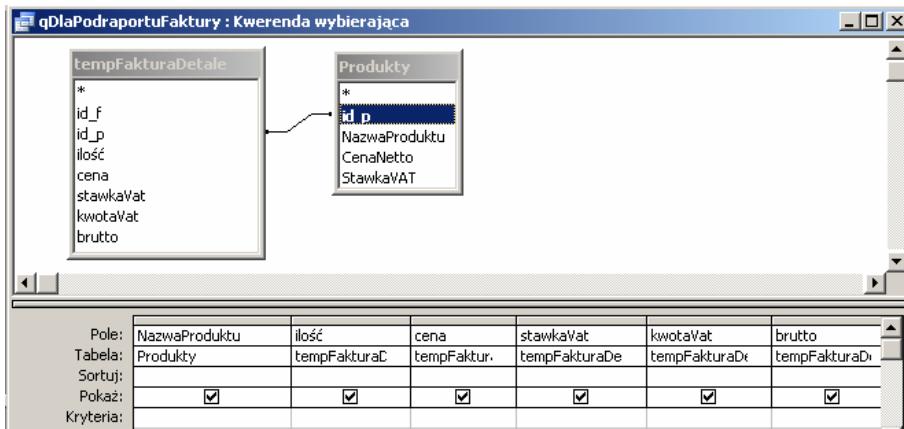
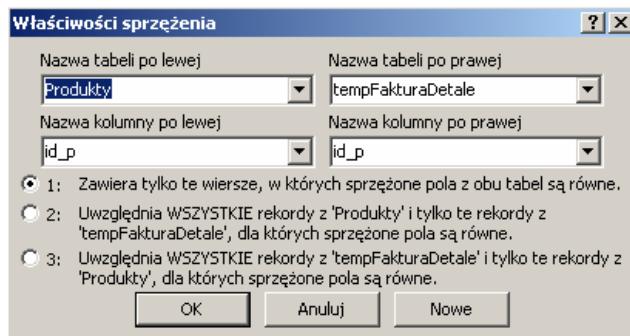
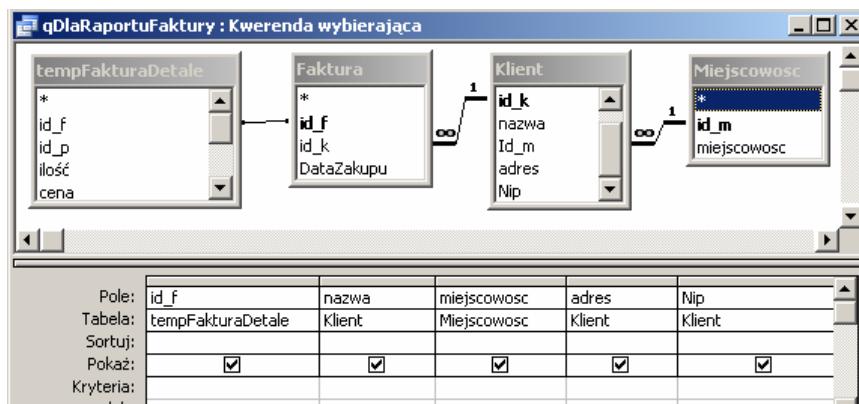


Tabela Produkty była potrzebna w tej kwerendzie po to, aby rozszerzyć identyfikator produktu zapisywany w trakcie rejestracji szczegółów sprzedaży w polu id_p tabeli tempFakturaDetale. Relacja tworzona w projekcie kwerendy ma charakter tymczasowy, stąd zupełnie inne jej oznaczenie niż te, które spotykamy w oknie relacji. Funkcjonalnie utworzona relacja zwróci z obu tabel te rekordy, dla których połączone pola mają te same wartości. Właściwości relacji można zmodyfikować wywołując z menu kontekstowego polecenie *Właściwości sprzężenia* (lub z menu *Widok*).



Raport główny musi być oparty o takie źródło danych, które opisze kupującego, dodatkowo powinno zwrócić numer kolejny faktury. Pokazana niżej kwerenda zwraca takie informacje.

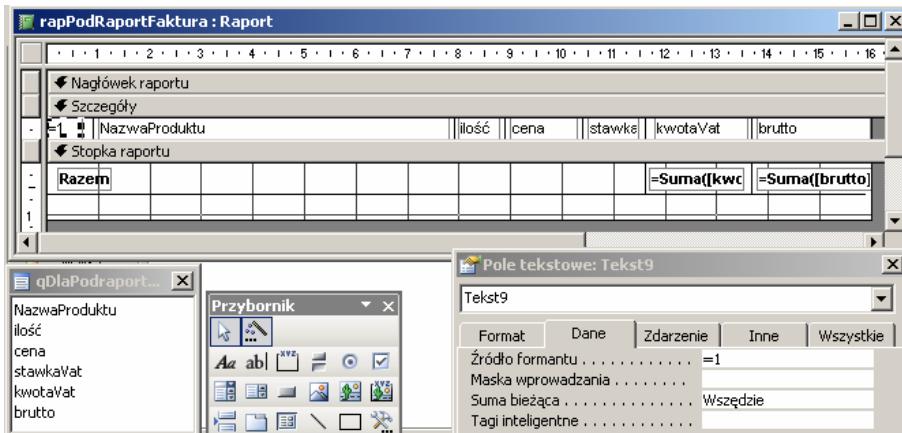


W zasadzie wszystkie potrzebne dane mamy w tabelach Klient, Miejscowosc i Faktura. Tabela tempFakturaDetale jest nam potrzebna po to, aby uzyskać aktualny numer faktury, dodatkowo musimy jeszcze we właściwościach kwerendy w wierszu *Wartości unikatowe* wybrać opcję Tak (jeżeli tego nie zrobimy, to kwerenda zwróci nam nie jeden rekord, a tyle, ile jest rekordów w tabeli tempFakturaDetale).

Po przygotowaniu obu źródeł danych możemy przejść do zaprojektowania obu raportów, zaczniemy oczywiście od przygotowania podraportu.

Otwieramy nowy projekt, jako jego źródło danych wskazujemy utworzoną przed chwilą kwerendę qDlaPodraportuFaktury. Domyślnie MS Access otwiera projekt raportu z włączonym *Nagłówkiem/Stopką strony*, w naszym przypadku obie te sekcje będą niepotrzebne, za to będzie nam potrzebna sekcja *Stopki raportu*. Korzystając z menu *Widok* włączamy *Nagłówek/Stopka raportu*.

Po zamknięciu sekcji *Nagłówek raportu* umieszczamy w sekcji *Szczegóły* wszystkie pola ze źródła danych pamiętając o tym, że w tej sekcji umieszczamy tylko pola tekstowe, etykiety tych pól nie są nam potrzebne, ponieważ chcemy, aby ten raport był w układzie arkusza danych. Pola te umieszczamy mniej więcej w takich pozycjach, jak to pokazano niżej. Dodatkowo, na początku tego zestawu pól umieścimy niezwiązane pole tekstowe, jego zadaniem będzie ponumerowanie wszystkich rekordów w tej sekcji. Sposób zdefiniowania tego pola pokazany jest we fragmencie okna jego właściwości.

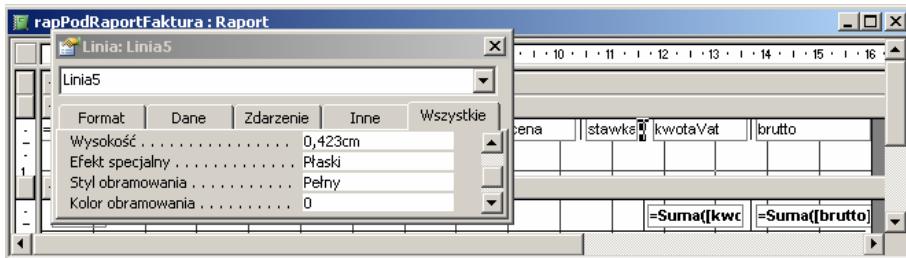


W sekcji *Stopka raportu* umieściliśmy dwa niezwiązane pola tekstowe z zadaniem podsumowania kwot podatku VAT oraz wartości faktury brutto. Na początku tego wiersza umieszczona została etykieta z tytułem „Razem”. Wszystkie trzy kontrolki zostały wyróżnione poprzez wypisanie ich czcionką pogrubioną.

W tym momencie projekt tego raportu nie zawiera opisu kolumn, opis ten umieścimy w raporcie głównym bezpośrednio nad podraportem, którym będzie aktualnie projektowany raport.

W zestawie kontrolek (w oknie *Przybornik*) mamy do dyspozycji kontrolę Linia, można ją wykorzystać do odseparowania poszczególnych kolumn w sekcji *Szczegóły*. Po zaznaczeniu myszą tej kontrolki rysujemy w sekcji *Szczegóły* pionową linię, jej wysokość powinna być dokładnie równa wysokości pozostałych kontrolki w tej sekcji. Zawsze można w oknie właściwości narysowanej linii sprawdzić, czy właściwości *Szerokość* jest przypisana liczba 0, oraz ustalić precyzyjnie jej wysokość zgodnie z wysokością pól tekstowych (w naszym przykładzie jest to 0,423 cm). Po ustaleniu tych właściwości umieszczamy kontrolkę linii dokładnie między dwoma wybranymi polami tekstowymi. Takie pozycjonowanie może być wykonane za pomocą strzałek kurSORA, można także wpisać odpowiednie wartości w oknie właściwości w wierszach *Lewy* i *Górny* (tu powinno być 0). Po ustawnieniu pierwszej z tych kontrolki (linii) selekcjonujemy ją i kopujemy ją do schowka. Metodą *Wklej* umieszczamy jej kopię w sekcji *Szczegóły*, a następnie lokujemy

miedzy innymi dwoma polami tekstowymi itd., operację te powtarzamy tak dugo, aż wszystkie pola zostaną oddzielone pionowymi liniami. Poniżej pokazane jest okno projektu z ustawianiem linii miedzy polami `stawkaVat` i `kwotaVat`.



W sekcji *Stopka raportu* wykorzystano dwie linie poziome, pierwsza z nich oddziela informacje umieszczone w tej sekcji (podsumowania) od rekordów zwracanych w sekcji *Szczegóły*, druga zamkna te informacje od dołu. Obie linie muszą być oczywiście tej samej szerokości, ich właściwość *Wysokość* musi być ustawiona na 0 (dla linii poziomej).

W sekcji *Stopka raportu* także możemy narysować pionowe linie separujące poszczególne informacje zwracane w tej sekcji, ich wysokość musi odpowiadać wartości właściwości *Górny* dolnej linii poziomej.

Efektem naszej dotychczasowej pracy powinien być taki widok tego raportu, tu pokazany w oknie podglądu (dla przykładowych danych).

rapPodReportFaktura : Raport					
1 Karta sieciowa bezprzewodowa	1 129,99 zł	7%	9,10 zł	139,09 zł	
2 Karta sieciowa D-Link 10/100 mbps	3 99,99 zł	22%	65,99 zł	365,96 zł	
3 Monitor 17" LCD	1 199,00 zł	7%	83,93 zł	1 282,93 zł	
4 Monitor 17 Samsung	2 300,00 zł	22%	264,00 zł	1 464,00 zł	
5 Wentylator procesora MX-24	1 45,68 zł	22%	10,05 zł	55,73 zł	
6 Krążki DVD	55 12,45 zł	22%	150,65 zł	835,40 zł	
Razem			583,72 zł	4 143,11 zł	
Strona: 1 / 1					

Po zapamiętaniu raportu pod nazwą np. `rapPodReportFaktura` możemy przejść do zaprojektowania raportu głównego.

Otwieramy okno nowego raportu wybierając jako źródło danych utworzoną wcześniej kwerendę `qDlaReportuFaktury`. Otwarty projekt zawiera sekcję *Szczegóły* i *Nagłówek/Stopkę strony*, będziemy jeszcze potrzebować sekcję *Nagłówka raportu*, a więc przy pomocy menu *Widok* włączamy opcję *Nagłówek/Stopka raportu*.

W sekcji nagłówka raportu musimy umieścić formant wyświetlający bieżącą datę, dane podmiotu sprzedającego oraz podmiotu kupującego oraz nazwę i numer dokumentu.

Jest to dość dużo informacji, zaczynamy więc od znacznego powiększenia wysokości tej sekcji. W prawym górnym narożniku umieszczamy niezwiązane pole tekstowe wpisując jako jego źródło danych wyrażenie `=Now()` oraz formatując to pole tak, aby zwracało datę krótką (w postaci 2005-12-02).

W lewym górnym narożniku umieszczamy etykietę, w której w kilku wierszach opisujemy podmiot sprzedający.

Pozostałe informacje w tej sekcji będą już pobierane ze źródła danych: umieszczamy kolejno nazwę klienta, jego miejscowości, adres i numer NIP. Pola te ustawiamy w odpowiednich miejscach tej sekcji i formatujemy zgodnie z ogólnymi standardami (jeżeli takie są).

Poniżej danych podmiotu kupującego umieścimy jeszcze niezwiązane pole tekstowe, w wierszu jego właściwości *Źródło formantu* wpisujemy następujące wyrażenie:

```
= "Faktura nr " & LTrim(Str([id_f])) & "/" &  
    LTrim(Str(Year(Now()))))
```

Zadaniem tego wyrażenia jest zwrócenie nazwy dokumentu (tu "Faktura") uzupełnionej o numer dokumentu skomponowany z jej (faktury) identyfikatora oraz roku bieżącego. Wyrażenie to zawiera informacje o różnym typie danych: nazwa dokumentu to tekst, identyfikator faktury (pole `id_f`) to liczba, podobnie jak rok bieżący. Dodać dany liczbowy do tekstu wymaga wcześniejszego ich przekształcenia na tekst, robimy to za pomocą funkcji systemowej `Str(wyrażenie)`, która zamienia argument `wyrażenie` na odpowiadający mu ciąg znaków. Specyfika pracy funkcji `Str` jest taka, że zwracany ciąg znaków jest dłuższy o jeden znak rezerwowany na symbol minus, jeżeli wyrażenie jest ujemne. Dla wyrażeń dodatnich ten pierwszy znak jest pusty (jest spacja). Stąd użycie funkcji systemowej `LTrim(tekst)`, jej zadaniem jest zwrócenie argumentu `tekst` bez spacji z przodu. I jeszcze problem uzyskania liczby reprezentującej bieżący rok, tu skorzystaliśmy z funkcji `Year(data)`, która zwraca rok z argumentu `data`. W naszym przypadku jako argument dla funkcji `Year` została wykorzystana funkcja `Now()`, bardzo przydatna funkcja systemowa zwracająca bieżącą datę systemową.

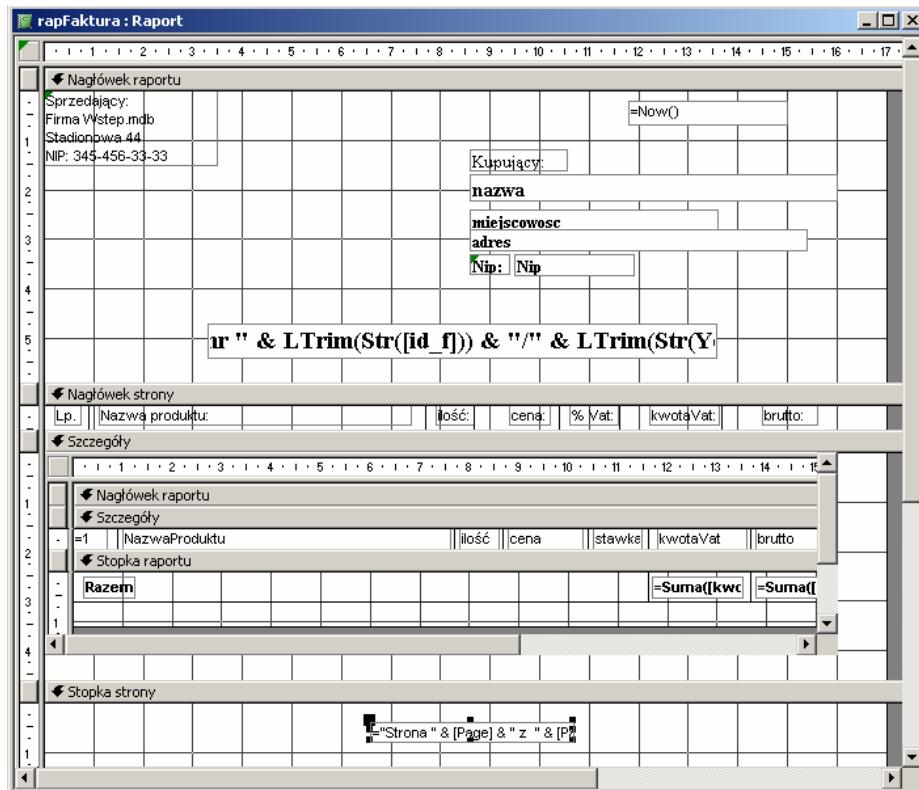
Możemy teraz przejść do sekcji *Nagłówek strony*, tu umieszczamy etykiety opisujące poszczególne kolumny danych zwracanych przez wcześniej zaprojektowany podraport. Ich zwymiarowanie i ustawienie w tej sekcji wymaga trochę pracy, ale warto ją włożyć, dzięki temu nasz raport będzie wyglądał w miarę elegancko.

Wiersz etykiet opisujących kolumny podraportu można ograniczyć dwoma liniami poziomymi, można także wprowadzić linie pionowe rozdzielające poszczególne etykiety. Po wstawieniu podraportu linie pionowe muszą być tak ustawione, aby były przedłużeniem linii pionowych podraportu.

Pozostaje nam jeszcze umieszczenie w sekcji *Szczegóły* obiektu podformularza, jako obiekt źródłowy wskazujemy utworzony wcześniej raport `rapPodraportFaktury`.

Z uwagi na charakter tworzonego dokumentu nie musimy wskazywać pól łączących raport główny z podraportem (tworzymy dokument, który będzie się odwoływał do jednej tylko faktury).

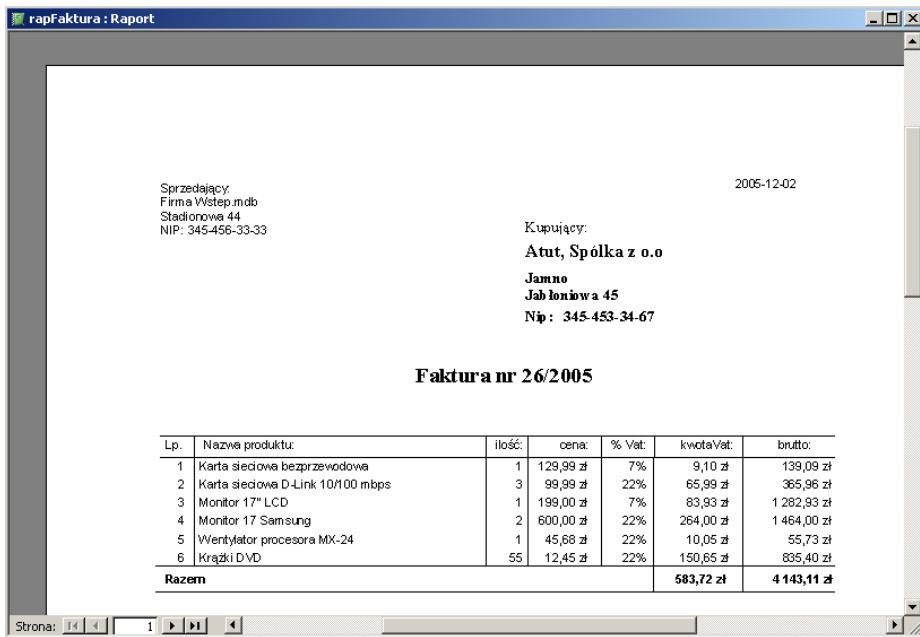
Pozostaje nam jeszcze umieszczenie w sekcji *Stopka raportu* niezwiązanej pola tekstowego z typowym wyrażeniem zwracającym numer strony raportu i projekt raportu *rapFaktura* jest gotowy. Poniżej widok projektu tego raportu, w takiej postaci jest on dostępny w bazie *Wstep.mdb* na dołączonym do tej książki krążku CD.



W tym momencie powinniśmy powrócić do projektu formularza *frmFaktura* po to, aby zmodyfikować makro uruchamiane przyciskiem *cmdZapisz*, chodzi po prostu o to, aby makro *mZamknijFormularzfrmFaktura* wykonało jeszcze jedną akcję, a mianowicie otworzyło raport *rapFaktura* w widoku podglądu wydruku (docelowo aby uruchomiło druk tego raportu).



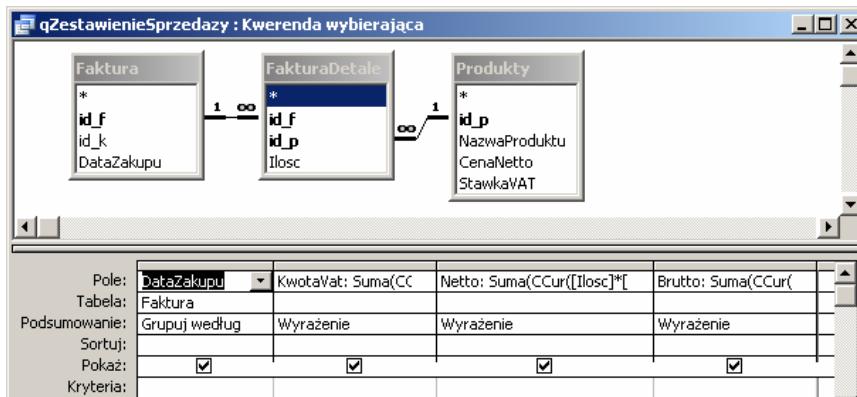
Po zapamiętaniu zmodyfikowanego makra wszystko jest już gotowe. Teraz w momencie kliku przycisku polecenia cmdZapisz w formularzu frmRaport poza wcześniejszymi czynnościami zostanie jeszcze otwarty raport faktury. Przykładowy widok pokazany jest poniżej.



Obliczenia w raportach

Powiedzmy, że interesuje nas zbiorcze zestawienie wartości sprzedaży brutto, należnego podatku VAT oraz wartości netto pogrupowanych wg dat transakcji. Dokument tego typu powinien zawierać także podsumowanie tych trzech informacji.

Wszystkie potrzebne dane możemy uzyskać projektując stosunkowo prosty raport, ale wcześniej potrzebne jest źródło danych dla tego raportu. W naszym przypadku źródłem tym będzie kwerenda grupująca, której projekt pokazany jest poniżej.



Kwerenda ta grupuje dane wg pola DataZakupu, pozostałe trzy pola są polami wyliczanymi wg formuł:

KwotaVat: Suma(CCur([Ilosc]*[CenaNetto]*[StawkaVAT]))

Netto: Suma(CCur([Ilosc]*[CenaNetto]))

Brutto: Suma(CCur([ilosc]*[CenaNetto]*(1+[StawkaVat])))

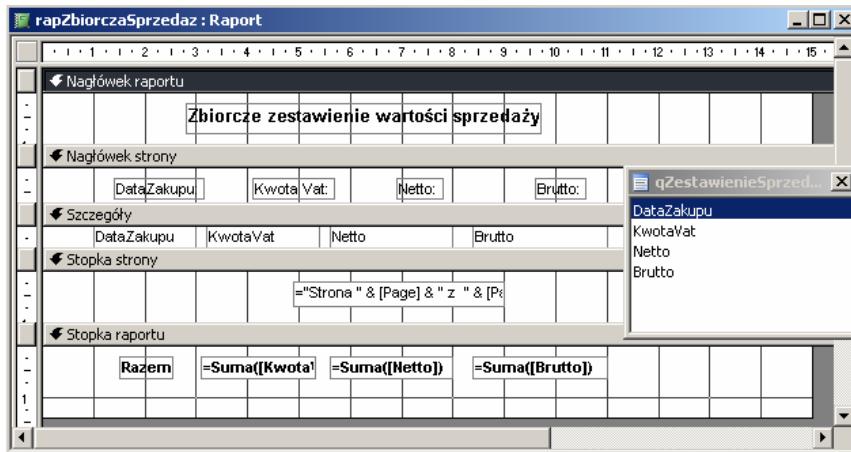
Po przygotowaniu źródła danych otwieramy nowy projekt raportu oparty o utworzoną przed chwilą kwerendę. W sposób podobny do dwóch wcześniejszych raportów projektujemy nasz raport sprzedaży.

W sekcji *Nagłówek raportu* umieszczamy etykietę z tytułem raportu, w sekcji *Nagłówek strony* tworzymy etykiety opisujące pola źródła danych, które umieścimy w sekcji *Szczegóły*.

W sekcji *Stopka raportu* tworzymy trzy niezwiązane pola tekstowe, ich źródłem danych będzie funkcja suma „pracująca” na odpowiednim polu liczbowych źródła danych. Przykładowo, dla podsumowania pola brutto będzie to wyrażenie =Suma([brutto]).

Wszystkie pola zwracające wartości liczbowe powinny być sformatowane na format *Walutowy*, a dokładność miejsc dziesiętnych ustawiona na dwa. Na zakończenie projektowania powinniśmy jeszcze dodać linię separującą etykiety nagłówka od rekordów sekcji *Szczegóły* oraz linię oddzielającą te rekordy od podsumowania w sekcji *Stopka raportu*.

Ostatecznie projekt tego raportu powinien być mniej więcej zgodny z pokazanym niżej.



A tak wygląda zbiorczy raport sprzedaży w widoku podglądu wydruku. Z uwagi na rozmiar tego „zrzutu ekranowego” nie jest widoczny numer strony wyświetlony w jej stopce.

DataZakupu	Kwota Vat:	Netto:	Brutto:
2005-11-02	34,33 zł	156,03 zł	190,36 zł
2005-11-05	68,40 zł	310,90 zł	379,30 zł
2005-11-07	46,76 zł	212,55 zł	259,31 zł
2005-11-29	1 855,75 zł	8 435,22 zł	10 290,97 zł
2005-12-01	21,88 zł	99,45 zł	121,33 zł
2005-12-02	1 509,21 zł	11 124,79 zł	12 634,00 zł
Razem	3 536,32 zł	20 338,94 zł	23 875,26 zł

Ten przykład zakończył rozdział 2 naszej książki poświęcony przeglądowi zasad projektowania baz danych z przykładami normalizacji, tworzenia relacji oraz projektowania kwerend, formularzy, raportów i makr. Dalsze rozdziały poświęcone są trochę bardziej zaawansowanym metodom projektowania obiektów baz danych.

3. Język SQL

W momencie wprowadzenia relacyjnych baz danych konieczne stało się opracowanie narzędzia, które umożliwia manipulowanie danymi zapisanymi w tabelach (i innych obiektach). Rolę taką spełnia SQL – standardowy język zapytań.

SQL (*Structured Query Language*) jest według definicji ANSI (*American National Standards Institute*) standardowym językiem komputerowym przeznaczonym do dostępu do danych oraz do manipulacji danymi zgromadzonymi w relacyjnych bazach danych. Wyrażenia SQL są wykorzystywane do pobierania i aktualizacji danych, a SQL współpracuje w zasadzie ze wszystkim popularnymi SQRBD (MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, MySQL itd.).

Istnieje wiele różnych wersji języka SQL, na szczęście dla zachowania zgodności ze standardem ANSI muszą one obsługiwać najważniejsze słowa kluczowe w bardzo podobny sposób. Mowa tu jest o takich słowach (poleceniach) jak Select, Update, Insert, Delete, Where itd.

3.1. Pojęcia podstawowe

Formalnie SQL jest zapytaniem skierowanym do bazy danych. Formalnie, bowiem SQL zawiera także polecenia, które pozwalają na aktualizację, wprowadzanie oraz usuwanie rekordów. Polecenia SQL związane zarówno z budowaniem zapytań, jak i te, które pozwalają na szeroko pojętą aktualizację danych są częścią standardowego SQL odpowiedzialnego za manipulację danymi (DML – *Data Manipulation Language*). Są to następujące polecenia:

- `Select` – pozwala na wyciągnięcie z bazy danych potrzebnych informacji (pół),
- `Update` – pozwala na aktualizację danych w tabeli,
- `Delete` – pozwala na usunięcie rekordów z tabeli,
- `Insert into` – pozwala na wprowadzenie nowego rekordu.

Oddzielną grupę poleceń języka SQL tworzą te polecenia, które pozwalają na zarządzanie tabelami, w tym ich tworzeniem i usuwaniem. Polecenia te można zaliczyć do grupy poleceń definiujących dane (DDL – *Data Definition Language*). Poza poleceniami tworzącymi tabele lub je usuwającymi można tu wymienić polecenia tworzące indeksy w tabeli, tworzące połączenia między tabelami (relacje) jak i polecenia wymuszające integralność danych.

Są to następujące polecenia (najważniejsze):

- Create table – polecenie służące do utworzenia nowej tabeli,
- Alter table – polecenie aktualizujące definicję tabeli,
- Drop Table – polecenie usuwające tabelę,
- Create index – polecenie tworzące indeks,
- Drop index – polecenie usuwające indeks.

3.2. Polecenie Select

Polecenie Select wykorzystywane jest do wybrania określonych informacji z jednej lub wielu tabel czy zdefiniowanych wcześniej kwerend (widoków). Zwracana informacja umieszczana jest w tzw. rekordsecie, czyli dynamicznie budowanej tabeli. Na tym dynamicznie zwróconym zestawie rekordów można wykonać szereg działań poprzez wykonanie odpowiedniej metody (np. zmianę wskaźnika rekordu bieżącego), można także odczytać szereg jego własności poprzez odwołanie się do odpowiedniej właściwości (np. odczytać liczbę zwróconych rekordów).

Podstawowa składnia polecenia Select zwracająca informacje z pojedynczej tabeli pokazana jest niżej.

```
Select [distinct] nazwa_pola_1, nazwa_pola_2, ...,
nazwa_pola_k from nazwa_tabeli
```

Powiedzmy, że w naszej bazie jest tabela o nazwie Studenci zawierająca pokazane niżej dane.

Id_s	Nazwisko	Imie1	Imie2	DataUrodzenia
2	Kowalski	Jan		1982-12-13
4	Pac	Adam	Jan	1979-11-23
6	Abacki	Adam	Waclaw	1984-03-12
18	Bazylik	Anna	Maria	1986-07-15

Chcemy napisać polecenie SQL zwracające z tej tabeli nazwisko i imię (pierwsze) studenta. Polecenie SQL będzie mieć postać:

```
Select Nazwisko, Imie1 from Studenci
```

Uruchomienie tego polecenia zwróci rekordset zawierający następujące dane:

Nazwisko	Imie1
Kowalski	Jan
Pac	Adam
Abacki	Adam
Bazylik	Anna

Polecenie zwracające imiona (pierwsze) studentów, których dane znajdują się w tabeli Studenci będzie miało składnię:

```
Select imiel from Studenci
```

Jego uruchomienie spowoduje utworzenie rekordsetu zawierającego następujące dane:

Imiel
Jan
Adam
Adam
Anna

Jak widzimy, mamy podwójnie zwrócone imię „Adam”, jeżeli chcemy uniknąć takich sytuacji, inaczej mówiąc chcemy mieć informacje unikalne, to musimy zmodyfikować polecenie Select poprzez dodanie klauzuli Distinct:

```
Select Distinct imiel from Studenci
```

Teraz po uruchomieniu tak zmodyfikowanego polecenia rekordset będzie zawierał jedynie unikalne imiona.

Imiel
Jan
Adam
Anna

Klauzula Join

W tych przypadkach, gdy polecenie Select ma zwrócić informacje z więcej niż jeden tabeli, to jego składnia jest bardziej skomplikowana. Tym razem nazwa każdego zwracanego pola **musi** być poprzedzona nazwą tabeli i kropką, musi być także przekazana informacja o relacjach między tabelami. Do opisania relacji wykorzystywana jest jedna z trzech klauzul: Inner Join | Right Join | Left Join oraz klauzula On dla wskazania pól wykorzystanych w relacji (poniżej z klauzulą Inner Join).

```
Select [Distinct] tabela1.pole1, tabela1.pole2,
       tabela2.pole 1 from tabelal Inner Join tabela2 On
       tabela1.pole0 = tabela2.pole3
```

Powiedzmy, że w naszej bazie mamy dwie tabele zawierające pokazane niżej informacje.

tKlienci		
Id_k	Nazwisko	Imie
1	Kowalski	Adam
2	Abacki	Jan
3	Pac	Witold

tTelefony		
Id_t	Id_k	NumerT
1	1	(46) 846 45 58
2	1	602 234 129
3	3	(22) 234 56 32

Powiedzmy dalej, że tabele te połączone są relacją *jeden-do-wielu* poprzez pola `id_k` w tabeli `tKlienci` oraz pole `id_k` w tabeli `Telefony`. Chcemy napisać takie zapytanie, aby w wyniku uzyskać nazwisko i imię klienta (z `tKlienci`) oraz jego numery telefonów z tabeli `tTelefony`.

```
SELECT tKlienci.Nazwisko, tKlienci.Imie, tTelefony.Numer
FROM tKlienci INNER JOIN tTelefony ON tKlienci.ID_K =
tTelefony.ID_K
```

Uruchomienie tego polecenia zwróci pokazany niżej rekordset zawierający trzy pola wybrane z obu tabel.

Nazwisko	Imie	Numer
Kowalski	Adam	(46) 846 45 58
Kowalski	Adam	602 234 129
Pac	Witold	(22) 234 56 32

Użycie klauzuli `Inner Join` w definicji relacji oznacza, że z obu połączonych tabel zostaną zwrócone tylko te rekordy, dla których pole łączące ma tę samą wartość w obu tabelach.

Jeżeli chcemy mieć zwrócone wszystkie nazwiska i imiona klientów niezależnie od tego, czy mają zarejestrowany numer telefonu czy też nie, to musimy użyć klauzuli `Left Join`:

```
SELECT tKlient.Nazwisko, tKlient.Imie, tTelefony.Numer
FROM tKlient LEFT JOIN tTelefony ON tKlient.id_k =
tTelefony.id_k
```

Nazwisko	Imie	Numer
Kowalski	Adam	(46) 846 45 58
Kowalski	Adam	602 234 129
Abacki	Jan	
Pac	Witold	(22) 234 56 32

Klauzula `Right Join` da w naszym przykładzie taki sam efekt, jak w przypadku użycia klauzuli `Inner Join`.

```
SELECT tKlient.Nazwisko, tKlient.Imie, tTelefony.Numer
FROM tKlient RIGHT JOIN tTelefony ON tKlient.id_k =
tTelefony.id_k
```

Dokładnie `Right Join` w powyższym zapytaniu oznacza: „zwrócić wszystkie rekordy z tabeli `tTelefony` i tylko te rekordy z tabeli `tKlient`, dla których sprzedane pola są równe”. Ta zgodność wynika z faktu, że między obu tabelami została zdefiniowana

relacja typu *jeden-do-wielu* z wymuszaniem więzów integralności, co oznacza, że w tabeli tTelefony może być tylko taki rekord, którego pole id_k zawiera swój odpowiednik w tabeli tKlienci.

Gdyby takiego warunku nie było, to oczywiście zapytanie z każdą z klauzul będzie zwracało inny zestaw rekordów. Poniżej taki zestaw dla klauzuli Right Join w przypadku, gdy utworzono relację bez wymuszania więzów integralności.

Nazwisko	Imię	Numer
		456788
Kowalski	Adam	(46) 846 45 58
Kowalski	Adam	602 234 129
Pac	Witold	(22) 234 56 32

Pierwszy zwrócony rekord nie ma swojego odpowiednika w tabeli tKlienci, a został zwrócony dlatego, że użyto klauzuli Right Join.

Klauzula As

Polecenie Select może także zwracać pole wyliczane, przykładem niech będzie modyfikacja poprzedniego polecenia w taki sposób, aby zwrócić nazwisko i imię klienta w jednym (nowym) polu o nazwie np. Klient.

```
SELECT tklienci.Nazwisko & " " & tklienci.Imie AS Klient,
       tTelefony.Numer FROM tKlienci INNER JOIN tTelefony ON
       tKlienci.ID_K = tTelefony.ID_K
```

W zależności od wersji SQL akceptowanej przez konkretny SZRBD sposób połączenia informacji typu tekstowego może być taki, jak pokazano to wyżej – użycie operatora & do „dodawania” tekstów i symboli cudzysłowni do jawnego poinformowania, że chodzi o informację typu tekstowego. Taka składnia obowiązuje np. w MS Access. Z kolei w takim systemie jak MS SQL Server do łączenia tekstów użyjemy zwykłego operatora +, a informacja typu tekstowego będzie zapisywana w apostrofach. Poniżej pokazany jest przykład takiego polecenia.

```
SELECT tklienci.Nazwisko + ' ' + tklienci.Imie AS Klient,
       tTelefony.Numer FROM tKlienci INNER JOIN tTelefony ON
       tKlienci.ID_K = tTelefony.ID_K
```

W przypadku tworzenia pól wyliczanych niejako obowiązkowo musimy zastosować słowo kluczowe As do zdefiniowania **aliasu** takiego pola. W pokazanym przykładzie zostanie zwrócona pokazana niżej informacja.

Klient	Numer
Kowalski Adam	(46) 846 45 58
Kowalski Adam	602 234 129
Pac Witold	(22) 234 56 32

Klauzula Where

Polecenie Select może zostać uzupełnione o klauzulę Where określającą warunek lub grupę warunków, jakie muszą być spełnione przez zwracane rekordy. Na kilku kolejnych przykładach pokażemy budowanie warunków z wykorzystaniem różnych operatorów.

Pokazany niżej polecenie zwróci te rekordy z tabeli tKlienci, dla których nazwisko lub imię klienta zaczyna się literą „A*” (wersja MS Access).

```
Select * from tKlienci Where imie Like "A*" or nazwisko Like "A*"
```

W powyższym zapytaniu symbol * przed słowem kluczowym From oznacza “zwróć wszystkie pola”. Operator Like można rozumieć jako „zgodny ze wzorcem”, stąd zapis imie Like „A*” będziemy rozumieć następująco: imię zaczyna się literą „A”, ponieważ wzorecz „A*” oznacza dowolny ciąg znaków zaczynający się dużą literą „A”. Analogiczny warunek został sformułowany odnośnie nazwiska klienta. Oba warunki zostały połączone operatorem or, tym samym zostaną zwrócone te rekordy, w których imię klienta lub jego nazwisko zaczynają się literą „A”.

Uruchomienie tego zapytania zwraca niżej pokazany zestaw rekordów

id_k	Nazwisko	Imie
1	Kowalski	Adam
2	Abacki	Jan

Kolejny przykład zwraca nazwiska i imiona klientów posiadających telefony komórkowe w sieci Era (zaczynające się od 60).

```
SELECT tklienci.Nazwisko + ' ' + tklienci.Imie AS Klient,
tTelefony.Numer FROM tKlienci INNER JOIN tTelefony ON
tKlienci.ID_K = tTelefony.ID_K where Telefony.Numer
Like "60*"
```

klient	Numer
Kowalski Adam	602 234 129

Dla zilustrowania działania operatora and powiedzmy, że chcemy zwrócić tych klientów, których nazwisko zaczyna się na literę „A”, a imię kończy literą „n”. Zapytanie będzie mieć postać:

```
Select * from tKlienci Where imie Like "*n" and nazwisko
Like "A*"
```

Uruchomienie tego zapytania zwróci (w naszym przykładzie) jeden rekord.

id_k	Nazwisko	Imie
2	Abacki	Jan

Z klawulą Where mogą wystąpić podane niżej operatory logiczne, pierwsze dwa mogą być wykorzystywane do badania warunków numerycznych i tekstowych, ostatni wyłącznie do budowania warunków tekstowych.

Operator	Opis
=	równa się
<>	nierówne
>	większe
<	mniejsze
>=	większe równe (nie mniejsze)
<=	mniejsze równe (nie większe)
Between	zawarte między
In	zawarte w (lista wartości)
Like	zgodne ze wzorcem

Dla ilustracji zastosowania kilku z wymienionych operatorów powiedzmy, że tabela tKlienci zawiera następujące dane:

Id_k	Nazwisko	Imie	DataUrodzenia
1	Kowalski	Adam	1972-12-14
2	Abacki	Jan	1984-05-23
3	Pac	Witold	1972-03-11
4	Kamiński	Zygmunt	1986-07-25

Zapytanie:

```
SELECT Nazwisko, Imie, DataUrodzenia FROM Klient WHERE
DataUrodzenia Between #1972-01-01# And #1984-12-31#
```

zwraca zestaw rekordów zawierający te rekordy, dla których data urodzenia zawarta jest między podanymi datami.

Nazwisko	Imie	DataUrodzenia
Kowalski	Adam	1972-12-14
Abacki	Jan	1984-05-23
Pac	Witold	1972-03-11

Jeżeli chcemy, aby zapytanie zwróciło dane tych klientów, którzy urodzili się np. w 1972 roku, to warunek musi być odnoszony do roku, a nie do samej daty urodzenia. Problem rozwiążemy poprzez umieszczenie w zapytaniu funkcji Year, która zwraca liczbę reprezentującą rok z argumentu tej funkcji.

```
SELECT Nazwisko, Imie, DataUrodzenia FROM Klient where  
year(DataUrodzenia)=1972
```

Nazwisko	Imie	DataUrodzenia
Kowalski	Adam	1972-12-14
Pac	Witold	1972-03-11

W definicji zapytania można wykorzystać klauzulę In (także Not In) w tych wszystkich przypadkach, gdy znamy dokładne wartości kryterium, które chcemy sformułować, zwłaszcza wtedy, gdy tych wartości jest dużo. Poniższy przykład zwraca dwa pola z tabeli Klient, dla których pole **id_k jest równe** 1 lub 3.

```
SELECT Klient.id_k, Klient.nazwa FROM Klient where  
Klient.id_k In (1,3)
```

To samo zapytanie zmodyfikowane o słowo Not przed In spowoduje zwrócenie tych rekordów, dla których pole **id_k nie jest równe** liczbie 1 ani liczbie 3.

```
SELECT Klient.id_k, Klient.nazwa FROM Klient where  
Klient.id_k Not In (1,3)
```

Jak widzimy z tych dwóch przykładów użycie operatora In jest równoważne serii operatorów Or, podobnie użycie Not In jest równoważne serii operatorów And.

Na zakończenie jeszcze jedno zapytanie, tym razem chcemy zwrócić te rekordy, które dotyczą klientów urodzonych co najmniej w miesiącu lipcu.

```
SELECT Nazwisko, Imie, DataUrodzenia FROM Klient where  
Month(DataUrodzenia)>=7
```

Tym razem została wykorzystana funkcja Month, która zwraca liczbę reprezentującą miesiąc z argumentu tej funkcji.

Nazwisko	Imie	DataUrodzenia
Kowalski	Adam	1972-12-14
Kamiński	Zygmunt	1986-07-25

Klauzula Order by

Polecenie Select może być także uzupełnione o klauzulę Order by determinującą sposób sortowania zwracanych rekordów. Przykładowo zapytanie:

```
SELECT Nazwisko, Imie, DataUrodzenia FROM Klient WHERE  
DataUrodzenia Between #1972-01-01# And #1984-12-31# Order by  
Nazwisko
```

spowoduje zwrócenie rekordów posortowanych rosnąco wg pola Nazwisko.

Nazwisko	Imie	DataUrodzenia
Abacki	Jan	1984-05-23
Kowalski	Adam	1972-12-14
Pac	Witold	1972-03-11

W przypadku, gdyby nam zależało na odwrotnym sortowaniu, to po nazwie pola wg którego sortujemy, musimy dopisać słowo kluczowe Desc.

```
SELECT Nazwisko, Imie, DataUrodzenia FROM Klient WHERE
DataUrodzenia Between #1972-01-01# And #1984-12-31# Order by
Nazwisko Desc
```

Nazwisko	Imie	DataUrodzenia
Pac	Witold	1972-03-11
Kowalski	Adam	1972-12-14
Abacki	Jan	1984-05-23

W przypadku sortowania rosnącego można było dopisać słowo kluczowe Asc, ale nie jest to konieczne. Po prostu domyślnie przyjmowane jest sortowanie rosnące.

3.2. Polecenie Insert into

Polecenie (instrukcja) Insert into wykorzystywana jest do wprowadzania do tabeli nowego rekordu lub rekordów danych. Formalna składnia jest następująca:

```
Insert into nazwa_tabeli (pole_1, pole_2, ..., pole_k)
values (wartosc_1, wartosc_2, ..., wartosc_k)
```

Liczba pól tabeli wymienionych w nawiasach po jej nazwie musi być równa liczbie wartości wyszczególnionych w nawiasie po słowie kluczowym values. W przypadku wstawiania informacji tekstowych lub daty trzeba obowiązkowo zatrzymać ją w apostrofach.

Powiedzmy, że do tabeli tKlienci chcemy dodać nowy rekord opisujący klienta Nowak Kazimierz, urodzony 1978-02-13. Polecenie Insert będzie mieć postać:

```
Insert into tKlienci (Nazwisko, Imie, DataUrodzenia)
values ('Nowak', 'Kazimierz', '1978-02-13')
```

W dalszej części tego skryptu będziemy wstawiać nowe rekordy po pobraniu wszystkich lub części danych z odpowiedniego formularza. Powiedzmy, że mamy formularz o nazwie frmKlienci, na tym formularzu są trzy pola tekstowe o nazwach odpowiednio txtNazwisko, txtImie, txtDataUrodzenia. Jest także przycisk polecenia o nazwie cmdZapisz. Klik tego przycisku uruchomi kod procedury zdarzeniowej, której zadaniem będzie dopisanie nowego rekordu do tabeli tKlienci, z tym, że

wartości do wprowadzenia muszą być pobrane z formy. W takiej sytuacji polecenie Insert przypisane do zmiennej testowej txtSQL będzie mieć postać:

```
txtSQL = "Insert into tKlienci (Nazwisko, Imie, " & _
DataUrodzenia) values ('" & Me.txtNazwisko & "','" & _
Me.txtImie & "','" & Me.txtDataUrodzenia & "')"
```

3.3. Polecenie Update

Polecenie (instrukcja) Update stosowana jest do modyfikacji jednego lub większej liczby pól wskazanej tabeli bazy danych, przy czym działanie dotyczy tych rekordów, które spełniają zdefiniowany warunek.

```
UPDATE nazwa_tabli SET nazwa_kolumny1 = nowa_wartość
[, nazwa_kolumny2 = nowa_wartość, ...]
WHERE warunek
```

W poniższym przykładzie instrukcja Update została wykorzystana do uzupełnienia pola NumerTelefonu o numer kierunkowy dla wybranego klienta.

```
UPDATE TelefonyKlient SET TelefonyKlient.NumerTelefonu =
"(42)" & TelefonyKlient.NumerTelefonu WHERE
TelefonyKlient.id_k =1
```

3.4. Polecenie Delete

Polecenie (instrukcja) Delete jest wykorzystywana do usunięcia ze wskazanej tabeli określonej warunkiem grupy rekordów (także wszystkich).

```
Delete nazwa_tabeli Where warunek
```

W poniższym przykładzie polecenie Delete spowoduje usunięcie z tabeli Faktura tych rekordów, które dotyczą klienta o identyfikatorze (id_k) równym 1.

```
DELETE FROM Faktura WHERE Faktura.id_k = 1
```

Użycie instrukcji Delete bez sformułowania warunku jest równoważne usunięciu wszystkich rekordów z danej tabeli.

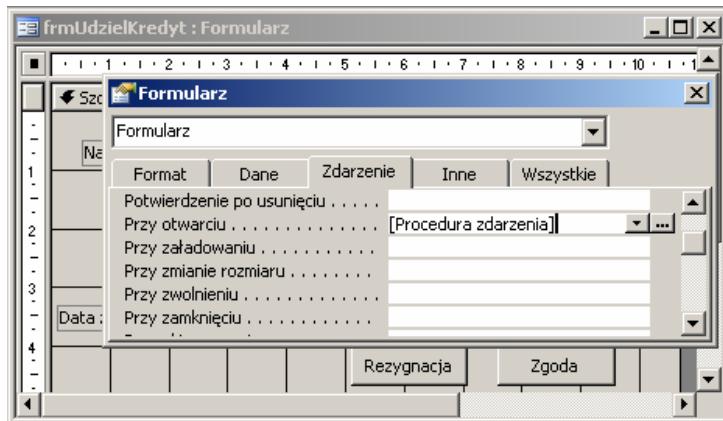
W dalszej rozdziałach tej książki zainteresowany Czytelnik znajdzie bardzo wiele przykładów użycia instrukcji języka SQL, będą tam zarówno proste zapytania jak i dość znacznie skomplikowane. Ich dokładna analiza pozwoli na zgłębienie tajemnic i poszerzenie Twojej wiedzy o tym, jakże przydatnym, języku dostępu do danych.

4. VBA w MS Access

4.1. Pojęcia podstawowe

Visual Basic for Applications (VBA) jest standardowym językiem makropoleceń dodawanym przez firmę Microsoft do praktycznie wszystkich składników pakietu biurowego MS Office. Przy jego pomocy można w stosunkowo prosty sposób znakomicie zwiększyć funkcjonalność aplikacji tworzących ten pakiet biurowy. W swojej zasadniczej części język VBA jest wspólny we wszystkich aplikacjach, jest także wspólny z typowym językiem programowania Visual Basic 6.0. Dokładny opis podstawowych struktur języka VBA można znaleźć w wielu pozycjach, także w pozycji wydanej w ramach tej serii wydawniczej². Z tego też powodu w tej pozycji zostaną omówione dokładniej jedynie te elementy języka VBA, które są specyficzne dla MS Access, a dotyczą programowania baz danych (model ADO).

W trakcie projektowania dowolnego formularza może zajść potrzeba przygotowania procedury reagującej na jakąś określzoną sytuację. Przygotowanie takiej procedury zaczynamy od przejścia do zakładki *Zdarzenia* w oknie projektu formularza. W kolejnym kroku dla określonego zdarzenia wybieramy opcję *[Procedura zdarzenia]* i poprzez klik przycisku z trzema kropkami przechodzimy do okna edytora VBA.



Okno edytora VBA możemy także otworzyć poprzez wywołanie odpowiedniego polecenia z menu *Narzędzia* i dalej *Makro* oraz *Edytor Visual Basic*, można także skorzystać z sekwencji klawiszy Alt+F11.

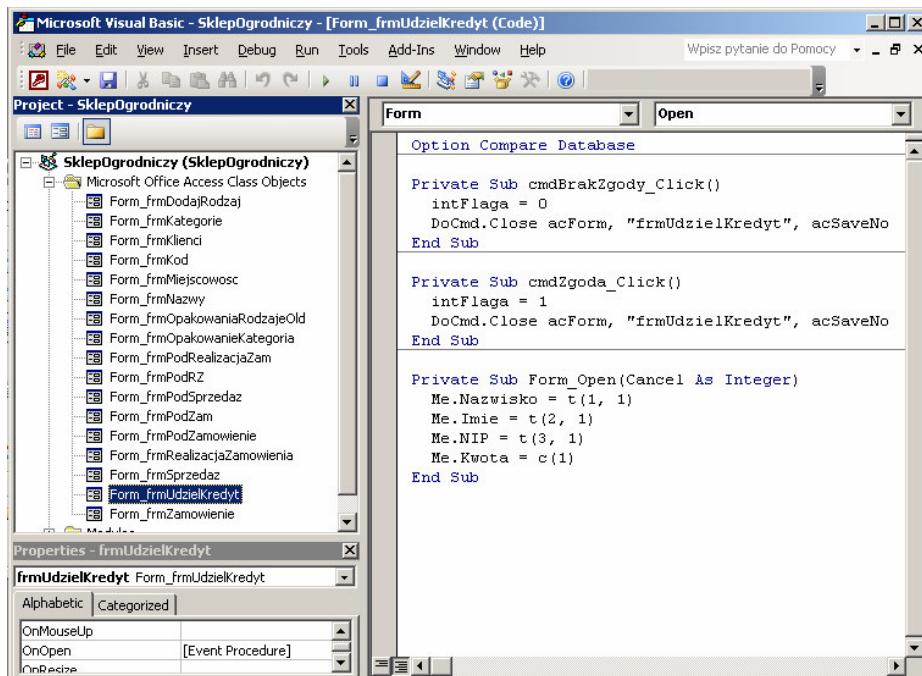
² J. Górczyński, *Makropolecenia i aplikacje VBA w MS Excel*, WSZiM Sochaczew, 2003

Widok, jaki zobaczymy po przejściu do edytora VBA zależy w dużym stopniu od naszych ustawień, w pokazanej niżej sytuacji pokazane są trzy okna:

Project Explorer – pokazuje organizację kodu w moduły w danym projekcie,

Properties – pokazuje właściwości aktywnego (zaznaczonego) obiektu,

Okno procedur – pokazuje procedury w danym module.



4.1.1. Zmienne i stałe

Jak wspomnieliśmy wcześniej cały kod grupowany jest w tzw. moduły odpowiadające utworzonym formularzom czy raportom, poza nimi mogą być jeszcze moduły ogólne oraz moduły klas. W ramach modułu kod grupowany jest w procedury lub funkcje, może być także umieszczony w tzw. sekcji deklaracji, gdzie deklaruje zmienne i stałe, które będziemy wykorzystywać w kodzie. W sekcji deklaracji mogą się także pojawić tzw. dyrektywy ustalające sposób pracy kompilatora języka VBA.

Z pojęciem modułów ściśle związane są takie pojęcia jak zasięg zmiennych i procedur. Procedury i funkcje zdefiniowane na poziomie modułu formularza (raportu) dostępne są tylko wewnątrz tego modułu. Z kolei procedury i funkcje zdefiniowane w modułach

ogólnych są dostępne dla wszystkich innych procedur w danym projekcie. Podobne regulacje dotyczą także deklaracji zmiennych. Zmienne zadeklarowane w sekcji deklaracji modułu ogólnego są dostępne dla wszystkich procedur i funkcji projektu, zmienne deklarowane w sekcji deklaracji modułu formy czy raportu są dostępne dla wszystkich procedur czy funkcji tego modułu. Najmniejszy zasięg mają zmienne deklarowane wewnętrz procedury czy funkcji, są bowiem dostępne wyłącznie wewnętrz tej procedury (funkcji).

Do zadeklarowania zmiennych będziemy używać jednego z trzech słów kluczowych: `Dim`, `Private` lub `Public`. Najprostsza deklaracja zmiennej składa się właśnie z jednego z tych słów, nazwy zmiennej oraz określenia jej typu. Przykładowo instrukcja:

```
Dim ileR as integer
```

deklaruje zmienną o nazwie `ileR`, która będzie przechowywać dane typu *liczba całkowita*. W przypadku rozpoczęcia deklaracji zmiennej za pomocą słowa kluczowego `Dim` jej zasięg regulowany jest wyłącznie miejscem deklaracji.

Zmienna może być zadeklarowana słowem kluczowym `Private` lub `Public`, ale **wyłącznie** w sekcji deklaracji modułu (a nie w procedurze czy funkcji). Zmienne deklarowane przy pomocy słowa `Private` są dostępne jedynie dla procedur czy funkcji w tym module. Deklaracja zmiennych przy pomocy słowa `Public` oznacza, że zmienna taka jest dostępna dla wszystkich procedur i funkcji we wszystkich aplikacjach uruchomionych w danym momencie.

Deklaracja może dotyczyć zmiennej skalarnej (pojedynczo wartości), może być także deklaracją zmiennej tablicowej (macierzowej). Przykładowo zapis:

```
Dim X(10, 2) as Currency
```

jest (domyślnie) deklaracją zmiennej o 11 wierszach i 3 kolumnach typu *walutowego*. Domyślnie zmienne macierzowe są indeksowane od zera, można zmienić tę regułę dyrektywą `Option Base 1` umieszczoną w sekcji deklaracji modułu. Tym samym zapis:

```
Option Base 1  
Dim X(10, 2) as Currency
```

oznacza deklarację zmiennej tablicowej o 10 wierszach i 2 kolumnach.

Zmienna tablicowa może być także zadeklarowana jako zmienna o nieznanych (na etapie deklaracji) wymiarach. Zapis

```
Dim Z() as Double
```

jest deklaracją zmiennej tablicowej `Z` o nieustalonych wymiarach przechowującą dane liczbowe typu *liczby rzeczywistej o podwójnej precyzji*. Przed pierwszym użyciem tak

zadeklarowanej zmiennej **musi** nastąpić jej zwymiarowanie za pomocą instrukcji ReDim. Przykładowo zapis

```
ReDim Z(20, 1)
```

ustala wymiar zmiennej macierzowej z na 21 wierszy i 2 kolumny (przy domyślnej dyrektywie Option Base 0).

Deklaracja może dotyczyć także tzw. obiektu, np. formularza, połączenia z bazą danych, skoroszytu w Excelu, zestawu rekordów zwracanego przez kwerendę czy klasy (pewnego rodzaju zmiennej wraz z jej właściwościami i procedurami). Przykładowo deklaracja

```
Dim con as ADODB.Connection
```

jest deklaracją zmiennej obiektowej con, która będzie zawierać informacje niezbędne do nawiązania połączenia z bazą danych (ogólnie ze źródłem danych), Zmienna obiektowa zadeklarowana w ten sposób nie jest jeszcze gotowa do przypisania jej wartości, musimy użyć instrukcji Set i New do utworzenia nowej instancji tej zmiennej.

```
Set con = New ADODB.Connection
```

Utworzenie nowej instancji zmiennej obiektowej można zadeklarować bezpośrednio w deklaracji poprzez dodanie słowa kluczowego New przed typem obiektu:

```
Dim con as New ADODB.Connection
```

W aplikacji SklepOgrodniczy.mdb będziemy chcieli przekazać dane personalne klienta, jego numer NIP oraz wartość dokonanych zakupów z jednego formularza do drugiego. Można to zrobić na wiele sposobów, np. deklarując potrzebne zmienne w module ogólnym, można także utworzyć klasę opisującą klienta i przekazać potrzebne zmienne przez obiekt tego typu³. W tym przypadku w module ogólnym zadeklarowano zmienną obiektową typu cKlient, gdzie cKlient jest nazwą klasy opisującej klienta.

```
Public DaneK As cKlient
```

W module pierwszego formularza (frmSprzedaz) utworzono nową instancję tej zmiennej

```
Set DaneK = New cKlient
```

przypisując do niej potrzebne dane.

```
DaneK.Nazwisko = rst!Nazwisko  
DaneK.Imie = rst!Imie  
DaneK.Nip = rst!Nip  
DaneK.Kwota = Me.Tekst5
```

³ Zobacz rozdział 6.4.3, strony 293, 295

W innym formularzu (`frmUdzierKredyt`) dane zawarte w zmiennej `DaneK` zostały wykorzystane do nadania wartości obiektom tego formularza.

```
Me.Nazwisko = DaneK.Nazwisko
Me.Imie = DaneK.Imie
Me.Nip = DaneK.Nip
Me.Kwota = DaneK.Kwota
```

Zmienna obiektowa zajmuje pewien obszar pamięci, trzeba też pamiętać o tym, że każda instrukcja typu `Set` tworzy nową instancję tej zmiennej (zajmując pamięć). Oznacza to, że zmienna typu obiektowego powinna być po wykorzystaniu usuwana, co spowoduje zwolnienie pamięci komputera. Podana niżej instrukcja niszczy zmienną obiektową poprzez przypisanie jej wartości `Nothing`.

```
Set DaneK = Nothing
```

Deklaracja zmiennej może też wskazać na typ danych zdefiniowany wcześniej przez użytkownika. Do deklaracji zmiennej użytkownika wykorzystywane są słowa kluczowe `Type ... End Type`, a między nimi są nazwy składowych i typ danych przechowywanych w składowej. Przykładowo, w aplikacji `SklepOgrodnicy.mdb` w module ogólnym zdefiniowano typ użytkownika `Slownie` wykorzystywany do uzyskania kwoty słownie danej liczby.

```
Public Type Slownie
    x As Currency
    xd As Integer
    t As String
End Type
```

W funkcji `KwotaSlownie` znajdziemy deklarację zmiennej tego typu:

```
Dim y As Slownie
```

oraz instrukcje przypisujące wartości składowym tej zmiennej jak i odczytujące te wartości.

Pod pojęciem **stałych** będziemy rozumieć nazwane zmienne określonego typu, którym w momencie uruchomienia aplikacji nadawane są określone wartości. Przykładem niech będzie poniższa deklaracja

```
Public Const conKom = "Sklep ogrodniczy"
```

która przy pomocy słowa kluczowego `Const` tworzy stałą typu *tekstowego* reprezentującą ciąg znaków „`Sklep ogrodniczy`”. Zasadnicza różnica między zmienną a stałą polega na tym, że w trakcie działania aplikacji zmienna może mieć wielokrotnie przypisywaną wartość, a stała **tylko raz** w momencie startu aplikacji. Inna różnica dotyczy miejsca deklaracji stałych: nie mogą być deklarowane w procedurach czy funkcjach.

W języku VBA istnieje wiele predefiniowanych stałych, które będziemy wykorzystywać w różnych sytuacjach. Przykładowo, procedura `MsgBox` może wykorzystywać stała `vbInformation`, która oznacza dodanie do okna komunikatu symbolu dużej litery I.

4.1.2. Procedury i funkcje

Kod aplikacji napisanej w VBA składa się, poza wspomnianymi deklaracjami, z wyróżnionych fragmentów instrukcji tworzących **procedury i funkcje**.

Procedura, to nazwany fragment kodu zawarty między słowami kluczowymi Sub ... End Sub, a jej zadaniem jest wykonanie pewnej sekwencji kodu. Przykładem niech będzie procedura uruchamiana w momencie kliku przycisku polecenia cmdZgoda w formularzu frmUstalKredyt aplikacji SklepOgrodnicy.mdb.

```
Private Sub cmdZgoda_Click()
    intFlaga = 1
    DoCmd.Close acForm, "frmUdzielKredyt", acSaveNo
End Sub
```

Procedura może wykonywać swoje działanie na zmiennych zadeklarowanych zewnętrznie, tak jest w pokazanym przykładzie, gdzie procedura modyfikuje zmienną intFlaga zadeklarowaną w module ogólnym tej aplikacji. Do procedury można **przekazać** argumenty, które następnie zostaną wykorzystane w treści procedury.

```
Private Sub PodzielModulo(liczba as double, _
                           d as integer)
    Debug.print liczba, d, liczba mod d
End Sub
```

Po wywołaniu tej procedury z argumentami 14,75 i 4 do okna *Immiediate* edytora VBA zostanie zwrócony pokazany niżej wynik.

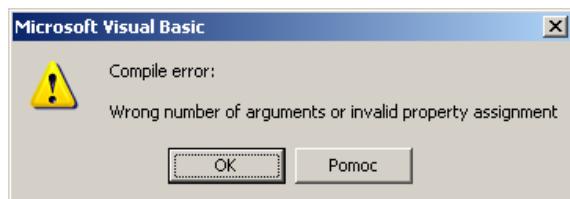
```
Public Sub PodzielModulo(liczba As Double, _
                           d As Integer)
    Debug.Print liczba, d, liczba Mod d
End Sub

Immediate
PodzielModulo 14.75, 4
14,75           4                  3
```

Przy okazji proszę zwrócić uwagę na problem umieszczania w kodzie liczb dziesiętnych, w liczbach tych separatorem kropki dziesiętnej jest kropka, a nie przecinek jak w polskich ustawieniach narodowych. Wywołanie tej procedury w pokazanej niżej postaci:

```
PodzielModulo 14,75, 4
```

wygeneruje błąd wynikający z niezgodności liczby argumentów, o czym poinformuje nas komunikat kompilatora.



Argumenty do procedury mogą być przekazanie dwojako, albo **przez adres** za pomocą słowa kluczowego `ByRef`, albo **przez wartość** za pomocą `ByVal`. Domyślnie przekazanie argumentu jest przez adres. Przekazanie argumentu przez wartość pozwala procedurze na wykorzystywanie aktualnej wartości zmiennej reprezentującej argument. Pokazana niżej procedura `JakDzialaByRef_ByVal` i procedury `DodajByRef` oraz `DodajByVal` demonstrują różnice między tymi dwoma sposobami przekazania argumentów. Proszę zwrócić uwagę, że w procedurze `DodajByRef` konieczne było użycie słowa kluczowego `ByVal` (inaczej byłoby to przekazanie przez referencję).

```
(General) JakDzialaByRef_ByVal
Public Sub DodajByRef(z As Integer)
    z = z + 10
End Sub
Public Sub DodajByVal(ByVal z As Integer)
    z = z + 10
End Sub

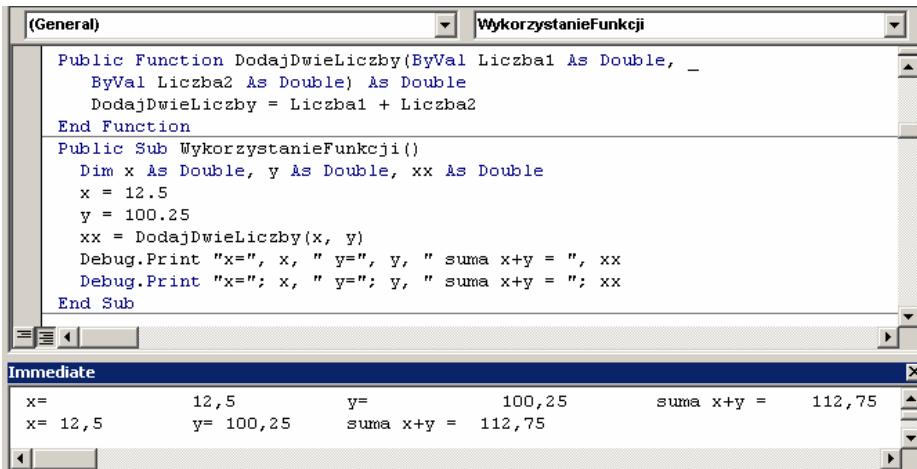
Public Sub JakDzialaByRef_ByVal()
    Dim i As Integer
    i = 5
    Debug.Print "przed wywolaniem i=", i
    DodajByRef i
    Debug.Print "Po DodajByRef() i=", i
    Debug.Print "przed wywolaniem i=", i
    DodajByVal i
    Debug.Print "Po DodajByVal() i=", i
End Sub
```

Immediate

przed wywolaniem i=	5
Po DodajByRef() i=	15
przed wywolaniem i=	15
Po DodajByVal() i=	15

Funkcja to również nazwany fragment kodu zawarty między słowami kluczowymi `Function ... End Function`, ale podstawowa różnica polega na tym, że funkcja w jawny sposób zwraca wartość określonego w jej definicji typu. Podobnie jak w przypadku procedur można do funkcji przekazać argumenty zarówno przez referencję, jak i przez wartość. Domyślnie jest to przekazanie przez referencję.

Poniżej przykład definicji i użycia funkcji DodajDwieLiczby, przy okazji proszę zwrócić uwagę na rezultat zwrócony do okna natychmiastowego. W pierwszym wywołaniu metody Debug.Print jako separatora użyto przecinka, w drugim do oddzielenia grup informacji wykorzystano średnik.



```

(General) WykorzystanieFunkcji
Public Function DodajDwieLiczby(ByVal Liczba1 As Double, _
    ByVal Liczba2 As Double) As Double
    DodajDwieLiczby = Liczba1 + Liczba2
End Function
Public Sub WykorzystanieFunkcji()
    Dim x As Double, y As Double, xx As Double
    x = 12.5
    y = 100.25
    xx = DodajDwieLiczby(x, y)
    Debug.Print "x=", x, " y=", y, " suma x+y = ", xx
    Debug.Print "x="; x, " y="; y, " suma x+y = "; xx
End Sub

```

Immediate			
x=	12,5	y=	100,25
x= 12,5	y= 100,25	suma x+y =	112,75

4.1.3. Klasy

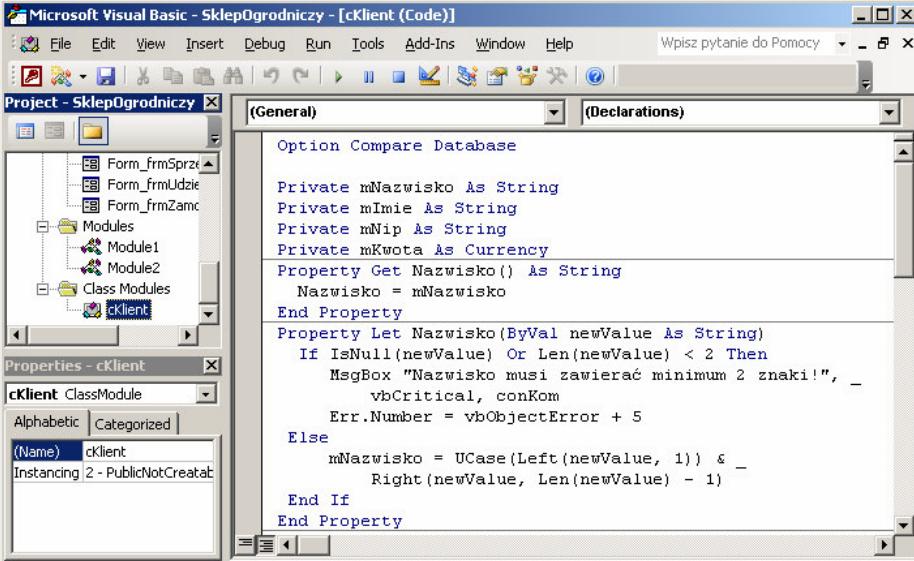
Klasy to we współczesnym, poważnym programowaniu standard. W pewnym sensie klasa jest podobna do zmiennych użytkownika, ale poprzez dołączenie kodu została wzbogacona o **metody** i **właściwości**. Dla ilustracji problemu rozważmy potrzebę definiowania i przechowywania informacji o kliencie zawierającej jego nazwisko, imię, numer NIP oraz jakąś wartość liczbową opisującą jego np. dopuszczalny limit kredytu (jest to nawiązanie do aplikacji Sklep ogrodniczy).

Dane te w pewnym momencie muszą być wprowadzone do aplikacji, ale przed przypisaniem ich do odpowiednich zmiennych konieczne będzie sprawdzenie ich poprawności (walidacja danych). Można to zrobić bezpośrednio w kodzie odpowiedniej procedury, ale znacznie lepszym rozwiązaniem jest zdefiniowanie odpowiedniej klasy i umieszczenie w klasie wszystkich potrzebnych sprawdeń poprawności danych.

Korzystając z menu *Insert* w edytorze VBA możemy do naszego projektu dodać moduł klasy, powiedzmy, że moduł ten otrzyma nazwę cKlient.

W sekcji deklaracji modułu klasy zadeklarujemy teraz cztery zmienne prywatne takiego typu, jaki odpowiada zamierzonej klasie (dokładniej jej **właściwościom**). W naszym przypadku będą to trzy zmienne typu tekstowego i jedna zmienna typu walutowego.

Poniżej pokazane jest okno procedur modułu cKlient z deklaracjami tych czterech zmiennych prywatnych.



The screenshot shows the Microsoft Visual Basic IDE interface. The title bar reads "Microsoft Visual Basic - SklepOgrodniczy - [cKlient (Code)]". The menu bar includes File, Edit, View, Insert, Debug, Run, Tools, Add-Ins, Window, Help, and a search bar "Wpisz pytanie do Pomocy". The toolbar has various icons for file operations. The Project Explorer on the left shows a project named "SklepOgrodniczy" with forms like Form_frmSprz, Form_frmUdzie, Form_frmZamc, and modules Module1, Module2, and the class module cKlient selected. The Properties window on the right shows the class module is named "cKlient" and has "Instancing 2 - PublicNotCreatab". The main code editor window displays the following VBA code:

```

Option Compare Database

Private mNazwisko As String
Private mImie As String
Private mNip As String
Private mKwota As Currency

Property Get Nazwisko() As String
    Nazwisko = mNazwisko
End Property

Property Let Nazwisko(ByVal newValue As String)
    If IsNull(newValue) Or Len(newValue) < 2 Then
        MsgBox "Nazwisko musi zawierać minimum 2 znaki!", _
            vbCritical, conKom
        Err.Number = vbObjectError + 5
    Else
        mNazwisko = UCase(Left(newValue, 1)) & _
            Right(newValue, Len(newValue) - 1)
    End If
End Property

```

Zmienne te są zadeklarowane jako prywatne po to, aby **wyłącznie** za ich pomocą można było komunikować się z wartościami przypisanymi do zmiennej typu cKlient.

Dla każdej ze zmiennych wchodzących w skład klasy tworzona jest **para** procedur Get i Let, pierwsza z nich służy do pobierania danej zmiennej, druga do przypisania wartości danej zmiennej. W pokazanym wyżej kodzie widoczne są te dwie procedury dla zmiennej Nazwisko klasy cKlient.

Procedura Get jest banalna, a jej jedynym zadaniem jest przypisanie zmiennej Nazwisko wartości przechowywanej w zmiennej prywatnej mNazwisko.

Procedura Let jest bardziej skomplikowana, ponieważ **przed** przypisaniem zmiennej prywatnej mNazwisko powinniśmy sprawdzić, czy taka operacja będzie poprawna. W pokazanej sytuacji wykorzystana jest funkcja IsNull oraz funkcja Len do sprawdzania, czy przypadkiem wartość, którą mamy zamiar przypisać zmiennej mNazwisko nie jest nieokreślona albo czy tekst reprezentujący nazwisko nie jest zbyt krótki. Gdyby zaszła taka sytuacja, to zostanie wyświetlony stosowny komunikat (użycie procedury MsgBox) oraz zmienna systemowa opisująca numer błędu Err.Number otrzyma numer błędu ustalony przez użytkownika. Za chwilę powiemy, jak będzie można wykorzystać ten numer błędu do kontrolowania, czy użytkownik podał poprawną informację.

Analogiczne pary procedur Let i Get muszą być utworzone dla pozostałych trzech zmiennych tworzących klasę cKlient.

Moduł klasy może zawierać również inne, „zwykłe” procedury i funkcje, które będą definiować **metody** klasy. Jedną z takich funkcji może być funkcja kontrolująca poprawność wprowadzenia przez użytkownika numeru NIP klienta. Jak wiadomo, numer NIP to zmienna typu tekstowego zgodna z formatem ‘xxx-xxx-xx-xx’, długość tego tekstu to dokładnie 13 znaków. Warunek ten jest badany w procedurze Let tej zmiennej.

```
Property Get Nip() As String
    Nip = mNip
End Property

Property Let Nip(ByVal newValue As String)
    If IsNull(newValue) Or Len(newValue) <> 13 Then
        Err.Number = vbObjectError + 5
        MsgBox "Zle podany numer Nip. Proszę podać" & _
            vbCrLf & _
            "numer zgodnie ze schematem 'xxx-xxx-xx-xx'", & _
            vbCritical, conKom
    Else
        mNip = newValue
    End If
End Property
```

Tworzymy teraz funkcję PobierzNip, jej zadaniem jest poproszenie użytkownika o podanie numeru NIP, ale funkcja będzie zadawać to pytanie tak dugo, dopóki nie otrzyma poprawnej (co do długości) informacji.

```
Public Function PobierzNip() As String
    Do
        Err.Clear
        Nip = InputBox("Nip", conKom)
    Loop Until Err.Number = 0
End Function
```

W funkcji tej wykorzystano w ciekawy sposób pętlę Do...Loop (o pętlach więcej w kolejnym rozdziale) do próby przypisania do zmiennej Nip informacji podanej przez użytkownika w funkcji InputBox. Przed wywołaniem przypisania zerowana jest zmienna systemowa Err po to, aby ewentualny błąd był wygenerowany przez procedurę Let zmiennej Nip. Pętla Do...Loop będzie powtarzana tak dugo (warunek Until), aż procedura Let nie zwróci błędu, czyli wprowadzony przez użytkownika tekst reprezentujący NIP będzie miał 13 znaków. W module ogólnym o nazwie Module2 napisaliśmy prostą procedurę testującą to rozwiązanie.

```

Public Sub TestKlasy()
    ' deklaracja i utworzenie nowej instancji
    ' klasy cKlient
    Dim jg As New cKlient
    ' wywołanie metody PobierzNip klasy cKlient
    jg.PobierzNip
    ' zwrócenie NIP poprzez właściwość klasy
    Debug.Print "Wprowadził NIP: " & jg.Nip
End Sub

```

Uruchomienie tej procedury wyświetla pokazane niżej okno funkcji `InputBox`. Wprowadzona została informacja niezgodna z oczekiwany formatem NIP.



Akceptacja przycisku OK uruchamia procedurę `Let Nip` klasy `cKlient`, zostanie wygenerowany komunikat o błędzie.

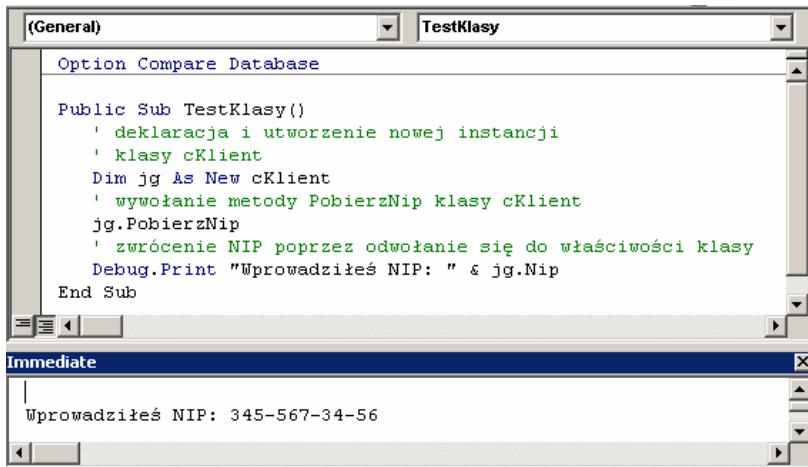


Po akceptacji przycisku OK ponownie zostanie pokazane okno `InputBox`.



Tym razem podana została informacja o (przynajmniej) długości zgodnej z oczekiwanych wzorcem, procedura `Let` nie generuje błędu, pętla `Do...Loop` funkcji

PobierzNip kończy pracę i do okna natychmiastowego zostanie zwrócony wprowadzony numer NIP.

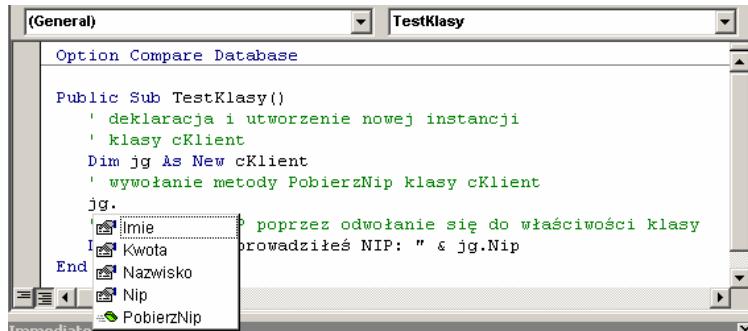


```
(General) TestKlasy
Option Compare Database

Public Sub TestKlasy()
    ' deklaracja i utworzenie nowej instancji
    ' klasy cKlient
    Dim jg As New cKlient
    ' wywołanie metody PobierzNip klasy cKlient
    jg.PobierzNip
    ' zwrócenie NIP poprzez odwołanie się do właściwości klasy
    Debug.Print "Uprowadziłeś NIP: " & jg.Nip
End Sub

Immediate
Urowadziłeś NIP: 345-567-34-56
```

Utworzenie nowej instancji klasy (w tym przypadku `cKlient`) powoduje, że VBA pozwala na wybór metody czy właściwości klasy z listy podpowiedzi. Poniżej pokazana jest sytuacja, jak po wpisaniu nazwy zmiennej `jg` i kropki VBA wyświetlił okno podpowiedzi. Widoczne są właściwości metody oznaczone symbolem ręki trzymającej kartkę oraz metody oznaczone symbolem rzuconej cegły (powiedzmy, że to cegła).



Myślę, że stosunkowo łatwo można sobie wyobrazić bardziej skomplikowane metody walidacyjne – przykładowo dla NIP można sprawdzać, czy użyte symbole to cyfry 0-9 oraz minus jako separator, czy separatory są na swoich miejscach, czy NIP jest poprawny w sensie jego poprawności określonej przez władze skarbowe itd.

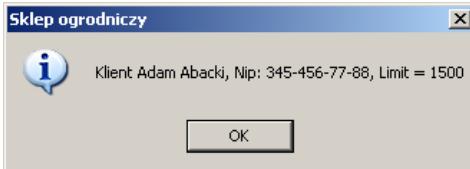
Oczywiście można zwiększyć liczbę metod klasy definiując odpowiednie funkcje. Poniżej pokazana jest metoda (utworzona w module cKlient) zwracająca imię i nazwisko klienta, jego numer NIP oraz dopuszczalny limit kredytu.

```
Public Function FullInfo() As String
    FullInfo = Imie & " " & Nazwisko & ", Nip: " & _
               Nip & ", Limit = " & LTrim(Str(Kwota))
End Function
```

W module ogólnym Module2 została utworzona procedura demonstrująca użycie nowo zdefiniowanej metody.

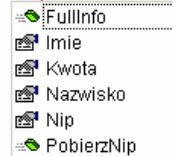
```
Public Sub WywolanieFullInfo()
    Dim nowyKlient As New cKlient
    nowyKlient.Nazwisko = "Abacki"
    nowyKlient.Imie = "Adam"
    nowyKlient.Nip = "345-456-77-88"
    nowyKlient.Kwota = 1500
    MsgBox "Klient " & _
            nowyKlient.FullInfo, vbInformation, conKom
End Sub
```

A tak wygląda wyświetlony komunikat prezentujący informację zwracaną przez metodę FullInfo klasy cKlient.



W trakcie pisania tej procedury nowo utworzona metoda FullInfo jest już dostępna w okienku podpowiedzi edytora VBA.

```
Public Sub WywolanieFullInfo()
    Dim nowyKlient As New cKlient
    nowyKlient.Nazwisko = "Abacki"
    nowyKlient.Imie = "Adam"
    nowyKlient.Nip = "345-456-77-88"
    nowyKlient.Kwota = 1500
    MsgBox "Klient " & nowyKlient.
End Sub
```



4.1.4. Pętle

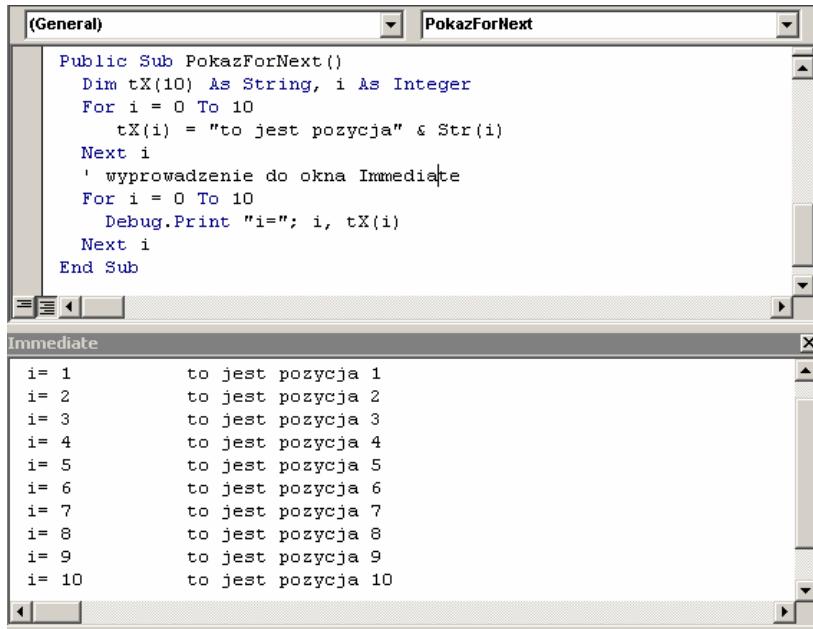
W trakcie tworzenia kodu aplikacji często zdarza się taka sytuacja, że jakiś fragment kodu musi być wykonany określoną liczbę razy lub do spełnienia (lub nie) pewnego warunku. Takie działanie jest realizowane za pomocą tzw. **pętli**. Większość języków programowania oferuje trzy podstawowe rodzaje pętli typu:

For ... Next

Do... Loop

For Each

Pętla **For ... Next** zwana inaczej pętlą z licznikiem ma z góry zadeklarowaną liczbę wykonań. Przykładem takiej pętli może być niżej pokazana, jej zadaniem jest najpierw przypisanie danych do zmiennej tablicowej `tX`, a następnie ich wyprowadzenie do okna *Immediate*.



The screenshot shows two windows from Microsoft Visual Studio. The top window is titled '(General)' and contains the following VBA code:

```

Public Sub PokazForNext()
    Dim tX(10) As String, i As Integer
    For i = 0 To 10
        tX(i) = "to jest pozycja" & Str(i)
    Next i
    ' wyprowadzenie do okna Immediate
    For i = 0 To 10
        Debug.Print "i="; i, tX(i)
    Next i
End Sub

```

The bottom window is titled 'Immediate' and displays the output of the code execution:

```

i= 1      to jest pozycja 1
i= 2      to jest pozycja 2
i= 3      to jest pozycja 3
i= 4      to jest pozycja 4
i= 5      to jest pozycja 5
i= 6      to jest pozycja 6
i= 7      to jest pozycja 7
i= 8      to jest pozycja 8
i= 9      to jest pozycja 9
i= 10     to jest pozycja 10

```

Zmienna `i` w powyższym przykładzie jest tzw. **licznikiem pętli**, jej wartość jest kolejno inkrementowana o 1 (to jest domyślny krok) i o ile aktualna wartość jest nie większa od końcowej wartości (tu 10), to instrukcje zawarte między słowami kluczowymi `For i Next` są **ponownie** wykonywane.

Pętla For ... Next może być warunkowo przerwana, inaczej mówiąc może być wykonana mniejszą liczbę razy niż to wynika z liczby określającej jej zakończenie. Przykładem takiej sytuacji widzimy w pokazanej niżej procedurze.

```
Public Sub ForNext_WarunkoweWyjscie()
    Dim i As Integer, xp As Double, koniec As Integer
    koniec = 10
    xp = 12.5
    For i = 1 To koniec Step 5
        xp = xp + i
        Debug.Print "i="; i, " xp= ", xp
        If xp > 20 Then Exit For
    Next i
End Sub
```

Pętla wykonuje dokładnie dwa przebiegi zwracając do okna *Immediate* następujące wyniki:

i= 1	xp=	13,5
i= 6	xp=	19,5

W sytuacji, gdy nie jesteśmy w stanie z góry określić liczby obrotów pętli, ale potrafimy określić warunek, który musi być spełniony, aby pętla zakończyła swoją pracę, to powinniśmy użyć pętli typu Do ... Loop. Przykład takiej pętli widzieliśmy przy omawianiu funkcji PobierzNip zdefiniowanej w module klasy cKlient.

```
Public Function PobierzNip() As String
    Do
        Err.Clear
        Nip = InputBox("Nip", conKom)
    Loop Until Err.Number = 0
End Function
```

Z uwagi na umieszczenie słowa warunku po słowie Loop pętla taka wykonywana jest co najmniej raz, a powtarzana tak długo, jak długo numer błędu nie będzie zerowy. Inny przykład pokazany jest poniżej.

```
Public Sub PokazDoLoopWersjaWhile()
    Dim xp As Double
    xp = 12.5
    Do While xp <= 20
        Debug.Print "xp="; xp
        xp = xp + 4
    Loop
End Sub
```

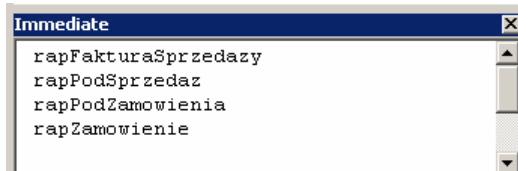
Do okna *Immediate* zostaną zwrócone dwie wartości zmiennej xp, odpowiednio 12,5 i 16,5. Kolejny obrót już nie jest możliwy, bo zmienna xp będzie miała wartość 20,5.

Jednym z istotnych obiektów we współczesnych językach programowania jest pojęcie **kolekcji**, inaczej zbioru elementów tego samego typu. Przykładem może być kolekcja zdefiniowanych kwerend, formularzy czy raportów. Kolekcją jest też zbiór arkuszy w skoroszycie Excela, zbiór akapitów w dokumencie Worda itd. Do przeglądania elementów kolekcji idealnie nadaje się pętla typu *For Each*.

Poniżej pokazana jest procedura zwracająca do okna *Immediate* nazwy wszystkich raportów utworzonych w aplikacji SklepOgrodniczy.mdb.

```
Public Sub PodajNazwyForm()
    Dim f As AccessObject
    For Each f In CurrentProject.AllReports
        Debug.Print f.Name
    Next
End Sub
```

W deklaracji tej procedury zadeklarowano zmienną *f* jako obiekt Accessa, ta zmieniona kolejno będzie wskazywać poszczególne raporty w ich kolekcji, do której odwołujemy się poprzez nazwę *AllReports* w aktualnym projekcie. Właściwość *Name* zmiennej *f* zwraca nazwę aktualnie wskazanego raportu. Poniżej przykładowe nazwy raportów zwrócone po uruchomieniu tej procedury.



I jeszcze jeden przykład użycia pętli *For Each*, tym razem do wyprowadzenia nazw pól i ich typów zdefiniowanych w tabeli *tKlienci*.

```
Public Sub PodajNazwyPol()
    Dim con As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    Dim txt As String, pole As ADODB.Field
    Set con = CurrentProject.Connection
    txt = "select * from tKlienci"
    rst.Open txt, con, adOpenKeyset, adLockReadOnly
    For Each pole In rst.Fields
        Debug.Print pole.Name, pole.Type
    Next
    rst.Close
    con.Close
    Set rst = Nothing
    Set con = Nothing
End Sub
```

4.1.5. Struktury warunkowe

Struktury warunkowe są niezbędne w trakcie tworzenia kodu do zróżnicowania działań zależnie od spełnienia jednego czy większej liczby warunków. Podstawowa struktura warunkowa budowana jest przy pomocy słowa kluczowego `If` badającego spełnienie pewnego warunku. Jeżeli warunek zwraca wartość `True` (prawda), to wykonywany jest odpowiedni fragment kodu, jeżeli nie jest spełniony, to inny fragment kodu.

W pokazanej niżej procedurze `Let Nazwisko` zdefiniowanej w klasie `cKlient` wykorzystano warunek `If` do sprawdzenia, czy zachodzi jedno z dwóch zdarzeń: czy zmienna `newValue` jest zainicjowana (czy nie jest `Null`) oraz czy długość tekstu zapisanego w tej zmiennej jest mniejsza niż 2. Operator `Or` powoduje, że cały warunek jest prawdą wtedy, gdy co najmniej jeden z warunków jest prawdziwy. Jeżeli tak, to wykonywany jest fragment kodu wywołujący procedurę `MsgBox`. Jeżeli warunek nie jest spełniony, to wykonywane jest przypisanie zmiennej prywatnej `mNazwisko` wprowadzowej wartości (po wymuszeniu zapisania tego tekstu z dużej litery). Słowo kluczowe `Else` rozdziela te dwa fragmenty kodu, a słowo kluczowe `End If` zamkna instrukcję `If`.

```
Property Let Nazwisko(ByVal newValue As String)
    If IsNull(newValue) Or Len(newValue) < 2 Then
        MsgBox "Nazwisko musi zawierać minimum 2 znaki!", _
            vbCritical, conKom
        Err.Number = vbObjectError + 5
    Else
        mNazwisko = UCase(Left(newValue, 1)) & _
            Right(newValue, Len(newValue) - 1)
    End If
End Property
```

W sytuacji, gdy trzeba zbadać możliwość wystąpienia więcej niż dwóch warunków struktura `If ... then ... else ... end if` nie jest zbyt efektywna. Znakomicie lepszym rozwiązaniem jest użycie struktury `Select Case ... End Select`.

W pokazanym przykładzie użyto instrukcji `Select Case` do przypisania zmiennej `d` jednej z trzech różnych wartości zależnie od wartości zmiennej `Cena`.

```
Select Case Cena
    Case Is < 50
        d = 2
    Case Is < 100
        d = 1
    Case Else
        d = 0
End Select
```

Mogą się zdarzyć też takie sytuacje, gdy zmienna, od wartości której chcemy uzależnić dalsze działanie przyjmuje wartości tworzące liczby naturalne (1, 2, 3 itd.) bezpośrednio, lub po przekształceniu. W takim przypadku dobrym rozwiązaniem jest wykorzystanie funkcji warunkowej Choose. Przy założeniu, że zmienna fp przyjmuje wartości od 1 do 4 poniższa instrukcja przypisuje zmiennej tekstowej Forma jeden z czterech tekstów jawnie podanych w funkcji Choose.

```
Forma= Choose(fp, "gotówka", "przelew",  
"karta płatnicza", "kredyt kupiecki")
```

W załączonych do tej książki dwóch przykładowych aplikacjach zainteresowany Czytelnik znajdzie wiele przykładów użycia instrukcji VBA do osiągnięcia zamierzonych celów.

4.2. Programowanie baz danych, metoda ADO

Umiejętność programowania baz danych jest dla współczesnego, zaawansowanego użytkownika aplikacji biurowych w zasadzie niezbędna. Jest to o tyle w miarę proste, że mamy do dyspozycji bardzo wygodny model dostępu do danych, model ADO.

ADO jest grupą obiektów zaprojektowanych po to, aby stworzyć programistom prosty i wydajny interfejs dostępu do danych. Historycznie ADO jest kontynuacją takich rozwiązań jak DAO (*Data Access Objects*) czy RDO (*Remote Data Objects*). Oba rozwiązania zostały zaproponowane przez Microsoft, w pewnym momencie okazało się, że ich możliwości są coraz bardziej ograniczone, że są skuteczne tylko wtedy, gdy dane są zapisane w ścisłe określonych formatach. Te ograniczenia były przesłanką do opracowania nowego standardu, znanego pod nazwą OLE DB. Standard ten nie miał już tych ograniczeń co DAO czy RDO, miał za to jedną zasadniczą wadę: był bardzo skomplikowany w zastosowaniach.

Dla rozwiązania tego problemu Microsoft wprowadził model wykorzystujący technologię ActiveX, czyli ADO (*ActiveX Data Objects*). ADO funkcjonuje w warstwie położonej bezpośrednio nad OLE DB, jest interfejsem programistycznym pozwalającym na łatwe korzystanie z OLE DB.

ADO zawiera 6 obiektów i 2 kolekcje, przy czym najważniejszymi obiektami są obiekty Connection i Recordset, te obiekty będą głównie wykorzystywane w prezentowanych aplikacjach i one zostaną dokładniej omówione.

4.2.1. Obiekt Connection

Obiekt Connection definiuje unikalną sesję aplikacji ze źródłem danych. W przypadku aplikacji typu klient-serwer może być równoważny aktualnemu połączeniu sieciowemu do serwera bazodanowego. Zakres możliwości tego obiektu opisany metodami,

właściwościami czy kolekcjami może być różny i jest ściśle uzależniony od dostarczyciela (*Provider*) tego obiektu.

Wykorzystując metody, właściwości i kolekcje obiektu `Connection` mamy (między innymi) możliwość:

- Skonfigurować połączenie przed jego otwarciem wykorzystując takie właściwości jak:
 - `ConnectionString` – informacja typu tekstowego specyfikująca źródło danych,
 - `ConnectionTimeOut` – informacja typu `Long` określająca maksymalny czas oczekiwania na otwarcie połączenia, domyślnie jest to 15 sekund,
 - `Mode` – wskaźnik możliwości modyfikacji właściwości w obiektach typu `Connection`, `Record` i `Stream` modelu ADO.
- Takiego ustawienia właściwości `CursorLocation`, aby było możliwe wykorzystanie biblioteki OLE DB odpowiedzialnej za wsadową aktualizację danych,
- Wskazania domyślnej bazy danych dla połączenia poprzez wykorzystanie właściwości `DefaultDatabase`,
- Ustawienia poziomu izolacji dla transakcji otwieranej na danym połączeniu poprzez właściwość `IsolationLevel`,
- Wskazanie dostawcy OLE DB we właściwości `Provider`,
- Zestawienie fizycznego połączenia ze źródłem danych przy pomocy metody `Open`, oraz zamknięcie takiego otwartego połączenia poprzez wykonanie metody `Close`,
- Wykonanie polecenia (obiekt `Command`) poprzez wywołanie metody `Execute`, a poprzez właściwość `CommandTimeout` konfigurowanie tego wykonania. Dla wykonania kwerendy nie jest konieczne wykorzystanie `Command`, wystarczy w `Execute` umieścić string polecenia SQL. Wykorzystanie `Command` jest jednak niezbędne w tych sytuacjach, gdy chcemy dane zapytanie wywoływać wielokrotnie, albo wtedy, gdy chcemy wykonać zapytanie z parametrem,
- Zarządzać transakcjami, w tym także zagnieżdżonymi za pomocą metod `BeginTrans`, `CommitTrans` i `RollbackTrans` oraz właściwością `Attributes`,
- Śledzić błędy zwrócone przez źródło danych przy pomocy kolekcji `Errors`.

Poniżej pokazana jest procedura, która otwiera połączenie do bazy danych o nazwie TestC.mdb podając nazwę dostawcy (Microsoft.Jet to silnik bazy danych Access) i opisując źródło danych (pełna ścieżka i nazwa pliku bazy danych). Na otwartym połączaniu budowany jest dynamiczny zestaw rekordów (tzw. rekordset) odpowiadający, w pokazanym przykładzie, tabeli Powiaty źródła danych. Rekordsetom poświęcony będzie kolejny rozdział tej książki.

```
Public Sub jj()
    Dim cn As New ADODB.Connection
    cn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source= c:\januszg\access8Listopad\TestC.mdb"
    Dim rs As New ADODB.Recordset
    rs.Open "Powiaty", cn, adOpenKeyset, adLockReadOnly
    MsgBox rs.RecordCount
End Sub
```

Inny przykład użycia metody ADO do uzyskania połączenia z bazą danych MS SQL pokazuje kolejna procedura. Przy jej pomocy uzyskano dostęp do bazy SQL z poziomu arkusza kalkulacyjnego Excel.

```
Public Sub DajInfo(id As Integer)
    Dim con As ADODB.Connection, txt As String
    Dim rst As ADODB.Recordset
    txt = "Provider=Microsoft.Access.OLEDB.10.0;" & _
        "Persist Security Info=False;" & _
        "Data Source=boss;" & _
        "Integrated Security=SSPI;" & _
        "Initial Catalog=ProjektWSZiM;" & _
        "Data Provider=SQLOLEDB.1"
    Set con = New ADODB.Connection
    con.Open txt
    (dalszy kod tej procedury)
End Sub
```

W bazach danych przedstawionych w tej książce wielokrotnie będzie wykorzystywana metoda ADO do odwołania się do tabel danej bazy. W takim przypadku obiekt Connection będzie definiowany w oparciu o dane pobierane z bieżącego projektu. Stosowny przykład pokazany jest poniżej.

```
Public Sub ADO_lokalnie()
    Dim con As New ADODB.Connection, txt As String
    Dim rs As New ADODB.Recordset
    Set con = CurrentProject.Connection
    rs.Open "tKlient", con, adOpenKeyset, adLockReadOnly
    MsgBox rs.RecordCount
End Sub
```

4.2.2. Obiekt Recordset

Obiekt recordsetu reprezentuje pełny zestaw rekordów zwrócony z tabeli bazy danych lub będący efektem wykonania polecenia. W każdym momencie recordset wskazuje jeden pojedynczy rekord w zestawie rekordów.

Kiedy używamy ADO, to mamy możliwość prawie bezpośredniej manipulacji danymi właśnie za pośrednictwem rekordsetu. Każdy rekordset składa się z rekordów (wierszy) i pól (kolumn).

Z obiektem recordset ściśle związane jest pojęcie **kursora**. Kursor to element bazy danych pozwalający na kontrolowanie nawigacji po zestawie rekordów, możliwość aktualizacji danych oraz pokazywanie lub nie zmian dokonanych w danych przez innych użytkowników.

Metoda ADO definiuje cztery różne typy kursorów:

- **Dynamic cursor** – pokazuje zmiany w danych obejmujące dodanie nowych rekordów, aktualizacje czy usunięcia rekordów dokonane przez innych użytkowników bazy, zabezpiecza wszystkie typy nawigacji po rekordsecie z wyłączeniem opartych o zakładki, ale pozwalających na ich stosowanie w sytuacji, gdy dostawca ADO to umożliwia.
- **Keyset cursor** – zachowuje się podobnie jak kursor dynamiczny z wyłączeniem możliwości zobaczenia rekordów dodanych przez innych użytkowników oraz blokując dostęp do tych rekordów, które zostały przez innego użytkownika skasowane. Wszystkie zmiany wprowadzone w danych są widoczne, pozwala na stosowanie zakładek, tym samym wszystkie typy nawigacji są dostępne.
- **Static cursor** – dostarcza statyczną kopię zestawu rekordów, którą można wykorzystać do wyszukania danych czy jako źródło rekordów dla np. raportu, zezwala na stosowanie zakładek, tym samych wszystkie techniki nawigacji po rekordsecie są dostępne. Zmiany wprowadzone przez innych użytkowników nie są widoczne. Tylko kursor tego typu może być zbudowany po stronie klienta (*client side*) w aplikacjach typu klient/serwer.
- **Forward-only cursor** – ogranicza nawigację zestawu rekordów tylko do przewijania do przodu, a zmiany wprowadzone przez innych użytkowników nie są widoczne.

Typ kursora może być zdefiniowany przed otwarciem rekordsetu, można go także zdefiniować bezpośrednio w metodzie Open otwierającej rekordset. W aplikacji można otworzyć tyle rekordsetów, ile jest nam potrzebne.

W momencie otwarcia rekordsetu bieżący rekord (`current record`) wskazuje na pierwszy rekord, a wskaźniki pierwszego (`BOF`) i ostatniego (`EOF`) rekordu są ustawiane na wartość `False`. W przypadku, gdy rekordset nie zwraca żadnego rekordu, to oba wskaźniki otrzymują wartość `True`.

Do poruszania się (nawigacji) po zestawie rekordów tworzących rekordset wykorzystujemy metody:

`MoveFirst` – wskaźnik bieżącego rekordu jest ustawiony na pierwszym rekordzie,
`MoveNext` – wskaźnik bieżącego rekordu jest przesuwany na kolejny rekord
`MovePrevious` – wskaźnik bieżącego rekordu jest cofany o jeden rekord,
`MoveLast` – wskaźnik bieżącego rekordu jest ustawiany na ostatnim rekordzie,
`Move` – pozwala na przejście do wskazanego rekordu, relatywnie względem bieżącego rekordu lub zakładki.

Do poruszania się po zestawie rekordów można wykorzystać takie właściwości jak:

`AbsolutePosition` – pozwala na przejście do rekordu o podanym numerze w ich rzeczywistej (porządkowej) numeracji,
`AbsolutePage` – pozwala na przejście do określonej strony w zestawie rekordów.
Przy dużej liczbie rekordów w rekordsecie możliwe jest ich podzielenie na strony (`Page`), których rozmiar jest definiowany właściwością `PageSize`. Wewnątrz strony rekordy są numerowane porządkowo, liczba stron jest zwracana przez właściwość `PageCount`,
`Filter` – pozwala na ograniczenie liczby rekordów do tych, które spełniają zdefiniowane kryteria.

Kursor typu `Forward-only` pozwala na stosowanie jedynie metody `MoveNext`. Przy poruszaniu się po zestawie rekordów za pomocą metody `Move` konieczne jest wykorzystywanie właściwości `BOF` i `EOF` do monitorowania, czy nie lokujemy wskaźnika bieżącego rekordu **przed** pierwszym rekordem lub **po** ostatnim rekordzie.

Obiekt `recordset` może zabezpieczać dwie metody aktualizacji danych w bazie: natychmiastową (`immediate`) i wsadową (`batched`). Przy aktualizacji natychmiastowej wszystkie zmiany w danych zostają przekazane do bazy danych natychmiast po wywołaniu metody `Update`. W ten sam sposób można przekazać do bazy większą porcję danych korzystając z metod `AddNew` i `Update`.

Przy aktualizacji wsadowej wszelkie zmiany czy dodania nowych rekordów są przechowywane po stronie klienta, a przekazywane do bazy w jednym wywołaniu metody `UpdateBatch`.

4.2.3 Praca z rekordsetem

Po zdefiniowaniu połączenia można otworzyć (zbudować) rekordset wywołując metodę Open obiektu Recordset. Formalna składnia tej metody pokazana jest poniżej.

```
recordset.Open source, active_connection, cursor_type, _  
lock_type, options
```

Wszystkie argumenty tej metody są opcjonalne w tym sensie, że mogą być przekazane także w inny sposób. Znaczenie tych argumentów jest następujące:

source – opisuje źródło danych, może być nazwą tabeli, poleceniem SQL, nazwą procedury przechowywanej (storage procedure) czy nazwą widoku (view).

active_connection – specyfikuje połączenie, które będzie wykorzystane do otwarcia rekordsetu.

cursor_type – specyfikuje rodzaj kursora, tym samym określając właściwości rekordsetu.

lock_type – determinuje rodzaj blokady zastosowanej do otwieranego rekordsetu, argument bardzo ważny w przypadku budowania aplikacji w technologii klient-serwer. Chodzi o mechanizm blokowania rekordu, pola czy całego zestawu rekordów w momencie ich edycji przy pracy w środowisku wielu użytkowników tej samej bazy. Dostępne są następujące typy blokowania:

adLockReadOnly – specyfikuje rekordset jedynie do odczytywania, nie jest możliwe dokonywanie żadnych zmian, jest to jednocześnie domyślny typ blokady.

adLockPessimistic – wskazuje na blokowanie pesymistyczne polegające na tym, że blokowany jest rekord w źródle danych (dla innych użytkowników) w momencie rozpoczęcia edycji (a czas jej trwania może być dowolnie długi).

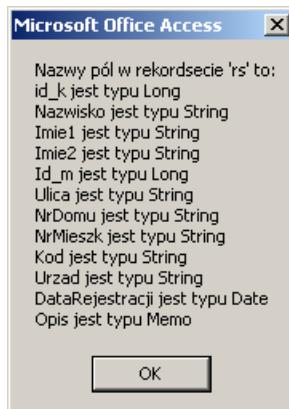
adLockOptimistic – wskazuje na zastosowanie blokowania optymistycznego polegającego na tym, że rekord (w źródle danych) jest blokowany bezpośrednio przed wykonaniem metody Update, a nie w momencie rozpoczęcia edycji. W praktyce oznacza to bardzo znaczne skrócenie czasu blokowania rekordów.

adLockBatchOptimistic – specyfikuje optymistyczną, wsadową aktualizację, obowiązkowy w przypadku stosowania metody BatchUpdate.

options – ten argument metody Open obiektu Recordset jest ścisłe powiązany z argumentem Source i określa sposób przetworzenia tego argumentu w tych przypadkach, gdy nie jest to obiekt Command.

W pokazanej niżej procedurze otwierany jest rekordset reprezentujący tabelę Klient bieżącej bazy danych. W pętli typu For Each przeglądana jest kolekcja pól tego rekordsetu, pobierane są nazwy tych pól i typ danych danego pola, który za pomocą funkcji (użytkownika) jest „tłumaczony” z kodu na opis słowny. Nazwa pola i opis jego typu są dodawane do zmiennej tekstowej t, która po przejrzeniu całej kolekcji pól zostanie wyświetlona za pomocą metody MsgBox.

```
Public Sub ADO_lokalnie()
    Dim con As New ADODB.Connection
    Dim rs As New ADODB.Recordset
    Set con = CurrentProject.Connection
    rs.Open "Klient", con, adOpenKeyset, adLockReadOnly
    Dim f As ADODB.Field, t As String
    t = "Nazwy pól w rekordsecie 'rs' to: " & vbCrLf
    For Each f In rs.Fields
        t = t & f.Name & " jest typu " & _
            TypPola(f.Type) & vbCrLf
    Next
    MsgBox t
End Sub
```



Funkcja TypPola wykorzystuje strukturę warunkową Select Case.

```
Public Function TypPola(id As Integer) As String
    Select Case id
        Case 3
            TypPola = "Long"
        Case 7
            TypPola = "Date"
```

```

Case 202
    TypPola = "String"
Case 203
    TypPola = "Memo"
Case Else
    TypPola = "brak opisu jeszcze"
End Select
End Function

```

W pokazanej niżej procedurze otwierany jest rekordset zwracający dane z zewnętrznej bazy danych z tabeli Wojewodztwa. Jako źródło metody Open (rekordsetu) podano zapytanie SQL w tym celu, aby rekordset zawierał posortowane rekordy wg nazwy województwa. Klauzula order by zawiera numer pola, wg którego mają być sortowane rekordy. Po otwarciu rekordsetu jest on przeglądany w pętli Do...Loop i do okna Immediate zwracane jest pierwsze i drugie pole rekordsetu (ale indeksy pól biegą od zera, stąd zwracane jest pole o indeksie 0 i pole o indeksie 1). Pętla będzie wykonywana tak długo, aż nie napotkamy znacznika końca rekordów EOF. Proszę zauważyc, że w pętli, po wyprowadzeniu zawartości pól bieżącego rekordu wywoływana jest metoda MoveNext w celu ustawienia wskaźnika bieżącego rekordu na kolejnym rekordzie. W pewnym momencie zostanie ustawiony na EOF, a wtedy pętla zakończy swoją działalność.

```

Public Sub Wojewodztwa()
    Dim cn As New ADODB.Connection
    Dim rs As New ADODB.Recordset
    cn.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
        "Data Source= c:\januszg\access8Listopad\Test.mdb"
    rs.Open "select * from Wojewodztwa order by 2", _
        cn, adOpenKeyset, adLockReadOnly
    Do While Not rs.EOF
        Debug.Print rs(0); rs(1)
        rs.MoveNext
    Loop
End Sub

```

Jeżeli znamy nazwy pól w rekordsecie, to można się do nich odwołać poprzez te nazwy, przy czym można to zrobić tak (w naszym przykładzie)

```
Debug.Print rs("id"); rs("nazwa")
```

lub tak

```
Debug.Print rs!id; rs!nazwa
```

Z tych trzech pokazanych metod odwołania się do pól rekordsetu zdecydowanie najlepsze są dwie ostatnie metody, w których wykorzystujemy nazwy pól. Po prostu taka instrukcja jest znakomicie bardziej czytelna niż odwołanie się poprzez indeks.

Kolejna procedura pokazuje otwarcie rekordsetu w taki sposób, aby była możliwość jego edycji. Zadaniem tej procedury jest przejrzenie tabeli Wojewodztwa w zewnętrznej bazie danych o nazwie Test.mdb w celu sprawdzenia, czy nazwy województw (pole Nazwa tej tabeli), są zapisane z dużej litery. Jeżeli nie, to konieczna jest aktualizacja takiej nazwy.

Rekordset zwraca w tym przypadku tylko nazwy województw, nie jest także istotne sortowanie tych danych, ale argument lock metody Open rekordsetu został ustawiony na adLockOptimistic w celu umożliwienia edycji. Badanie, czy nazwa województwa zaczyna się małą literą alfabetu wykonywane jest przez funkcję CzyLCase, której kod jest też pokazany.

```
Public Sub Wojewodztwa_DuzaLitera()
    Dim cn As New ADODB.Connection
    Dim rs As New ADODB.Recordset
    cn.Open "Provider=Microsoft.Jet.OLEDB.4.0; " & _
        "Data Source= c:\januszg\access8Listopad\Test.mdb"
    rs.Open "select nazwa from Wojewodztwa", cn, _
        adOpenKeyset, adLockOptimistic
    Do While Not rs.EOF
        If CzyLCase(Left(rs!nazwa, 1)) Then
            ' trzeba zmienić na dużą literę, edytujemy rs
            rs!nazwa = UCASE(Left(rs!nazwa, 1)) & _
                Right(rs!nazwa, Len(rs!nazwa) - 1)
            ' pokaż status trybu edycji
            Debug.Print rsEditMode
            rs.Update
        End If
        rs.MoveNext
    Loop
End Sub
```

Funkcja CzyLCase bada kod ASCII znaku przekazanego jako jej argument. Wykorzystana jest struktura warunkowa Select Case.

```
Public Function CzyLCase(t As String) As Boolean
    ' funkcja zwraca True, jeżeli t jest mała litera
    Select Case Asc(t)
        Case 97 To 122 'kody małych liter angielskich
            CzyLCase = True
        Case 185, 230, 234 ' kody a, á, é
            CzyLCase = True
        Case 241, 179, 156 'kody á, í, ó
            CzyLCase = True
        Case 243, 191, 159 ' kody ó, ž, ſ
            CzyLCase = True
    End Select
End Function
```

```

    CzyLCASE = True
    Case Else
        CzyLCASE = False ' w innym przypadku
    End Select
End Function

```

W pokazanej wyżej procedurze Wojewodztwa_DuzaLitera do okna *Immediate* wprowadzono także właściwość *EditMode*, która może być wykorzystana do monitorowania stanu edycji rekordu. W trakcie trwania edycji rekordu właściwość ta ustawiana jest na 1, a po pomyślnym wykonaniu metody *Update* jest zerowana. Takie badanie powinno być robione w aplikacjach typu klient-serwer, ponieważ próba przejścia do kolejnego rekordu w sytuacji, gdy metoda *Update* zawiodła zawsze zakończy się wygenerowanie błędu wykonania (*Run-time terror*) i zakończeniem pracy aplikacji. W przypadku, gdy wywołanie metody *Update* nie zostało zakończone sukcesem (EditMode zwraca 1), aktualizacja powinna być anulowana wywołaniem metody *CancelUpdate*.

Metoda *AddNew* obiektu *recordset* pozwala na dodanie do źródła danych nowego wiersza (nowego rekordu). Przykład takiej operacji ilustruje pokazana niżej procedura, w której do tabeli *Wojewodztwa* w bazie *Test.mdb* dodawana jest nazwa „Łódzkie”, ale dopiero po zbadaniu, że takiej nazwy już nie ma w tej tabeli. Przed dodaniem nowego województwa ustalana jest jeszcze maksymalna wartość pola *id* tej tabeli, pole to jest kluczem tej tabeli i nie zostało zdefiniowane jako *autonumer*, stąd konieczność ustalenia wartości tego pola przez nas i przekazania jej do źródła danych w metodzie *AddNew*.

```

Public Sub Wojewodztwa_DodanieNowego()
    Dim cn As New ADODB.Connection
    Dim rs As New ADODB.Recordset, id_nowy As Integer
    cn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source= c:\januszg\access8Listopad\Test.mdb"
    rs.Open "select nazwa from Wojewodztwa" & _
        "where nazwa = 'Łódzkie'", _
        cn, adOpenKeyset, adLockReadOnly
    If rs.RecordCount = 0 Then
        ' zamknięcie rekordsetu przed ponownym otwarciem
        rs.Close
        ' ustalenie, jakie jest maksymalne id (klucz)
        rs.Open "select max(id) as " & _
            "MaxID from Wojewodztwa", _
            cn, adOpenKeyset, adLockReadOnly
        ' zapamietanie maksymalnego klucza
        id_nowy = rs!MaxID
        ' zamykamy rekordset przed ponownym otwarciem
        rs.Close
        'otwarcie rekordsetu z możliwością edycji
    End If
End Sub

```

```

rs.Open "select * from wojewodztwa", cn, _
    adOpenKeyset, adLockOptimistic
' wywołanie metody AddNew
rs.AddNew
' przypisanie nowego, unikalnego identyfikatora
rs!id = id_nowy + 1
' przypisanie do pola nazwa
    nowej nazwy województwa
rs!nazwa = "Łódzkie"
' wywołanie metody Update
rs.Update
If rsEditMode = adEditModeNone Then
    MsgBox "Pomyślnie dodano nowy rekord", _
        vbInformation, "Praca z ADO"
Else
    MsgBox "Rekord nie został dodany, " & _
        "operacja zostanie anulowana", _
        vbCritical, "Praca z ADO"
    rs.CancelUpdate
End If
Else
    MsgBox "Nazwa 'Łódzkie' jest już w bazie", _
        vbCritical, "Praca z ADO"
End If
End Sub

```

Po uruchomieniu pokazanej wyżej procedury zobaczymy komunikat informujący o pomyślnym wykonaniu metody AddNew.



Ponowne wywołanie tej procedury spowoduje wyświetlenie komunikatu o tym, że nazwa, którą chcemy dodać, już jest w bazie.



Gdyby pole `id` tabeli `Wojewodztwa` było zdefiniowane jako `autonumer`, to w powyższej procedurze można byłoby pominąć instrukcję ustalającą maksymalną wartość tego pola jak i instrukcję, która przypisze wartość tego pola w dodawanym rekordzie.

Oczywiście dodanie nowego rekordu może być także wykonane z pominięciem metody `AddNew`, można po prostu wykonać polecenie na otwartym połączeniu. W naszym przykładzie odpowiednia instrukcja miałyby postać:

```
cn.Execute "insert into Wojewodztwa (id, nazwa) " & _
    values (" & id_nowy + 1 & ", 'Łódzkie')"
```

W kolejnym przykładzie pokażemy zastosowanie metody `Delete` obiektu `Recordset` do usunięcia ze źródła danych wskazanego rekordu lub grupy rekordów. Wykorzystana zostanie także metoda `Filter` do ograniczenia liczby rekordów wg zdefiniowanego kryterium oraz metoda `Requery` do odświeżenia rekordsetu wg wcześniej zdefiniowanych argumentów. Badana będzie kolekcja `Err` błędów dla ustalenia, czy metoda `Delete` została wykonana. Badanie takie jest potrzebne z dwóch zasadniczych powodów: nie można usunąć rekordu w sytuacji, gdyby zostały naruszone więzy integralności oraz rekordu, który został usunięty przez innego użytkownika.

Metoda `Delete` obiektu `Recordset` nie usuwa rekordu czy grupy rekordów, a jedynie markuje je (oznacza), jako rekordy do usunięcia. Dopiero wywołanie metody `Update` lub `UpdateBatch` usuwa zaznaczone rekordy.

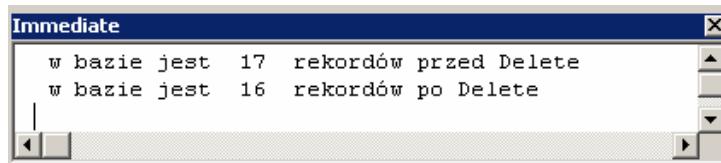
```
Public Sub Wojewodztwa_Delete()
    Dim cn As New ADODB.Connection
    Dim rs As New ADODB.Recordset, t As String
    cn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
        "Data Source= c:\januszg\access8Listopad\test.mdb"
    rs.Open "select nazwa from Wojewodztwa", _
        cn, adOpenKeyset, adLockOptimistic
    Debug.Print " w bazie jest "; rs.RecordCount; _
        " rekordów przed Delete"
    t = "Łódzkie"
    rs.Filter = "nazwa = '" & t & "'"
    ' badanie, czy jest taki rekord
    If rs.RecordCount < 1 Then
        MsgBox "Rekordu o nazwie '" & _
            t & "' nie ma w bazie!", vbInformation, "ADO"
        Exit Sub
    End If
    'wywołanie metody Delete, zerowanie kolekcji błędów
    Err.Clear
    rs.Delete
    rs.Update
```

```

' badanie, czy metoda Delete została wykonana
If Err > 0 Then
    MsgBox "W trakcie usuwania wystąpił błąd " & _
        Err.Number & ", usuwanie zatrzymane", _
        vbInformation, "ADO"
    rs.CancelUpdate
Else
    ' skasowanie filtru
    rs.Filter = adFilterNone
    ' ponowne wywołanie (odświeżenie) rekordsetu
    rs.Requery
    Debug.Print " w bazie jest "; rs.RecordCount; _
        " rekordów po Delete"
End If
' sprzątanie
rs.Close
cn.Close
Set rs = Nothing
Set cn = Nothing
End Sub

```

Poniżej okienko *Immediate* ze zwróconą informacją po wywołaniu wyżej pokazanej procedury.



W omówionych w dalszej części dwóch aplikacjach zainteresowany Czytelnik znajdzie wiele przykładów wykorzystania metody ADO zastosowanej w celu manipulowania danymi w bazie bieżącego projektu. Znajdzie tam także wiele przykładów kodu VBA zastosowanego w celu uzyskania założonej funkcjonalności obu projektów.

Oczywiście zamieszczone procedury nie wyczerpują wszystkich możliwości języka VBA, wydaje nam się jednak, że mogą być przydatne przy samodzielnym zgłębianiu tajemnic projektowania relacyjnych baz danych w takim systemie, jakim jest MS Access.

5. Przykłady aplikacji – obsługa biblioteki

W przykładzie tym przedstawimy kolejne kroki projektowania bazy danych przeznaczonej dla informatycznej obsługi średniej wielkości biblioteki (miejscowej, gminnej czy uczelnianej).

5.1. Funkcjonalność

Od projektowanej bazy będziemy oczekiwali następujących możliwości:

- Rejestracji użytkowników biblioteki wraz z ich danymi adresowymi i kontaktami (telefony, adresy mailowe),
- Prowadzenia rejestracji zakupionych pozycji bibliotecznych wraz z automatycznym nadawaniem sygnatur,
- Rejestracja wypożyczeń z aktualizacją bieżącego stanu pozycji bibliotecznych i indywidualnych kont klientów biblioteki,
- Rejestracja zwrotów wypożyczonych pozycji z aktualizacją bieżącego stanu zasobów bibliotecznych i indywidualnych kont klientów,
- Bieżące śledzenie tych pozycji, których termin zwrotu został przekroczony,
- Raportowanie o stanie zasobów i poziomie czytelnictwa w funkcji czasu.

Zadania powyższe będą mogły być zrealizowane wtedy, jeżeli poprawnie określmy liczbę i rodzaj gromadzonych informacji, a następnie ich rozdzielenie na poszczególne tabele.

5.2. Tabele i relacje

Pracę nad projektem zaczynamy od uruchomienia MS Access z pustą bazą danych. Bazę tę zapiszemy na lokalnym dysku w dedykowanym folderze pod nazwą Biblioteka. Po utworzeniu pliku bazy danych możemy przejść do etapu projektowania tabel i definiowania relacji między nimi.

5.2.1. Dane klientów

W aktywnym obiekcie *Tabele* możemy przystąpić do utworzenia pierwszych tabel przeznaczonych do gromadzenia informacji o użytkownikach naszej biblioteki. Wydaje się konieczne, aby o każdym z użytkowników mieć następujące informacje:

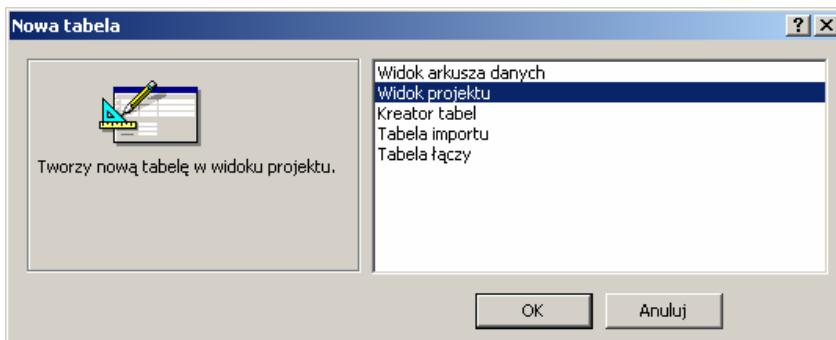
Nazwisko

Imię

Data urodzenia
 Miejscowość
 Ulica
 Numer domu
 Numer mieszkania
 Kod
 Urząd pocztowy
 Pesel
 Numery telefonów kontaktowych
 Adres (lub adresy) mailowe

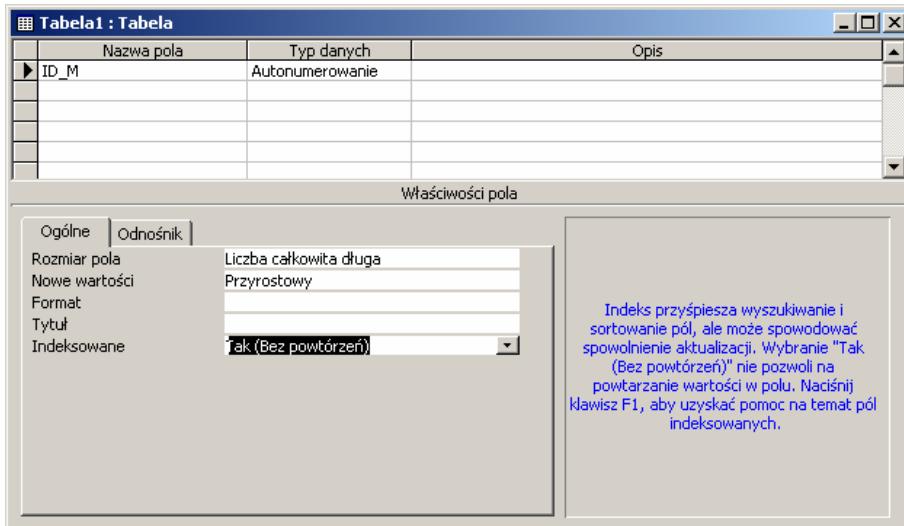
Natychmiast pojawi się problem w **ilu** tabelach i w **jakich** polach tę informację gromadzić. Wydaje się, że rozsądnym rozwiązaniem będzie przesunięcie informacji o nazwie miejscowości do oddzielnej tabeli, podobnie możemy utworzyć oddzielną tabelę dla takich informacji jak Kod czy Urząd pocztowy. Oczywiście w tabeli zasadniczej przechowującej informacje o użytkownikach muszą się pojawić pola wskazujące odpowiednie rekordy z tabel miejscowości czy kody.

Zaczniemy naszą pracę od utworzenia tabel przechowujących właśnie nazwy miejscowości oraz kody pocztowe i nazwy urzędów pocztowych. Z okna bazy danych, w aktywnym module *Tabele* wywołujemy polecenie *Nowy...*, co skutkuje wyświetleniem pokazanego niżej okna.

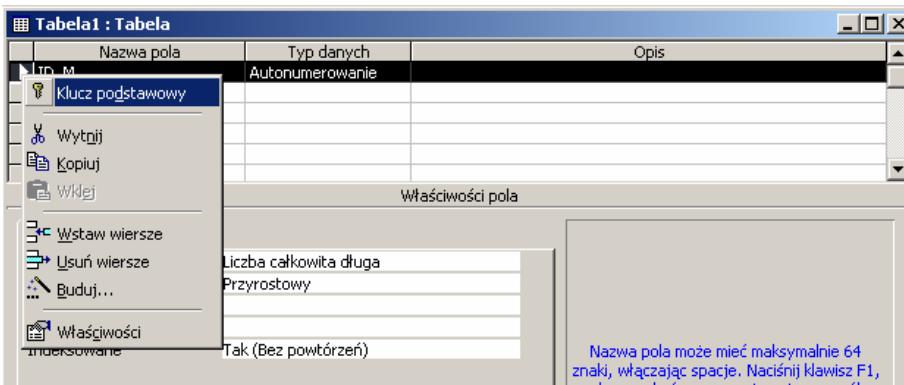


W oknie tym wybieramy opcję *Widok projektu* i po akceptacji przycisku OK możemy już przejść do projektowania pierwszej z tabel.

Pierwszym polem tabeli, której za chwilę nadamy nazwę *Miejscowosci* (bez polskich liter), będzie pole identyfikujące rekordy tej tabeli. Przyjmiemy jako zasadę, że pola tego typu będąemy zaczynać od prefiku *ID_* dopisując dalej jakiś identyfikator pola. Przykładowo nasze pole może otrzymać nazwę *ID_M* (od nazwy tabeli). Dla zapewnienia unikalności pole to zdefiniujemy jako *autonumer*. W pokazanej niżej sytuacji widzimy etap tworzenia tego pola, typ pola jest już wybrany, a w dolnej części okna projektowania tabel widzimy wyświetlone właściwości tego pola.

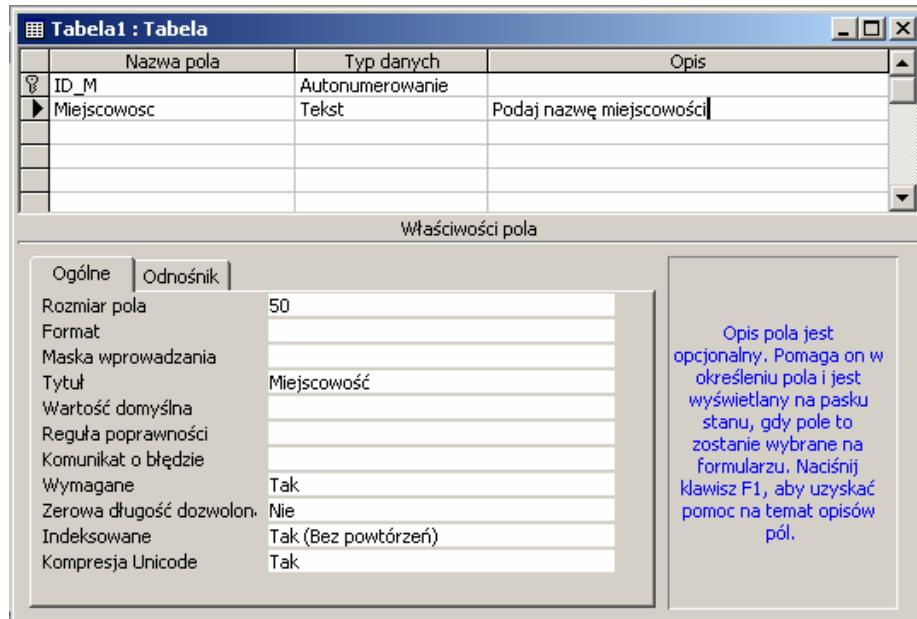


Pole `ID_M` ma pełnić w naszej tabeli rolę klucza głównego, możemy nadać mu ten status już teraz lub odłożyć definicję klucza na później (nie ma znaczenia kiedy to zrobimy). Ustanowienie danego pola kluczem głównym (klucz prosty) jest bardzo łatwe, wystarczy w pasku narzędziowym kliknąć przycisk oznaczony symbolem klucza. Inną metodą to skorzystanie z menu kontekstowego uruchamianego prawym przyciskiem myszy, gdzie znajdziemy odpowiednie polecenie. Sytuacja taka pokazana jest poniżej.



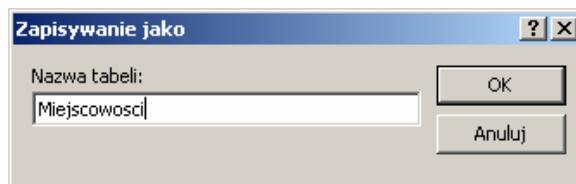
Kolejnym polem tej tabeli będzie pole o nazwie `Miejscowosc`, pole to będzie przechowywać wymagającą nazwę miejscowości o maksymalnej długości, powiedzmy 50 znaków. W momencie ustalenia typu przechowywanej informacji w oknie właściwości pola wyświetlanych jest szereg właściwości wybranego typu. W przypadku pola *tekstowego*

istotne właściwości to właśnie maksymalna długość pola, to warunek wymagalności oraz unikalności. Takie właściwości jak *Format* czy *Maska wprowadzania* w naszym konkretnym przypadku nie mają znaczenia (nie istnieje coś takiego jak format informacji nazwa miejscowości, a tym bardziej maska wprowadzania).

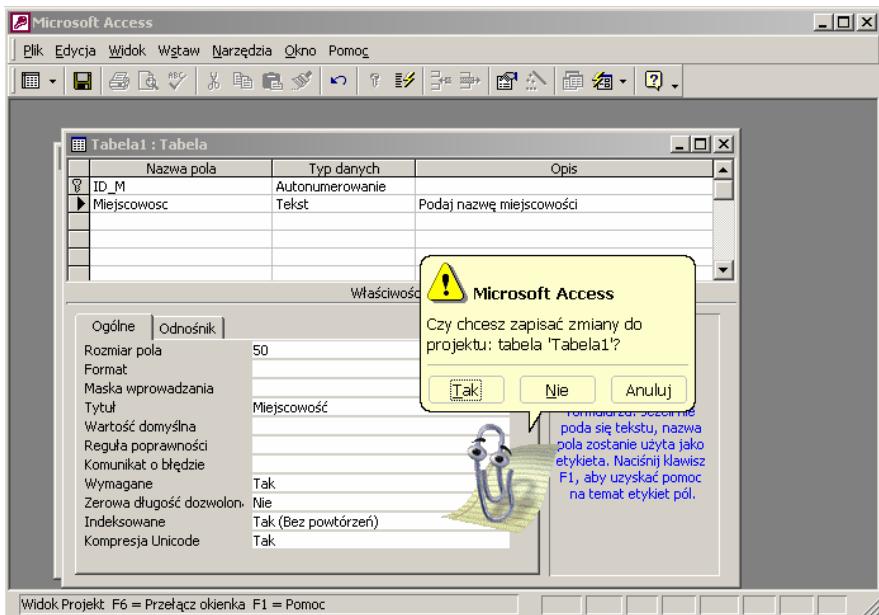


Właściwość *Tytuł* powinna być wypełniona, ponieważ może być wykorzystywana do opisywania pola w momencie tworzenia formularza. Przy definiowaniu tytułu możemy używać dowolnych znaków, w tym oczywiście i polskich.

Po zdefiniowaniu wszystkich pól projektowana tabela powinna być zapisana jako obiekt w bazie danych, przy czym można to zrobić poprzez wykorzystanie polecenia *Zapisz* w pasku *Projekt tabeli* lub poprzez próbę zamknięcia okna projektu tabeli.



W przypadku próby zamknięcia tabeli przed nadaniem jej nazwy MS Access wyświetli stosowny komunikat ostrzegawczy oczekując od nas odpowiedniej decyzji.

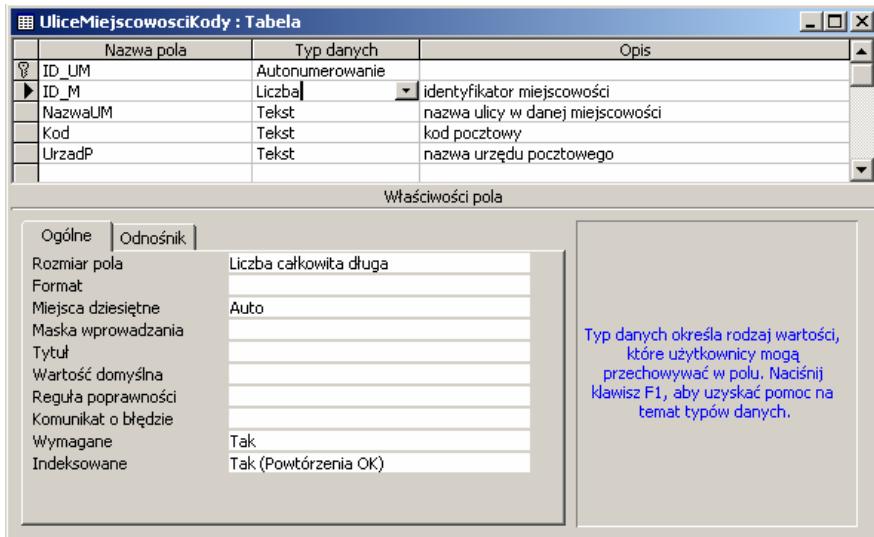


MS Access wyświetli pokazany wyżej komunikat z pytaniem, czy chcemy zapisać tabelę, a odpowiedź pozytywna spowoduje wyświetlenie pokazanego wcześniej okna z pytaniem o nazwę tabeli.

Oczywiście wprowadzona przez nas nazwa musi być unikalna w danej grupie (kolekcji) obiektów, inaczej MS Access nie zaakceptuje takiej nazwy.

Kolejny problem, który musimy rozwiązać, to zbudowanie tabeli do przechowywania kodów pocztowych i nazw urzędów pocztowych. W mniejszych miejscowościach nie ma z tym większego problemu, ponieważ z reguły cała miejscowość ma jeden kod i urząd pocztowy. Zupełnie inaczej jest w większych i dużych miejscowościach, w których istnieje wiele kodów i urzędów pocztowych. Jednym z możliwych rozwiązań będzie zbudowanie tabeli, która będzie kojarzyć miejscowości i ich ulice z odpowiadającymi im kodami pocztowymi i nazwami urzędów pocztowych.

Powiedzmy, że tabelę tę nazwiemy `UliceMiejscowosciKody`, a jej projekt może wyglądać tak, jak pokazany niżej.



Pierwszym polem w tej tabeli jest pole `ID UM` typu *autonumer*, pole to będzie pełniło rolę klucza tej tabeli. Kolejne pole to `ID M`, pole identyfikujące miejscowości. Jak pamiętamy, w tabeli `Miejscowosci` pole to jest typu *autonumer*, w aktualnie projektowanej tabeli pole to musi być typu *liczba całkowita dłuża*, z opcją *Wymagane* ustawioną na *Tak* oraz indeksowane, ale z powtórzeniami. Taki typ danych dla pola `ID M` jest niezbędny, jeżeli będziemy chcieli ustawić relacje między tabelami `Miejscowosci` a `UliceMiejscowosciKody`. Opcja *Wymagane* musi być ustalona na *Tak*, ponieważ nie chcemy dopuścić do takiej sytuacji, aby w tej tabeli pojawił się jakiś rekord, którego nie jesteśmy w stanie powiązać z żadną z miejscowości. Z kolei indeksowanie z powtórzeniami jest konieczne, jeżeli dopuszczać sytuację, że z jedną miejscowością może być związanych kilka kodów pocztowych czy ulic.

Kolejne trzy pola o nazwach odpowiednio `NazwaUM`, `Kod i UrzedP są typu tekstowego i mają przechowywać nazwę ulicy w danej miejscowości, kod pocztowy oraz nazwę urzędu pocztowego. Spośród tych trzech pól jedynie informacja wprowadzana do pola Kod ma ustalony format (postać), tym samym możemy zdefiniować wzorzec wprowadzania danych do tego pola.`

Pola `Kod` i `UrzedP` muszą mieć ustawioną właściwość *Wymagane* na *Tak*, chodzi oczywiście o to, aby w naszej tabeli nie pojawił się jakiś rekord z wadliwie podanym kodem czy nazwą urzędu pocztowego lub z brakującą informacją.

W przypadku pola `NazwaUM` właściwość *Wymagane* **musi** być ustaliona na *Nie*, co pozwoli nam na gromadzenie w tej tabeli kodu pocztowego i nazwy urzędu pocztowego dla malej miejscowości, która nie ma ulic (np. wioska).

Po zaprojektowaniu tabel `Miejscowosci` oraz `UliceMiejscowosciKody` możemy przejść do zaprojektowania właściwej tabeli `Klienci`. Projekt tabeli `Klienci` pokazany jest niżej.

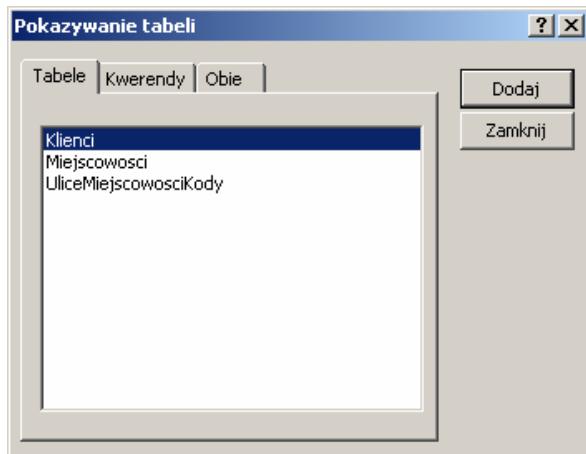
Pole `ID_UM` tej tabeli wykorzystamy do ustanowienia relacji typu *jeden-do-wielu* z tabelą `UliceMiejscowosciKody`, a poprzez nią także z tabelą `Miejscowosci`.

Pole `NrMieszk` **musi** mieć ustawioną właściwość *Wymagane* na *Nie*, co pozwoli nam na rejestrowanie w tej tabeli klientów, którzy mieszkają zarówno w mieszkaniach (w bloku) jak i domach. Dla wszystkich pozostałych pól właściwość *Wymagane* ustawiamy oczywiście na *Tak*.

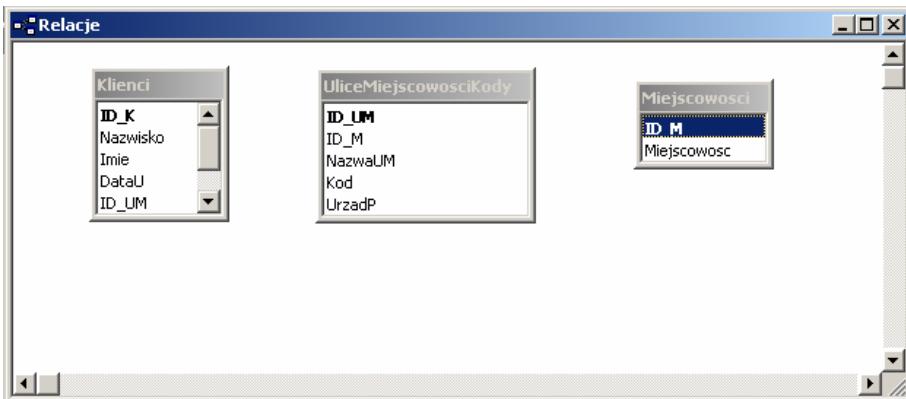
Pole `DataU` ma przechowywać datę urodzenia klienta w formacie „rrrr-mm-dd”, można wymusić taki format poprzez zdefiniowanie odpowiedniej maski wprowadzania.

Po utworzeniu pierwszych trzech tabel (`Klienci`, `Miejscowosci` oraz `UliceMiejscowosciKody`) możemy już zdefiniować relacje między nimi. Korzystając z polecenia *Relacje* w menu *Narzędzia*, lub z przycisku *Relacje* w pasku projektowania tabel albo z polecenia *Relacje* w menu kontekstowym projektu dowolnej tabeli otwieramy okno *Relacje*. W kolejnym kroku umieszczamy w nim te tabele, pomiędzy którymi chcemy ustanowić relacje. Wystarczy w tym celu wywołać menu kontekstowe w oknie relacji, a następnie wybrać polecenie *Pokaż tabele*. Dokładnie taki sam efekt osiągniemy wykorzystując polecenie *Widok* w pasku menu.

Okno dialogowe *Pokazywanie tabeli* pozwala na wybór obiektów z kolekcji tabel, kwerend lub obu tych kolekcji jednocześnie. W pokazanej sytuacji aktywna jest zakładka *Tabele*, wszystkie obiekty tej kolekcji są pokazane w oknie listy, a ich włączenie do okna relacji wymaga wykorzystania przycisku *Dodaj*. Dokładnie taki sam efekt osiągniemy poprzez podwójny klik wybranej tabeli czy kwerendy.



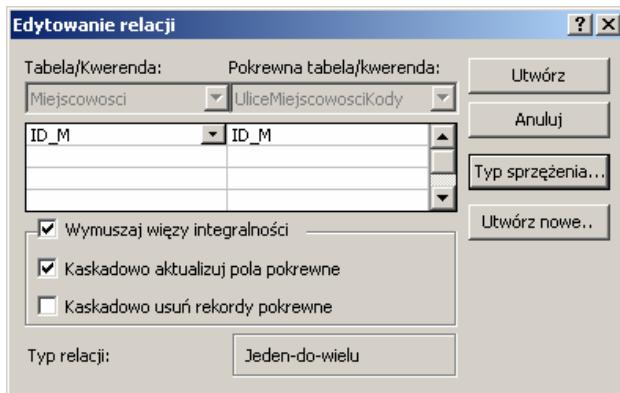
Po dodaniu wszystkich potrzebnych tabel lub kwerend okno *Pokazywanie tabeli* zamkamy i możemy przejść do tworzenia relacji. Niżej pokazane jest okno relacji z tymi tabelami, które dottyńczas utworzyliśmy.



Zaczniemy od utworzenia relacji typu *jeden-do-wielu* między tabelą *Miejscowosci* a tabelą *UliceMiejscowosciKody*. Do utworzenia tej relacji wykorzystamy pole *ID_M*, jak wiemy pole to jest takiego samego typu numerycznego (*liczba całkowita*

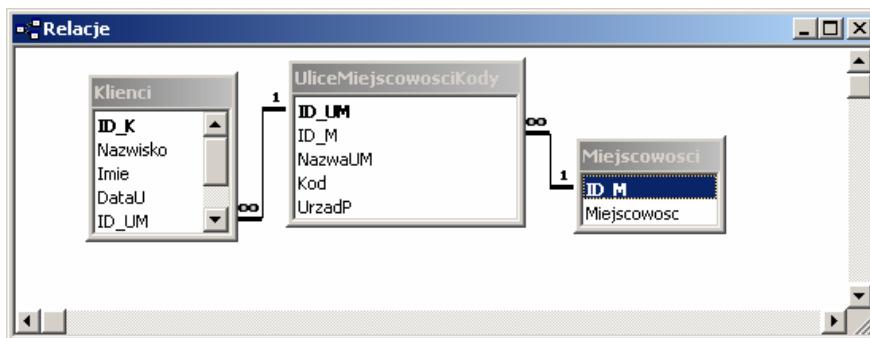
dluga), tym samym utworzenie relacji jest możliwe. Technicznie wykonamy to techniką „ciagnij-upuść” przeciągając pole `ID_M` z tabeli `Miejscowosci` (ze strony jeden w relacji) na pole `ID_M` w tabeli `UliceMiejscowosciKody` (po stronie wiele w relacji).

Po zwolnieniu myszy zobaczymy pokazane niżej okno dialogowe *Edytowanie relacji*, w oknie tym możemy ustawić wszystkie potrzebne właściwości tworzonej relacji.



W pokazanej sytuacji widzimy pola łączące w tabeli nadzędnej i podrzędnej oraz typ relacji *jeden-do-wielu*. Aktywne jest pole wyboru *Wymuszaj więzy integralności*, tym samym MS Access **nie pozwoli** na wprowadzenie do tabeli podrzędnej rekordu z taką wartością pola `ID_M`, która nie będzie istniała w tabeli nadzędnej.

Analogicznie tworzymy relację między tabelą `UliceMiejscowosciKody` a tabelą `Klienci`, tym razem do ustanowienia relacji wykorzystujemy pole `ID_UM`. Efekt końcowy pokazany jest niżej, widzimy dwie relacje typu *jeden-do-wielu*, strona „1” zapisana jest symbolem 1, a strona „wiele” symbolem nieskończoności.



Jednym z efektów utworzonych relacji jest możliwość hierarchicznego pokazywania zapisanych w tych tabelach informacji. Poniżej pokazany jest widok tabeli Miejscowosci, rekord Łowicz jest rozwinięty – widzimy odpowiadające mu rekordy z tabeli UliceMiejscowosciKody. Rekord zawierający nazwę ulicy Bratkowice jest rozwinięty, widzimy odpowiadające mu rekordy z tabeli Klienci (tu jeden rekord).

Miejscowosci : Tabela	
ID_M	Miejscowość
5	Błonie
7	Jackowice
1	Łowicz

ID_UM	NazwaUM	Kod pocztowy	UrzadP
1	Zduńska	23-345	Łowicz
2	Bratkowice	23-346	Łowicz

ID_K	Nazwisko	Imię	Data urodzeni	NrDomu	NrMieszk
2	Abacki	Adam	1978-02-23	1	12
*	(numerowanie)				

Musimy jeszcze rozwiązać problem przechowywania numerów telefonów kontaktowych oraz adresów poczty elektronicznej naszych klientów. Pomysł, aby dodać odpowiednie pola do tabeli Klienci nie jest najlepszy z wielu powodów. Po pierwsze, nie wiadomo ile numerów telefonicznych powinniśmy przewidzieć dla każdego z klientów, analogiczna uwaga dotyczy adresów mailowych. Po drugie, jeżeli dany klient nie ma telefonu lub adresu mailowego, to pola zostają puste.

Eleganckim rozwiązaniem będzie utworzenie dwóch nowych tabel: tabeli o nazwie Telefonie oraz tabeli o nazwie AdresyMailowe. W obu tabelach wprowadzimy pole ID_K, dzięki któremu ustanowimy relacje z tabelą Klienci.

Telefony : Tabela		
Nazwa pola	Typ danych	Opis
ID_T	Autonumerowanie	
ID_K	Liczba	
NumerT	Tekst	Podaj numer telefonu (dla stacjonarnych z kierunkiem)

Ogólne	Odnośnik
Rozmiar pola	15
Format	
Maska wprowadzania	
Tytuł	
Wartość domyślna	
Reguła poprawności	
Komunikat o błędzie	
Wymagane	Tak
Zerowa długość dozwolona	Nie
Indeksowane	Tak (Powtórzenia OK)
Kompresja Unicode	Tak

Użyj hadowy wyświetlanego dla pola. Wybierz format gotowy lub określ niestandardowy. Naciśnij klawisz F1, aby uzyskać pomoc na temat formatów.

Przed rozpoczęciem projektowania tabeli AdresyMailowe warto się zastanowić, czy nie powinniśmy rozdzielić nazwy konta pocztowego od nazwy domeny. Krok taki wynika z jednego z postulatów normalizacji, który zaleca unikania nadmiernego powtarzania informacji, a tak będzie, jeżeli w projektowanej tabeli będziemy gromadzić pełne adresy mailowe. Jeżeli zdecydujemy się na trzymanie adresów domenowych w oddzielnej tabeli, np. o nazwie Domeny, to projekty obu tabel mogą wyglądać jak niżej pokazane.

AdresyMailowe : Tabela		
Nazwa pola	Typ danych	Opis
ID_AM	Autonumerowanie	
ID_K	Liczba	Identyfikator Klienta
Przedrostek	Tekst	Podaj tę część adresu mailowego, która jest przed symbolem @
ID_D	Liczba	Identyfikator domeny

Właściwości pola

Ogólne	Odnośnik	
Rozmiar pola	25	
Format		
Maska wprowadzania		
Tytuł		
Wartość domyślna		
Reguła poprawności		
Komunikat o błędzie		
Wymagane	Tak	
Zerowa długość dozwolona	Nie	
Indeksowane	Nie	
Kompresja Unicode	Tak	

Etykieta pola na formularzu. Jeżeli nie poda się tekstu, nazwa pola zostanie użyta jako etykieta. Naciśnij klawisz F1, aby uzyskać pomoc na temat etykiet pól.

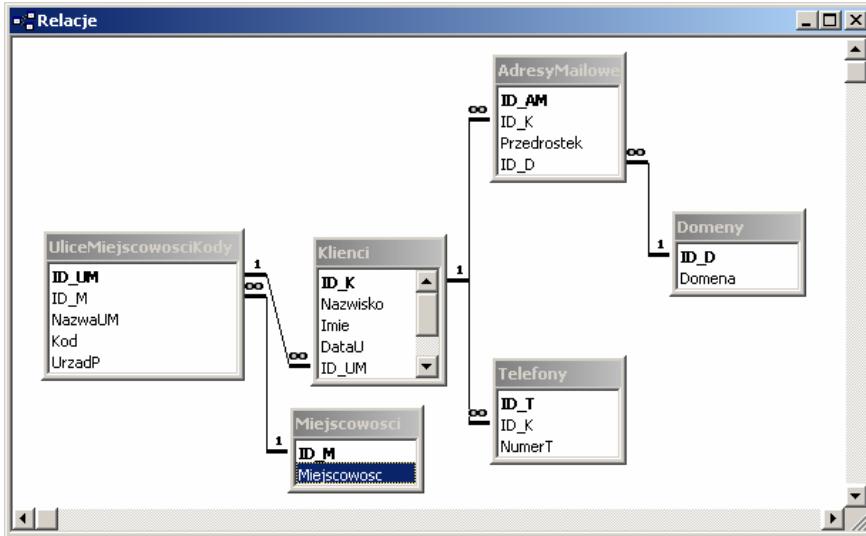
Domeny : Tabela		
Nazwa pola	Typ danych	Opis
ID_D	Autonumerowanie	
Domena	Tekst	Podaj adres domeny (część adresu mailowego za znakiem @)

Właściwości pola

Ogólne	Odnośnik	
Rozmiar pola	30	
Format		
Maska wprowadzania		
Tytuł	Domena	
Wartość domyślna		
Reguła poprawności		
Komunikat o błędzie		
Wymagane	Tak	
Zerowa długość dozwolona	Nie	
Indeksowane	Tak (Bez powtórzeń)	
Kompresja Unicode	Tak	

Etykieta pola na formularzu. Jeżeli nie poda się tekstu, nazwa pola zostanie użyta jako etykieta. Naciśnij klawisz F1, aby uzyskać pomoc na temat etykiet pól.

W zasadzie tabele przechowujące dane klientów mamy już gotowe, pozostało jeszcze dodanie do relacji ostatnio utworzonych tabel i ustanowienie relacji między wszystkimi sześcioma tabelami.



5.2.2. Dane zakupionych książek

W tej części projektu musimy podjąć decyzję o tym, jakie dane odnośnie zakupionych lub podarowanych pozycji bibliotecznych chcemy gromadzić oraz jak rozdzielić te informacje na poszczególne tabele.

Wydaje się, że o każdej zakupionej pozycji powinniśmy mieć następujące informacje:

- ID_P (identyfikator pozycji),
- Tytuł pozycji,
- Autor (lub autorzy),
- Wydawnictwo,
- ISBN,
- Liczba stron,
- Data zakupu,
- Cena,
- Ilość egzemplarzy,
- Kategoria,
- Sygnatura.

Analiza proponowanych informacji wskazuje na kilka problemów, które musimy rozwiązać. Po pierwsze takie informacje jak Wydawnictwo czy Kategoria powinniśmy umieścić w oddzielnich tabelach (w celu uniknięcia redundancji danych). Problem sygnatur powinien być rozwiązany podobnie do telefonów czy adresów mailowych, co wynika z faktu, że zakupiona pozycja może mieć wiele sygnatur – tak będzie, jeżeli zakupimy więcej niż jedną pozycję. Wystarczy więc, że utworzymy nową tabelę o nazwie np. *Sygnatury*, gdzie będziemy przechowywać same sygnatury oraz identyfikatory pozycji. Tabelę tę uzupełnimy o jeszcze jedno pole numeryczne, o nazwie np. *Status*, którego zadaniem będzie określanie statusu danej pozycji w sensie jej dostępności dla klienta biblioteki.

Pozostaje jeszcze problem autorów zakupionych książek, o tyle skomplikowany, że dana pozycja może mieć zmienną liczbę autorów, a jednocześnie dany autor może być autorem wielu pozycji. Jednym z możliwych rozwiązań będzie zbudowanie tabeli *Autorzy*, gdzie będziemy przechowywać podstawowe dane autorów. Kolejna tabela, powiedzmy o nazwie *PozycjeAutorzy* będzie kojarzyć identyfikatory pozycji z identyfikatorami autorów.

Ostatecznie zbudujemy następujące tabele (klucze główne pogrubione):

Zakupy

Pole	Opis	Typ pola	Wymagane	Indeksowane
ID_P	Identyfikator pozycji	autonumer	tak	tak
Tytul	Tytuł pozycji	tekstowy	tak	Nie
ISBN	Nr biblioteki narodowej	tekstowy	tak	Nie
Stron	Liczba stron	liczbowy	tak	Nie
DataZ	Data zakupu	Data/Czas	tak	Nie
Cena	Cena zakupu	walutowy	Tak	Nie
Ile	Ilość pozycji	liczbowy	Tak	Nie
ID_W	Identyfikator wydawnictwa	liczbowy	Tak	Nie
ID_K	Identyfikator kategorii	liczbowy	Tak	Nie

Autorzy

Pole	Opis	Typ pola	Wymagane	Indeksowane
ID_A	Identyfikator autora	autonumer	Tak	Tak
Nazwisko	Nazwisko autora	tekstowy	tak	Nie
Imie	Imię autora	Tekstowy	Tak	nie

PozycjeAutorzy

Pole	Opis	Typ pola	Wymagane	Indeksowane
ID_P	Identyfikator pozycji	liczbowy	tak	Nie
ID_A	Identyfikator autora	Liczbowy	Tak	nie

Wydawnictwa

Pole	Opis	Typ pola	Wymagane	Indeksowane
ID_W	Identyfikator wydawnictwa	Autonumer	Tak	Tak
NazwaW	Nazwa wydawnictwa	Tekstowy	Tak	Tak

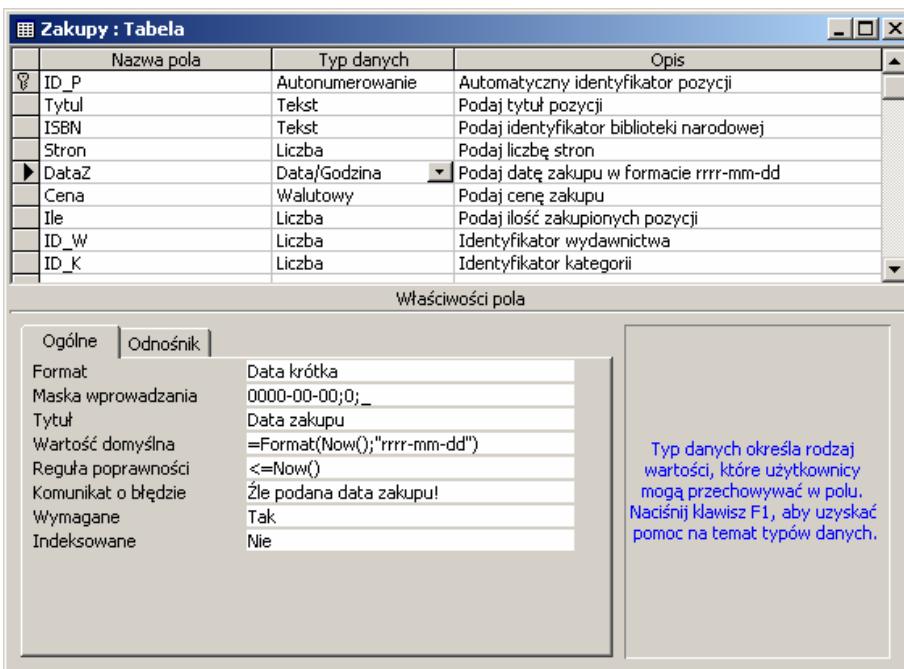
Kategorie

Pole	Opis	Typ pola	Wymagane	Indeksowane
ID_K	Identyfikator kategorii	Autonumer	Tak	Tak
NazwaK	Nazwa kategorii	Tekstowe	Tak	Tak

Sygnatury

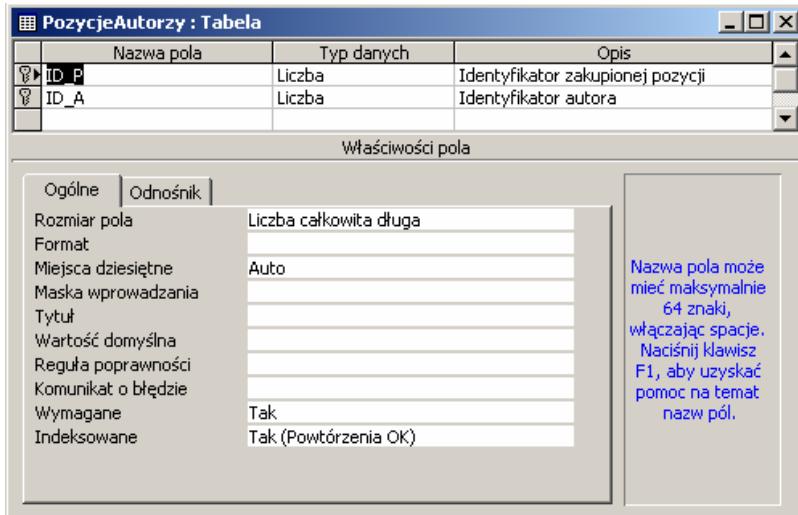
Pole	Opis	Typ pola	Wymagane	Indeksowane
ID_S	Identyfikator sygnatury	Autonumer	Tak	Tak
ID_P	Identyfikator pozycji	Liczbowe	Tak	Nie
Sygnatura	Sygnatura biblioteczna	Tekstowy	Tak	Tak
Status	Status pozycji	Liczbowy	Tak	Nie

Poniżej pokazany jest projekt tabeli *Zakupy* ze szczegółami definicji pola *DataZ*, warto zwrócić uwagę na takie właściwości tego pola jak *Format*, *Maska wprowadzania*, *Wartość domyślna*, *Reguła poprawności* czy *Komunikat o błędzie*.

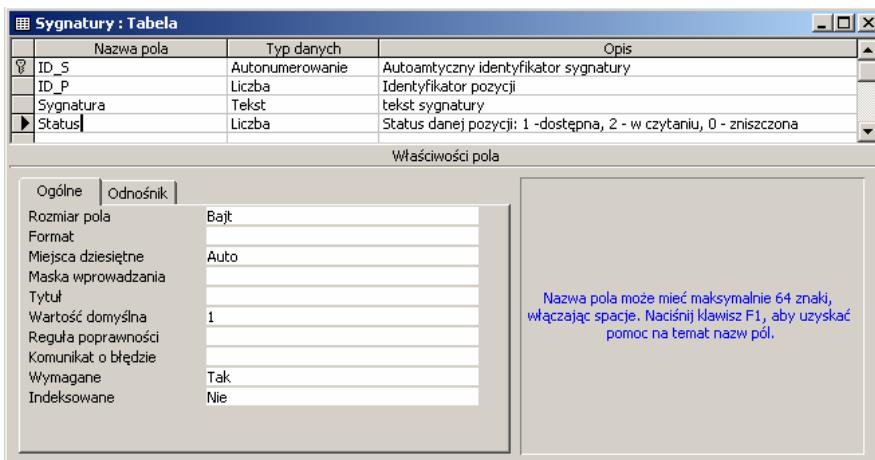


W polu definiowania właściwości *Wartość domyślna* została wpisana **formula** `=Format(Now(); "rrrr-mm-dd")`, jej zadaniem jest wstępne wpisanie bieżącej daty w ustalonym formacie. Z kolei w polu właściwości *Reguła poprawności* formula `<=Now()` będzie pilnować, aby data zakupu nie była wcześniejsza od bieżącej daty.

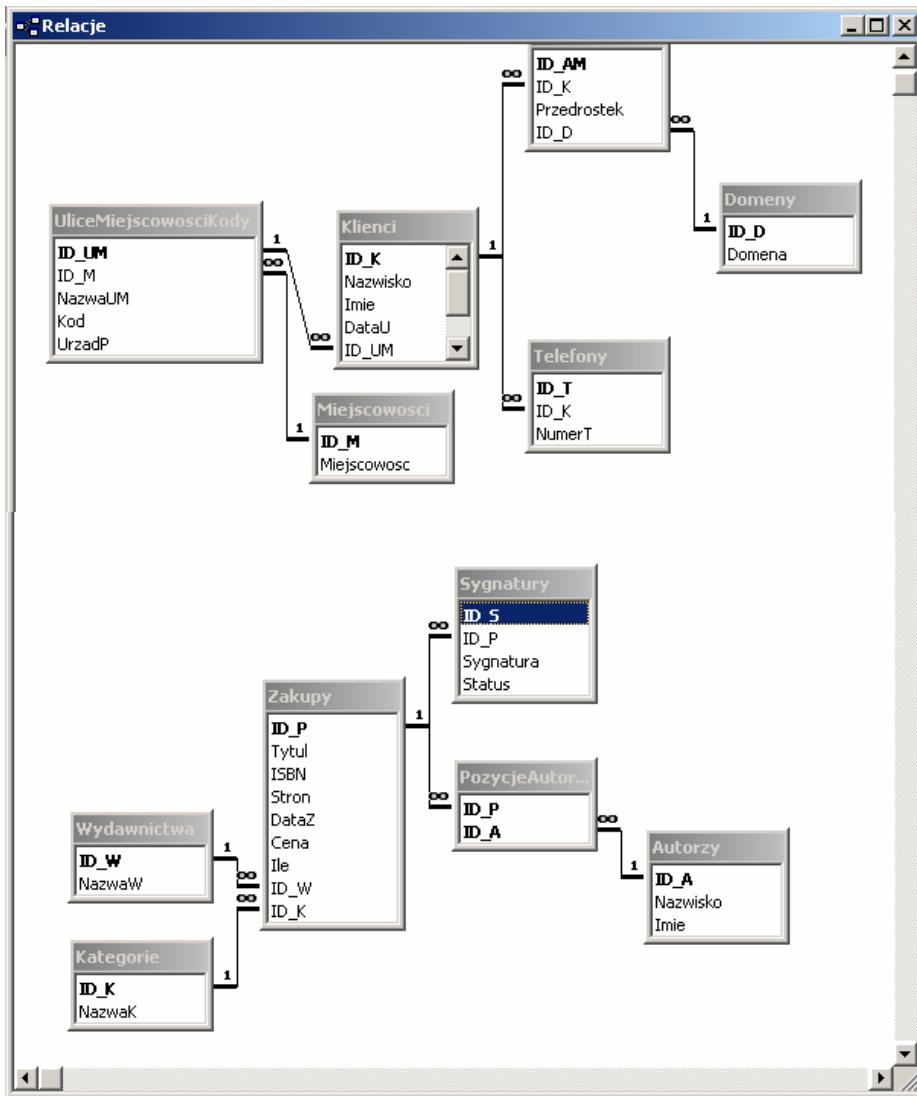
Dość oryginalnie wygląda projekt tabeli PozycjeAutorzy, gdzie znajdują się tylko dwa pola będące kluczami głównymi w swoich tabelach. Pola te tworzą klucz złożony tej tabeli.



Na zakończenie projektowania tabel w tej części naszego projektu pokażemy jeszcze projekt tabeli Sygnatury ze szczegółami pola Status.



Pozostaje dodanie utworzonych tabel do okna relacji oraz ich ustanowienie między poszczególnymi tabelami.



Z pokazanych wyżej dotychczas utworzonych relacji wynika, że mamy w tym momencie dwie grupy tabel absolutnie z sobą niepowiązanych. W górnej części mamy sześć tabel opisujących klienta naszej biblioteki, a w dolnej również sześć tabel opisujących pozycje biblioteczne. Dla powiązania obu grup tabel potrzebna jest jeszcze jedna tabela, będziemy w niej przechowywać wszelkie informacje o wypożyczonych pozycjach.

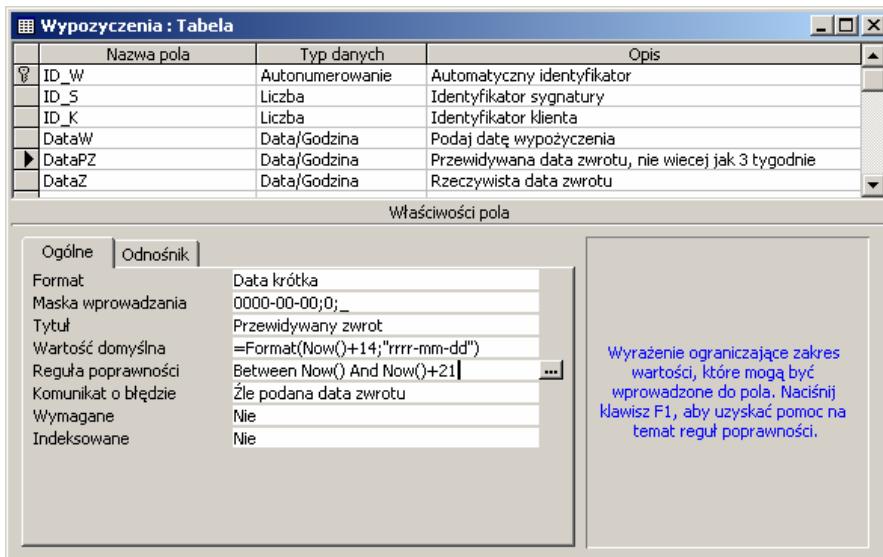
5.2.3. Rejestracja wypożyczeń

Utworzymy tabelę o nazwie `Wypozyzczenia` zawierającą następujące pola:

`Wypozyzczenia`

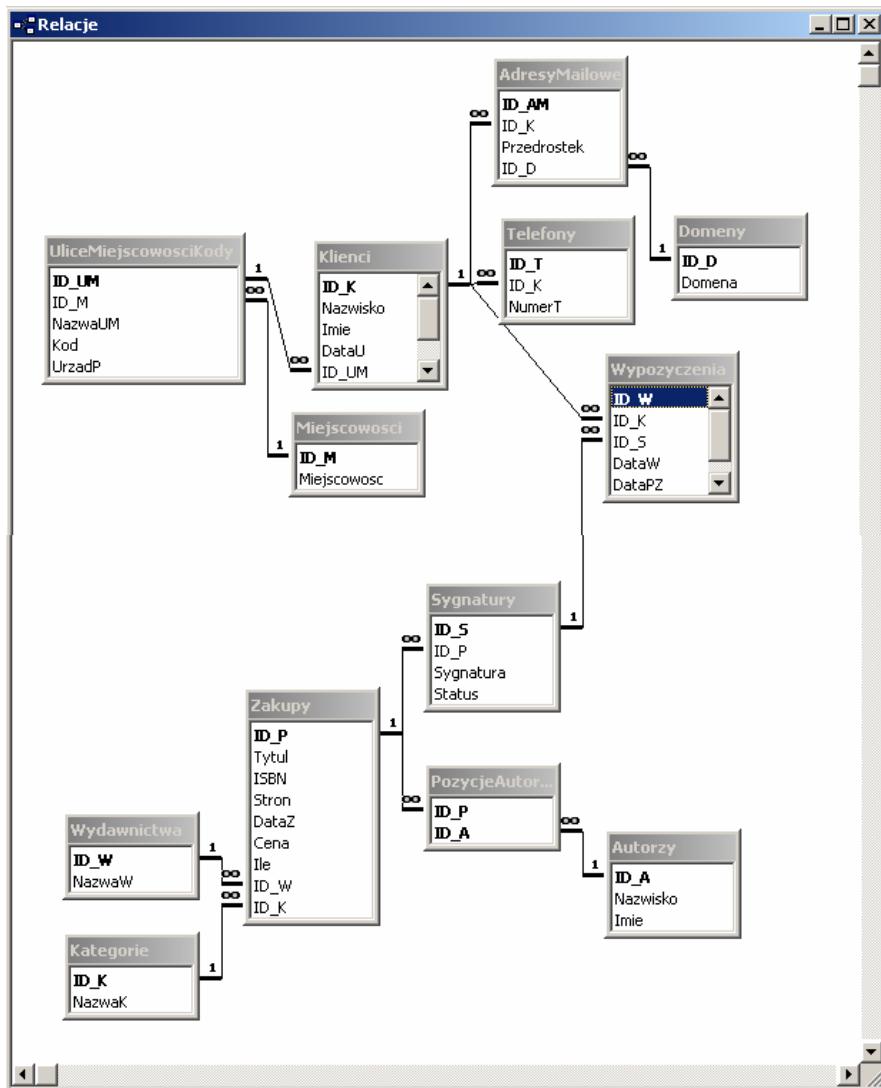
Pole	Opis	Typ pola	Wymagane	Indeksowane
<code>ID_W</code>	Identyfikator wypożyczenia	Autonumer	Tak	Tak
<code>ID_S</code>	Identyfikator sygnatury	Liczbowe	Tak	Nie
<code>ID_K</code>	Identyfikator klienta	Liczbowe	Tak	Nie
<code>DataW</code>	Data wypożyczenia	Data/Czas	Tak	Nie
<code>DataPZ</code>	Przewidywana data zwrotu	Data/Czas	Tak	Nie
<code>DataZ</code>	Rzeczywista data zwrotu	Data/Czas	Nie	Nie

Tabela ta, poprzez pola `ID_S` i `ID_K`, powiąże obie grupy tabel tworząc jedną całość. Pole `DataZ` jest przeznaczone na przechowywanie rzeczywistej daty zwrotu wypożyczonej pozycji, stąd jego właściwość **Wymagane** **musi** być ustawiona na **Nie**. Poniżej pokazane jest okno projektu tej tabeli ze szczegółami definicji pola `DataPZ`.



Warto zwrócić uwagę na zdefiniowanie właściwości tego pola, takie a nie inne definicje są wprowadzone po to, aby z jednej strony ułatwić wprowadzanie danych do tego pola, a z drugiej pilnować ich poprawności. Umownie przyjęto, że średni czas wypożyczenia danej pozycji to 2 tygodnie (14 dni), a maksymalny czas wypożyczenia nie może być większy od 3 tygodni (21 dni).

Poniżej pokazane jest okno relacji po wyłączeniu tabeli *Wypożyczenia*, jak łatwo zobaczyć, mamy kompletny i spójny zestaw tabel (dla lepszej przejrzystości połączeń zmieniona została kolejność pól *ID_K* i *ID_S* w tabeli *Wypożyczenia*).

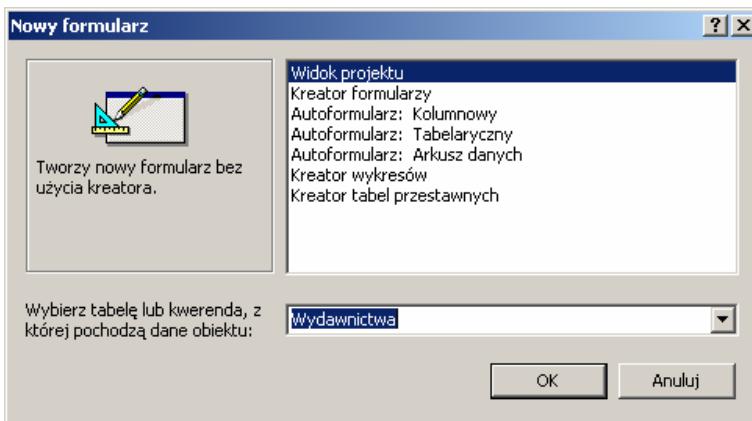


5.3. Formularze i kwerendy

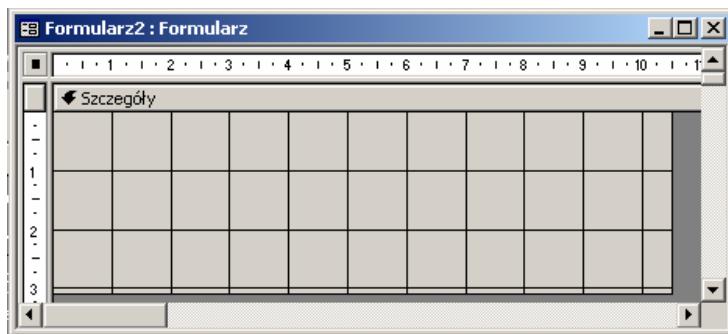
Po zaprojektowaniu tabel i utworzeniu relacji możemy przejść do kolejnego etapu budowy naszej aplikacji – do przygotowania formularzy ekranowych. Źródłem danych dla nich będą albo istniejące tabele, albo specjalnie zaprojektowane kwerendy.

5.3.1. Kilka prostych formularzy

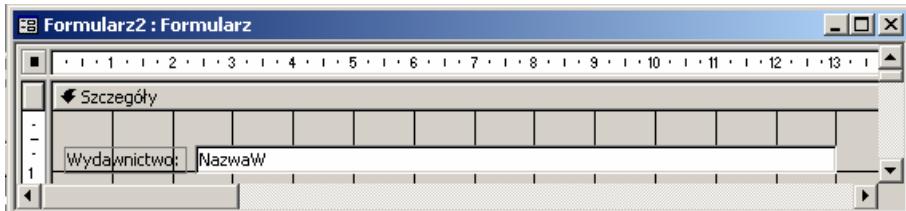
Zaczniemy od utworzenia kilku prostych formularzy przeznaczonych do wprowadzania, edycji i przeglądu takich informacji, jak nazwy wydawnictw czy kategorii. W pokazanej niżej sytuacji zaczynamy budowę formularza ekranowego opartego o tabelę Wydawnictwa. Etap wstępny tych prac pokazany jest poniżej; tworzymy w widoku projektu nowy formularz oparty o tabelę Wydawnictwa.



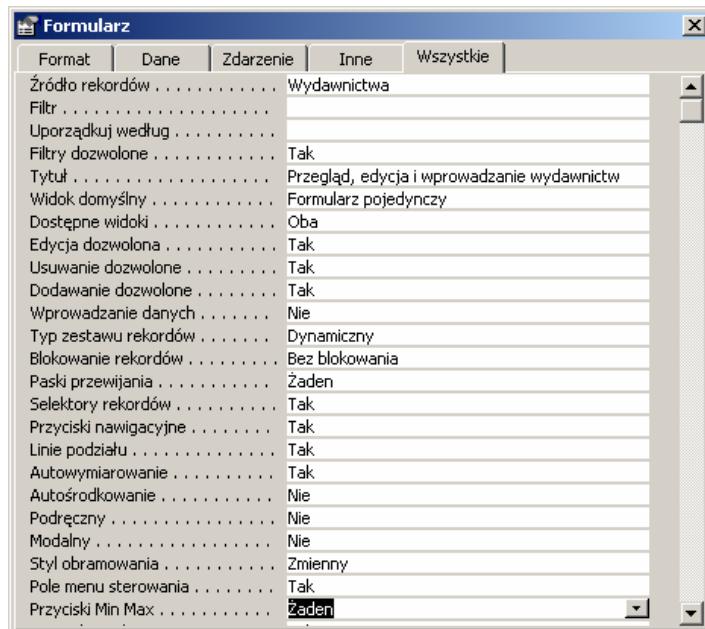
Po akceptacji przycisku OK mamy ekran projektowania formularzy w standardowej postaci, czyli bez nagłówków i stopek.



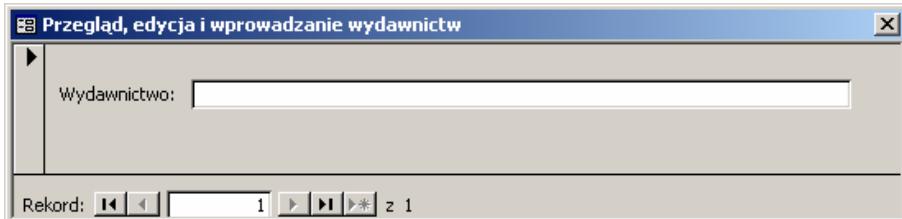
Korzystając z listy pól przeciągamy do sekcji *Szczegóły* pole *NazwaW*, w efekcie w formularzu umieszczone zostaną dwie kontrolki: etykieta opisujące pole (jeżeli jego właściwość *Tytuł* była zdefiniowana, jeżeli nie, to tytuł etykiety jest nazwą pola oraz pole tekstowe, dla którego źródłem danych jest pole *NazwaW*). Po osadzeniu obu kontrolek na powierzchni formularza dokonujemy kilku istotnych zmian: zmieniamy tytuł etykiety na jasno opisujący pole tekstowe, dopasowujemy rozmiary obu kontrolek do przewidywanych potrzeb i ustawiamy je we właściwym miejscu w formularzu. Przykładowy widok tego etapu prac nad formularzem pokazany jest niżej.



Kolejny krok to ustawienie istotnych właściwości formularza. Po uaktywnieniu przycisku formularza (lewy górny narożnik w pokazanej wyżej sytuacji z czarnym kwadratem) wywołujemy okno właściwości formularza.

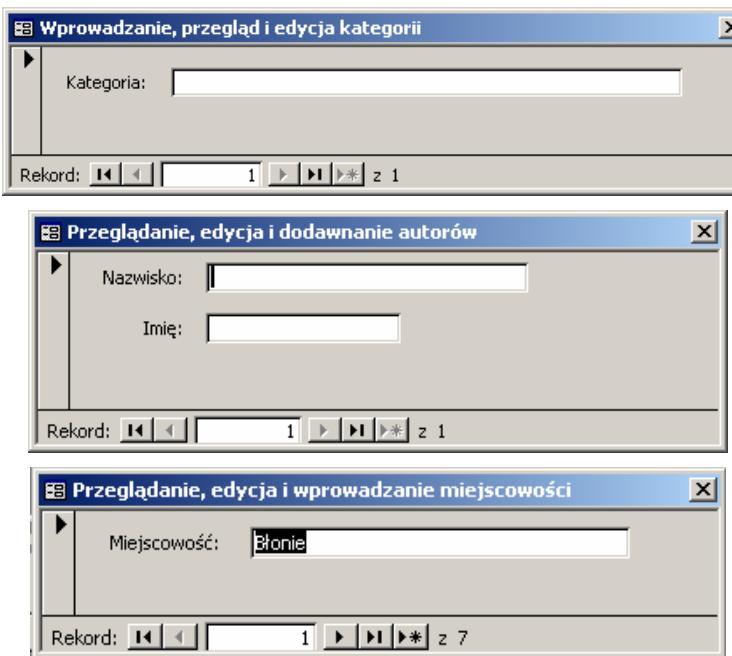


W oknie tym zmieniamy (definiujemy) takie właściwości jak: *Tytuł*, *Paski przewijania*, *Przyciski Min Max*, *Widok domyślny*. Efekt wprowadzonych zmian widoczny jest poniżej.



Pozostaje zapisanie formularza w kolekcji formularzy, zgodnie z pewną konwencją tworzenia nazw obiektów nadamy mu nazwę `frmWydawnictwa`, gdzie `frm` jest przedrostkiem identyfikującym obiekty typu formularzy.

W sposób analogiczny tworzymy formularze `frmKategorie`, `frmAutorzy`, `frmMiejscowosci` oraz `frmDomeny`.





W tej chwili pięć utworzonych formularzy (frmWydawnictwa, frmKategorie, frmAutorzy, frmMiejscowosci oraz frmDomeny) można wykorzystywać do przeglądania i edycji zawartości tabel będących ich źródłem danych. Dodatkowo, po przejściu do nowego rekordu można także wykorzystać te formularze do dodania nowych rekordów.

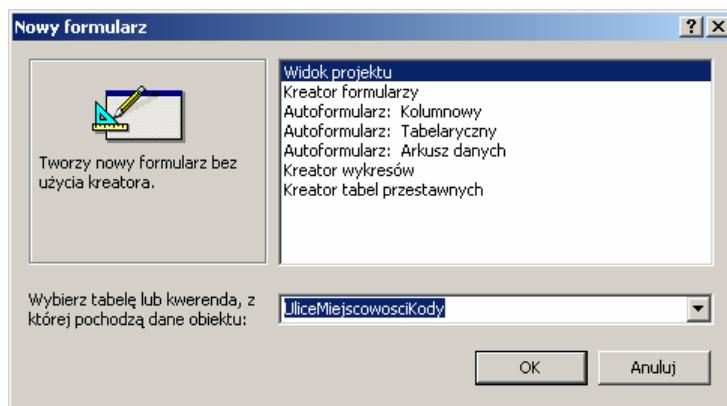
Z reguły „porządnie” zaprojektowana baza danych powinna oddzielić operację dodawania nowych rekordów od operacji przeglądania i edycji istniejących, tak też i my postąpimy w dalszej części tego skryptu, ale będzie to wymagać wykorzystania języka VBA.

5.3.2. Formularze z kontrolką typu ComboBox

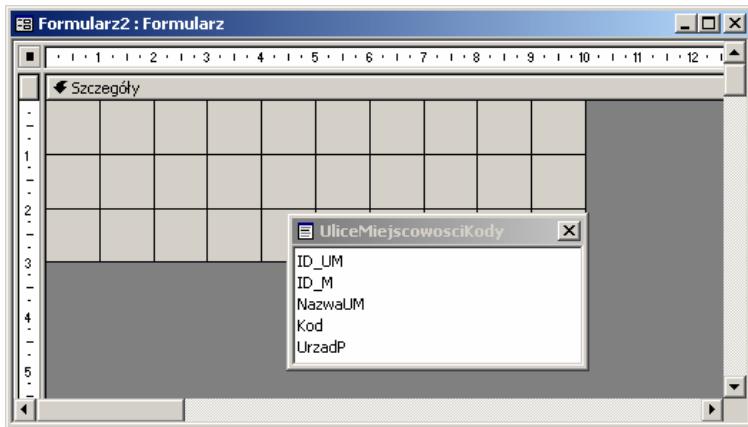
Przejdziemy teraz do zaprojektowania trochę bardziej złożonych formularzy, ich wspólną cechą będzie wykorzystywanie kontrolki typu *ComboBox* (rozwijana lista).

Jedną z tabel naszej bazy danych jest tabela *UliceMiejscowosciKody*, a jednym z pól tej tabeli jest pole *ID_M* będące kluczem głównym tabeli *Miejscowosci*. Formularz, który za chwilę utworzymy ma maksymalnie ułatwić wprowadzanie wartości tego właśnie pola.

Zaczynamy standardowo od otwarcia projektu formularza w oparciu o tabelę *UliceMiejscowosciKody*.



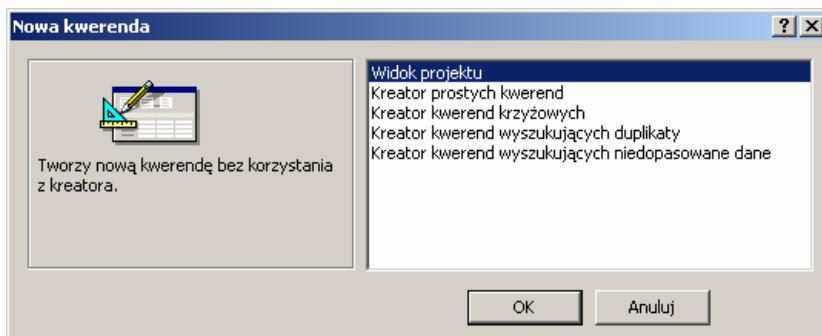
W okienku listy pól widzimy wszystkie pola występujące w źródle danych dla tego formularza, w tym pole ID_M i trzy pola tekstowe (NazwaUM, Kod, UrzadP).



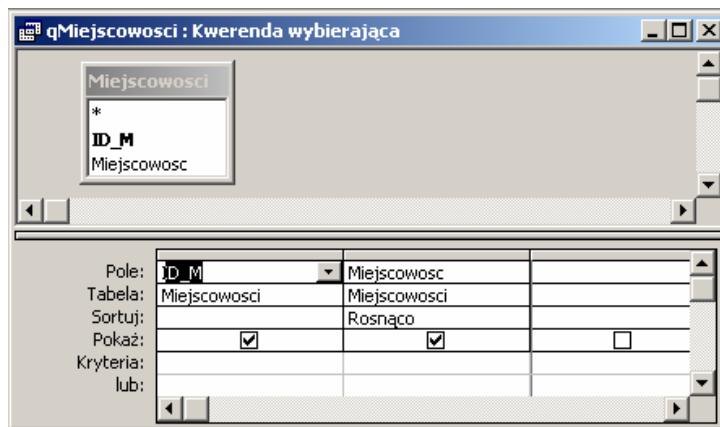
Ostatnie trzy pola (tekstowe) możemy po prostu przeciągnąć do sekcji szczegółów, ale nie możemy tego zrobić z polem ID_M (dokładnie mówiąc możemy, tylko to nie jest żadne rozwiązanie).

Rozwiązaniem będzie wykorzystanie kontroli *ComboBox*, ale przed jej wykorzystaniem musimy przygotować źródło danych dla tej kontrolki. Źródłem tym może być tabela Miejscowosci ze swoimi polami ID_M i NazwaM, ale znacznie lepszym rozwiązaniem będzie zbudowanie kwerendy zwracającej te pola. Przewaga kwerendy nad tabelą jako źródłem danych dla kontrolek typu związanej listy wynika po prostu z tego, że kwerenda może zwrócić dane posortowane alfabetycznie po polu wyświetlonym w tego typu kontrolce.

W standardowy sposób rozpoczynamy tworzenie nowej kwerendy w widoku projektu.



Po włączeniu do projektu kwerendy tabeli Miejscowosci umieszczamy na liście zwracanych pól kolejno oba jej pola, z tym, że dla pola Miejscowosc dysponujemy sortowaniem rosnącym. Na zakończenie zapisujemy projekt pod nazwą qMiejscowosci (litera q jest prefiksem dla obiektów typu kwerenda)



Po powrocie do okna projektu formularza umieszczamy w sekcji *Szczegóły* kontrolę listy rozwijanej, po jej umieszczeniu (zwolnieniu myszy) MS Access wyświetla okno kreatora pól kombi. Możemy dalej korzystać z kreatora, lepszym rozwiązaniem będzie jego anulowanie i „ręczne” skonfigurowanie tej kontroli.



Istotne właściwości, które musimy skonfigurować to kolejno: *Nazwa*, *Źródło formantu*, *Typ źródła wierszy*, *Źródło wierszy*, *Liczba kolumn*, *Szerokości kolumn*, *Kolumna związana* i *Ogranicz do listy*.

Nazwa – przypisuje kontrolce (formantowi) nazwę pod jaką dana kontrola jest rozpoznawana przez aplikację – w naszym przypadku jest to *cboMiejscowosci* (cbo jest prefiksem dla kontrolek typu rozwijanej listy).

Źródło formantu – jest nazwą pola w źródle danych z którą związana jest dana kontrola – w naszym przypadku jest to pole *ID_M*,

Typ źródła wierszy – określa rodzaj danych dostępnych dla rozwijanej listy, standardowo jest to Tabela/kwerenda.

Źródło wierszy – nazwa obiektu dostarczającego dane do kontrolki, musi być w logicznej zgodności z typem źródła – u nas jest to kwerenda *qMiejscowosci*.

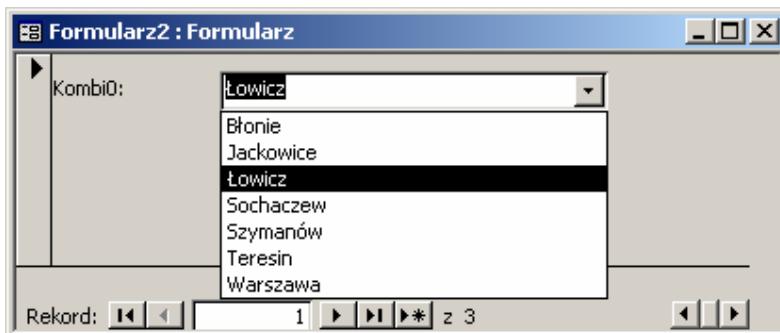
Liczba kolumn – określa liczbę pól (kolumn) dostępnych w źródle wierszy – tu 2.

Szerokości kolumn – określa szerokości każdego z pól, jeżeli jakieś pole chcemy ukryć, to jego szerokość ustawiamy na 0 jednostek – u nas 0 cm dla pola *ID_M* i 4 cm dla pola *Miejscowosc*.

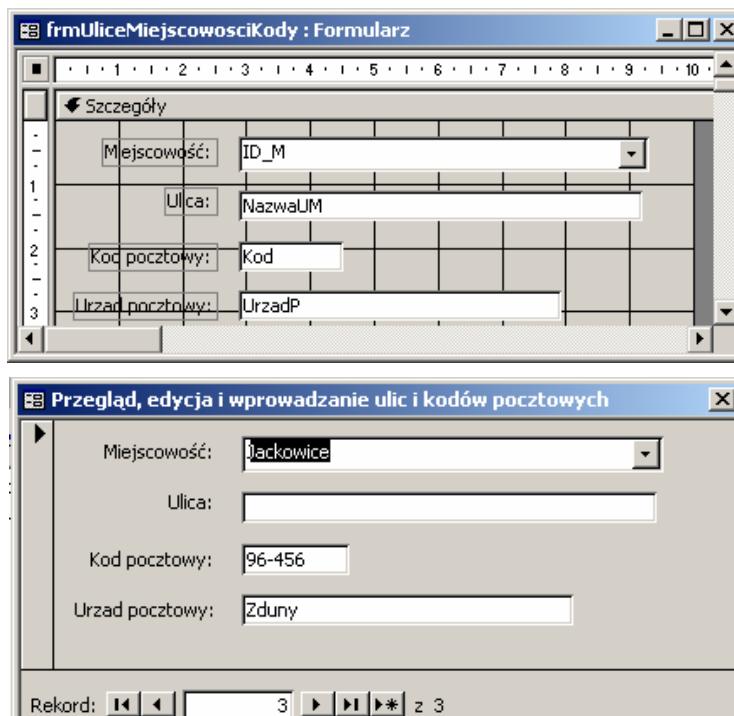
Kolumna związana – określa numer kolumny w źródle wierszy, której wartość zostanie przekazana do źródła formantu – u nas jest to 1, tym samym kontrolka zwraca zawartość pola *ID_M* odpowiadającą wybranej miejscowości.

Ogranicz do listy – ustawiona na Tak nie pozwala na dodawanie nowych pozycji do rozwijanej listy – u nas ustawiona na Tak z tego powodu, że źródłem danych dla kontroli są dane dwukolumnowe.

Po tych kilku ustawieniach nasza kontrola już funkcjonuje wyświetlając w swojej liście nazwy miejscowości pobieranych w sposób dynamiczny z tabeli *Miejscowosci*.



Po umieszczeniu w sekcji *Szczegóły* pozostałych trzech pól, ewentualnej modyfikacji ich etykiet oraz wyrównaniu wszystkich kontrolek mamy gotowy projekt formularza dla obsługi tabeli UliceMiejscowosciKody. Poniżej widok projektu tego formularza oraz widok formularza na ekranie (w widoku Dane)

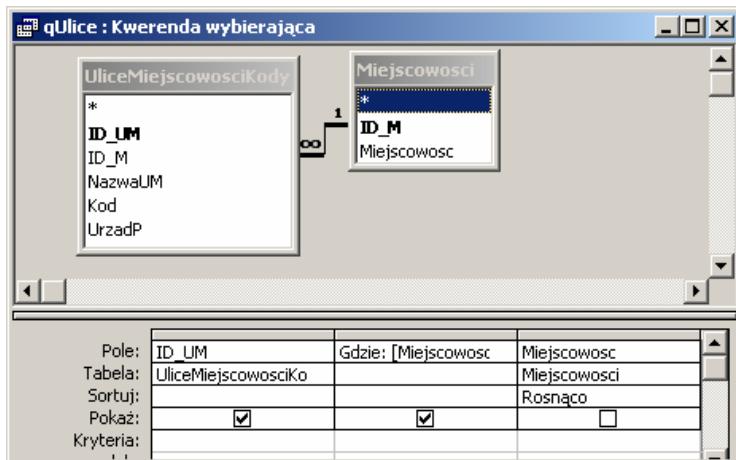


W praktyce, przy wykorzystywaniu tego formularza zarówno do edycji istniejących rekordów jak i do wprowadzania nowych może pojawić się jeden, dość trudny problem. W trakcie wprowadzania nowych danych może okazać się, że na liście miejscowości nie znajdziemy potrzebnej nazwy – bo mamy nowego klienta z miejscowości, której nie ma jeszcze w tabeli Miejscowosci. Tym samym nie możemy wprowadzać nazwy ulicy ani jej kodu, najpierw musimy wprowadzić miejscowości i dopiero później ulice i jej kod.

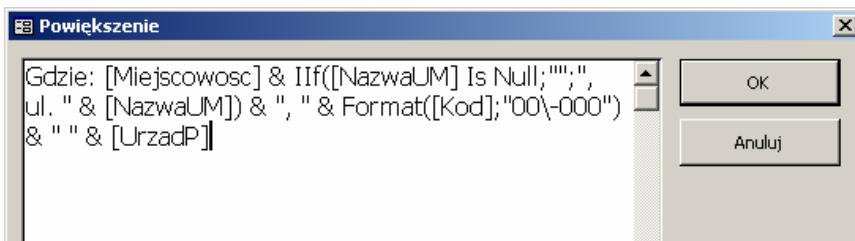
Na etapie obecnego rozwiązania musimy zamknąć otwarty formularz frmUliceMiejscowosciKody, otworzyć formularz frmMiejscowosci, dodać brakującą miejscowości i ponownie otworzyć formularz dodawania ulic i ich kodów. Można temu zaradzić, ale będzie to wymagało wykorzystania języka VBA. Wróćmy do rozwiązania tego problemu w dalszej części tego skryptu

W podobny sposób tworzymy formularze dla takich tabel jak: Klienci, Telefony, AdresyMailowe. Poniżej projekty tych formularzy wraz z projektami kwerend dostarczających danych do kontrolek typu rozwijanej listy.

Przed rozpoczęciem prac nad formularzem dla tabeli Klienci musimy zaprojektować kwerendę, która będzie źródłem danych dla listy rozwijanej pomagającej ustalić ulicę w danej miejscowości i jej kod pocztowy. Projekt takiej kwerendy pokazany jest poniżej. Kwerenda o nazwie qUlice oparta jest o tabelę UliceMiejscowosciKody oraz tabelę Miejscowosci.



Z tablicy UliceMiejscowosciKody kwerenda będzie zwracać pole ID UM, pole wyliczane Gdzie będzie zwracać jawnego odpowiednika pola ID UM skomponowany z wartościami takich pól jak Miejscowosc, NazwaUM, Kod i UrzadP. Pełna konstrukcja pole wyliczanego Gdzie pokazana jest poniżej (jest to fragment okna Powiększenie).



Pole Gdzie wykorzystuje dwie funkcje VBA; funkcja IIF bada, czy pole NazwaUM nie jest puste, jeżeli tak, to zwraca pusty ciąg znaków, jeżeli nie, to zwraca ciąg „, ul.” wraz z nazwą ulicy. Funkcja Format została wykorzystana do zwrócenia kodu pocztowego zgodnie z przyjętym schematem jego zapisu. Dodatkowo w projekcie

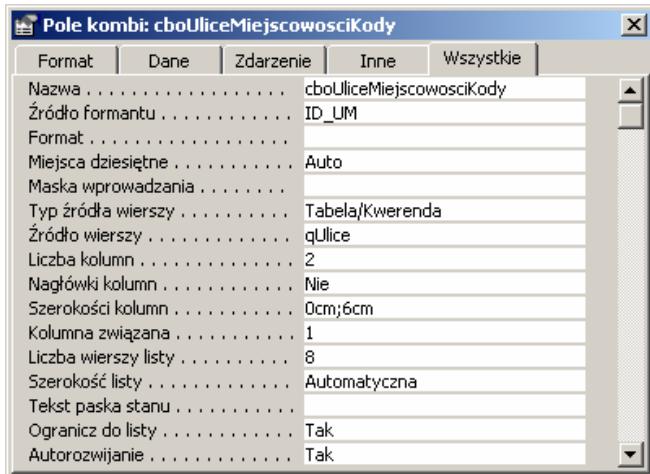
umieszczone zostało pole **Miejscowość**, ale z opcją bez pokazywania wartości tego pola, za to z sortowaniem rosnącym. Inaczej mówiąc zadaniem tego pola jest posortowanie zwracanych rekordów wg miejscowości, co powinno ułatwić ich wyszukiwanie w liście. Poniżej pokazane są informacje zwracane przez tak zaprojektowaną kwerendę.

	ID UM	Gdzie
▶	3	Jackowice, 96-456 Zdyny
	2	Łowicz, ul. Bratkowice, 23-346 Łowicz
	1	Łowicz, ul. Zduńska, 23-345 Łowicz

Możemy już rozpocząć projektowanie formularza dla tabeli **Klienci**, jego projekt pokazany jest poniżej. W oknie właściwości formularza, podobnie jak we wcześniejszych przypadkach, ustawiamy takie właściwości jak: *Tytuł* – „Przeglądanie, edycja i wprowadzanie danych klientów”, *Paski przewijania* – Żaden oraz *Przyciski Min Max* – Żaden.

W sekcji *Szczegóły*, wykorzystując okienko listy pól źródła danych formularza, techniką „ciagnij i upuść” umieszczone zostały kontrolki takich pól jak: **Nazwisko**, **Imię**, **DataU**, **NrDomu** i **NrMieszk**. W przypadku pola **ID UM** została zastosowana kontrolka typu rozwijana lista, której właściwości zostały zdefiniowane „ręcznie” (z pominięciem kreatora).

Podobnie jak w przypadku wcześniejszego formularza definiujemy takie właściwości rozwijanej listy jak *Nazwa*, *Źródło formantu*, *Typ źródła wierszy*, *Źródło wierszy*, *Liczba kolumn*, *Szerokości kolumn*, *Kolumna związana* i *Ogranicz do listy*. Poniżej pokazane jest okno właściwości tej kontrolki z ustawionymi wartościami.



A tak wygląda formularz `frmKlienci` w widoku Dane.

Formalnie jest wszystko w porządku, możemy w elegancko sposób przeglądać, edytować czy wprowadzać dane do tabeli `Klienci`, ale co z telefonami klienta czy jego adresami mailowymi?

Wydaje się, że idealnym rozwiązaniem byłoby zbudowanie takiego formularza dla danych klienta, który pozwalałby na dostęp nie tylko do danych tabeli `Klienci`, ale także tabel `Telefony` oraz `AdresyMailowe`. Budowaniem takich złożonych formularzy poświęcony będzie kolejny rozdział tego skryptu. Prześledzimy ich budowę kontynuując prace nad formularzem `frmKlienci`.

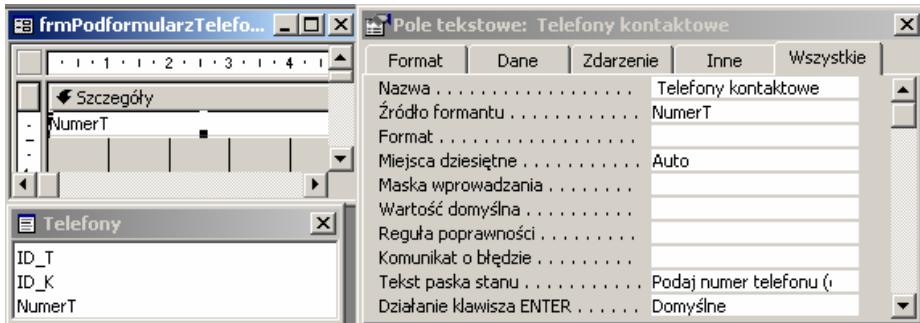
5.3.3. Formularze złożone

Będziemy dalej pracować nad projektem formularza `frmKlienci`, a naszym celem będzie umieszczenie w nim dodatkowych formularzy, jednego dla obsługi telefonów, a drugiego do obsługi adresów mailowych. Formularze te mają umożliwić dostęp do tabel `Telefony` i `AdresyMailowe`, proszę zauważyc, że w relacjach z tabelą `Klienci` te dwie tabele występują po stronie wiele. Naszym zamiarem będzie takie zmodyfikowanie formularza `frmKlienci`, aby przy każdej zmianie rekordu w tym formularzu (czyli po stronie jeden w relacji) były pokazywane powiązane rekordy z tabel `Telefony` i `AdresyMailowe`.

Tak zbudowane formularze, jak wiemy, noszą nazwę złożonych. Formularz obsługujący stronę jeden w relacji jest formularzem głównym, a formularze obsługujące stronę wiele są podformularzami. Formularz główny z podformularzami jest powiązany poprzez pole wykorzystywane do ustanowienia relacji między ich źródłami danych.

Formularz główny już mamy, jest to oczywiście `frmKlienci`, musimy teraz przygotować dwa formularze dla tabel `Telefony` i `AdresyMailowe`. Formularze te, z uwagi na ich przeznaczenie (jako podformularze), przygotujemy w widoku arkusza danych a nie w widoku pojedynczego formularza. Chodzi o to, aby mając w formularzu głównym danego klienta widzieć w podformularzach wszystkie jego telefony czy adresy mailowe.

Poniżej pokazany jest projekt formularza `frmPodformularzTelefony`, w sekcji `Szczegóły` umieszczone jest jedynie pole tekstowe skojarzone z polem `NumerT` tabeli `Telefony`. Z uwagi na widok arkusza danych nie jest potrzebna etykieta opisująca to pole. Rozmiary tego formularza (szerokość) zostały tak dobrane, aby mógł spełnić rolę podformularza w formularzu `frmKlienci` (chodzi o proporcje jego szerokości do rzeczywistych potrzeb i rozmiarów formularza głównego). Obok projektu formularza pokazanego jest okno właściwości pola tekstowego, istotna jest tu właściwość `Nazwa`. Tekst wpisany w tym polu będzie także opisem kolumny danych w tym podformularzu, stąd symbol spacji czy ewentualnie polskie znaki w nazwie.



Warto jeszcze spojrzeć na okno właściwości tego formularza, istotne właściwości to: *Źródło rekordów*, *Widok domyślny* oraz *Paski przewijania*. Taka właściwość jak *Tytuł*, choć została zdefiniowana, z uwagi na zamiar wykorzystania tego formularza jako podformularza jest nieistotna – pasek tytułowy i tak nie będzie wyświetlany.

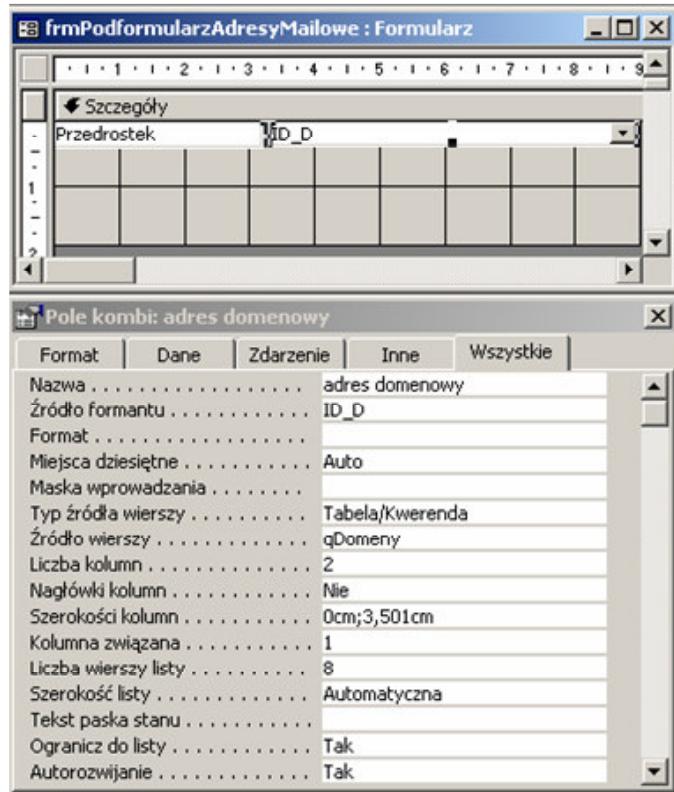


W podobny sposób przygotujemy formularz (podformularz) wyświetlający adresy mailowe, z tym, że z uwagi na kontrolkę rozwijanej listy z adresami domen zaczniemy od przygotowania odpowiedniej kwerendy. Projekt kwerendy o nazwie qDomeny pokazany jest poniżej.



W standardowy sposób zaczynamy projektować formularz oparty na tabeli AdresyMailowe, oczywiście musimy pamiętać o tym, żeby właściwość *Widok domyślny* ustawić na Arkusz danych.

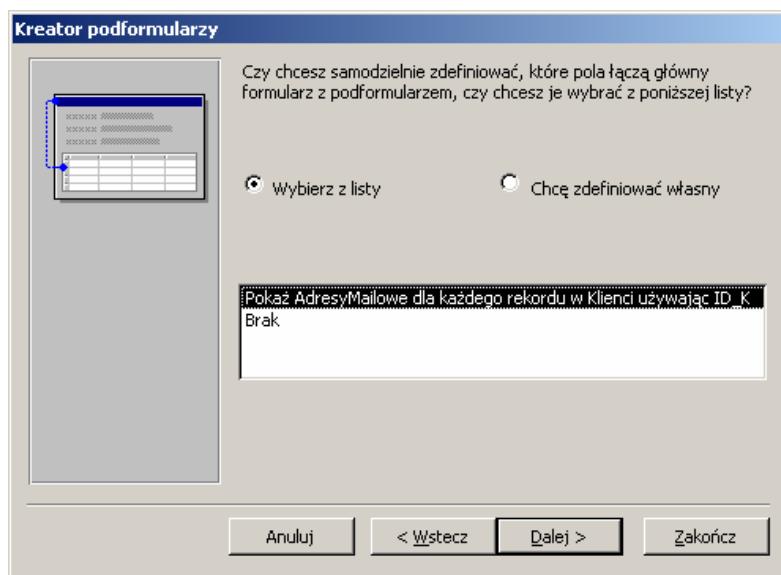
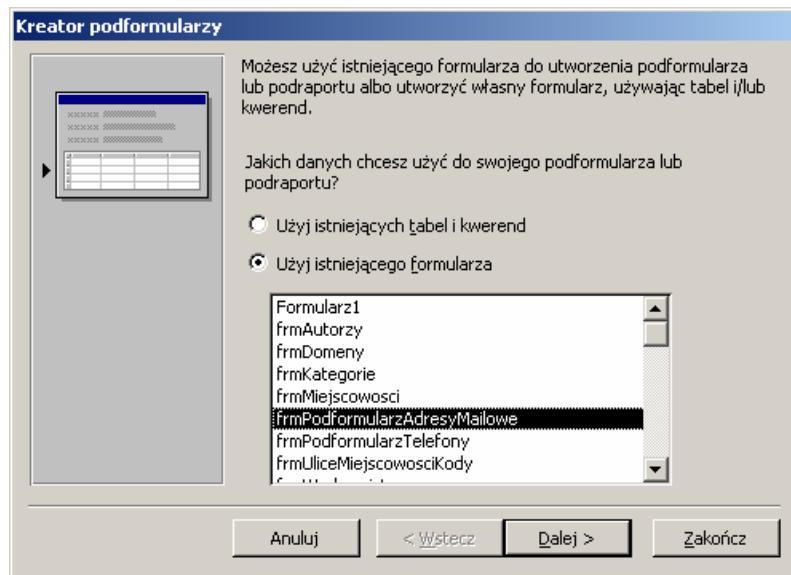
W sekcji *Szczegóły* umieszcza się pole tekstowe, dla którego źródłem danych będzie pole Przedrostek oraz kontrolkę typu rozwijanej listy. Poniżej widok projektu tego formularza oraz okno właściwości kontrolki combo box.



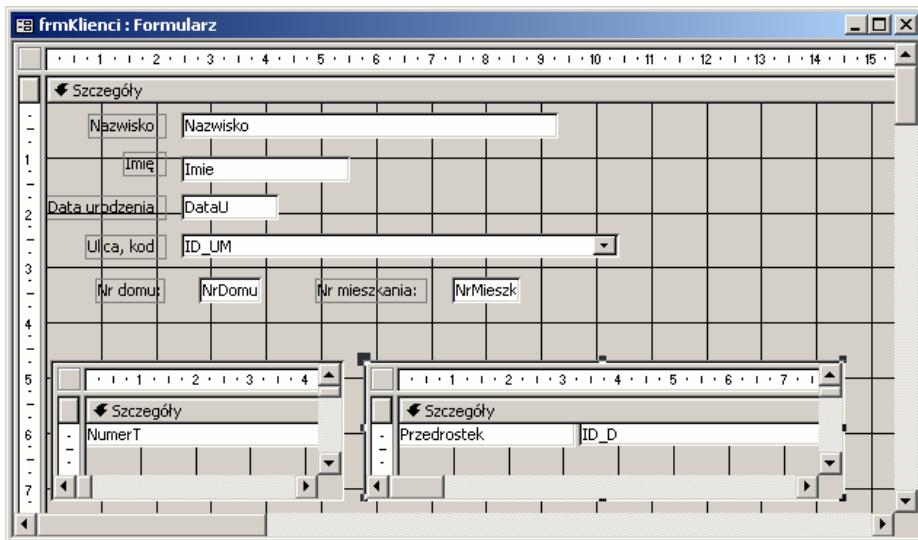
Podobnie jak w poprzednim podformularzu zmienione zostały właściwości obu formantów (czyli pola tekstu i pola kombi) tak, aby mogły pełnić rolę etykiet opisujących dwie kolumny wyświetlane w tym podformularzu.

Oba potrzebne podformularze już mamy, można więc powrócić do zmodyfikowania projektu formularza frmKlienci. Poniżej pokazane będą kolejne etapy pracy związanej złączeniem podformularza frmPodformularzAdresyMailowe oraz ustanowieniem połączenia między formularzem głównym i podformularzem.

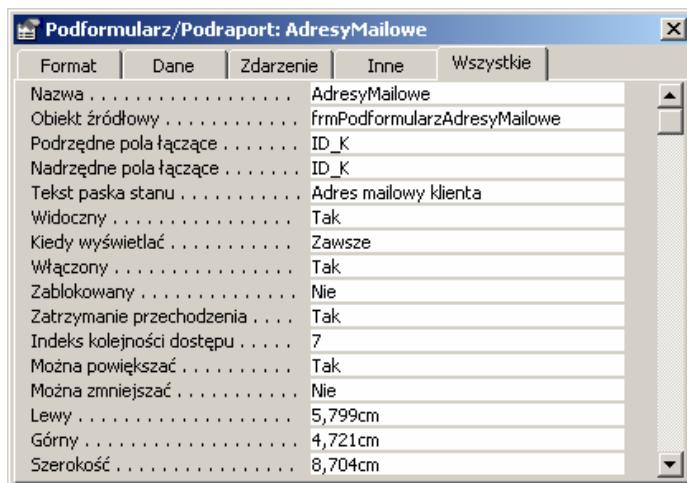
Początek pracy, ikona podformularza została przeciągnięta do formularza głównego, MS Access wyświetlił pierwsze okno kreatora podformularzy.



Po zakończeniu pracy kreatora podformularzy w analogiczny sposób umieszczamy drugi podformularz do wyświetlania numerów telefonów. Poniżej widok projektu tak zmodyfikowanego formularza frmKlienci.



W oknie właściwości podformularza frmPodformularzAdresyMailowe możemy prześledzić dwie ważne właściwości: *Podrzędne pola łączące* oraz *Nadrzędne pola łączące*.



Poniżej pokazany jest formularz frmKlienci w widoku Dane, łatwo zauważyc, że znakomicie została zwiększoną jego funkcjonalność. W tej chwili przy każdej zmianie rekordu w formularzu głównym w podformularzach zostaną wyświetlane powiązane z nim rekordy numerów telefonów oraz adresów mailowych.

Pokazany wyżej formularz jest ciekawy także i tego powodu, że wyświetla informacje z trzech różnych tabel. Jest także ciekawy również dlatego, że pozwala nam na wprowadzanie danych do trzech różnych tabel.

W rozdziale poświęconym makropoleceniom wróćmy raz jeszcze do projektu tego formularza w celu dalszego zwiększenia jego funkcjonalności. Między innymi będziemy chcieli stworzyć sobie możliwość kontynuowania wprowadzania czy edycji danych dla danego klienta w takiej sytuacji, gdy np. nie znajdziemy w tabeli Domeny odpowiedniego wpisu. Podobna sytuacja dotyczyć będzie braku odpowiedniej pozycji określającej miejscowości, nazwę ulicy i jej kod. Inny element funkcjonalności to rozdzielenie wprowadzania danych nowego klienta od przeglądu czy edycji istniejących danych. Formalnie można to zrobić bez potrzeby korzystania z procedur VBA, ale wymaga to tworzenia oddzielnych formularzy przeznaczonych wyłącznie do wprowadzania danych. Rozwiążanie takie jest mało eleganckie, a poza tym niepotrzebnie zwiększa rozmiar bazy danych.

Jak wcześniej powiedzieliśmy wróćmy do tej tematyki w rozdziale poświęconym makropoleceniom, ale wcześniej jeszcze zacznijemy prace nad kolejnym, dużym formularzem złożonym przeznaczonym do obsługi tabeli Zakupy i tabel z nią powiązanych.

Pracę nad formularzem frmZakupy zaczniemy od przygotowania źródeł danych dla dwóch kontrolek typu rozwijanej listy, z których będziemy wprowadzać takie informacje jak wydawnictwo czy kategoria zakupionej pozycji. Projekty obu kwerend pokazane są poniżej.

The image shows two separate query builder windows side-by-side.

Top Window (qWydawnictwa):

- Panel Top:** Wydawnictwa, *.
- Panel Bottom:**
 - Pole: ID_W
 - Tabela: Wydawnictwa
 - Sortuj: Rosnąco
 - Pokaż:
 - Kryteria: ...

Bottom Window (qKategoria):

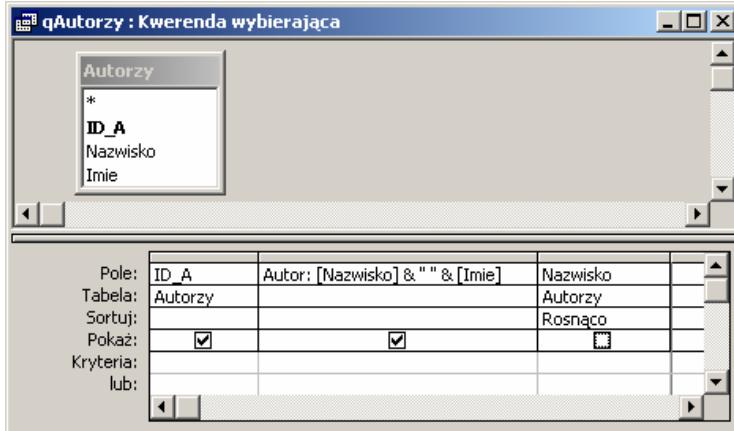
- Panel Top:** Kategorie, *.
- Panel Bottom:**
 - Pole: ID_K
 - Tabela: Kategorie
 - Sortuj: Rosnąco
 - Pokaż:
 - Kryteria: ...

Proszę zwrócić uwagę na konieczność przygotowania obu kwerend wynikającą z możliwości posortowania danych wyświetlanych w liście kontrolki typu kombo. Gdyby nie ta konieczność, równie dobrze źródłem danych dla obu pól kombi mogłyby być tabele Wydawnictwa oraz Kategorie.

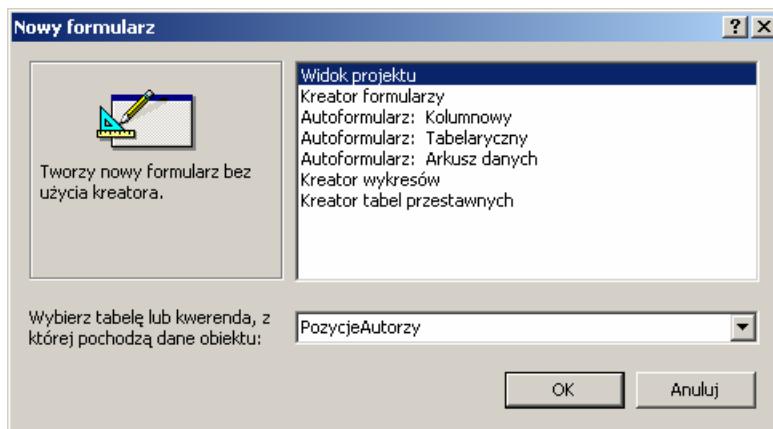
Będzie nam także potrzebny podformularz pozwalający na przeglądanie czy edycję danych autorów zakupionych pozycji. Podformularz ten przygotujemy w widoku danych, a jego jedynym formantem (kontrolką) będzie pole kombi wyświetlające dane pobierane z tabeli Autorzy.

Prace nad podformularzem zaczniemy od przygotowania źródła danych dla jego kontrolki typu kombo. W polu listy tej kontrolki chcemy mieć posortowane wg nazwiska

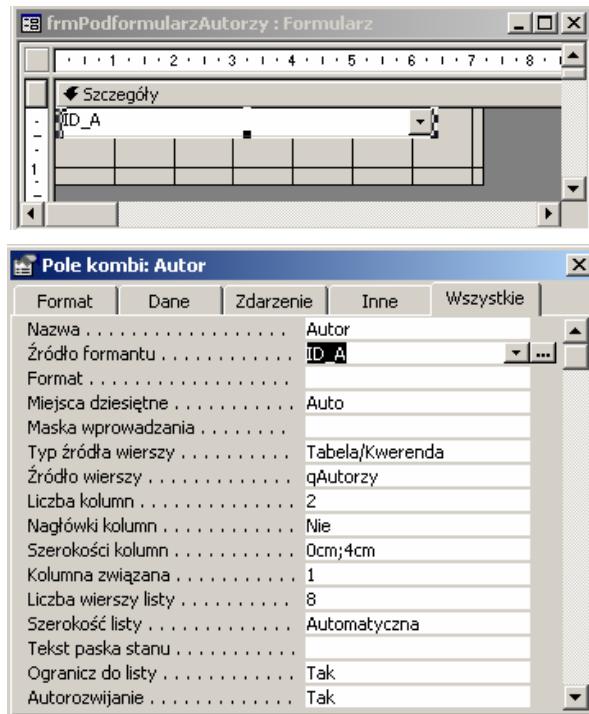
dane autorów. Zbudujemy kwerendę pracującą na polach tabeli Autorzy, kwerenda ta zwróci nam pole **ID_A** oraz pole wyliczane **Autor** zbudowane wg pokazanej niżej formuły. Pole **Nazwisko** zostało wykorzystane jedynie do posortowania zwracanych rekordów, natomiast same wartości tego pola nie będą zwracane.



Możemy już utworzyć projekt podformularza w widoku arkusza danych z polem kombi w sekcji **Szczegóły**. Podformularz ten będzie oparty o tabelę **PozycjeAutorzy**, wartości pola **ID_Z** tej tabeli będą pobierane z formularza głównego, a pola **ID_A** z rozwijanej listy, którą umieścimy w podformularzu.



Poniżej pokazany jest projekt podformularza oraz okno właściwości pola kombi umieszczonego w sekcji **Szczegóły**.



W zasadzie możemy już zacząć prace nad formularzem frmZakupy. Po otwarciu projektu formularza (na tabeli Zakupy) w sekcji Szczegóły umieszczamy wszystkie potrzebne kontrolki i przygotowany podformularz dla wyświetlania (wprowadzania) autorów zakupionych pozycji.

Pole tekstowe skojarzone z polem Tytuł tabeli Zakupy warto tak powiększyć, aby można było wpisać potencjalnie długi tytuł książki. W pokazanym dalej widoku projektu tego formularza pole to zostało tak powiększone, aby można było wpisać do dwóch wierszy tytułu.

Poza pokazanymi kontrolkami i podformularzem formularz frmZakupy powinien umożliwić nadanie sygnatur zakupionym pozycjom. Sygnatur tych powinno być oczywiście tyle, ile egzemplarzy danej pozycji zostało zakupionych, oczywiście najlepszym rozwiązaniem byłoby automatyczne utworzenie sygnatur w momencie zarejestrowania danego rekordu w tabeli Zakupy. Na poziomie „zwykłego” wykorzystywania MS Access nie jest to możliwe, ale przy wykorzystaniu VBA oczywiście tak. Za chwilę w rozdziale poświeconym makropoleceniom zajmiemy się tym problemem, a teraz możemy zobaczyć projekt i formularz frmZakupy w widoku Dane.

frmZakupy : Formularz

Szczegóły				
Tytuł:	<input type="text" value="Tytuł"/>			
Wydawnictwo:	<input type="text" value="ID_W"/>			
Kategoria:	<input type="text" value="ID_K"/>			
ISBN:	<input type="text" value="ISBN"/> Stron:	<input type="text" value="Stron"/>	Data zakupu:	<input type="text" value="DataZ"/>
Cena:	<input type="text" value="Cena"/> Ile:	<input type="text" value="Ile"/>		
<input type="button" value="Szczegóły"/> <input type="text" value="ID_A"/>				

Przegląd, edycja i wprowadzanie danych o zakupionych pozycjach

Tytuł:	<input type="text" value="Projektowanie baz danych w MS Access"/>				
Wydawnictwo:	<input type="text" value="Wydawnictwo WSiM w Sochaczewie"/>				
Kategoria:	<input type="text" value="Programowanie baz danych"/>				
ISBN:	<input type="text" value="123-345-23"/>	Stron:	<input type="text" value="354"/>	Data zakupu:	<input type="text" value="2004-04-17"/>
Cena:	<input type="text" value="35,00 zł"/>	Ile:	<input type="text" value="2"/>		
Autor <input type="text" value="Górzczyński Janusz"/> <input type="button" value="Rekord: 1 < > 2 >> >>> z 1 z 2 "/>					
Rekord: <input type="text" value="1"/> <input type="text" value="2"/> <input type="text" value="3"/> <input type="text" value="4"/> <input type="text" value="5"/> <input type="text" value="6"/> <input type="text" value="7"/> <input type="text" value="8"/> <input type="text" value="9"/> <input type="text" value="10"/> <input type="text" value="11"/> <input type="text" value="12"/> <input type="text" value="13"/> z 1					

5.4. Makropolecenia i procedury VBA

5.4.1. Modyfikacja frmZakupy

W poprzednim rozdziale projektowaliśmy formularz frmZakupy, projekt ten będzie zakończony w momencie, gdy stworzymy możliwość automatycznego generowania sygnatur zakupionych pozycji. Przed rozpoczęciem prac musimy ustalić format sygnatury, przy czym warto już rozważyć możliwość wykorzystania kodów paskowych.

W naszym przekonaniu sygnatura powinna dostarczyć takich informacji:

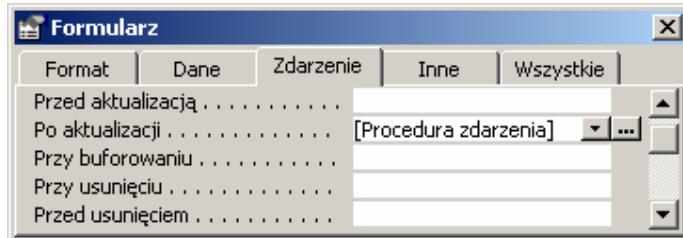
Rok zakupu danej pozycji – 4 cyfry,

Numer kolejny tej pozycji w tabeli zakupy (`ID_P`) wyświetlony w oparciu o maskę np. 6 cyfr, co wystarczy na 999 999 pozycji w naszej bibliotece,

Numeru egzemplarza w ramach danej pozycji wyświetlanego w oparciu o maskę 2 cyfr.

Przykładowo sygnatura może mieć postać: 199800012302 (rok zakupu 1998, `ID_P` równe 123, egzemplarz nr 2).

Wiemy już jak może wyglądać sygnatura, kolejne pytanie to kiedy musi być utworzona? Jedynym rozsądny rozwiązaniem będzie przeniesienie fokusu z formularza głównego do podformularza wprowadzania autorów. Sytuacji tej odpowiada zdarzenie *Po aktualizacji*, w takim razie w oknie właściwości formularza frmZakupy w polu tej właściwości ustawiamy opcję Procedura zdarzenia.



Podwójny klik przycisku z trzema kropkami przenosi nas do edytora Visual Basic, gdzie procedura `Form_AfterUpdate` jest już rozpoczęta.

Dostęp do tabeli Sygnatury w celu dodawania nowych rekordów zapewnmy sobie poprzez rekordset otwarty na połączeniu pobranym z bieżącego projektu. Generalnie korzystać będziemy ze standardowej metody dostępu do bazy danych ADO. Rekordy dodawane będą w pętli z licznikiem typu `For Next`, a do ukształtowania sygnatury z zaprezentowanym wyżej wzorem wykorzystamy funkcję `Format`.

```

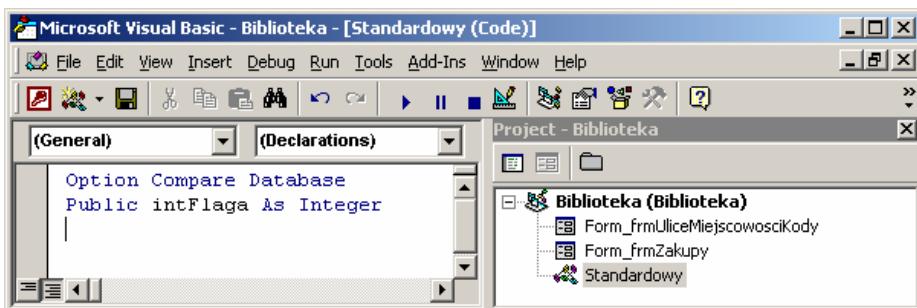
Private Sub Form_AfterUpdate()
    On Error GoTo errHandler
    Dim rst As New ADODB.Recordset, i As Integer
    rst.Open "SELECT * FROM Sygnatury", _
        CurrentProject.Connection, adOpenKeyset, _
        adLockOptimistic
    For i = 1 To CInt(Me.Ile.Text)
        rst.AddNew
        rst!ID_P = Me.ID_P
        rst!Sygnatura = Format(Year(Me.DataZ), "0000") & _
            Format(Me.ID_P, "000000") & Format(i, "00")
        rst!Status = 1
    Next i
    rst.MovePrevious
    rst.Close
    Me.Refresh
    Exit Sub
errHandler:
    MsgBox Err.Description
End Sub

```

Pozostaje jeszcze przygotowanie odpowiedniego podformularza wyświetlającego sygnatury związane z daną pozycją i ten fragment formularza frmZakupy jest gotowy.

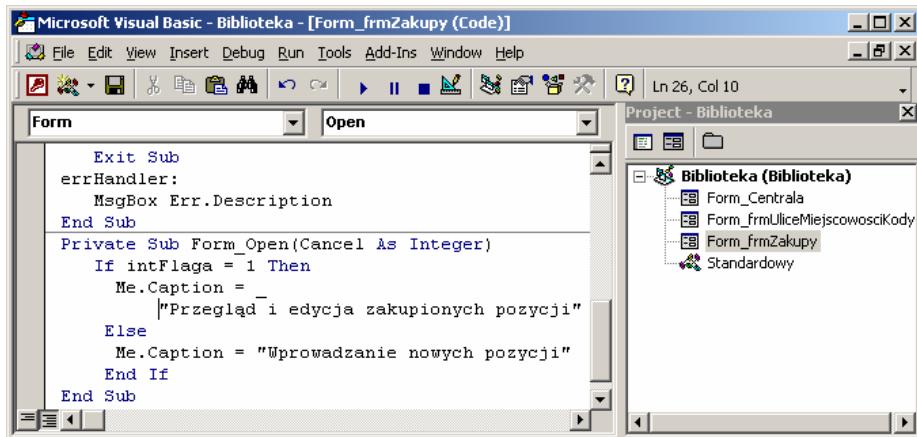
Autor		Sygnatura	
Górzczyński Janusz	*	200400000901	
*		200400000902	
		200400000903	
Rekord: [Navigation Buttons] 1 [Navigation Buttons] z 1		*	
Rekord: [Navigation Buttons] 8 [Navigation Buttons] z 8			

Wcześniej wspominaliśmy, że dobrym rozwiązaniem jest rozdzielenie czynności wprowadzania nowych rekordów od czynności przeglądania i edycji już istniejących. Eleganckim rozwiązaniem będzie programowa modyfikacja właściwości *Dodawanie dozwolone* wraz z odpowiednią modyfikacją tytułu formularza. Prace nad tym fragmentem modyfikacji formularza zaczniemy od zadeklarowania pomocniczej zmiennej o charakterze flagi wskazującej sposób uruchomienia formularza. Zmienna ta powinna być dostępna dla różnych obiektów naszej bazy danych, stąd musi być zadeklarowana w module ogólnym edytora VBA. Po wstawieniu do projektu modułu i zmianie jego nazwy na *Standardowy* w sekcji deklaracji została zadeklarowana zmienna typu *Integer* o nazwie *intFlaga*.

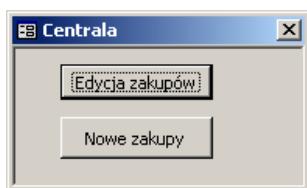


Możemy dalej przyjąć zasadę, że jeżeli zmienna *intFlaga* ma wartość 1, to formularz otwierany jest do przeglądu i edycji; jeżeli wartość 2, to formularz otwierany jest w trybie wprowadzania nowych danych.

Stosowne procedury modyfikujące sposób otwarcia formularza muszą być wykonywane w odpowiedzi na zdarzenie *Przy otwarciu*, dla formularza *frmZakupy* procedura ta może mieć pokazaną niżej postać.



Dla przetestowania naszego rozwiązania możemy stworzyć pomocniczy formularz z dwoma przyciskami poleceń, ich procedury będą modyfikować zmienną intFlaga i zależnie od jej wartości wywoływać formularz frmZakupy. Poniżej widok tego formularza oraz procedury uruchamiane przez klik obu przycisków.



```
Private Sub cmdZakupyEdit_Click()
    intFlaga = 1
    DoCmd.OpenForm "frmZakupy", _
        acNormal, , , acFormEdit
End Sub

Private Sub cmdZakupyAdd_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmZakupy", _
        acNormal, , , acFormAdd
End Sub
```

Po przypisaniu zmiennej intFlaga odpowiedniej wartości wywoływane jest polecenie DoCmd z metodą OpenForm, której poszczególne atrybuty precyzują sposób wykonania tego polecenia. W momencie wywołania formularza frmZakupy jego procedura modyfikuje dodatkowo tytuł formularza.

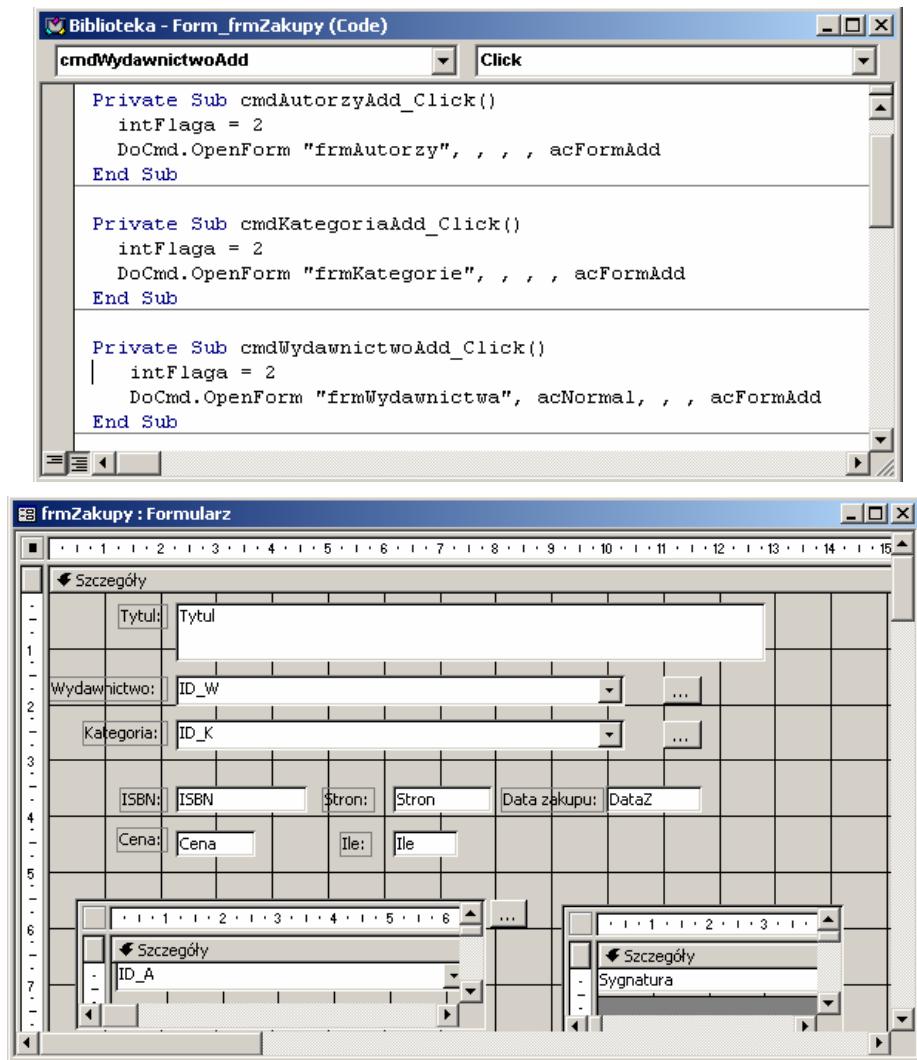
Aby formularz frmZakupy był w pełni funkcjonalny, musimy jeszcze stworzyć możliwość dodawania z jego poziomu nowego wydawnictwa, kategorii oraz autorów. Będzie to wymagało dodania w formularzu frmZakupy trzech przycisków poleceń, których procedury będą wywoływać potrzebne formularze dla wydawnictw, kategorii czy autorów. Konieczna będzie także modyfikacja tych trzech formularzy w taki sposób, aby były tylko do wprowadzania nowych danych, oraz aby po zapisie rekordu nastąpił powrót do formularza wywołującego (czyli frmZakupy) wraz z odpowiednią aktualizacją danych wykorzystywanych w polach kombi.

W projekcie formularza frmZakupy wprowadzimy następujące zmiany: polom kombi odpowiedzialnym za wprowadzanie wydawnictw i kategorii zmienimy nazwy na cboWydawnictwa oraz cboKategorie. Robimy to dla czystości kodu procedur, które musimy utworzyć w formularzach frmWydawnictwa oraz frmKategorie, ich zadaniem będzie odświeżenie źródeł wierszy tych dwóch pól kombi po dodaniu nowego wydawnictwa czy kategorii.

Podobne działania musimy także zrobić w odniesieniu do pola kombi podformularza frmPodformularzAutorzy, z tym, że w tym wypadku pozostawimy dotychczasową nazwę Autor (z uwagi na to, że jest wykorzystywana jednocześnie jako opis całego podformularza).

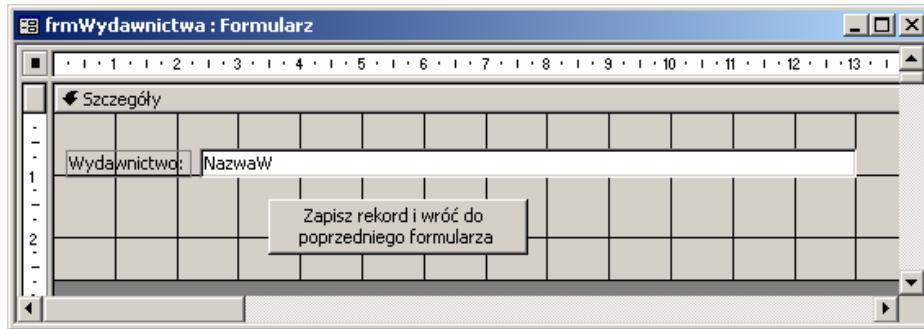
Zaczynamy od umieszczenia na prawo od pól kombi cboWydawnictwa i cboKategorie oraz na prawo od podformularza frmPodformularzAutorzy trzech przycisków poleceń o nazwach cmdWydawnictwaAdd, cmdKategorieAdd

oraz cmdAutorzyAdd. Właściwość *Tytuł* każdego z tych przycisków ustawiamy na symboliczne trzy kropki, a właściwość *Przy kliknięciu* na [Procedura zdarzenia]. Kolejny krok, to utworzenie tych trzech procedur, ich zadaniem będzie otwarcie odpowiednich formularzy w trybie dodawania nowych rekordów. Poniżej te procedury oraz widok projektu formularza frmZakupy z przyciskami poleceń.



5.4.2. Modyfikacja formularzy frmWydawnictwa i frmKategorie

Modyfikacja obu formularzy będzie polegała na tym, aby w sekcji *Szczegóły* umieścić przyciski poleceń wykorzystywane wtedy, gdy formularze te będą otwierane z poziomu innego formularza. Zadaniem procedur uruchamianych poprzez klik tych przycisków będzie zapisanie rekordu ze sprawdzeniem poprawności wprowadzonej informacji, odświeżenie odpowiedniego źródła danych w formularzu wywołującym oraz zamknięcie danego formularza i powrót do formularza wywołującego. Przygotujemy także procedury uruchamiane w momencie otwierania tych formularzy, ich zadaniem będzie modyfikacja tytułu formularza w zależności od wartości zmiennej *intFlaga*.



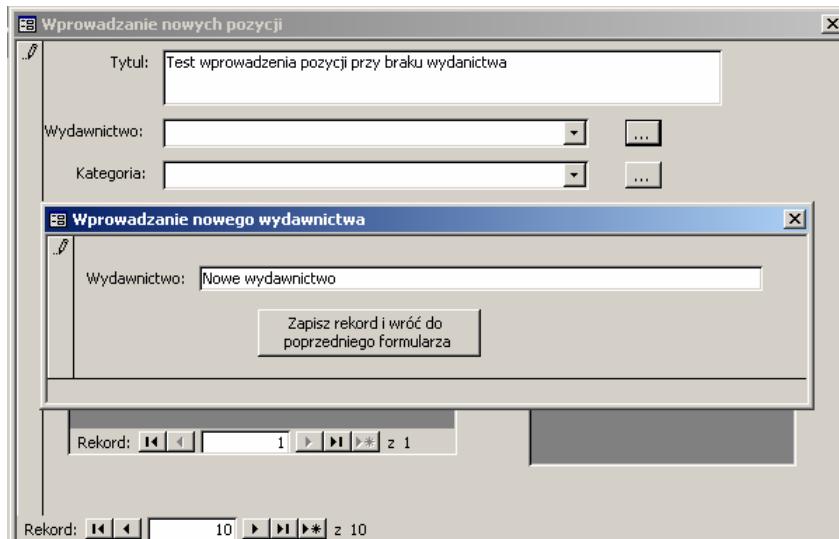
W pokazanym wyżej projekcie formularza *frmWydawnictwa* dodaliśmy przycisk polecenia o nazwie *cmdZapisz*, **przycisk** ten ma być dostępny wtedy, gdy formularz ten będzie wywoływany z poziomu innego formularza w celu dodania nowego rekordu w tabeli *Wydawnictwa*. W takiej sytuacji nie mogą być pokazane przyciski nawigacyjne, ponieważ chcemy wymusić zapisanie wprowadzanego rekordu poprzez ten przycisk właśnie. Poniżej pokazana jest procedura uruchamiana w momencie otwarcia tego formularza. Konstrukcja warunkowa *If* różnicuje sposób otwarcia tego formularza w zależności od wartości zmiennej *intFlaga*.

```
Private Sub Form_Open(Cancel As Integer)
    If intFlaga = 2 Then
        Me.Caption = "Wprowadzanie nowego wydawnictwa"
        Me.NavigationButtons = False
        cmdZapisz.Visible = True
    Else
        Me.Caption = "Przegląd i edycja wydawnictw"
        Me.NavigationButtons = True
        cmdZapisz.Visible = False
    End If
End Sub
```

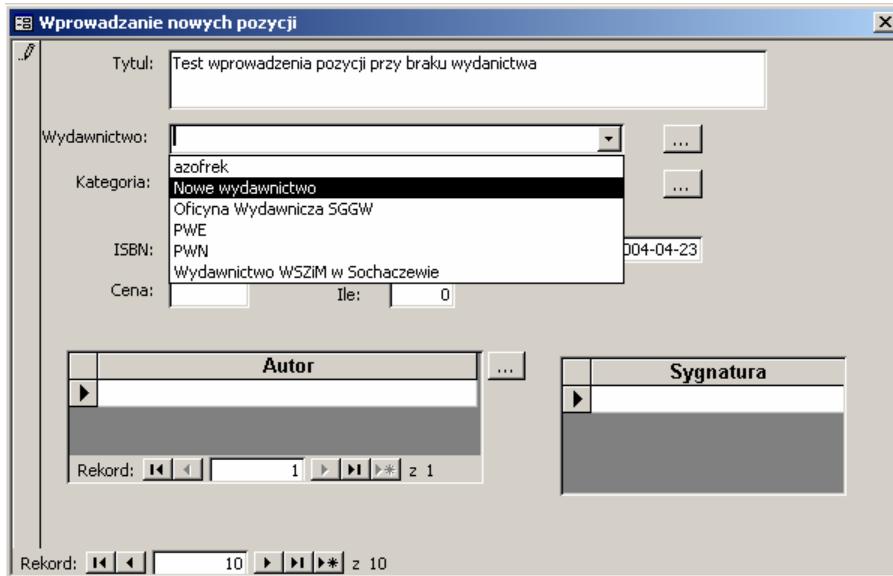
Procedura uruchamiana przyciskiem cmdZapisz musi zrealizować trzy zadania: podjąć próbę zapisania nowego rekordu; po jego zapisaniu odświeżyć źródło danych pola cboWydawnictwa na formularzu frmZakupu, a przy nieudanej próbie zapisu zgłosić stosowny komunikat; zamknąć formularz frmWydawnictwa. Poniżej pokazana jest ta procedura, konstrukcja On Error Resume Next wymusza podjęcie działania przy nieudanej próbie zapisu rekordu – być może spowodowanej tym, że zostanie naruszony indeks (zdublowana wartość pola nazwa wydawnictwa).

```
Private Sub cmdZapisz_Click()
    On Error Resume Next
    Dim ctlList As Control
    DoCmd.GoToRecord
    If Err.Number = 2105 Then
        MsgBox "Takie wydawnictwo już jest lub pole tekstowe
                jest puste!", vbInformation, "Biblioteka"
    Else
        Set ctlList = Forms!frmZakupy.cboWydawnictwa
        ctlList.Requery
    End If
    DoCmd.Close
End Sub
```

Poniżej efekt naszej pracy: z poziomu formularza frmZakupy został otwarty formularz frmWydawnictwa, możemy dodać brakujące wydawnictwo i kontynuować prace nad zarejestrowaniem nowej pozycji.



Nowe wydawnictwo zostało dodane, źródło danych pola kombi pozwalającego na wprowadzanie wydawnictw odświeżone, na liście tego pola znajdziemy już wprowadzone w poprzednim kroku wydawnictwo (tu zaznaczone „Nowe wydawnictwo”).

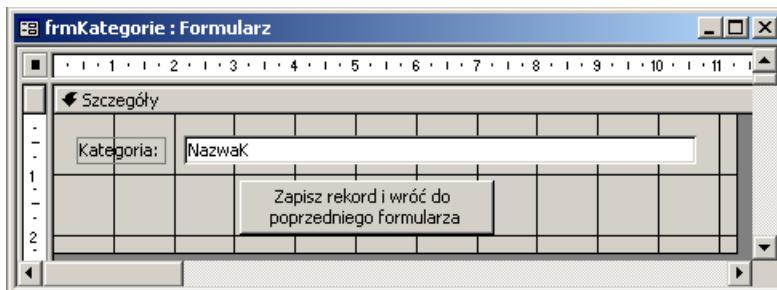


A co się stanie, jeżeli będziemy chcieli wprowadzić jako nowe wydawnictwo np. PWN (już jest taki wpis)? Procedura uruchamiana przyciskiem cmdZapisz obsługuje błąd naruszenia niepowtarzalnego indeksu o numerze 2105 wyświetlając pokazany niżej komunikat.



Po akceptacji przycisku OK formularz zostanie zamknięty i jeżeli chcemy skorygować nasz błąd, to musimy ponownie uruchomić ten formularz. Można zmodyfikować także procedurę cmdZapisz_Clik() tak, aby formularz nie był zamknięty w momencie wystąpienia błędu, a po prostu przenosił fokus ponownie do pola Wydawnictwo oczekując poprawnej nazwy (przed Else dodajemy Exit Sub).

W bardzo podobny sposób modyfikujemy formularz frmKategorie. W projekcie dodajemy przycisk polecenia o nazwie cmdZapisz.



Tworzymy procedurę uruchamianą w momencie otwarcia formularza oraz obsługującą klik przycisku cmdZapisz.

```

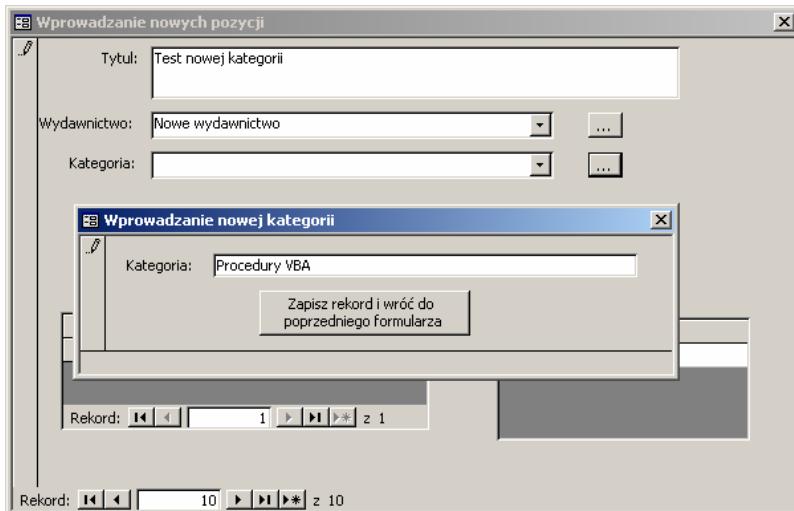
Private Sub cmdZapisz_Click()
    On Error Resume Next
    Dim ctlList As Control
    DoCmd.GoToRecord
    If Err.Number = 2105 Then
        MsgBox "Taka kategoria już jest lub pole tekstowe jest puste!", _
            vbInformation, "Biblioteka"
    Else
        Set ctlList = Forms!frmZakupy.cboKategoria
        ctlList.Requery
    End If
    DoCmd.Close
End Sub

Private Sub Form_Open(Cancel As Integer)
    If intFlaga = 2 Then
        Me.Caption = "Wprowadzanie nowej kategorii"
        Me.NavigationButtons = False
        cmdZapisz.Visible = True
    Else
        Me.Caption = "Przegląd i edycja kategorii"
        Me.NavigationButtons = True
        cmdZapisz.Visible = False
    End If
End Sub

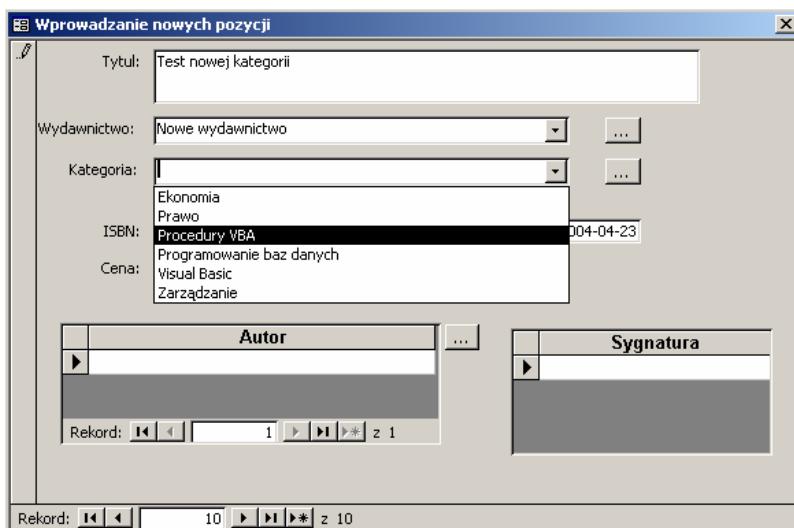
```

Na kolejnej stronie pokazany jest efekt naszej pracy: rozpoczęta rejestracja nowego zakupu, podaliśmy tytuł i wydawnictwo, nagle okazuje się, że nie ma takiej kategorii.

Klik przycisku z trzema kropkami uruchamia (bez potrzeby zamykania formularza frmZakupy) formularz frmKategorie w trybie wprowadzania nowego rekordu.



Po wpisaniu nowej kategorii (tu: Procedury VBA) i zapisaniu rekordu przyciskiem „Zapisz rekord i wróć do poprzedniego formularza” możemy kontynuować rejestrację zakupu.

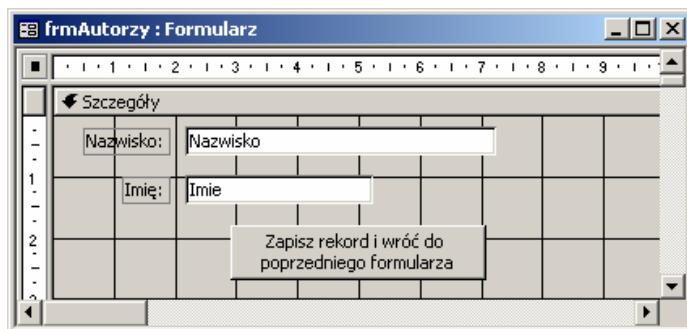


5.4.3. Modyfikacja formularza frmAutorzy

Modyfikacja formularza frmAutorzy w taki sposób, aby z poziomu formularza frmZakupy stworzyć możliwość dodania brakującego nazwiska i imienia autora rejestrowanej właśnie pozycji jest trochę bardziej skomplikowana z dwu powodów.

Po pierwsze, po dodaniu rekordu do tabeli Autorzy musi nastąpić aktualizacja pola kombi nie w głównym formularzu, lecz w podformularzu frmPodformularzAutorzy. Po drugie komplikuje się problem zapewnienia, abyśmy nie dodali takiego nazwiska i imienia, które już jest w tabeli Autorzy. Trudność spowodowana jest tym, że nie mamy możliwości indeksowania jednocześnie zawartości pól Nazwisko i Imię w tej tabeli (indeksowanie bez powtórzeń), tym samym nie będziemy mogli wykorzystać błędu Accessa w przypadku naruszenia unikalnego indeksu. Rozwiążemy ten problem poprzez programowe sprawdzenie, czy rekord o takich wartościach pól Nazwisko i Imię już jest w tabeli Autorzy, jeżeli tak, to zgłosimy stosowny komunikat, jeżeli nie, to operacja dodania nowego rekordu będzie kontynuowana.

Podobnie jak w poprzednich modyfikacjach zaczniemy od zmiany projektu formularza frmAutorzy poprzez dodanie przycisku polecenia o nazwie cmdZapisz.



Kolejny krok to napisanie dwóch procedur, pierwszej uruchamianej w momencie otwarcia tego formularza z zadaniem modyfikowania sposobu jego otwarcia w zależności od wartości zmiennej intFlaga oraz drugiej obsługującej klik przycisku cmdZapisz. Poniżej pokazana jest pierwsza z tych procedur, w zasadzie jest ona analogiczna do tych, które były pisane wcześniej w tym rozdziale.

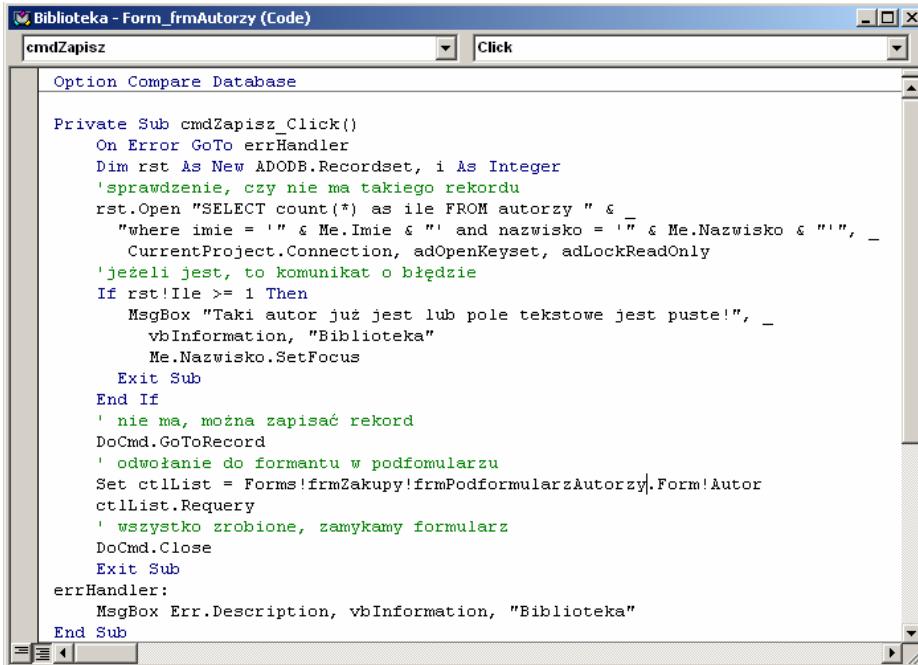
```
Private Sub Form_Open(Cancel As Integer)
    If intFlaga = 2 Then
        Me.Caption = "Wprowadzanie nowego autora"
        Me.NavigationButtons = False
        cmdZapisz.Visible = True
    Else
```

```

Me.Caption = "Przegląd i edycja wydawnictw"
Me.NavigationButtons = True
cmdZapisz.Visible = False
End If
End Sub

```

Druga z procedur jest trochę bardziej skomplikowana, co wynika z konieczności odświeżenia źródła danych dla formantu podformularza oraz programowego sprawdzenia możliwości dodania nowego rekordu. Poniżej okna edytora VBA z tą procedurą, umieszczone w niej komentarze wyjaśniają sens kolejnych instrukcji.

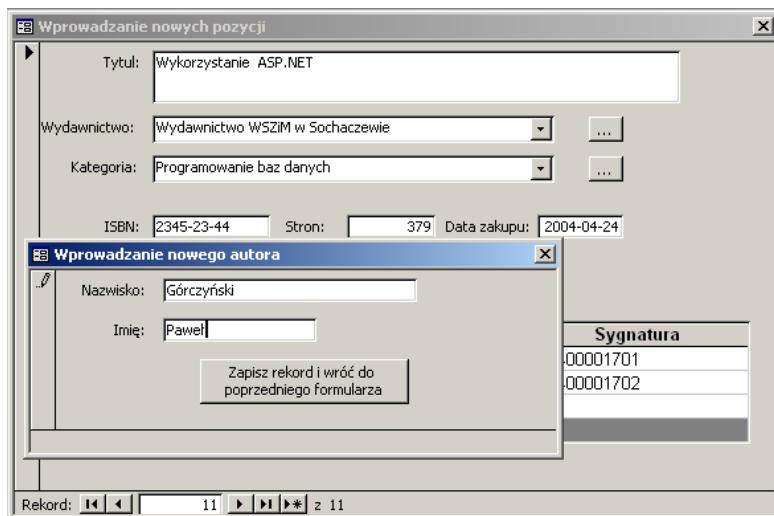


W szczególności na uwagę zasługuje sposób odwołania się do formantu Autor w podformularzu frmPodformularzAutorzy formularza frmZakupy. W instrukcji:

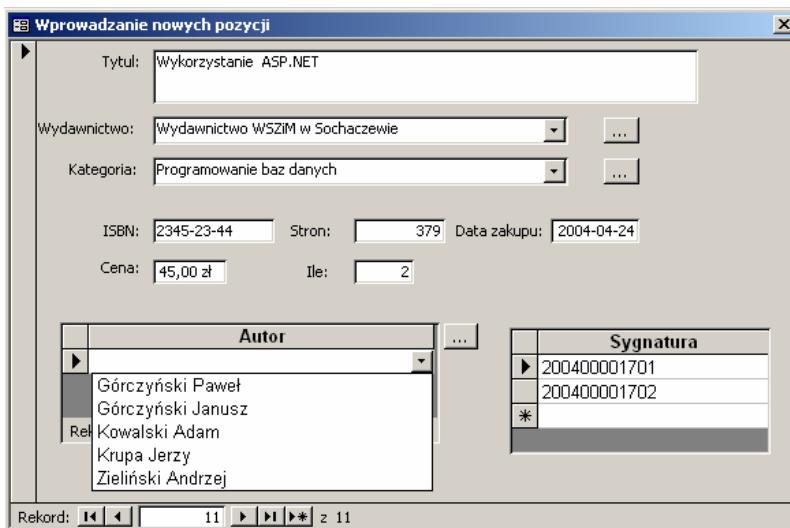
```
Set ctlList = _
Forms!frmZakupy!frmPodformularzAutorzy.Form!Autor
```

ważne są symbole wykrzyknika i kropki wykorzystane do jednoznacznego wskazania obiektu w podformularzu.

Na kolejnej stronie efekt naszej pracy, w trakcie rejestracji nowej pozycji okazało się, że brak danych autora, nastąpiło więc otwarcie podformularza frmAutorzy.



Możemy już skorzystać z przycisku „Zapisz rekord”, formularz frmAutorzy zostanie zamknięty, a w podformularzu autorów znajdziemy już dane nowego autora.



5.4.4. Modyfikacja formularza frmKlienci

Podobnie jak w przypadku formularza frmZakupy musimy tak przeprojektować formularz frmKlienci, aby była możliwość dodania nowej ulicy i jej kodu w trakcie rejestrowania danych nowego klienta. Realizacja tego zadania wymagać będzie zmian w trzech formularzach:

W frmKlienci dodamy przycisk wywołujący w trybie dodawania rekordów formularz frmUliceMiejscowosciKody

W frmUliceMiejscowosciKody dodamy przycisk polecenia cmdZapisz wraz ze stosowną procedurą weryfikującą wprowadzone dane poprzez zapytanie SQL-owe (podobnie jak w formularzu frmAutorzy). Utworzymy także procedurę modyfikującą wygląd tego formularza w zależności od wartości zmiennej intFlaga. Dodatkowo utworzymy przycisk polecenia o nazwie cmdMiejscowoscAdd otwierający formularz frmMiejscowosci w trybie dodawania nowego rekordu – na te sytuacje, gdyby w trakcie wprowadzania nowej ulicy i jej kodu okazało się, że nie ma takiej miejscowości.

W frmMiejscowosci dodamy przycisk polecenia cmdZapisz wraz ze stosowną procedurą weryfikującą wprowadzoną nazwę i odświeżającą źródło danych w polu kombi cboMiejscowosci formularza frmUliceMiejscowosciKody. Oczywiście będzie także potrzebna procedura modyfikująca wygląd tego formularza zależnie od wartości zmiennej sterującej intFlaga.

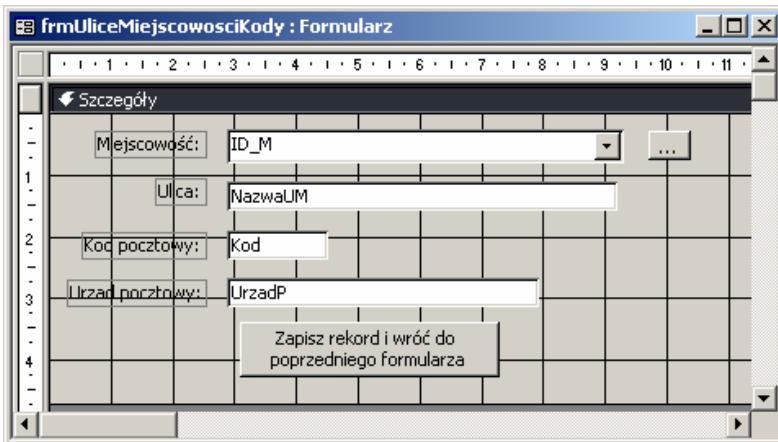
W formularzu frmKlienci potrzebny będzie jeszcze jeden przycisk polecenia otwierający formularz frmDomeny w celu dodania brakującej domeny przy rejestracji adresu mailowego klienta. Pociągnie to za sobą konieczność zmian w projekcie formularza frmDomeny, gdzie procedura obsługująca zamknięcie formularza musi być tak napisana, aby odświeżyć źródło danych w podformularzu AdresyMailowe formularza frmKlienci

Pracę nad przedstawionymi wyżej modyfikacjami zaczynamy od umieszczenia w formularzu frmKlienci dwóch przycisków polecen o nazwach cmdUlicaKodAdd oraz cmdDomenaAdd, dla obu przycisków utworzymy pokazane niżej procedury zdarzeniowe reagujące na klik tych przycisków.

```
Private Sub cmdUlicaKodAdd_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmUliceMiejscowosciKody", acNormal, ,
    , acFormAdd
End Sub

Private Sub cmdDomenaAdd_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmDomeny", acNormal, , , acFormAdd
End Sub
```

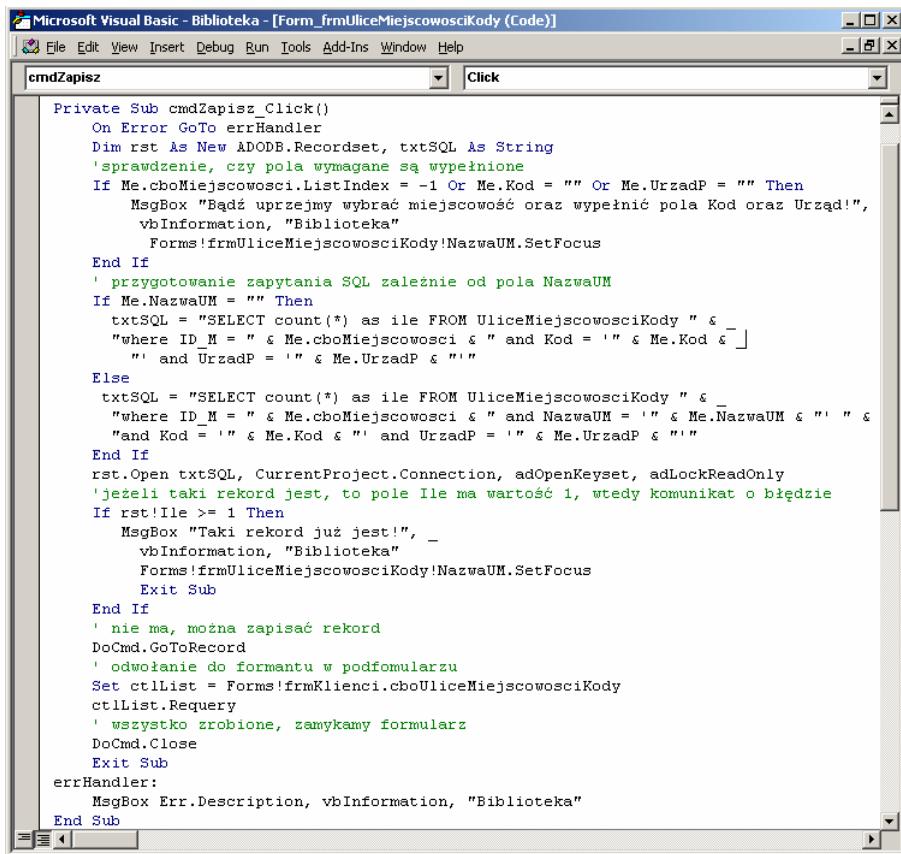
W projekcie formularza frmUliceMiejscowosciKody dodajemy dwa przyciski polecień o nazwach odpowiednio cmdZapisz oraz cmdMiejscowoscAdd.



Kolejny krok, to utworzenie trzech procedur zdarzeniowych obsługujących zdarzenie otwarcia tego formularza (modyfikacja jego wyglądu) oraz zdarzenie *Przy kliknięciu* dla obu przycisków.

```
Private Sub Form_Load()
    If intFlaga = 2 Then
        Me.Caption = "Wprowadzanie nowej ulicy i jej kodu"
        Me.NavigationButtons = False
        cmdZapisz.Visible = True
    Else
        Me.Caption = "Przegląd i edycja ulic i kodów"
        Me.NavigationButtons = True
        cmdZapisz.Visible = False
    End If
End Sub
```

Z projektu tabeli UliceMiejscowosciKody wynika, że procedura sprawdzająca unikalność rekordu musi badać, czy wśród już istniejących rekordów tej tabeli nie ma takiego, którego pola zawierają dokładnie tę samą kombinację danych, które użytkownik naszej bazy wprowadził do formularza frmUliceMiejscowosciKody. Procedura ta będzie rozbudowaną wersją podobnej procedury z formularza frmAutorzy. Dodatkowo, przed próbą zapisu rekordu będziemy jeszcze sprawdzać, czy pola wymagane w tabeli UliceMiejscowosciKody zostały wypełnione w formularzu (dotyczy to wybrania miejscowości w polu kombi cboMiejscowosci oraz wprowadzenia kodu i nazwy urzędu pocztowego). Poniżej widok tej procedury wraz z komentarzami.



The screenshot shows the Microsoft Visual Basic IDE interface with the title bar "Microsoft Visual Basic - Biblioteka - [Form_frmUliceMiejscowosciKody (Code)]". The menu bar includes File, Edit, View, Insert, Debug, Run, Tools, Add-Ins, Window, and Help. The toolbar has icons for New, Open, Save, Print, Find, Replace, Copy, Paste, Cut, Delete, Undo, Redo, and Properties. The code editor window contains the following VB code:

```

Private Sub cmdZapisz_Click()
    On Error GoTo errHandler
    Dim rst As New ADODB.Recordset, txtSQL As String
    'sprawdzenie, czy pola wymagane są wypełnione
    If Me.cboMiejscowosci.ListIndex = -1 Or Me.Kod = "" Or Me.UrzadP = "" Then
        MsgBox "Bądź uprzejmy wybrać miejscowości oraz wypełnić pola Kod oraz Urząd!", vbInformation, "Biblioteka"
        Forms!frmUliceMiejscowosciKody!NazwaUM.SetFocus
    End If
    ' przygotowanie zapytania SQL zależne od pola NazwaUM
    If Me.NazwaUM = "" Then
        txtSQL = "SELECT count(*) as ile FROM UliceMiejscowosciKody " &
        "where ID_M = " & Me.cboMiejscowosci & " and Kod = '" & Me.Kod & "' " &
        "and UrzadP = '" & Me.UrzadP & "'"
    Else
        txtSQL = "SELECT count(*) as ile FROM UliceMiejscowosciKody " &
        "where ID_M = " & Me.cboMiejscowosci & " and NazwaUM = '" & Me.NazwaUM & "' " &
        "and Kod = '" & Me.Kod & "' and UrzadP = '" & Me.UrzadP & "'"
    End If
    rst.Open txtSQL, CurrentProject.Connection, adOpenKeyset, adLockReadOnly
    'jeżeli taki rekord jest, to pole Ile ma wartość 1, wtedy komunikat o błędzie
    If rst!Ile >= 1 Then
        MsgBox "Taki rekord już jest!", _
        vbInformation, "Biblioteka"
        Forms!frmUliceMiejscowosciKody!NazwaUM.SetFocus
        Exit Sub
    End If
    ' nie ma, można zapisać rekord
    DoCmd.GoToRecord
    ' odwołanie do formantu w podformularzu
    Set ctlList = Forms!frmKlienci.cboUliceMiejscowosciKody
    ctlList.Requery
    ' wszystko zrobione, zamykamy formularz
    DoCmd.Close
    Exit Sub
errHandler:
    MsgBox Err.Description, vbInformation, "Biblioteka"
End Sub

```

Pozostaje jeszcze przygotowanie procedury dla obsługi zdarzenia *Przy kliknięciu* przycisku cmdMiejscowosciAdd:

```

Private Sub cmdMiejscowoscAdd_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmMiejscowosci", acNormal, , , acFormAdd
End Sub

```

W formularzu frmMiejscowosci dodajemy przycisk polecenia o nazwie cmdZapisz oraz tworzymy dwie procedury zdarzeniowe, jedną uruchamianą przy otwarciu formularza i drugą wykonywaną na zdarzenie *Przy kliknięciu*. Pierwsza z nich steruje sposobem otwarcia formularza zależnie od wartości zmiennej intFlaga, zadanym drugiej jest sprawdzenie, czy można zapisać rekord oraz odświeżenie źródła danych formantu cboMiejscowosci formularza frmUliceMiejscowosciKody.

```
Private Sub Form_Open(Cancel As Integer)
    If intFlaga = 2 Then
        Me.Caption = "Wprowadzanie nowej miejscowości"
        Me.NavigationButtons = False
        cmdZapisz.Visible = True
    Else
        Me.Caption = "Przegląd i edycja miejscowości"
        Me.NavigationButtons = True
        cmdZapisz.Visible = False
    End If
End Sub

Private Sub cmdZapisz_Click()
    On Error Resume Next
    Dim ctlList As Control
    DoCmd.GoToRecord
    If Err.Number = 2105 Then
        MsgBox "Taka miejscowość już jest lub pole tekstowe
                jest puste!", _
                vbInformation, "Biblioteka"
    Else
        Set ctlList =
            Forms!frmUliceMiejscowosciKody.cboMiejscowosci
        ctlList.Requery
    End If
    DoCmd.Close
End Sub
```

Pozostała nam jeszcze modyfikacja formularza frmDomeny, gdzie dodamy znany już przycisk cmdZapisz i napiszemy dwie procedury pokazane poniżej.

```
Private Sub Form_Open(Cancel As Integer)
    If intFlaga = 2 Then
        Me.Caption = "Wprowadzanie nowej domeny"
        Me.NavigationButtons = False
        cmdZapisz.Visible = True
    Else
        Me.Caption = "Przegląd i edycja domen"
        Me.NavigationButtons = True
        cmdZapisz.Visible = False
    End If
End Sub
```

```

Private Sub cmdZapisz_Click()
    On Error Resume Next
    Dim ctlList As Control
    DoCmd.GoToRecord
    If Err.Number = 2105 Then
        MsgBox "Taka domena już jest lub pole tekstowe jest
                puste!", -
                vbInformation, "Biblioteka"
    Else
        Set ctlList =
            Forms!frmKlienci!AdresyMailowe.Form! [adres domenowy]
        ctlList.Requery
    End If
    DoCmd.Close
End Sub

```

Pokazana wyżej procedura `cmdZapisz_Click()` musi odświeżyć źródło danych pola kombi w podfomularzu o nazwie `AdresyMailowe` formularza `frmKlienci`, stąd rozbudowana referencja w przypisaniu:

```

Set ctlList =
    Forms!frmKlienci!AdresyMailowe.Form! [adres domenowy]

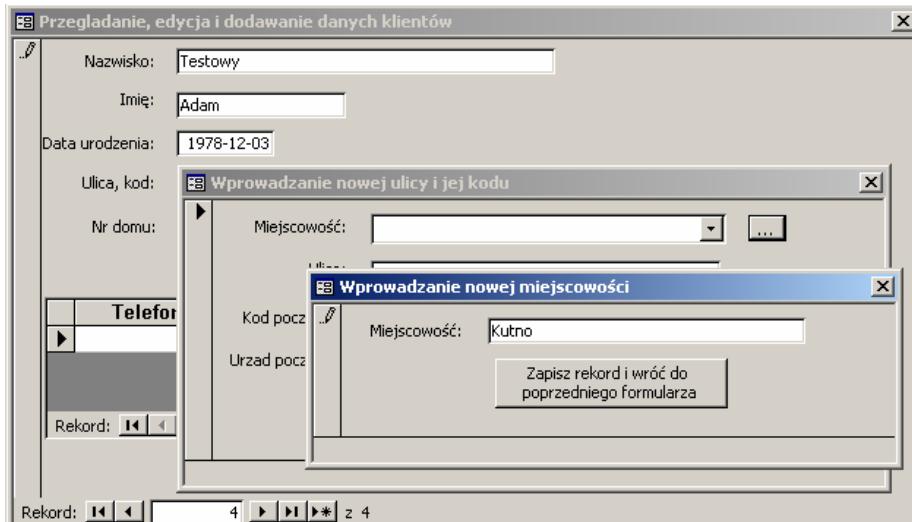
```

Nazwa pola kombi w podfomularzu zawiera spację, co wymusza zamknięcie tej nazwy w nawiasach kwadratowych.

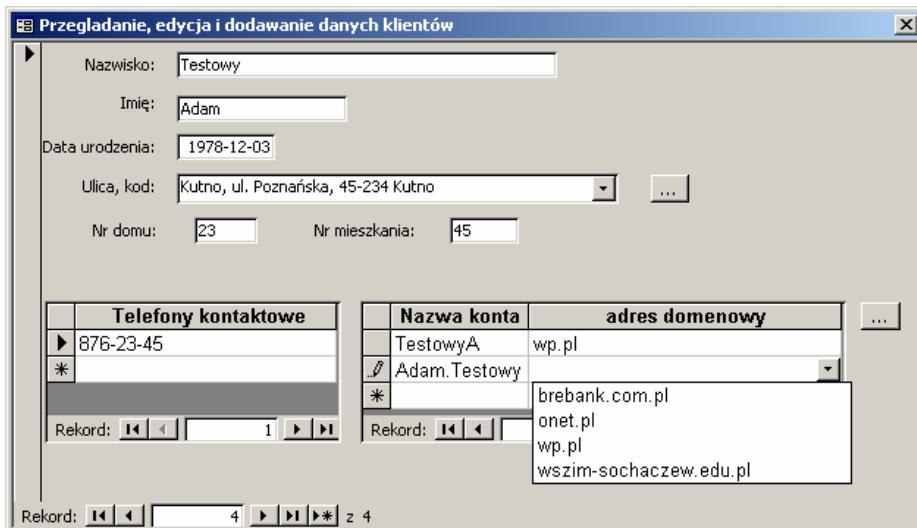
Dla pokazania funkcjonowania tak poprawionych formularzy (`frmKlienci`, `frmUliceMiejscowosciKody`, `frmMiejscowosci` i `frmDomeny`) założymy, że chcemy zarejestrować w naszej bazie nowego klienta o danych:

Testowy Adam (nazwisko i imię)
 1978-12-03 (data urodzenia)
 Kutno, ul. Poznańska 23 m. 45, 45-234 Kutno (adres zamieszkania)
 876-23-45 (telefon)
 Testowy.A@wp.pl (adres mailowy)
 Adam.Testowy@poczta.kutno.pl (drugi adres mailowy)

W trakcie wprowadzania tych danych do formularza `frmKlienci` okazuje się, że w polu kombi „Ulica, kod” nie mamy ulicy Poznańskiej w Kutnie ani jej kodu i urzędu pocztowego. Przyciskiem otwieramy formularz `frmUliceMiejscowosciKody`, ale w polu kombi „Miejscowość” nie znajdujemy Kutna, otwieramy więc kolejny formularz `frmMiejscowosci`. Sytuacja taka pokazana jest poniżej.



Kolejno wpisujemy potrzebne informacje i po powrocie do frmKlienci kontynuujemy wprowadzanie reszty danych tego nowego klienta.

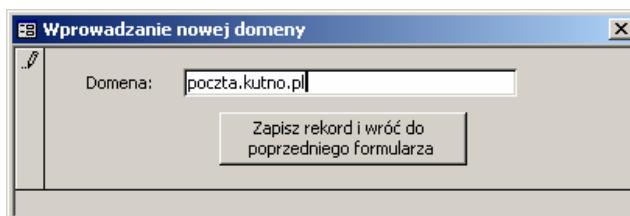


Domeny „poczta.kutno.pl” nie znajdujemy w rozwiniętej liście adresów domenowych, co pociąga za sobą konieczność dodania tej brakującej domeny. Klik przycisku [...] dodania nowej domeny spowoduje jednak komunikat MS Access o niemożności zapisania

rozczętego rekordu (zobacz symbol ołówka na pasku selektora pól) z powodu nieokreślenia ID_D (identyfikatora domeny).



Jedynym wyjściem w tej sytuacji jest zaakceptowanie tego komunikatu, a następnie klawiszem Esc rezygnujemy z wprowadzenia adresu mailowego, co umożliwia otwarcie formularza frmDomeny.



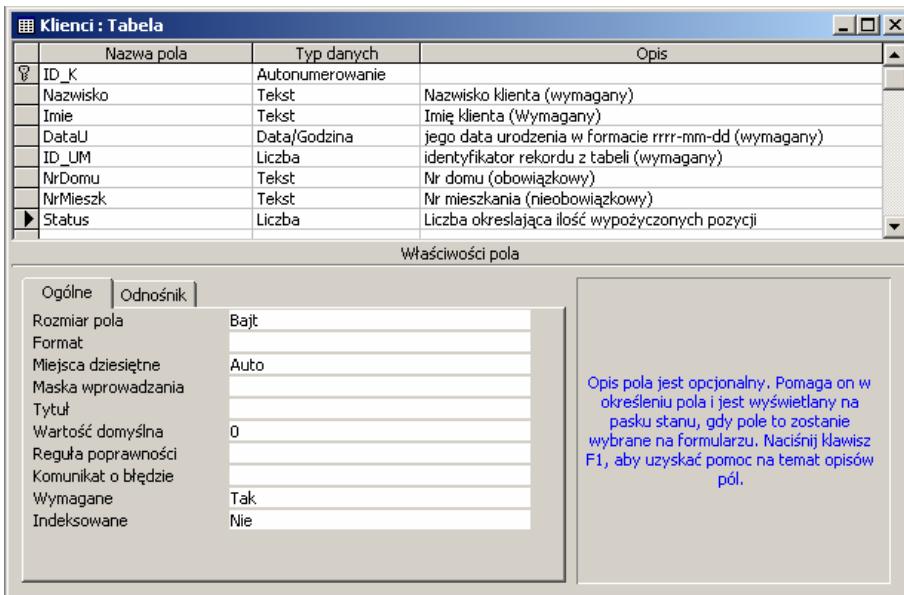
Po wpisaniu brakującej domeny i akceptacji przycisku „Zapisz rekord ...” wracamy do formularza frmKlienci i bez problemu możemy już dopisać brakujący adres mailowy nowego klienta naszej biblioteki.

5.5. Obsługa wypożyczeń i zwrotów książek

Dla pełnej obsługi procesu wypożyczania książki zarejestrowanemu użytkownikowi naszej biblioteki, a następnie dla rejestracji jej zwrotu będziemy musieli przygotować szereg nowych obiektów, w tym dwa główne formularze, kwerendy wybierające, aktualizujące i parametryczne, także zapytania SQL-owe.

5.5.1. Formularz frmWypozyzenia

Ten etap pracy zaczniemy od małej modyfikacji tabeli `Klienci`, gdzie dodamy pole o nazwie `Status`. Zadaniem tego pola będzie przechowywanie informacji o liczbie aktualnie wypożyczonych pozycji.

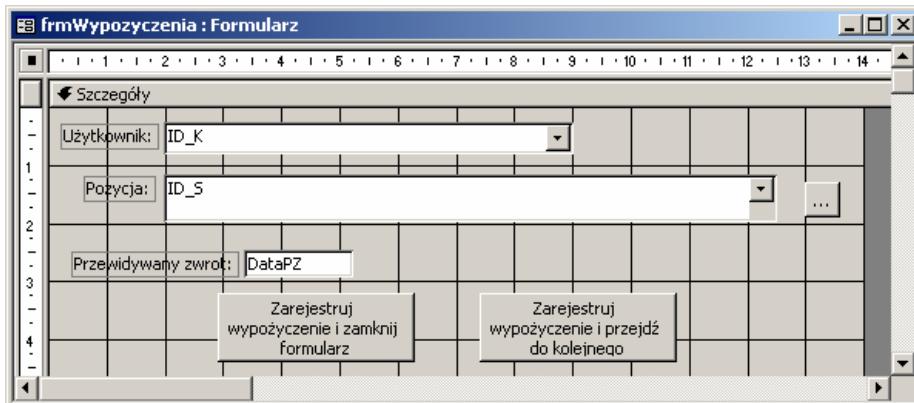


W momencie rejestracji nowej wypożyczonej pozycji wartość tego pola musi się zwiększyć o jeden, podobnie przy rejestracji zwrotu wartość tego pola będzie zmniejszana o jeden. Wartość pola `Status` będzie także determinować możliwość wypożyczenia kolejnej pozycji, dla potrzeb tej książki ustalimy, że jeden użytkownik może maksymalnie wypożyczyć 4 pozycje. Przyjmiemy również jako zasadę, że kolejna pozycja może być wypożyczona jedynie wtedy, gdy dany użytkownik nie ma pozycji z przekroczonym terminem zwrotu.

Zwiększanie lub zmniejszanie wartości pola **Status** dla danego klienta zrealizujemy poprzez parametryczną kwerendę aktualizującą wywoływaną w momencie rejestracji wypożyczenia lub zwrotu książki.

Prace zaczniemy od zaprojektowania formularza **frmWypozyczenia**, formularz ten będzie korzystał z tabeli **Wypożyczenia** jako źródła danych. Na powierzchni formularza osadzimy dwa pola kombi o nazwach **cboKlient** i **cboSygnatura**, pierwsze z nich będziemy wykorzystywać do ustalenia wartości **ID_K** (identyfikatora) osoby wypożyczającej, a drugie do uzyskania wartości **ID_S** (identyfikatora sygnatury) wypożyczanej pozycji.

W formularzu umieścimy także trzy przyciski poleceń o nazwach odpowiednio **cmdPokazSzczegoly**, **cmdZarejestrujAndClose** oraz **cmdZarejestrujAndNext**. Zadaniem pierwszego z nich będzie pokazanie szczegółowych informacji o wybranej pozycji książkowej, zadaniem dwóch pozostałych będzie zarejestrowanie faktu wypożyczenia książki (z zamknięciem lub nie formularza rejestracji).

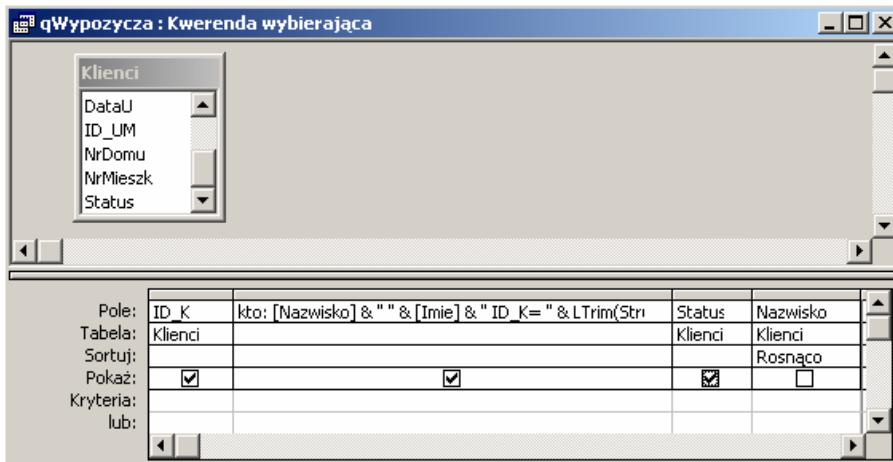


Z uwagi na fakt, że formularz **frmWypozyczenia** będziemy wykorzystywać **wyłącznie** do operacji zarejestrowania wypożyczenia oraz z uwagi na konieczność wykonania dwóch kwerend aktualizujących zmienimy trzy ważne właściwości formularza.

Właściwość *Wprowadzanie danych* ustawimy na **Tak**, tym samym przy każdym otwarciu formularza wskaźnik rekordów będzie ustawiony na nowy, pusty rekord. Właściwość *Przyciski nawigacyjne* ustawimy na **Nie**, co uniemożliwi rejestrację rekordu za pomocą kliku przycisku przejścia do kolejnego rekordu. Właściwość *Cykliczny* ustawimy na **Bieżący rekord**, co nie pozwoli na przejście do nowego rekordu za pomocą klawisza tabulacji. Dwie ostatnie właściwości wymuszają zapisanie rekordu jedynie poprzez klik jednego z dwóch przycisków umieszczonych na dole formularza.

Kolejny krok to przygotowanie kwerend, które mają dostarczać danych do obu pól kombi. Pole `cboKlient` powinno zwrócić `ID_K`, a w liście powinniśmy mieć nazwisko i imię osoby wypożyczającej, jej numer identyfikacyjny oraz informację o wartości pola `Status` po to choćby, aby poinformować zainteresowanego, że nie może pożyczyć kolejnej książki z uwagi na wypełniony limit wypożyczeń. Praktycznie nie ma łatwej możliwości umieszczenia informacji o ewentualnym niezwróceniu wcześniej wypożyczonej pozycji, rozwiązujemy ten problem później na drodze odpowiedniej procedury zdarzeniowej. Poniżej pokazany jest widok projektu kwerendy `qWypozycza`, warto zwrócić uwagę na pole wyliczane o nazwie `kto`, jego zadaniem jest zwrócenie „zbitki” informacji z kilku pól.

```
kto: [Nazwisko] & " " & [Imie] & " ID_K= " &
      LTrim(Str([ID_K])) & ", Status=" & [Status]
```

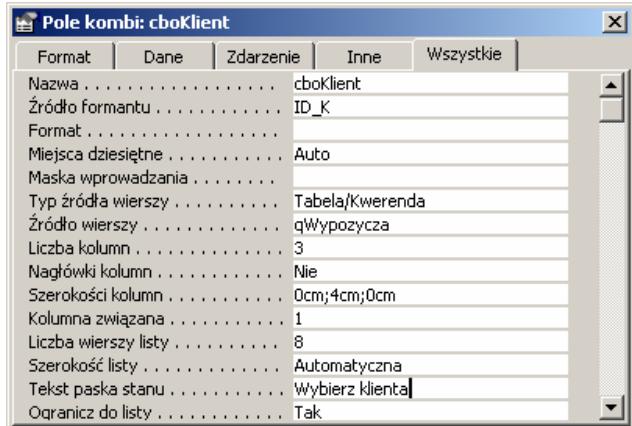


Pole wyliczane `kto` zwraca informację typu tekstowego, stąd wykorzystanie funkcji `Str` do zamiany wartości numerycznej na tekstową oraz funkcji `LTrim` do usunięcia spacji z lewej strony tekstu (która standardowo zostawia funkcja `Str` na ujemny znak liczby).

Pole `Status` jest zwracane niejako podwójnie, raz jako element pola `kto`, drugi raz samodzielnie. W liście pola kombi `cboKlient` pole `Status` (samodzielnie) nie będzie wykorzystane, a potrzebne nam jest z tego powodu, abyśmy mieli możliwość zgłoszenia komunikatu w sytuacji, gdy liczba wypożyczeń przekroczy ustalony limit.

Pole `Nazwisko` wykorzystujemy w kwerendzie `qWypozycza` jedynie do potrzeb posortowania zwracanych rekordów, co pozwoli na przyspieszone wyszukiwanie danego klienta wg pierwszych liter nazwiska.

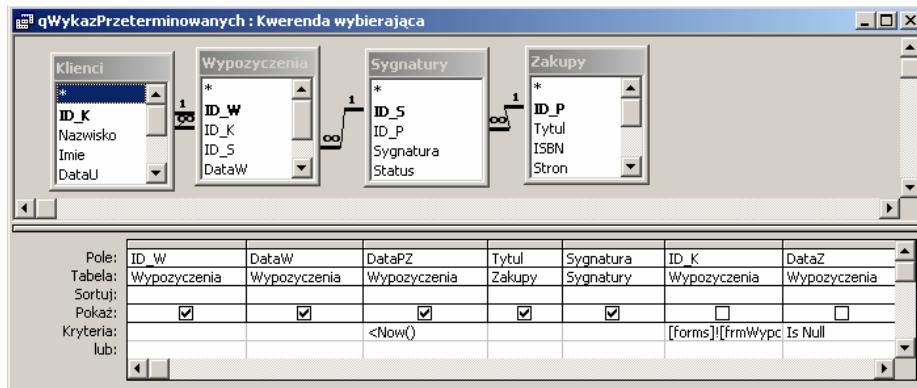
Po zapisaniu pokazanej kwerendy w bazie danych można już ustawić najważniejsze właściwości pola kombi `cboKlient`. W pokazanym oknie właściwości warto zwrócić uwagę na liczbę kolumn oraz szerokości kolumn; ostatnie zero oznacza, że pole Status nie jest pokazywane.



Przygotujemy teraz dwie kwerendy parametryczne, będą one uruchamiane wtedy, gdy wypożyczenie kolejnej książki nie będzie możliwe z dwu powodów:

1. Niedotrzymania terminu zwrotu jednej z wypożyczonych wcześniej pozycji;
2. Przekroczenia ustalonego limitu wypożyczeń.

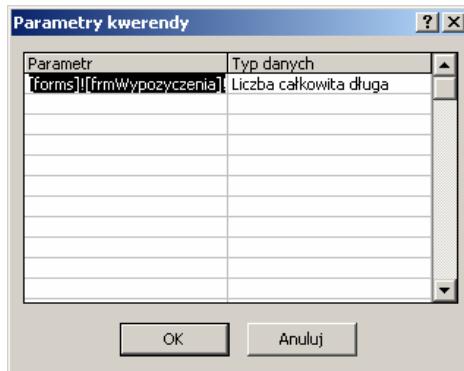
Obie kwerendy będą uruchamiane przez procedurę zdarzeniową uruchamianą jako efekt zdarzenia *Po aktualizacji* pola `cboKlient`. Poniżej widok projektu kwerendy zwracającej dla danego klienta przeterminowane pozycje.



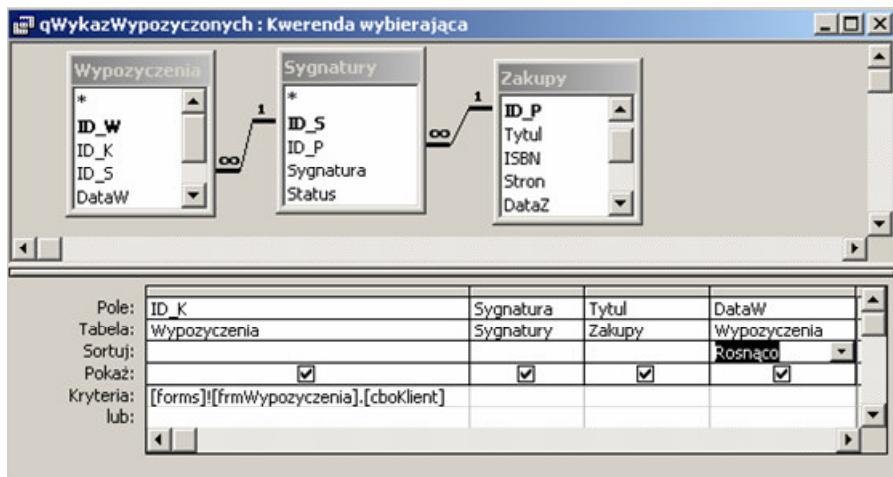
Pole DataZ ma ustawione kryterium Is Null, pole DataPZ ma zdefiniowany warunek <Now(), a pole ID_K odwołuje się w kryterium do wartości zwracanej przez obiekt cboKlient formularza frmWypożyczczenia:

```
[forms]![frmWypożyczczenia]![cboKlient]
```

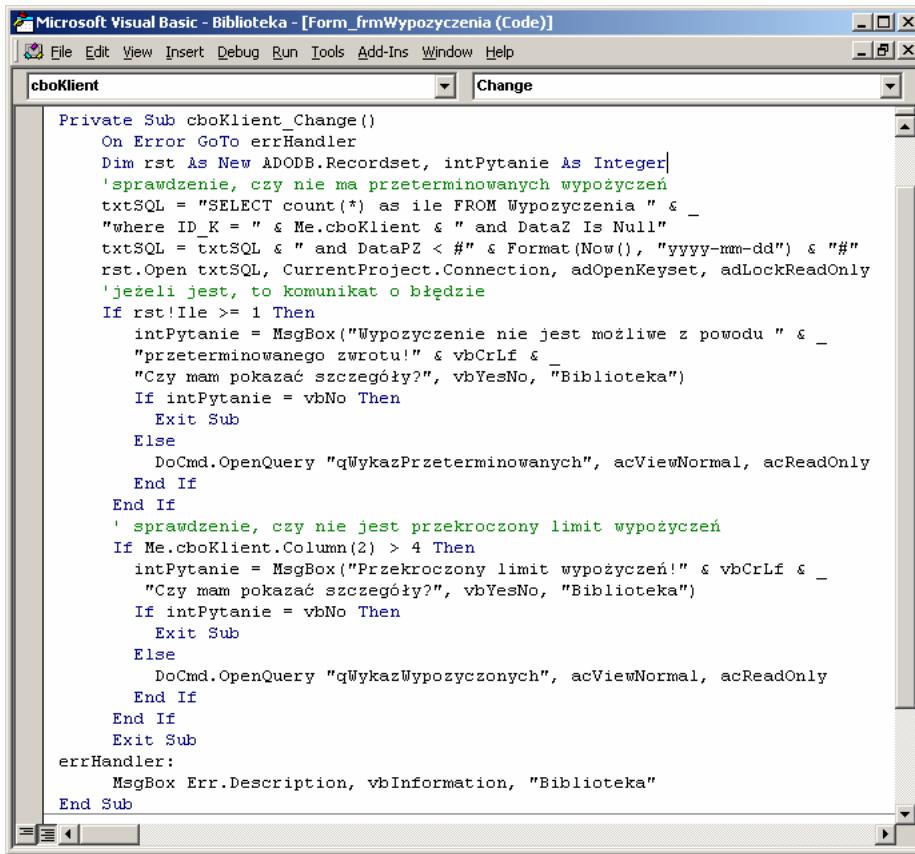
Dla uniknięcia niejednoznaczności w oknie *Parametry kwerendy* w kolumnie *Parametry* wpiszemy pokazany wyżej parametr (najlepiej wkleić), a w kolumnie *Typ danych* wybierzemy właściwy typ, czyli liczbę całkowitą długą.



W bardzo podobny sposób przygotujemy kolejną kwerendę parametryczną pokazującą aktualnie wypożyczone pozycje dla danego klienta.



Mając gotowe obie kwerendy możemy utworzyć procedurę wykonywaną po aktualizacji pola cboKlient. Poniżej pokazane jest okno projektu tej procedury.



```

Microsoft Visual Basic - Biblioteka - [Form_frmWypozyzenia (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
cboKlient Change
Private Sub cboKlient_Change()
    On Error GoTo errHandler
    Dim rst As New ADODB.Recordset, intPytanie As Integer
    ' sprawdzenie, czy nie ma przeterminowanych wypożyczeń
    txtSQL = "SELECT count(*) as ile FROM Wypozyzenia " & _
    "where ID_K = " & Me.cboKlient & " and DataZ Is Null"
    txtSQL = txtSQL & " and DataPZ < #" & Format(Now(), "yyyy-mm-dd") & "#"
    rst.Open txtSQL, CurrentProject.Connection, adOpenKeyset, adLockReadOnly
    'jeżeli jest, to komunikat o błędzie
    If rst!Ile >= 1 Then
        intPytanie = MsgBox("Wypożyczenie nie jest możliwe z powodu " & _
        "przeterminowanego zwrotu!" & vbCrLf & _
        "Czy mam pokazać szczegółów?", vbYesNo, "Biblioteka")
        If intPytanie = vbNo Then
            Exit Sub
        Else
            DoCmd.OpenQuery "qWykazPrzeterminowanych", acViewNormal, acReadOnly
        End If
    End If
    ' sprawdzenie, czy nie jest przekroczony limit wypożyczeń
    If Me.cboKlient.Column(2) > 4 Then
        intPytanie = MsgBox("Przekroczony limit wypożyczeń!" & vbCrLf & _
        "Czy mam pokazać szczegółów?", vbYesNo, "Biblioteka")
        If intPytanie = vbNo Then
            Exit Sub
        Else
            DoCmd.OpenQuery "qWykazWypożyczonych", acViewNormal, acReadOnly
        End If
    End If
    Exit Sub
errHandler:
    MsgBox Err.Description, vbInformation, "Biblioteka"
End Sub

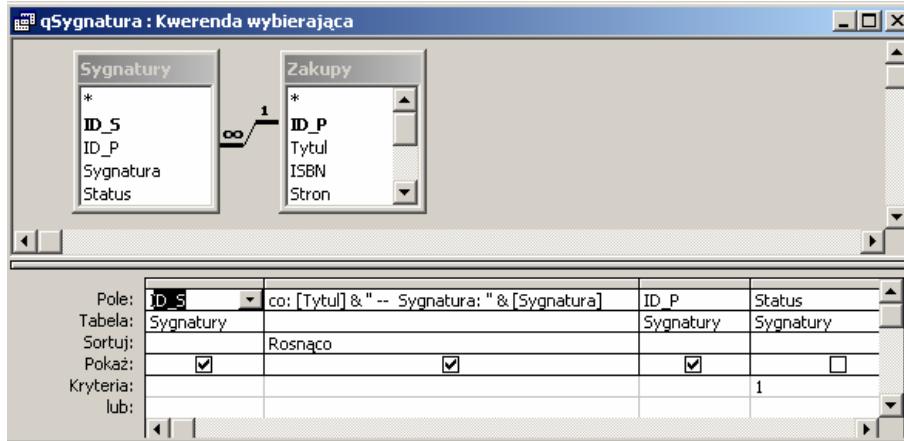
```

Funkcja MsgBox została wykorzystana do odebrania od użytkownika bazy decyzji co do pokazywania lub nie szczegółów odmowy wypożyczenia kolejnej książki, a stała vbCrLf została wykorzystana do „złamania” wiersza komunikatu.

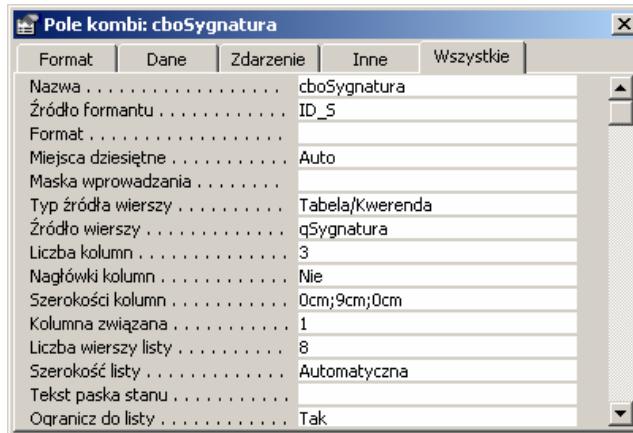
Przejdziemy teraz do przygotowania kwerendy dostarczającej dane do pola kombi cboSygnatury. Kwerenda ta musi pokazać tylko te pozycje, dla których pole Status w tabeli Sygnatury ma wartość jeden (jak pamiętamy wartość dwa ma oznaczać pozycję w czytaniu, a wartość zero pozycję wycofaną).

Kwerenda o nazwie qSygnatury oparta będzie o tabele Sygnatury oraz Zakupy, a zwracanymi polami będą ID_S (identyfikator sygnatury), pole wyliczane co

zwracające zbitkę informacji z kilku pól oraz pole `ID_P` (identyfikator pozycji), które wykorzystamy do pokazania szczegółów wybranej pozycji książki. Pole `Status` zostało wykorzystane jedynie do ograniczenia rekordów do tych pozycji, które aktualnie są dostępne do wypożyczenia.



Po utworzeniu pokazanej wyżej kwerendy ustawiamy odpowiednio właściwości pola `cboSygnatura`. Proszę zwrócić uwagę na właściwości *Liczba kolumn* oraz *Szerokości kolumn*. Wartość zero dla pierwszej i trzeciej kolumny oznacza, że w liście pola kombi znajdują się tylko wartości zwarcane przez pole wyliczane `co`.



Z uwagi na ograniczoną szerokość pola kombi `cboSygnatura` może zajść potrzeba uzyskania dodatkowych informacji o wybranej pozycji (wydawnictwo, autorzy,

liczba stron, ISBN itd.). Dla realizacji tego zadania wykorzystamy wartość pola `ID_P` zwracaną niejawnie przez pole kombi `cboSygnatura` oraz procedurę uruchamianą przyciskiem `cmdPokazSzczegoly`.

Pokazana niżej procedura wywołuje znany formularz `frmZakupy`, ale z warunkiem filtrującym nałożonym na pole `ID_P`. Konstrukcja:

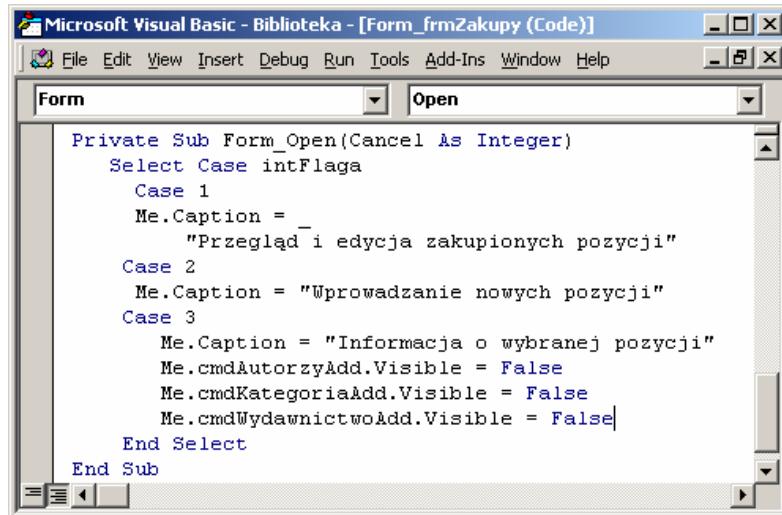
```
Me.cboSygnatura.Column(2)
```

odwołuje się do wartości drugiej kolumny (numeracja kolumn od zera) w wybranym wierszu pola kombi `cboSygnatura`, inaczej mówiąc do wartości `ID_P` wybranej książki. Zmienna `intFlaga` zostanie wykorzystana do modyfikacji tytułu formularza.

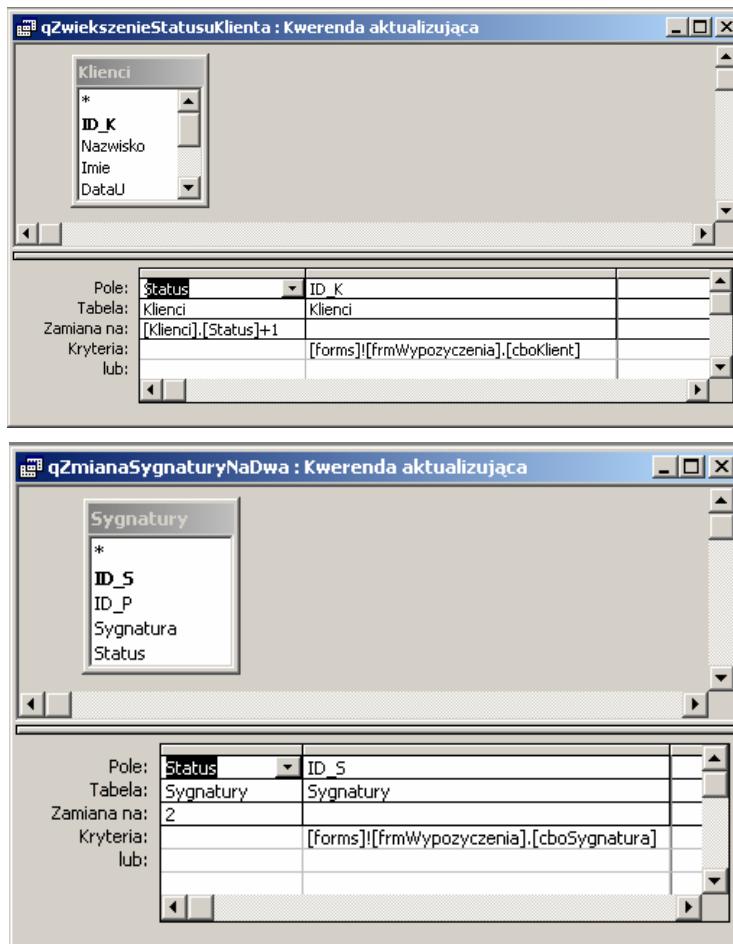
```
Private Sub cmdPokazSzczegoly_Click()
    IntFlaga=3
    DoCmd.OpenForm "frmZakupy", acNormal, , ,
        "ID_P = " & Me.cboSygnatura.Column(2), acFormReadOnly
End Sub
```

Elegancja programowania wymusza niewielką modyfikację procedury uruchamianej w momencie otwierania formularza `frmZakupy`, gdzie dotychczasowy warunek `If` zastąpimy konstrukcją `Select Case .. End Select`.

Przy okazji w sytuacji, gdy zmienna `intFlaga` będzie równa 3, to ukryjemy przyciski wykorzystywane do dodania nowego wydawnictwa, kategorii czy danych autora.



Pozostało nam przygotowanie dwu kwerend parametrycznych aktualizujących pola Status w tabelach Klienci oraz Sygnatury. Kwerenda pracująca na tabeli Klienci musi dynamicznie zwiększać wartość pola Status o jeden, z kolei kwerenda pracująca na tabeli Sygnatury musi nadać polu Statut wartość 2. Poniżej projekty obu kwerend, z uwagi na warunek parametryczny musimy także zdefiniować parametry kwerendy.



Możemy już utworzyć procedury uruchamiane przyciskami poleceń umieszczonych w dolnej części formularza. Pierwszym zadaniem tych procedur jest wywołanie obu kwerend aktualizujących, a następnie zapisanie rekordu w tabeli Wypożyczenia.

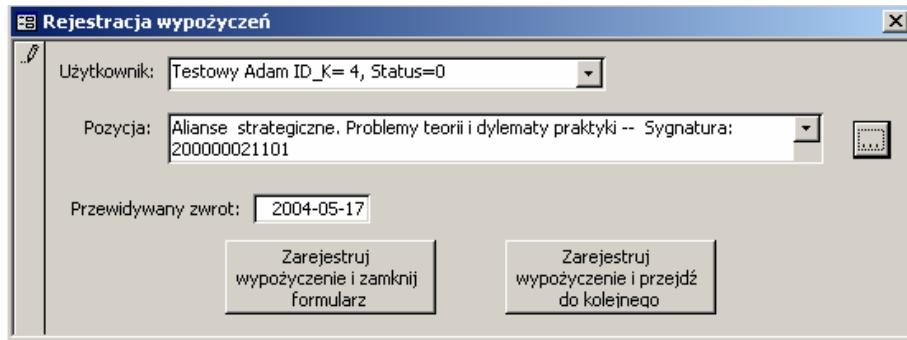
```

Private Sub cmdZarejestrujAndClose_Click()
    DoCmd.OpenQuery "qZwiekszenieStatusuKlienta"
    DoCmd.OpenQuery "qZmianaSygnaturyNaDwa"
    DoCmd.GoToRecord
    DoCmd.Close
End Sub

Private Sub cmdZarejestrujAndNext_Click()
    DoCmd.OpenQuery "qZwiekszenieStatusuKlienta"
    DoCmd.OpenQuery "qZmianaSygnaturyNaDwa"
    DoCmd.GoToRecord
End Sub

```

Możemy już zobaczyć efekt naszej pracy. Pan Testowy Adam chce wypożyczyć pozycję „Alianse strategiczne. Problemy teorii i dylematy praktyki”. W otwartym formularzu frmWypożyczenia w polu kombi cboKlient mamy już wybranego pana Adama, widzimy jego numer identyfikacyjny, mamy także informację o tym, że aktualnie nie ma żadnej książki wypożyczonej w naszej bibliotece.



W polu kombi cboSygnatura mamy już znalezioną pozycję „Alianse strategiczne...” o sygnaturze 200000021101, przewidywana data zwrotu jest już wpisana (efekt zdefiniowania wartości domyślnej w tabeli Wypożyczenia), jeżeli nie interesują nas szczegółowe informacje o wybranej pozycji, to po prostu klikamy jeden z przycisków służących do zarejestrowania faktu wypożyczenia.

Jeżeli pan Adam nie jest pewien, czy to jest dokładnie ta pozycja, którą powinien wypożyczyć i chciałby znać takie dane jak nazwę wydawnictwa, numer ISBN, liczbę stron i dane autorów, to osoba obsługująca bibliotekę po prostu wykona klik przycisku szczegółów, co spowoduje otwarcie formularza frmZakupy z filtrem ustawionym na ID_P wybranej w polu cboSygnatura pozycji.

Informacja o wybranej pozycji

Tytuł:	Alianse strategiczne. Problemy teorii i dylematy praktyki																		
Wydawnictwo:	Profesjonalna Szkoła Biznesu w Krakowie																		
Kategoria:	Zarządzanie																		
ISBN:	83-7230-035-6	Stron:	220																
Cena:	20,70 zł	Ile:	1																
Data zakupu: 2000-06-13																			
<table border="1"> <thead> <tr> <th colspan="2">Autor</th> <th colspan="2">Sygnatura</th> </tr> </thead> <tbody> <tr> <td colspan="2">Sroka Włodzimierz</td> <td colspan="2">200000021101</td> </tr> <tr> <td colspan="2">Chwistecka-Dudek Halina</td> <td colspan="2"></td> </tr> <tr> <td>Rekord:</td> <td>< < 1 > > >* z 2</td> <td colspan="2"></td> </tr> </tbody> </table>				Autor		Sygnatura		Sroka Włodzimierz		200000021101		Chwistecka-Dudek Halina				Rekord:	< < 1 > > >* z 2		
Autor		Sygnatura																	
Sroka Włodzimierz		200000021101																	
Chwistecka-Dudek Halina																			
Rekord:	< < 1 > > >* z 2																		
Rekord: < < 1 > > >* z 1 (Filtr)																			

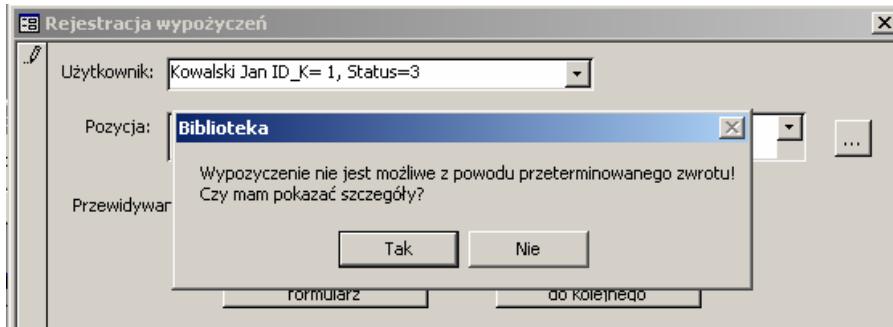
Po upewnieniu się, że jest to dokładnie ta pozycja i powrocie do formularza frmWypożyczenia można zarejestrować wypożyczenie. W efekcie nastąpi dodanie rekordu do tabeli Wypożyczenia, pole Status w tabeli Klienci w wierszu zawierającym dane pana Adama zostanie powiększone o jeden, a pole Status pozycji o sygnaturze 200000021101 w tabeli Sygnatury zostało zmienione z 1 na 2.

Wypożyczenia : Tabela						
ID_W	ID_K	ID_S	Data wypożyc	Przewidywan	DataZ	
24	3	2159	2004-05-03	2004-05-17	2004-05-17	
25	4	473	2004-05-03	2004-05-17	2004-05-17	
Rekord: < < 10 > > >* z 10						

Klienci : Tabela							
ID_K	Nazwisko	Imię	Data urodzeni	ID UM	NrDomu	NrMieszk	Status
3 Test	Zenon		1987-02-12	1 23			1
4 Testowy	Adam		1978-12-03	8 23	45		1
*	(nerozwieranie)						0
Rekord: < < 4 > > >* z 4							

Sygnatury : Tabela				
ID_S	ID_P	Sygnatura	Status	
472	210	200000021002	1	
473	211	200000021101	2	
474	212	200000021201	1	
Rekord: < < 473 > > >* z 2203				

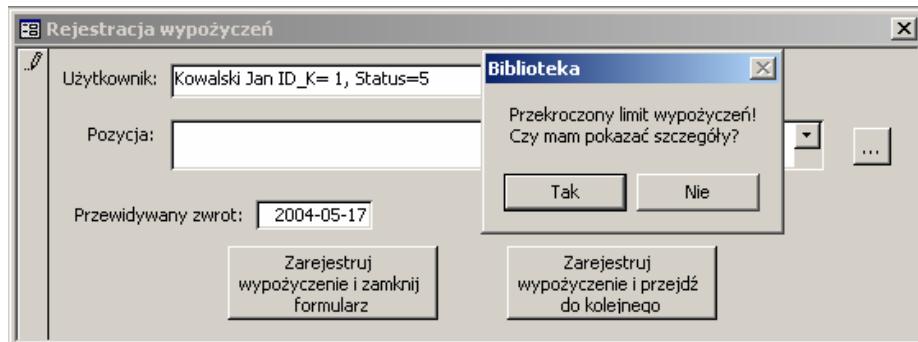
W sytuacji, gdy kolejną książkę chce wypożyczyć osoba, która zalega ze zwrotem wcześniej wypożyczonej pozycji procedura obsługująca zdarzenie Po aktualizacji pola kombi cboKlient zgłosi stosowny komunikat.



Klik przycisku Tak w oknie komunikatu otwiera kwerendę pokazującą szczegóły problemu.

WykazPrzeterminowanych : Kwerenda wybierająca				
ID_W	Data wypożyczenia	Przewidywany zwrot	Tytuł	Sygnatura
3	2004-04-02	2004-04-16	Savoir-vivre dzisiaj	200300040901
Rekord: [Navigation Buttons] 1 [Next] [Last] z 1				

Inną przyczyną odmowy wypożyczenia kolejnej pozycji może być przekroczony limit wypożyczeń. Podobnie jak w poprzednim przypadku klik przycisku Tak w oknie komunikatu otwiera kwerendę ze szczegółami wypożyczeń.



qWykazWypożyczonych : Kwerenda wybierająca			
ID_K	Sygnatura	Tytuł	Data wypożyczeń
▶ 1	200300040901	Savoir-vivre dzisiaj	2004-04-02
	1 200300004301	Motywowanie pracowników na przykładzie Mazocza	2004-04-03
	1 200000019603	Zachowania w organizacji	2004-05-03
	1 200300003101	Gospodarka budżetowa państwa	2004-05-03
	1 200300002501	Kontrola w zarządzaniu ochroną środowiska	2004-05-03

Rekord: [◀] [◀] [1] [▶] [▶] [▶] z 5

5.5.2. Formularz frmRejestracjaZwrotu

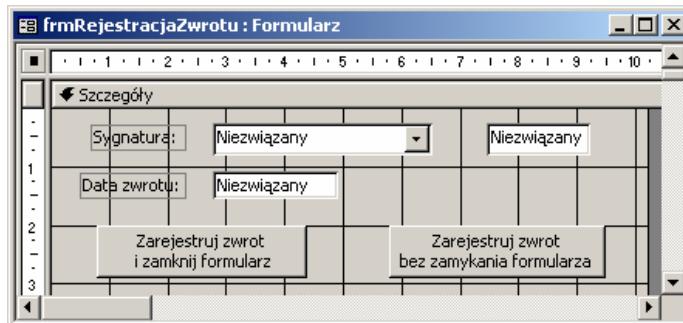
Formularz przeznaczony do zarejestrowania zwrotu wypożyczonej książki powinien być maksymalnie prosty w użyciu. Z projektu naszej bazy danych wynika, że do odnotowania faktu zwrotu wystarczy nam numer sygnatury. Na tej podstawie odszukamy odpowiadający jej identyfikator (`ID_S`) oraz identyfikator klienta (`ID_K`) w tabeli `Wypożyczenia`. Wystarczy dalej, że przygotujemy trzy kwerendy parametryczne aktualizujące:

Datę zwrotu w tabeli `Wypożyczenia` (pole `DataZ` ma otrzymać faktyczną datę zwrotu),

Pole `Status` w tabeli `Klienci` (zmniejszenie o jeden),

Pole `Status` w tabeli `Sygnatury` (zmiana z wartości 2 na 1).

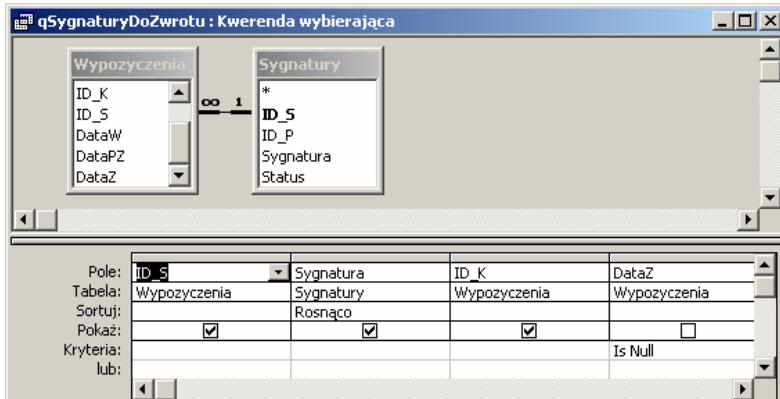
Zacznijmy od utworzenia nowego formularza, tym razem bez wskazania żadnego źródła danych. W sekcji Szczegóły umieszczać pole kombi o nazwie `cboSygnatury` oraz dwa przyciski poleceń o nazwach odpowiednio `cmdZarejestrujAndClose` oraz `cmdZarejestrujAndNext`.



Poza wymienionymi wyżej formantami w sekcji tej zostało także umieszczone pomocnicze pole tekstowe o nazwie `txtPoleUkryte`, wykorzystamy je w momencie

tworzenia kwerendy aktualizującej pole Status w tabeli Klienci. Taka pośrednia metoda wynika z faktu, że w parametrze tej kwerendy nie możemy odwołać się do właściwości *Column* obiektu *cboSygnatura*.

Kolejny krok to zaprojektowanie kwerendy dostarczającej dane do pola kombi *cboSygnatura*. Pokazana niżej kwerenda *qSygnaturyDoZwrotu* realizuje to zadanie.



Na pole DataZ nałożony został warunek Is Null (puste) – chodzi o to, aby zwrócić te rekordy, które odpowiadają pozycjom jeszcze nie zwróconym.

Po utworzeniu kwerendy *qSygnaturyDoZwrotu* zmieniamy istotne właściwości pola *cboSygnatura*.



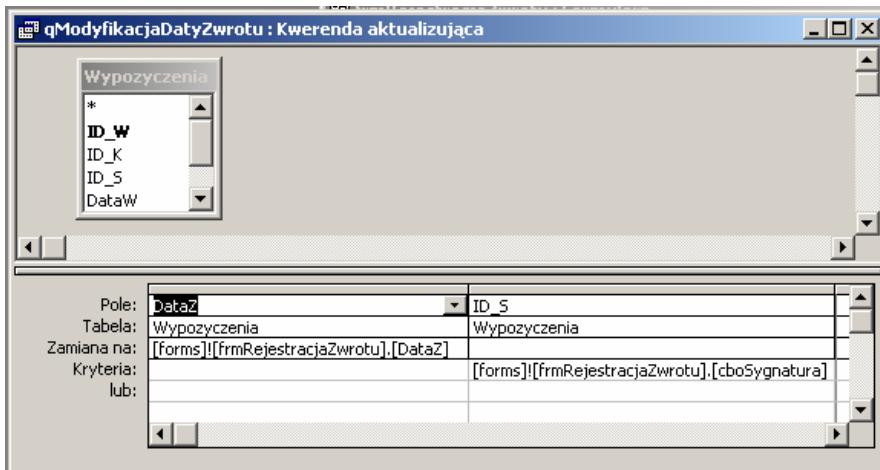
Musimy jeszcze utworzyć procedurę zdarzeniową realizowaną w momencie zajścia zdarzenia *Po aktualizacji*. Jej zadaniem jest odebranie z pola cboSygnatura wartości pola ID_K i przypisanie jej do pomocniczego pola txtPoleUkryte.

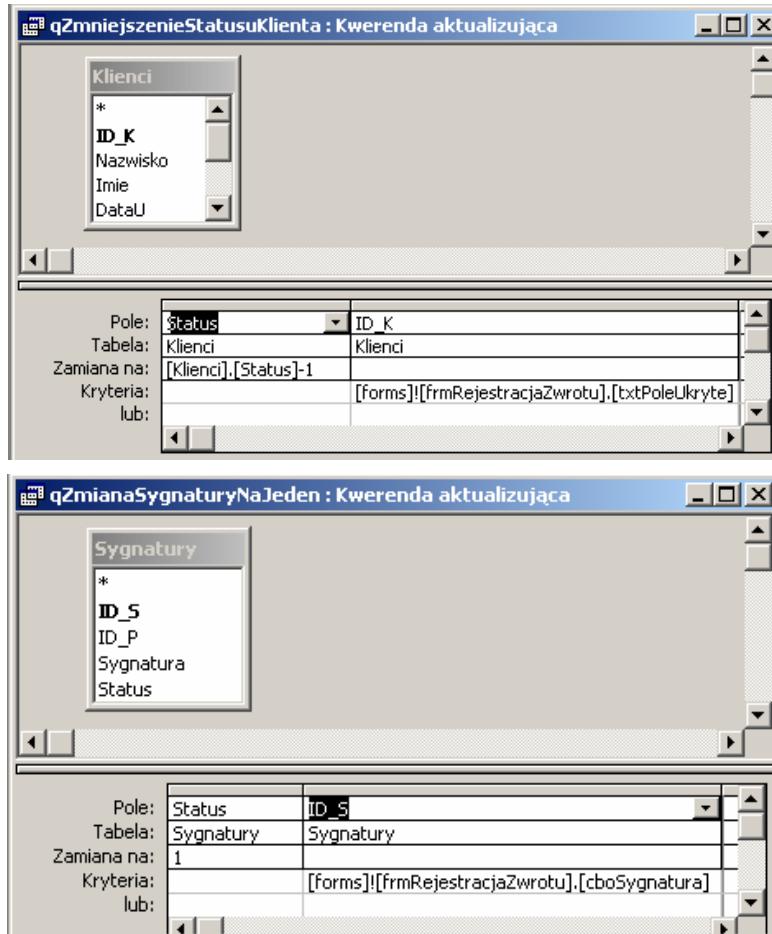
```
Private Sub cboSygnatura_AfterUpdate()
    txtPoleUkryte = Me.cboSygnatura.Column(2)
End Sub
```

Poza polem kombi w formularzu umieszczone jest także pole tekstowe o nazwie DataZ (ale nie jest połączone z polem o tej samej nazwie w tabeli Wypożyczenia), jego wartość domyślna została tak zdefiniowana, aby pokazywała bieżącą datę.



Przed napisaniem procedur obsługujących zdarzenie rejestracji zwrotu musimy zaprojektować trzy kwerendy aktualizujące.





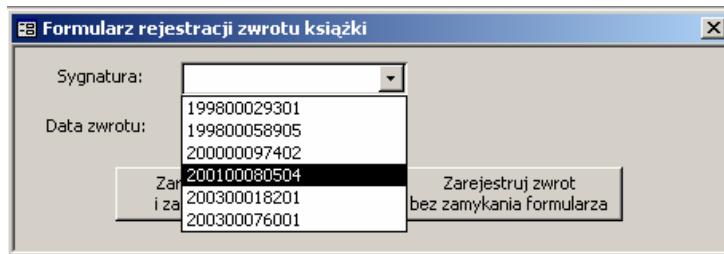
Można już napisać obie procedury reagujące na klik przycisków poleceń. Druga z tych procedur musi uporać się także z odświeżeniem źródła danych dla pola kombi cboSygnatura (zwrócona sygnatura **musi** zniknąć z listy tego pola).

```
Private Sub cmdZarejestrujAndClose_Click()
    DoCmd.OpenQuery "qModyfikacjaDatyZwrotu"
    DoCmd.OpenQuery "qZmniejszenieStatusuKlienta"
    DoCmd.OpenQuery "qZmianaSygnaturyNaJeden"
    DoCmd.Close
End Sub
```

```
Private Sub cmdZarejestrujAndNext_Click()
    DoCmd.OpenQuery "qModyfikacjaDatyZwrotu"
    DoCmd.OpenQuery "qZmniejszenieStatusuKlienta"
    DoCmd.OpenQuery "qZmianaSygnaturyNaJeden"
    Me.cboSygnatura.Requery
End Sub
```

Możemy już pokazać formularz frmRejestracjaZwrotu w działaniu; zobaczymy, czy to, co zaprojektowaliśmy spełnia nasze oczekiwania.

Powiedzmy, że w naszej bibliotece pojawił się jeden z naszych klientów z zamiarem zwrotu jednej z wcześniej wypożyczonych książek. Dla zarejestrowania tego zdarzenia otwieramy formularz frmRejestracjaZwrotu i w polu „Sygnatura” znajdujemy sygnaturę zwracanej pozycji.



Po wyborze sygnatury możemy zarejestrować zwrot jednym z dwóch przycisków rejestrujących, zależnie od tego, czy mamy dalsze pozycje do zwrotu czy też nie.

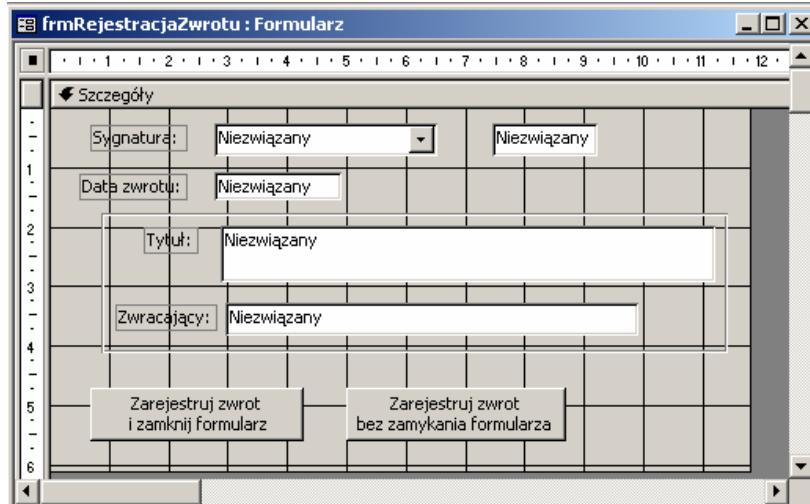


Formalnie wszystko jest w porządku, fakt zwrócenia książki jest odnotowany w tabeli Wypożyczenia poprzez wpis w polu DataZ (data zwrotu), modyfikowane są pola Status w tabelach Klienci i Sygnatury.

Powinniśmy się zastanowić, czy nie warto uzupełnić naszego formularza o dwa dodatkowe pola, w których na podstawie sygnatury zidentyfikujemy tytuł zwracanej książki oraz nazwisko i imię osoby, która ją wypożyczyła. Te dwie informacje będą miały **wyłącznie** charakter uzupełniający i kontrolny, przykładowo zmniejszą ryzyko pomylenia sygnatur.

Realizacja tego dodatkowego zadania wymagać będzie umieszczenia w formularzu odpowiednich kontrolek (dwie etykiety i dwa pola tekstowe), ustawienia właściwości pól tekstowych w taki sposób, aby nie można było zmienić ich zawartości. Musimy także rozbudować znacznie procedurę `cboSygnatura_AfterUpdate()` o instrukcje, które dostarczą danych do tych dwóch dodatkowych pól.

Poniżej pokazany jest zmodyfikowany projekt naszego formularza, dwa dodatkowe pola tekstowe o nazwach `txtTytul` i `txtNazwiskoImie` wraz z ich etykietami zostały umieszczone w prostokącie dla zaakcentowania ich odrębnego charakteru.

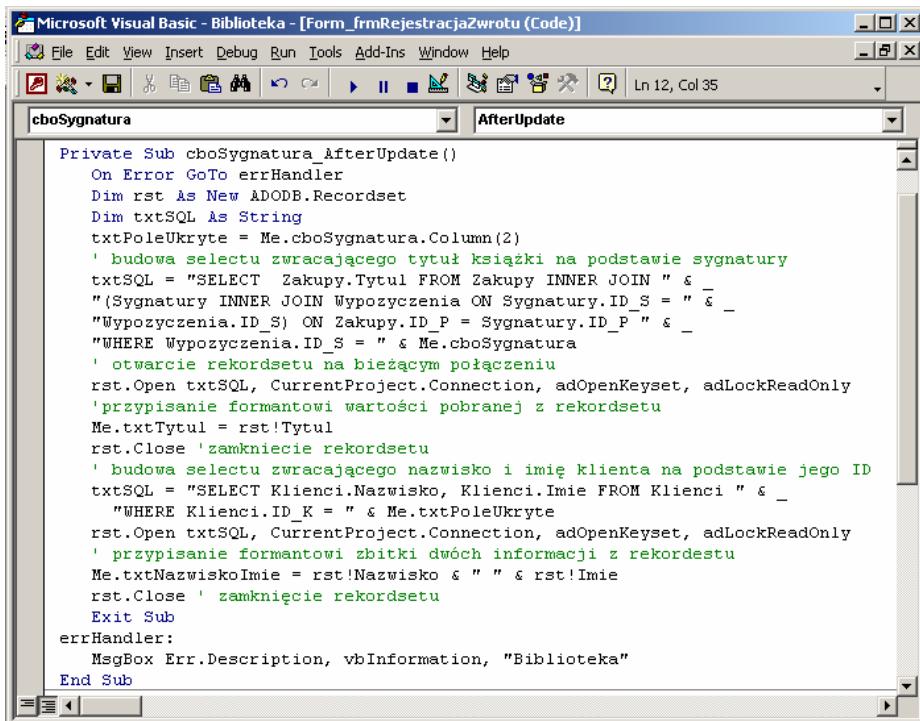


Z uwagi na charakter informacyjny tych dwóch pól zostały odpowiednio ustawione ich właściwości *Włączone* (na *Nie*) oraz *Zablokowane* (na *Tak*), co **nie pozwoli** na przeniesienie fokusu do tych pól.

Kolejny krok to taka rozbudowa procedury zdarzenia *Po aktualizacji* pola `cboSygnatura` tak, aby po wyborze sygnatury z listy pola kombi dostarczyć do obu pól odpowiednie informacje. Zadanie to zrealizujemy poprzez odpowiednio skonstruowane zapytania SQL-owe. Poniżej pokazany jest widok okna kodu tak zmodyfikowanej procedury wraz z komentarzami.

Przed pokazaniem zmodyfikowanego formularza w działaniu musimy jeszcze dodać dwie instrukcje w procedurze `cmdZarejestrujAndNext_Click()`, ich zadaniem jest „wyczyszczanie” obu pól tekstowych.

```
Me.txtTytul = ""
Me.txtNazwiskoImie = ""
```

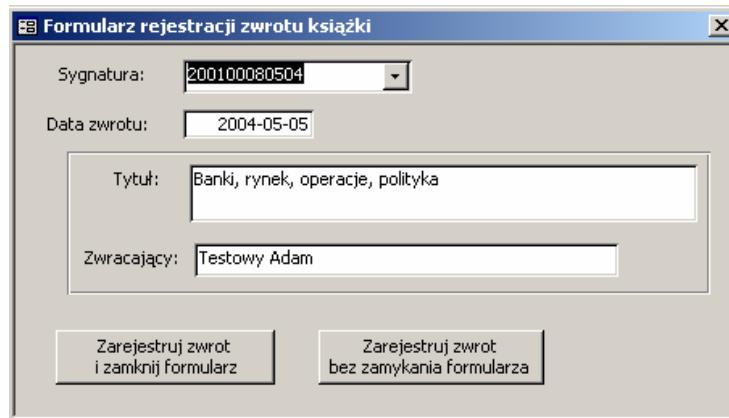


```

Private Sub cboSygnatura_AfterUpdate()
    On Error GoTo errHandler
    Dim rst As New ADODB.Recordset
    Dim txtSQL As String
    txtPoleUkryte = Me.cboSygnatura.Column(2)
    ' budowa selectu zwracającego tytuł książki na podstawie sygnatury
    txtSQL = "SELECT Zakupy.Tytul FROM Zakupy INNER JOIN " & _
    "(Sygnatury INNER JOIN Wypożyczenia ON Sygnatury.ID_S = " & _
    "Wypożyczenia.ID_S) ON Zakupy.ID_P = Sygnatury.ID_P" & _
    "WHERE Wypożyczenia.ID_S = " & Me.cboSygnatura
    ' otwarcie rekordsetu na bieżącym połączaniu
    rst.Open txtSQL, CurrentProject.Connection, adOpenKeyset, adLockReadOnly
    ' przypisanie formantowi wartości pobranej z rekordsetu
    Me.txtTytul = rst!Tytul
    rst.Close ' zamknięcie rekordsetu
    ' budowa selectu zwracającego nazwisko i imię klienta na podstawie jego ID
    txtSQL = "SELECT Klienci.Nazwisko, Klienci.Imie FROM Klienci" & _
    "WHERE Klienci.ID_K = " & Me.txtPoleUkryte
    rst.Open txtSQL, CurrentProject.Connection, adOpenKeyset, adLockReadOnly
    ' przypisanie formantowi zbitki dwóch informacji z rekordestu
    Me.txtNazwiskoImie = rst!Nazwisko & " " & rst!Imie
    rst.Close ' zamknięcie rekordsetu
    Exit Sub
errHandler:
    MsgBox Err.Description, vbInformation, "Biblioteka"
End Sub

```

A tak wygląda zmodyfikowany formularz rejestracji zwrotu w „działaniu”, po wyborze sygnatury oba pola tekstowe dostarczają dodatkowych informacji o zwracanej pozycji i osobie, która miała tę pozycję w czytaniu.



The dialog box contains the following fields:

- Sygnatura: dropdown menu showing "200100080504"
- Data zwrotu: text input field showing "2004-05-05"
- Tytuł: text input field showing "Banki, rynek, operacje, polityka"
- Zwracający: text input field showing "Testowy Adam"
- Buttons at the bottom:
 - Zarejestruj zwrot i zamknij formularz
 - Zarejestruj zwrot bez zamykania formularza

5.6. Wyszukiwanie informacji

Możemy teraz przejść do przygotowania formularza ułatwiającego wyszukiwanie interesującej potencjalnego klienta pozycji książkowej. Można tu zastosować wiele rozwiązań, my zaproponujemy przykładowo wyszukiwanie pozycji wg pierwszej litery ich tytułu pokazując jednocześnie jej autorów. Rozwiążanie takie podyktowane jest także chęcią pokazania Czytelnikowi stosunkowo łatwego i eleganckiego rozwiązania.

5.6.1. Według tytułów pozycji

Naszym zamiarem jest zbudowanie formularza, który będzie zwracał tytuł pozycji i dane jej autora lub autorów w postaci nazwiska i pierwszej litery imienia zależnie od wskazanej przez użytkownika pierwszej litery alfabetu.

Prace musimy zacząć od przygotowania źródła danych dla tak określonego formularza, źródło to powinno zwracać dla każdej zakupionej pozycji dwie informacje:

Tytuł pozycji pobrany z tabeli `Zakupy`,

Autorzy, pole tekstowe skonstruowane w oparciu o tabele `Zakupy`, `Autorzy` i `PozycjeAutorzy`, a zwracające dla każdej pozycji listę jej autorów w formacie nazwiska i pierwszej litery imienia.

Dla większości zakupionych pozycji nie ma problemu, co wynika z faktu, że pozycje te napisał jeden autor. Problem staje się skomplikowany wtedy, gdy autorem danej pozycji jest dwóch i więcej autorów. Jako źródło danych dla projektowanego formularza zaproponujemy tymczasową tabelę o nazwie `tempAutorzyTytuly`, jej projekt pokazany jest niżej.

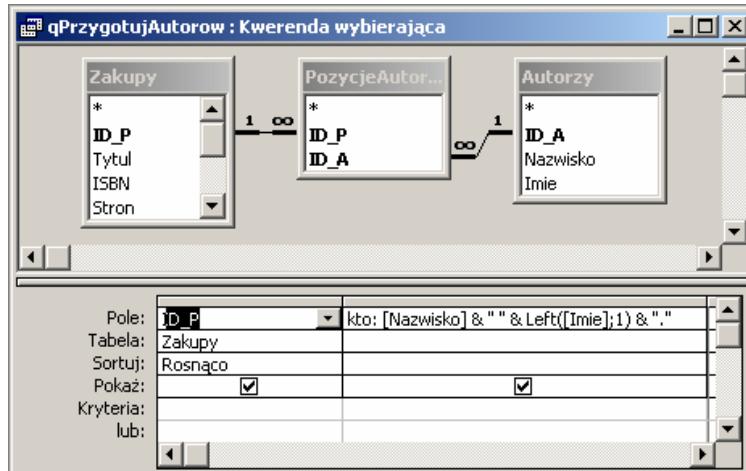
tempAutorzyTytuly : Tabela		
	Nazwa pola	Typ danych
ID_P		Liczba
Autorzy		Tekst

Właściwości pola	
Ogólne	Odnośnik
Rozmiar pola	Liczba całkowita długa
Format	
Miejsca dziesiętne	Auto
Maska wprowadzania	
Tytuł	
Wartość domyślna	
Reguła poprawności	
Komunikat o błędzie	
Wymagane	Tak
Indeksowane	Tak (Bez powtórzeń)

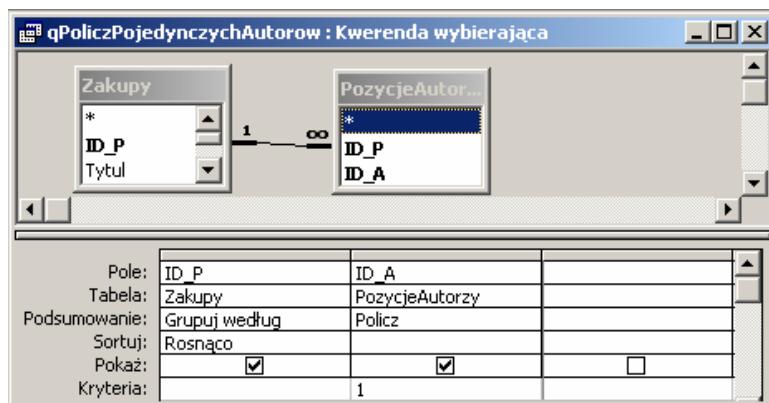
Nazwa pola może mieć maksymalnie 64 znaki, włączając spacje. Naciśnij klawisz F1, aby uzyskać pomoc na temat nazw pól.

Przyjmiemy takie ustalenie, że w momencie otwierania naszego formularza dane tej tabeli będą na nowo budowane, częściowo poprzez kwerendy dołączające (dla pojedynczych autorów), częściowo poprzez procedury VBA (dla wielu autorów jednej pozycji).

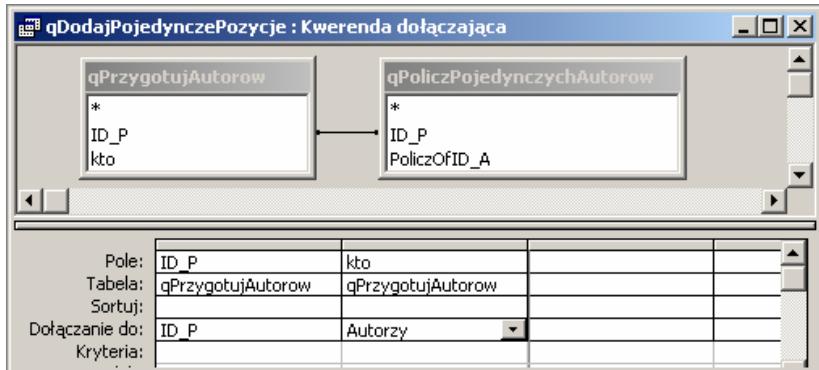
Zaczynamy od przygotowania kwerendy qPrzygotujAutorow, jej zadaniem jest zwrócenie dla każdej pozycji identyfikowanej przez ID_P pola wyliczanego kto zwracającego nazwisko autora i pierwszą literę imienia zakońzoną kropką.



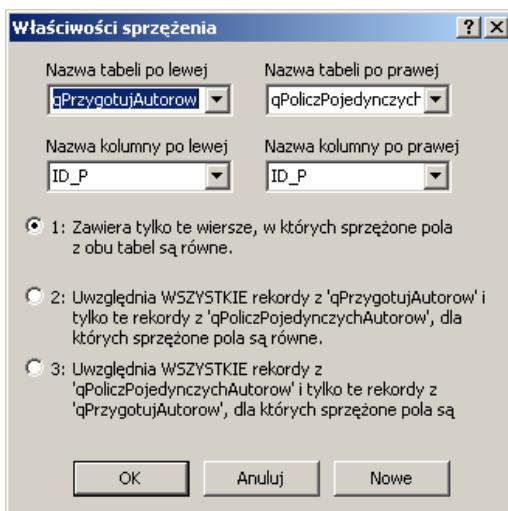
Kolejny krok, to wykaz tych pozycji, które napisał jeden autor. Zadanie to realizuje prosta kwerenda podsumowująca qPoliczPojedynczychAutorow.



Obie pokazane wyżej kwerendy wykorzystujemy w kwerendzie dołączającej, której projekt pokazany jest poniżej. Po jej uruchomieniu do tabeli `tempAutorzyTytuly` dołączone zostaną rekordy odpowiadające pozycjom z pojedynczym autorem..

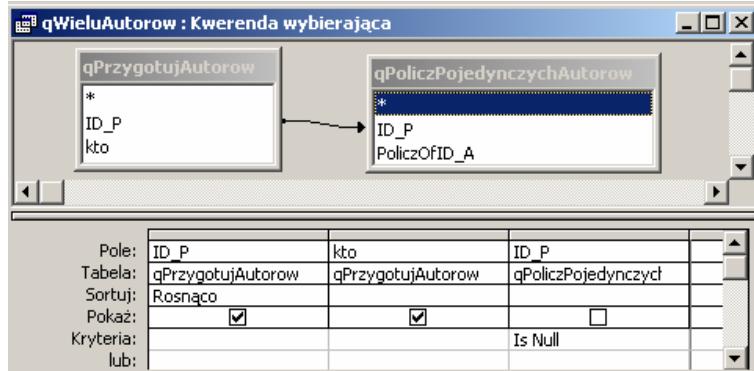


Takie ograniczenie zostaje wymuszone dzięki odpowiedniej relacji między polami `ID_P` obu kwerend wykorzystanych w projekcie kwerendy dołączającej

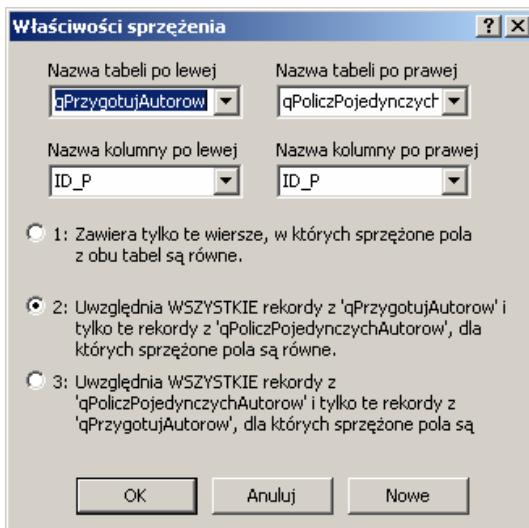


Pozostało nam przygotowanie jeszcze jednej kwerendy, którą wykorzystamy jako źródło danych w procedurze VBA, której zadaniem będzie dołączenie do tabeli pomocniczej `tempAutorzyTytuly` rekordów odpowiadających tym pozycjom z naszego księgozbioru, które mają więcej niż jednego autora.

Do zbudowania tej kwerendy wykorzystamy przygotowane wcześniej kwerendy, czyli `qPrzygotujAutorow` oraz `qPoliczPojedynczychAutorow` z warunkiem `Is Null` nałożonym na pole `ID_P` ostatniej z kwerend i odpowiednio zdefiniowanej relacji między obu kwerendami.



Od tej kwerendy oczekujemy zwrotu takich rekordów, gdzie pole `ID_P` dla danego zasobu będzie powtórzone co najmniej dwa razy.



Połączenie typu relacji i warunku nałożonego na pole `ID_P` pochodzące z kwerendy `qPoliczPojedynczychAutorow` powoduje, że tak zaprojektowana kwerenda wybierająca zwraca potrzebne nam rekordy danych.

ID_P	kto
76	Jędrzejczyk Z.
76	Kukula K.
80	Warężak A.
80	Fabijańczyk M.
82	Kłopotowski J.
82	Nykowski I.
92	Macaire D.
92	Nicolas G.
94	Jager A.
94	Iasenov S.

Dane zwracane przez kwerendę qWieluAutorow wykorzystamy w kodzie VBA do programowego dołączenia do tabeli tempAutorzyTytuly rekordów zawierających dla każdej ID_P z tej kwerendy zbitki informacji z pola kto. Przykładowo, dla pokazanej wyżej sytuacji dla ID_P = 76 pole Autorzy pomocniczej tabeli otrzyma wartość tekstową „Jędrzejczyk Z., Kukula K.”

Tak sformułowane zadanie zrealizujemy za pomocą funkcji utworzonej w module ogólnym o nazwie Standardowy, a wybór funkcji wynika z przygotowań do uruchomienia własnego paska narzędziowego naszej aplikacji. Poszczególne przyciski z tego paska będą wywoływać odpowiadające im funkcje, a jedną z nich będzie właśnie funkcja uruchamiająca formularz wyszukiwania zasobów wg pierwszych liter ich tytułów.

Pokazane dalej okno projektu funkcji SearchOnTitle wykorzystuje jako metodę dostępu do danych standardowy model ADO. Po pierwsze budowane jest zapytanie usuwające wszystkie rekordy z pomocniczej tabeli tempAutorzyTytuly.

```
txtSQL = "DELETE tempAutorzyTytuly.* from _  
tempAutorzyTytuly"
```

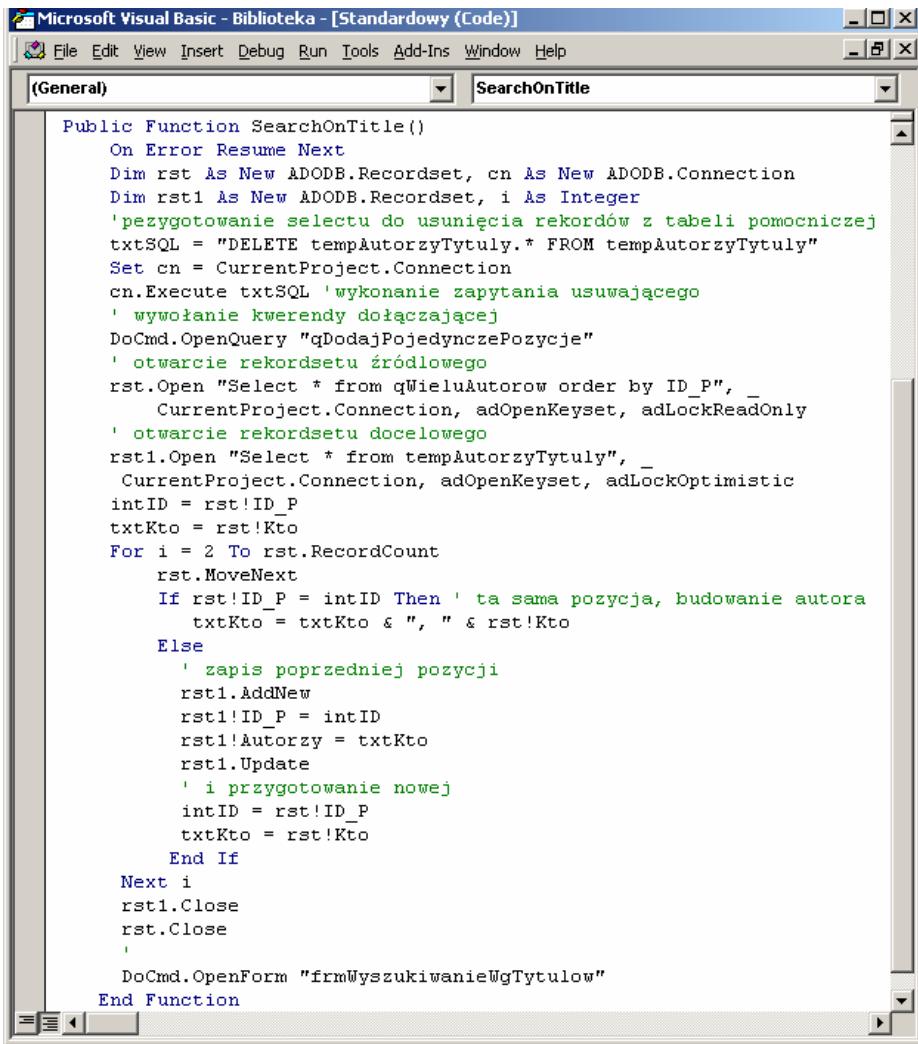
Zapytanie to jest wykonywane na bieżącym połączeniu (na połączeniu pobranym z aktualnego projektu).

```
Set cn = CurrentProject.Connection  
cn.Execute txtSQL
```

Po usunięciu dotychczasowych rekordów z pomocniczej tabeli dodawana jest pierwsza porcja rekordów poprzez uruchomienie kwerendy dołączającej.

```
DoCmd.OpenQuery "qDodajPojedynczePozyce"
```

Dalsze instrukcje funkcji SearchOnTitle operują na rekordsetach źródła danych (kwerenda qWieluAutorow) i miejsca przeznaczenia (tempAutorzyTytuly).



```

Microsoft Visual Basic - Biblioteka - [Standardowy (Code)]
File Edit Insert Debug Run Tools Add-Ins Window Help
(General) SearchOnTitle
Public Function SearchOnTitle()
    On Error Resume Next
    Dim rst As New ADODB.Recordset, cn As New ADODB.Connection
    Dim rst1 As New ADODB.Recordset, i As Integer
    ' przygotowanie selectu do usunięcia rekordów z tabeli pomocniczej
    txtSQL = "DELETE tempAutorzyTytuly.* FROM tempAutorzyTytuly"
    Set cn = CurrentProject.Connection
    cn.Execute txtSQL ' wykonanie zapytania usuwającego
    ' wywołanie kwerendy dołączającej
    DoCmd.OpenQuery "qDodajPojedynczePozycje"
    ' otwarcie rekordsetu źródłowego
    rst.Open "Select * from qWieluAutorow order by ID_P",
        CurrentProject.Connection, adOpenKeyset, adLockReadOnly
    ' otwarcie rekordsetu docelowego
    rst1.Open "Select * from tempAutorzyTytuly",
        CurrentProject.Connection, adOpenKeyset, adLockOptimistic
    intID = rst!ID_P
    txtKto = rst!Kto
    For i = 2 To rst.RecordCount
        rst.MoveNext
        If rst!ID_P = intID Then ' ta sama pozycja, budowanie autora
            txtKto = txtKto & ", " & rst!Kto
        Else
            ' zapis poprzedniej pozycji
            rst1.AddNew
            rst1!ID_P = intID
            rst1!Autorzy = txtKto
            rst1.Update
            ' i przygotowanie nowej
            intID = rst!ID_P
            txtKto = rst!Kto
        End If
    Next i
    rst1.Close
    rst.Close
    '
    DoCmd.OpenForm "frmWyszukiwanieWgTytulow"
End Function

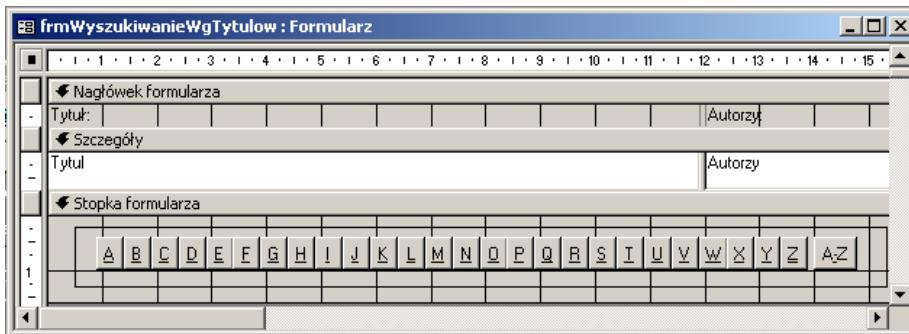
```

Przed zakończeniem funkcji wywoływany jest docelowy formularz tego rozdziału, czyli frmWyszukiwanieWgTytulow. Oczywiście formularz ten musimy teraz zaprojektować, ale mamy już przygotowane jego źródło danych – tabelę pomocniczą tempAutorzyTytuly.

W sekcji *Szczegóły* projektu nowego formularza umieszczamy dwa pola tekstowe odpowiadające polom ID_P i Autorzy tabeli tempAutorzyTytuly. W sekcji

Nagłówek formularza umieszczamy dwie etykiety opisujące kolumny danych wyświetlane w sekcji *Szczegóły* – rozwiązanie takie jest niezbędne z uwagi na to, że chcemy mieć formularz w widoku formularzy ciągłych.

W sekcji *Stopka formularza* umieścimy teraz ramkę grupy opcji o nazwie *Filtr Tytuly*, a w niej 27 przycisków przełączników. W efekcie nasz formularz powinien wyglądać tak, jak pokazany niżej.



Właściwość *Wartość domyślna* ramki grupy opcji została ustawiona na 27, umownie przyjmijmy, że będzie to oznaczać pokazanie wszystkich rekordów ze źródła danych. Właściwość *Po aktualizacji* została tak ustawiona, aby było wywoływanie makropolecenie ustawiające właściwość *Filtr* formularza. Makropolecenie to o nazwie *FiltrTytuly* za moment napiszemy, jego zadaniem będzie uruchamianie właściwego, zależnie od użytego przełącznika, filtru ograniczającego wyświetlane rekordy.

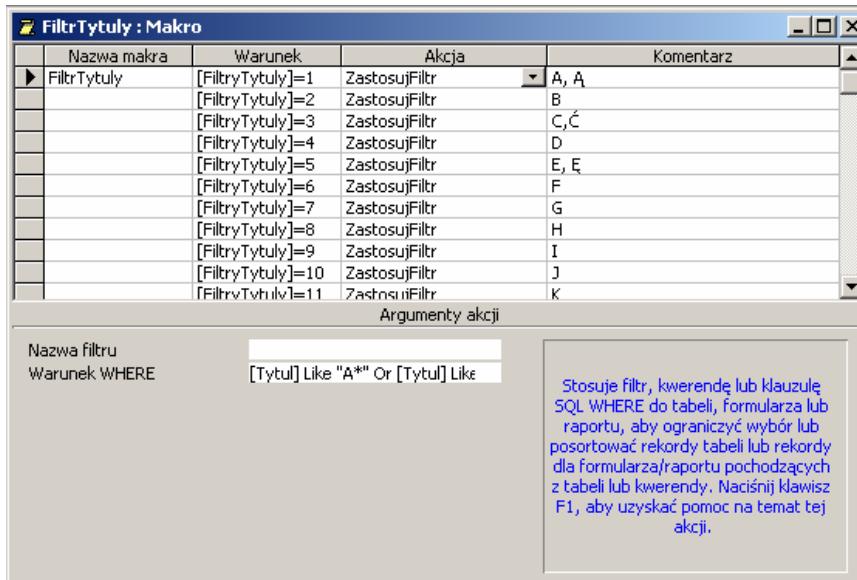
Każdy z 27 przełączników opcji ma odpowiednio zmienioną właściwość *Tytuł* oraz *Wartość opcji* – od 1 do 27.



Klik przycisku przełącznika oznaczonego literą „A” spowoduje, że ramka grupy opcji zwróci wartość 1, z literą „B” 2 itd. Przycisk oznaczony symbolem „A-Z” zwróci wartość 27, czyli taką, która spowoduje pokazanie wszystkich rekordów.

Przed przystąpieniem do tworzenia makropolecenia *FiltrTytuly* zmienimy jeszcze właściwość *Filtr dozwolony* na *Tak*, chodzi o to, aby makro mogło modyfikować właściwość *Filtr*.

Pracę nad makropoleceniem zaczynamy od utworzenia nowego makra, a następnie od wyświetlenia kolumn *Nazwa makra* oraz *Warunki* (z menu *Widok*). W kolumnie *Nazwa makra* wpisujemy dowolną nazwę makropolecenia (ale zgodnie z regułami nazw), a w kolumnie *Warunek* kolejno wpisujemy warunki nakładane na formant *FiltrTytyly*. W kolumnie *Akcja* każdemu z warunków przypisujemy akcję *ZastosujFiltr*. Warto także wypełnić pole *Komentarz*.



Dla warunku *[FiltrTytyly]=27* w kolumnie *Akcja* ustawiamy akcję *PokażWszystkieRekordy*. Po zdefiniowaniu wszystkich warunków musimy określić działania, jakie ma podjąć makro po zastosowaniu filtru. Działania te muszą być zróżnicowane zależnie od tego, czy były jakieś rekordy pokazane, czy też nie.

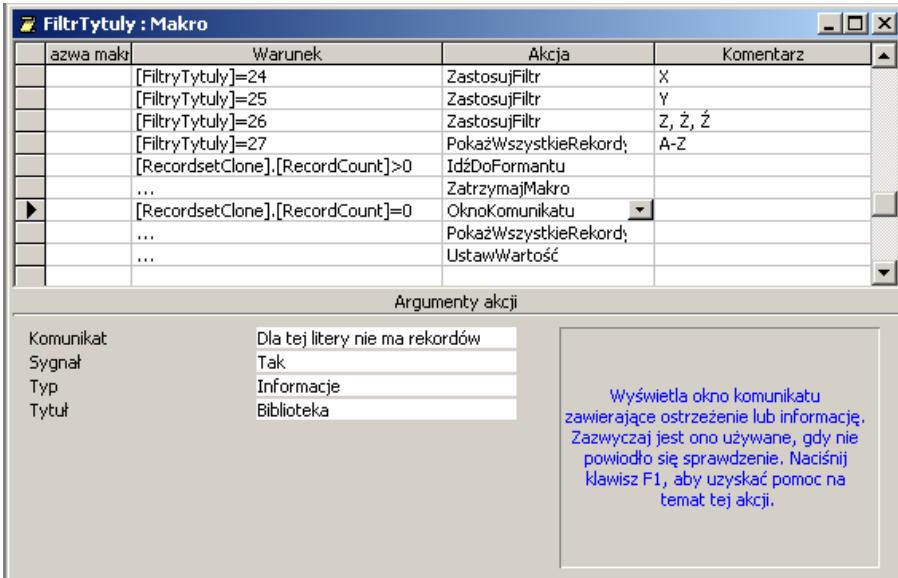
W sytuacji, gdy były rekordy do pokazania musimy umieścić wskaźnik bieżącego rekordu w polu tekstowym *txtTytul* pierwszego zwróconego rekordu i zatrzymać akcję makra.

W polu *Warunek* wpisujemy *[RecordsetClone]. [RecordCount]>0*, a w polu *Akcja* *IdźDoFormantu* podając jako argument tej akcji nazwę *txtTytul*. W kolejnym wierszu pola *Akcja* wpisujemy (dokładniej wybieramy z listy) akcję *ZatrzymajMakro*. Ta akcja dzieje się jako kontynuacja poprzedniego warunku, dlatego też w polu *Warunek* są wpisane trzy kropki jako symbol kontynuacji.

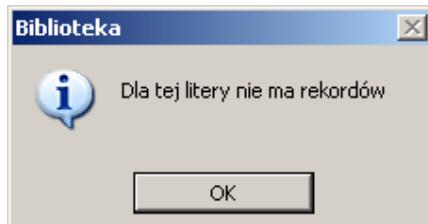
W sytuacji, gdy w wyniku zastosowania filtra nie zostanie zwrocony żaden rekord potrzebny jest jakiś komunikat dla użytkownika, aby wiedział co się dzieje. Zadanie to jest realizowane wtedy, gdy spełniony jest warunek:

[RecordsetClone]. [RecordCount] = 0

Akcje podejmowane przy prawdziwości tego warunku to OknoKomunikatu, PokażWszystkieRekordy oraz UstawWartość. Poniżej pokazane jest okno projektu tego makropolecenia ze szczegółami pierwszej z podejmowanych akcji.



Komunikat wyświetlany w sytuacji, gdy prawdziwy jest warunek [RecordsetClone]. [RecordCount] = 0.



Po akceptacji kolejna akcja spowoduje wyświetlenie wszystkich rekordów, a ramka grupy opcji FiltrTytuly otrzyma wartość 27.

Możemy już zobaczyć formularz frmWyszukiwanieWgTytulow w działaniu, w pokazanej sytuacji wyświetlane są te pozycje biblioteczne i ich autorzy, dla których tytuł zaczyna się na literę „K”.

Wyszukiwanie tytułów wg liter alfabetu	
Tytuł:	Autorzy:
Kontrola w zarządzaniu ochroną środowiska	Sobczak K.
Kredytowanie działalności gospodarczej małych i średnich przedsiębiorstw na przykładzie Fortis Banku	Rutkowska K.
Kontrola jako element zarządzania firmą - aspekt audytu i controlingu	Boczkowska Z.
Kapitał intelektualny	Edvinsson L., Malone M.
Kierowanie	Freeman E., Wankel C., Gilbert D., Stoner J.
Kierować sobą i innymi	Muri P., Kalin K.
Kierowanie operacyjne w przedsiębiorstwie. Metody...	Bierkowska K., Szumarski W.
Kontrola finansowo-księgowa	Wiśnierska K.
Kieszonkowy słownik niem.-pol. pol.-niem	Schmitzek S., Czochalski J.
Kodeks Spółek Handlowych z przepisami okókokodekso wymi	Wiśniewski A.

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | A-Z |

Rekord: [| < | > | >> | >>> | z 41 (Filtr)

5.7. Raporty

Dla prześledzenia problematyki tworzenia raportów jako pisemnych wyciągów informacji zgromadzonych w bazie danych prześledzimy budowę dwóch raportów. Pierwszy z nich będzie klasycznym raportem statycznym w sensie liczby kolumn raportu, drugi będzie raportem dynamicznym (w sensie liczby kolumn) budowanym całkowicie programowo za pomocą procedur VBA.

5.7.1. Raport zbiorczy zakupów

Zacznijmy nasze rozważania od zbudowania raportu, którego zadaniem jest pokazanie wartości zakupionych zasobów bibliotecznych w ujęciu „lata x miesiące”. Prace nad tym raportem zacznijmy od przygotowania źródła danych w postaci kwerendy krzyżowej. Poniżej pokazany jest projekt takiej kwerendy, jest ona oparta jedynie o tabelę Zakupy, a jej pola są polami wyliczonymi.

Pierwsze dwa zwracają odpowiednio numer roku (poprzedzony literą „r”) i numer miesiąca, dwa kolejne zwracają wartość zakupu (w miesiącu danego roku) oraz sumaryczną

wartość zakupów w danym miesiącu. Pierwsze pole wyliczane o umownej nazwie „rr” zwraca informacje, które zostaną wykorzystane jako nagłówki kolumn. Dodanie litery „r” przed cyframi roku podyktowane było wymogiem, aby nazwa pola nie była liczbą, co mogłoby spowodować trudności w projekcie formularza, gdzie powinniśmy uzyskać podsumowania kolumn (lat).

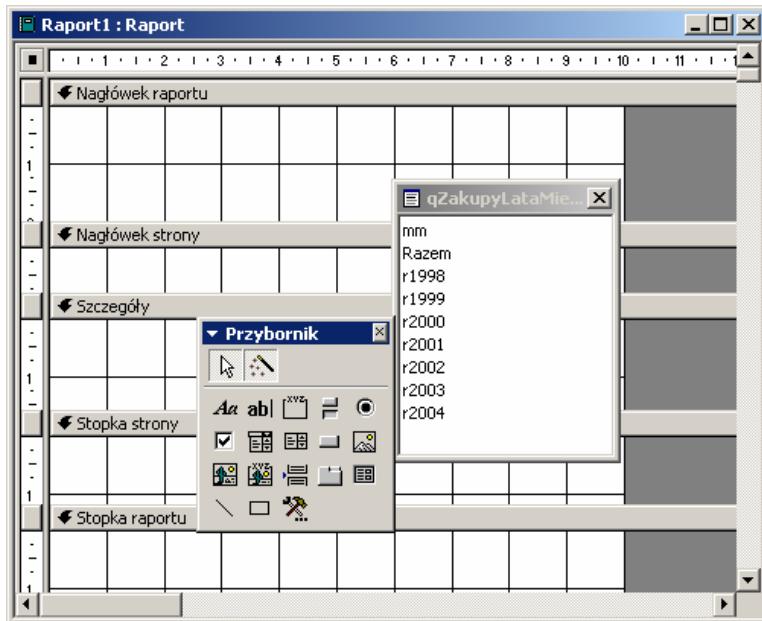


Pole wyliczane o umownej nazwie „mm” zwraca numer miesiąca, wykorzystamy je jako nagłówek wiersza. W tym wypadku nie było potrzeby poprzedzenia zwracanego numeru miesiąca literą np. „m” z tego prostego powodu, że podsumowanie sprzedaży wierszami jest wykonywane bezpośrednio w tej kwerendzie (pole „Razem”), a nie w raporcie. Poniżej widok zwracanych danych przez tak zaprojektowaną kwerendę.

	mm	Razem	r1998	r1999	r2000	r2001	r2002	r2003	r2004
►	1	8 345,52 zł	440,00 zł	987,88 zł	506,80 zł	245,00 zł	230,50 zł	935,34 zł	
	2	4 307,96 zł	885,44 zł	1 350,70 zł	1 365,22 zł	165,00 zł	42,00 zł	499,60 zł	
	3	7 561,66 zł	3 686,51 zł	763,08 zł	547,60 zł		22,00 zł	468,72 zł	73,75 zł
	4	4 492,44 zł	244,10 zł	435,00 zł	1 508,40 zł	608,20 zł	249,00 zł	647,74 zł	
	5	6 661,43 zł	1 120,28 zł	1 862,50 zł	346,00 zł	176,65 zł	120,00 zł	497,00 zł	539,00 zł
	6	2 139,61 zł		70,00 zł	1 773,61 zł	35,00 zł		261,00 zł	
	7	130,00 zł				85,00 zł	45,00 zł	0,00 zł	
	8	147,50 zł	11,50 zł	20,00 zł	71,00 zł		45,00 zł	0,00 zł	
	9	6 506,14 zł	1 473,08 zł	4 025,56 zł	92,00 zł		530,50 zł	385,00 zł	
	10	6 383,32 zł	2 421,80 zł	1 178,70 zł	62,00 zł	203,00 zł	70,00 zł	167,82 zł	280,00 zł
	11	4 593,63 zł	3 256,13 zł	335,10 zł	65,00 zł	149,40 zł	144,00 zł	414,00 zł	230,00 zł
	12	6 418,60 zł	2 548,00 zł	404,00 zł	2 618,80 zł		431,80 zł	416,00 zł	

Po przygotowaniu źródła danych możemy przystąpić do projektowania raportu. Zaczynamy tradycyjnie od otwarcia szablonu raportu opartego o przygotowaną kwerendę qZakupyLataMiesiące.

Poniżej widok projektu tworzonego raportu z listą pól oferowanych przez źródło danych oraz włączonym nagłówkiem i stopką raportu. W stopce raportu umieścimy formanty podsumowań zakupów w kolejnych latach.



Korzystając z pól dostępnych w źródle danych oraz formantów z przybornika doprowadzamy projekt raportu do postaci pokazanej niżej.



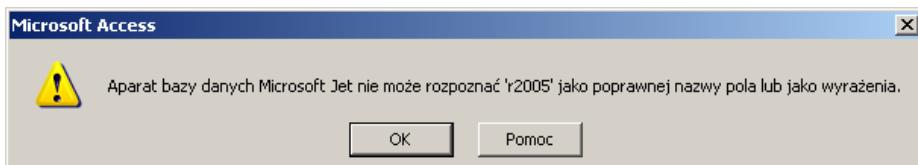
W stopce raportu zostały umieszczone formanty skopiowane z sekcji *Szczegóły* (z wyjątkiem pola *mm*), z tym, że źródło każdego formantu zostało zmodyfikowane o funkcję *Suma*, tak jak to widać np. dla podsumowania kolumny *Razem*. Gotowy raport pokazany jest poniżej.

Zbiorczy raport wartości zakupów w kolejnych latach i miesiącach								
Miesiąc	Razem	1998	1999	2000	2001	2002	2003	2004
1	8 345,52	440,00	987,88	506,80	245,00	230,50	5 935,34	
2	4 307,96	885,44	1 350,70	1 365,22	165,00	42,00	499,60	
3	7 561,66	3 686,51	763,08	547,60		22,00	2 468,72	73,75
4	4 492,44	244,10	435,00	1 508,40	1 008,20	249,00	1 047,74	
5	6 661,43	1 120,28	1 862,50	346,00	2 176,65	120,00	497,00	539,00
6	2 139,61		70,00	1 773,61	35,00		261,00	
7	130,00				85,00	45,00	0,00	
8	147,50	11,50	20,00	71,00		45,00	0,00	
9	6 506,14	1 473,08	4 025,56	92,00		530,50	385,00	
10	6 383,32	2 421,80	1 178,70	62,00	1 203,00	70,00	1 167,82	280,00
11	4 593,63	3 256,13	335,10	65,00	149,40	144,00	414,00	230,00
12	6 418,60	2 548,00	404,00	2 618,80		431,80	416,00	
	57 687,81	16 086,84	11 432,52	8 956,43	5 067,25	1 929,80	13 092,22	1 122,75

Strona: [1]

Przygotowany przez nas raport ma jednak jedną zasadniczą wadę – jest statyczny w sensie liczby kolumn. Oznacza to, że w kolejnym roku (tu 2005) odpowiednie pole pojawi się w źródle danych (kwerendzie *qZakupyLataMiesiące*), ale nie zostanie wyświetcone w przygotowanym raporcie do czasu, dopóki nie zmodyfikujemy projektu. Oczywiście nie jest możliwe przygotowanie takiego raportu, aby pewne dodatkowe kolumny były w nim zaprojektowane z pewnym wyprzedzeniem.

Przykładowo, modyfikacja utworzonego wcześniej raportu poprzez dodanie w sekcji szczegółów jeszcze jednego pola tekstowego, dla którego źródłem danych będzie np. pole *r2005* (kolejny rok), spowoduje wyświetlenie pokazanego niżej komunikatu o błędzie.



Jedynym rozwiązaniem naszego problemu będzie programowe zbudowanie raportu, temu zagadnieniu będzie poświęcony kolejny rozdział tej książki.

5.7.2. Dynamiczny raport zakupów

Przed napisaniem stosownej procedury VBA przygotujemy szablon naszego raportu, jest to o tyle konieczne, że domyślnie projekt raportu nie zawiera sekcji *Nagłówek/Stopka* raportu, a programowe jej włączenie nie jest możliwe. Wzorzec raportu będzie miał także zmienioną orientację strony z pionowej na poziomą – domyślnie przyjmujemy, że chodzi o otrzymanie raportu ze stosunkowo dużą liczbą lat. Oczywiście istnieje możliwość takiej modyfikacji procedury, którą za moment pokażemy, aby orientację strony uzależnić od rzeczywistych potrzeb.

Prace nad szablonem raportu zaczynamy od otwarcia projektu szablonu bez definiowania źródła jego danych, a następnie pokazujemy nagłówek/stopkę raportu i zapisujemy raport pod jakąś nazwą, np. *rWzor1*. Resztę prac będziemy już prowadzić w środowisku VBA tworząc funkcję o nazwie np. *DajRaportLataMiesiące*. Mamy nadzieję, że zamieszczone komentarze ułatwią zrozumienie działania tej funkcji. Jako miejsce jej utworzenia wybraliśmy moduł *Standardowy*, ponieważ jej wywołanie będzie miało miejsce z menu użytkownika, które utworzymy w kolejnym rozdziale. Z tych samych powodów funkcja ta zostanie zadeklarowana jako *Public*.

```
Public Function DajRaportLataMiesiące()
    Dim tt() As String, k As Integer
    Dim rap As Report, t As Control
    Dim conn As New ADODB.Connection, _
        rst As New ADODB.Recordset
    ' otwarcie połączenia na bieżącym projekcie
    Set conn = CurrentProject.Connection
        ' utworzenie obiektu raport na bazie
        przygotowanego szablonu
    Set rap = Application.CreateReport(, "rWzor1")
    ' określenie źródła danych dla raportu
    rap.RecordSource = "qZakupyLataMiesiące"
    ' otwarcie rekordsetu
    rst.Open "qZakupyLataMiesiące", conn
    ' ustalenie liczby pól w rekordsecie
    k = rst.Fields.Count
        ' przewymiarowanie zmiennej tablicowej, tu
        będą nazwy pól
    ReDim tt(k)
        ' pobranie w pętli listy nazw pól i zapisanie
        ich w zmiennej tablicowej
    For i = 0 To k - 1
        tt(i) = rst.Fields(i).Name
    Next i
    ' zdefiniowanie etykiety w nagłówku raportu
```

```
' podajemy nazwę raportu, rodzaj formantu, sekcję,
    położenie
Set t = Application.CreateReportControl(rap.Name, _
    acLabel, acHeader, , , 4000, 200, 6000, 530)
' modyfikacja kilku właściwości tej etykiety
With t
    .Caption = "Raport zbiorczy zakupów wg lat i
        miesięcy"
    .FontSize = 14
    .TextAlign = 2
    .SpecialEffect = 5
    .FontName = "TimesNewRoman"
End With
' dla kolejnych pól występujących w źródle danych
' zostaną utworzone pola tekstowe w sekci Szczegóły,
    etykiety pól w sekcji
' nagłówek strony oraz pola tekstowe podsumowań w
    sekcji stopka raportu
For i = 0 To k - 1
    ' utworzenie pola tekstowego z dynamicznie zmienianą
    ' właściwością Left - odsunięcie od lewej krawędzi
    Set t = Application.CreateReportControl(rap.Name, _
        acTextBox, acDetail, , , i * 1500, 0, 1000, 275)
    ' przypisanie źródła danych dla utworzonego formantu
    t.ControlSource = tt(i)
    ' jeżeli nie jest to pierwsze pole, to narzucenie
        formatu danych
    If i > 0 Then t.Format = "#,#0.00"
    ' utworzenie etykiety pola
    Set t = Application.CreateReportControl(rap.Name,
        acLabel, acPageHeader, _
        , , i * 1500 + 500, 700, 1000, 275)
    ' jeżeli jest to pole opisujące rok, to usunięcie
        litery r
    If i > 1 Then
        t.Caption = Right(tt(i), 4)
    Else
        t.Caption = tt(i)
    End If
    ' jeżeli jest to inne pole niż pierwsze (miesiąc),
        to utworzenie pola tekstowego w sekcji stopka
        raportu dla podsumowania
    If i > 0 Then
```

```

Set t =
    Application.CreateReportControl(rap.Name, _
        acTextBox, acFooter, , , i * 1500, _
        200, 1000, 275)
t.Format = "#,##0.00"
' przypisanie jako źródło danych wartości
zwróconej przez funkcję
' DSum
t.ControlSource = "=" & _
    Str(Round(Application.DSum(tt(i), _
rap.RecordSource), 2))
End If
Next i
' ustawienie wysokości sekcji Szczegóły
rap.Section(acDetail).Height = 275
' utworzenie dwóch linii w nagłówku strony oraz na
    początku
Set t = Application.CreateReportControl(rap.Name, _
    acLine, acFooter, , , 0, 0, rap.Width, 0)
Set t = Application.CreateReportControl(rap.Name, _
    acLine, acPageHeader, , , 0, 1000, rap.Width, 0)
' pokazanie utworzonego raportu na ekranie
DoCmd.OpenReport rap.Name, acViewPreview
End Function

```

Poniżej pokazany jest widok gotowego raportu, jest on bardzo podobny do utworzonego w poprzednim rozdziale z jedną, zasadniczą różnicą – ten raport automatycznie dostosowuje się do listy pól w źródle danych.

Raport zbiorczy zakupów wg lat i miesięcy

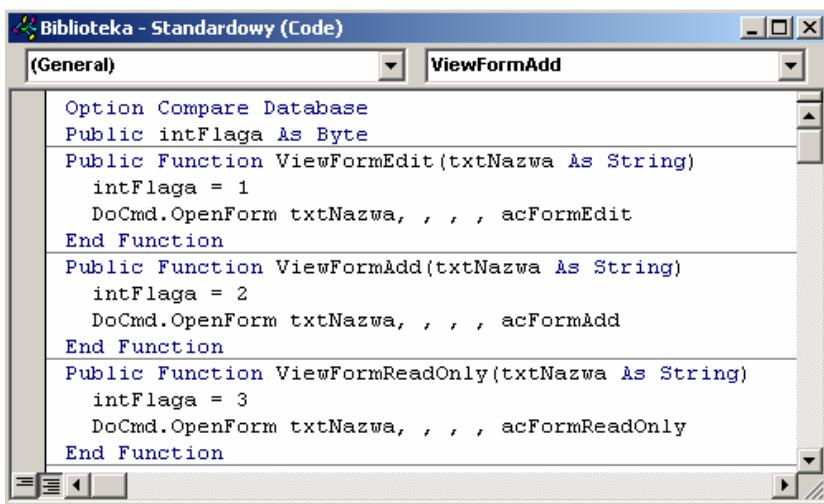
mm	Razem	1998	1999	2000	2001	2002	2003	2004
1	8345,52	440,00	987,88	905,80	245,00	230,50	5 935,24	
2	4301,96	885,44	1 350,00	1 365,22	165,00	42,00	499,60	
3	7561,06	3 686,51	763,08	947,60		22,00	2 468,12	737,15
4	4 492,44	244,10	435,00	1 508,40	1 008,20	249,00	1 047,74	
5	6661,43	1 120,28	1 862,50	346,00	2 176,65	120,00	497,00	539,00
6	2 139,61		70,00	1 773,61	35,00		261,00	
7	130,00				85,00		45,00	0,00
8	147,50	11,50	20,00	71,00			45,00	0,00
9	6505,14	1 473,08	4 025,56	92,00		530,50	385,00	
10	6383,32	2 421,80	1 178,70	62,00	1 203,00	70,00	1 167,82	280,00
11	4593,63	3 256,13	338,10	65,00	149,40	144,00	414,00	230,00
12	6 418,60	2 548,00	404,00	2 618,80		431,00	416,00	
	57 687,81	16 086,84	11 432,52	8 966,43	5 067,25	1 929,80	13 092,22	1 122,75

Strona: 14 / 1 | 1 ▶ ▶ ▶ ▶ ▶

5.8. Własny pasek narzędziowy

Po zbudowaniu wszystkich potrzebnych obiektów bazy danych pozostało jeszcze do rozwiązania problem łatwego ich uruchamiania. Jedną z lepszych możliwości jest przygotowanie własnego paska narzędziowego lub paska menu. W tej aplikacji ograniczyliśmy się do przygotowania paska narzędziowego o nazwie Biblioteka. Przyciski umieszczone w tym pasku będą sterować sposobem uruchamiania potrzebnych w danym momencie obiektów bazy danych.

W module standardowym VBA napiszemy jeszcze kilka funkcji, ich zadaniem będzie zróżnicowania otwierania formularzy. Będzie konieczna także modyfikacja kilku formularzy poprzez napisanie procedur obsługujących zdarzenie polegające na otwarciu formularza. W zależności od wartości zmiennej sterującej intFlaga będzie modyfikowany tytuł formularza oraz będą pokazywane (czy ukrywane) pewne formanty. Na kolejnej stronie pokazany jest fragment okna modułu Standardowy z trzema funkcjami pozwalającymi na otwarcie danego formularza w trzech różnych trybach pracy. Każda z tych funkcji ustawia także odpowiednio wartość zmiennej sterującej intFlaga.



```

Biblioteka - Standardowy (Code)
(General) ViewFormAdd

Option Compare Database
Public intFlaga As Byte
Public Function ViewFormEdit(txtNazwa As String)
    intFlaga = 1
    DoCmd.OpenForm txtNazwa, , , acFormEdit
End Function
Public Function ViewFormAdd(txtNazwa As String)
    intFlaga = 2
    DoCmd.OpenForm txtNazwa, , , acFormAdd
End Function
Public Function ViewFormReadOnly(txtNazwa As String)
    intFlaga = 3
    DoCmd.OpenForm txtNazwa, , , acFormReadOnly
End Function

```

Na przykładzie formularza frmKlient pokażemy konstrukcję procedury uruchamianej w momencie otwarcia tego formularza. Wykorzystując strukturę warunkową Select case modyfikowany jest tytuł formularza, a w przypadku formularza tylko do odczytu ukrywane są dwa przyciski poleceń, które w dwóch pozostałych trybach umożliwiają pokazanie dodatkowych formularzy.

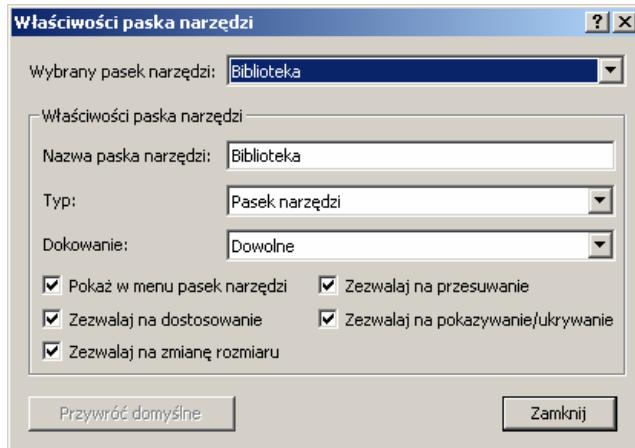
```

Private Sub Form_Open(Cancel As Integer)
    Select Case intFlaga
        Case 1
            Me.Caption = "Przegląd i edycja danych klienta"
        Case 2
            Me.Caption = "Rejestracja danych nowego klienta"
        Case 3
            Me.Caption = "Przegląd danych klientów - brak edycji"
            Me.cmdDomenaAdd.Visible = False
            Me.cmdUlicaKodAdd.Visible = False
    End Select
End Sub

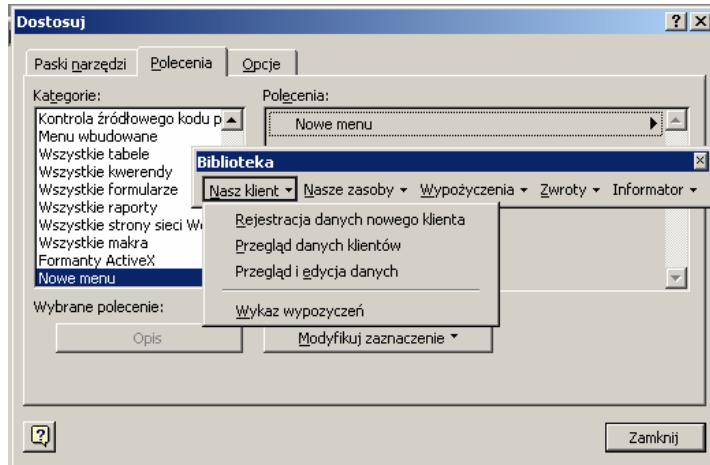
```

W analogiczny sposób powinniśmy zmodyfikować także i inne formularze, z pewnością taką potrzebą zachodzi dla formularza frmZakupy.

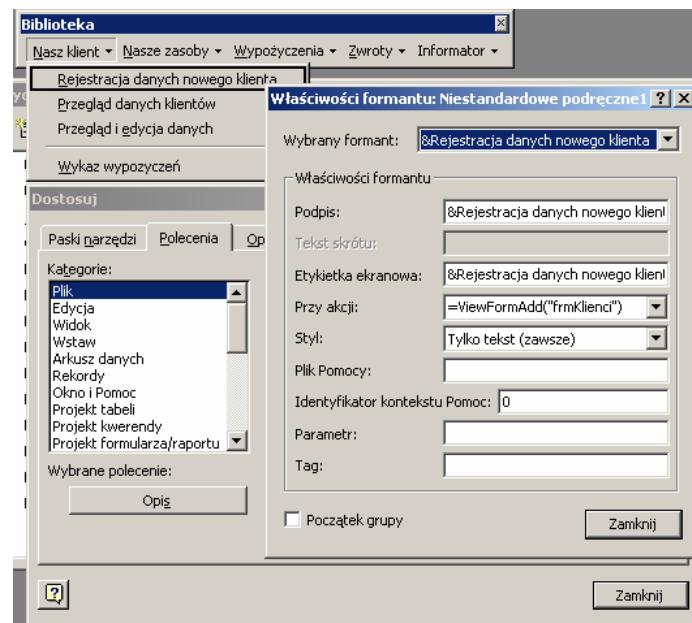
Po tych pracach przygotowawczych możemy już utworzyć nowy pasek narzędziowy, my nadaliśmy mu nazwę Biblioteka, a kilka jego własności pokazanych jest niżej.



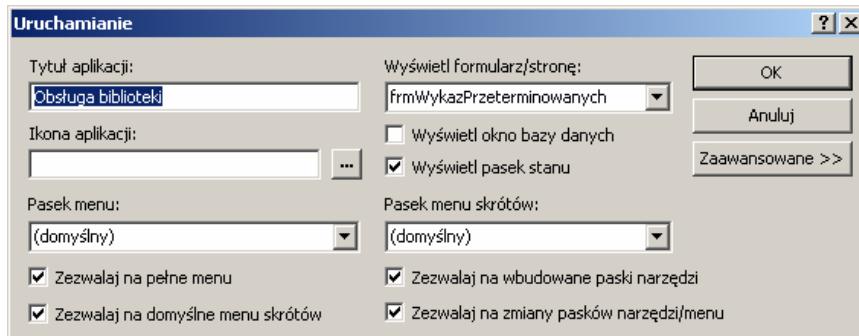
Do utworzonego paska narzędziowego dodajemy kolejno przyciski typu *Nowe menu* (dla hierarchizacji menu) oraz przyciski *niestandardowe*, które będą uruchamiały odpowiednie obiekty bazy danych.



Poniżej pokazana jest modyfikacja właściwości przycisku niestandardowego z pierwszej pozycji tworzonego paska narzędziowego. Zadaniem tego przycisku ma być wyświetlenie formularza frmKlienci w trybie dodawania nowego rekordu, stąd wywołanie utworzonej wcześniej funkcji ViewFormAdd.

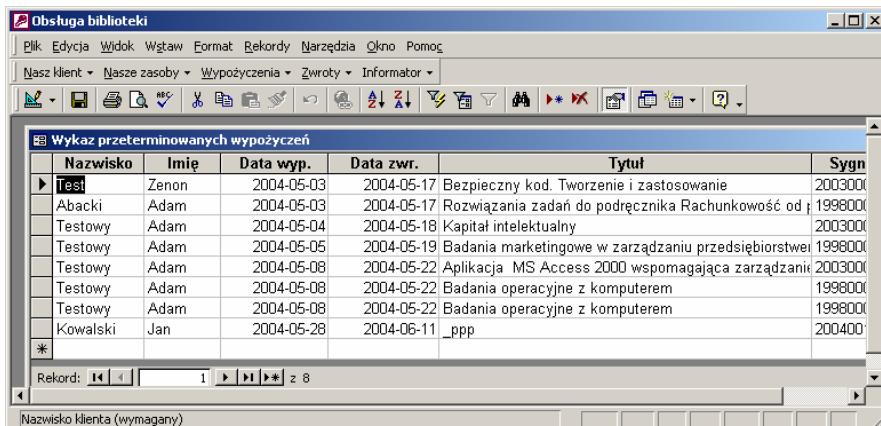


W podobny sposób definiujemy pozostałe przyciski i pasek narzędziowy jest gotowy do użycia. Korzystając z polecenia Autostart w menu Narzędzia możemy zdefiniować sposób otwierania naszej aplikacji.



W pokazanej sytuacji określona została nazwa aplikacji wyświetlaną w pasku tytułowym oraz nazwa formularza wyświetlanego na ekranie. W tym przypadku zdecydowaliśmy się na formularz frmWykazPrzeterminowanych, jego zadaniem jest przypominanie pracownikowi obsługi biblioteki o konieczności odzyskania pozycji, które dawno już powinny być zwrócone. W oknie dialogowym polecenia Uruchamianie zostało uaktywnione pole wyboru determinujące ukrycie okna bazy danych, w miarę potrzeby można je ponownie otworzyć korzystając z polecenia Odkryj w menu Okno.

Poniżej widok okna tej aplikacji bezpośrednio po uruchomieniu.



Na tym zakończymy omawianie (wybranych) fragmentów aplikacji „Biblioteka”, na załączonym krążku znajduje się kompletny plik Biblioteka.mdb, polecamy jego analizę.

6. Przykłady aplikacji – obsługa sklepu

Kolejna aplikacja, której istotne fragmenty przedstawimy w tym rozdziale, będzie przeznaczona do zarządzania informacją handlową w przykładowym sklepie zaopatrzenia ogrodniczego. Założymy, że jest to stosunkowo większy sklep na tyle, że jest w nim zbudowana lokalna sieć komputerowa, kilka stanowisk kasowych oraz dział zaopatrzenia. Oznacza to, że musimy naszą aplikację tak zaprojektować, aby mogła funkcjonować w sieci LAN oraz aby była dostępna dla wielu użytkowników jednocześnie.

Konieczne będzie także rozwiążanie problemu zabezpieczenia bazy danych przed niepowołanym dostępem.

6.1. Funkcjonalność

Naszym zadaniem jest zbudowanie aplikacji, która ułatwi realizację następujących zadań:

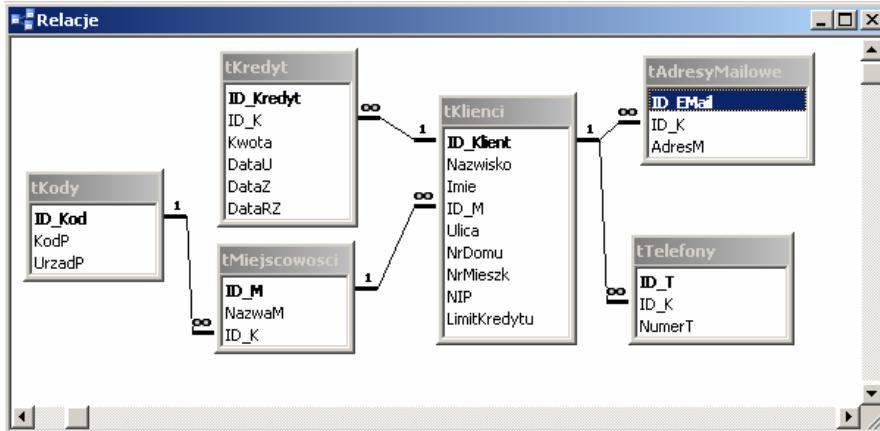
- rejestrację klientów tej firmy wraz z prowadzeniem historii ich kontaktów handlowych,
- obsługę zaopatrzenia firmy w produkty przeznaczone do dalszej sprzedaży,
- obsługę sprzedaży wraz z wystawianiem dokumentów sprzedaży,
- obsługę kredytu handlowego dla klientów firmy,
- raportowanie istotnych zdarzeń z prowadzonej działalności handlowej,
- pracę wielostanowiskową w sieci LAN wraz z odpowiednimi zabezpieczeniami.

6.2. Rejestracja klientów

Podstawowe dane o klientach naszego sklepu będziemy zapisywali w kilku powiązanych z sobą tabelach. Tabelą główną w tym fragmencie projektu będzie tabela `tKlienci` z kluczem głównym typu *autonumer* o nazwie `id_k`. Z tabelą tą współpracują tabele:

- `tMiejscowosci` – tu przechowywane są dane o nazwach miejscowości,
- `tKody` – tabela pomocnicza, dostarcza kody urzędów pocztowych
- `tTelefony` – przechowuje numery telefonów klientów,
- `tAdresyMailowe` – przechowuje adresy mailowe klientów,
- `tKredyt` – przechowuje informacje o kredycie kupieckim.

Niżej pokazane są te tabele wraz z relacjami.



6.2.1. Formularze obsługujące klientów

Formularze obsługujące rejestrację klientów będą budowane w sposób analogiczny jak w poprzedniej aplikacji; będzie to po prostu jeden złożony formularz z dwoma podformularzami pozwalającymi na wprowadzanie numerów telefonów i adresów mailowych.

Formularz rejestracji klienta

Nazwisko:	Abacki								
Imię:	Adam								
Miejscowość:	Rybno - 23-011 Rybno								
Ulica:									
Nr domu:	234	N mieszk:							
NIP:	321-333-23-32								
Telefony									
<table border="1"> <tr> <th>Nr telefonu</th> </tr> <tr> <td>506 345 34 56</td> </tr> <tr> <td>*</td> </tr> </table>		Nr telefonu	506 345 34 56	*	<table border="1"> <tr> <th>Adresy mailowe</th> </tr> <tr> <td>abacki.adam@wp.pl</td> </tr> <tr> <td>*</td> </tr> </table>		Adresy mailowe	abacki.adam@wp.pl	*
Nr telefonu									
506 345 34 56									
*									
Adresy mailowe									
abacki.adam@wp.pl									
*									
Rekord: [◀◀] [◀] [▶] [▶▶] [◀◀◀◀] [▶▶▶▶] z 1									
Rekord: [◀◀] [◀] [▶] [▶▶] [◀◀◀◀] [▶▶▶▶] z 2									

6.3. Obsługa zaopatrzenia

Obsługa zaopatrzenia to w naszej aplikacji jeden z trudniejszych fragmentów, co wynika z faktu dość dużego asortymentu oferowanych produktów. Elementem głównym tego fragmentu aplikacji jest tabela `tZamowienia`, w niej będą rejestrowane wszystkie zamówienia produktów oferowanych później w sklepie ogrodniczym.

tZamowienia : Tabela

	Nazwa pola	Typ danych	Opis
1	ID_Z	Autonumerowanie	automatyczny identyfikator
2	ID_D	Liczba	identyfikator dostawcy
3	DataZ	Data/Godzina	data wystawienia zamówienia
4	DataR	Data/Godzina	data zrealizowania zamówienia

Właściwości pola

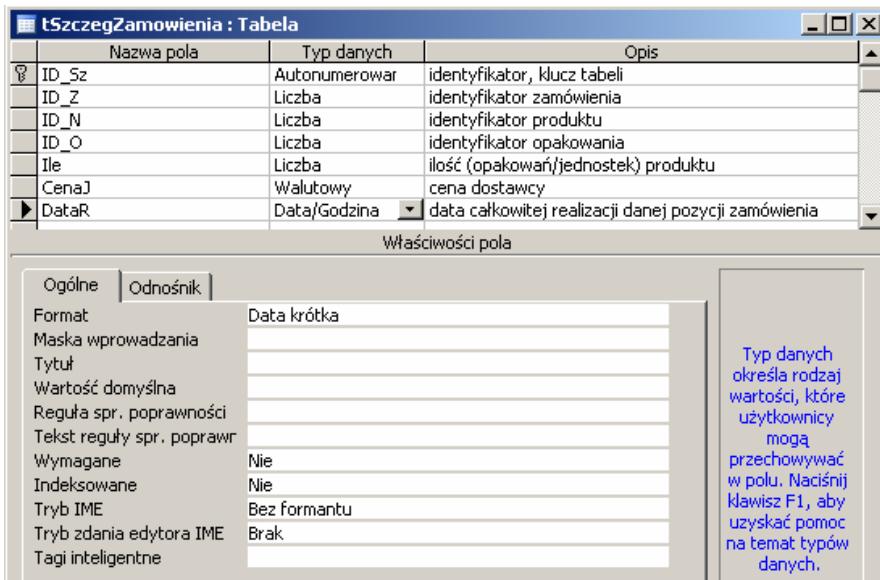
Ogólne	Odnośnik
Format	Data krótka
Maska wprowadzania	
Tytuł	
Wartość domyślna	
Reguła spr. poprawności	
Tekst reguły spr. poprawr	
Wymagane	Nie
Indeksowane	Nie
Tryb IME	Bez formantu
Tryb zdania edytora IME	Brak
Tagi inteligentne	

Opis pola jest opcjonalny.
Pomaga on w określeniu pola i jest wyświetlany na pasku stanu, gdy pole to zostanie wybrane na formularzu.
Naciśnij klawisz F1, aby uzyskać pomoc na temat opisów pól.

Pole DataR będzie przechowywać datę całkowitej realizacji danego zamówienia, stąd właściwość *Wymagane* tego pola jest ustawiona na Nie. Pole DataZ przechowuje datę złożenia zamówienia, stąd jego właściwość *Wartość domyślna* zawiera formułę zwracającą datę bieżącą w formacie rok-miesiąc-dzień:

```
=Format(Now(); "rrrr-mm-dd")
```

Tabela tZamówienia została połączona relacją *jeden-do-wielu* poprzez pole ID_Z z tabelą tSzczegZamówienia, w której będą przechowywane detale złożonego zamówienia. Obejmują one identyfikator nazwy produktu, identyfikator opakowania, ilość zamówionych opakowań (jednostek), cenę żądaną przez dostawcę (z jego oferty) oraz pole DataR, gdzie zapiszemy datę pełnej realizacji danej pozycji zamówienia.

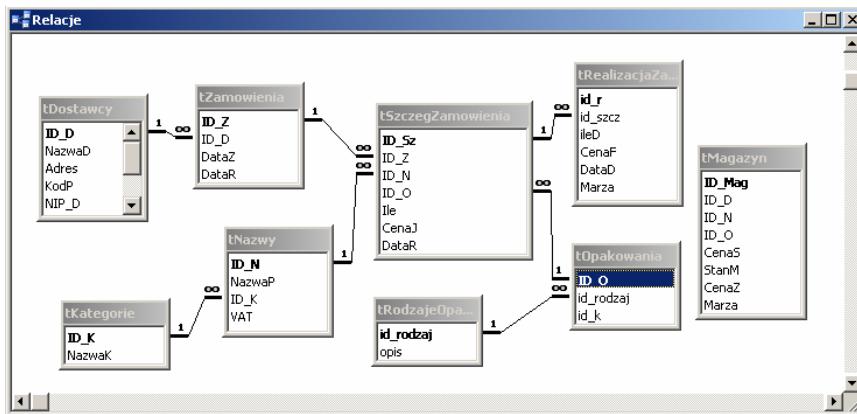


Pole DataR pozostanie puste tak długo, dopóki dana pozycja zamówienia nie zostanie w całości zrealizowana, tym samym będziemy mogli je wykorzystać jako kryterium wyszukania tych zamówień, które nie zostały zrealizowane.

Tabela szczegółów zamówienia współpracuje bezpośrednio lub pośrednio z czterema kolejnymi tabelami precyzującymi zamawiany produkt. Dla poprawnego zbudowania formularzy obsługujących proces składania zamówienia, między innymi z uwagi na zakładany obszerny asortyment produktów, wprowadzono ich podział na kilka czy kilkanaście kategorii. Takie rozwiązanie ma ułatwić wybór produktu poprzez ograniczenie ich listy do danej kategorii.

Podobnie, dla uniknięcia skojarzenia danego produktu z nieadekwatnym opakowaniem zdecydowaliśmy się na wprowadzenie pośredniczącej tabeli *tOpakowania*. Tabela ta wiąże daną kategorię produktu z możliwymi dla niej opakowaniami. Obrazowo chodzi o to, aby nie było możliwości zamówienia np. sekatorów określając ich opakowanie np. w metrach bieżących czy pojemnikach 10-cio litrowych.

Poniżej pokazane są wszystkie te tabele i relacje między nimi, które związane są ze składaniem zamówień. Poza wymienionymi wyżej pokazana jest tabela *tDostawcy* oraz tabela *tRealizacjaZam*. W sumie mamy osiem tabel zabezpieczających odpowiedni przepływ informacji na etapie składania zamówienia jak i rozliczenia jego realizacji.

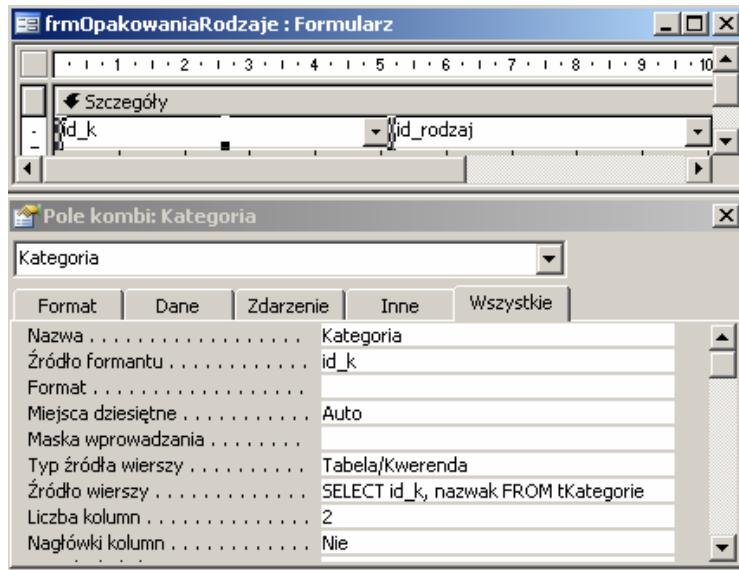


Dodatkowo została pokazana tabela *tMagazyn*, jest ona przeznaczona do pełnego ewidencjonowania ruchu produktów przychodzących (z zamówień) do naszego sklepu jak i wychodzących poprzez sprzedaż. Chwilowo nie są pokazane relacje tej tabeli z innymi, zrobimy to trochę później. Tu została pokazana w tym celu, aby uświadomić Czytelnikowi konieczność rejestracji w tej tabeli **każdej** zrealizowanej dostawy, przy czym rejestracja ta będzie polegała na aktualizacji pola *StanM* dla danego produktu, ale tylko wtedy, gdy jego cena sprzedaży będzie taka sama. Jeżeli nie, to w tabeli *tMagazyn* musi pojawić się nowy rekord. Te, dość skomplikowane sytuacje będziemy rozwiązywać przy pomocy procedur VBA.

6.3.1. Formularze obsługujące składanie zamówień

Przed przygotowaniem właściwego formularza złożonego musimy przygotować formular pozwalający na wprowadzanie i edycję danych do tabeli *tOpakowania*. Z uwagi na dość dużą liczbę rekordów do zapisania w tej tabeli z jednej strony, z drugiej na pewną łatwość przeglądania informacji w niej zapisanych przygotujemy formularz w widoku arkusza danych z dwoma polami kombi pozwalającymi na łatwy wybór kategorii i odpowiedniego opakowania.

Poniżej pokazane jest okno projektu tego formularza z dwoma polami kombi oraz właściwością pierwszego z nich. To pole kombi, o nazwie Kategoria, pobiera dane z tabeli tKategorie poprzez zdefiniowane polecenie Select bezpośrednio we właściwości Źródło wierszy. Rozwiążanie takie jest lepsze, niż pobieranie danych z wcześniej zdefiniowanej kwerendy, ponieważ nie tworzymy fizycznego obiektu bazy danych, jakim jest kwerenda.



W analogiczny sposób definiowane jest źródło danych dla drugiego z pól kombi:

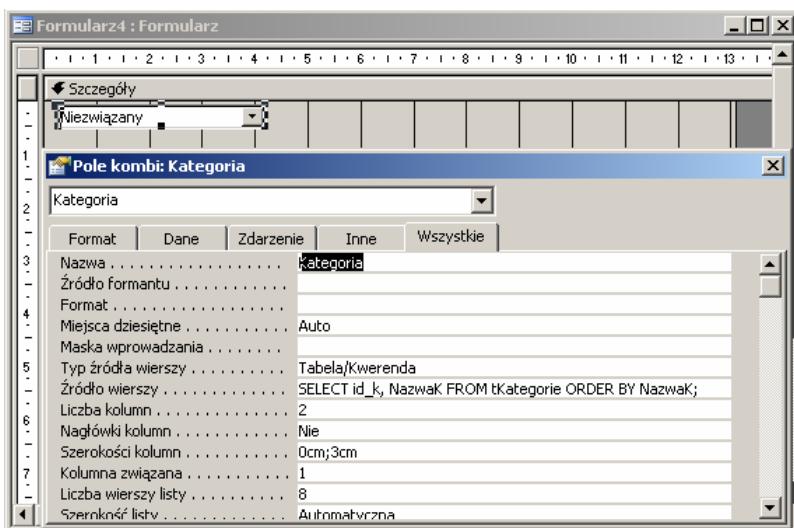
```
SELECT tRodzajeOpakowania.id_rodzaj,
tRodzajeOpakowania.opis FROM tRodzajeOpakowania ORDER BY
tRodzajeOpakowania.opis
```

Poniżej pokazany jest ten formularz w trakcie przeglądania wprowadzonych danych o kategoriach produktu i możliwych opakowaniach.

	Kategoria	Opakowanie/Jednostka
▶	Herbicydy	5 kg
	Herbicydy	20 kg
	Środki grzybobójcze	0,25 litra
	Środki grzybobójcze	0,5 litra
Rekord: [Navigation Buttons] 11 [Next] [Last] z 24		

Kolejny krok jaki powinniśmy podjąć, to zbudowanie podformularza, w którym będziemy mogli z pola kombi wybrać kategorię produktu, a następnie z kolejnych pól kombi jego nazwę oraz rodzaj opakowania. Dalej już w polach tekstowych powinniśmy wprowadzić liczbę opakowań wybranego produktu oraz jego cenę katalogową.

Te oczekiwania powodują, że musimy tak budować pole kombi dla produktów i rodzajów opakowania, aby lista pozycji wyświetlanych w tych kontrolach była zależna od wybranej kategorii. Pozornie sprawa jest prosta. Zobaczmy więc, co się nam uda zbudować. Zaczynamy od otwarcia nowego projektu formularza, dla którego źródłem danych będzie tabela `tSzczegZamowienia`, a wybranym widokiem będzie arkusz danych. Poniżej pokazany jest projekt takiego formularza z pierwszym polem kombi, jego źródłem danych jest odpowiednie polecenie `select`. Proszę zauważać, że właściwość `Źródło formantu` pozostała pusta – po prostu nie ma gdzie zwrócić identyfikatora kategorii. To pole kombi ma nam służyć jedynie do ograniczania listy pozycji w dwóch kolejnych polach.



W kolejnym kroku umieszczamy dwa dalsze pola kombi, pierwsze o nazwie `Produkt` ma zwracać informacje do pola `ID_N`, a drugie o nazwie `Opakowanie` do pola `ID_O`. Pozostaje problem, skąd oba pola kombi mają pobierać dane?

Z pewnością nie może to być żaden wpis na etapie projektowania formularza (w czasie *design time*), bo byłoby to niezgodne z naszym zamierzeniem ograniczenia informacji w obu listach zależnie od tego, jaka kategoria zostanie wybrana w polu kombi `Kategoria`. Jedyne wyjście, to napisanie odpowiedniej procedury uruchamianej po aktualizacji informacji w kombi `Kategoria`.

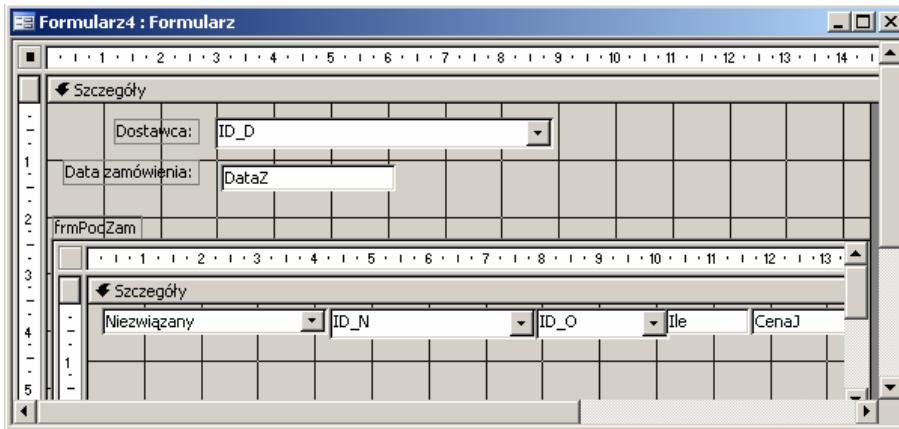
Poniżej pokazany jest pełny zapis tej procedury.

```
Private Sub Kategoria_AfterUpdate()
    Me.Nazwa.RowSource = _
        "SELECT tNazwy.ID_N, tNazwy.NazwaP " & _
        "FROM tKategorie INNER JOIN tNazwy ON tKategorie.ID_K = "
        tNazwy.ID_K " & _
        "WHERE tNazwy.ID_K = " & Me.Kategoria & _
        " order by tNazwy.NazwaP"
    Me.Nazwa.Requery
    '
    Me.opakowanie.RowSource = _
        "SELECT tOpakowania.ID_O, tRodzajeOpakowania.opis " & _
        "FROM tRodzajeOpakowania INNER JOIN tOpakowania ON "
        tRodzajeOpakowania.id_rodzaj = " & _
        "tOpakowania.id_rodzaj WHERE tOpakowania.id_k = " & _
        Me.Kategoria & _
        " order by tRodzajeOpakowania.opis"
    Me.opakowanie.Requery
End Sub
```

Utworzony w ten sposób formularz możemy zapisać pod nazwą frmPodZam, za chwilę wykorzystamy go jako podformularz obsługujący wystawienie zamówienia.

Otwieramy nowy formularz bazujący na tabeli tZamówienia i umieszczamy w nim dwie kontrolki: pole kombi pozwalające na wybór dostawcy oraz pole daty zamówienia.

Na powierzchni formularza umieszczamy także utworzony wcześniej podformularz frmPodZam, w sumie powinniśmy otrzymać formularz podobny do pokazanego niżej.



Przed jego uruchomieniem musimy jeszcze przygotować źródło danych dla kombi cboDostawca poprzez przypisanie właściwości *Źródło wiersza* polecenia select.

```
SELECT tDostawcy.ID_D, tDostawcy.NazwaD FROM tDostawcy
ORDER BY tDostawcy.NazwaD
```

Przed uruchomieniem tak utworzonego formularza złożonego ustawiamy jeszcze jego właściwość *Wprowadzanie danych* na Tak. Po otwarciu formularz powinien wyglądać bardzo obiecująco, zobaczymy czy jego funkcjonowanie będzie zgodne z naszymi oczekiwaniami.

Po jego uruchomieniu bez żadnych problemów możemy wybrać dostawcę w polu kombi, widzimy także, że data zamówienia została automatycznie ustawiona na datę bieżącą, możemy więc przejść do podformularza i wybrać pierwszą kategorię.

The screenshot shows a Windows application window titled "Formularz4 : Formularz". At the top left, there is a dropdown menu labeled "Dostawca" with the value "Hurtownia ogrodnicza "Spomasz"" selected. Below it is a text input field labeled "Data zamówienia" containing the date "2005-10-28". A sub-form titled "frmPodZam" is displayed below these fields. It contains a table with columns: Kategoria, Nazwa, Opakowanie, Ile, and CenaJ. The first row shows "Herbicydy" in the Kategoria column, with a dropdown menu open over it showing "Goal" and "Roundap". The bottom of the sub-form has two sets of navigation buttons: "Rekord:" followed by arrows and numbers, and "Rekord:" followed by arrows and numbers.

Po przejściu do następnego pola kombi możemy wybrać nazwę produktu z listy ograniczonej wybraną kategorią. Po wyborze jednego z zaproponowanych produktów czeka nas jednak niespodzianka – pojawia się już jakby „nowy” rekord z wybraną tą samą kategorią co w rekordzie bieżącym.

This screenshot shows the same application window as before, but the sub-form "frmPodZam" now displays a different state. In the "Nazwa" column of the table, the previously selected "Goal" has been replaced by a new record "Goal" with a different internal identifier. The rest of the table and the navigation controls at the bottom remain the same.

Niestety, to nie jest koniec nieoczekiwanych niespodzianek. Kolejna czeka nas po dokonaniu wprowadzania danych w pierwszym rekordzie podformularza i po przejściu do kolejnego rekordu. Dla ilustracji zamówimy jeszcze jeden herbicyd, otrzymując przykładowo taką sytuację jak niżej pokazana.

The screenshot shows a Windows application window titled "Formularz4 : Formular". At the top, there are two input fields: "Dostawca:" with the value "Hurtownia ogrodnicza \"Spomasz\"", and "Data zamówienia:" with the value "2005-10-28". Below these is a table titled "frmPodZam" containing three rows of data:

Kategoria	Nazwa	Opakowanie	Ile	CenaJ
Herbicydy	Goal	1 litr	12	47,50 zł
Herbicydy	Roundap	20 kg	55	189,99 zł
▶ Herbicydy				

At the bottom of the table, there is a navigation bar labeled "Rekord:" with buttons for navigating between records. The current record is highlighted with a gray background.

Powiedzmy, że chcemy teraz zamówić np. jakieś środki grzybobójcze, musimy więc zmienić kategorię w polu Kategoria. Po wyborze „Środków grzybobójczych” czeka nas jednak niespodzianka – wszystkie wcześniej wprowadzone kategorie zostały zmienione, w konsekwencji zniknęły nazwy produktów i opakowań, wprawdzie tylko w formularzu, ale to wystarczy, aby uznać, że tak przygotowany formularz nie nadaje się do pracy!

This screenshot shows the same application window after changing the category. The table "frmPodZam" now contains three rows of fungicides:

Kategoria	Nazwa	Opakowanie	Ile	CenaJ
Środki grzybobójcz			12	47,50 zł
Środki grzybobójcz			55	189,99 zł
▶ Środki grzybobójcz				

The navigation bar at the bottom remains the same, showing record 1 of 3.

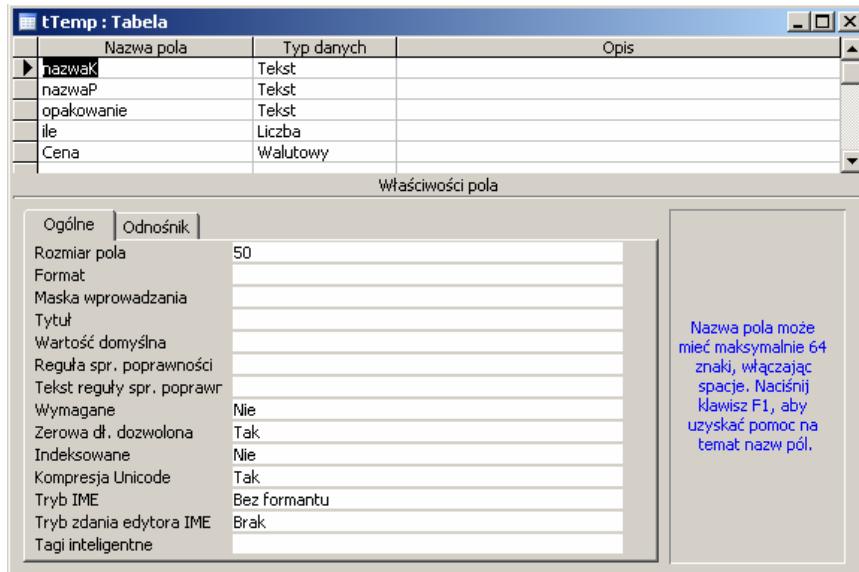
Mankament polegający na tym, że we wszystkich rekordach pola Kategoria mamy ostatnio wybraną kategorię możemy usunąć stosunkowo łatwo, wystarczy do tabeli tSzczegZamówienia dodać pole id_k do przechowywania identyfikatora kategorii

i związać z nim pole kombi Kategoria. Niestety, nie jest to skuteczne na drugą „niedogodność” tak zaprojektowanego formularza, w dalszym ciągu zmiana kategorii będzie powodować znikanie nazw produktów i nazw opakowań w tych wcześniej wprowadzonych rekordach, które dotyczą innej, niż ostatnio wybrana kategoria. Oznacza to, że musimy zupełnie inaczej rozwiązać problem budowy tego formularza.

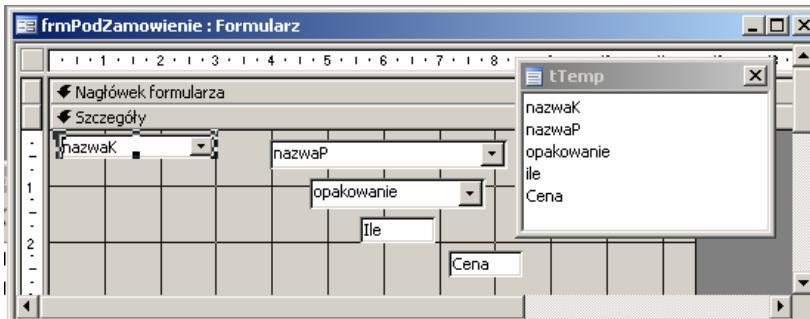
Jednym z możliwych rozwiązań jest zastosowanie tymczasowej tabeli do gromadzenia informacji z podformularza, dopiero po zakończeniu definiowania wszystkich zamawianych produktów nastąpi przeniesienie wprowadzonych informacji do właściwej tabeli tSzczegZamowienia.

Rozwiązanie takie postawi przed nami kilka dodatkowych wyzwań. Z uwagi na brak powiązań (relacji) tej pomocniczej tabeli z takimi tabelami jak tKategoria, tNazwy czy tOpakowania musimy przechowywać w niej jawne informacje, a nie ich identyfikatory. Z kolei do tabeli szczegółów zamówienia musimy zapisywać identyfikatory (nazwy, opakowania), pojawi się więc problem gromadzenia gdzieś tych identyfikatorów przed ich zapisaniem.

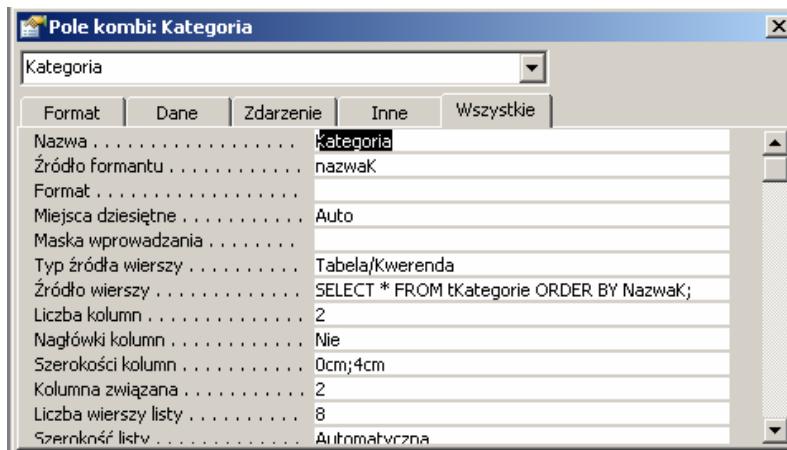
Zaczniemy od zaprojektowania pomocniczej tabeli o nazwie tTemp. Tabela ta będzie zawierać pokazane niżej pola, a jej cechą charakterystyczną jest brak zdefiniowanego klucza. Rozwiązanie takie jest do przyjęcia w tymczasowej tabeli, łatwo także zauważyc, że tabela ta jest zaprojektowana niezgodnie z zasadami normalizacji, ale w tym przypadku jest to dopuszczalne.



W kolejnym kroku utworzymy formularz bazujący na tej tabeli tymczasowej z trzema polami kombi i dwoma polami tekstowymi, tak jak to pokazano niżej (ułożenie formantów nie jest istotne, bo będzie on wyświetlany jako arkusz danych),



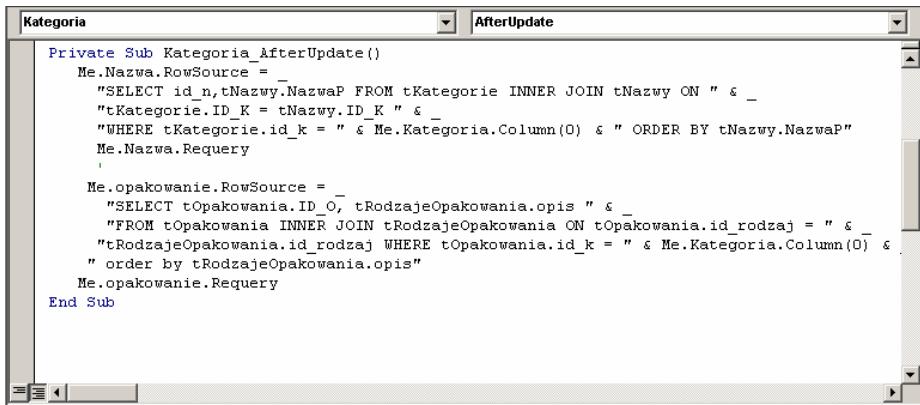
Warto zwrócić uwagę na właściwości pól kombi, poniżej pokazane są właściwości pola Kategoria, a interesujące jest to, że zapytanie zwraca dwa pola: `id_k` i nazwak, a pole kombi zwraca bezpośrednio drugie z nich, czyli nazwę kategorii.



Jak wiemy będzie nam także potrzebny identyfikator kategorii, jest on także zwracany przez tak zdefiniowane pole kombi, choć w sposób niejawny. Odzyskamy go pisząc odpowiednią procedurę VBA.

W podobny sposób definiowane są właściwości dwóch pozostałych pól kombi (zawsze zwracamy dwa pola, ale kolumną związaną jest kolumna zwracająca nazwę produktu czy opakowania, a nie identyfikator). Zasadnicza różnica dotyczy źródła danych dla tych pól – będziemy je tworzyć dynamicznie w momencie wyboru kategorii.

Poniżej pokazane jest okno procedury zdarzeniowej uruchamianej po aktualizacji pola Kategoria. Proszę zwrócić uwagę na sposób odwołania się do pola Kategoria w celu odzyskania identyfikatora kategorii. Jak pamiętamy instrukcja select zwracała dwa pola (id_k, nazwak), te pola przechowywane są w kolumnach źródła danych, a numeracja kolumn zaczyna się od zera. Tym samym identyfikator id_k jest przechowywany w kolumnie zerowej, a nazwak w kolumnie pierwszej. Można się do tych informacji odwołać za pomocą właściwości *Column(index)* pola kombi i tak jest to rozwiązane poniżej.



```

Kategoria
AfterUpdate

Private Sub Kategoria_AfterUpdate()
    Me.Nazwa.RowSource =
        "SELECT id_n,tNazwy.NazwaP FROM tKategorie INNER JOIN tNazwy ON " &
        "tKategorie.ID_K = tNazwy.ID_K " &
        "WHERE tKategorie.id_k = " & Me.Kategoria.Column(0) & " ORDER BY tNazwy.NazwaP"
    Me.Nazwa.Requery

    Me.opakowanie.RowSource =
        "SELECT tOpakowania.ID_O, tRodzajeOpakowania.opis " &
        "FROM tOpakowania INNER JOIN tRodzajeOpakowania ON tOpakowania.id_rodzaj = " &
        "tRodzajeOpakowania.id_rodzaj WHERE tOpakowania.id_k = " & Me.Kategoria.Column(0) &
        " order by tRodzajeOpakowania.opis"
    Me.opakowanie.Requery
End Sub

```

Po przypisaniu nowego źródła danych w zdarzeniu *AfterUpdate* trzeba odświeżyć to źródło za pomocą metody *Requery* pola kombi.

Musimy jeszcze rozwiązać problem gromadzenia identyfikatorów nazw produktów, nazw opakowań oraz ilości i cen produktów przed ich zapisaniem do właściwej tabeli szczegółów zamówienia. Jednym z możliwych rozwiązań jest zadeklarowanie kilku zmiennych macierzowych i rejestrowanie w nich potrzebnych danych w zdarzeniu *BeforeUpdate* (*Przed aktualizacją*) tego formularza.

Zaczniemy od wstawienia w edytorze VBA do naszego projektu nowego modułu, a następnie zadeklarujemy w nim pokazane niżej zmienne.

```

Public x(100, 1) As Long, y(100) As Single, _
c(100) As Currency
Public ileR As Integer

```

Zmienna macierzowa *x(100, 1)* przeznaczona jest do przechowywania identyfikatorów nazw produktów i nazw opakowań, stąd jest zadeklarowana jako liczba całkowita dłuża (*Long*). Zmienna *y(100)* ma przechowywać informacje o ilościach produktów, stąd typ danych pojedyncza precyzja (*Single*). Zmienna *c(100)* została przeznaczona na

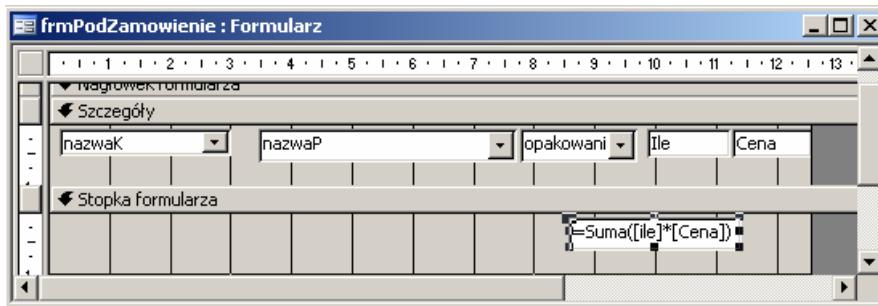
gromadzenie informacji o cenach produktów, stąd typ walutowy (*Currency*). Zmienna skalarana *ileR* ma przechowywać liczbę rekordów, stąd typ *Integer*.

Pokazana niżej procedura zapisuje do tych zmiennych odpowiednie informacje bezpośrednio przed przejściem do kolejnego rekordu w budowanym przez nas formularzu.

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    ileR = Me.CurrentRecord - 1
    x(ileR, 0) = Me.Nazwa.Column(0)
    x(ileR, 1) = Me.opakowanie.Column(0)
    y(ileR) = Me.Ile
    c(ileR) = Me.Cena
End Sub
```

Do uzyskania numeru bieżącego rekordu została wykorzystana właściwość *CurrentRecord* formularza, do którego odwołaliśmy się poprzez uniwersalne słowo kluczowe *Me*.

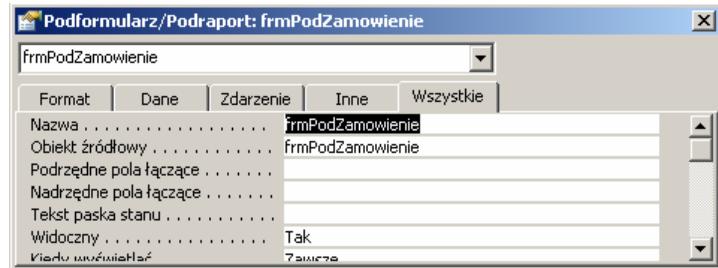
Tworzony przez nas formularz będzie wykorzystywany jako podformularz w formularzu złożonym, stąd jego właściwość *Widok domyślny* ustawiona na Arkusz danych. W tym widoku nie jest pokazywany ani nagłówek formularza, ani jego stopka. Mimo tego my zadeklarujemy pokazywanie nagłówka/stopki formularza po to, aby w stopce formularza umieścić pole tekstowe (tu o nazwie *txtRazem*), którego zadaniem będzie obliczanie sumarycznej wartości netto zamawianych produktów. Do pola tego odwołamy się z formularza głównego, ale żeby to było możliwe, to trzeba było dodać je w sekcji stopki właśnie. Źródłem danych dla tego pola jest formula =*Suma ([ile]*[cena])*.



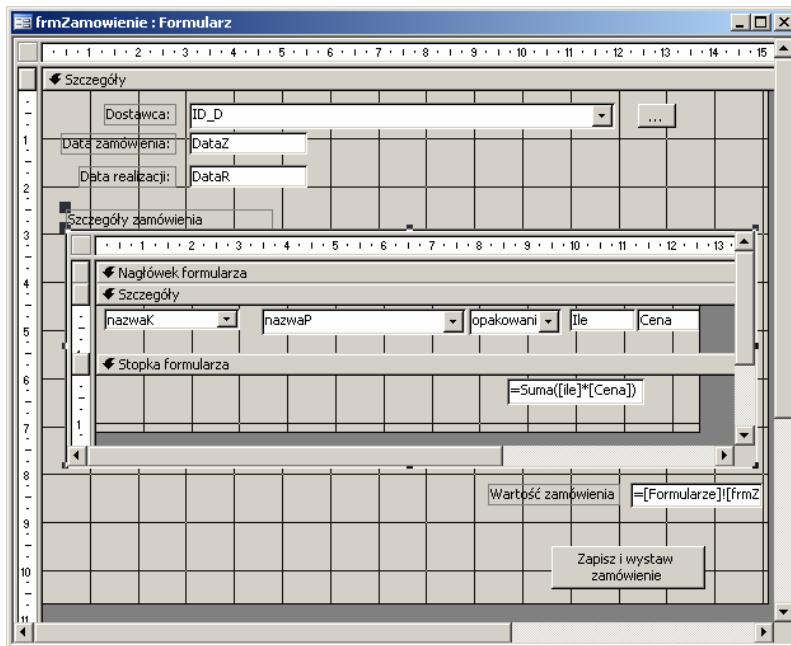
Projektowanie formularza jest zakończone, możemy zapisać go do dalszego wykorzystania pod nazwą *frmPodZamowienie*.

Kolejny krok to projekt formularza głównego, możemy zrobić go od nowa, możemy wykorzystać wcześniej budowany (pokazany wcześniej Formularz4) po usunięciu z projektu nieudanego podformularza *frmPodZam*.

Do projektu dodajemy podformularz wskazując kreatorowi do wykorzystania utworzony wcześniej formularz frmPodZamowienie. Proszę pamiętać o tym, że formularz główny nie może być związyany z podformularzem, po prostu **nie ma żadnego wspólnego pola** między tabelą tZamowienia a tabelą tTemp.



Jak widzimy w pokazanym oknie właściwości podformularza obie właściwości *Podrzędne pola łączące* i *Nadrzędne pola łączące* nie są zdefiniowane, co standardowo byłoby niedopuszczalne, ale w naszym przypadku – z uwagi na konkretne zadanie do wykonania przez budowany formularz – jest jak najbardziej poprawne. Dla jasności, my budujemy formularz do wprowadzania danych do tabel tZamowienie i tTemp, a nie do ich przeglądania.



W pokazanym projekcie formularza głównego widać jeszcze dwie dodatkowe kontrolki. Jedną z nich jest pole tekstowe, jego zadaniem jest odwołanie się do wcześniej omawianego pola txtRazem w celu wyświetlenia aktualnej wartości netto zamówienia. Jako źródło danych dla tego pola podana jest dość złożona formula:

```
= [Formularze] ! [frmZamowienie] ! [frmPodZamowienie] . [Form] !
    [txtRazem]
```

Drugą kontrolką jest przycisk polecenia, jego zadaniem jest uruchomienie procedur VBA realizujących kilka ważnych zadań:

- Zapisanie wprowadzonych szczegółów zamówienia do tabeli tSzczegZamowienia,
- Wydrukowanie formularza zamówienia,
- Skasowanie danych z pomocniczej tabeli tTemp.

Poniżej widok tak przygotowanego formularza w trakcie tworzenia zamówienia na dostawę przykładowych produktów. W porównaniu z poprzednio przygotowanym formularzem tym razem wszystko jest poprawnie zrobione, a sam formularz jest wygodny w użyciu.

Kategoria	Nazwa	Opakowanie	Ile	Cena
Herbicydy	Goal	1,5 kg	12	34,00 zł
Herbicydy	Roundap	10 litrów	39	123,00 zł
Środki grzybobójcze	Miedzian	3 kg	25	45,00 zł
Środki grzybobójcze	Sylit	1 kg	145	24,00 zł
Herbicydy	Roundap	20 litrów	40	210,00 zł
			0	0,00 zł

Rekord: [Navigation Buttons] 6 [Navigation Buttons] z 6 Wartość zamówienia: 18 210,00 zł

Zapisz i wystaw zamówienie

Jak wcześniej wspominaliśmy, przycisk polecenia „Zapisz i wystaw zamówienie” ma uruchomić procedurę, której zadaniem jest zapisanie szczegółów zamówienia w tabeli tSzczegZamowienia. Zrobimy to za pomocą polecenia Insert into działającego w pętli z licznikiem.

```
Private Sub cmdZapisz_Click()
    Dim con As ADODB.Connection, txt As String
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    Dim j As Integer
    For j = 0 To ileR
        txt = "insert into tSzczegZamowienia (id_z, " & _
            "id_n, id_o, ile, cenaj) values (" & _
            Me.ID_Z & ", " & x(j, 0) & ", " & x(j, 1) & ", " & _
            y(j) & ", " & c(j) & ")"
        con.Execute txt
    Next j
    ' tu będą dalsze instrukcje związane z wystawieniem
    ' zamówienia
    con.Close
    set con = Nothing
    Me.Close
End Sub
```

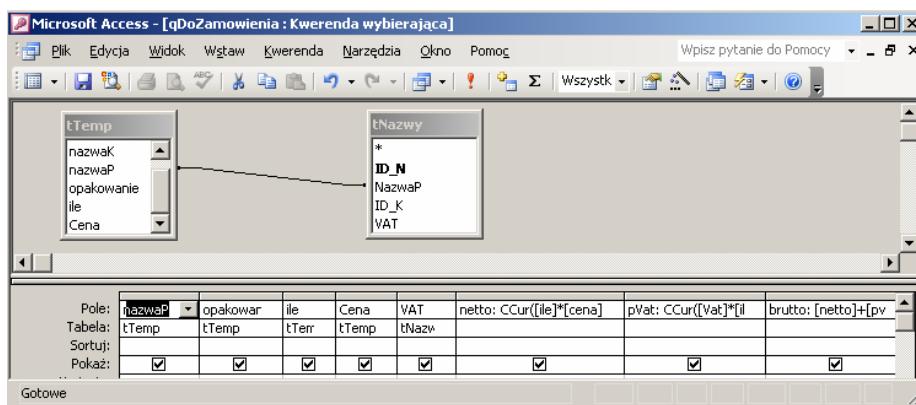
Poniżej pokazany jest fragment tabeli tSzczegZamowienia z wprowadzonymi powyższą procedurą informacjami odpowiadającymi szczegółom zamówień widocznych w formularzu frmZamowienia pokazany na poprzedniej stronie.

tSzczegZamowienia : Tabela						
ID_Sz	ID_Z	ID_N	ID_O	Ile	CenaJ	DataR
51	19	3	9	12	34,00 zł	
52	19	1	5	39	123,00 zł	
53	19	2	20	25	45,00 zł	
54	19	4	18	145	24,00 zł	
55	19	1	6	40	210,00 zł	

Przejdziemy teraz do przygotowania druku zamówienia, zrealizujemy to poprzez przygotowanie raportu złożonego.

6.3.2. Przygotowanie wydruku zamówienia

Zacznijmy od przygotowania raportu drukującego szczegóły zamówienia, raport ten umieścimy później jako podraport w raporcie złożonym odpowiedzialnym za wydrukowanie kompletnego zamówienia. Generalnie raport ten będzie oparty na danych z tabeli pomocniczej tTemp, ale będą także potrzebne dodatkowe dane (np. stawka VAT) jak i pewne obliczenia. W tej sytuacji przygotujemy odpowiednią kwerendę, której projekt pokazany jest poniżej.

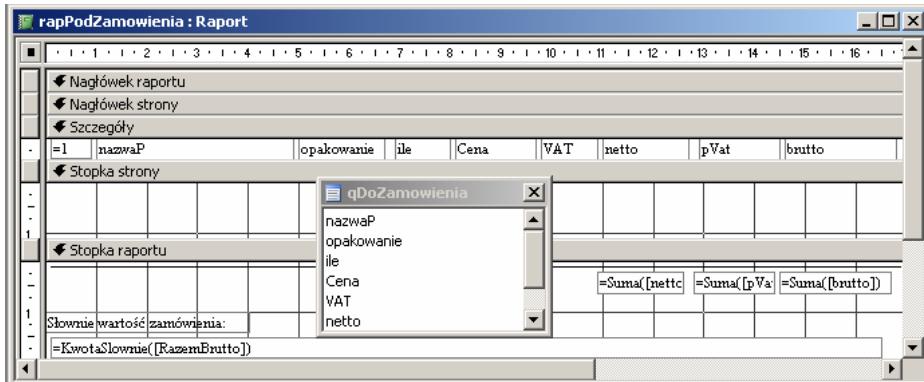


Kwerenda ta zwraca informacje z tabel tTemp i tNazwa, które na potrzeby tego zapytania zostały połączone relacją poprzez pola tekstowe opisujące nazwy produktów z obu tabel. Było to niezbędne dla pobrania stawki podatku VAT z tabeli tNazwa. Dodatkowo kwerenda zawiera trzy pola wyliczane, pole netto wylicza wartość netto danej pozycji zamówienia, przy czym zastosowana została funkcja systemowa Ccur konwertująca wynik obliczeń do formatu waluty. Analogiczna funkcja została użyta dla pola pVat w celu wyznaczenia kwoty podatku VAT. W przypadku pola brutto zwracającego wartość brutto danej pozycji użycie funkcji konwertującej nie było już potrzebne.

Po przygotowaniu tej kwerendy i jej zapisaniu pod nazwą np. qDoZamowienia można przejść do budowy pierwszego z raportów.

Po przejściu do obiektu *Raporty* w oknie bazy danych otwieramy nowy raport oparty o zdefiniowaną wyżej kwerendę. W projekcie nowego, pustego jeszcze raportu pokazujemy zarówno nagłówek/stopkę raportu, przy czym zamkamy nagłówek raportu – po prostu nie będzie nam potrzebny. Z kolei stopka raportu jest nam potrzebna, ponieważ umieścimy w niej pola tekstowe sumujące wartość zamówienia w pozycjach netto, kwota VAT i brutto. Tu także umieścimy etykietę i pole tekstowe zwracające słownie wartość brutto zamówienia.

Poniżej pokazany jest projekt tego raportu, w sekcji *Szczegóły* umieszczone są pola tekstowe zwracające potrzebne informacje ze źródła danych. Dodatkowo na początku zostało umieszczone pole tekstowe zwracające kolejny numer pozycji zamówienia.



W tym raporcie nie ma etykiet opisujących poszczególne kolumny danych, te opisy umieścimy w raporcie głównym w sekcji nagłówka strony.

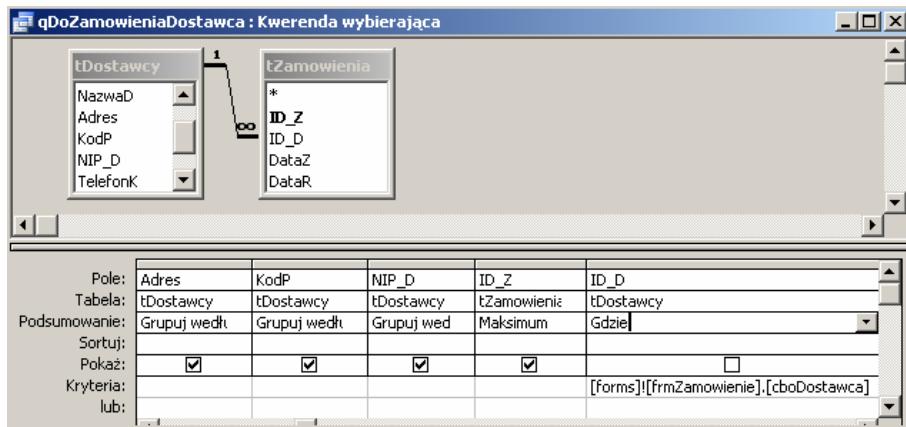
W lewym dolnym narożniku zostało umieszczone pole tekstowe zwracające kwotę słownie wartości brutto zamówienia. Do jej wyprowadzenia została wykorzystana funkcja *KwotaSłownie*⁴. Kod tej funkcji został skopiowany do modułu Module1 w naszej aplikacji.

Można już przejść do budowania raportu głównego, z tym, że zaczynamy od utworzenia odpowiedniego źródła danych opisującego dostawcę, do którego nasze zamówienie zostanie skierowane. Wszystkie potrzebne dane są przechowywane w tabeli *tDostawcy*, ale będzie nam potrzebny jeszcze identyfikator zamówienia (pole *id_z*) po to, aby jakość zdefiniować numer zamówienia. Jednym z możliwych rozwiązań będzie połączenie identyfikatora zamówienia z rokiem bieżącym poprzedzone odpowiednim tekstem.

Jednym z łatwiejszych rozwiązań będzie przygotowanie prostej kwerendy parametrycznej, do której przekażemy wartość *id_d* (identyfikator dostawcy) wybranego w polu kombi *cboDostawca* formularza *frmZamowienie*. Ponieważ kwerenda ma zwrócić dane dostawcy plus ostatni numer jego zamówienia, to dla uzyskania takich właśnie informacji zastosujemy grupowanie rekordów, a dla pola *id_z* zastosujemy funkcję *Maximum* (co nam zwróci największy numer zamówienia dla danego dostawcy).

Projekt takiej kwerendy pokazany jest niżej

⁴ Opisana w pozycji *Makropolecenia i aplikacje VBA w MS Excel*



Poprawna praca tej kwerendy wymaga, aby skorzystać z menu *Kwerenda/Parametry* w celu jednoznacznego zdefiniowania typu zwracanej informacji w kryteriach pola *ID_D*.

Mając gotowe źródło danych dla raportu głównego można przystąpić do jego budowy. Zaczynamy tradycyjnie od otwarcia nowego raportu w widoku projektu, przy czym źródłem danych jest omówiona wyżej kwerenda. Korzystając z menu *Widok* pokazujemy nagłówek/stopkę zarówno raportu jak i strony.

W nagłówku raportu o dość znacznej wysokości umieścimy w lewym narożniku umowne dane naszej firmy podając jej nazwę, dane adresowe oraz numer NIP. W prawym górnym narożniku umieścimy pole tekstowe, gdzie we właściwości *Źródło formantu* wpiszemy np. taką formułę:

```
= "Rabno Duże, dn. " & Date()
```

W efekcie będziemy mieć nazwę miejscowości wraz z aktualną datą wystawienia zamówienia.

Trochę niżej pod polem daty umieszczaćmy pole ze źródła danych opisujące dostawcę, a więc pola *NazwaD*, *Adres*, *KodP*. Pola te muszą być odpowiednio ustawione i sformatowane, w naszej aplikacji do całego raportu użyto czcionki Times New Roman CE w rozmiarze podstawowym 10 pkt. Pewne wybrane informacje zostały sformatowane na 12 pkt z pogrubieniem czcionki.

Po części adresowej umieszczaćmy kolejne pole tekstowe, gdzie jako źródło formantu wpiszemy formułę:

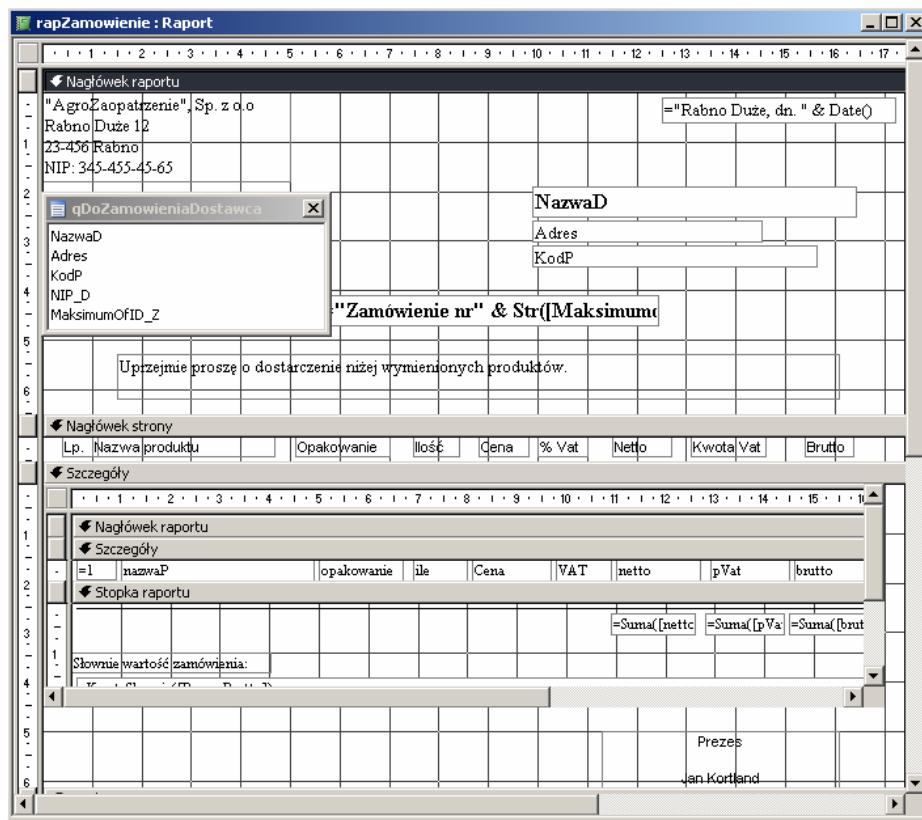
```
= "Zamówienie nr" & Str([MaksimumofID_Z]) & _
"/" & Str(Year(Now()))
```

W efekcie otrzymamy elegancki tytuł dokumentu z odpowiednim numerem porządkowym.

W nagłówku raportu powinniśmy jeszcze dopisać jakiś ogólny tekst odwołujący się do pozycji wyszczególnionych w podraporcie.

W nagłówku strony musimy teraz umieścić serię etykiet opisujących poszczególne kolumny danych zwracanych przez podformularz, ich pozycjonowanie może być wykonane także metodą kolejnych prób. Po ich wstawieniu została dodana jeszcze linia pozioma oddzielająca etykiety od danych podformularza.

Przechodzimy teraz do sekcji szczegółów i tu umieszczamy utworzony wcześniej podraport, najlepiej bez etykiety z jego nazwą i z przezroczystym obramowaniem. Pod obiektem podformularza możemy dodać jeszcze etykietę z nazwiskiem osoby odpowiedzialnej za wystawienie zamówienia. W sumie nasz projekt powinien wyglądać tak, jak pokazany niżej.



W niewidocznej na pokazanym rysunku sekcji stopki strony zostało umieszczone pole tekstowe zwracające numer strony zamówienia poprzez podanie jako źródło danych formuły:

= "Strona " & [Page] & " z " & [Page]

Uzyskany efekt końcowy pokazany jest poniżej, wszystko co zostało do zrobienia to wydrukowanie tego raportu na drukarce i wysłanie do dostawcy.

"AgroZaopatrzenie", Sp. z o.o.
Rabno Duże 12
23-456 Rabno
NIP: 345-455-45-65

Rabno Duże, dn. 2005-10-29

Hurtownia ogrodnicza "Spomasz"
Sternowo 13
65-456 Sternowo

Zamówienie nr 24/ 2005

Uprzejmie proszę o dostarczenie niżej wymienionych produktów.

Lp.	Nazwa produktu	Opakowanie	Ilość	Cena	% Vat	Netto	Kwota Vat	Brutto
1	Goal	1,5 kg	12	34,00 zł	7,00%	408,00 zł	28,56 zł	436,56 zł
2	Roundap	10 litrów	39	123,00 zł	7,00%	4 797,00 zł	335,79 zł	5 132,79 zł
3	Miedzian	3 kg	25	45,00 zł	7,00%	1 125,00 zł	78,75 zł	1 203,75 zł
4	Sylit	1 kg	145	24,00 zł	7,00%	3 480,00 zł	243,60 zł	3 723,60 zł
5	Roundap	20 litrów	40	210,00 zł	7,00%	8 400,00 zł	588,00 zł	8 988,00 zł
						18 210,00 zł	1 274,70 zł	19 484,70 zł

Słownie wartość zamówienia:
dziewiętnaście tysięcy czterysta osiemdziesiąt cztery złote i siedemdziesiąt groszy

Prezes

Jan Kortland

Po wydrukowaniu raportu `rapZamowienie` musi zajść jeszcze jedno zdarzenie polegające na skasowaniu rekordów z tabeli `tTemp`. Wymaga to zmodyfikowania procedury obsługującej przycisk polecenia „Zapisz i wystaw zamówienie” do pokazanej niżej postaci.

```
Private Sub cmdZapisz_Click()
    Dim con As ADODB.Connection, txt As String
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    Dim j As Integer
    For j = 0 To ileR
        txt = "insert into tSzczegZamowienia (id_z, " & _
            "id_n, id_o, ile, cenaj) values (" & _
            Me.ID_Z & ", " & x(j, 0) & ", " & x(j, 1) & _
            ", " & y(j) & ", " & c(j) & ")"
        con.Execute txt
    Next j
    ' pokazanie raportu Zamówienia
    DoCmd.OpenReport "rapZamowienie", acViewPreview
    ' usunięcie danych z tabeli pomocniczej
    txt = "delete from tTemp"
    con.Execute txt
    ' sprzątanie po sobie
    con.Close
    Set con = Nothing
    DoCmd.Close acForm, "frmZamowienie", acSaveNo
End Sub
```

6.3.3. Dodatkowe usprawnienia formularza `frmZamowienie`

W praktyce, przy codziennym wykorzystywaniu formularza `frmZamowienie` może okazać się, że formularz ten powinien mieć wbudowany jakiś mechanizm pozwalający na usunięcie błędnie wprowadzonego rekordu w szczegółach zamówienia (w podformularzu). Spróbujmy teraz dokonać takich modyfikacji, aby było to możliwe.

Z faktu, że pracując w formularzu mamy zamiar usunąć wskazany rekord w podformularzu wynikają dość poważne komplikacje. Po pierwsze musimy założyć, że będziemy usuwać ten rekord, który albo jest zaznaczony, albo ma fokus (jest bieżącym rekordem). Trudność polega na tym, że stosowna procedura usuwająca musi być powiązana z przyciskiem umieszczonym w formularzu głównym, a w takiej sytuacji nie ma bezpośredniego odwołania się do właściwości innego formularza z pytaniem, jaki jest rekord bieżący. Rozwiążemy ten problem poprzez zdefiniowanie publicznej zmiennej `intRekord` typu `integer` (w module ogólnym) i będziemy ją przypisywać numer bieżącego rekordu w procedurze zdarzeniowej *Przy bieżącym* podformularza.

```
Private Sub Form_Current()
    intRekord = Me.CurrentRecord
End Sub
```

Wartość zmiennej intRekord może być już odczytywana przez procedurę utworzoną w module formularza frmZamowienie. W instrukcji Delete wykorzystamy tę zmienną do wskazania rekordu, który mamy zamiar usunąć z tabeli tTemp, ale będzie to możliwe dopiero wtedy, gdy tabela ta będzie zawierała pole określające numer danego rekordu.

Dodanie takiego pola do projektu tabeli tTemp jest oczywiście proste, niech będzie to pole o nazwie pozycja typu *liczba całkowita*. Trochę trudniejsze jest spowodowanie, aby pole to otrzymywało kolejne liczby naturalne w momencie rejestrowania danej pozycji zamówienia. Rozwiązaniem będzie modyfikacja procedury BeforeUpdate formularza frmPodZamowienie do pokazanej niżej postaci.

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    ileR = Me.CurrentRecord - 1
    x(ileR, 0) = Me.Nazwa.Column(0)
    x(ileR, 1) = Me.opakowanie.Column(0)
    y(ileR) = Me.Ile
    c(ileR) = Me.Cena
    ' to jest modyfikacja pola pozycja
    Me.pozycja = Me.CurrentRecord
End Sub
```

Warto zauważyć, że samo pole pozycja nie musi być jawnie dodane do formularza, aby otrzymywać bieżący numer rekordu.

Możemy już zmodyfikować projekt formularza frmZamowienie poprzez umieszczenie w lewym dolnym narożniku przycisku polecenia o nazwie cmdUsunRekord i podobnym opisie. Z przyciskiem tym zwiążemy procedurę zdarzeniową pokazaną niżej.

```
Private Sub cmdUsunRekord_Click()
    Dim con As ADODB.Connection, txt As String, _
        rst As ADODB.Recordset
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    txt = "delete from tTemp where pozycja = " & intRekord
    con.Execute txt
    'trzeba teraz na nowo ponumerowac pole pozycja
    txt = "select pozycja from tTemp"
    Set rst = New ADODB.Recordset
    rst.Open txt, con, adOpenKeyset, adLockOptimistic
```

```

For i = 1 To rst.RecordCount
    txt = "update tTemp set pozycja = " & i & _
          " where pozycja = " & rst!pozycja
    con.Execute txt
    rst.MoveNext
Next i
rst.Close
con.Close
Set rst = Nothing
Set con = Nothing
Me.Refresh
End Sub

```

W pierwszej części wywoływanie jest polecenie Delete w celu usunięcia bieżącego rekordu z tabeli tTemp, to jest dokładnie ten rekord, który jest wyselekcjonowany w podformularzu lub jest aktywnym rekordem. Jak wcześniej zostało to wyjaśnione numer tego rekordu jest zapisany w zmiennej intRekord.

Po usunięciu rekordu, który nie był ostatnim rekordem następuje zakłócenie kolejności rekordów opisanych w polu pozycja. Jeżeli chcemy zapewnić sobie możliwość usuwania dalszych rekordów, to musimy uporządkować numerację rekordów w tabeli tTemp. Dalsza część pokazanej wyżej procedury zajmuje się tym właśnie zadaniem.

Zaczynamy od utworzenia rekordsetu zwracającego pole pozycja ze wszystkich rekordów tabeli tTemp. Ten dynamiczny zestaw rekordów będzie teraz przeglądany w pętli For Next z wywołaniem polecenia Update w celu przypisania do pola pozycja kolejnej liczby naturalnej. Proszę zwrócić uwagę na warunek zdefiniowany w poleceniu SQL, bez tego warunku aktualizowane byłyby wszystkie rekordy, a nie każdy z nich oddziennie.

Po zakończeniu pracy pętli w kolejnych czterech instrukcjach „sprzątamy po sobie” zamykając zmienne rst i con, a następnie usuwając te zmienne z pamięci komputera. Zadaniem ostatniej instrukcji jest odświeżenie formularza frmZamowienie.

Przetestowanie funkcjonowania przycisku „Usuń rekord” pozostawiamy naszemu Czytelnikowi.

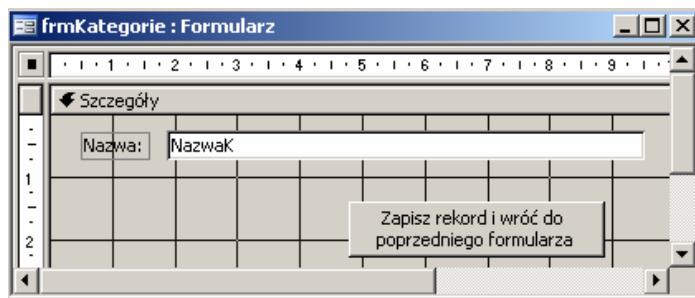
W trakcie definiowania zamówienia może się okazać, że w bazie nie mamy odpowiedniej kategorii produktu, albo produktu w ramach danej kategorii, albo nie mamy przyporządkowanego rodzaju opakowania dla danej kategorii, być może w ogóle nie mamy potrzebnego rodzaju opakowania. Rozwiążaniem takiej sytuacji będzie stworzenie możliwości dodania niejako w „biegu” potrzebnej kategorii, nazwy produktu czy opakowania. Zajmiemy się teraz stworzeniem odpowiednich formularzy oraz procedur je wywołujących i obsługujących.

Zaczniemy od umieszczenia w dolnej części formularza frmZamowienie trzech przycisków poleceń o nazwach cmdDodajKategorie, cmdDodajProdukt oraz cmdDodajOpakowanie. Ich zadaniem będzie otwarcie w odpowiednim widoku formularzy umożliwiających dodanie nowej kategorii, nowego produktu czy skojarzenia kategorii z rodzajem opakowania, a w miarę potrzeby także dodanie nowego rodzaju opakowania.

Będziemy musieli także zadbać o to, aby dodawane w ten sposób nowe pozycje do tabel tKategoria, tNazwy, tOpakowanie czy tRodzajOpakowania były unikalne. Zastosujemy tu między innymi metodę znaną z omówionej wcześniej aplikacji bibliotecznej, a polegającej na badaniu naruszenia indeksu w danej tabeli. Wymaga to wcześniejszego sprawdzenia, czy w projekcie tabeli tKategoria pole NazwaK ma ustawioną właściwość *Indeksowanie* na opcję Tak (bez duplikatów). Dokładnie taką samą właściwość ustawiamy dla pola NazwaP w tabeli tNazwy oraz pola Opis w tabeli tRodzajeOpakowania. W przypadku tabeli tOpakowania unikalna musi być kombinacja pól id_rodzaj oraz id_k, stworzenie możliwości badania naruszenia indeksu wymagałoby zbudowania na tych dwóch polach klucza (złożonego) tej tabeli. W naszym przypadku zastosujemy (również znane z bazy Biblioteka.mdb) rozwiązanie polegające na zadaniu zapytania do bazy o istnienie rekordu w tej tabeli o zadanej kombinacji tych pól. Odpowiedź negatywna pozwoli na dodanie nowego rekordu.

Zaczynamy od stworzenia formularza frmKategorie, przy czym zaprojektujemy go w taki sposób, aby mógł być wykorzystywany zarówno do przeglądania i edycji dotychczas zarejestrowanych kategorii jak i do dopisania nowej kategorii w momencie wypełniania formularza zamówienia.

Poniżej pokazany jest widok projektu tego formularza, jest on stosunkowo prosty, zawiera bowiem tylko jedno pole tekstowe z etykietą oraz przycisk polecenia o nazwie cmdZamknij.



Jeżeli chcemy, aby ten formularz (i inne także) mógł być wykorzystywany do różnych zadań, to potrzebna jest nam jakaś zmienna sterująca i badanie jej wartości w procedurze zdarzeniowej uruchamianej w momencie otwierania tego formularza. W naszym

przypadku zmienną tą będzie zmienna `intFlaga`, którą zadeklarujemy w module ogólnym.

Pozostaje napisanie procedury różnicującej funkcje i widok tego formularza, jej kod pokazany jest poniżej.

```
Private Sub Form_Open(Cancel As Integer)
    If intFlaga = 1 Then
        ' edycja i przegląd
        Me.NavigationButtons = True
        Me.cmdZapisz.Visible = False
        Me.AllowEdits = True
        Me.AllowAdditions = False
        Me.Caption = "Przegląd i edycja kategorii"
    Else
        ' wprowadzanie nowego rekordu
        Me.NavigationButtons = False
        Me.cmdZapisz.Visible = True
        Me.AllowEdits = False
        Me.DataEntry = True
        Me.Caption = "Wprowadzanie nowej kategorii"
    End If
End Sub
```

Pozostaje jeszcze napisanie procedury uruchamianej poprzez klik przycisku `cmdZapisz`, jej zadaniem będzie dopisanie nowej kategorii do tabeli `tKategorie`, ale pod warunkiem, że będzie to nazwa unikalna. Po dopisaniu musimy odświeżyć źródło danych pola kombi `Kategoria` w podformularzu `frmPodZamowienie`. Poniżej pokazany jest kod tej procedury, warto zwrócić uwagę na sposób odwołania się do wspomnianego wyżej pola `Kategoria`.

```
Private Sub cmdZapisz_Click()
    On Error GoTo errHandler
    ' badanie, czy została podana nazwa
    If IsNull(Me.nazwaK) Then
        MsgBox "Pole 'Nazwa' jest puste!", vbCritical, conKom
        Exit Sub
    Else
        Dim ctl As Control
        DoCmd.GoToRecord
        Set ctl =
            Forms!frmZamowienie!frmPodZamowienie.Form!Kategoria
        ctl.Requery
        DoCmd.Close 'acForm, "frmKategorie", acSaveNo
        Exit Sub
    End If
End Sub
```

```
End If
ErrorHandler:
If Err.Number = 2105 Then
    MsgBox "Taka Kategoria już jest w bazie!", _
        vbCritical, conKom
    Me.nazwaK.SetFocus
Else
    MsgBox Err.Description, vbCritical, conKom
End If
End Sub
```

W procedurze tej kilkukrotnie wywoływana jest procedura `MsgBox` zwracająca na ekran odpowiedni komunikat w przypadku wystąpienia różnego rodzaju błędu. Jednym z parametrów, z którym wywoływana jest ta procedura, jest nazwa okna dialogowego. Z uwagi na wielokrotne wywoływanie tej procedury nazwa okna dialogowego została zdefiniowana jako stała `conKom` w module ogólnym:

```
Public Const conKom = "Sklep ogrodniczy"
```

W zasadzie pozostało nam tylko napisanie procedury zdarzeniowej związanej z przyciskiem polecenia `cmdDodajKategorie`, jej zadaniem będzie przypisanie zmiennej sterującej `intFlaga` odpowiedniej wartości i wywołanie formularza `frmKategorie`. Formularz ten otworzymy w trybie okna dialogowego, co wymusi na użytkowniku pracę z formularzem dodawania kategorii do momentu jego zamknięcia.

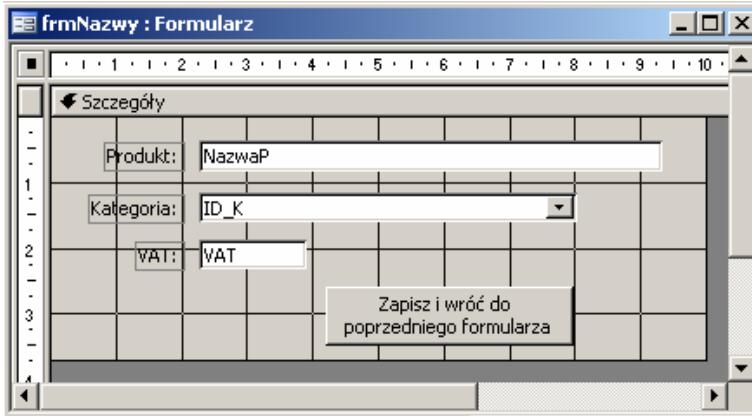
```
Private Sub cmdDodajKategorie_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmKategorie", acNormal, , , acFormAdd, _
        acDialog
End Sub
```

Jak widzimy do zmiennej `intFlaga` przekazana jest wartość 2, a polecenie `DoCmd` wywołuje metodę `OpenForm` otwierając formularz `frmKategorie` w widoku normalnym w trybie dodawania nowego rekordu i z opcją okna dialogowego.

Działanie przycisku `cmdDodajKategorie` możemy już wypróbować, potrzeba jego wykorzystania może zdarzyć się w takiej przykładowej sytuacji, gdy jesteśmy w trakcie przygotowywania zamówienia i nagle przy kolejnej pozycji okazuje się, że nie mamy odpowiedniej kategorii. Klik tego przycisku ratuje sytuację, dodajemy nową kategorię i możemy kontynuować definiowanie zapytania, choć raczej tylko teoretycznie. Za moment okaże się, że nie mamy także nazwy tego produktu, który chcemy właśnie zamówić. Inaczej mówiąc musimy przygotować teraz kolejny formularz, jego zadaniem będzie umożliwienie dodania nowej nazwy produktu oraz przypisanie jej do odpowiedniej kategorii produktów.

Ten nowy formularz, powiedzmy o nazwie frmNazwy, będziemy chcieli zaprojektować w taki sposób, aby możliwe było jego wykorzystanie w dwojakiej sytuacji: z jednej strony do zwykłego przeglądania nazw produktów i odpowiadających im kategorii wraz z możliwością edycji, z drugiej strony do dopisania brakującej nazwy.

Postępując w standardowy sposób otwieramy nowy formularz bazujący na tabeli tNazwy i umieszczać w nim takie kontrolki, abytrzymać pokazany niżej formularz.



Pole tekstowe NazwaP odwołuje się do pola o tej samej nazwie w źródle danych, pole kombi o nazwie cboKategoria jest powiązane z polem id_k, a pole tekstowe VAT z polem o tej samej nazwie w tabeli tNazwa. Źródłem danych dla pola kombi jest pokazane niżej zapytanie:

```
Select id_k, nazwak FROM tKategorie ORDER BY nazwak
```

Przycisk cmdZapisz będzie wykorzystywany w sytuacji, gdy formularz ten zostanie wywołany w celu dodania nowej nazwy produktu w trakcie przygotowywania zamówienia. Procedura obsługująca klik tego przycisku jest bardzo podobna do podobnej procedury przedstawionej w formularzu frmKategorie.

```
Private Sub cmdZapisz_Click()
    On Error GoTo errHandler
    'badanie, czy została podana nazwa produktu
    If IsNull(Me.nazwaP) Then
        MsgBox "Pole 'Produkt' jest puste!", vbCritical, _
            conKom
        Exit Sub
    Else
        Dim ctl As Control
        DoCmd.GoToRecord
```

```

Set ctl = _
    Forms!frmZamowienie!frmPodZamowienie.Form!Nazwa
ctl.Requery
DoCmd.Close
Exit Sub
End If
errHandler:
If Err.Number = 2105 Then
    MsgBox "Taki produkt już jest w bazie!", _
        vbCritical, conKom
    Me.nazwaP.SetFocus
Else
    MsgBox Err.Description, vbCritical, conKom
End If
End Sub

```

Dla wywołania formularza frmNazwy musimy napisać procedurę zdarzeniową wywoływaną przez klik przycisku cmdDodajProdukt w formularzu frmZamowienie.

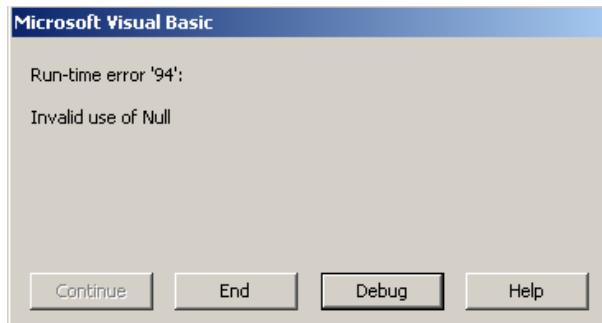
```

Private Sub cmdDodajProdukt_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmNazwy", acNormal, , ,
        acFormAdd, acDialog
End Sub

```

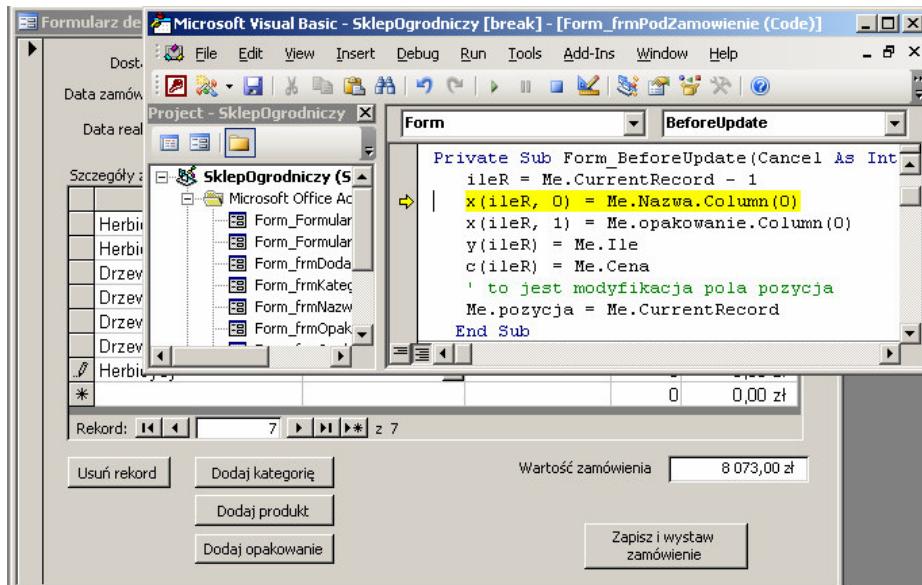
Próba wykorzystania przycisku cmdDodajKategorie jest pozytywna, otwierany jest formularz frmKategorie, możemy dodać nową kategorię, a po powrocie do formularza definiowania szczegółów zapytania znajdujemy dodaną przed chwilą kategorię na liście pozycji w polu kombi Kategoria.

Próba dodania nowej nazwy produktu kończy się jednak niepowodzeniem, a VBA zgłasza pokazany niżej komunikat o niepoprawnym użyciu wartości Null.



Bezpośrednią przyczyną wystąpienia tego błędu jest procedura BeforeUpdate formularza frmPodZamowienie, jak być może pamiętamy jej zadaniem było przejęcie i zapamiętanie w zmiennych macierzowych wprowadzanych w danym momencie szczegółów danej pozycji zamówienia. Klik przycisku cmdDajProdukt umieszczonego w formularzu głównym jest dla systemu sygnałem do zapisania rekordu w podformularzu, co z kolei uruchamia wspomnianą przed chwilą procedurę zdarzenia BeforeUpdate.

Klik przycisku Debug w oknie komunikatu o błędzie dokładnie pokazuje co się dzieje, co jest złego w naszej procedurze.



Widzimy, że błąd wystąpił w momencie przypisywania do zmiennej macierzowej `x` wartości zwróconej przez kolumnę 0 w polu kombi `Nazwa`. Problem polega na tym, że my nie wybraliśmy żadnej pozycji, tym samym zwracana jest wartość Null i stąd problem.

Usuniemy go poprzez badanie, czy procedura `BeforeUpdate` ma prawo być wykonana, jako kryterium na tak uznamy wprowadzenie do pola `Cena` wartości większej od zera. Poniżej poprawna postać tej procedury, od tego momentu nie ma żadnych problemów z dodaniem czy to nowej nazwy produktu, czy (za moment) także nowego opakowania.

```

Private Sub Form_BeforeUpdate(Cancel As Integer)
    On Error GoTo errHandler
    ' jeżeli pole Cena jest puste lub zero, to wyjdź
    If IsNull(Me.Cena) Or Me.Cena = 0 Then

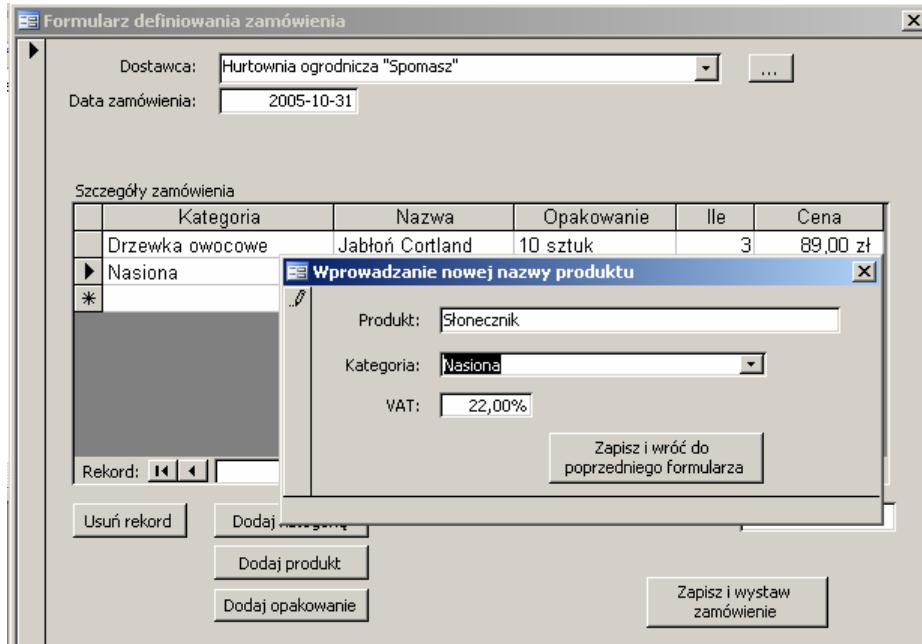
```

```

    Exit Sub
End If
ileR = Me.CurrentRecord - 1
x(ileR, 0) = Me.Nazwa.Column(0)
x(ileR, 1) = Me.opakowanie.Column(0)
y(ileR) = Me.Ile
c(ileR) = Me.Cena
' to jest modyfikacja pola pozycja
Me.pozycja = Me.CurrentRecord
Exit Sub
errHandler:
Cancel = True
End Sub

```

Poniżej pokazana jest sytuacja dodawania nowego produktu w kategorii Nasiona. Ta kategoria była dodana przed chwilą i musiała poprzedzić wprowadzenie nazwy produktu. Po zapisaniu wprowadzonej nazwy system powróci do formularza definiowania zamówienia, a wprowadzoną nazwę produktu znajdziemy już na liście pola kombi Nazwa.

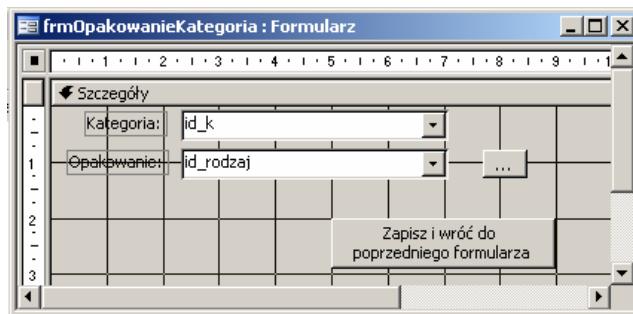


Na zakończenie tego etapu modyfikacji formularza frmZamówienie zostało nam jeszcze przygotowanie formularza frmOpakowanieKategoria, który, podobnie jak

wcześniejsze dwa formularze, także powinien pracować w dwóch różnych sytuacjach. Po pierwsze, w widoku arkusza danych powinien pozwolić na przeglądanie i edycję rodzajów opakowań w danej kategorii. Po drugie, w widoku formularza pojedynczego powinien umożliwić dodanie rodzaju opakowania do kategorii w trakcie sporządzania zamówienia na produkty do naszego sklepu. Wydaje się także za konieczne dodanie do tego formularza w tym widoku przycisku polecenia, który w miarę potrzeby uruchomi kolejny formularz pozwalający na dopisanie do bazy danych brakującego rodzaju opakowania.

Przy projektowaniu tego formularza także w inny sposób rozwiążemy problem sprawdzenia, czy wprowadzana kombinacja pól opisujących rodzaj opakowania i kategorię jest unikalna.

Przy projektowaniu tego formularza możemy wykorzystać zbudowany wcześniej formularz lub zaprojektować go całkowicie od nowa. Zaczniemy jak zawsze od otwarcia nowego formularza wskazując jako jego źródło danych tabelę tOpakowania. W formularzu tym umieścimy dwa pola kombo, pierwsze z nich ma pozwolić na wybór kategorii, drugie zaś na wybór rodzaju opakowania. Te dwa formanty w zupełności wystarczą do funkcjonowania tego formularza w pierwszym zaplanowanym scenariuszu, czyli w przeglądaniu i edycji informacji zapisanych w źródle danych. Dla zabezpieczenia drugiego z zaplanowanych scenariuszy musimy dodać jeszcze dwa formanty, będzie to przycisk cmdZapisz zapisujący wprowadzone dane oraz przycisk cmdDajRodzaj wyświetlający dodatkowy formularz pozwalający na dodanie brakującego rodzaju opakowania. W pokazanym niżej projekcie przycisk ten umieszczony jest na prawo od pola kombi Opakowanie i oznaczony symbolem trzech kropiek.



Przycisk cmdZapisz wymaga napisania procedury zdarzeniowej uruchamianej w momencie kliku tego przycisku. Poniżej kod tej procedury.

```
Private Sub cmdZapisz_Click()
    On Error GoTo errHandler
    If IsNull(Me.cboKategoria) Or _
        IsNull(Me.cboOpakowanie) Then
        MsgBox "Oba pola kombo muszą być wybrane!", _
            vbCritical, conKom
```

```
    Exit Sub
End If
Dim con As ADODB.Connection, rst As ADODB.Recordset, _
    txt As String
Set con = New ADODB.Connection
Set con = CurrentProject.Connection
txt = "select * from tOpakowania where id_rodzaj = " & _
    Me.cboOpakowanie & " and id_k = " & Me.cboKategoria
Set rst = New ADODB.Recordset
rst.Open txt, con, adOpenKeyset, adLockReadOnly
' badanie, czy liczba rekordów jest większa od zera
If rst.RecordCount > 0 Then
    MsgBox "Taka kombinacja rodzaju opakowania i kategorii
        już jest w bazie!", _
    vbCritical, conKom
    Exit Sub
End If
Dim ctl As Control
' mozna zapisać rekord
DoCmd.GoToRecord
Set ctl = _
    Forms!frmZamowienie!frmPodZamowienie.Form!opakowanie
ctl.Requery
DoCmd.Close
rst.Close
con.Close
Set rst = Nothing
Set con = Nothing
Exit Sub
errHandler:
    MsgBox Err.Number, vbInformation, conKom
End Sub
```

Procedura zdarzeniowa przycisku cmdDodajRodzaj jest analogiczna do wielu wcześniejszej omawianych procedur.

```
Private Sub cmdDodajRodzaj_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmDodajRodzaj", acNormal, , , ,
        acFormAdd, acDialog
End Sub
```

Musimy jeszcze przygotować procedurę uruchamianą w momencie otwierania formularza frmOpakowanieKategoria.

```

Private Sub Form_Open(Cancel As Integer)
    If intFlaga = 1 Then ' widok arkusza danych
        Me.AllowEdits = True
        Me.DataEntry = False
        Me.cmdDodajRodzaj.Visible = False
        Me.cmdZapisz.Visible = False
        Me.NavigationButtons = True
        Me.Caption = "Przegląd i dodawanie opakowań kategoriom"
    Else
        Me.cmdDodajRodzaj.Visible = True
        Me.cmdZapisz.Visible = True
        Me.NavigationButtons = False
        Me.Caption = "Dodanie opakowania do kategorii"
        Me.DataEntry = True
    End If
End Sub

```

W pokazanych wyżej właściwościach formularza nie ma zdefiniowanego typu widoku formularza, zdefiniujemy sposób otwarcia formularza (arkusz danych czy pojedynczy) w instrukcji DoCmd.OpenForm procedury wywołującej ten formularz. W przypadku formularza frmZamowienie i jego przycisku cmdDodajOpakowanie procedura wywołująca będzie mieć poniższą postać.

```

Private Sub cmdDodajOpakowanie_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmOpakowanieKategoria", acNormal, , , ,
                    acFormAdd, acDialog
End Sub

```

Jeżeli będziemy chcieli, aby ten formularz był pokazany w widoku arkusza danych, to procedura wywołująca powinna mieć taką postać jak niżej (nazwa Polecenie1 jest umowną nazwą jakiegoś przycisku).

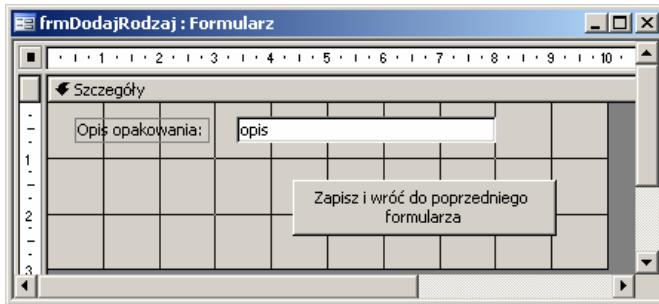
```

Private Sub Polecenie1_Click()
    intFlaga = 1
    DoCmd.OpenForm "frmOpakowanieKategoria", acFormDS, , , ,
                    acFormEdit
End Sub

```

Musimy jeszcze zasygnalizować konieczność utworzenia formularza umożliwiającego dodawanie nowego rodzaju opakowania do tabeli tRodzajOpakowania. Podobnie jak we wcześniejszych przypadkach warto ten formularz zaprojektować tak, aby mógł być wykorzystany zarówno do przeglądania i ewentualnej edycji wcześniej wprowadzonych rodzajów opakowań (w trybie arkusza danych) jak i do dodania brakującego opakowania w trakcie przygotowywania zamówienia. W tym ostatnim przypadku po wprowadzeniu rekordu musi nastąpić odświeżenie źródła danych dla pola cboOpakowanie

w formularzu frmOpakowanieKategoria. Widok projektu tego formularza pokazany jest poniżej, podobnie jak wcześniej będziemy musieli napisać kilka procedur do jego obsługi.



Pierwszą z nich będzie procedura uruchamiana w momencie otwierania tego formularza.

```
Private Sub Form_Open(Cancel As Integer)
    If intFlaga = 1 Then ' edycja i przegląd
        Me.AllowEdits = True
        Me.NavigationButtons = True
        Me.cmdZapisz.Visible = False
        Me.Caption = "Przegląd i edycja opakowań"
    Else
        'wprowadzenie nowego rekordu
        Me.AllowAdditions = True
        Me.NavigationButtons = False
        Me.cmdZapisz.Visible = True
        Me.DataEntry = True
        Me.Caption = "Dodanie rodzaju opakowania"
    End If
End Sub
```

Kolejna procedura będzie powiązana z przyciskiem cmdZapisz.

```
Private Sub cmdZapisz_Click()
    On Error GoTo errHandler
    ' badanie, czy została podana nazwa
    If IsNull(Me.opis) Then
        MsgBox "Pole 'Opis opakowania' jest puste!", _
            vbCritical, conKom
        Exit Sub
    Else
        Dim ctl As Control
```

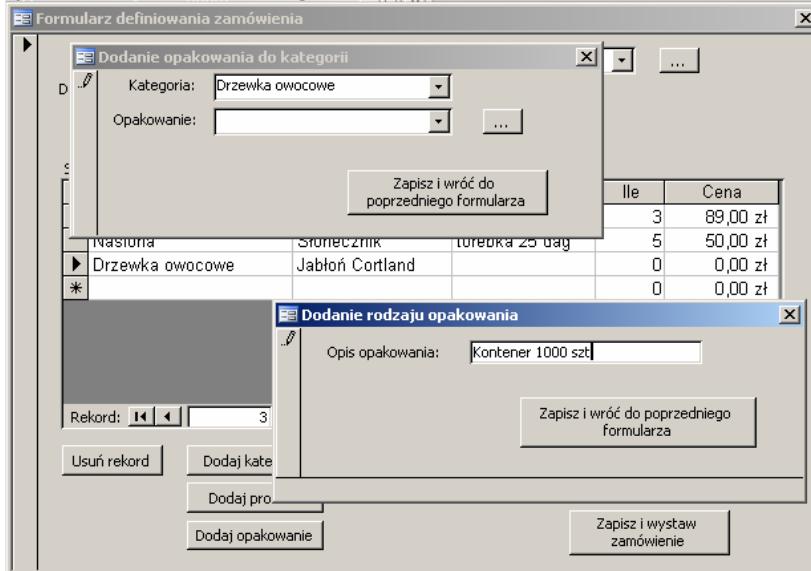
```

DoCmd.GoToRecord
Set ctl = Forms!frmOpakowanieKategoria.cboOpakowanie
ctl.Requery
DoCmd.Close 'acForm, "frmKategorie", acSaveNo
Exit Sub
End If
ErrorHandler:
If Err.Number = 2105 Then
    MsgBox "Taki opis opakowania już jest w bazie!", _
        vbCritical, conKom
    Me.opis.SetFocus
Else
    MsgBox Err.Description, vbCritical, conKom
End If
End Sub

```

Wcześniej, przy omawianiu formularza frmOpakowanieKategoria została przedstawiona procedura przypisana do przycisku cmdDodajRodzaj, jej zadaniem jest otwarcie właśnie tego formularza.

Poniżej pokazana jest sytuacja, gdy w trakcie zamawiania drzewek jabłoni odmiany Cortland chceliśmy wskazać nowy rodzaj opakowania (kontener 1000 drzewek), stąd został wyświetlony formularz „Dodanie opakowania do formularza”, ale okazało się, że w bazie nie ma takiego rodzaju opakowania, stąd przyciskiem z trzema kropeczkami został otwarty formularz „Dodanie nowego formularza”.



Po wprowadzeniu brakujących informacji wracamy do formularza definiowania zamówienia i możemy już wybrać potrzebny nam rodzaj opakowania.

Szczegóły zamówienia					
	Kategoria	Nazwa	Opakowanie	Ile	Cena
▶	Drzewka owocowe	Jabłoń Cortland	10 sztuk	3	89,00 zł
▶	Nasiona	Słonecznik	torebka 25 dag	5	50,00 zł
▶	Drzewka owocowe	Jabłoń Cortland	10 sztuk	0	0,00 zł
*			100 sztuk	0	0,00 zł
			Kontener 1000 sztuk		

Rekord: [Navigation Buttons] 3 [Next] [Last] z 3

Usuń rekord Dodaj kategorię Dodaj produkt Dodaj opakowanie Wartość zamówienia 517,00 zł Zapisz i wystaw zamówienie

Na zakończenie usprawnień formularza frmZamówienie powinniśmy jeszcze dopisać procedurę do przycisku polecenia umieszczonego na prawo od pola kombi pozwalającego na wybór dostawcy (standardowe trzy kropki). Procedura ta powinna otworzyć formularz pozwalający na dopisanie do bazy danych nowego dostawcy.

Przy założeniu, że formularz frmDostawcy będzie przeznaczony do rejestracji danych dostawcy procedura ta może mieć niżej pokazaną postać.

```
Private Sub cmdDodaj_Click()
    intFlaga = 2
    DoCmd.OpenForm "frmDostawcy", acNormal, , , _
        acFormAdd, acDialog
End Sub
```

6.3.4 Kontrola realizacji zamówień

Złożone zamówienia na dostawę produktów są w pewnym momencie realizowane, następuje dostawa zamówionych pozycji. Jest oczywiste, że projektowana baza musi mieć odpowiednie narzędzia do rozliczania i kontroli dostaw. W momencie realizacji dostawy nastąpi też ostateczna weryfikacja ceny zakupu, ustalenie wysokości marży dla firmy oraz zarejestrowanie dostarczonej pozycji na stanie magazynowym.

Będziemy oczekiwali, że aplikacja pozwoli na wybranie nam jednego z pośród dotychczas nie zrealizowanych zamówień – wykorzystamy tu brak daty w polu DataR tabeli tZamowienia. Po wyborze zamówienia powinniśmy mieć możliwość wyboru odpowiedniej, dotychczas nie zrealizowanej pozycji z danego zamówienia. Wybór takiej pozycji powinien skutkować przypomnieniem ilości, ceny jednostkowej oraz stawki podatku VAT tej pozycji. Automatycznie powinna być także naliczona marża sklepu. Jeżeli dane te będą zgodne z fakturą dostawy (rodzaj produktu, opakowanie, ilość i cena jednostkowa), to nie będziemy musieli niczego korygować i wystarczy po prostu przejść do kolejnej pozycji. Musimy jednak tak zaprojektować aplikacje, aby była możliwość zmiany ilości produktu (inaczej mówiąc dostawa jest częściowa) jak i jego ceny. Będziemy także oczekiwali, że w momencie zarejestrowania w całości danej pozycji zamówienia zniknie ona z listy pozycji do wyboru – to po to, aby wyeliminować możliwość pomyłki.

Działania, które musimy umożliwić można ująć w takich punktach:

1. Z listy dotychczas nie zrealizowanych zamówień wybieramy potrzebne zamówienie,
2. W pomocniczym formularzu w polu kombi mamy możliwość wyboru dotychczas niedostarczonego w całości (nie rozliczonego) produktu. System przypomina ilość, cenę i stawkę VAT, dodatkowo wstępnie ustalana jest marża i cena sprzedaży.
3. Mamy możliwość dokonać zmian ilości i ceny jednostkowej produktu, co automatycznie przeliczy marżę i cenę sprzedaży.
4. System odnotuje w tabeli tRealizacjaZam fakt dostarczenia odpowiedniej pozycji zapisując jej identyfikator z zamówienia, dostarczoną ilość, cenę sprzedaży, datę dostawy oraz wysokość marży,
5. W przypadku, gdy dana pozycja zamówienia zostanie zrealizowana w całości, to w tabeli tSzczegZamowienia w polu DataR zostanie wpisana data realizacji
6. Dokonane zostaną odpowiednie zmiany w tabeli tMagazyn polegające na powiększeniu pola Stan o dostarczona ilość produktu w sytuacji, gdy jest w tej tabeli już taki produkt i w tej samej cenie sprzedaży. Jeżeli nie, zrealizowana dostawa zostanie zapisana jako nowy rekord.

7. W sytuacji, gdy zamówienie jest zrealizowane w całości nastąpi wpisanie daty realizacji do pola DataR tabeli tZamowienie.

Jednym z istotnych fragmentów projektowanej aplikacji jest automatyczne wyznaczanie marży dla oferowanych produktów i ustalanie ceny sprzedaży. Do tego celu wykorzystamy pokazaną niżej tabelę tMarza.

	ID_M	Marza	Prog
▶	7	20,0%	1
	6	17,5%	50
	5	15,0%	100
	4	12,5%	150
	3	10,0%	200
	2	7,5%	400
	1	5,0%	1000
*	8	2,5%	2000
	(numerowanie)	15,0%	0
Rekord: 1 ▶▶▶*			

Przyjęto, że wysokość marży zależy będzie od wysokości ceny zakupionego u dostawcy produktu i będzie się zmieniać od 2,5% do maksymalnie 20%. Wyznaczenie marży dla konkretnego produktu zrealizujemy za pomocą odpowiedniej procedury VBA w momencie rozliczania dostawy.

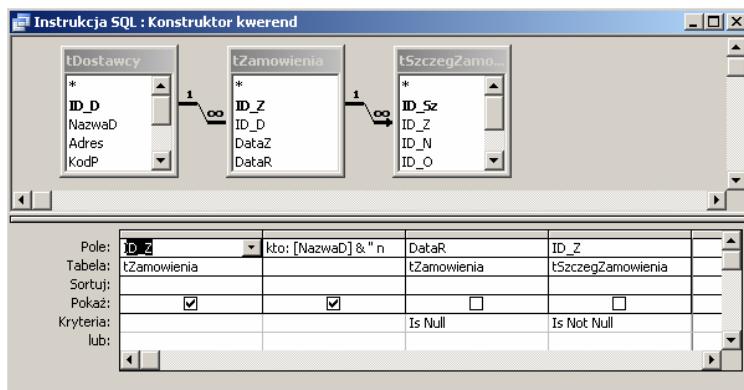
Prace nad przygotowaniem formularza (złożonego) do obsługi kontroli i rozliczania dostaw zaczniemy od utworzenia formularza głównego, w którym umieścimy pole kombi pozwalające na wybór niezrealizowanych zamówień. Z tego założenia wynika, że formularz **nie będzie** skojarzony z żadnym źródłem danych. Jego zadaniem będzie ulokowanie wspomianego wyżej pola kombi oraz drugiego formularza, który formalnie będzie występował w roli podformularza. Formalnie, ponieważ oba formularze nie będą z sobą powiązane.

Zaczynamy od otwarcia nowego formularza w widoku projektu i bez wskazywania źródła danych. W oknie tego formularza umieszczać pole kombi nadając mu nazwę cboZamowienie. W kolejnym kroku, w oknie właściwości tego formantu ustawiamy liczbę kolumn na dwie, szerokości odpowiednio na 0 i 5 cm, a jako kolumnę związaną kolumnę 1.

W pozycji *Źródło wierszy* musimy teraz wstawić albo jawnie polecenie select, albo nazwę odpowiedniej kwerendy, którą trzeba będzie utworzyć. Generalnie lepszym rozwiązaniem jest jawnie zdefiniowanie polecenia select i tak też postąpimy w naszym przypadku. Przed jego zbudowaniem musimy zastanowić się, co musimy zwrócić do pola kombi. Z pierwszym polem nie ma problemu, musi to być identyfikator zamówienia, czyli pole ID_Z z tabeli tZamowienia. Drugie zwrócone pole musi dostarczyć takie informacje, które w sposób jednoznaczny zidentyfikuje dostawcę: w naszym przekonaniu

może to być nazwa dostawcy uzupełniona numerem zamówienia i datą jego wystawienia. To jest już stosunkowo skomplikowane zapytanie, trzeba będzie skorzystać z kilku tabel, dodatkowo musimy jeszcze ustawić kryterium na pokazywanie tylko tych zamówień, gdzie pole DataR jest puste.

W oknie właściwości pola `cboZamowienie` w pozycji *Źródło wiersza* znajdziemy przycisk poleceń oznaczony trzema kropkami. Jego klik otwiera okno graficznego definiowania zapytania.



Potrzebne do zbudowania instrukcji SQL tabele dodajemy z menu kontekstowego uruchamianego z prawego przycisku myszy, jeżeli wcześniej były zdefiniowane relacje między tabelami, to są automatycznie pokazywane w oknie projektu. Zawsze oczywiście można te relacje usunąć, zmodyfikować czy zdefiniować nowe.

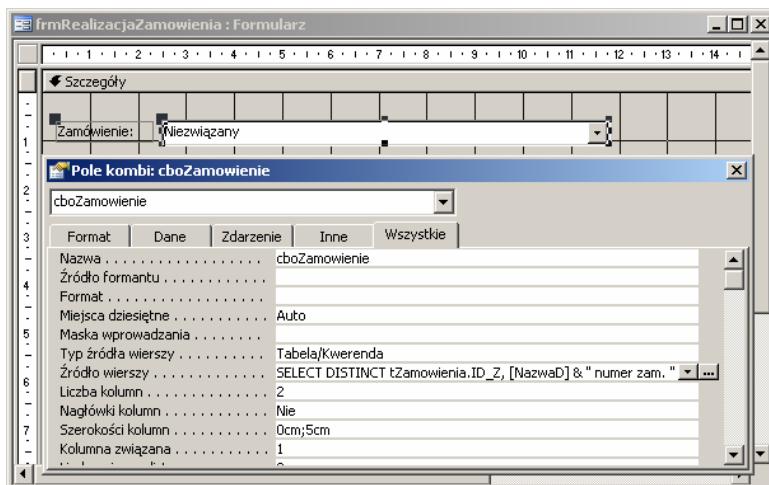
W pokazanej sytuacji zwracane jest pole `ID_Z` z tabeli `tZamowienia` oraz pole wyliczane `kto` zwracające informacje wg poniższej definicji:

```
kto: [NazwaD] & " nr zam. " & Year([DataZ]) & "/" &
      LTrim(Str([tZamowienia].[ID_Z])) & " z dnia " &
      [DataZ]
```

Pozostałe dwa pola zostały wykorzystane do zdefiniowania kryteriów wyszukiwania; dla pola `DataR` tabeli `tZamowienia` zdefiniowany jest warunek `Is Null` (pokaż te rekordy, dla których pole to jest puste, czyli zamówienie nie jest rozliczone). Drugi warunek jest bardziej skomplikowany, jego zadaniem jest pokazanie tylko tych zamówień, dla których zostały zdefiniowane szczegóły zamówienia. Aby uzyskać taki efekt konieczne było zmiana relacji między tabelami `tZamowienia` i `tSzczegZamowienia` na relację typu „*pokaż wszystko z pierwszej tabeli i tylko te rekordy z drugiej, dla której połączone pola są równe*” z dodaniem warunku `Is Not Null`. Czytelnik ma prawo zauważyć, że tak naprawdę taka sytuacja nie powinna mieć miejsca! W zasadzie będzie miał rację, kiedy bowiem coś takiego może się zdarzyć? Mianowicie wtedy, gdy w momencie definiowania

zapytania użytkownik wybrał dostawcę, ale nie zdefiniował żadnej pozycji zamówienia, po prostu zamknął formularz definiowania zapytań i powstała taka dziwaczna sytuacja. Czy można temu zapobiec? Oczywiście tak, wystarczy dopisać odpowiednią procedurę uruchamianą w momencie zamykania formularza frmZamowienie, która sprawdza czy taki fakt miał miejsce czy też nie. Jeżeli tak, to rekord odpowiadający takiemu „pustemu” zamówieniu zostanie po prostu usunięty z tabeli tZamowienia. Taką procedurę znajdzie Czytelnik w aplikacji SklepOgrodniczy.mdb na krążku dołączonym do tego skryptu. Rozwiązanie zaproponowane w tej instrukcji oczywiście nie kłoci się z podanym wyżej rozwiązaniem, jest w pewnym sensie dodatkowym zabezpieczeniem.

Po zamknięciu okna definiowania instrukcji SQL następuje powrót do okna właściwości pola kombi cboZamowienie ze wstawieniem utworzonego selektu jako źródło danych.



Dla tego pola kombi musimy jeszcze napisać procedurę zdarzeniową uruchamianą po wyborze zamówienia do rozliczenia. Jej zadaniem będzie odświeżenie źródła danych pola kombi podformularza pozwalającego na wybór produktu dla danego zamówienia.

Przy założeniu, że pole kombi będzie miało nazwę Produkt, a podformularz nazwę frmPodRZ procedura ta może mieć pokazaną niżej postać.

```
Private Sub cboZamowienie_AfterUpdate()
    Dim c1t As Control
    Set c1t = _
        Forms!frmRealizacjaZamowienia!frmPodRZ.Form!produkt
    c1t.Requery
End Sub
```

Wcześniej powiedzieliśmy, że jednym z zadań projektowanego formularza będzie automatyczne ustalanie wysokości marży. Realizacja tego zadania będzie wymagać dostępu do informacji zapisanych w tabeli tMarza. Jedną z możliwości jest zapisanie potrzebnych danych z tej tabeli w zmiennej tablicowej. Przed napisaniem odpowiedniej procedury musimy zadeklarować w module ogólnym zmienną tablicową `vm` o nieokreślonej liczbie elementów.

```
Public vm() As Single
```

Mając deklarację tej zmiennej macierzowej można już napisać procedurę uruchamianą w momencie otwarcia projektowanego formularza.

```
Private Sub Form_Open(Cancel As Integer)
    Dim con As ADODB.Connection, rst As ADODB.Recordset
    Dim txt As String, i As Integer
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    txt = "select tMarza.Prog, tMarza.Marza from tMarza " & _
        "order by tMarza.Prog DESC"
    Set rst = New ADODB.Recordset
    rst.Open txt, con, adOpenKeyset, adLockReadOnly
    If rst.RecordCount > 0 Then
        'przedefiniowanie wymiarów zmiennej tablicowej
        ReDim vm(rst.RecordCount - 1, 1)
        For i = 1 To rst.RecordCount
            vm(i - 1, 0) = rst!Prog
            vm(i - 1, 1) = rst!Marza
            rst.MoveNext
        Next i
    Else
        MsgBox "Nie można załadować tabeli marż", vbCritical, _
            conKom
        End
    End If
    rst.Close
    con.Close
    Set rst = Nothing
    Set con = Nothing
End Sub
```

Od tego momentu dane zapisane w zmiennej `vm()` są dostępne dla wszystkich procedur naszej aplikacji, my sięgniemy do tych danych w formularzu `frmPodRZ`.

Przy projektowaniu tego podformularza formalnie powinniśmy wykorzystać jako źródło danych tabelle `tSzczegZamowienia`, bo tam są zapisane potrzebne dla nas informacje. Jeżeli jednak tak zrobimy, to napotkamy podobne problemy jak przy

budowaniu podformularza dla formularza zakupów. Po prostu nie da się zrealizować jednego z zachowań tego formularza polegającego na tym, aby zrealizowana pozycja zamówienia nie była dalej widoczna na liście pola kombi Produkt. Zastosujemy podobne rozwiązanie, czyli tabelę tymczasową o nazwie np. tTempR, której projekt pokazany jest poniżej.

tTempR : Tabela			
	Nazwa pola	Typ danych	Opis
▶	produkt	Tekst	
	ilosć	Liczba	
	cena	Walutowy	
	marża	Liczba	
	vat	Liczba	

Ogólne	Odnośnik
Rozmiar pola	50
Format	
Maska wprowadzania	
Tytuł	
Wartość domyślna	
Reguła spr. poprawności	
Tekst reguły spr. poprawr	
Wymagane	Nie

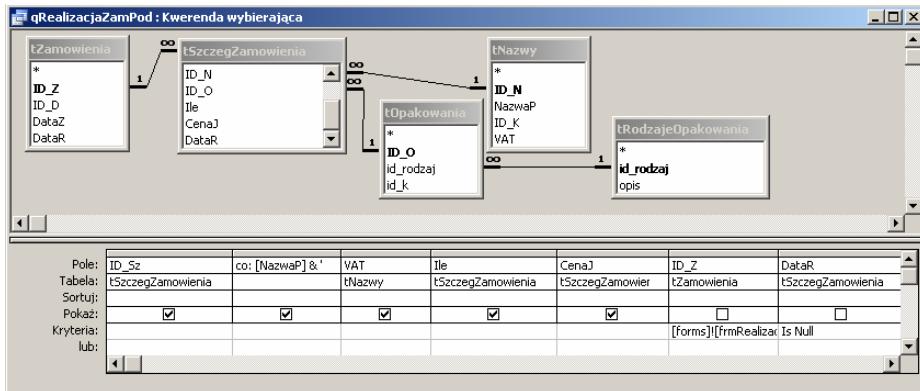
Korzystając z tej tabeli jako źródła danych tworzymy formularz frmPodRZ w widoku arkusza danych. Formularz ten będzie zawierał pole kombi dla wyboru produktu i cztery pola tekstowe odpowiadające pozostałym polom źródła danych (pole ilosc otrzyma nazwę Ilość, podobnie pole marza otrzyma nazwę Marża – z uwagi na widok arkusza danych i wykorzystanie nazw pól jako etykiet kolumn).

The screenshot shows the Microsoft Access environment with three windows open:

- frmPodRZ : Formularz**: A data sheet view showing a single row of data with columns labeled "produkt", "ilosć", "cena", "vat", and "marża".
- Formularz**: The properties dialog for the form, with the following settings:
 - Format tab: Źródło rekordów: tTempR
 - Dane tab: Filtr: , Uporządkuj według: , Filtry dozwolone: Tak
 - Zdarzenie tab: Tytuł: , Widok domyślny: Arkusz danych
 - Inne tab: Allow Form View: Tak
- tTempR**: A table view showing the structure of the temporary table with columns: produkt, ilosc, cena, marza, vat.

Najważniejszym formantem w tym formularzu jest pole kombi Produkt, pole to musi zwrócić w sposób jawny (jako kolumna związana) nazwę produktu, ale powinno zwrócić także takie informacje jak identyfikator produktu, jego ilość czy cena. Wiemy już z wcześniejszych rozważań, że możemy do tego celu wykorzystać właściwość *Column* źródła wierszy tego formantu, oczywiście pod warunkiem poprawnego przygotowania tego źródła.

Tym razem, z uwagi na dość znaczne skomplikowanie tego zapytania oraz konieczność zbudowania zapytania parametrycznego wykorzystamy dedykowaną kwerendę o nazwie `qRealizacjaZamPod`, której ogólny projekt pokazany jest niżej.



Jak widzimy korzystając z pięciu tabel zwarcane są takie pola jak: `ID_Sz`, pole wyliczane `co` zwieracjące informacje o nazwie produktu i rodzaju opakowania wg formuły:

`co: [NazwaP] & " op. " & [opis]`

dalej pole `VAT`, `Ile` oraz `CenaJ`.

Pole `ID_Z` z tabeli `tZamowienia` jest wykorzystane do zdefiniowania warunku parametrycznego. W wierszu kryteria tego pola zostało wpisane odwołanie do pola kombi o nazwie `cboZamowienie` w formularzu `frmRealizacjaZamowienia`:

`[forms]![frmRealizacjaZamowienia].[cboZamowienie]`

Dzięki temu zapytanie powyższe będzie zwracać tylko te rekordy produktów, które dotyczą wybranego zamówienia.

Pole `DataR` tabeli `tSzczegZamowienia` zostało wykorzystane do zdefiniowania warunku wyszukującego tylko te produkty, których dostawa nie została jeszcze w pełni zrealizowana, czyli pole daty realizacji jest puste.

Na zakończenie omawiania projektu jeszcze jedna uwaga odnośnie kwerend parametrycznych: jeżeli chcemy mieć pewność ich poprawnej pracy, to warto uzupełnić

projekt kwerendy o informacje o typie danych przekazanych przez odwołanie do formantu jakiegoś formularza (tu do `cboZamowienie`). Pole `ID_Z` jest polem typu *autonumer*, tym samym informacja zwrócona do pola kryteria musi być tego samego typu. Można to wymusić poprzez zdefiniowanie typu zwracanego przez parametr.



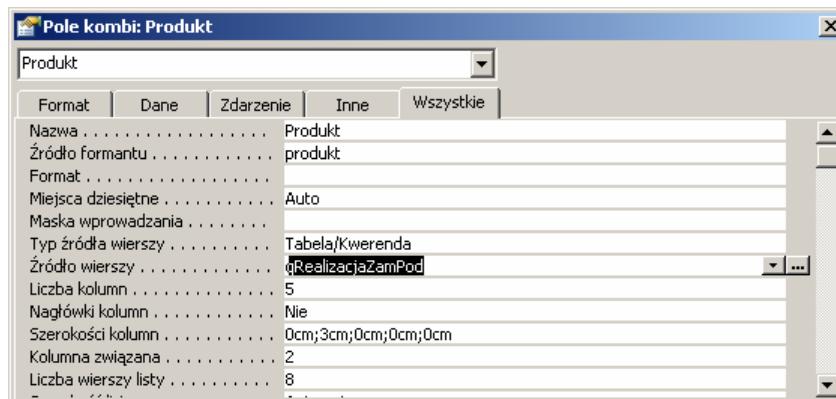
W menu *Kwerenda* lub w menu kontekstowym okna projektu kwerendy znajdziemy polecenie *Parametry*, jego wywołanie wyświetla pokazane obok okno.

W oknie tym w lewej kolumnie wklejamy skopiowane wcześniej odwołanie (parametr), a w prawej kolumnie wybieramy z pola kombi właściwy typ danych. W naszym przypadku wybieramy typ *Liczba całkowita dłuża* jako odpowiednik typu *autonumer*. Przy wyborze tego typu będziemy mieć mały problem wynikający z faktu, że okno to jest wąskie i nazwa typu jest obcinana, co gorzej okna tego nie można poszerzyć. My możemy podpowiedzieć, że wybieramy drugą z nazw „*Liczba całkowita*”, bo tak naprawdę jest to potrzebny typ *Liczba całkowita dłuża*.

Jeżeli chcemy być pewni wyboru, to stawiamy kursor w polu typ danych i klawiszem End idziemy na koniec wiersza, zobaczymy wtedy resztę nazwy.

W sumie kwerenda ta dostarczy dla pola kombi *Produkty* 5 pól danych, w sposób jawnym wykorzystamy pole `C0`, a do pozostałych odwołamy się poprzez właściwość *Column* z odpowiednim indeksem.

Po zdefiniowaniu zapytania `qRealizacjaZamPod` możemy już ustawić właściwości pola kombi *Produkt* formularza `frmPodRZ`.



Kolejny krok to przygotowanie procedury uruchamianej po aktualizacji pola kombi Produkt. Jej zadaniem jest sprawdzenie, czy już była odnotowana częściowa realizacja dostawy wybranego produktu, przypisanie do pól opisujących cenę, ilość, stawkę podatku VAT informacji pobranych ze źródła wierszy tego pola oraz wstępne ustalenie marży.

```
Private Sub Produkt_AfterUpdate()
    Dim con As ADODB.Connection, txt As String
    Dim rst As ADODB.Recordset, i As Integer
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    idsz = Me.produkt.Column(0)
    mIle = Me.produkt.Column(3)
    'ustalenie, jaką część ilości danej pozycji już
    zrealizowano
    txt = "SELECT Sum(tRealizacjaZam.ileD) AS SumaIle " & _
        "FROM tRealizacjaZam WHERE tRealizacjaZam.id_szcz = " & _
        idsz
    rst.Open txt, con, adOpenKeyset, adLockReadOnly
    ' ile zostało do realizacji
    If Not IsNull(rst!SumaIle) Then mIle = mIle - rst!SumaIle
    Me.ilosc = mIle
    Me.cena = Me.produkt.Column(4)
    Me.vat = Me.produkt.Column(2)
    ' wstępne ustalenie marzy
    For i = 0 To UBound(vm)
        If Me.cena > vm(i, 0) Then
            Me.Marza = vm(i, 1)
            Exit For
        End If
    Next i
    wIle = mIle
    wCena = Me.cena
    Me.Refresh
    rst.Close
    con.Close
    Set rst = Nothing
    Set con = Nothing
End Sub
```

W procedurze tej wykorzystywane są pomocnicze zmienne, ich zadaniem jest przechowanie identyfikatora wybranego produktu, jego ilości i ceny. Zmienne te muszą być zadeklarowane w sekcji deklaracji modułu tego formularza. Zmienne poprzedzone prefiksem „w” są przeznaczone do przechowania wprowadzonej ilości i ceny produktu.

```
Private idsz As Long, mIle As Single  
Private wIle As Single, wCena As Currency
```

Zmienna `mIle` jest wykorzystywana do przechowania aktualnej ilości zamówionego produktu pozostałej do zrealizowania w pełni dostawy. W treści procedury zmienna ta jest wyznaczana jako (ewentualna) różnica między zamówioną ilością produktu a jego wcześniejszą realizacją (jeżeli taka była).

Może tak się zdarzyć, że po wyborze produktu z listy pola `Produkt` wszystkie dalsze informacje będą zgodne z zamówieniem, a więc powinny być dokonane odpowiednie wpisy do tabeli `tSzczegZamowienia` oraz `tRealizacjaZam`. Zostanie to wykonane przez procedurę zdarzenia `BeforeUpdate` dla formularza `frmPodRZ`.

```
Private Sub Form_BeforeUpdate(Cancel As Integer)  
    'zbadanie, czy pola Me.ilosc lub Me.cena były modyfikowane  
    If Me.ilosc <> wIle Or Me.cena <> wCena Then Exit Sub  
    Dim con As ADODB.Connection, txt As String  
    Set con = New ADODB.Connection  
    Set con = CurrentProject.Connection  
    ' wstępnie zrealizowano w całości zamówienie danej pozycji  
    ' wstawiamy datę do pola DataR tabeli tSzczegZamowienia  
    txt = "update tSzczegZamowienia set dataR = '" & _  
        Format(Now(), "yyyy-mm-dd") & _  
        "' where id_Sz = " & idsz  
    con.Execute txt  
    ' do tabeli tRealizacjaZam wstawiamy ilość, cena  
    txt = "insert into tRealizacjaZam (id_szcz, ileD, " & _  
        "cenaF, DataD, Marza) values ( " & _  
        idsz & ", " & Zmien(Me.ilosc) & ", " & _  
        Zmien(Me.cena) & ", '" & _  
        Format(Now(), "yyyy-mm-dd") & "', " & _  
        Zmien(Me.Marza) & ")"  
    con.Execute txt  
    con.Close  
    Set con = Nothing  
    Me.produkt.Requery  
End Sub
```

Przy wstawianiu ilości i ceny produktu może tak się zdarzyć, że wstawiane liczby nie będą całkowite (na pewno dotyczy to marży). Ponieważ w poleceniu `insert` użyta jest klauzula `values`, to format wprowadzanych liczb **musi być zgodny** z formatem przyjętym w językach programowania, a to oznacza, że separatorem części dziesiętnej musi być kropka a nie przecinek. Zamiana dokonywana jest przez specjalnie napisaną (w module ogólnym) funkcję `Zmien`.

```
Public Function Zmien(x As Single) As String
    Zmien = Replace(Str(x), ",", ".")
End Function
```

Może się tak jednak zdarzyć, że dostarczona ilość produktu jest mniejsza od zamówionej, tym samym wpisana przez procedurę zdarzenia *AfterUpdate* pola kombi Produkt ilość produktu w polu tekstowym Ilość **musi** być zmieniona. Po zmianie konieczne jest usunięcie wcześniej wpisanej daty do pola DataR tabeli tSzczegZamowienia (bo nie jest dostarczona cała zamówiona ilość produktu), zmiana musi także ulec wpis w tabeli tRealizacjaZam. Działania takie będą wykonane przez procedurę zdarzeniową *AfterUpdate* pola Ilość formularza frmPodRZ.

```
Private Sub Ilość_AfterUpdate()
    If Me.ilosc < wIle Then
        'dostarczona ilość jest mniejsza od zamówionej,
        'tym samym trzeba zmienić wpis w
        'tRealizacjaZam oraz usunąć datę w tSzczegZamowienia
        Dim con As ADODB.Connection, txt As String
        Set con = New ADODB.Connection
        Set con = CurrentProject.Connection
        txt = "update tRealizacjaZam set ileD = " & _
            Zmien(Me.ilosc) & " where id_szcz = " & idsz & _
            " and dataD = #" & Format(Now(), "yyyy-mm-dd") & "#"
        con.Execute txt
        txt = "update tSzczegZamowienia set DataR = " & _
            "Null where id_sz = " & idsz
        con.Execute txt
        con.Close
        Set con = Nothing
    End If
End Sub
```

Analogiczna sytuacja może się zdarzyć odnośnie pola Cena w momencie, gdy cena na zamówieniu jest inna niż cena dostawy. Po aktualizacji ceny musi być aktualizacja wpisu w tabeli tRealizacjaZam. Podobnie jak wyżej zadanie to zrealizuje procedura zdarzenia *UfterUpdate* pola Cena.

```
Private Sub Cena_AfterUpdate()
    If Me.cena <> wCena Then
        ' nastąpiła zmiana ceny, ponownie ustalamy marżę
        ' i aktualizujemy pola CenaF i Marza w tRealizacjaZam
        Dim con As ADODB.Connection, txt As String, i As Integer
        Set con = New ADODB.Connection
        Set con = CurrentProject.Connection
```

```

' ustalenie zmienionej marży
For i = 0 To UBound(vm)
    If Me.cena > vm(i, 0) Then
        Me.Marza = vm(i, 1)
        Exit For
    End If
Next i
' definicja zapytania aktualizujacego
txt = "update tRealizacjaZam set cenaF = " & _
    Zmien(Me.cena) & _
    ", Marza = " & Zmien(Me.Marza) & _
    " where id_szcz = " & idsz & _
    " and dataD = #" & Format(Now(), "yyyy-mm-dd") & "#"
con.Execute txt
con.Close
Set con = Nothing
End If
End Sub

```

Poniżej widok przygotowanego formularza frmRealizacjaZam w trakcie rejestrowania dostaw jednego z zamówień.

Produkt	Ilość	Cena	Vat	Marża
Roundap op. 20 kg	55	189,99 zł	7,00%	12,50%
Goal op. 1 litr	6	47,50 zł	7,00%	20,00%
*	0	0,00 zł	0,00%	0,00%

Pierwszy z produktów został dostarczony zgodnie z zamówieniem, drugi jedynie w części (6 zamiast 12 jednostek), stąd aktualizacja pola Ilość. Po przejściu do innego pola lub kolejnego rekordu dokonane zostaną odpowiednie aktualizacje w tabelach tRealizacja i tSzczegZamowienia.

W pokazanym niżej fragmencie tabeli tSzczegZamowienia widoczne są dwa rekordy dla ID_Z równego 16; dla pierwszego z tych rekordów odpowiadającemu produktowi „Goal” pole DataR jest puste (efekt działania procedury AfterUpdate pola

Ilość). Dla drugiego rekordu odpowiadającego produktowi „Roundap” pole DataR jest wypełnione, co jest sygnałem zrealizowania w całości dostawy tego produktu.

	ID_Sz	ID_Z	ID_N	ID_O	ile	CenaJ	DataR
+	47	16	3	3	12	47,50 zł	
▶ +	48	16	1	12	55	189,99 zł	2005-11-04
+	49	17	3	7	2	2,00 zł	

Kolejny zrzut pokazuje fragment tabeli tRealizacja, pierwszy rekord odpowiada produktowi „Roundap”, drugi produktowi „Goal”. W tym drugim rekordzie widzimy, że pole ileD zostało zaktualizowane do 6 jednostek.

	id_r	id_szcz	ileD	CenaF	DataD	Marza
▶	36	48	55	189,99 zł	2005-11-04	0,125
	37	47	6	47,50 zł	2005-11-04	0,2
*	(numerowanie)		0	0,00 zł		0

Na zakończenie prac nad formularzem frmRealizacjaZam zostało nam jeszcze oprogramowanie przycisku cmdZamknij tego formularza. Zadaniem tej procedury będzie zarejestrowanie dostarczonych produktów w tabeli tMagazyn z jednoczesnym wyliczeniem ceny sprzedaży (pole CenaS) w zależności od ceny zakupu (pole CenaF) i marży (pole Marza) z tabeli tRealizacjaZam oraz stawki podatku Vat (pole VAT) z tabeli tNazwy wg wzoru:

$$CenaS = CenaF \cdot (1 + Vat + Marza + Marza \cdot Vat)$$

Wyliczona cena sprzedaży (CenaS) jest skomponowana z:

- wartości netto produktu (CenaF),
- kwoty podatku Vat od ceny netto (CenaF·Vat),
- kwoty wynikającej z marży sklepu (CenaF·Marza)
- kwoty należnego podatku Vat z tytułu usługi sprzedaży (CenaF·Marza·Vat).

W procedurze zastosujemy zaokrąglenie wyliczonej kwoty do określonej liczby miejsc po kropce uzależniając wielkość zaokrąglenie (od zera do dwóch miejsc) od wyliczonej ceny sprzedaży. I tu pojawi się kolejny problem, ponieważ ostatni składnik ceny sprzedaży jest kwotą podatku Vat od wartości dodanej i musimy się z niego rozliczyć z odpowiednim urzędem. W wyniku zaokrąglenia ceny sprzedaży powyższa równość zostanie zakłócona, chyba że skorygujemy wysokość naszej marży wg wzoru:

$$nowaMarza = round\left(\frac{round(CenaS, d)}{CenaF \cdot (1 + Vat)} - 1, 4\right)$$

Przed przystąpieniem do pracy nad tą procedurą zmodyfikujemy jeszcze projekt tabeli tMagazyn poprzez dodanie dwóch pól: CenaF, które będzie przechowywać cenę zakupu danego produktu (z tabeli tRealizacjaZam) oraz Marza do przechowania marży firmy na dany produkt.

Procedura, którą mamy napisać, jest dość skomplikowana. Po pierwsze nie mamy (bezpośrednio) większości potrzebnych danych do zapisania w tej tabeli (pole ID_D, ID_N, ID_O), po drugie wskazanie potrzebnych danych z tabeli tRealizacjaZam wcale nie jest takie proste, a po trzecie, dane o zakupionym produkcie albo mają powiększyć stan magazynowy (pole StanM), albo trzeba będzie je zapisać jako nowy rekord tej tabeli.

W przypadku, gdy tabela tMagazyn zawiera już rekord odpowiadający temu samemu dostawcy, produktowi, opakowaniu i ceny sprzedaży, to powinniśmy powiększyć stan magazynowy, w innym przypadku konieczny jest nowy rekord. Poniżej pełny kod tej procedury, mamy nadzieję, że obszerne komentarze pozwolą zrozumieć jej działanie.

```
Private Sub cmdZamknij_Click()
    Dim con As ADODB.Connection, rst As ADODB.Recordset, _
        txt As String
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    txt = "select * from tRealizacjaZam where " & _
        "id_szcz in (" & _
        "select id_sz from tSzczegZamowienia where id_z = " & _
        Me.cboZamowienie & ")"
    Set rst = New ADODB.Recordset
    rst.Open txt, con, adOpenKeyset, adLockReadOnly
    Dim rst2 As ADODB.Recordset, d As Integer
    Set rst2 = New ADODB.Recordset
    Dim rst3 As ADODB.Recordset
    Dim Cena As Currency, nowaMarza as Single
    Set rst3 = New ADODB.Recordset
    If rst.RecordCount > 0 Then
        For i = 1 To rst.RecordCount
            txt = "SELECT tZamowienia.ID_D as ID_D, " & _
                "tSzczegZamowienia.ID_N as ID_N, " & _
                "tSzczegZamowienia.ID_O as ID_O, " & _
                "tNazwy.VAT as Vat " & _
                "FROM tNazwy INNER JOIN (tZamowienia INNER " & _
                "JOIN tSzczegZamowienia ON " & _
                "tZamowienia.ID_Z = tSzczegZamowienia.ID_Z) ON " & _
                "tNazwy.ID_N = tSzczegZamowienia.ID_N " & _
                "WHERE tSzczegZamowienia.ID_Sz = " & rst!id_szcz
            rst2.Open txt, con, adOpenKeyset, adLockReadOnly
```

```

' obliczenie ceny sprzedaży wg wzoru
' CenaS=CenaF*(1+Vat+Marza+Marza*Vat)
Cena = rst!CenaF * (1 + rst2!VAT + rst!Marza+ _
    rst2!VAT * rst!Marza)
'okreslenie dokładności zaokraglenia
Select Case Cena
    Case Is < 50
        d = 2
    Case Is < 100
        d = 1
    Case Else
        d = 0
End Select
Cena = Round(Cena, d)
nowaMarza = Round((Cena / (rst!CenaF * _
    (1 + rst2!VAT))) - 1, 4)
' sprawdzenie, czy w tMagazyn istnieje rekord o takich
' czterech parametrach
txt = "select * from tMagazyn where ID_D = " & _
    rst2!ID_D & _
    " and ID_N = " & rst2!ID_N & " and ID_O = " & _
    rst2!ID_O & " and CenaS = " & Zmien((Cena))
rst3.Open txt, con, adOpenKeyset, adLockReadOnly
If rst3.RecordCount = 0 Then
    ' wstawiamy nowy rekord
    txt = "insert into tMagazyn (ID_D, ID_N, ID_O, " & _
        CenaS, StanM, CenaZ, Marza) values (" & _
        rst2!ID_D & ", " & rst2!ID_N & ", " & _
        rst2!ID_O & _
        ", " & Zmien((Cena)) & ", " & Zmien(rst!ileD) & _
        ", " & Zmien(rst!CenaF) & ", " & _
        Zmien(nowaMarza) & ")"
    con.Execute txt
Else
    ' istnieje taki rekord, tylko update pola StanM
    txt = "update tMagazyn set StanM = " & _
        rst3!StanM + rst!ileD & _
        " where ID_D = " & rst2!ID_D & _
        " and ID_N = " & rst2!ID_N & " and ID_O = " & _
        rst2!ID_O & " and CenaS = " & Zmien(rst!CenaF)
    con.Execute txt
End If
rst.MoveNext
rst2.Close

```

```
        rst3.Close
    Next i
End If
txt = "delete from tTempR"
con.Execute txt
' trzeba jeszcze sprawdzić, czy zamówienie o wybranym
' numerze zostało zrealizowane,
' jeżeli tak, to do tabeli tZamowienia wstawiamy datę
txt = "select * from tSzczegZamowienia where id_sz = " & _
      Me.cboZamowienie & " and DataR = Null"
' zamknięcie rekordsetu przed ponownym otwarciem
rst.Close
rst.Open txt, con, adOpenKeyset, adLockReadOnly
If rst.RecordCount = 0 Then
    ' zrealizowane, wstawiamy datę
    txt = "update tZamowienia set DataR = '" & _
          Format(Now(), "yyyy-mm-dd") & _
          "' where id_z = " & Me.cboZamowienie
    con.Execute txt
End If
' sprzątamy po sobie
rst.Close
con.Close
Set rst3 = Nothing
Set rst2 = Nothing
Set rst = Nothing
Set con = Nothing
' zamykamy formularz
DoCmd.Close
End Sub
```

Procedura ta kończy prace nad formularzem rozliczającym i kontrolującym proces dostawy zamówionych produktów, kończy także nasze prace nad obsługą realizacji dostaw produktów.

Poniżej zrzut ekranu definiowania zamówienia na cztery wybrane produkty oraz zrzut fragmentów tabeli tSzczegZamowienia z rekordami odpowiadającymi zamówianym produktom. W przypadku formularza rozliczenia zamówienia proszę zwrócić uwagę, że w ostatnim wierszu pole Produkt nie zawiera już żadnej pozycji (całe zamówienie zostało zrealizowane).

Formularz rozliczenia zamówienia

Zamówienie: Zakłady Chemiczne "Mesko" nr zam. 2005/61 z dnia 2005-11-05

frmPodRZ

Produkt	Ilość	Cena	Vat	Marża
Goal op. 1 litr	25	23,50 zł	7,00%	20,00%
Miedzian op. 1,5 kg	50	33,00 zł	7,00%	20,00%
Słonecznik op. torba 25 dag	25	12,50 zł	22,00%	20,00%
Sylit op. 3 kg	90	45,00 zł	7,00%	20,00%
	0	0,00 zł	0,00%	0,00%

Rekord: 1 z 31 | Zamknij

Po zamknięciu tego formularza następuje aktualizacja pola DataR w tabeli szczegółów zamówień.

tSzczegZamówienia : Tabela

	ID_Sz	ID_Z	ID_N	ID_O	Ile	CenaJ	DataR
*	70	61	3	3	25	23,50 zł	2005-11-05
*	71	61	2	19	50	33,00 zł	2005-11-05
*	72	61	4	20	90	45,00 zł	2005-11-05
*	73	61	9	31	25	12,50 zł	2005-11-05

Rekord: 1 z 31 | Zamknij

Jest także wpisana data w tabeli tZamówienia przy tym zamówieniu.

tZamówienia : Tabela

	ID_Z	ID_D	DataZ	DataR
*	61	1	2005-11-05	2005-11-05
*	(numerowanie)			2005-11-05

Rekord: 1 z 50 | Zamknij

Są też odpowiednie wpisy w tabeli tMagazyn, widać także skorygowane wartości marży.

tMagazyn : Tabela

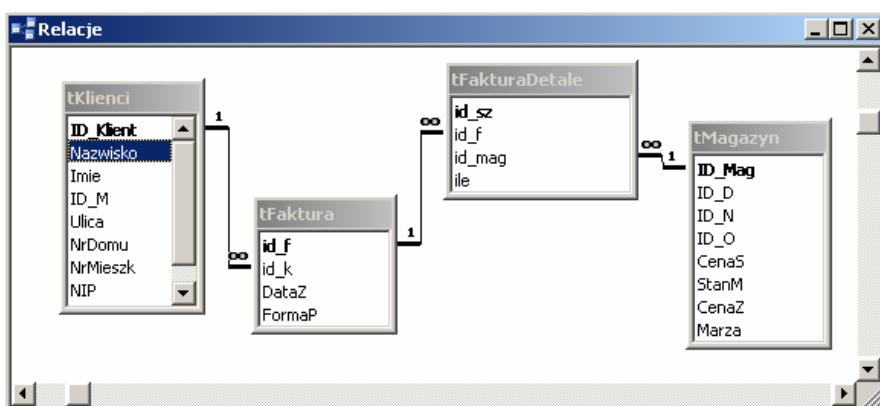
	ID_Mag	ID_D	ID_N	ID_O	CenaS	StanM	CenaZ	Marża
	32	1	2	19	42,37 zł	50	33,00 zł	0,1999
	33	1	9	31	18,30 zł	25	12,50 zł	0,2
	34	1	4	20	57,80 zł	90	45,00 zł	0,2004
	35	1	3	5	151,00 zł	23	123,00 zł	0,1473

Rekord: 1 z 30 | Zamknij

6.4. Obsługa sprzedaży

W tym fragmencie przygotowywanej aplikacji musimy przygotować formularz złożony obsługujący bezpośrednio sprzedaż oraz raport złożony drukujący fakturę sprzedaży.

Dane dotyczące sprzedaży będą rejestrowane w dwóch tabelach: tFaktury oraz tFakturyDetaile powiązanych relacjami z innymi tabelami naszej bazy. Te dwie tabele oraz tabele tKlienci i tMagazyn pokazane są niżej we fragmencie okna relacji.



Podstawową tabelą jest tFaktura, tu poza kluczem (id_f) przechowywany jest identyfikator klienta (id_k), oraz data dokonania zakupu (DataZ) i forma płatności (FormaP). To ostatnie pole zostało zdefiniowane jako liczba typu bajt, a będzie przechowywać wartości od 1 do 4 odpowiadające płatnością gotówką, przelewem, kartą płatniczą i kredytom kupieckim.

Szczegóły dokonanego zakupu będą rejestrowane w tabeli tFakturaDetaile, będzie to identyfikator faktury (id_f), identyfikator pozycji magazynowej (id_mag) oraz informacja o ilości zakupionego produktu (ile). Poprzez relację tej tabeli z tabelą tMagazyn i dalszymi tabelami (tNazwy, tOpakowania, tRodzajeOpakowania) uzyskamy wszystkie informacje o produkcie, rodzaju opakowania, cenie zakupu, stawce podatku VAT, marży firmy, także dostęp do danych dostawcy produktu.

Podobnie poprzez ustanowienie relacji między tabelą tFaktura a tabelą tKlienci i innymi tabelami opisującymi klienta uzyskamy wszystkie niezbędne dane do wystawienia faktury sprzedaży

6.4.1. Formularze sprzedaży

Zaczniemy od przygotowania koncepcji formularza sprzedaży, powiedzmy, że będzie to formularz o nazwie `frmSprzedaz` bazujący na tabeli `tFaktura`. Formularz ten wyświetlimy w widoku pojedynczego formularza, a umieścimy w nim pole kombi pozwalające na wybór zarejestrowanego klienta sklepu, podformularz pozwalający na wybór produktów dostępnych na stanie magazynowym, ramkę z przyciskami opcji określających formę płatności oraz przycisk polecający drukującą fakturę sprzedaży.

Zasadniczym elementem formularza `frmSprzedaz` będzie podformularz, jego zadaniem będzie ułatwienie wyboru odpowiedniego produktu, przekazanie informacji o rodzaju opakowania, cenie i stanie magazynowym. Po wyborze produktu podformularz ten musi także obliczyć wartość brutto wybranego produktu. Formalnie źródłem danych dla tego podformularza powinna być tabela `tFakturaDetale`, ale z podobnych względów jak przy formularzu zamówienia lepszym rozwiązaniem będzie skorzystanie z tabeli pośredniczącej. Konkretnie chodzi o zamiar wykorzystania kategorii produktu do ograniczenia jego pozycji w polu kombi wyboru produktu.

Dobrym rozwiązaniem w naszym przypadku będzie wykorzystanie wcześniej zdefiniowanej tabeli `tTemp`. W jej polach będziemy przechowywać odpowiednio:

`nazwaK` – nazwę wybranej kategorii,

`nazwaP` – kombinację nazwy produktu, rodzaju opakowania, ceny sprzedaży i stanu magazynowego,

`opakowanie` – nie będzie wykorzystane,

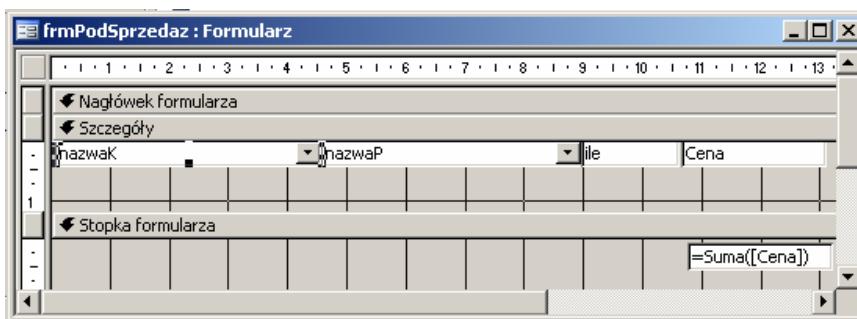
`ile` – ilość zakupionego produktu,

`Cena` – wartość brutto wybranego produktu,

`pozycja` – identyfikator magazynowy wybranego produktu.

W momencie drukowania faktury dane z tej pomocniczej tabeli (z pól `pozycja` oraz `ile`), uzupełnione o identyfikator faktury, programowo dopiszemy do tabeli `tFakturaDetale`.

Poniżej pokazany jest projekt takiego formularza o nazwie `frmPodSprzedaz`.

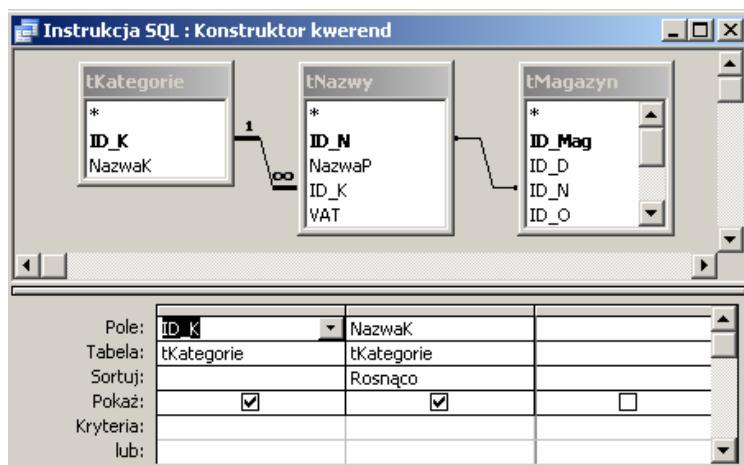


Formularz ten projektowany jest w widoku arkusza danych, mimo to ma włączony nagłówek nagłówków/stopkę raportu po to, aby w stopce można było umieścić pole tekstowe sumujące wartość zakupionych produktów. Ta sumaryczna wartość z tego formantu zostanie pokazana w formularzu głównym sprzedaży.

W formularzu frmPodSprzedaz umieszczone dwa pola kombi o nazwach odpowiednio Kategoria i Produkt oraz dwa pola tekstowe o nazwach Ilość i Wartość. Nazwy tych formantów są inne niż ich źródła danych, ale wynika to ze specyfiki pracy formularza w widoku arkusza danych. Jak już wcześniej mówiliśmy, w tym widoku nazwy formantów są jednocześnie opisami kolumn w widoku danych, dlatego jest Kategoria, a nie standardowe cboKategoria. Uwaga ta dotyczy analogicznie pozostałych formantów.

Zadaniem pola Kategoria jest pokazanie listy kategorii oferowanych produktów wg stanu magazynowego i niezależnie od rodzaju opakowania produktu z danej kategorii. Informacja zwrotna w postaci nazwy kategorii zostanie zapamiętana w polu nazwaK tabeli tTemp i pokazana w formularzu. Pole to zwróci także, ale w sposób niejawny (poprzez właściwość *Column*) identyfikator kategorii, który wykorzystamy do ograniczenia listy produktów pokazywanych w polu kombi Produkt.

W instrukcji Select definiującej źródło danych dla pola Kategoria wykorzystamy tabele tMagazyn, tNazwy i tKategorie, a do jej zbudowania można wykorzystać graficzny konstruktor kwerend.

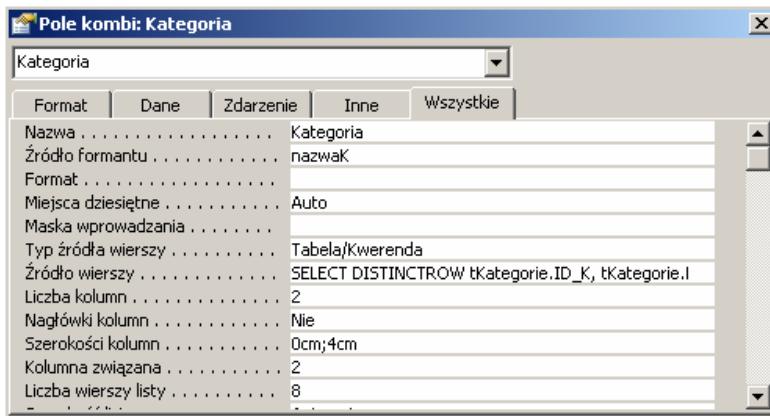


Korzystając z właściwości budowanej instrukcji (kwerendy) musimy jeszcze ustawić właściwość *Wartości unikatowe* na Tak, co spowoduje wyprowadzenie unikalnych nazw kategorii. Oczywiście instrukcję Select można napisać całkowicie samodzielnie, ale z kreatorem jest po prostu łatwiej.

Poniżej pełna postać instrukcji dostarczającej danych do pola Kategoria, instrukcja ta zwraca dwie kolumny danych; pierwsza zawiera identyfikator kategorii, a druga jej nazwę.

```
SELECT DISTINCTROW tKategorie.ID_K, tKategorie.NazwaK
FROM tKategorie INNER JOIN (tNazwy INNER JOIN tMagazyn ON
tNazwy.ID_N=tMagazyn.ID_N) ON tKategorie.ID_K=tNazwy.ID_K
ORDER BY tKategorie.nazwak
```

Pozostałe, istotne właściwości pola kombi Kategoria pokazane są poniżej.



Po wyborze kategorii oczekujemy, że zostanie zmodyfikowane źródło danych dla kolejnego pola kombi o nazwie Produkt. Zrealizujemy to za pomocą procedury zdarzeniowej *Po aktualizacji* pola Kategoria.

Zadaniem tej procedury będzie przypisanie do właściwości *RowSource* tego pola odpowiednio spreparowanej instrukcji Select, instrukcji stosunkowo skomplikowanej z uwagi na szereg informacji o produkcie, które chcemy wyświetlić w liście tego formantu. Poza nazwą produktu pobieraną z tabeli tNazwy chcemy mieć rodzaj opakowania (trzeba sięgnąć do tabeli tRodzajOpakowania poprzez tablety tOpakowanie) oraz informacje o cenie sprzedaży i stanie magazynowym. Te dwie ostatnie informacje wymagają sięgnięcia do tabeli tMagazyn, z niej zwrócimy także identyfikator produktu. W sumie instrukcja SQL ma zwrócić cztery kolumny danych:

Id_mag – identyfikator magazynowy produktu,

Pole wyliczane zawierające nazwę produktu, rodzaj opakowania, cenę sprzedaży oraz stan magazynowy,

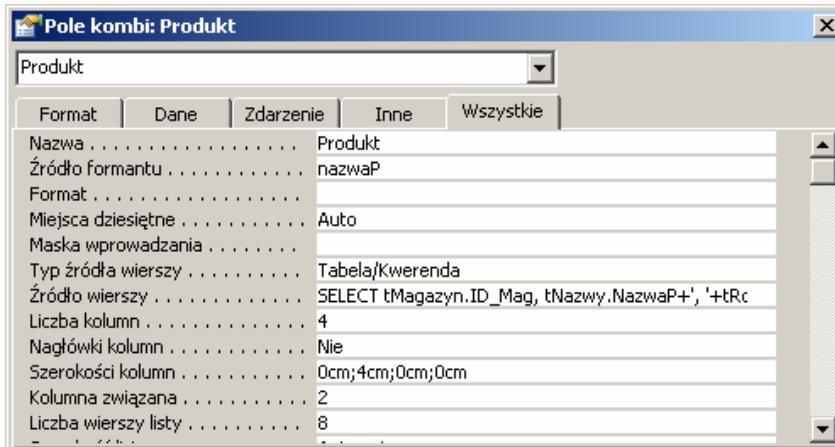
CenaS – cena sprzedaży,

StanM – stan magazynowy.

Poniżej pełna postać procedury zdarzenia AfterUpdate pola Kategoria z instrukcją SQL dostarczającą potrzebne dane do pola kombi Produkt,

```
Private Sub Kategoria_AfterUpdate()
    Me.produkt.RowSource = _
        "SELECT tMagazyn.ID_Mag, tNazwy.NazwaP" & " + " & _
        "' , ' " & " + " & "tRodzajeOpakowania.Opis" & _
        " + " & " , cena: '" & " + " & "Cstr(tMagazyn.CenaS)" & _
        " + " & "'zł stan: '" & " + " & " & _
        "Cstr(tMagazyn.StanM)" & _
        ", tMagazyn.CenaS, tMagazyn.StanM" & _
        "FROM tRodzajeOpakowania INNER JOIN " & _
        "(tOpakowania INNER JOIN " & _
        "(tNazwy INNER JOIN tMagazyn ON tNazwy.ID_N = " & _
        "tMagazyn.ID_N) " & _
        "ON tOpakowania.ID_O = tMagazyn.ID_O) ON " & _
        "tRodzajeOpakowania.id_rodzaj = " & _
        "tOpakowania.id_rodzaj" & _
        "WHERE tOpakowania.id_k = " & Me.Kategoria.Column(0) & _
        " and tMagazyn.StanM > 0"
End Sub
```

Pozostałe, istotne właściwości pola kombi Produkt pokazane są poniżej.



Wynika z nich, że pole jawnie zwraca opis produktu (z kolumny 2), do pozostałych informacji będziemy mogli się odwołać poprzez właściwość *Column*. Zrobimy to poprzez procedurę zdarzeniową *AfterUpdate* dla tego formantu.

```
Private Sub Produkt_AfterUpdate()
    CenaJednostkowa = Me.produkt.Column(2)
    StanMagazynu = Me.produkt.Column(3)
    Me.pozycja = Me.produkt.Column(0)
End Sub
```

Zmienne CenaJednostkowa oraz StanMagazynu występujące w tej procedurze zostały zadeklarowane w sekcji ogólnej (General) modułu tego formularza, a informacje w nich zapamiętane wykorzystamy za moment przy obliczaniu wartości produktu czy badaniu możliwości zrealizowania zakupu.

Dim CenaJednostkowa As Currency, StanMagazynu As Single

W przypadku pola tekstowego Ilość musimy zadbać o to, aby wprowadzona ilość produktu była większa od zera, ale nie większa od jego stanu magazynowego. Zadanie to może wykonywać procedura zdarzenia *Przed aktualizacją* tego pola.

```
Private Sub Ilość_BeforeUpdate(Cancel As Integer)
    If Me.Ilość > StanMagazynu Then
        MsgBox "Dostępne jest co najwyżej " & _
            Str(StanMagazynu) & " jednostek produktu!", _
            vbInformation, conKom
        Cancel = True
        Exit Sub
    End If
    If Me.Ilość <= 0 Then
        MsgBox "Proszę podać ilość produktu większą od zera!", _
            vbCritical, conKom
        Cancel = True
        Exit Sub
    End If
End Sub
```

Jeżeli podana ilość produktu jest poprawna, to konieczne jest zmniejszenie stanu magazynowego o tę właśnie ilość oraz wyliczenie wartości wybranego produktu. Zadanie to wykonuje procedura zdarzenia *Po aktualizacji* pola Ilość.

```
Private Sub Ilość_AfterUpdate()
    Dim con As ADODB.Connection, txt As String
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    Me.Wartość = CCur(Me.Ilość * CenaJednostkowa)
    txt = "update tMagazyn set StanM = " & _
        Zmien(StanMagazynu - Me.Ilość) & _
        " where id_Mag = " & Me.produkt.Column(0)
    con.Execute txt
End Sub
```

```
con.Close
Set con = Nothing
End Sub
```

Pole Wartość zostało zablokowane dla użytkownika, jego zadaniem jest pokazanie wartości brutto wybranego produktu, ale wartość ta jest wyliczana w pokazanej wyżej procedurze.

Musimy jeszcze przygotować kilka procedur zabezpieczających poprawną pracę tego formularza. Jedną z nich jest procedura zdarzenia *Przed aktualizacją* tego formularza. Jej zadaniem jest zareagowanie w sytuacji, gdy użytkownik chce przejść do kolejnego wiersza **przed** poprawnym zakończeniem poprzedniego rekordu.

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
If Not IsNull(Me.Kategoria) And IsNull(Me.produkt) Then
    MsgBox "Proszę wybrać produkt!", vbCritical, conKom
    Cancel = True
    Exit Sub
End If
If Not IsNull(Me.produkt) And _
    Not IsNull(Me.Kategoria) Then
    If Me.Ilość > StanMagazynu Then
        MsgBox "Dostępne jest co najwyżej " & _
            Str(StanMagazynu) & " jednostek produktu!", _
            vbInformation, conKom
        Cancel = True
        Exit Sub
    End If
    If Me.Ilość <= 0 Then
        MsgBox "Proszę podać ilość produktu większą od
            zera!", vbCritical, conKom
        Cancel = True
        Exit Sub
    End If
End If
End Sub
```

Druga z procedur została zaprojektowana po to, aby stworzyć sobie możliwość zrezygnowania z zakupionego już produktu. Do wycofania takiego rekordu będzie potrzebna informacja o numerze bieżącego rekordu, ale będziemy się do niej musieli odwołać z formularza głównego, stąd konieczność przechowywania takiej informacji w zewnętrznej zmiennej. My wykorzystamy do tego celu zmienną *intRekord*, która została przez nas zadeklarowana w module ogólnym (przy omawianiu obsługi dostaw). Przyda nam się także identyfikator magazynowy produktu, z którego chcemy zrezygnować. Do jego przechowania wykorzystamy zmienną *ileR*, jej deklaracja jest już także w module ogólnym. Do

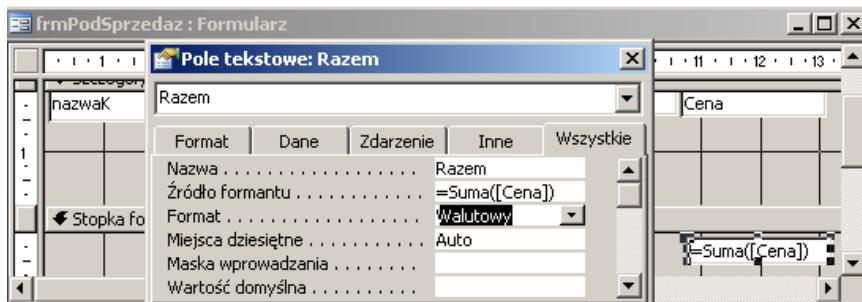
przypisania wartości tym zmiennym wykorzystamy zdarzenie *Przy bieżącym* projektowanego formularza.

```
Private Sub Form_Current()
    intRekord = Me.CurrentRecord
    If Not IsNull(Me.produkt) Then
        ileR = Me.pozycja
    Else
        ileR = 0
    End If
End Sub
```

Na zakończenie projektowania podformularza jeszcze widok właściwości pola tekstowego o nazwie *Razem* umieszczonego w stopce formularza. W pozycji *Źródło formantu* wpisana jest formula:

=Suma ([Cena])

której zadaniem jest sumowanie wartości zakupionych produktów.



Po zapisaniu utworzonego formularza pod nazwą *frmPodSprzedaz* możemy zaprojektować formularz główny.

Źródłem danych dla tego formularza będzie tabela *tFaktura*, a na jego powierzchni umieścimy pole kombi o nazwie *cboKlient*. Pole to ma pozwolić na wybranie w sposób jednoznaczny klienta zwracając jego identyfikator i opis zawierający podstawowe dane plus numer NIP.

Właściwość *Źródło wierszy* tego pola ustawiamy na pokazaną niżej instrukcję SQL, a do jej uzyskania możemy wykorzystać kreator.

```
SELECT tKlienci.ID_Klient, [Nazwisko] & " " & [Imie] & "
NIP: " & [Nip] AS kto FROM tKlienci ORDER BY [Nazwisko] & " "
& [Imie] & " NIP: " & [Nip]
```

Kolejny krok, to umieszczenie w formularzu utworzonego wcześniej podformularza, przy czym nie deklarujemy żadnych pól wiążących formularz główny z podformularzem.

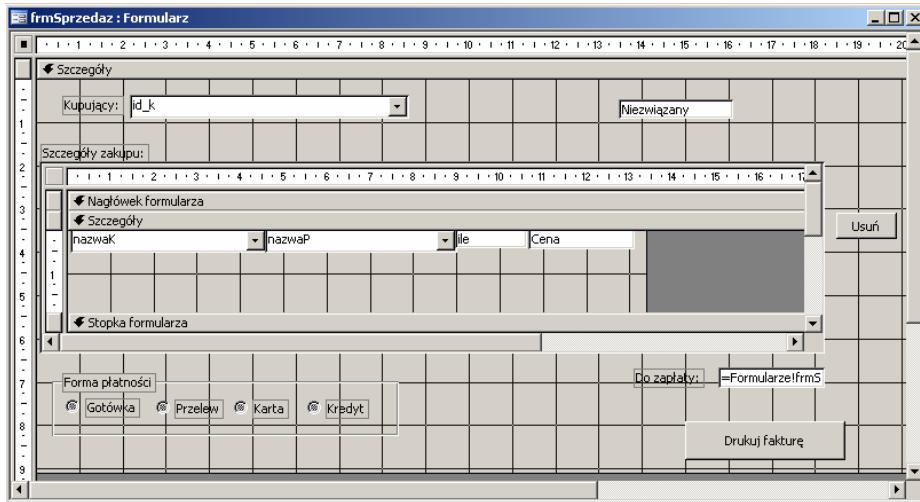
Pod podformularzem umieścimy pole tekstowe odwołujące się do pola Razem podformularza frmPodSprzedaz. Właściwości *Źródło formantu* tego pola przypisujemy odwołanie (adres):

```
=Formularze!frmSprzedaz!frmPodSprzedaz.Form!Razem
```

W lewym dolnym narożniku formularza umieścimy teraz ramkę z czterema przyciskami opcji. Jej zadaniem jest umożliwienie wyboru odpowiedniej formy płatności. Ramka ta zwraca wartości od 1 do 4, formalnie formant ten powinien być związany z polem FormaP tabeli tFaktura, ale my tego nie zrobimy.

Pozostawienie tego formantu jako nie związanego podyktowane jest sposobem pracy tego formularza. Niejak naturalne będzie takie działanie, że po wyborze kupującego przeniesiemy kurSOR do podformularza, co spowoduje **zapisanie rekordu** w tabeli tFaktura. Jeżeli teraz po wypełnieniu podformularza będziemy chcieli zmienić formę płatności, to zmienimy już istniejący rekord w tFaktura, co w konsekwencji zgłosi komunikat Accessa o tym, że rekord został zmieniony i pytaniem co z tym fantem zrobić. Tak byłoby, gdyby ramka była **związana**. Jeżeli jednak nie będzie związana, to po prostu nic się nie będzie działało w tym sensie, że zmiana formy płatności nie wywołuje żadnego zdarzenia. Pewną konsekwencją przyjętego rozwiązania będzie to, że to my musimy zadbać o to, aby odpowiednia informacja o zmianie formy zapłaty została zapisana w tabeli tFaktura. Zrobimy to programowo w obsłudze przycisku „Drukuj fakturę”.

Projekt tak przygotowanego formularza pokazany jest poniżej.



W projekcie tym widoczny jest jeszcze jeden niezwiązany formant, jest to pole tekstowe o nazwie **Tekst19**, pole to będzie niewidoczne dla użytkownika, a jego zadanie polega na przechowywaniu do dalszego wykorzystania identyfikatora faktury (**id_f**). W momencie wyboru osoby kupującej rozpoczęmy proces dodawania nowego rekordu do tabeli **tFaktury**, co sygnalizowane jest symbolem ołówka na krawędzi okna formularza. Rekord ten zostanie utworzony wtedy, gdy przejdziemy do podformularza (symbol ołówka ginie). W tabeli **tFaktura** został utworzony nowy rekord zawierający automatyczny identyfikator, identyfikator klienta, datę bieżącą i domyślną wartość formy własności. Dla dalszych potrzeb **musimy** znać wartość pola **id_f**. Zadanie to zrealizuje procedura zdarzenia *Po wstawieniu* formularza **frmSprzedaz**.

```
Private Sub Form_AfterInsert()
    Me.Tekst19 = Me.id_f
End Sub
```

Na prawo od podformularza został umieszczony przycisk polecenie „Usuń”, jego zadaniem jest usunięcie wcześniej zakupionego produktu. Procedura realizująca to zadanie pokazana jest poniżej, mamy nadzieję, że komentarze dokładnie opisują jej działanie.

```
Private Sub cmdUsun_Click()
    Dim DoZwrotu As Single
    If ileR = 0 Then
        MsgBox "Nie można usunąć rekordu, którego jeszcze nie
                ma!", vbMsgBoxHelpButton, conKom
        Exit Sub
    End If
    'trzeba usuwana pozycję dodac do pola StanM
    Dim con As ADODB.Connection, txt As String, _
        rst As ADODB.Recordset
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    ' numer wybranego do usunięcia rekordu jest w ileR
    ' selektorem odwołujemy się do pola ile źródle podformularza
    txt = "select ile from tTemp where pozycja = " & ileR
    Set rst = New ADODB.Recordset
    rst.Open txt, con, adOpenKeyset, adLockReadOnly
    ' w pomocniczej zmiennej informacji o ilości
    DoZwrotu = rst!ile
    ' zamykamy rekordset przed ponownym otwarciem
    rst.Close
    ' jaki jest bieżący stan magazynowy?
    txt = "select StanM from tMagazyn where id_Mag = " & ileR
    rst.Open txt, con, adOpenKeyset, adLockReadOnly
    ' update pola StanM
```

```

txt = "update tMagazyn set StanM = " & _
      Zmien(rst!StanM + DoZwrotu) & _
      " where id_Mag = " & ileR
con.Execute txt
' usuwamy rekord z tTemp
txt = "delete from tTemp where pozycja = " & ileR
con.Execute txt
' odświeżenie formularza
Me.Refresh
rst.Close
con.Close
Set rst = Nothing
Set con = Nothing
End Sub

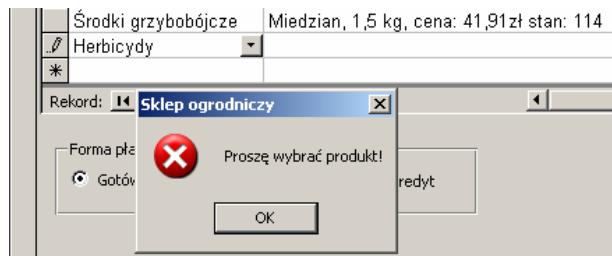
```

Poniżej pokazany jest widok formularza frmSprzedaz w trakcie transakcji sprzedaży, aktualnie mamy zamiar zarejestrować sprzedaż jednego z dwóch dostępnych środków w kategorii „środki grzybobójcze”. W polu kombi Produkt podformularza widzimy pełny opis danego produktu wraz z rodzajem opakowania, ceną jednostkową i stanem magazynowym.

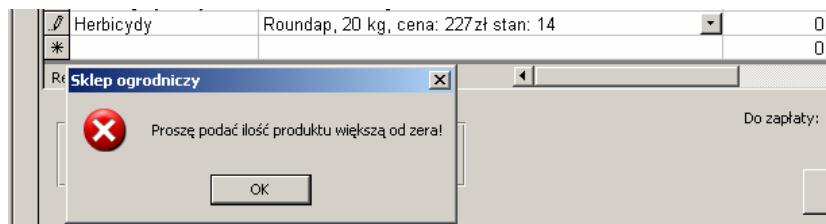
Kategoria	Produkt	Ilość	Wartość
Herbicydy	Roundap, 20 kg, cena: 229 zł stan: 26	12	2 748,00 zł
Herbicydy	Roundap, 10 litrów, cena: 275 zł stan: 134	4	1 100,00 zł
Herbicydy	Goal, 0,5 kg, cena: 2,57 zł stan: 2	2	5,14 zł
Środki grzybobójcze	Miedzian, 20 kg, cena: 98,1 zł stan: 4	1	98,10 zł
Środki grzybobójcze	Miedzian, 1,5 kg, cena: 42,37 zł stan: 75	23	974,51 zł
Środki grzybobójcze	Miedzian, 20 kg, cena: 98,1 zł stan: 3	0	0,00 zł
	Miedzian, 1,5 kg, cena: 42,37 zł stan: 52	0	0,00 zł
	Miedzian, 1,5 kg, cena: 53,9 zł stan: 23		
	Miedzian, 1,5 kg, cena: 64,2 zł stan: 33		
	Miedzian, 1,5 kg, cena: 42,37 zł stan: 50		
	Sylit, 3 kg, cena: 57,82 zł stan: 90		
	Miedzian, 3 kg, cena: 30,82 zł stan: 34		

Kończymy definiowanie transakcji zakupu poprzez klik przycisku „Drukuj fakturę”, procedurę realizującą zapisanie informacji o zakupionych produktach, ich ilości i wartości przedstawimy za moment, wcześniej jeszcze kilka słów o reakcji formularza na możliwe błędy użytkownika.

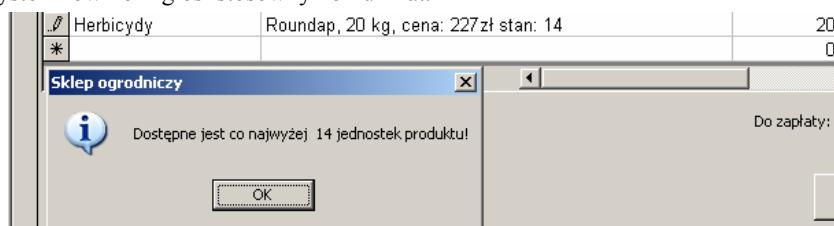
Może tak się zdarzyć, że po wyborze kategorii próbujemy zmienić rekord, system zareaguje na to zgłoszeniem pokazanego niżej komunikatu (komunikat na tle fragmentu okna formularza).



Po zatwierdzeniu możemy albo zrezygnować z wprowadzania rekordu (klawiszem ESC), albo wybrać produkt. Jeżeli wybierzemy produkt, a następnie będziemy chcieli zmienić rekord bieżący bez podania ilości produktu, to system zgłosi pokazany niżej komunikat.



Jeżeli po zatwierdzeniu komunikatu podamy ilość większą od stanu magazynowego, to system również zgłosi stosowny komunikat.



Po akceptacji komunikatu możemy albo zrezygnować z wprowadzania tego produktu (klawiszem ESC), albo podać ilość nie większą niż stan magazynowy (w pokazanym przykładzie 14 opakowań 20 kilogramowych).

Mogemy przejść już do omówienia procedury zdarzenia *Przy kliknięciu* przycisku cmdDrukFaktury.

```
Private Sub cmdDrukFaktury_Click()
    Dim con As ADODB.Connection, txt As String
    Dim rst As ADODB.Recordset, i As Integer
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
```

```
' update pola FormaP tabeli tfaktura
txt = "update tFaktura set FormaP = " & Me.Ramka7 & _
      " where id_f = " & Me.id_f
con.Execute txt
' dopisanie rekordów do tFakturaDetale
txt = "select pozycja, ile from tTemp"
Set rst = New ADODB.Recordset
rst.Open txt, con, adOpenKeyset, adLockReadOnly
If rst.RecordCount > 0 Then
    For i = 1 To rst.RecordCount
        txt = "insert into tFakturaDetale (id_f, " & _
              "id_mag, ile) values (" & _
              Me.id_f & ", " & rst!pozycja & ", " & _
              Zmien(rst!ile) & ")"
        con.Execute txt
        rst.MoveNext
    Next i
    ' wywołanie raportu faktury
    DoCmd.OpenReport "rapFakturaSprzedazy", acViewPreview
    ' zamkniecie formularza sprzedazy
    DoCmd.Close acForm, "frmSprzedaz", acSaveNo
    ' czyscimy tabele tymczasowa tTemp
    txt = "delete from tTemp"
    con.Execute txt
Else
    'brak rekordów w podformularzu
    i = MsgBox("Brak danych do wydrukowania faktury!" & _
               vbCrLf & _
               "Czy chcesz kontynuować sprzedaż?", _
               vbCritical + vbYesNo, conKom)
    If i = vbNo Then
        ' trzeba z tabeli tFaktura usunąć wpis identyfikatora
        txt = "delete from tFaktura where id_f = " & Me.id_f
        con.Execute txt
        ' zamknięcie formularza
        DoCmd.Close acForm, "frmSprzedaz", acSaveNo
    End If
End If
' sprzątamy po sobie
rst.Close
con.Close
Set rst = Nothing
Set con = Nothing
End Sub
```

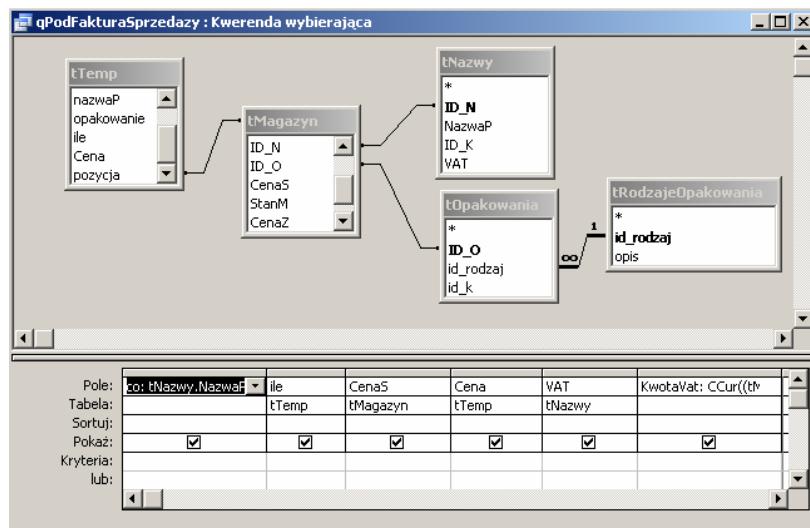
6.4.2 Faktura sprzedaży

Drugim istotnym fragmentem obsługi sprzedaży jest przygotowanie raportu złożonego, raport ten będzie pełnił funkcję dokumentu potwierdzającego dokonany zakup. Podobnie jak przy wystawianiu dokumentu zamówienia zaczniemy pracę od ustalenia, jakie informacje muszą być wyprowadzone w takim dokumencie, w jakiej części raportu mają być pokazane (w podraporcie, czy w raporcie głównym), oraz jakie źródła danych musimy przygotować.

W podraporcie powinniśmy wyprowadzić nazwę zakupionego produktu, rodzaj opakowania (lub jednostka miary), cenę jednostkową, % podatku VAT, ilość, wartość brutto produktu oraz kwotę podatku VAT. Już widać, że będziemy musieli przygotować specjalne źródło danych dla uzyskania wymienionych wyżej informacji (dane z samej tabeli tTemp nie wystarczą).

Dokument potwierdzający zakup musi zawierać dane personalne kupującego oraz jego dane adresowe i numer NIP. Będzie także potrzebna jakaś numeracja tych dokumentów, do jej konstrukcji wykorzystamy identyfikator id_f z tabeli tFaktury. Wszystkie wymienione wyżej informacje umieścimy w raporcie głównym, przy czym konieczne będzie przygotowanie specjalnego źródła danych.

Poniżej pokazany jest projekt kwerendy qPodFakturaSprzedazy, zostanie ona wykorzystana jako źródło danych dla podraportu.



Pierwsze zwracane pole o roboczej nazwie `co` jest polem wyliczonym wg formuły:

```
co: tNazwy.NazwaP & ", op. " & tRodzajeOpakowania.Opis
```

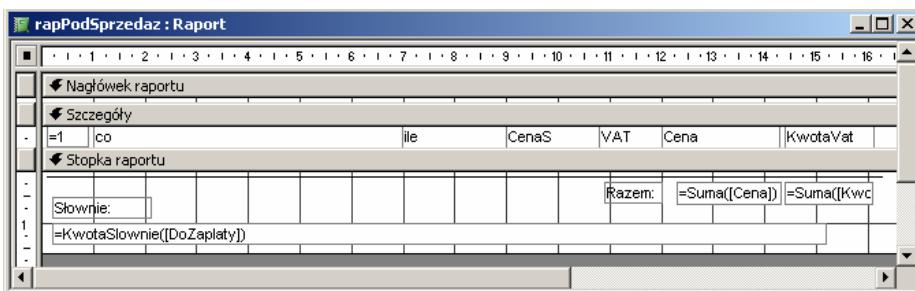
zwracającym nazwę produktu uzupełnioną informacją o rodzaju opakowania (jednostce miary).

Polem wyliczonym jest także `KwotaVat`, gdzie wyliczana jest kwota podatku VAT zawarta w danej wartości brutto produktu.

```
KwotaVat: CCur((tMagazyn.CenaS-
    tMagazyn.CenaZ*(1+tMagazyn.Marza))*tTemp.ile)
```

Funkcja `CCur` konwertuje wyrażenie do typu waluty, jej używanie jest niezbędne w tych wszystkich przypadkach, gdy wyliczamy jakieś należności finansowe (chodzi między innymi o zasady zaokrągleń).

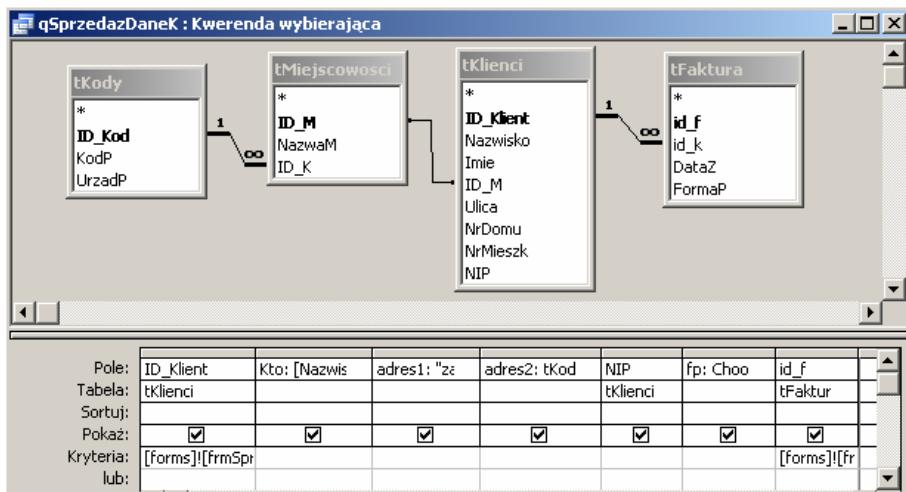
Mając źródło danych możemy zaprojektować raport o nazwie `rapPodSprzedaz`, wykorzystamy go jako podraport w raporcie złożonym.



W sekcji szczegółów umieszczamy pola źródła danych dbając o ich odpowiednie ustawienie i właściwą szerokość tych formantów. Dodatkowo na początku zostało umieszczone pole tekstowe z zadaniem numerowania poszczególnych produktów. Wszystkie te formanty nie mają etykiet opisujących zwracaną informację – etykiety takie utworzymy w raporcie głównym.

W projektowanym raporcie pokazujemy nagłówek/stopkę raportu w celu umieszczenia w stopce raportu podsumowania dwóch ostatnich kolumn (cena brutto i kwota podatku VAT). W tej sekcji umieścimy także pole tekstowe zwracające słownie kwotę do zapłaty. Podobnie jak przy dokumencie zamówienia wykorzystana została funkcja `KwotaSlownie`. Sekcja stopki raportu została oddzielona poziomą linią od listy produktów.

Mając podraport możemy przejść do przygotowania źródła danych dla raportu głównego. Z uwagi na konieczność odwołania się do ściśle określonego klienta i równie ściśle określonej pozycji w tabeli `tFaktury` zbudujemy kwerendę parametryczną.



Kwerenda ta zwróci dane tego klienta, którego identyfikator otrzyma z formularza frmSprzedaz z pola cboKlient. Dla pola ID_klient zdefiniowane zostało kryterium:

```
[forms]![frmSprzedaz].[cboKlient]
```

Klientowi temu w tabeli tFaktury może odpowiadać wiele faktur, ale nas interesuje tylko ta, której numer jest przechowywany w pomocniczym polu Tekst19 tego samego formularza. Stąd w polu id_f zdefiniowaliśmy kryterium:

```
[forms]![frmSprzedaz].[Tekst19]
```

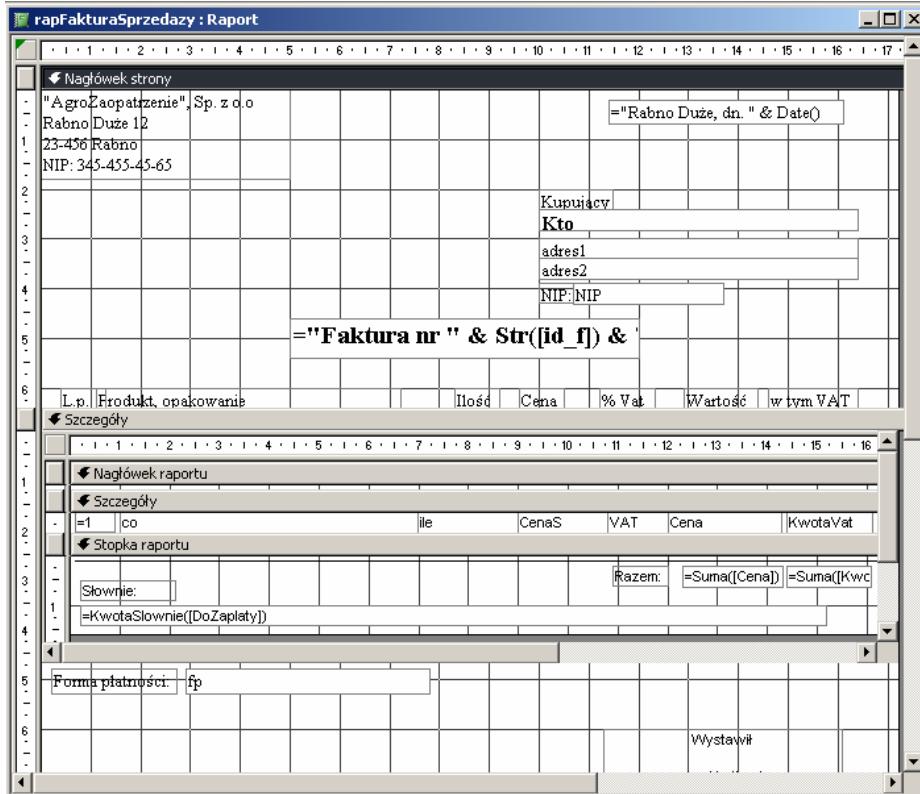
Pola Kto, adres1 i adres2 są polami wyliczanymi zwracającymi dane personalne kupującego i dane adresowe wg poniższych formuł.

```
Kto: [Nazwisko] & " " & [Imie]
adres1: "zam. " & tMiejscowosci.NazwaM & IIf(Not
    IsNull(tKlienci.Ulica); " ul. " & tKlienci.Ulica)
    & " " & tKlienci.NrDomu & IIf(Not
    IsNull(tKlienci.NrMieszk); " m. " &
    tKlienci.NrMieszk)
adres2: tKody.KodP & " " & [UrzadP]
```

Pole fp ma zwrócić tekst opisujący formę płatności, do uzyskania odpowiedniej informacji zwrotnej została wykorzystana funkcja systemowa Choose.

```
fp: Choose([FormaP]; "gotówka"; "przelew"; "karta
    płatnicza"; "kredyt kupiecki")
```

Mając źródło danych można przystąpić do projektowania raportu głównego.



W nagłówku strony raportu umieszczono informacje o podmiocie wystawiającym fakturę oraz dane osobowe i adresowe osoby dokonującej zakupu. W polu tekstowym zwrócono nazwę dokumentu uzupełnioną jego numerem skomponowanym wg formuły:

```
= "Faktura nr " & Str([id_f]) & "/" &
LTrim(Str(Year(Now()))))
```

W sekcji *Nagłówka strony* umieszczono także wiersz etykiet opisujących kolumny danych zwracanych w podraporcie. Etykiety te zostały oddzielone poziomą linią od sekcji szczegółów, gdzie umieszczono podraport bez wskazywania powiązań z raportem głównym. Pod podraportem umieszczone jeszcze pole tekstowe informujące o formie płatności oraz dane osoby wystawiającej fakturę.

W stopce strony umieszczono pole zwracające informacje o numerze strony raportu.

```
= "Strona " & [Page] & " z " & [Pages]
```

A tak wygląda przykładowa faktura w widoku podglądu wydruku.

"AgroZaopatryenie", Sp. z o.o.
Rabno Duże 12
23-456 Rabno
NIP: 345-455-45-65

Rabno Duże, dn. 2005-11-08

Kupujący
Pacanowski Zygmunt
zam. Duplice Małe ul. Jasna 23 m. 6
23-456 Duplice
NIP 234-345-45-55

Faktura nr 11/2005

L.p.	Produkt, opakowanie	Ilość	Cena	% Vat	Wartość	w tym VAT
1	Roundap, op. 20 kg	6	227,00 zł	7%	1 362,00 zł	79,57 zł
2	Sylt, op. 3 kg	2	57,20 zł	7%	114,40 zł	6,40 zł
3	Miedzian, op. 1,5 kg	6	41,91 zł	7%	251,46 zł	13,86 zł
4	Włosnia tutówka, op. 10 sztuk	2	126,00 zł	22%	252,00 zł	40,50 zł
Suma:					Razem:	1 979,86 zł
						140,33 zł

je jeden tysiąc dziewięćset siedemdziesiąt dziewięć złotych i osiemdziesiąt sześć groszy

Forma płatności: gotówka

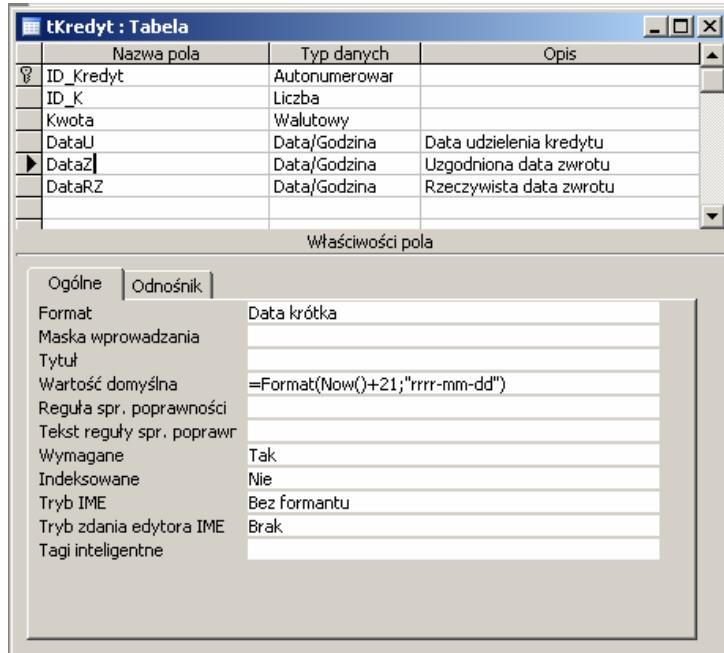
Wystawili:
Jan Kortland

6.4.3 Kredyt kupiecki

Omawiając sugerowaną funkcjonalność tej aplikacji powiedzieliśmy także o tym, że powinniśmy stworzyć możliwość udzielenia naszym klientom kredytu kupieckiego rozumianego jako zakup z odroczoną płatnością. Zastanowimy się teraz, jak taką możliwość wkomponować do formularza obsługującego sprzedaż, prawdopodobnie konieczne będzie przeprojektowanie istniejących tabel, zbudowanie jeszcze jednego formularza oraz dodanie nowych procedur w module formularza frmSrzedaz.

Zaczniemy od zmian projektu tabeli tKlienci, w którym dodamy pole LimitKredytu typu *walutowego*, pole to będzie przechowywać indywidualnie ustalany limit kredytu dla danego klienta. Na tym etapie ustalmy jego wysokość na poziomie 500 zł, w praktyce na podstawie wartości dokonywanych zakupów, terminowości spłat udzielonego kredytu limit ten może być zmieniany w obu kierunkach, także do zera.

W tabeli tKredyt będziemy przechowywać informacje o tym, komu kredyt został udzielony, w jakiej wysokości, kiedy, jaka jest data jego zwrotu (domyślnie 21 dni) oraz kiedy nastąpiła spłata.



Pola DataZ oraz DataRZ będą przez nas wykorzystywane do monitorowania losów udzielonego kredytu. W sytuacji, gdy minął uzgodniony termin spłaty (data bieżąca jest większa od daty zwrotu) przy braku wpisu w polu daty rzeczywistego zwrotu jest podstawą nieudzielania kolejnego kredytu. Podobnie badana będzie relacja udzielonego kredytu z jego limitem i wystąpieniem o kolejne skredytowanie zakupu (wybrana opcja „kredyt kupiecki” w formularzu sprzedawy).

Wynika z powyższego, że procedury związane z udzieleniem kredytu lub nie muszą być wmontowane w procedurę obsługującą klik przycisku cmdDrukFaktury formularza frmSprzedaz.

```
Private Sub cmdDrukFaktury_Click()
    ' fragment kodu sprawdzający, czy można kredytować zakup
    If SprawdzLimitKredytu(Me.cboKlient) = False Then
        Me.Ramka7 = 1
        Exit Sub
    End If
```

W pokazanym fragmencie tej procedury dodane zostały cztery wiersze kodu (pięć z komentarzem) sprawdzające możliwość udzielenia kredytu poprzez wywołanie specjalnie napisanej funkcji SprawdzLimitKredytu.

Funkcja ta zwraca wartość `False` w sytuacji, gdy nie ma możliwości udzielenia kredytu. W takim przypadku ramka opcji otrzymuje wartość 1 wymuszającą płatność gotówką i następuje wyjście z procedury, czyli powrót do formularza sprzedaży. Powrót jest po to, aby uzyskać potwierdzenie od kupującego na gotówkową formę zapłaty, ewentualnie na płatność kartą czy w najgorszym przypadku na rezygnację z zakupu.

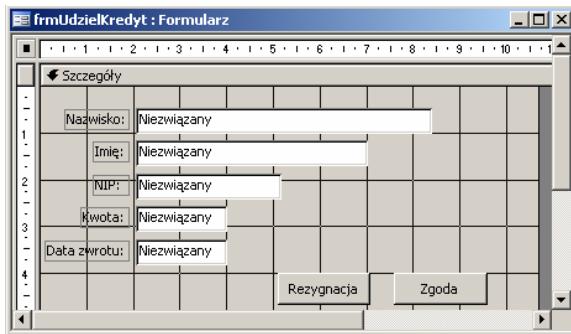
Do funkcji `SprawdzLimitKredytu` zostaje przekazany identyfikator kupującego pobrany z pola kombi `cboKlient`, reszta działań sprawdzających wraz ze stosownymi komunikatami zostanie wykonana w tej funkcji. Pełny kod tej funkcji pokazany jest poniżej, mamy nadzieję, że komentarze ułatwią zrozumienie jej pracy.

```
Private Function SprawdzLimitKredytu(id As Long) As Boolean
    ' wyjdź z wartością True, jeżeli nie kredyt
    If Me.Ramka7 < 4 Then
        SprawdzLimitKredytu = True
        Exit Function
    End If
    ' kredyt, rezerwacja zmiennych dla ADO i pomocniczych
    Dim Limit As Currency, dk As Date
    Dim con As ADODB.Connection, txt As String
    Dim rst As ADODB.Recordset, i As Integer
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    ' zapytanie o dane klienta, w tym limit kredytu
    txt = "select * from tKlienci where id_klient = " & id
    rst.Open txt, con, adOpenKeyset, adLockReadOnly
    If Not IsNull(rst!LimitKredytu) Then Limit = _
        rst!LimitKredytu
    ' zapamiętanie wysokości potrzebnego kredytu i danych
    ' kupującego w zmiennej DaneK zadeklarowanej w module
    ' ogólnym jako obiekt klasy cKlient. Rozwiązanie dla
    ' potrzeb zademonstrowania klasy, można było także
    ' wykorzystać inne zmienne z modułu ogólnego
    Set DaneK = New Klient
    DaneK.Nazwisko = rst!Nazwisko
    DaneK.Imie = rst!Imie
    DaneK.NIP = rst!NIP
    DaneK.Kwota = Me.Tekst5
    rst.Close
```

```
' zapytanie o historię kredytów nie zwróconych (jeżeli są)
txt = "SELECT * from tKredyt where DataRZ is Null" & _
      " and id_k = " & id
rst.Open txt, con, adOpenKeyset, adLockReadOnly
dk = Format(Now(), "yyyy-mm-dd")
' ewentualne zmniejszenie limitu
Do While Not rst.EOF
    If rst!DataZ >= dk Then
        ' poprzedni kredyt nie został rozliczony
        Limit = 0
        Exit Do
    Else
        Limit = Limit - rst!Kwota
        rst.MoveNext
    End If
Loop
' badanie relacji wartości zakupu do aktualnego limitu
If Me.Tekst5 <= Limit Then
    ' udzielamy kredytu, identyfikator klienta jest
    ' przypisywany do zmiennej zdefiniowanej w module
    ' ogólnym w celu jego wykorzystania przez procedury
    ' otwieranego nowego formularza
    intFlaga = id
    DoCmd.OpenForm "frmUdzielKredyt", _
                   acNormal, , , acFormAdd, acDialog
    If intFlaga = 1 Then
        'jest zgoda na kredyt, dopisujemy dane do tKredyt
        txt = "insert into tKredyt (id_k, Kwota) values (" & _
              id & ", " & Zmien((c(1))) & ")"
        con.Execute txt
        SprawdzLimitKredytu = True
    Else
        MsgBox "Ponieważ nie zaakceptowałeś kredytu, to " & _
                vbCrLf & "proszę zmienić formę płatności.", _
                vbInformation, conKom
        SprawdzLimitKredytu = False
    End If
Else
    MsgBox "Przykro mi, ale wartość zakupu jest większa" & _
            " niż Twój " & vbCrLf & _
            " limit kredytu. Proszę zmienić formę płatności.", _
            vbInformation, conKom
    SprawdzLimitKredytu = False
End If
```

```
' poniżej dwie instrukcje w wierszu, separatorem jest :
rst.Close: con.Close
Set rst = Nothing: Set con = Nothing
End Function
```

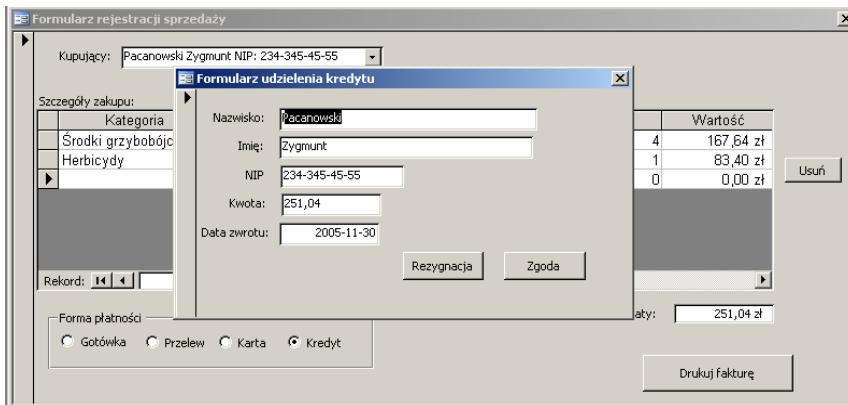
Zadaniem formularza frmUdzielKredyt jest wyświetlenie danych identyfikacyjnych kupującego i kwoty udzielanego kredytu z prośbą o akceptację. Poniżej projekt tego formularza, jest to formularz niezwiązanego z żadnym źródłem danych.



Procedura uruchamiana w momencie otwierania tego formularza dostarcza potrzebnych danych do pierwszych czterech formantów.

```
Private Sub Form_Open(Cancel As Integer)
Me.Nazwisko = DaneK.Nazwisko
Me.Imie = DaneK.Imie
Me.NIP = DaneK.NIP
Me.Kwota = DaneK.Kwota
End Sub
```

A tak wygląda ten formularz w trakcie „pracy”.



W polu DataZ formularza frmUdzielKredyt przypisano właściwości *Wartość domyślna* wyrażenie:

```
=Format(Now() +21; "rrrr-mm-dd")
```

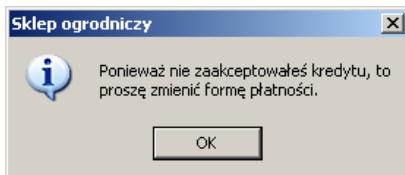
które ma powiększyć datę bieżącą o założone wcześniejs 21 dni na spłacenie udzielonego kredytu.

Zadaniem przycisków poleceń cmdBrakZgody oraz cmdZgoda jest przypisanie zmiennej intFlaga zwrotnej informacji i zamknięcie tego formularza, co będzie równoznaczne z powrotem do dalszego wykonywania kodu funkcji SprawdzLimitKredytu.

```
Private Sub cmdBrakZgody_Click()
    intFlaga = 0
    DoCmd.Close acForm, "frmUdzielKredyt", acSaveNo
End Sub

Private Sub cmdZgoda_Click()
    intFlaga = 1
    DoCmd.Close acForm, "frmUdzielKredyt", acSaveNo
End Sub
```

W przypadku akceptacji warunków kredytu formularz frmUdzielKredyt jest zamknięty i wyświetlany jest dokument sprzedaży. Jeżeli jednak nie wyrazimy zgody na warunki kredytu poprzez klik przycisku „Brak zgody”, to do funkcji zostanie zwrócona wartość 0 w zmiennej intFlaga, co w konsekwencji spowoduje wyświetlenie pokazanego niżej komunikatu.



Jego akceptacja powoduje powrót do formularza frmSprzedaż, gdzie możemy wybrać inną niż kredyt formę płatności.

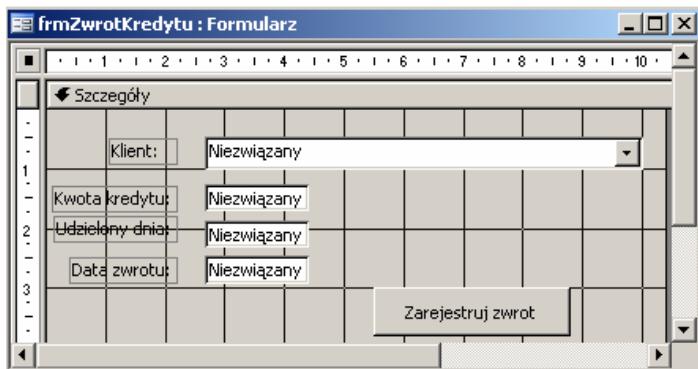
6.4.4 Spłata kredytu

Udzielony kredyt kupiecki powinien być spłacony w ustalonym terminie, a fakt spłaty odnotowany w bazie danych. Przygotujemy teraz prosty formularz, jego zadaniem będzie odnotowanie dla danego kredytu daty jego spłaty.

Utworzymy prosty formularz bez opierania go o jakieś źródło danych, w sekcji szczegóły umieścimy pole kombi pozwalające na wybranie określonego kredytu do spłaty plus trzy pola tekstowe i przycisk polecenia. Pola tekstowe wykorzystamy do przypomnienia kwoty udzielonego kredytu i daty udzielenia oraz datę oczekiwanej spłaty. Te trzy informacje są funkcjonalnie powiązane z pozycją wybraną w polu kombi, inaczej mówiąc, po wybraniu pozycji w polu kombi odpowiednie informacje mają być wpisane w te pola.

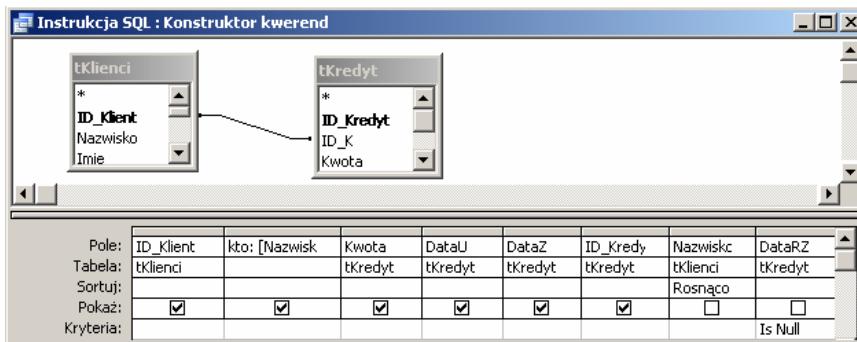
Przycisk polecenia cmdZapisz ma uruchomić procedurę zdarzeniową, jej zadaniem będzie zapytać użytkownika o poprawność zamierzonej operacji, a po jej uzyskaniu zostanie zaktualizowany rekord odpowiadający wybranemu kredytowi w tabeli tKredyt. Aktualizacja polegać będzie na wpisaniu daty bieżącej do pola DataRZ.

Projekt takiego formularza pokazany jest poniżej, pole kombi otrzymało nazwę cboKlient, pola tekstowe odpowiednio txtKwota, txtDataU, txtDataZ.



Pole kombi cboKlient wg przedstawionej wyżej koncepcji ma pozwolić na wybór pojedynczego rekordu opisującego udzielony kredyt, musi więc pokazywać dane klienta i jego numer NIP oraz kwotę udzielonego kredytu.

Dodatkowo będziemy chcieli otrzymać także takie informacje jak kwota kredytu, data udzielenia, oczekiwana data zwrotu, identyfikatory klienta (id_Klient) i kredytu (Id_Kredyt). Wszystkie te informacje zwrócimy do pola kombi poprzez przypisanie właściwości *Źródło wierszy* instrukcji SQL zwracającej potrzebne informacje z tabel tKredyt, tKlient. Do jej zbudowania można wykorzystać graficzny kreator instrukcji SQL.

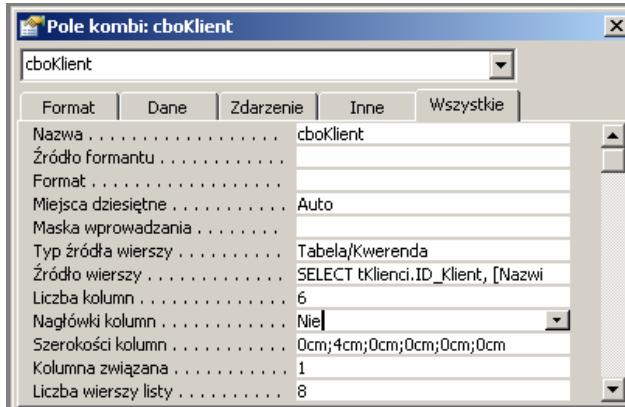


Listę pola kombi komponujemy oczywiście tylko z tych kredytów, które nie zostały spłacone, stąd warunek Is Null dla pola DataRZ. Pole Nazwisko zostało wykorzystane do sortowania zwracanych rekordów.

Po zamknięciu kreatora instrukcji SQL właściwości *Źródło wierszy* zostanie przypisany pokazany niżej select.

```
SELECT tKlienci.ID_Klient, [Nazwisko] & " " & [Imie] &
", Nip: " & [Nip] & " - " & LTrim(Str([Kwota])) & " zł." AS
kto, tKredyt.Kwota, tKredyt.DataU, tKredyt.DataZ,
tKredyt.ID_Kredyt FROM tKlienci INNER JOIN tKredyt ON
tKlienci.ID_Klient = tKredyt.ID_K WHERE (((tKredyt.DataRZ) Is
Null)) ORDER BY tKlienci.Nazwisko;
```

Zapytanie powyższe zwraca sześć kolumn informacji, bezpośrednio w polu listy wykorzystamy pole drugie, jest to pole wyliczane zwracające dane kredytobiorcy, jego numer NIP oraz wysokość udzielonego kredytu. Pozostałe informacje zwracane przez tak zdefiniowane źródło wierszy będziemy odzyskiwać korzystając z właściwości *Column*.



Zdarzeniu *Po aktualizacji* pola cboKlient musimy teraz przypisać procedurę, której zadaniem będzie przypisanie do pól tekstowych odpowiednich informacji.

```
Private Sub cboKlient_AfterUpdate()
    Me.txtKwota = CCur(Me.cboKlient.Column(2))
    Me.txtDataU = Me.cboKlient.Column(3)
    Me.txtDataZ = Me.cboKlient.Column(4)
End Sub
```

Zostało nam jeszcze napisanie procedury zdarzenia *Przy kliknięciu* dla przycisku polecenia cmdZapisz i projekt formularza będzie gotowy.

```
Private Sub cmdZapisz_Click()
    Dim con As ADODB.Connection, txt As String, i As Integer
    ' sprawdzenie, czy wybrano klienta w polu kombi
    If IsNull(Me.cboKlient) Then
        MsgBox "Proszę wybrać klienta z listy!", _
            vbInformation, conKom
        Exit Sub
    End If
    ' zapytanie, czy poprawna czynność
    i = MsgBox("Masz zamiar zarejestrować zwrot kredytu " & _
        "dla" & vbCrLf & Me.cboKlient.Column(1) & vbCrLf & _
        "Czy jest to poprawne?", _
        vbInformation + vbYesNo, conKom)
    ' jeżeli nie, to wyjście z procedury
    If i = vbNo Then Exit Sub
    ' poprawnie, będzie aktualizacja
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    txt = "update tKredyt set DataRZ = '" & _
        Format(Now(), "yyyy-mm-dd") & "' where " & _
        "Id_kredyt = " & Me.cboKlient.Column(5)
    con.Execute txt
    ' zamknięcie formularza
    DoCmd.Close acForm, "frmZwrotKredytu", acSaveNo
    ' sprzątanie po sobie
    con.Close
    Set con = Nothing
End Sub
```

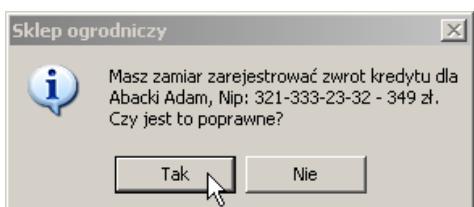
Na zakończenie projektowania jeszcze uwaga odnośnie pól tekstowych, z uwagi na charakter informacyjny zostały one zablokowane przed ingerencją użytkownika bazy danych. Jak pamiętamy, pola te są niezwiązane (nie mają odniesienia do źródła danych), tym samym wszelkie zmiany ich zawartości nie mają sensu, stąd blokowanie.

300

Poniżej kilka zrzutów ekranowych tego formularza w trakcie rejestrowania zwrotu kredytu dla wybranego klienta (jego identyfikator to 2).

The first screenshot shows a dropdown menu under the 'Klient:' label with two options: 'Abacki Adam, Nip: 321-333-23-32 - 349 zł.' and 'Pacanowski Zygmunt, Nip: 234-345-45-55 - 405 zł.'. The second screenshot shows the dropdown has been selected, showing the same two options. Both screenshots also show empty fields for 'Kwota kredytu:', 'Udzielony dnia:', and 'Data zwrotu:', and a 'Zarejestruj zwrot' button.

Pytanie o potwierdzenie operacji.



A to już efekt działania procedury, w polu DataRZ wpisana data splaty kredytu.

A screenshot of a database table window titled 'tKredyt : Tabela'. The table has columns: ID_Kredyt, ID_K, Kwota, DataU, DataZ, and DataRZ. The data shows three rows of credits, with the last row being a new record being inserted. The 'DataRZ' column for the new record is currently empty. The status bar at the bottom shows 'Rekord: 3 z 3'.

	ID_Kredyt	ID_K	Kwota	DataU	DataZ	DataRZ
	1	1	25,00 zł	2005-10-10	2005-11-23	2005-12-02
	4	4	405,00 zł	2005-11-10	2005-12-01	
▶	5	2	349,00 zł	2005-11-11	2005-12-02	2005-12-03
*	(numerowanie)			2005-12-03	2005-12-24	

6.5. Raporty

6.5.1 Dzienny raport kasowy

Na zakończenie każdego dnia sprzedaży może być konieczne przygotowanie pisemnego zestawienia każdej dokonanej transakcji. Źródłem danych dla tak pomyślanego zestawienia będzie kwerenda wybierająca wykorzystującą takie tabele jak:

- tFaktura - potrzebna dla ustalenia dnia zakupu,
- tFakturaDetale – dostarczy danych o identyfikatorze magazynowym produktu i zakupionej ilości,
- tMagazyn – dostarczy danych o cenie sprzedaży, cenie zakupu i wysokości marży,
- tNazwa – dostarczy nazwę produktu,
- tRodzajOpakowania – dostarczy nazwę opakowania (jednostkę) produktu.

Kwerenda na potrzeby raportu powinna zwrócić takie informacje jak:

- Nazwę produktu
- Rodzaj opakowania
- Ilość
- Cenę jednostkową sprzedaży
- Stawkę podatku VAT
- Wysokość marży
- Wartość brutto zakupionej ilości produktu
- Kwotę podatku VAT w wartości brutto
- Wartość netto zakupionej ilości produktu
- Przychód firmy z tytułu marży

Kwerenda qRaportDziennySprzedazy została tak zaprojektowana, aby zwróciła podane wyżej informacje. Poza polami ze źródeł danych zdefiniowano w niej kilka pól wyliczanych:

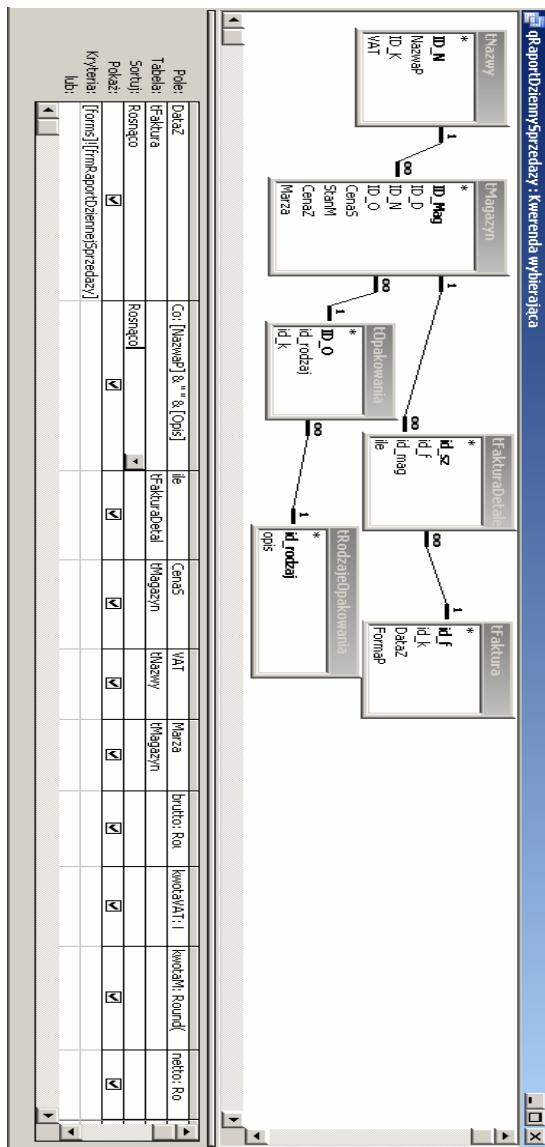
```

brutto: Round(CCur([ile]*[CenaS]);2)
kwotaVAT: Round(CCur([cenaZ]*[Vat]*[ile]);2)
netto: Round(CCur([ile]*[Cenaz]);2)
kwotaM: Round(CCur([ile]*[cenaZ]*[marza]);2)

```

Tutaj pojawi się pewien problem, mianowicie suma kwot podatku Vat, wartości netto i kwoty wynikającej z marży może być większa niż wartość brutto. Jest to konsekwencją przyjętej przez nas zasady zaokrąglania ceny sprzedaży na etapie rozliczania zamówień. Problem można rozwiązać w ten sposób, że rzeczywisty przychód firmy z tytułu marży wyliczymy jako różnicę między wartością brutto a kwotą podatku VAT i wartością netto zakupu (co oznacza, że koszty zaokrągleń zmniejszą nasz przychód). Rzeczywisty

przychód firmy z marży handlowej wyliczymy bezpośrednio w dziennym raporcie sprzedaży.



Pole `Co` tej kwerendy zostało skomponowane z nazwy produktu i nazwy opakowania, pole to zostało także wykorzystane do sortowania rekordów.

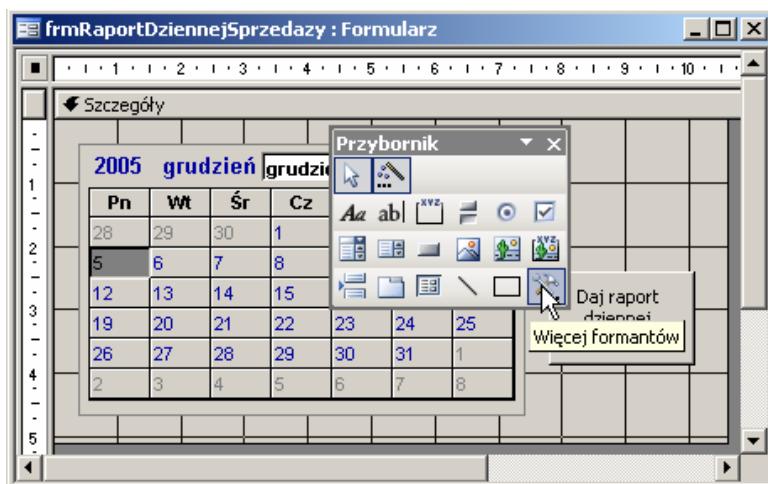
Pole `DataZ` zostało wykorzystane do zdefiniowania kryterium ograniczającego liczbę rekordów do tych, które dotyczą konkretnej daty. Wyboru daty będziemy dokonywać w formularzu `frmRaportDziennejSprzedazy` w kontrolce typu kalendarz, stąd w wierszu kryterium wyrażenie:

```
[forms]![frmRaportDziennejSprzedazy]![Calendar1]
```

gdzie `Calendar1` jest nazwą kontrolki kalendarza.

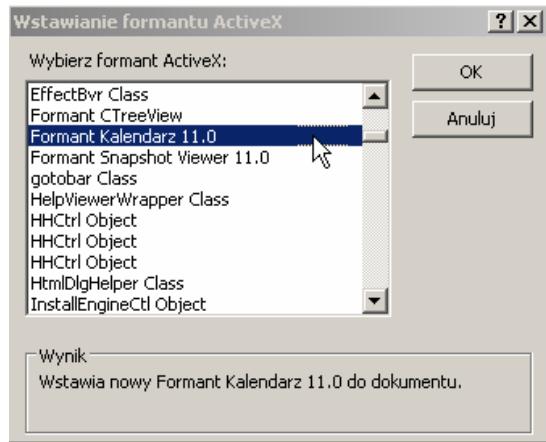
Przejdziemy teraz do zaprojektowania formularza pozwalającego na wybór dnia do przygotowania dziennego raportu sprzedaży.

Otwieramy projekt nowego formularza **bez** wskazywania jego źródła danych. W sekcji *Szczegóły* chcemy teraz osadzić kontrolkę kalendarza, jednak nie znajdziemy tej kontrolki w zestawie kontrolek standardowych. Radzimy sobie w ten sposób, że szukamy możliwości dodania dodatkowych kontrolek, w przypadku MS Access 2003 odpowiedni przycisk znajdziemy w przyborniku (pokazany poniżej na tle projektu formularza).



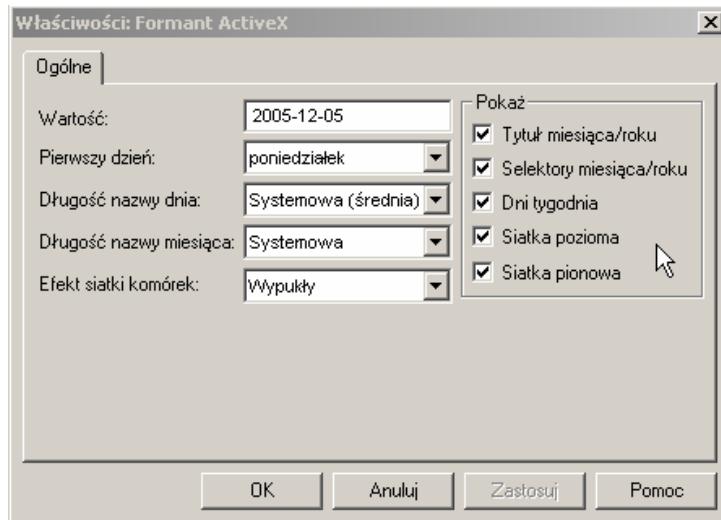
Klik tego przycisku wyświetla listę dodatkowych formantów ActiveX, odszukujemy na tej liście potrzebny formant, selekcjonujemy go myszą i rysujemy formant w projekcie formularza czy raportu.

Inna możliwość to skorzystanie z menu *Wstaw*, gdzie znajdziemy polecenie *Formant ActiveX*, jego uruchomienie spowoduje wyświetlenie pokazanego niżej okna dialogowego. Dalsze postępowanie jest podobne, odszukujemy potrzebny formant, zaznaczamy i przyciskiem OK wracamy do projektu, gdzie osadzamy wybrany formant.



Formant kalendarza może wymagać dodatkowego formatowania polegającego na zmianie czcionki tytułu (z reguły jest wypisany zbyt dużym rozmiarem) czy sposobu formatowania daty. Mamy dostęp do tych elementów poprzez okno właściwości tego formantu.

Właściwość *Value* daje dostęp do dodatkowego okna właściwości formantu kalendarza, gdzie można ustawić co i w jaki sposób ma być wyświetlane w kalendarzu. W pokazanej sytuacji zmieniony został sposób prezentowania nazw miesięcy tak, aby były to pełne nazwy, a nie skróty.



W sekcji *Szczegóły* umieściliśmy także przycisk polecenia cmdDajRaport, zadaniem procedury uruchamianej tym przyciskiem będzie pilnowanie, aby wybrana data nie była datą z przeszłości, a w sytuacji jej poprawnego określenia procedura ma uruchomić raport dziennej sprzedaży.

```
Private Sub Polecenie2_Click()
    If CDate(Me.Calendar1) > Format(Now(), "yyyy-mm-dd") Then
        MsgBox "Źle wybrana data (większa od daty bieżącej)!", _
            vbCritical, conKom
        Me.Calendar1.SetFocus
        Me.Calendar1 = Format(Now(), "yyyy-mm-dd")
        Exit Sub
    End If
    DoCmd.OpenReport "rapDziennySprzedazy", acViewPreview
    DoCmd.Close acForm, "frmRaportDziennejSprzedazy", acSaveNo
End Sub
```

W procedurze tej zastosowano funkcję konwertującą CDate(wyrażenie w wyrażeniu warunkowym badającym, czy wybrana w formancie kalendarza data nie jest przypadkiem datą z przeszłości. Gdyby tak było, to procedura zwróci stosowny komunikat, przypisze do formantu datę bieżącą i powróci do formularza.



W pokazanej sytuacji wybrana jest data z listopada, klik przycisku polecenia uruchomi raport *rapDziennySprzedazy* wykorzystujący jako źródło danych kwerendę *qRaportDziennySprzedazy*.

Projekt takiego raportu oraz jego widok w podglądzie wydruku pokazany jest na kolejnej stronie. Z uwagi na dość dużą liczbę pól raport ten zbudowaliśmy w poziomym układzie strony.

RapDziennySprzedazy : Raport

= "Raport dzienny sprzedaży z dnia "& [DataZ]

Lp.	Produkt i opakowanie	ile:	Cena jedn.	VAT:	Kwota brutto	Kwota VAT:	Kwota netto	Kwota namięty
=1	C0	ile:	CenaS	VAT:	brutto	VAT:	netto	=brutto] [kwot]
Strona :> [Page] & "z" & [Pages]								
Razem dzienny obrót								
=Suma(brutto)								
=Suma(kwot)								
=Suma(netto)								
=[tekst123] [t1]								

RapDziennySprzedazy : Raport

Raport dzienny sprzedaży z dnia 2005-11-08

Lp.	Produkt i opakowanie	ile:	Cena jedn.	VAT:	Marcza	kwota brutto:	kwota VAT:	kwota netto	kwota namięty
1	Jakon Cortland 100 sztuk	1	049,00 zł	22%	7,5%	1 049,00 zł	189,16 zł	800,00 zł	59,34 zł
2	Gocia 0,5 kg	2	2,57 zł	7%	20,1%	5,14 zł	0,34 zł	4,00 zł	0,80 zł
3	Roundap 20 kg	6	229,00 zł	7%	12,6%	1 374,00 zł	89,88 zł	1 139,94 zł	144,17 zł
4	Syl 3 kg	2	57,80 zł	7%	20,0%	115,60 zł	7,56 zł	90,00 zł	18,04 zł
5	Miedzian 1,5 kg	6	42,37 zł	7%	20,0%	254,22 zł	16,63 zł	198,00 zł	39,59 zł
6	Wismutuówka 10 sztuk	2	129,00 zł	22%	17,5%	258,00 zł	46,52 zł	180,00 zł	31,47 zł
Razem dzienny obrót									
3 055,96 zł									
350,11 zł									
2 411,94 zł									
293,91 zł									

Strona: 1 / 1

6.5.2 Rozliczanie podatku VAT

Jednym z ważnych raportów, które powinniśmy przygotować, jest zbiorcze zestawienie sprzedaży w danym miesiącu z wyliczeniem kwoty podatku VAT z rozbiciem na część już zapłaconą (w momencie zakupu produktów do naszego sklepu zapłaciliśmy tę część podatku VAT, która wynikała z wartości netto produktu i stawki podatku VAT) oraz tę, którą musimy odprowadzić do odpowiedniego urzędu. Dobrze byłoby także znać wielkość przychodów netto z tytułu marży handlowej.

W ramach prac nad takim zestawieniem musimy przygotować:

- formularz pozwalający na wybór roku i miesiąca, dla którego przygotowujemy zestawienie sprzedaży,
- źródło danych dla raportu wyliczające dla wybranego miesiąca stosowane dane w ujęciu dziennym (wartość brutto sprzedaży, zapłacony podatek VAT, kwota podatku VAT do zapłaty, wartość netto sprzedaży, kwota przychodu z tytułu marży),
- raport zbiorczy sprzedaży.

Zacznijmy od przygotowania niezwiązanego formularza o nazwie docelowej frmRaportMiesiecznejSprzedazy, jego zadaniem będzie umożliwienie wyboru roku i miesiąca oraz wywołanie raportu. Wybór roku i miesiąca musi być poprzez pola kombi, przy czym pola te muszą być tak zdefiniowane, aby nie było możliwe wybranie miesiąca, dla którego nie było sprzedaży. Podobnie wywołanie raportu może być dopiero w tym momencie, gdy wybrany jest rok i miesiąc.

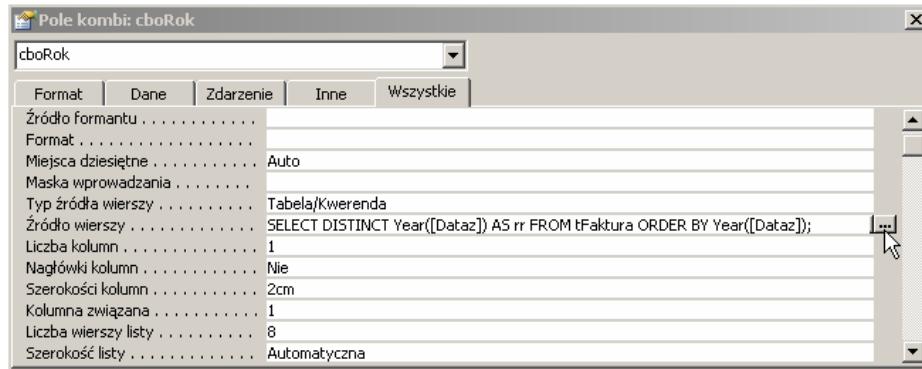
Budowę tego formularza zaczynamy od otwarcia nowego projektu bez wskazywania jego źródła danych, we właściwościach formularza wyłączamy pokazywanie pasków pionowego i poziomego przewijania, zostawiamy tylko przycisk zamknięcia formularza, nie pokazujemy przycisków nawigacyjnych.

W kolejnym kroku umieszczamy (w sekcji szczegóły) dwa pola kombi o nazwach cboRok i cboMiesiac oraz przycisk polecenia o nazwie cmdDajRaport. Zmieniamy odpowiednio tytuły etykiet opisujących pola kombi oraz tytuł przycisku.

Przejdzmy teraz do przygotowania procedur obsługujących nasz formularz, a zacznijmy od procedury uruchamianej w momencie otwarcia formularza, jej zadaniem będzie zablokowanie dostępu do pola cboMiesiac oraz przycisku cmdDajRaport. Chodzi po prostu o to, aby wymusić na użytkowniku właściwą sekwencję działań (wybór roku, następnie miesiąca i wywołanie raportu).

```
Private Sub Form_Open(Cancel As Integer)
    Me.cboMiesiac.Enabled = False
    Me.cmdDajRaport.Enabled = False
End Sub
```

Musimy teraz przygotować źródło danych dla pola kombi cboRok, oczywiście będą to te lata, które odpowiadają datom zakupu z tabeli tFaktura. W pokazanym niżej oknie właściwości tego pola widoczne jest zapytanie SQL zwracające potrzebne informacje, dyrektywa Distinct zapewnia unikalność lat. Zapytanie to można było zdefiniować „ręcznie”, można także skorzystać z graficznego kreatora instrukcji SQL (dostępny z przycisku z trzema kropkami z wiersza właściwości *Źródło wierszy*).



W momencie wyboru roku z pola kombi cboRok musi być uruchomiona procedura dostarczająca listę dostępnych miesięcy do pola kombi cboMiesiac, jednocześnie trzeba będzie udostępnić to pole dla użytkownika. Zadanie to wykona kolejna procedura zdarzeniowa, tym razem przypisana do zdarzenia *Po aktualizacji* pola cboRok.

W procedurze tej budujemy rekordset zwracający numer miesiąca z pola DataZ tabeli tFaktura, ale tylko z tych rekordów, dla których rok zakupu jest zgodny z tym, który został wybrany w polu cboRok. Klauzula (dyrektywa) Distinct zapewnia unikalność numerów miesięcy. W pętli z licznikiem przebiegającej po wszystkich rekordach rekordsetu rst budowana jest zmienna tekstowa txt zawierająca nazwy miesięcy (wykorzystana jest funkcja MonthName) oddzielone średnikiem. Zmienna ta zostanie następnie przypisana właściwości RowSource pola kombi cboMiejscowosc. Poprawne funkcjonowanie pokazanej niżej procedury wymaga zmiany właściwości *Typ źródła Wierszy* pola kombi cboMiesiec z domyślnej Tabela/kwerenda na Lista wartości.

```
Private Sub cboRok_AfterUpdate()
    Dim con As ADODB.Connection, rst As ADODB.Recordset, _
        txt As String
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    txt = "SELECT DISTINCT Month(DataZ) AS mm " & _
        " FROM tFaktura WHERE Year(DataZ) = " & cboRok & _

```

```

    " ORDER BY Month(DataZ)"
Set rst = New ADODB.Recordset
rst.Open txt, con, adOpenKeyset, adLockReadOnly
' wykorzystuje właściwość RowSource pola kombi cboMiesiac
txt = ""
For i = 1 To rst.RecordCount
    txt = txt & MonthName(rst!mm) & ";"
    rst.MoveNext
Next i
Me.cboMiesiac.RowSource = txt
' pole cboMiesiac ma stosowne dane, sprzątamy po sobie
rst.Close
con.Close
Set rst = Nothing
Set con = Nothing
' udostępniamy pole kombi
Me.cboMiesiac.Enabled = True
End Sub

```

Po wyborze miesiąca w polu kombi `cboMiesiac` musimy udostępnić przycisk polecenia `cmdDajRaport`, zrobi to procedura zdarzenia *Po aktualizacji* tego pola.

```

Private Sub cboMiesiac_AfterUpdate()
    Me.cmdDajRaport.Enabled = True
End Sub

```

Pozostało nam jeszcze napisanie procedury obsługującej zdarzenie Przy kliknięciu przycisku `cmdDajRaport` i formularz jest gotowy.

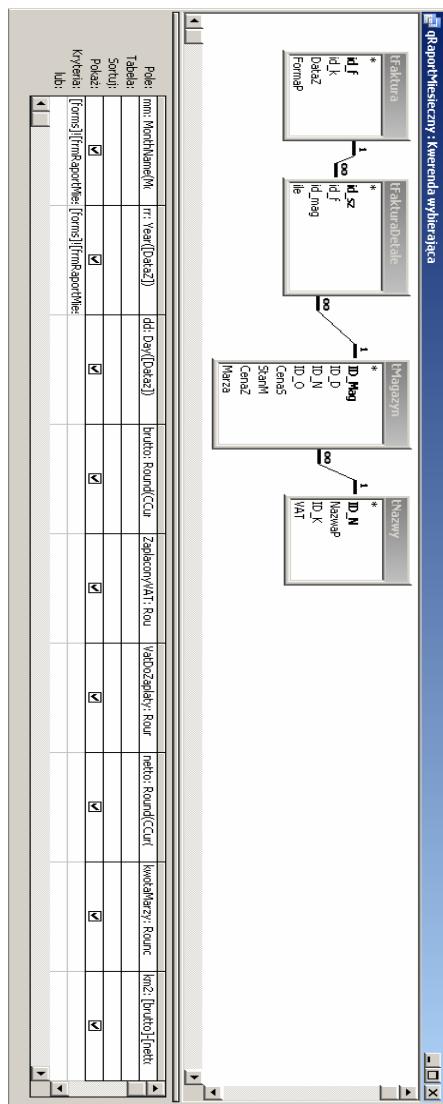
```

Private Sub cmdDajRaport_Click()
    DoCmd.OpenReport "rapRaportSprzedazyMiesiecznej", _
        acViewPreview
    DoCmd.Close acForm, "frmRaportMiesiecznejSprzedazy", _
        acSaveNo
End Sub

```

Polecenie `DoCmd` otwiera w widoku podglądu wydruku raport zbiorczy miesięcznej sprzedaży o nazwie `rapRaportSprzedazyMiesiecznej`, my oczywiście jeszcze tego raportu nie mamy, ale za moment, po zbudowaniu źródła danych, taki raport zaprojektujemy. Drugie polecenie `DoCmd` wywołuje metodę `Close` zamkującą formularz wyboru roku i miesiąca dla sporządzenia raportu zbiorczego. Oczywiście zmiana kolejności tych poleceń nie wchodzi w rachubę, po prostu formularz **musi** być otwarty w momencie wywoływania raportu, ponieważ dostarcza informacji o wybranym miesiącu (i roku) do źródła danych raportu.

Dla zbudowania źródła danych dla raportu sprzedaży przygotujemy dwie współpracujące z sobą kwerendy. Pierwsza z nich ograniczy liczbę rekordów i obliczy wszystkie potrzebne wielkości dla każdego sprzedanego produktu w wybranym miesiącu, druga dokona podsumowań grupując dane wg dni sprzedaży. Projekt pierwszej z kwerend pokazany jest poniżej.



Dwa pierwsze pola tej kwerendy to pola wyliczane wykorzystane do ograniczenia liczby rekordów do tych, dla których DataZ z tabeli tFaktyry odpowiada wybranym wartościom w formularzu frmRaportSprzedazyMiesiecznej:

```
mm: MonthName(Month([DataZ]))
rr: Year([DataZ])
```

z kryteriami odpowiednio:

```
[forms]![frmRaportMiesiecznejSprzedazy]![cboMiesiac]
[forms]![frmRaportMiesiecznejSprzedazy]![cboRok]
```

Dalsze pola to także pola wyliczane, z wyjątkiem pola zwracającego numer dnia pozostałe wyliczają wartości sprzedaży wg założonych wcześniej założeń. Pola te wykorzystują funkcję konwertującą CCur (konwertuje wyrażenie do formatu waluty) oraz funkcję zaokrąglającą Round.

```
dd: Day([Dataz])
brutto: Round(CCur([ile]*[CenaS]);2)
ZaplaconyVAT: Round(CCur([ile]*[CenaZ]*[Vat]);2)
VatDoZaplaty: Round(CCur([ile]*[cenaZ]*[marza]*[vat]);2)
netto: Round(CCur([ile]*[CenaZ]);2)
kwotaMarzy: Round(CCur([ile]*[cenaZ]*[marza]);2)
km2: [brutto]-[netto]-[ZaplaconyVat]-[VatDoZaplaty]
```

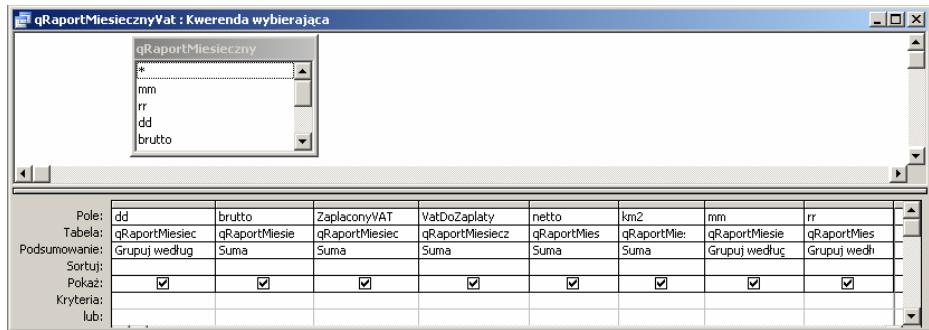
Wyjaśnienia wymaga pole km2 (skrót od „kwota marży dwa”), formalnie między wyliczonymi polami powinna zachodzić równość

$$\text{brutto} = \text{ZaplaconyVAT} + \text{VatDoZaplaty} + \text{netto} + \text{kwotaMarzy}$$

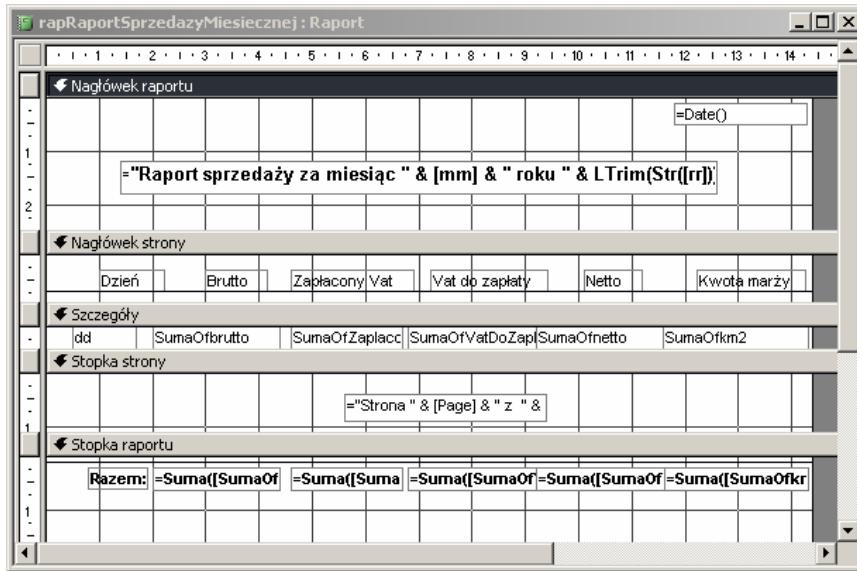
ale z uwagi na stosowane zaokrąglenia może się zdarzyć, że dla danej pozycji zakupu ten warunek nie będzie spełniony. Pole km2 rozwiązuje ten problem wyliczając kwotę marży nie według formalnego wzoru (jak w definicji pola kwotaMarzy), ale jako różnicę między pozycją brutto a pozostałymi składnikami. Inaczej mówiąc wszelkie różnice z tytułu zaokrągleń „idą” w koszty naszej firmy. W raporcie zostanie wykorzystane pole km2 do zwrócenia przychodów firmy z tytułu marży.

Zbudujemy teraz kolejna kwerendę, jej źródłem danych będzie omówiona wcześniej kwerenda qRaportMiesiecznejSprzedazy, a będzie to kwerenda sumująca (grupującą) wartości sprzedaży brutto, netto, kwotę podatku VAT (zapłaconego i do zapłaty) oraz kwotę przychodu z tytułu marży. Dla tych pól została wykorzystana funkcja Suma, a grupowanie dotyczy pola zwracającego numer dnia. Do kwerendy tej zostały

włączone także pola `mm` (nazwa wybranego miesiąca) oraz `rr` (wybrany rok), także z opcją grupowania – po prostu wartości te będą nam potrzebne do opisania tytułu naszego raportu.

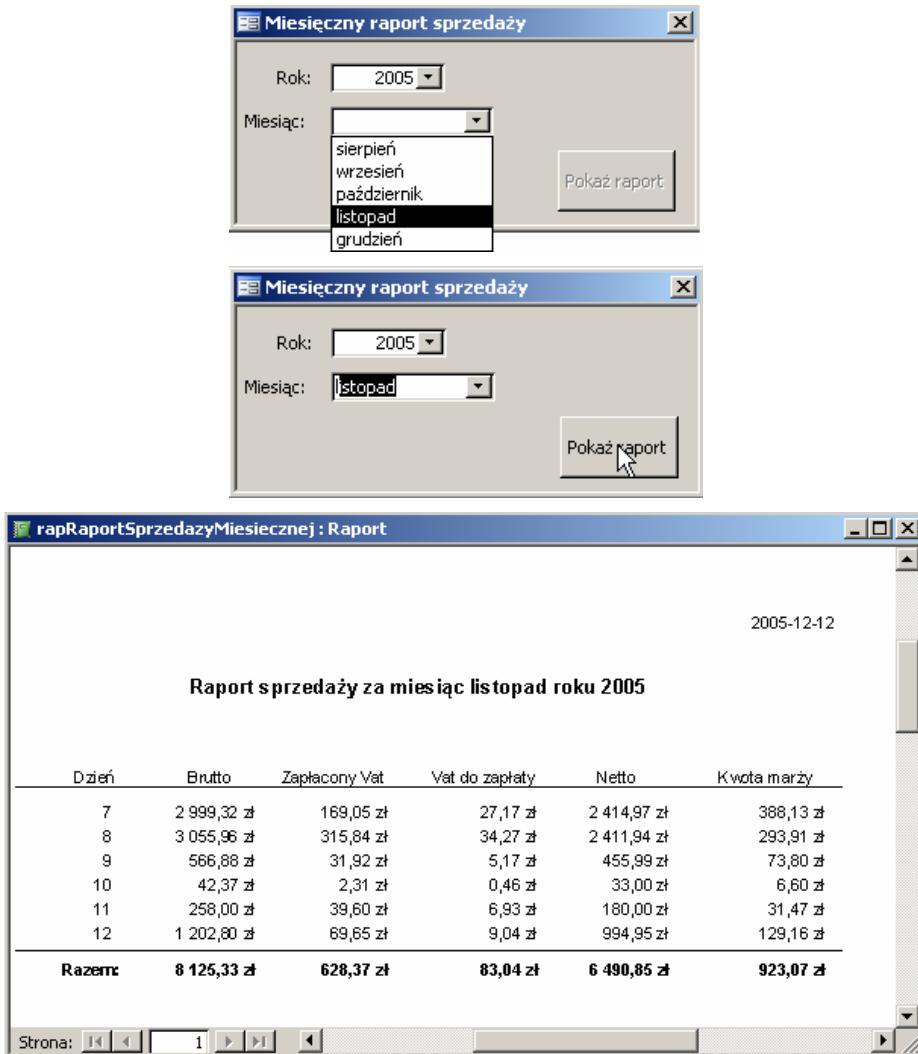


Kwerenda `qRaportMiesiecznyVat` będzie źródłem danych dla raportu sprzedaży miesięcznej, projekt tego raportu jest pokazany niżej.



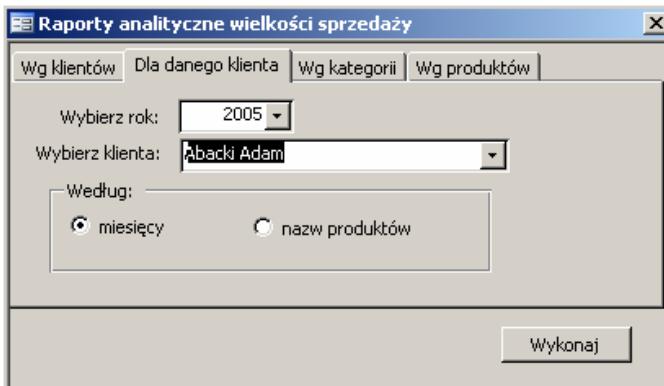
W sekcji *Szczegóły* zostały umieszczone pola tekstowe źródła danych (z wyjątkiem pól `mm` i `rr`), a ich etykiety zostały umieszczone w nagłówku strony. Pola zwracające informacje finansowe (wszystkie poza `dd`) zostały sformatowane na format waluty z dwoma miejscami dziesiętnymi. Pola `mm` i `rr` zostały wykorzystane w niezwiązanych polach tekstowych do zbudowania tytułu raportu. W sekcji *Stopka raportu* umieszczeno pola tekstowe sumujące dane z kolumn raportu.

Poniżej widok formularza frmRaportMiesiecznejSprzedazy w trakcie wyboru miesiąca do sporządzenia raportu i sam raport sprzedaży za wybrany miesiąc.



6.5.3 Analiza wielkości obrotów wg różnych kryteriów

Na zakończenie rozdziału poświęconego raportom w bazie SklepOgrodnicy zjeszcze kilka prostych raportów analizujących wielkość sprzedaży w wybranym roku według różnych ujęć. Dla wyboru odpowiedniego raportu zbudujemy niezwiązany formularz z formantem karta, taki, jakim pokazany jest poniżej.



Zakładki tego formantu pozwalają na wybór określonego raportu, a kontrolki umieszczone na wybranej stronie formantu precyzuje sposób przygotowania danych źródłowych dla odpowiedniego raportu.

W pokazanej sytuacji przewidziano przygotowanie czterech różnych grup raportów analitycznych:

Wg klientów – zestawienie wielkości zakupów brutto oraz kwoty przychodów naszego sklepu z tytułu marży dla poszczególnych klientów w ujęciu miesięcznym,

Dla danego klienta – zestawienie ilościowe i wartościowe (brutto) zakupów poszczególnych produktów w poszczególnych miesiącach lub wg nazw produktów,

Wg kategorii – zbiorcze zestawienie wartości brutto zakupów i przychodów z tytułu marży w ujęciu miesięcznym dla poszczególnych kategorii produktów,

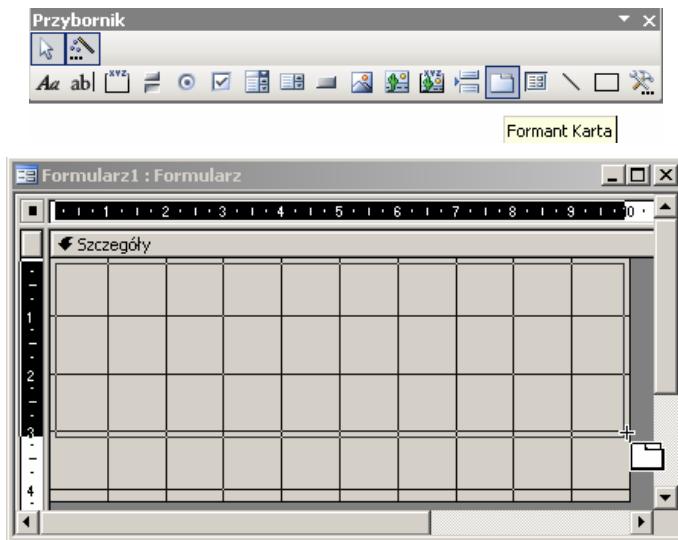
Wg produktów - zbiorcze zestawienie wartości brutto zakupów i przychodów z tytułu marży w ujęciu miesięcznym dla poszczególnych produktów.

Dla każdego z tych raportów będziemy musieli przygotować odpowiednie źródło danych, w każdym przypadku będzie to zapytanie z parametrem, którego wartość zostanie przekazana do zapytania z odpowiedniego formantu pokazanego wyżej formularza.

Kolejno prześledzimy budowanie odpowiednich raportów, zawsze zaczynając od omówienia jego strony (zakładki) w formancie karta pokazanego formularza, poprzez przygotowanie źródła danych i projektu raportu.

Zaczynamy od otwarcia projektu nowego, niezwiązanego formularza, który zapiszemy w kolekcji formularzy pod nazwą frmRozneAnalizySprzedazy. W oknie właściwości tego formularza zmieniamy ustawienia kilku z nich tak, aby nie pokazywać przycisków Min i Max okna formularza, obu jego pasków przewijania, ani też przycisków nawigacyjnych. Dodajemy także odpowiedni tytuł formularza.

Z przybornika formantów pobieramy formant Karta i rysujemy odpowiedni prostokąt w sekcji szczegółów pamiętając o tym, żeby zostawić od dołu wolny pasek na umieszczenie przycisku polecenia, który ma być wspólnym formantem dla wszystkich stron formantu Karta.



Po zwolnieniu myszy formant Karta zawiera dwie zakładki (Strony) o domyślnych nazwach (i tytułach) Strona z odpowiednim numerem. Z menu kontekstowego tego formantu mamy dostęp do wielu ważnych poleceń, między innymi do polecenia *Wstaw stronę* – wywołujemy je dwukrotnie, w efekcie mamy cztery strony w formancie karta. Korzystając z okna właściwości dla wybranej strony możemy zmienić jej tytuł, w naszym przypadku mogą to być takie tytuły, jakie zaproponowaliśmy wcześniej dla poszczególnych grup raportów.

Poniżej formantu karta umieszczamy jeszcze przycisk polecenia o nazwie cmdWykonaj, jego zadaniem będzie wyświetlenie wybranego raportu.

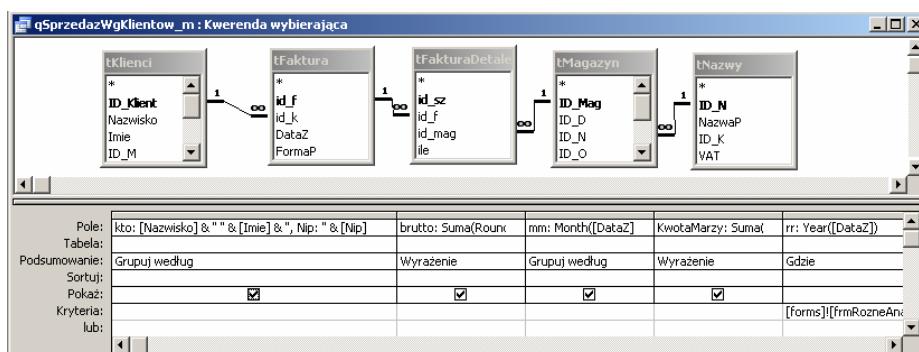
Przejdziemy teraz do pierwszej ze stron formantu karta, z poziomu VBA strona ta będzie rozpoznawana pod numerem 0 (kolejne 1, 2 itd.). Zadaniem tej strony będzie przygotowanie raportu pokazującego, dla wybranego roku, wielkość sprzedaży brutto i wielkość przychodów naszego sklepu z tytułu marży dla naszych klientów w układzie miesięcznym.

Takie określenie zakresu zwracanych informacji wymaga od nas umieszczenia na powierzchni tej strony pola kombi pozwalającego na wybór roku do analizy. Ustawiamy taki formant nadając mu nazwę cboRokS1 (będziemy umieszczać podobny formant także na innych stronach, stąd S1 dla rozróżnienia strony, na której pole kombi umieszczone).

Jako źródło wiersza dla tego formantu definiujemy zapytanie typu select postaci jak niżej:

```
SELECT DISTINCT Year([DataZ]) AS rok FROM tFaktura ORDER BY Year([DataZ]);
```

Możemy już przygotować kwerendę zwracającą potrzebne dane dla tak pomyślanego raportu, jej projekt pokazany jest niżej.



Jest to typowa kwerenda wybierająca z włączoną opcją grupowania dla wyliczanego pola `kto` (zwracającego nazwisko klienta i jego numer NIP) oraz pola `mm` zwracającego numer miesiąca dokonania zakupu przez danego klienta.

```
kto: [Nazwisko] & " " & [Imie] & ", Nip: " & [Nip]
mm: Month([DataZ])
```

Pole wyliczane `rr:Year([data])` zwraca numer roku dokonania zakupu, pole to wykorzystamy do ograniczenia liczby zwracanych rekordów poprzez przekazanie do wiersza `Kryteria` tego pola numeru roku wybranego w polu kombi `cboRokS1` (stąd klauzula `Gdzie` w wierszu `Podsumowanie` tego pola):

```
[forms]![frmRozneAnalizySprzedazy]![cboRokS1]
```

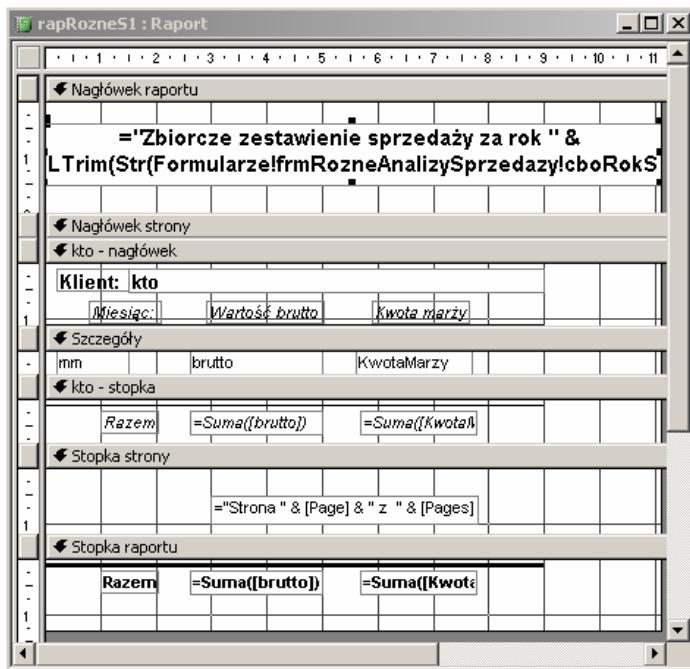
Pozostałe dwa pola zostały tak zdefiniowane, aby zwracały sumaryczną wartość zakupów brutto oraz kwotę marży dla klientów i miesięcy wybranego roku.

```
brutto: Suma(Round(CCUR([ile]*[CenaS]));2))
KwotaMarzy: Suma(Round(CCUR([ile]*[CenaZ]*[Marza]));2))
```

Funkcja agregatująca (Suma) została umieszczona w definicji pola, stąd w wierszu *Podsumowanie* obu pól klauzula *Wyrażenie* (a nie spodziewana *Suma*).

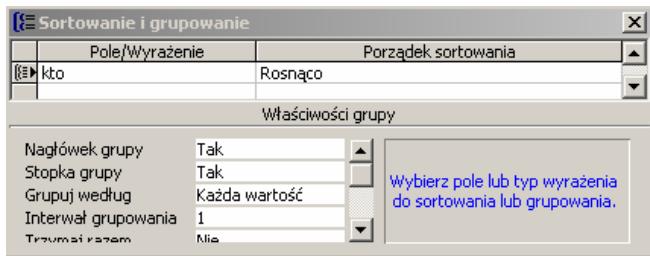
Tak zdefiniowana kwerenda została zapisana w kolekcji kwerend pod nazwą *qSprzedazWgKlientow_m* do dalszego wykorzystania jako źródło danych dla raportu, który będziemy teraz projektować.

Projekt takiego raportu (o nazwie *rapRozneS1*) pokazany jest niżej, w projekcie uaktywniona została opcja *Sortowanie i grupowanie* wg pola *kto* po to, aby dla każdego z naszych klientów wyprowadzić wartość ich zakupów w poszczególnych miesiącach wybranego roku wraz ze stosownym podsumowaniem.



Opcja *Sortowanie i grupowanie* dostępna jest w menu *Widok* lub w menu kontekstowym raportu, w naszym przypadku do grupowania rekordów wykorzystane zostało pole *kto* zawierające dane klienta z sortowaniem rosnącym oraz z włączonym nagłówkiem i stopką grupy.

W nagłówku grupy umieścimy pole *kto* (dane klienta) oraz etykiety pól umieszczonych w sekcji *Szczegóły* raportu. W stopce grupy umieścimy wiersz podsumowania dla danego klienta, czyli roczną wartość zakupów brutto oraz naszych „korzyści” z tytułu marży.



W sekcji *Nagłówek raportu* umieściliśmy niezwiązane pole tekstowe, źródłem danych dla tego formantu jest wyrażenie odwołujące się do pola cboRokS1:

```
= "Zbiorcze zestawienie sprzedaży za rok " &
  LTrim(Str(Formularze!frmRozneAnalizySprzedazy!cboRokS1))
```

W sekcji *Stopka raportu* umieszczamy jeszcze wiersz podsumowań (roczna wartość sprzedaży brutto i roczna kwota przychodów z tytułu marży), w *stopce strony* pole tekstowe zwracające numer strony, formatujemy odpowiednio raport i możemy zobaczyć jego przykładową postać (dla roku 2005 i tych danych, które są aktualnie w bazie).

Zbiorcze zestawienie sprzedaży za rok 2005

Klient: Abacki Adam, Nip: 321-333-23-32		
Miesiąc:	Wartość brutto	Kwota marży
8	78,75 zł	11,03 zł
11	4 202,12 zł	517,29 zł
12	4 925,75 zł	546,61 zł
<i>Razem</i>	<i>9 206,62 zł</i>	<i>1 074,93 zł</i>

Klient: Kowalski Jan, Nip: 231-234-45-45		
Miesiąc:	Wartość brutto	Kwota marży
9	9 765,94 zł	857,27 zł
10	2 010,34 zł	187,92 zł
11	258,00 zł	31,48 zł
<i>Razem</i>	<i>12 034,28 zł</i>	<i>1 076,67 zł</i>

Klient: Pacanowski Zygmunt, Nip: 234-345-45-55		
Miesiąc:	Wartość brutto	Kwota marży
11	3 665,21 zł	374,34 zł
<i>Razem</i>	<i>3 665,21 zł</i>	<i>374,34 zł</i>
Razem	24 906,11 zł	2 525,94 zł

Strona: [|] < < 1 > > [|]

Analiza sprzedaży dla wybranego klienta

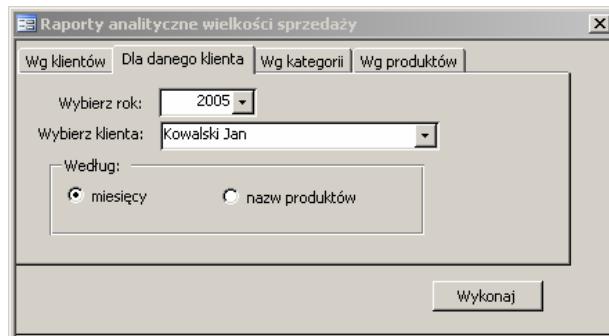
Otwieramy formularz frmRozneAnalizySprzedazy w widoku projektu i uaktywniamy drugą stronę formantu karta. Na powierzchni tej karty umieszczamy pole kombi o nazwie cboRokS2 zdefiniowane analogicznie jak na stronie pierwszej. Bezpośrednio pod nim umieszczamy kolejne pole kombi o nazwie cboKlient, jego zadaniem jest ułatwienie wyboru klienta naszego sklepu spośród tych, którzy dokonywali zakupów w wybranym roku. Właściwość *Typ źródła wierszy* tego formantu ustawiamy na Listę wartości, a właściwość *Źródło wierszy* zdefiniujemy programowo tworząc procedurę zdarzeniową *Po aktualizacji* dla pola kombi cboRokS2.

```
Private Sub cboRokS2_AfterUpdate()
    Dim con As ADODB.Connection, rst As ADODB.Recordset, _
        txt As String
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    txt = "SELECT DISTINCT id_k, [Nazwisko] " & " + ' ' + " & _
        "[Imie] AS kto FROM tKlienci " & _
        "INNER JOIN tFaktura ON tKlienci.ID_Klient=" & _
        "tFaktura.id_k WHERE " & _
        "Year([DataZ])= " & Me.cboRokS2 & _
        " ORDER BY [Nazwisko] " & " + ' ' + " & "[Imie]"
    rst.Open txt, con, adOpenKeyset, adLockReadOnly
    ' tworzymy zmienną txt zawierającą listę wartości
    txt = ""
    For i = 1 To rst.RecordCount
        txt = txt & rst!id_k & ";" & rst!kto & ";"
        rst.MoveNext
    Next i
    ' sprzątamy po sobie
    rst.Close
    con.Close
    Set rst = Nothing
    Set con = Nothing
    ' przypisujemy źródło danych
    Me.cboKlient.RowSource = txt
End Sub
```

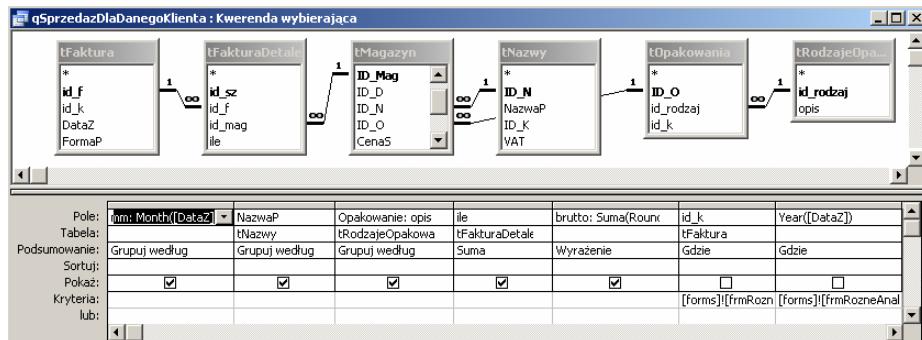
W procedurze tej warto zwrócić uwagę na definicję pola wyliczanego *kto*, tzn. na sposób rozdzielenia spacją dwóch pól tekstowych (*Nazwisko* i *Imie*).

Poprawne funkcjonowanie pola kombi *cboKlient* wymaga jeszcze zmiany takich jego właściwości jak *Liczba kolumn* (na dwie) i szerokości kolumn (odpowiednio 0 cm i 4 cm).

Poniżej pola `cboKlient` umieścimy jeszcze formant typu ramka z dwoma przyciskami opcji. Formanty te odpowiednio formatujemy i opisujemy, tak, aby uzyskać taki wygląd jak pokazany niżej.



Kolejny krok, to przygotowanie kwerendy jako źródła danych dla kolejnego raportu, a najlepiej dla obu raportów (według miesięcy lub nazw produktów). Projekt takiej kwerendy o nazwie `qSprzedazDlaDanegoKlienta` pokazany jest niżej, jest to oczywiście kwerenda wybierająca, z włączoną opcją grupowania i dwoma polami wykorzystywanymi do ograniczenia liczby rekordów.



W przypadku pola `id_k` zwracającego identyfikator klientów w wierszu *Kryteria tego pola* wpisaliśmy wyrażenie:

```
[forms]![frmRozneAnalizySprzedazy]![cboKlient]
```

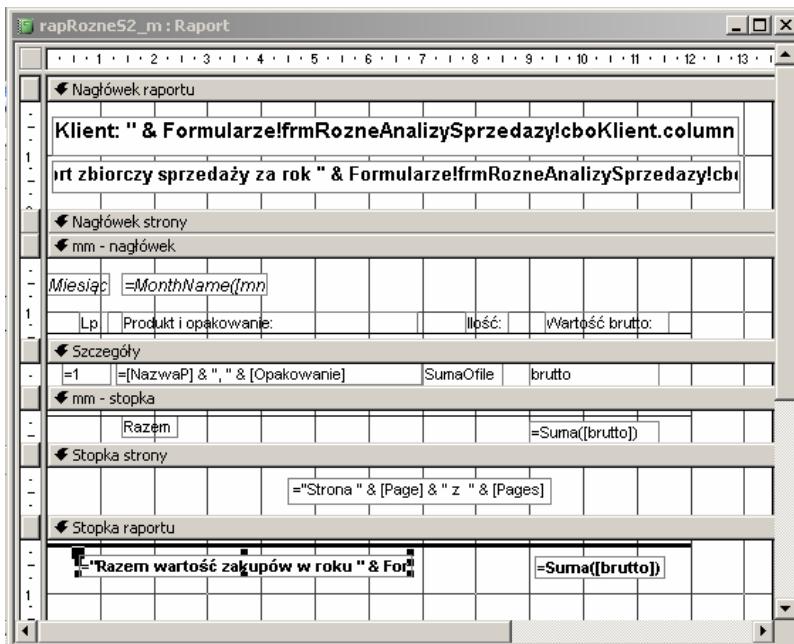
a dla pola wyliczanego `Year ([DataZ])` kryterium definiuje wyrażenie:

```
[forms]![frmRozneAnalizySprzedazy]![cboRokS2]
```

W wierszu *Podsumowanie* włączono klauzulę *Grupuj według* dla wyliczanego pola `mm` zwracającego numer miesiąca oraz pól `NazwaP` (nazwa produktu) i pola

wyliczanego Opakowanie, które tak naprawdę jest równoważne polu opis z tabeli tRodzajeOpakowania. Pole to zostało wprowadzone (Opakowanie **zamiast** opis) z jednego, prozaicznego powodu – w momencie projektowania raportu z wyliczonym polem odwołującym się do pola opis silnik bazy danych zamieniał nazwę tego pola na angielski odpowiednik Description i **nie było innego wyjścia**, jak po prostu ominięcie tego dziwnego zachowania poprzez inną nazwę pola niż opis!

Po przygotowaniu źródła danych przechodzimy do zaprojektowania pierwszego z dwóch przewidzianych raportów. Raport ten ma prezentować ilościowo i wartościowo wielkości zakupów poszczególnych produktów w kolejnych miesiącach wybranego roku. Projekt takiego raportu pokazany jest poniżej.



Jego budowa jest bardzo podobna do poprzedniego raportu, tym razem opcja sortowania i grupowania dotyczy kolejnych miesięcy roku, wykorzystany jest nagłówek i stopka pola grupującego, w nagłówku wykorzystano funkcję MonthName do wyświetlenia nazwy miesiąca.

W nagłówku raportu umieszczono niezwiązane pole tekstowe wyświetlające nazwę klienta pobraną w sposób niejawny z pola kombi cboKlient:

```
= "Klient: " &
Formularze!frmRozneAnalizySprzedazy.cboKlient.column(1)
```

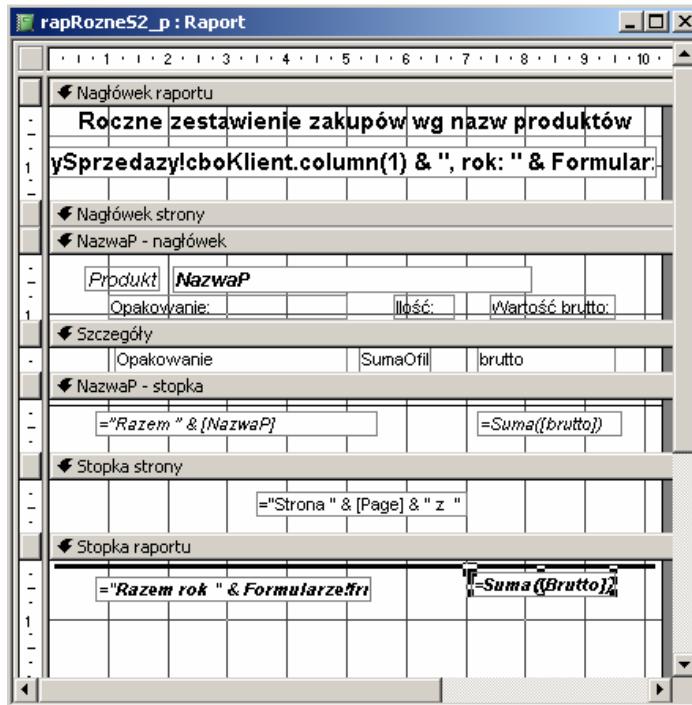
A tak wygląda widok tego formularza dla przykładowego klienta.

RapRozneS2_m : Raport			
Klient: Kowalski Jan			
Raport zbiorczy sprzedaży za rok 2005			
<i>Miesiąc wrzesień</i>			
Lp.	Produkt i opakowanie:	Ilość:	Wartość brutto:
1	Goal, 1 litr	12	362,04 zł
2	Roundap, 20 litrów	23	9 338,00 zł
3	Słonecznik, torba 1 kg	1	65,90 zł
		Razem	9 765,94 zł
<i>Miesiąc październik</i>			
Lp.	Produkt i opakowanie:	Ilość:	Wartość brutto:
1	Goal, 0,5 kg	12	30,84 zł
2	Słonecznik, torba 1 kg	5	329,50 zł
3	Sylit, 20 kg	6	1 650,00 zł
		Razem	2 010,34 zł
<i>Miesiąc listopad</i>			
Lp.	Produkt i opakowanie:	Ilość:	Wartość brutto:
1	Wiśnia Łutówka, 10 sztuk	2	258,00 zł
		Razem	258,00 zł
Razem wartość zakupów w roku 2005			12 034,28 zł
Strona: [] 1 [] [] [] []			

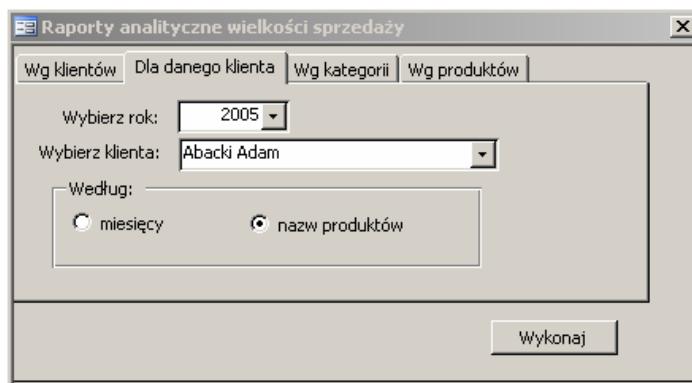
W zakładce „Dla danego klienta” formularza frmRozneAnalizySprzedazy stworzyliśmy możliwość wygenerowania raportu w układzie nazw produktów. Niestety, nie ma prostej możliwości wykorzystania tej samej kwerendy jako źródło danych dla nowego raportu z tej przyczyny, że zawiera ona pole grupujące mm zwracające numer miesiąca.

W tej sytuacji najprostszym rozwiązaniem jest przygotowanie lekko zmodyfikowanej kopii kwerendy qSprzedazDlaDanegoKlienta, w której po prostu usuniemy pole wyliczane mm:MonthName ([Data]). Korzystając ze schowka Windows kopujemy tę kwerendę, a następnie wklejamy pod nazwą np. qSprzedazDlaDanegoK_p. Otwieramy teraz kopię w widoku projektu i usuwamy pole wyliczane mm zapisując dokonane zmiany. Mamy już zmodyfikowane źródło danych i możemy przejść do zaprojektowania nowego raportu opartego na tej kwerendzie.

Widok projektu tego raportu pokazany jest poniżej, jest on bardzo podobny do ostatnio utworzonych dwóch raportów.



Możemy teraz zobaczyć efekt naszej pracy, zaczniemy oczywiście od otwarcia formularza `frmRozneAnalizySprzedazy`, przejścia do zakładki „Dla wybranego klienta”, wybrania roku analizy i przykładowego klienta.



Sam raport musimy jeszcze uruchomić ręcznie (bo nie ma jeszcze procedur obsługujących przycisk polecenia cmdWykonaj).

Roczne zestawienie zakupów wg nazw produktów		
Klient: Abacki Adam, rok: 2005		
Produkt: Goal		
Opakowanie: Ilość Wartość brutto:		
0,25 litra	21	1 213,80 zł
0,5 kg	7	17,99 zł
<i>Razem Goal</i>		1 231,79 zł
Produkt: Medzian		
Opakowanie: Ilość Wartość brutto:		
1,5 kg	29	1 228,73 zł
20 kg	4	392,40 zł
<i>Razem Medzian</i>		1 621,13 zł
Produkt: Roundap		
Opakowanie: Ilość Wartość brutto:		
10 litrów	4	1 100,00 zł
20 kg	20	4 580,00 zł
<i>Razem Roundap</i>		5 680,00 zł
Produkt: Skonecznik		
Opakowanie: Ilość Wartość brutto:		
torebka 1 kg	1	65,90 zł
<i>Razem Skonecznik</i>		65,90 zł
Produkt: Sylit		
Opakowanie: Ilość Wartość brutto:		
20 kg	2	550,00 zł
3 kg	1	57,80 zł
<i>Razem Sylit</i>		607,80 zł
Razem rok 2005		9 206,62 zł

Strona: [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] [72] [73] [74] [75] [76] [77] [78] [79] [80] [81] [82] [83] [84] [85] [86] [87] [88] [89] [90] [91] [92] [93] [94] [95] [96] [97] [98] [99] [100]

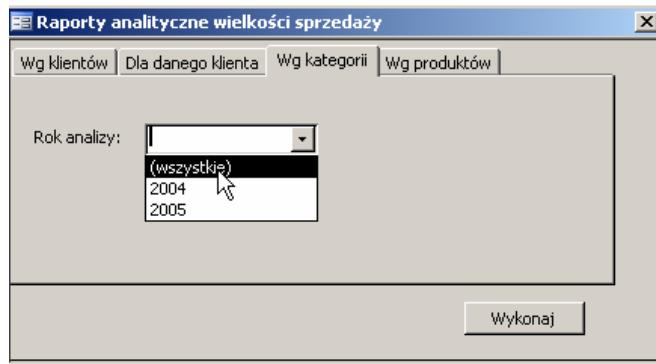
Analiza sprzedaży wg kategorii

Strona trzecia formantu karta formularza frmRozneAnalizySprzedazy została przeznaczona do określenia roku analizy sprzedaży wg kategorii produktów. Inaczej niż w poprzednich dwóch raportach będziemy teraz budować pole kombi pozwalające na wybór roku, ponieważ będziemy chcieli tak to zorganizować, aby raport dotyczył wszystkich lat ogółem lub wybranego roku. Inną konsekwencją takiego założenia będzie także konieczność zróżnicowania źródła danych dla raportu (wszystkie lata sprzedaży lub wybrany rok). Tym razem spróbowujemy rozwiązać nasz problem nie poprzez budowanie kolejnych kwerend jako nazwanych obiektów bazy danych, ale metodami programistycznymi.

Zaczniemy od otwarcia formularza frmRozneAnalizySprzedazy w widoku projektu, gdzie po uaktywnieniu strony trzeciej formantu karta umieszczymy w niej pole kombi o nazwie cboRokS3. Zmieniamy właściwość *Typ źródła wierszy* z domyślnej Tabela/Kwerenda na Lista wartości. Listę tych wartości zbudujemy w procedurze zdarzenia *Przy zmianie* dla formantu karta w odpowiedzi na uaktywnienie strony o indeksie 2 (numeracja stron biegnie od zera).

```
Private Sub FormantKarta0_Change()
    If Me.FormantKarta0 = 2 Or Me.FormantKarta0 = 3 Then
        Dim con As ADODB.Connection, rst As ADODB.Recordset
        Dim txt As String, i As Integer
        Set con = New ADODB.Connection
        Set con = CurrentProject.Connection
        Set rst = New ADODB.Recordset
        txt = "SELECT DISTINCT Year(DataZ) AS rr FROM " & _
            "tFaktura ORDER BY Year(DataZ)"
        rst.Open txt, con, adOpenKeyset, adLockReadOnly
        ' umieszczymy na pierwszej pozycji słowo "(wszystkie)"
        txt = "(wszystkie);"
        ' dodajemy liczby reprezentujące rok sprzedaży
        For i = 1 To rst.RecordCount
            txt = txt & rst!rr & ";"
            rst.MoveNext
        Next i
        Me.cboRokS3.RowSource = txt
        rst.Close
        con.Close
        Set rst = Nothing
        Set con = Nothing
    End If
End Sub
```

Dzięki tej procedurze lista pola kombi cboRokS3 będzie zawierać pozycję symbolizującą wybór wszystkich lat do analizy.



Przejdziemy teraz do zaprojektowania nowego raportu, tym razem bez wskazywania jego źródła danych na etapie projektowania. Zrobimy to programowo w procedurze zdarzenia Przy otwarciu tego raportu. Zanim tę procedurę utworzymy zapisujemy raport pod nazwą `rapRozneS3`.

Zadaniem procedury zdarzenia Przy otwarciu będzie przypisanie do właściwości `RecordSource` raportu odpowiedniego zapytania, które zwróci nazwę kategorii produktu, sumaryczną wartość brutto i sumaryczną kwotę marży z grupowaniem po nazwie kategorii. W zależności od wybranej pozycji w polu kombi `cboRokS3` dodamy lub nie warunek ograniczający liczbę rekordów.

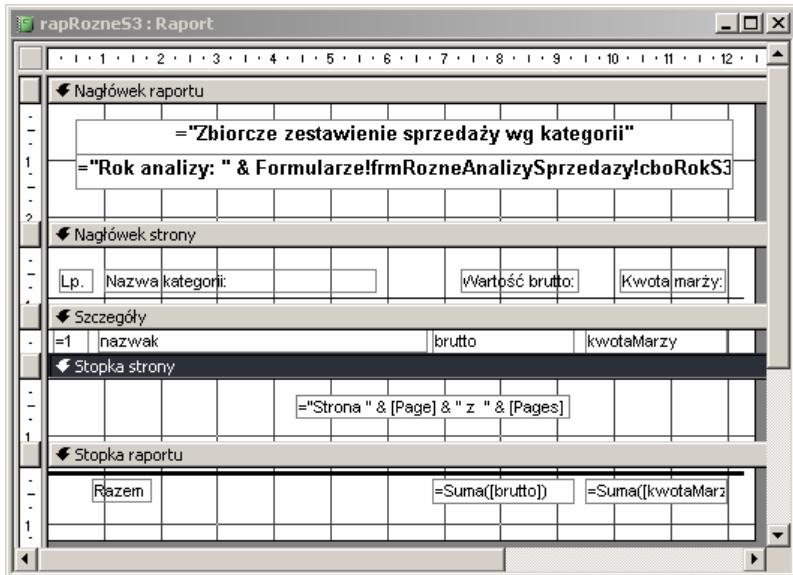
```
Private Sub Report_Open(Cancel As Integer)
    Dim txt As String
    txt = "SELECT tKategorie.NazwaK, " & _
        "Sum(CCurr(tFakturaDetale.ile*tMagazyn.CenaS)) " & _
        "AS brutto, " & _
        "Sum(CCurr(tFakturaDetale.ile*tMagazyn.CenaZ) & _
        "*tMagazyn.Marza) AS kwotaMarzy " & _
        "FROM (tKategorie INNER JOIN tNazwy ON " & _
        "tKategorie.ID_K = tNazwy.ID_K) " & _
        "INNER JOIN (tMagazyn INNER JOIN (tFaktura " & _
        "INNER JOIN " & _
        "tFakturaDetale ON tFaktura.id_f = " & _
        "tFakturaDetale.id_f) " & _
        "ON tMagazyn.ID_Mag = tFakturaDetale.id_mag) ON " & _
        "tNazwy.ID_N = tMagazyn.ID_N "
    ' badanie, czy ma być ograniczona liczba rekordów
    If Forms!frmRozneAnalizySprzedazy.cboRokS3 = _
        "(wszystkie)" Then
        txt = txt & "GROUP BY tKategorie.NazwaK Order by " & _
        "tKategorie.NazwaK"
```

```

Else
    txt = txt & "WHERE Year(tFaktura.DataZ) = " & _
        Forms!frmRozneAnalizySprzedazy.cboRokS3 & _
        " GROUP BY tKategorie.NazwaK Order by " & _
        "tKategorie.NazwaK"
End If
' przypisanie źródła danych
Me.RecordSource = txt
End Sub

```

W podobny sposób do wcześniejszych raportów projektujemy nasz raport, jedyna trudność polega na tym, że tym razem nie mamy listy pól do dyspozycji. Konsekwencją tego jest konieczność budowania wszystkich obiektów raportu wyłącznie przy pomocy formantów pobieranych z przybornika. W sekcji szczegóły umieszczałyśmy trzy niezwiązane pola tekstowe, którym od razu przypisujemy pola nazwak, brutto i kwotaMarzy zdefiniowane w zapytaniu pokazanej wyżej procedury. Listę pól w tej sekcji uzupełnia pole numerujące zwracane rekordy.



W nagłówku strony wstawiamy etykiety opisujące kolumny danych w sekcji szczegóły, w nagłówku raportu dwa pola tekstowe zwracające odpowiedni tytuł raportu (zależnie od wybranej pozycji w polu cboRokS3). Tradycyjnie stopka strony zawiera pole tekstowe zwracające numer strony raportu, a stopka raportu pola tekstowe zwracające podsumowanie wartości brutto sprzedaży i sumarycznej kwoty przychodów z tytułu marży handlowej.

I na zakończenie trzy przykładowe widoki tego raportu (różne lata).

The image displays three separate windows of a software application, each showing a report titled "Zbiorcze zestawienie sprzedaży wg kategorii".

Top Window (Year 2003):

Lp.	Nazwa kategorii:	Wartość brutto:	Kwota marży:
1	Drzewka owocowe	1 565,00 zł	122,80 zł
2	Herbicydy	20 251,21 zł	2 002,64 zł
3	Nasiona	1 080,00 zł	147,77 zł
4	Środki grzybobójcze	4 576,20 zł	547,70 zł
Razem		27 472,41 zł	2 820,92 zł

Middle Window (Year 2004):

Lp.	Nazwa kategorii:	Wartość brutto:	Kwota marży:
1	Herbicydy	1 832,00 zł	192,27 zł
2	Nasiona	618,70 zł	84,84 zł
3	Środki grzybobójcze	115,60 zł	18,04 zł
Razem		2 566,30 zł	294,95 zł

Bottom Window (Year 2005):

Lp.	Nazwa kategorii:	Wartość brutto:	Kwota marży:
1	Drzewka owocowe	1 565,00 zł	122,80 zł
2	Herbicydy	18 419,21 zł	1 810,38 zł
3	Nasiona	461,30 zł	63,13 zł
4	Środki grzybobójcze	4 460,60 zł	529,66 zł
Razem		24 906,11 zł	2 525,97 zł

Analiza sprzedaży wg nazw produktów

Ostatni z serii czterech raportów analizujących wielkość sprzedaży przygotujemy w taki sposób, aby było możliwe wykonanie analizy dla wszystkich lat lub dowolnie wybranego, podobną zasadę wprowadzimy odnośnie miesiąca sprzedaży – wszystkie lub wskazany. Technicznie rozwiązujemy to w sposób bardzo podobny do raportu wg kategorii.

Na stronie czwartej formantu karta formularza frmRozneAnalizySprzedazy umieszczały dwa pola kombi o nazwach odpowiednio cboRokS4 i cboMiesiac. Właściwość *Typ źródła wierszy* obu formantów zmieniamy na *Lista wartości*, dla pola kombi cboMiesiac zmieniamy liczbę kolumn na 2 ustawiając ich szerokość na 0 i 3 cm.

Musimy teraz troszeczkę zmodyfikować procedurę zdarzeniową *Przy zmianie* formantu karta, chodzi po prostu o zróżnicowanie przypisania źródła wierszy do odpowiedniego pola kombi cboRokS3 lub cboRokS4.

```
Private Sub FormantKarta0_Change()
    ' . . .
    ' zmiana przypisania Me.cboRokS3.RowSource = txt
    ' w sposób zależny od numeru aktywnej strony
    If Me.FormantKarta0 = 2 Then
        Me.cboRokS3.RowSource = txt
    Else
        Me.cboRokS4.RowSource = txt
    End If
    ' . . .
End Sub
```

Musimy napisać procedurę zdarzenia *Po aktualizacji* dla pola kombi cboRokS4, jej zadaniem będzie dostarczenie danych dla pola cboMiesiac.

```
Private Sub cboRokS4_AfterUpdate()
    Dim con As ADODB.Connection, rst As ADODB.Recordset
    Dim txt As String, i As Integer
    Set con = New ADODB.Connection
    Set con = CurrentProject.Connection
    Set rst = New ADODB.Recordset
    txt = "SELECT DISTINCT Month(DataZ) AS mm " & _
          "FROM tFaktura"
    If Me.cboRokS4 = "(wszystkie)" Then
        txt = txt & " order by Month(DataZ)"
    Else
        txt = txt & " where year(DataZ) = " & _
              Me.cboRokS4 & " order by month(DataZ)"
    End If
```

```

rst.Open txt, con, adOpenKeyset, adLockReadOnly
txt = "0;(wszystkie);"
For i = 1 To rst.RecordCount
    txt = txt & LTrim(Str(rst!mm)) & _
        ";" & MonthName(rst!mm) & ";"
    rst.MoveNext
Next i
Me.cboMiesiac.RowSource = txt
rst.Close
con.Close
Set rst = Nothing
Set con = Nothing
End Sub

```

Efekt działania obu procedur przygotowujących listę wartości dla pól kombi pokazany jest poniżej.



Możemy teraz zaprojektować raport, podobnie jak poprzednio jego źródło danych zbudujemy dynamicznie w momencie otwierania raportu. Raport ten zaprojektujemy z grupowaniem rekordów wg nazwy produktów, w sekcji szczegóły pokażemy nazwy opakowań, w których te produkty były sprzedawane, ilość sprzedanych produktów i ich wartość brutto. W zależności od wybranych opcji w obu polach kombi pokazanego wyżej formularza będziemy odpowiednio ograniczać liczbę rekordów lub nie.

```

Private Sub Report_Open(Cancel As Integer)
    Dim txt As String
    txt = "SELECT tNazwy.NazwaP, " & _
        "tRodzajeOpakowania.opis as opakowanie, " & _
        "Sum(tFakturaDetaile.ile) AS SumaOfile, " & _
        "Sum(Round(CCUR(tFakturaDetaile.ile)*" & _
        "tMagazyn.CenaS),2)) AS brutto " & _
        "FROM tRodzajeOpakowania INNER JOIN (tOpakowania " & _
        "INNER JOIN (tNazwy INNER JOIN " & _

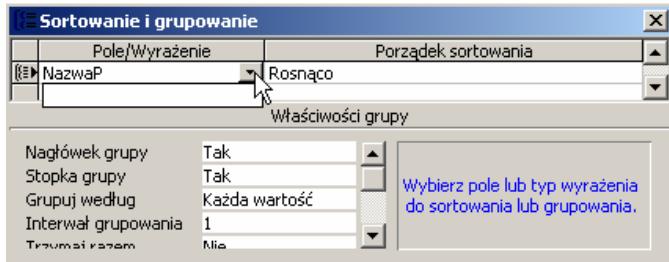
```

```

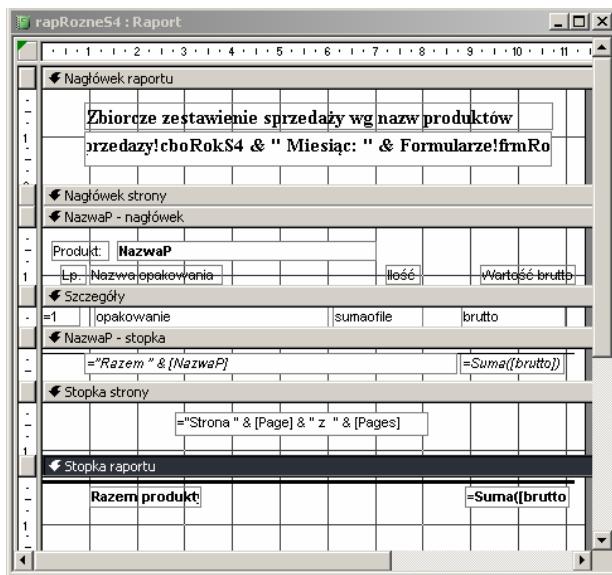
" (tMagazyn INNER JOIN " & _
"(tFaktura INNER JOIN tFakturaDetale ON " & _
"tFaktura.id_f = tFakturaDetale.id_f) ON " & _
"tMagazyn.ID_Mag = tFakturaDetale.id_mag) " & _
"ON tNazwy.ID_N = tMagazyn.ID_N) ON " & _
"tOpakowania.ID_O = tMagazyn.ID_O) ON " & _
"tRodzajeOpakowania.id_rodzaj = tOpakowania.id_rodzaj"
' badanie, jak skonstruować warunek WHERE
If Forms!frmRozneAnalizySprzedazy.cboRokS4 = _
    "(wszystkie)" Then
    If Forms!frmRozneAnalizySprzedazy.cboMiesiac > 0 Then
        txt = txt & " where month(tFaktura.DataZ) = " & _
            Forms!frmRozneAnalizySprzedazy.cboMiesiac
    End If
Else
    txt = txt & " where year(tFaktura.DataZ) = " & _
        Forms!frmRozneAnalizySprzedazy.cboRokS4
    If Forms!frmRozneAnalizySprzedazy.cboMiesiac > 0 Then
        txt = txt & " and month(tFaktura.DataZ) = " & _
            Forms!frmRozneAnalizySprzedazy.cboMiesiac
    End If
End If
' dodanie grupowania
txt = txt & _
    " group by tNazwy.NazwaP, tRodzajeOpakowania.opis"
Me.RecordSource = txt
End Sub

```

Mając zdefiniowane źródło danych raportu (choć nie mamy listy pól) rozpoczynamy projektowanie raportu od wywołania polecenia *Sortowanie i grupowanie* z menu *Widok* lub menu kontekstowego raportu. Ponieważ nie możemy wybrać pola grupującego z listy (bo jej nie ma!), to po prostu nazwę tę wpisujemy – u nas będzie to *NazwaP*.



Pozostałe działania projektowe są już typowe i wydaje nam się, że możemy ich wykonanie pozostawić Czytelnikowi. Poniżej widok projektu tego raportu.



Nadeszła także pora na napisanie procedury zdarzenia *Przy kliknięciu* przycisku polecenia cmdWykonaj formularza frmRozneAnalizySprzedazy. Jej zadaniem jest uruchomienie w widoku podglądu wydruku wybranego raportu.

```
Private Sub cmdWykonaj_Click()
    Select Case Me.FormantKarta0
        Case 0
            DoCmd.OpenReport "rapRozneS1", acViewPreview
        Case 1
            If Me.Ramka14.Value = 1 Then
                DoCmd.OpenReport "rapRozneS2_m", acViewPreview
            Else
                DoCmd.OpenReport "rapRozneS2_p", acViewPreview
            End If
        Case 2
            DoCmd.OpenReport "rapRozneS3", acViewPreview
        Case 3
            DoCmd.OpenReport "rapRozneS4", acViewPreview
    End Select
End Sub
```

Na kolejnej stronie widok analizy sprzedaży wg nazw produktów dla wszystkich lat i miesięcy prowadzenia działalności handlowej przez nasz przykładowy sklep – z uwagi na rozmiar raportu pokazany jest fragment piewszej strony.

Zbiorcze zestawienie sprzedaży wg nazw produktów Rok: (wszystkie) Miesiąc: (wszystkie)			
Produkt: Goal			
Lp. Nazwa opakowania Ilość Wartość brutto			
1	0,25 litra	21	1 213,80 zł
2	0,5 kg	21	53,97 zł
3	1 litr	12	362,04 zł
4	1,5 kg	2	168,40 zł
<i>Razem Goal</i>			1 798,21 zł
Produkt: Jabłoń Cortland			
Lp. Nazwa opakowania Ilość Wartość brutto			
1	100 sztuk	1	1 049,00 zł
<i>Razem Jabłoń Cortland</i>			1 049,00 zł
Produkt: Miedzian			
Lp. Nazwa opakowania Ilość Wartość brutto			
1	1,5 kg	40	1 694,80 zł
2	20 kg	4	392,40 zł
<i>Razem Miedzian</i>			2 087,20 zł
Produkt: Roundap			
Lp. Nazwa opakowania Ilość Wartość brutto			
1	10 litrów	4	1 100,00 zł
2	20 kg	35	8 015,00 zł
3	20 litrów	23	9 338,00 zł
<i>Razem Roundap</i>			18 453,00 zł
Produkt: Stonecznik			
Strona: 1			

6.6. Menu użytkownika

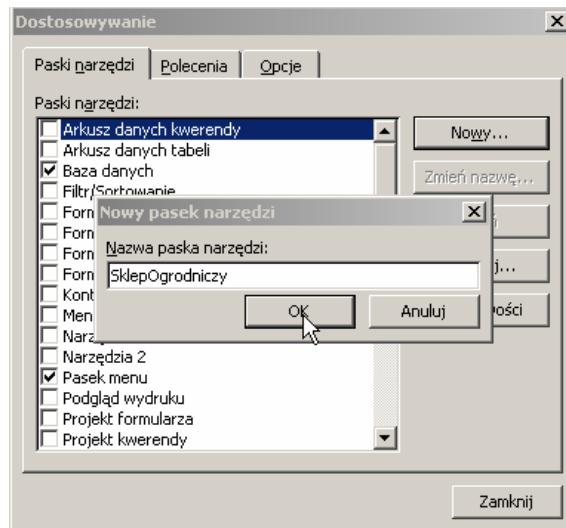
Utworzenie menu użytkownika bazy SklepOgrodnicy.mdb będzie wymagało od nas trzech kroków:

1. Zaprojektowania nowego paska narzędziowego jako menu,
2. Przygotowanie funkcji wywołujących polecenia umieszczone w menu wraz z niezbędnymi modyfikacjami utworzonych wcześniej formularzy i raportów,
3. Modyfikacji ustawień startowych MS Access.

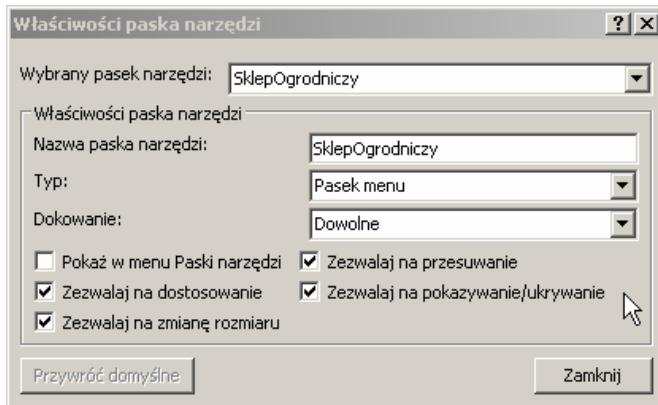
Kolejno przedstawimy realizację tych kroków.

6.6.1 Projektowanie paska menu

Z menu kontekstowego dowolnego paska narzędziowego wywołujemy polecenie *Dostosuj*, a następnie z otwartego okna dialogowego tego polecenia, z zakładki *Paski narzędziowe* wywołujemy polecenie *Nowy*. Zostaje wyświetlone okno dialogowe nowego paska z żądaniem nadania mu nazwy. Dokładnie taka sytuacja pokazana jest poniżej, po akceptacji przycisku OK zostanie utworzony nowy pasek narzędziowy (a nie pasek menu).

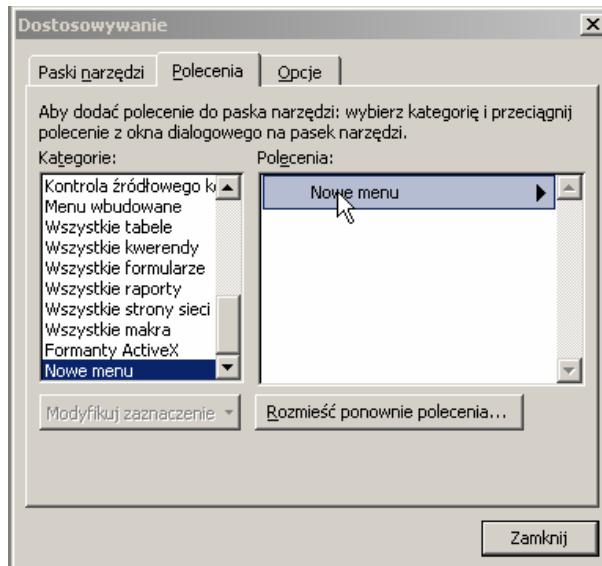


Bezpośrednio po utworzeniu nowego paska narzędziowego wywołujemy jego właściwości (z okna *Dostosowywanie*), co pozwoli nam na zmianę typu paska z narzędziowego na pasek menu.

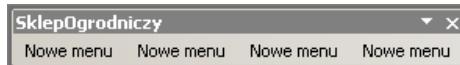


Po zamknięciu pokazanego okna właściwości nowo utworzony pasek menu jest gotowy i oczekuje na umieszczenie w nim polecień menu.

W oknie *Dostosowywanie* wybieramy zakładkę *Polecenia*, kategorię *Nowe menu* i polecenie *Nowe menu*.

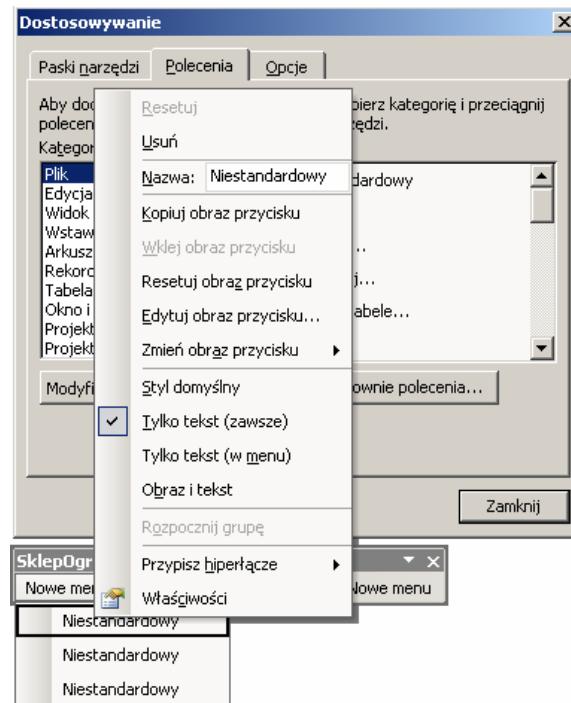


Myszą, techniką „ciagnij i upuść” przeciągamy przycisk polecenia *Nowe menu* do tworzonego paska menu, my w tym przykładzie umieścimy cztery przyciski odpowiadające takim zagadnieniom jak: klienci, zaopatrzenie, sprzedaż i raportowanie.



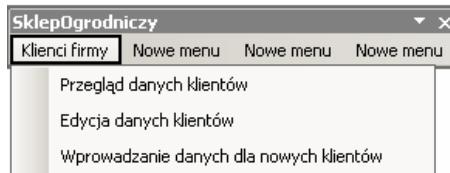
Wstawione w ten sposób przyciski polecenia *Nowe menu* muszą być teraz uzupełnione o standardowe przyciski poleceń, ewentualnie o inne przyciski menu pozwalające na budowę menu zagnieździonego. W naszym przypadku pierwsze polecenie nazwiemy *Klienci*, a umieścimy w nim trzy standardowe przyciski poleceń, których zadaniem będzie uruchomienie formularza frmKlient w różnych wersjach (przegląd, edycja i dodawanie danych dla nowego klienta).

W oknie *Dostosowywanie* w zakładce *Polecenia* uaktywniamy kategorię *Plik*, a w liście polecenia wybieramy przycisk *Niestandardowy*. Techniką „ciagnij i upuść” umieszczamy trzy tego typu przyciski w pierwszym przycisku *Nowe menu*. Kolejny krok to opisanie wstawionych przycisków standardowych, zrobimy to wykorzystując menu kontekstowe każdego z nich (jak pokazano niżej).

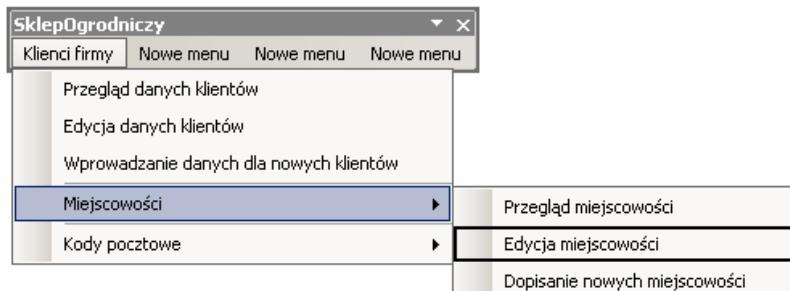


Dla każdego z trzech przycisków przypisujemy właściwości *Nazwa* stosowny tekst akceptując go każdorazowo klawiszem Enter. Będziemy musieli jeszcze raz wrócić do menu kontekstowego tych przycisków po to, aby w jego właściwościach określić funkcję, która ma być uruchomiona w momencie wyboru danego polecenia.

W analogiczny sposób zmieniamy także nazwę pierwszego przycisku menu na np. „Klienci firmy”. Po zakończeniu tego etapu pracy powinniśmy mieć taką sytuację.



W tym menu powiniśmy jeszcze umieścić przyciski dające dostęp do miejscowości i kodów pocztowych, możemy to zrobić umieszczając jeszcze dwie serie po 3 przyciski niestandardowe dla działań podejmowanych na tych tabelach, możemy też zbudować menu hierarchiczne. W tym ostatnim przypadku umieścimy dwa przyciski typu Nowe menu (tym razem menu będzie się rozwijało w prawo a nie w dół), a w każdym rozwinięciu po 3 przyciski niestandardowe. Dla zaakcentowania odrębności tych dwóch przycisków menu oddzielimy je od siebie i od wcześniejszych przycisków za pomocą linii poprzez wybranie opcji *Rozpocznij grupę* we właściwościach danego przycisku menu. Poniżej widok menu „Klienci firmy” z rozwiniętym podmenu „Miejscowości”



W analogiczny sposób budujemy pozostałe pozycje naszego menu, z uwagi na dość znaczny rozmiar tej książki zrezygnujemy w tym miejscu od omawiania krok po kroku etapów tej rozbudowy, na krążku jest baza danych, można zobaczyć jak to zostało zrobione. My przejdziemy teraz do przygotowania funkcji, które przypiszemy do przycisków standardowych naszego menu. Będziemy także zmuszeni do dopisania kilku procedur do formularzy po to, aby dostosować ich sposób otwierania do poleceń paska menu.

6.6.2 Funkcje obsługujące polecenia menu

Zadanie nasze polega na napisaniu publicznej funkcji otwierającej wskazany formularz w trybie przeglądu danych, przeglądu i edycji oraz wprowadzania danych. Funkcję tę przygotujemy w module Standardowy (poprzednia nazwa Module1), a jej kod pokazany jest niżej.

```
Public Function DajFormularz(nazwaF As String, _
    tryb As Integer)
    ' nazwaF to nazwa formularza do otwarcia
    ' tryb to co ma robić: 1 - przegląd, 2 - edycja,
    '      3 - dodawanie nowego rekordu
    intFlaga = tryb
    Select Case tryb
        Case 1
            DoCmd.OpenForm nazwaF, acNormal, , , acFormReadOnly
        Case 2
            DoCmd.OpenForm nazwaF, acNormal, , , acFormEdit
        Case 3
            DoCmd.OpenForm nazwaF, acNormal, , , acFormAdd
    End Select
End Function
```

Funkcja DajFormularz dostaje w momencie wywołania dwa argumenty, nazwę formularza do otwarcia oraz liczbę całkowitą określającą sposób jego otwarcia. Wewnątrz funkcji zmienna tryb jest przypisana do zmiennej publicznej intFlaga, robimy to w tym celu, aby móc dostosować wygląd otwieranego formularza do trybu jego otwarcia – będzie to robione za pomocą procedury zdarzenia *Przy otwarciu* danego formularza. Przy pomocy struktury warunkowej select case badany jest wartość argumentu tryb i stosownie do niej uruchamiane polecenie DoCmd.OpenForm ze zróżnicowanym ostatnim argumentem.

Poniżej pokazana jest procedura zdarzenia *Przy otwarciu* formularza frmKlient, jej zadaniem jest dostosowanie wyglądu tego formularza do sposobu, w jaki ma być otwarty. Dla ustalenia, co ma się działać badany jest stan zmiennej publicznej intFlaga i w zależności od jej wartości modyfikowane są niektóre własności otwieranego formularza.

```
Private Sub Form_Open(Cancel As Integer)
    Select Case intFlaga
        Case 1
            Me.Caption = "Przegląd zarejestrowanych klientów"
        Case 2
            Me.Caption = _
                "Przegląd i edycja zarejestrowanych klientów"
```

```
    Me.AllowAdditions = False
Case 3
    Me.Caption = "Dodanie nowego klienta"
    Me.AllowAdditions = True
End Select
End Sub
```

Musimy jeszcze zmienić dotychczasową procedurę zdarzenia *Przy kliknięciu* przycisku cmdDodaj – zmiana podyktowana koniecznością ujednolicenia wartości przy pisywanej zmiennej intFlaga w momencie wywoływania pomocniczego formularza.

```
Private Sub cmdDodaj_Click()
    intFlaga = 4 ' zmiana, wcześniej było tu 2
    DoCmd.OpenForm "frmMiejscowosc", acNormal, , , ,
                    acFormAdd, acDialog
End Sub
```

Będą także konieczne zmiany procedury zdarzenia *Przy otwarciu* formularzy frmMiejscowosc i frmKody.

```
Private Sub Form_Open(Cancel As Integer)
    Select Case intFlaga
        Case 1
            Me.Caption = "Przegląd miejscowości"
            Me.cmdZapisz.Visible = False
            Me.NavigationButtons = True
            Me.AllowEdits = False
            Me.AllowAdditions = False
            Me.cmdDodaj.Visible = False
        Case 2
            Me.Caption = "Przegląd i edycja miejscowości"
            Me.cmdZapisz.Visible = False
            Me.NavigationButtons = True
            Me.AllowEdits = True
            Me.AllowAdditions = False
            Me.cmdDodaj.Visible = True
        Case 3
            Me.Caption = "Wprowadzanie nowych miejscowości"
            Me.cmdZapisz.Visible = False
            Me.NavigationButtons = True
            Me.AllowEdits = True
            Me.cmdDodaj.Visible = False
        Case 4 ' dodanie nowej, ale z innego formularza
            Me.Caption = "Wprowadzanie nowej miejscowości"
            Me.cmdZapisz.Visible = True
    End Select
End Sub
```

```
        Me.NavigationButtons = False
        Me.AllowEdits = True
        Me.cmdDodaj.Visible = True
    End Select
End Sub

Private Sub Form_Open(Cancel As Integer)
    Select Case intFlaga
        Case 1
            Me.Caption = "Przegląd kodów"
            Me.cmdZapisz.Visible = False
            Me.NavigationButtons = True
        Case 2
            Me.Caption = "Przegląd i edycja kodów"
            Me.cmdZapisz.Visible = False
            Me.NavigationButtons = True
            Me.AllowAdditions = False
        Case 3
            Me.Caption = "Wprowadzanie nowych kodów"
            Me.cmdZapisz.Visible = False
            Me.NavigationButtons = True
            Me.AllowEdits = True
        Case 4
            Me.Caption = "Wprowadzanie nowego kodu"
            Me.cmdZapisz.Visible = True
            Me.NavigationButtons = False
            Me.AllowEdits = True
    End Select
End Sub

I jeszcze kosmetyczna zmiana w procedurze Przy kliknięciu przycisku cmdDodaj
w formularzu frmMiejscowosc

Private Sub cmdDodaj_Click()
    intFlaga = 4
    DoCmd.OpenForm "frmKod", acNormal, , , acFormAdd, acDialog
End Sub
```

Mamy już wszystko to, co jest nam potrzebne dla pełnego skonfigurowanie menu *Klienci firmy* w naszym pasku menu SklepOgrodnicy. Ponownie wywołujemy polecenie *Dostosuj* z menu kontekstowego dowolnego paska narzędziowego i rozwijamy menu *Klienci firmy*. Prawym przyciskiem myszy otwieramy menu kontekstowe przycisku standardowego *Przegląd danych klientów* i wybieramy polecenie *Właściwości*. W liście pola kombi *Przy akcji* wpisujemy wyrażenie =DajFormularz("frmKlienci",1).

Jak widzimy, jest to wywołanie zdefiniowanej wcześniej funkcji z przekazaniem nazwy formularza i trybu otwarcia.



Wpisane wyrażenie zatwierdzamy przyciskiem *Zamknij* i w analogiczny sposób przypisujemy wywołanie tej funkcji pozostałym dwóm przyciskom standardowym zmieniając jedynie tryb wywołania (odpowiednio na 2 i 3).

Dokładnie takie same działania podejmujemy w stosunku do przycisków standardowych podmenu *Miejscowości* i *Kody pocztowe*. W przypadku podmenu *Miejscowości* będą to (kolejno) przypisania:

```
=DajFormularz("frmMiejscowosc", 1)
=DajFormularz("frmMiejscowosc", 2)
=DajFormularz("frmMiejscowosc", 3)
```

a w przypadku podmenu *Kody pocztowe* przypisania:

```
=DajFormularz("frmKod", 1)
=DajFormularz("frmKod", 2)
=DajFormularz("frmKod", 3)
```

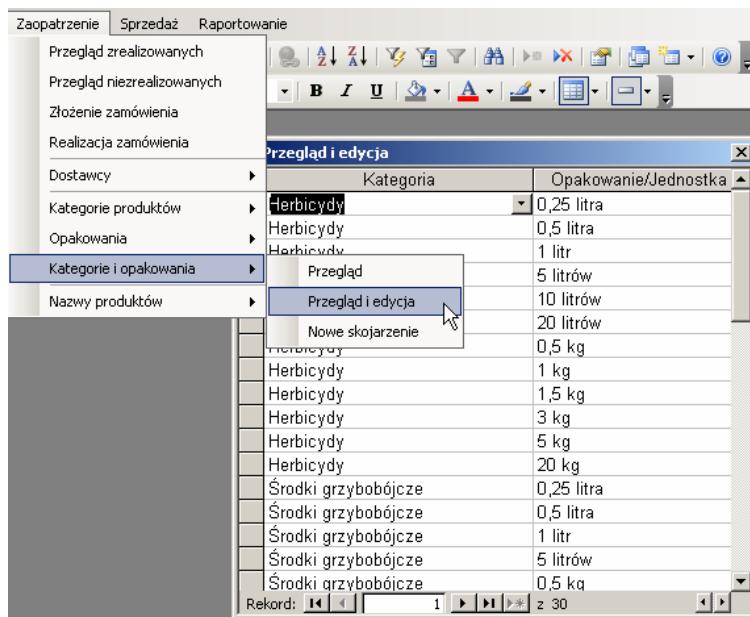
Po zamknięciu okna dialogowego Dostosuj możemy już sprawdzić funkcjonowanie menu *Klienci firmy*, wszystko powinno funkcjonować bez zarzutu.

W analogiczny sposób powinniśmy zdefiniować zdarzenie *Przy akcji* dla wszystkich pozostałych przycisków standardowych zbudowanego menu. W praktyce okazało się, że konieczne będzie napisanie jeszcze jednej funkcji wywołującej formularz w widoku

arkusza danych. Niejako „przy okazji” do obu funkcji wywołujących formularze dodano dwa wiersze zamykające wywoływany formularz (jeżeli był otwarty).

```
Public Function DajFormularzDS(nazwaF As String,
    tryb As Integer)
On Error Resume Next
DoCmd.Close acForm, nazwaF, acSaveNo
intFlaga = tryb
Select Case tryb
Case 1
    DoCmd.OpenForm nazwaF, acFormDS, , , acFormReadOnly
Case 2
    DoCmd.OpenForm nazwaF, acFormDS, , , acFormEdit
Case 3
    DoCmd.OpenForm nazwaF, acFormDS, , , acFormAdd
End Select
End Function
```

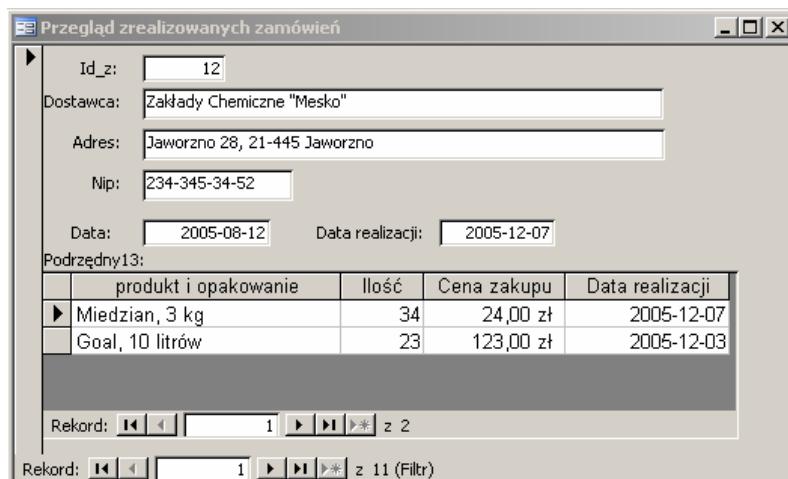
Poniżej pokazany jest fragment okna programu MS Access z rozwiniętym menu *Zaopatrzenie* i podmenu *Kategorie i opakowania*. W tle widoczny jest otwarty w widoku arkusz danych formularz frmOpakowaniaRodzajeOld.



Do otwarcia tego formularza wykorzystano przedstawioną wyżej nową funkcję `DajFormularzDS`. Funkcję tę wykorzystano także w kilku innych przypadkach.

W ramach menu *Zaopatrzenie* umieściliśmy dwa standardowe przyciski polecen pozwalających na przegląd złożonych zamówień z podziałem na zrealizowane i niezrealizowane. Tu mamy pewien problem, ponieważ nie mamy gotowego formularza – formularz `frmZamowienie` nie może być wykorzystany, ponieważ jego podformularz nie korzysta z tabeli `tSzczegZamowienia`.

W tej sytuacji konieczne było napisanie takiego formularza całkowicie od nowa, polecamy zainteresowanym Czytelnikom szczegółową analizę formularza głównego `frmZakupyPrzeglad` oraz podformularza `frmPodRealizacjaZamPrzeglad`, oba obiekty są dość ciekawie zaprojektowane. Formularz ten wykorzystuje technikę filtrowania rekordów, dzięki temu jednym formularzem obsłużyliśmy dwa różne polecenia menu. Poniżej widok formularza `frmZakupyPrzeglad` wywołanego poprzez klik polecenia *Przegląd zrealizowanych* w menu *Zaopatrzenie*.



Analizę pozostałych pozycji menu SklepOgrodnicy pozostawiamy Czytelnikowi, włącznie z dalszą, samodzielna rozbudową menu (być może będzie trzeba zaprojektować jakieś dodatkowe formularze czy raporty).

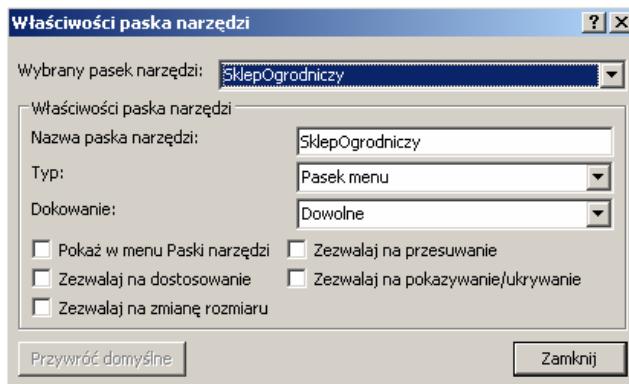
Przejdziemy teraz do dokonania takich zmian ustawień startowych, które pozwolą na bardziej efektywne wykorzystywanie przygotowanej bazy.

6.6.3 Modyfikacja ustawień startowych MS Access

Zmiany, które za chwilę zrobimy w ustawieniach startowych najlepiej wykonać na kopii bazy danych SklepOgrodniczy.mdb, a nie na oryginalnym pliku. Chodzi po prostu o to, żebyśmy zawsze mieli gdzieś schowany oryginalny plik bazy danych, jeżeli coś źle skonfigurujemy, to po prostu będziemy mogli skorzystać z oryginału.

Celem zmian, które chcemy wprowadzić jest po pierwsze ukrycie oryginalnego menu, a pozostałe jedynie zdefiniowanego menu użytkownika. W związku z tym zablokujemy możliwość modyfikowania tego menu. Będziemy także chcieli ukryć okno bazy danych, co częściowo utrudni „normalnym” użytkownikom „grzebanie” w obiektach bazy danych. Będzie to jakaś namiastka zabezpieczenia bazy danych, dokładniej zajmiemy się tym zagadnieniem w rozdziale 7.

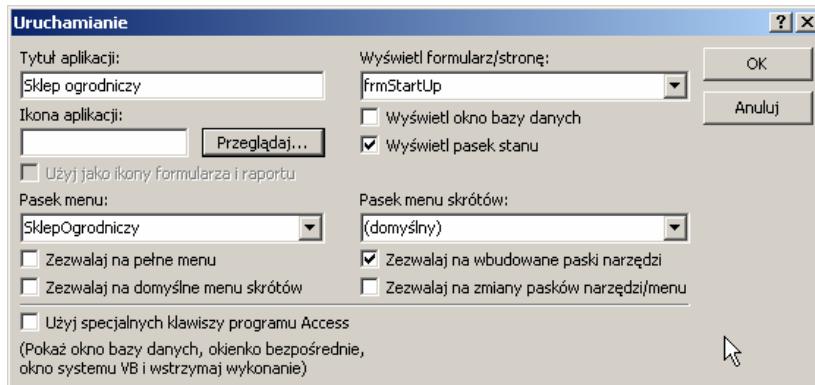
Zaczniemy od zmian właściwości naszego paska menu. Z menu kontekstowego dowolnego paska menu wywołujemy polecenie *Dostosuj*, w oknie dialogowym tego polecenia uaktywniamy zakładkę *Paski narzędzi*, odszukujemy i selekcjonujemy pasek SklepOgrodniczy i pokazujemy jego właściwości. W otwartym oknie dialogowym właściwości zdejmujemy zaznaczenia we wszystkich polach wyboru, tak jak to pokazano niżej.



Po zatwierdzeniu tych zmian nasz pasek menu będzie zabezpieczony przed dokonaniem w nim jakichkolwiek zmian, będzie także cały czas widoczny w naszej aplikacji.

Powinniśmy jeszcze przygotować formularz startowy naszej aplikacji, będzie to po prostu niezwiązany z żadnym źródłem danych formularz pokazywany w widoku pojedynczego formularza z etykietą informującą o autorach projektu. Formularz ten można dowolnie skomponować, można np. poza etykietą z informacją o aplikacji czy autorach umieścić w nim jakieś elementy graficzne. W naszym przypadku taki formularz został przygotowany i zapamiętany pod nazwą frmStartUp.

W menu *Narzędzia* znajdziemy polecenie *Uruchamianie*, po jego wywołaniu wyświetlane zostanie pokazane niżej okno dialogowe.

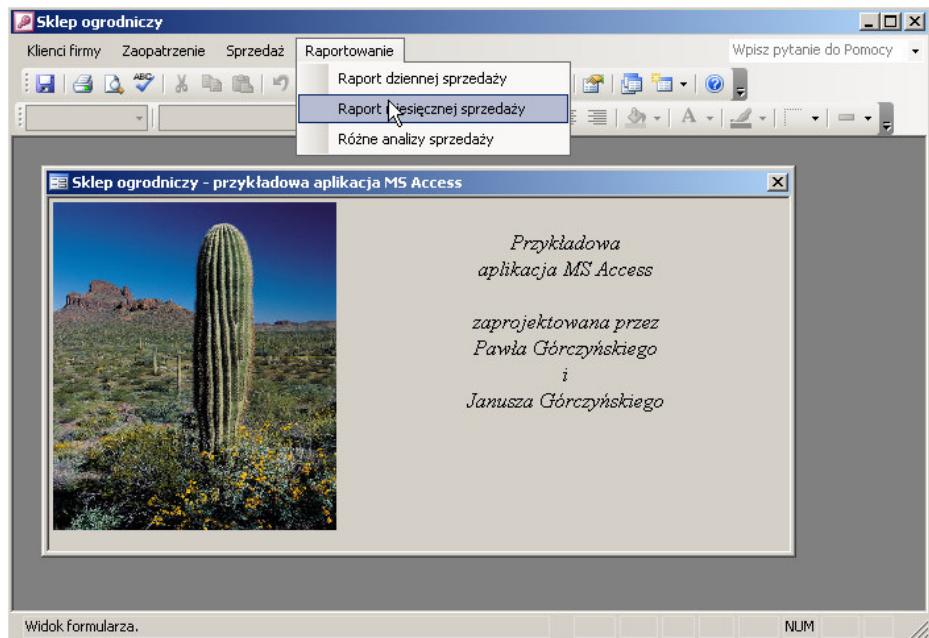


W polu tekstowym *Tytuł aplikacji* wpisujemy taką nazwę, jaka chcemy mieć w pasku tytułowym okna programu MS Access po uruchomieniu naszej bazy. W polu kombi *Pasek menu* wybieramy z listy nasze menu SklepOgrodniczy. Jeżeli nie chcemy dać możliwości przełączania się między naszym menu a menu standardowym, to pozostawiamy puste pole wyboru *Zezwalaj na pełne menu* oraz pole *Zezwalaj na domyślne menu skrótów*.

Obowiązkowo powinniśmy pozostawić puste pole wyboru *Użyj specjalnych klawiszy programu Access* (po to, aby utrudnić dostęp do okna bazy danych czy innych istotnych obiektów).

W polu kombi *Wyświetl formularz/stronę* wybieramy formularz *frmStartUp*, a pole wyboru *Wyświetl okno bazy danych* pozostawiamy puste. Pole wyboru *Wyświetl pasek stanu* powinno być wybrane. W polu kombi *Pasek menu skrótów* nie musimy nic zmieniać, z pozostałych dwóch pól wyboru uaktywniamy pole *Zezwalaj na wbudowane paski narzędzi*.

Po dokonaniu tych zmian możemy już zamknąć bazę danych. Po jej ponownym otwarciu zobaczymy efekt wprowadzonych zmian.



7. Baza danych w sieci LAN

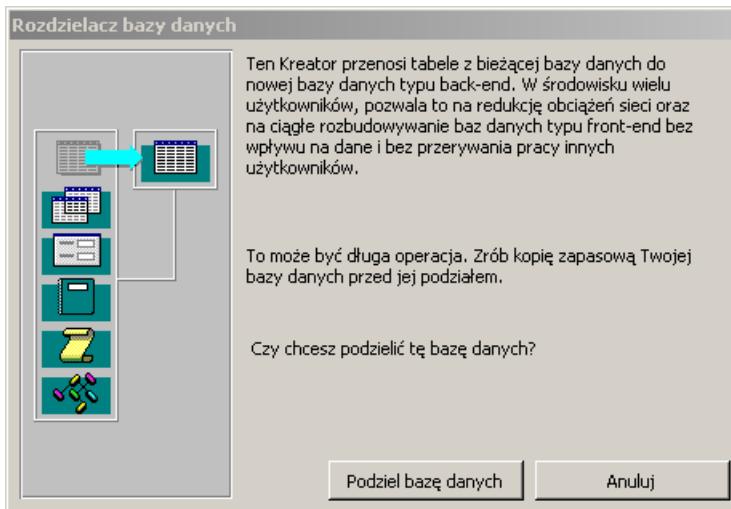
Standardowo baza danych zaprojektowana w MS Access przeznaczona jest do pracy jednostanowiskowej, a osoba uruchamiająca tak przygotowaną bazę otrzymuje uprawnienia administratora bazy. W zastosowaniach praktycznych jest to dość poważne ograniczenie, ponieważ baza danych powinna pracować w lokalnej sieci komputerowej i powinna pozwalać na pracę wielu użytkowników jednocześnie.

Zagadnieniom przygotowania bazy danych MS Access do pracy w lokalnej sieci komputerowej jest poświęcony ten rozdział naszej książki. Dla ilustracji problemu wykorzystamy bazę danych *SklepOgrodniczy.mdb*, ale wszelkie prace będziemy prowadzić na kopii tej bazy.

Prace nasze rozpocznemy od rozdzielenia bazy danych na dwie bazy; pierwsza z nich będzie zawierać wyłącznie tabele, druga pozostałe obiekty bazy danych plus łącza do tabel pierwszej z baz.

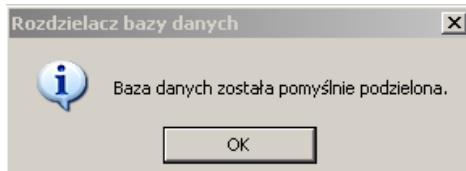
7.1. Podział bazy danych na dane i interfejs

Po skopiowaniu bazy danych *SklepOgrodniczy.mdb* otwieramy tę bazę, a następnie w menu *Narzędzia/Narzędzia bazy danych* znajdujemy i uruchamiamy polecenie *Rozdzielacz bazy danych*.

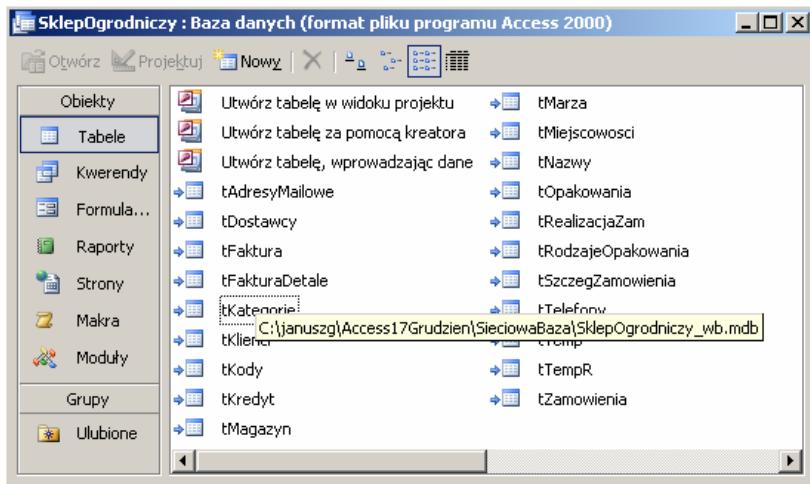


Po wyborze przycisku *Podziel bazę danych* kreator rozdzielacza zaczyna swoją pracę od zapytania, gdzie i pod jaką nazwą ma być zapisana baza z tabelami.

Po podaniu tych informacji (domyślnie MS Access proponuje nazwę bazy uzupełnioną symbolami „_wb”) i po krótkiej pracy kreator wyświetla komunikat o jej pomyślnym zakończeniu.



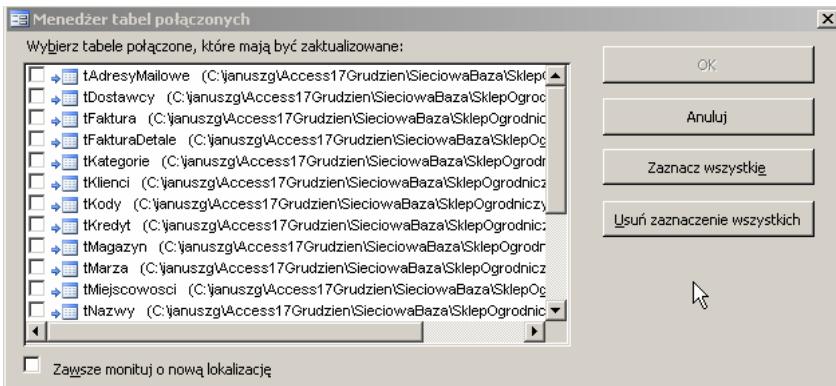
Po akceptacji powyższego komunikatu możemy zobaczyć w oknie bazy danych, w widoku tabel, dość istotną zmianę – nazwa każdej tabeli została poprzedzona symbolem strzałki. Jest to informacja o tym, że w miejsce oryginalnej tabeli mamy tzw. **tabelę połączoną**, w gruncie rzeczy jest to adres do bazy danych zawierającej tę tabelę. W pokazanej sytuacji widzimy taki adres w odniesieniu do tabeli tKategorie.



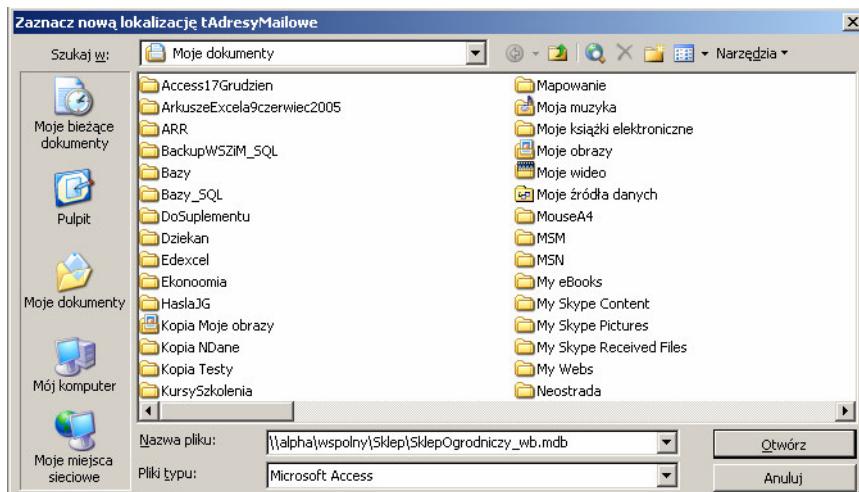
Po rozdzieleniu bazy danych na dwie bazy – bazę danych i bazę interfejsu, mamy znacznie większe możliwości niż poprzednio. Po pierwsze bazę danych będziemy mogli ulokować na serwerze plików w sieci lokalnej, a bazę interfejsu umieścić na stacji roboczej pracowników (jej lokalne kopie). Po drugie będziemy mogli indywidualnie specyfikować uprawnienia pracowników do obiektów bazy danych zwiększając w istotny sposób bezpieczeństwo danych.

MS Access dostarcza specjalne narzędzie do zarządzania tabelami połączonymi (menu *Narzędzia/Narzędzia baz danych/Menedżer tabel połączonych*). Usunięcie tabeli połączonej nie oznacza jej faktycznego usunięcia, a jedynie usunięcie łącza do tej tabeli. Przy jego pomocy można będzie zmienić adres wszystkich czy części tabel połączonych.

W pokazanej sytuacji widzimy, że adresy wszystkich tabel przyłączonych wskazują na bazę danych SklepOgrodniczy_wb.mdb umieszczoną na lokalnym dysku komputera jednego z autorów tej pracy.

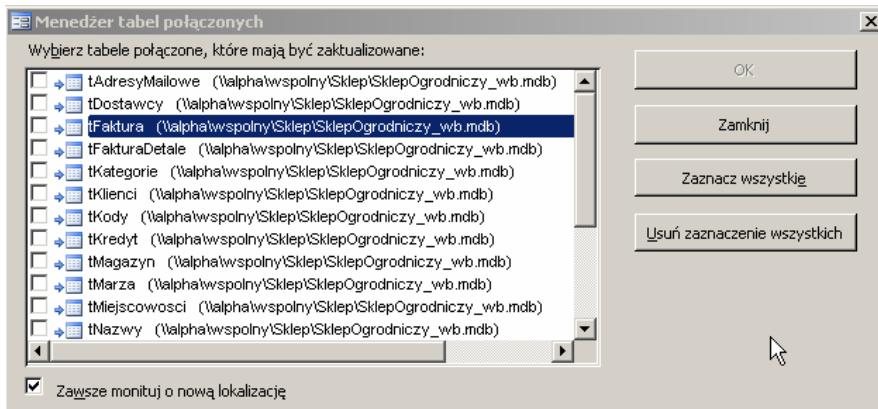


Za chwilę bazę danych umieścimy w udostępnionym folderze Sklep na serwerze plików o nazwie Alpha w folderze Wspolny. Folder Sklep musi mieć takie uprawnienia, aby pozwolić potencjalnym użytkownikom naszej bazy danych na dostęp i modyfikacje danych. Po tych pracach przygotowawczych można już uruchomić polecenie *Menedżer tabel połączonych* w celu zmiany odwołania dla tabel połączonych. W pokazanej sytuacji dla pierwszej z tabel połączonych podana jest pełna ścieżka sieciowa prowadząca do bazy SklepOgrodniczy_wb.mdb na komputerze sieciowym alpha.



W trakcie wskazywania dysku sieciowego bardzo ważne jest użycie kompletnego adresu sieciowego komputera alpha, a nie jego oznaczenia literowego oznaczającego zmapowany dysk sieciowy na komputerze administratora bazy wykonującego przymocowanie tabel. Unikamy w ten sposób takiej sytuacji, że baza danych z połączonymi tabelami działa tylko na tych stacjach roboczych, które mapują dysk sieciowy pod tą samą literą alfabetu.

Po zatwierdzeniu następuje odświeżenie łącz do wszystkich tabel połączonych, a aktualne adresy zobaczymy w oknie *Menedżera*.



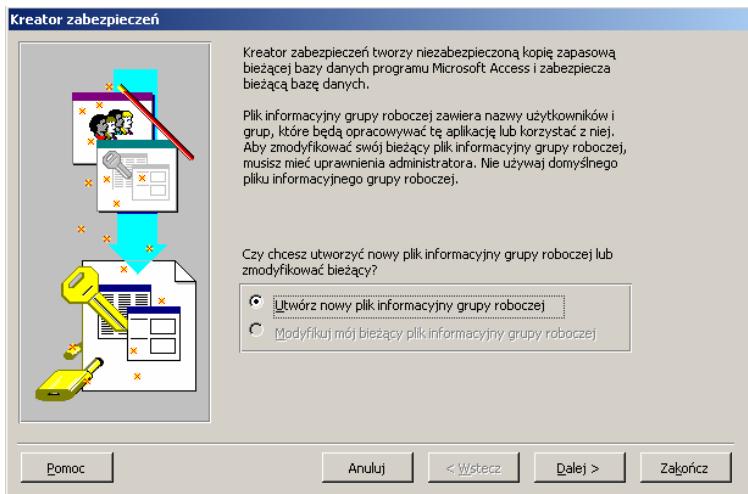
7.2. Zabezpieczenie bazy danych

Prześledzmy teraz kroki prowadzące do zabezpieczenia bazy danych na poziomie użytkownika. Wykorzystamy kreator zabezpieczeń dostępny w menu *Narzędzia/Narzędzia bazy danych*. Przy jego pomocy zabezpieczymy bazę *SklepOgrodniczy.mdb*, baza wewnętrzna *SklepOgrodniczy_wb.mdb* pozostanie formalnie niezabezpieczona, ale tylko formalnie. Baza ta będzie umieszczona na dedykowanym serwerze w lokalnej sieci komputerowej, do jej zabezpieczenia wykorzystamy standardowe metody uprawnień dostępne w środowisku Windows.

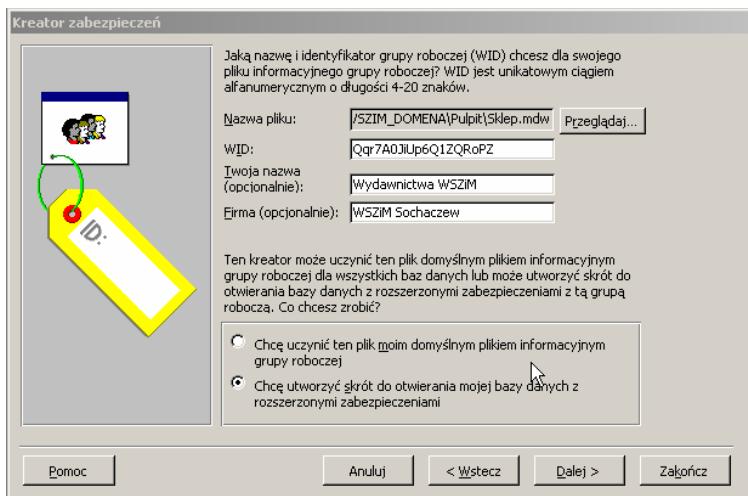
Wszelkie informacje o zabezpieczonych bazach danych oraz o ich użytkownikach MS Access zapisuje w specjalnym pliku, tzw. **pliku grupy roboczej** z rozszerzeniem *.mdw. Każda baza danych otwierana w MS Access korzysta z takiego pliku, standardowo jest to plik o nazwie System.mdw, do zarządzania plikiem grupy roboczej mamy dedykowane narzędzie dostępne w menu *Narzędzia/Narzędzia bazy danych/Administrator grupy roboczej* (w Access 2003). W przypadku wcześniejszych wersji MS Access jest to specjalna aplikacja pod nazwą Wrkgadm.exe, standardowo umieszczana w folderze C:\Program Files\Microsoft Office\Office\1045.

7.2.1. Kreator zabezpieczeń

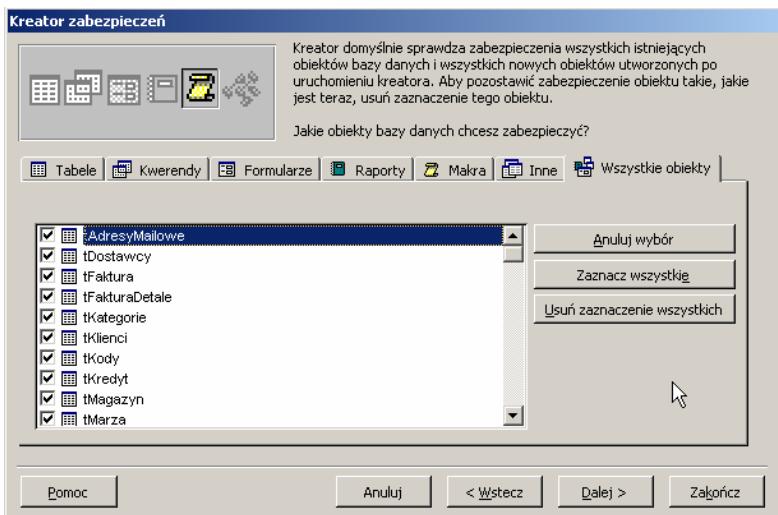
W naszym przypadku zabezpieczeniu poddamy plik SklepOgrodnicy.mdb, plik bazy zawierający łącza do tabel połączonych i wszystkie pozostałe, wcześniej utworzone obiekty. Po otwarciu bazy danych uruchamiamy z menu *Narzędzia/ Zabezpieczenia/Kreator zabezpieczeń na poziomie użytkownika*.



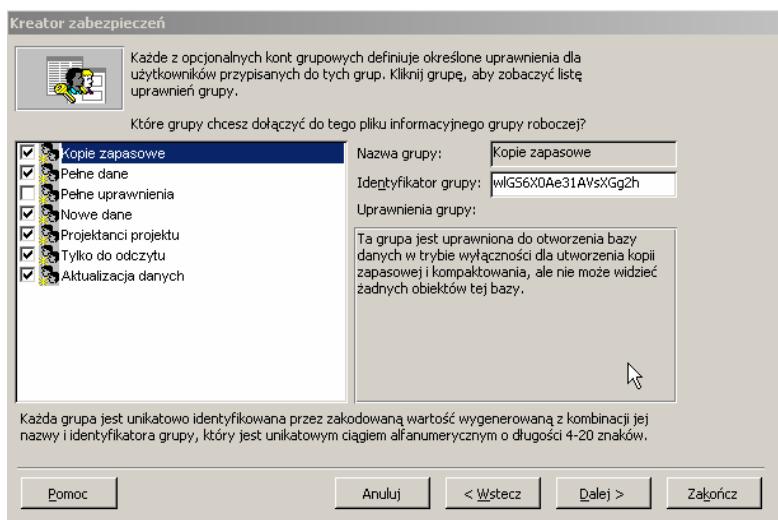
Specyfikacja nazwy i innych właściwości pliku grupy roboczej.



Wskazanie obiektów do zabezpieczenia, w naszym przypadku została wybrana opcja *Wszystkie obiekty*.



Wybór opcjonalnych kont grupowych.



Bardzo ważny moment – zablokowanie dostępu dla standardowej grupy *Użytkownicy*.

Kreator zabezpieczeń

We wszystkich plikach informacyjnych grupy roboczej wszyscy użytkownicy należą do grupy Użytkownicy. Domyślnie kreator nie przypisuje żadnych uprawnień grupie Użytkownicy, ale możesz przypisać tej grupie ograniczone uprawnienia. Nie przypisuj grupie Użytkownicy pełnych uprawnień, gdyż usunie to wszystkie zabezpieczenia.

Czy chcesz udzielić grupie Użytkownicy niektórych uprawnień?

Tak, chcę udzielić grupie Użytkownicy pewnych uprawnień
 Nie, grupa Użytkownicy nie powinna mieć żadnych uprawnień

Baza danych | Tabele | Kwerendy | Formularze | Raporty | Makra |

<input type="checkbox"/> Otwórz/Uruchom	<input checked="" type="checkbox"/> Odczytaj dane
<input type="checkbox"/> Otwórz z wyłącznością	<input checked="" type="checkbox"/> Aktualizuj dane
<input checked="" type="checkbox"/> Modyfikuj projekt	<input checked="" type="checkbox"/> Wstaw dane
<input type="checkbox"/> Administruj	<input checked="" type="checkbox"/> Usuń dane

Pomoc | Anuluj | < Wstecz | Dalej > | Zakończ |

Dodanie nazw, haseł dostępu i identyfikatorów dla użytkowników naszej bazy.

Kreator zabezpieczeń

Możesz teraz dodać użytkowników do swojego pliku informacyjnego grupy roboczej i przypisać każdemu użytkownikowi hasło i unikatowy identyfikator osobisty (PID). Aby edytować hasło lub identyfikator PID, kliknij nazwę w polu po lewej stronie.

Których użytkowników chcesz umieścić w swoim pliku informacyjnym grupy roboczej?

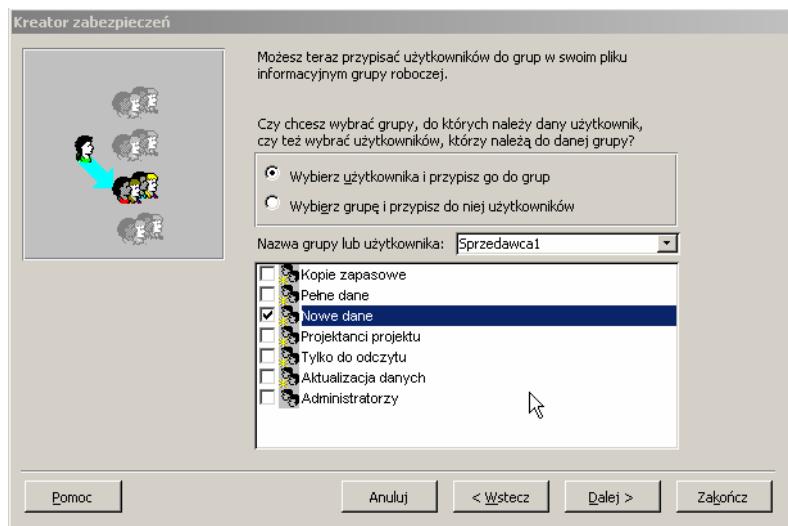
Dodaj nowego użytkownika

Sprzedawca1	Nazwa użytkownika: Sprzedawca2
jan_gor	Haseł: 223456
	Identyfikator PID: 44sibc6gROx57GJHWdy
	Dodaj użytkownika do listy Usuń użytkownika z listy

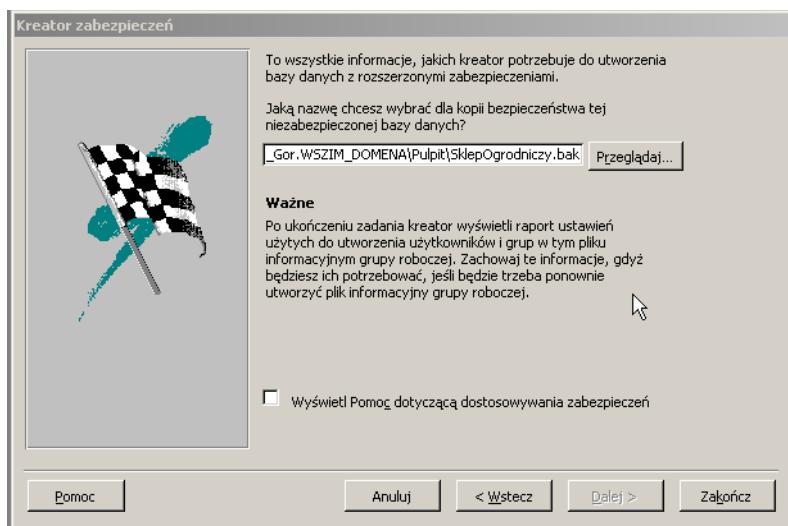
Każdy użytkownik jest unikatowo identyfikowany przez zakodowaną wartość generowaną z nazwy użytkownika i identyfikatora PID. PID jest unikalnym ciągiem alfanumerycznym długości 4-20 znaków.

Pomoc | Anuluj | < Wstecz | Dalej > | Zakończ |

Kolejny ważny etap zabezpieczeń – przypisanie użytkowników do opcjonalnych kont grupowych. Czynność taką powtarzamy dla każdego z (przyszłych) użytkowników bazy.



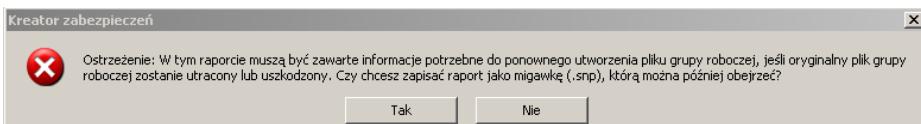
Określenie nazwy i miejsca zapisania niezabezpieczonej bazy danych.



Kreator kończy swoją pracę wygenerowaniem zbiorczego raportu zastosowanych zabezpieczeń.

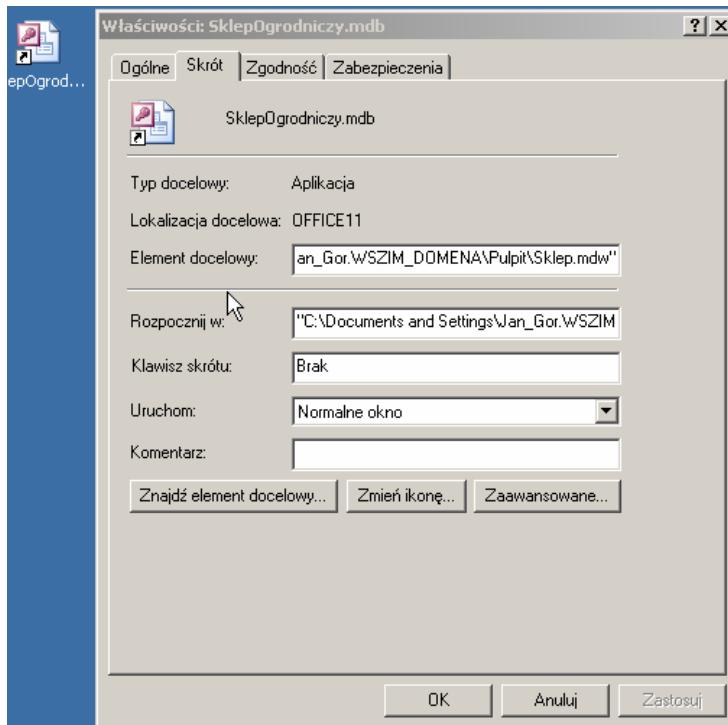


Sugestia zapisania raportu do późniejszego wykorzystania.



Końcowy komunikat kreatora, po jego zamknięciu na pulpicie znajdziemy skrót do zabezpieczonej bazy danych.





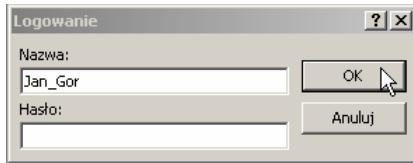
W wierszu *Element docelowy* wpisana jest pokazana niżej instrukcja uruchamiająca bazę danych SklepOgrodniczy .mdb z plikiem systemowym Sklep .mdw.

```
"C:\Program Files\Microsoft Office\OFFICE11\MSACCESS.EXE"
"C:\Documents and Settings\Jan_Gor.WSZIM_DOMENA
\Pulpit\SklepOgrodniczy.mdb" /WRKGRP "C:\Documents and
Settings\Jan_Gor.WSZIM_DOMENA\Pulpit\Sklep.mdw"
```

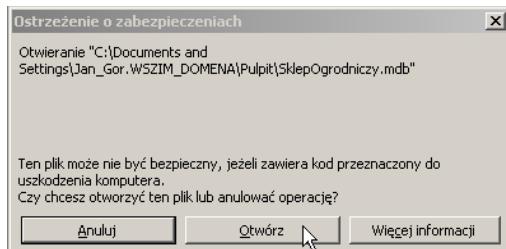
Uruchomienie zabezpieczonej bazy SklepOgrodniczy .mdb bez pośrednictwa utworzonego skrótu ekranowego spowoduje wyświetlenie pokazanego niżej komunikatu o niemożności otwarcia bazy danych z powodu braku odpowiednich uprawnień.



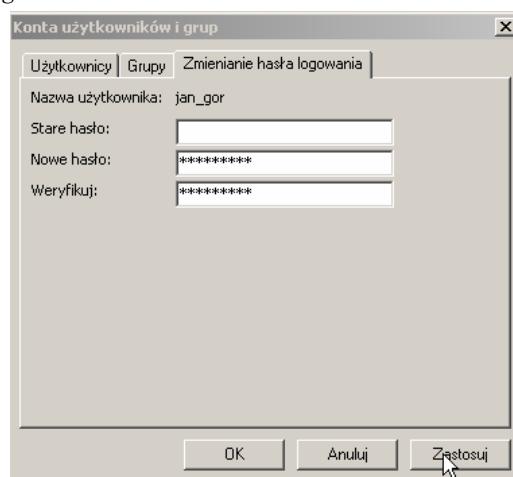
Przy uruchomieniu bazy ze skrótu ekranowego MS Access wyświetla formularz logowania do bazy.



Otwarcie zabezpieczonej bazy będzie możliwe dopiero wtedy, gdy poprawnie zostanie zidentyfikowany użytkownik. W pokazanym przykładzie użytkownik *Jan_Gor* ma uprawnienia administratora bazy danych, tym samym będzie miał prawo dodać dalszych użytkowników do bazy czy też zmodyfikować ich uprawnienia. W tym momencie użytkownik *Jan_Gor* nie ma zdefiniowanego jeszcze hasła, zostanie to za moment zmienione po otwarciu bazy danych. Pokazane niżej okno dialogowe ostrzeżenia o zabezpieczeniach jest efektem ustawienia tych zabezpieczeń na poziomie średnim (menu *Narzędzia/Makro/Zabezpieczenia*).



Po wybraniu przycisku *Otwórz* baza danych jest otwierana, możemy teraz przejść do menu *Narzędzia/Zabezpieczenia/Konta użytkowników i grup*, a następnie do zakładki *Zmianianie hasła logowania*.



Po wpisaniu nowego hasła i jego powtórzeniu, a następnie wybraniu przycisku *Zastosuj* hasło administratora zabezpieczonej bazy *SklepOgrodniczy.mdb* zostało zmienione. W tym przypadku pole *Stare hasło* pozostało puste, co było konsekwencją niezdefiniowania takiego hasła na etapie zabezpieczania bazy danych. Jeżeli teraz zamkniami bazę danych i ponownie ją uruchomimy (z poprzednio utworzonego skrótu ekranowego), to w oknie logowania użytkownik „Jan_Gor” musi już podać poprawne hasło dla otwarcia bazy danych.

7.2.2. Modyfikacja pliku grupy roboczej

Administrator bazy danych może dodawać nowych użytkowników bazy danych, może modyfikować ich uprawnienia, może tworzyć konta grupowe i nadawać im stosowne uprawnienia, może przypisywać użytkowników do kont grupowych, może także usuwać użytkowników bazy danych.

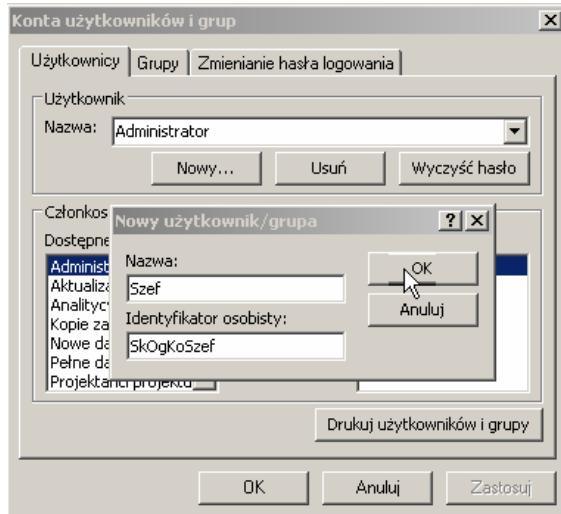
Dla prześledzenie tego problemu utworzymy dwa nowe konta grupowe o nazwach np. *Szefowie* i *Analitycy*, dodamy także dwóch nowych użytkowników o nazwach np. *Szef* i *Analityk* należących do utworzonych wcześniej kont grupowych. Konto *Szefowie* otrzyma uprawnienia jedynie do odczytu danych, konto *Analitycy* dostanie uprawnienia do przeglądania i edycji danych, ale bez tworzenia nowych danych.

Po wywołaniu menu *Narzędzia/Zabezpieczenia/Konta użytkowników i grup* przechodzimy do zakładki *Grupy*, gdzie wybieramy przycisk *Nowa*. Wprowadzamy nazwę i identyfikator dla pierwszej z naszych grup i akceptujemy przycisk *OK*.

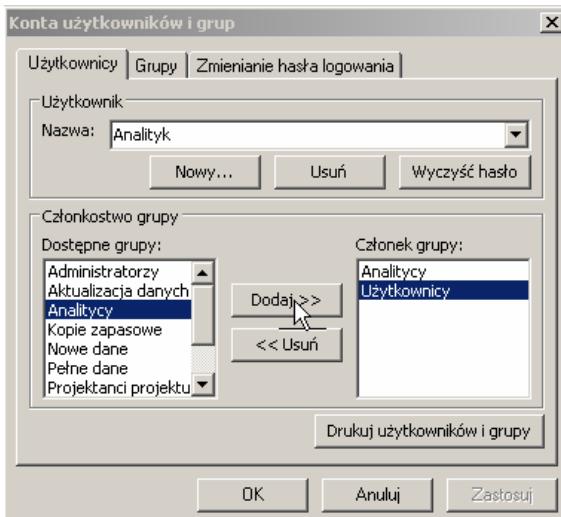


W analogiczny sposób tworzymy konto grupowe *Analitycy*.

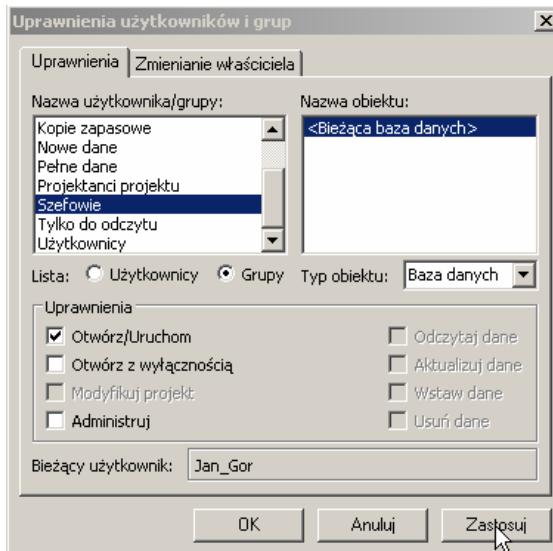
Przechodzimy teraz do zakładki *Użytkownicy*, gdzie przyciskiem *Nowy* otwieramy formularz definiowania nowego użytkownika.



W analogiczny sposób tworzymy użytkownika o nazwie *Analityk*, oczywiście obu dodanych użytkowników przypisujemy (odpowiednio) do wcześniej utworzonych kont grupowych.



Musimy teraz nadać odpowiednie uprawnienia obu utworzonym kontom grupowym. Wykorzystamy w tym celu polecenie *Uprawnienia użytkowników i grup* z menu *Narzędzia/Zabezpieczenia*. W oknie dialogowym tego polecenia w grupie przycisków *Lista* uaktywniamy *Grupy*, a następnie w liście *Nazwa użytkownika/grupy* zaznaczamy grupę *Szefowie*.



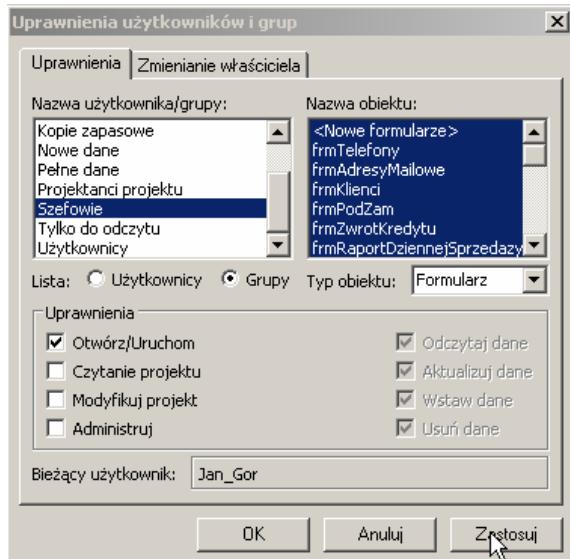
W polu kombi *Typ obiektu* wybieramy pozycję *Baza danych*, a w grupie pól wyboru *Uprawnienia* zaznaczamy *Otwórz/Uruchom*. Przyciskiem *Zastosuj* przypisujemy wybrane uprawnienie do grupy *Szefowie*.

Zmieniamy *Typ obiektu* na *Tabele*, zaznaczamy wszystkie pozycje w liście *Nazwa obiektu* poprzez klik myszą na ostatnią pozycję przy wciśniętym klawiszem Shift i przypisujemy uprawnienie *Odczytaj dane* (także związane z nim uprawnienie *Czytanie projektu*). Istotne jest, aby w liście nazw obiektów w zaznaczeniu znalazła się także pozycja *<Nowe tabele/kwerendy>*, dzięki temu każda nowo utworzona tabela dostanie te same uprawnienia.

Analogiczne uprawnienia nadajemy dla wszystkich kwerend, tutaj też musimy zadbać o to, aby w zaznaczeniu w liście *Nazwa obiektu* znalazły się wszystkie pozycje z listy.

Zarówno w przypadku tabel jak i kwerend grupa *Szefowie* dostała uprawnienia do czytania projektów obu typów obiektów, ale bez prawa ich modyfikacji. Członkowie tej grupy będą mogli jedynie przeglądać dane.

W kolejnym kroku w polu kombi *Typ obiektu* wybieramy *Formularze*, w liście *Nazwa obiektu* zaznaczamy wszystkie pozycje i przypisujemy im uprawnienie *Otwórz/Uruchom*.



W analogiczny sposób postępujemy z rapportami dla grupy *Szefowie*. W przypadku grupy *Analitycy* ustawiamy dokładnie takie same uprawnienie dla bazy danych, dla formularzy i raportów, a dla tabel i kwerend dodajemy uprawnienie *Aktualizuj dane*.

Wydaje się, że powinniśmy dodać jeszcze jedną grupę o nazwie *Sprzedawcy* z uprawnieniami do odczytania danych i wstawiania nowych danych. Dokonamy jeszcze jednej, dość istotnej zmiany polegającej na usunięciu łączy do tabel *tTemp* i *tTempR*, a w ich miejsce zaimportujemy te tabele z bazy wewnętrznej *SklepOgrodniczy_wb.mdb*. Po ich zainportowaniu do bazy *SklepOgrodniczy.mdb* usuwamy je z bazy wewnętrznej. Rozwiążanie takie jest lepsze od poprzedniego z tego powodu, że są to tabele pomocnicze, a więc nie powinniśmy generować niepotrzebnego ruchu przez sieć. Konto grupowe *Sprzedawcy* musi dostać dodatkowo uprawnienie do usuwania danych z obu tabel.

Do grupy *Sprzedawcy* przeniesiemy także dwa wcześniej utworzone konta sprzedawców. Utworzenie tej grupy wynika z konieczności rozpoznawania przynależności aktualnego użytkownika bazy danych do określonej grupy w celu zróżnicowania jego dostępu do obiektów bazy danych. Będzie to wymagać dodania dodatkowych procedur do naszego modułu standardowego i zmodyfikowania funkcji wywołujących poszczególne pozycje menu.

7.2.3. Dodatkowe procedury

Zaczniemy od utworzenia funkcji identyfikującej grupę, do której należy bieżący użytkownik zwracany przez funkcję systemową CurrentUser. Pokazana niżej funkcja została utworzona w module standardowym, za pomocą dwóch zagnieźdzonych funkcji For Each przeglądających kolekcję grup zdefiniowanych w pliku grupy roboczej Sklep.mdw, a dla każdej grupy kolejka użytkowników przypisanych do niej. W momencie zgody bieżącego użytkownika z użytkownikiem danej grupy zwracana jest jej nazwa. Rozwiążanie takie sprawdza się w tych sytuacjach, gdy każdy użytkownik należy do jednej grupy, z wyjątkiem grupy *Users (Użytkownicy)*, a tak jest w naszym przypadku.

```
Public Function JakaGrupa() As String
    Dim usr As Variant, grp As Variant, strUser As String
    Set wsp = DBEngine.Workspaces(0)
    strUser = CurrentUser
    ' przejrzenie kolekcji grup i użytkowników
    For Each grp In wsp.Groups
        For Each usr In grp.Users
            If usr.Name = strUser Then
                If grp.Name <> "Users" Then
                    JakaGrupa = grp.Name
                    Exit Function
                End If
            End If
        Next usr
    Next grp
End Function
```

Przygotujemy teraz dwie procedury, których zadaniem będzie udostępnianie lub nie pozycji w menu „SklepOgrodnicy”.

```
Public Sub DisableEnableButtonMenu(poziomo As Integer, _
    pionowo As Integer, jak As Boolean)
    Dim jg As CommandBarPopup, jgb As CommandBarButton
    Set jg = _
        Application.CommandBars("SklepOgrodnicy"). _
        Controls(poziomo)
    Set jgb = jg.Controls(pionowo)
    jgb.Enabled = jak
End Sub

Public Sub DisableEnablePopUpButtonMenu(poziomo As Integer, _
    pionowo As Integer, pionowo2 As Integer, jak As Boolean)
    Dim jg As CommandBarPopup, _
        jgb As CommandBarButton, kk As CommandBarPopup
```

```

Set jg = _
    Application.CommandBars("SklepOgrodnicy"). _
    Controls(poziomo)
Set kk = jg.Controls(pionowo)
Set jgb = kk.Controls(pionowo2)
jgb.Enabled = jak
End Sub

```

Przed utworzeniem obu procedur konieczne jest **włączenie** w referencjach VBA biblioteki Microsoft Office 11.0 Object Library (menu *Tools/References*).

Pozostało nam jeszcze napisanie procedury zdarzenia *Przy otwarciu* dla formularza startowego naszej aplikacji, czyli dla frmStartUp. Zadaniem tej procedury będzie zróżnicowanie menu użytkownika zależenie od grupy, do której należy bieżący użytkownik bazy danych.

```

Private Sub Form_Open(Cancel As Integer)
Select Case JakaGrupa
Case "Szefowie"
    ' pierwsza pozycja menu
    DisableEnableButtonMenu 1, 1, True
    DisableEnableButtonMenu 1, 2, False
    DisableEnableButtonMenu 1, 3, False
    DisableEnablePopUpButtonMenu 1, 4, 1, True
    DisableEnablePopUpButtonMenu 1, 4, 2, False
    DisableEnablePopUpButtonMenu 1, 4, 3, False
    DisableEnablePopUpButtonMenu 1, 5, 1, True
    DisableEnablePopUpButtonMenu 1, 5, 2, False
    DisableEnablePopUpButtonMenu 1, 5, 3, False
    ' druga pozycja menu
    DisableEnableButtonMenu 2, 1, True
    DisableEnableButtonMenu 2, 2, True
    DisableEnableButtonMenu 2, 3, False
    DisableEnableButtonMenu 2, 4, False
    DisableEnablePopUpButtonMenu 2, 5, 1, True
    DisableEnablePopUpButtonMenu 2, 5, 2, False
    DisableEnablePopUpButtonMenu 2, 5, 3, False
    DisableEnablePopUpButtonMenu 2, 6, 1, True
    DisableEnablePopUpButtonMenu 2, 6, 2, False
    DisableEnablePopUpButtonMenu 2, 6, 3, False
    DisableEnablePopUpButtonMenu 2, 7, 1, True
    DisableEnablePopUpButtonMenu 2, 7, 2, False
    DisableEnablePopUpButtonMenu 2, 7, 3, False
    DisableEnablePopUpButtonMenu 2, 8, 1, True
    DisableEnablePopUpButtonMenu 2, 8, 2, False

```

```
DisableEnablePopUpButtonMenu 2, 8, 3, False
DisableEnablePopUpButtonMenu 2, 9, 1, True
DisableEnablePopUpButtonMenu 2, 9, 2, False
DisableEnablePopUpButtonMenu 2, 9, 3, False
' trzecia pozycja menu
DisableEnableButtonMenu 3, 1, False
' czwarta pozycja menu
DisableEnableButtonMenu 4, 1, True
DisableEnableButtonMenu 4, 2, True
DisableEnableButtonMenu 4, 3, True
Case "Analitycy"
    DisableEnableButtonMenu 1, 1, True
    DisableEnableButtonMenu 1, 2, True
    DisableEnableButtonMenu 1, 3, False
    DisableEnablePopUpButtonMenu 1, 4, 1, True
    DisableEnablePopUpButtonMenu 1, 4, 2, True
    DisableEnablePopUpButtonMenu 1, 4, 3, False
    DisableEnablePopUpButtonMenu 1, 5, 1, True
    DisableEnablePopUpButtonMenu 1, 5, 2, True
    DisableEnablePopUpButtonMenu 1, 5, 3, False
    ' druga pozycja menu
    DisableEnableButtonMenu 2, 1, True
    DisableEnableButtonMenu 2, 2, True
    DisableEnableButtonMenu 2, 3, False
    DisableEnableButtonMenu 2, 4, False
    DisableEnablePopUpButtonMenu 2, 5, 1, True
    DisableEnablePopUpButtonMenu 2, 5, 2, True
    DisableEnablePopUpButtonMenu 2, 5, 3, False
    DisableEnablePopUpButtonMenu 2, 6, 1, True
    DisableEnablePopUpButtonMenu 2, 6, 2, True
    DisableEnablePopUpButtonMenu 2, 6, 3, False
    DisableEnablePopUpButtonMenu 2, 7, 1, True
    DisableEnablePopUpButtonMenu 2, 7, 2, True
    DisableEnablePopUpButtonMenu 2, 7, 3, False
    DisableEnablePopUpButtonMenu 2, 8, 1, True
    DisableEnablePopUpButtonMenu 2, 8, 2, True
    DisableEnablePopUpButtonMenu 2, 8, 3, False
    DisableEnablePopUpButtonMenu 2, 9, 1, True
    DisableEnablePopUpButtonMenu 2, 9, 2, True
    DisableEnablePopUpButtonMenu 2, 9, 3, False
    ' trzecia pozycja menu
    DisableEnableButtonMenu 3, 1, False
    ' czwarta pozycja menu
    DisableEnableButtonMenu 4, 1, True
```

```

DisableEnableButtonMenu 4, 2, True
DisableEnableButtonMenu 4, 3, True
Case "Sprzedawcy", "Admins"
    DisableEnableButtonMenu 1, 1, True
    DisableEnableButtonMenu 1, 2, True
    DisableEnableButtonMenu 1, 3, True
    DisableEnablePopUpButtonMenu 1, 4, 1, True
    DisableEnablePopUpButtonMenu 1, 4, 2, True
    DisableEnablePopUpButtonMenu 1, 4, 3, True
    DisableEnablePopUpButtonMenu 1, 5, 1, True
    DisableEnablePopUpButtonMenu 1, 5, 2, True
    DisableEnablePopUpButtonMenu 1, 5, 3, True
    ' druga pozycja menu
    DisableEnableButtonMenu 2, 1, True
    DisableEnableButtonMenu 2, 2, True
    DisableEnableButtonMenu 2, 3, True
    DisableEnableButtonMenu 2, 4, True
    DisableEnablePopUpButtonMenu 2, 5, 1, True
    DisableEnablePopUpButtonMenu 2, 5, 2, True
    DisableEnablePopUpButtonMenu 2, 5, 3, True
    DisableEnablePopUpButtonMenu 2, 6, 1, True
    DisableEnablePopUpButtonMenu 2, 6, 2, True
    DisableEnablePopUpButtonMenu 2, 6, 3, True
    DisableEnablePopUpButtonMenu 2, 7, 1, True
    DisableEnablePopUpButtonMenu 2, 7, 2, True
    DisableEnablePopUpButtonMenu 2, 7, 3, True
    DisableEnablePopUpButtonMenu 2, 8, 1, True
    DisableEnablePopUpButtonMenu 2, 8, 2, True
    DisableEnablePopUpButtonMenu 2, 8, 3, True
    DisableEnablePopUpButtonMenu 2, 9, 1, True
    DisableEnablePopUpButtonMenu 2, 9, 2, True
    DisableEnablePopUpButtonMenu 2, 9, 3, True
    ' trzecia pozycja menu
    DisableEnableButtonMenu 3, 1, True
    ' czwarta pozycja menu
    DisableEnableButtonMenu 4, 1, True
    DisableEnableButtonMenu 4, 2, True
    DisableEnableButtonMenu 4, 3, True
End Select
End Sub

```

W zasadzie mamy już wszystko zrobione; wyjściowa baza danych została rozdzielona na dwie części: bazę wewnętrzną zawierającą tabele i bazę zewnętrzną będącą interfejsem użytkownika. Baza ta została zabezpieczona przy pomocy kreatora zabez-

pieczęń, utworzony został plik grupy roboczej o nazwie Sklep.mdw, utworzyliśmy nowe grupy użytkowników, nadaliśmy im odpowiednie uprawnienia do bazy danych i jej obiektów. Utworzyliśmy także przykładowych użytkowników naszej bazy przypisując ich do wcześniej utworzonych grup. Dodatkowo dodaliśmy procedury i funkcję dostosowujące menu użytkownika w momencie otwierania bazy danych do uprawnień bieżącego użytkownika. Pokazaliśmy także, jak umieścić wewnętrzną bazę danych na serwerze plików w lokalnej sieci komputerowej i jak zmodyfikować łącza do tabel połączonych.

7.2.4. Utworzenie pliku MDA

Plik bazy danych (*.mdb) można przekonwertować do formatu *.mda, baza danych zapisana w tym formacie nie daje dostępu do kodu VBA, nie pozwala także na zmianę projektu formularzy czy kwerend. W połączeniu z wcześniejszymi zabezpieczeniami znakomicie zwiększy to bezpieczeństwo naszej bazy danych.

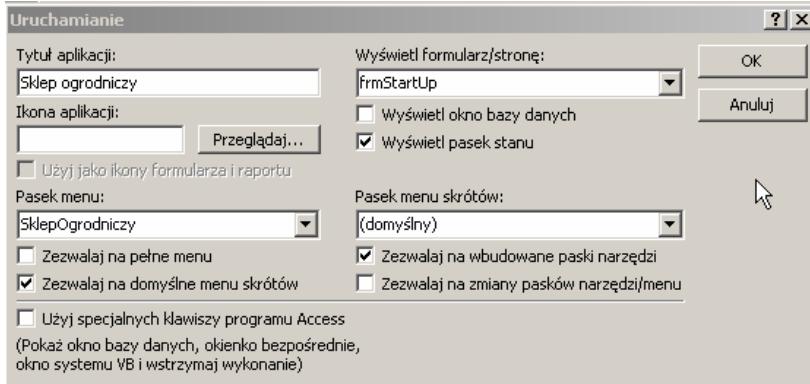
W naszym przypadku przed utworzeniem pliku MDA musimy dokonać konwersji bazy danych do formatu Access 2003, co wynika z faktu, że aplikacja była tworzona na tej wersji MS Access, ale plik bazy był w formacie Access 2000.

Do konwersji wykorzystujemy menu *Narzędzia/Narzędzia bazy danych/Konwertuj bazę danych/Na format pliku Access 2002-2003*. Po konwersji plik bazy danych został zapisany pod nazwą SklepOgrodnicy2003.mdb, teraz ten plik wykorzystamy do utworzenia pliku MDA.

Plik SklepOgrodnicy2003.mdb jest zabezpieczony plikiem informacyjnym grupy roboczej Sklep.mdw, a więc przed jego otwarciem musimy dokonać jednej z dwóch rzeczy: albo zmienimy skrót ekranowy tak, aby wskazywał na ten plik bazy danych, albo zmienimy plik grupy roboczej w MS Access z System.mdw na Sklep.mdw. Oba rozwiązania mają swoje plusy i minusy, rozwiązanie pierwsze jest lepsze o tyle, że nie zmieniamy domyślnego pliku grupy roboczej. Poniżej zmieniona instrukcja właściwości *Element docelowy* skrótu ekranowego.

```
"C:\Program Files\Microsoft Office\OFFICE11\  
MSACCESS.EXE" "C:\Documents and  
Settings\Jan_Gor.WSZIM_DOMENA\Pulpit\SklepOgrodnicy2003.mdb"  
/WRKGRP "C:\Documents and  
Settings\Jan_Gor.WSZIM_DOMENA\Pulpit\Sklep.mdw"
```

Po otwarciu pliku SklepOgrodnicy2003.mdb z uprawnieniami administratora bazy danych sprawdzamy jeszcze ustawienia startowe (menu *Narzędzia/Uruchamianie*).



Po zatwierdzeniu ustawień startowych wywołujemy z menu *Narzędzia/Narzędzia bazy danych* polecenie *Utwórz plik MDE*. Po podaniu nazwy nowego pliku MS Access tworzy plik typu MDE i zapisuje go pod tą nazwą, a po utworzeniu zamknięta tymczasową bazę danych i otwiera plik *SklepOgrodnicy2003.mde*.

Musimy jeszcze raz zmodyfikować skrót ekranowy tak, aby uruchamiać ten właśnie plik, a nie wersję w formacie MDB.

```
"C:\Program Files\Microsoft Office\OFFICE11\MSACCESS.EXE" "C:\Documents and Settings\Jan_Gor.WSZIM_DOMENA\Pulpit\SklepOgrodnicy2003.mde" /WRKGRP "C:\Documents and Settings\Jan_Gor.WSZIM_DOMENA\Pulpit\Sklep.mdw"
```

W sieci Internet przedstawiona wyżej metoda zabezpieczenia bazy danych przy pomocy kreatora zabezpieczeń na poziomie użytkownika jest dość powszechnie krytykowana jako mało skuteczna. Jednym z podstawowych zarzutów jest możliwość uruchomienia zabezpieczonej bazy danych w taki sposób, że zostanie pokazane okno bazy danych. Można tego dokonać poprzez przytrzymanie klawisza Shift w momencie otwierania bazy danych. W naszym przekonaniu nie do końca jest to prawda, jeżeli poprawnie dokonuje się zabezpieczenia bazy danych na etapie pracy kreatora, jeżeli nie pozwoli się na modyfikację pasków narzędzi (inaczej dodaje się przycisk polecenia *Odkryj* i po zabezpieczeniach) i jeżeli umie się zablokować klawisz Shift w momencie otwierania bazy danych, to baza jest solidnie zabezpieczona.

Baza *SklepOgrodnicy2003.mde* została tak właśnie zabezpieczona, niestety metoda jak to zostało zrobione **musi** pozostać tajemnicą autorów tej książki.

Dodatkowym elementem zabezpieczenia obu baz danych w lokalnej sieci komputerowej będą odpowiednie uprawnienia nadane folderowi bazy zewnętrznej oraz folderowi na stacji roboczej lub dysku sieciowym (lepsze rozwiązanie) użytkownika bazy.

Na załączonym krążku CD znajduje się folder SklepOgrodnicy, a w nim wszystkie pliki demonstrujące omówione wyżej techniki zabezpieczenia bazy danych. W folderze znajduje się plik informacyjny grupy roboczej Sklep.mdw, zabezpieczone pliki baz danych SklepOgrodnicy.mdb (format 2000), SklepOgrodnicy2003.mdb oraz plik SklepOgrodnicy2003.mde. Wszystkie bazy korzystają z tabel połączonych bazy SklepOgrodnicy_wb.mdb umieszczonej w folderze FolderDanych.

W folderze SklepOgrodnicy umieściliśmy także skrót ekranowy wskazujący na uruchomienie bazy SklepOgrodnicy2003.mde z wykorzystaniem pliku Sklep.mdw. Dla sprawdzenia działania tych baz prosimy o skopiowanie całego folderu SklepOgrodnicy na dysk lokalny C do jego folderu głównego. Do listy użytkowników tej bazy dodaliśmy jeszcze jednego użytkownika o nazwie *boss* i hasło *sklep12345* z uprawnieniami administratora bazy SklepOgrodnicy.

7.2.5. Konfiguracja stacji roboczej klienta

W przypadku pracy sieciowej jednym z lepszych rozwiązań jest udostępnienie każdemu z użytkowników bazy folderu na dysku sieciowym. Powiedzmy, że będzie to folder Jan.Kowalski na komputerze (serwerze plików) Gamma. Administrator sieciowej bazy danych umieści w tym folderze kopię pliku Sklep.mdw oraz kopię zabezpieczonej bazy SklepOgrodnicy2003.mde. Na pulpicie stacji roboczej użytkownika „Jan Kowalski” tworzy skrót ekranowy przypisując właściwości *Element docelowy* pokazaną poniżej instrukcję.

```
"C:\Program Files\Microsoft Office\OFFICE11\  
MSACCESS.EXE" "\\Gamma\Jan.Kowalski\SklepOgrodnicy2003.mde"  
/WRKGRP "\\Gamma\Jan.Kowalski\Sklep.mdw"
```

Administrator sieci lokalnej może na oba pliki nałożyć takie uprawnienia, które pozwolą wyłącznie na ich wykonanie – bez prawa zajrzenia do tego folderu czy skopowania tych plików. Jest to kolejny element zabezpieczenia bazy danych przed jej nieuprawnionym wykorzystaniem.

Na zakończenie naszej książki chcemy wyrazić nadzieję, że Czytelnik znalazł w niej wiele przydatnych informacji. Mamy też świadomość, że wiele tematów zostało jedynie zasygnalizowanych, z pewnością są też takie, które w ogóle nie zostały przez nas poruszone, może inaczej trzeba było rozłożyć akcenty. No cóż, nasza pozycja jest jedną z wielu dostępnych na rynku wydawniczym, jest także wiele ciekawych miejsc w sieci Internet poświęconych problematyce MS Access i wykorzystania VBA, tam też warto szukać brakujących informacji.

8. Literatura

1. Alison Balter, *Access 2003 PL dla każdego*, Helion, 2004, tłum. Tomasz Pędziwiatr
2. Ben Forta, *SQL w mgnieniu oka*, Helion 2004
3. Bill Marklyn, Mark Whitehorn, *Relacyjne bazy danych*, Helion 2003
4. Bogdan Czogalik, *Access 2002. Tworzenie baz danych*. Helion 2003
5. Charles E. Brown, *Access. Programowanie w VBA*, Helion 2005
6. Danuta Mendrala, Marcin Szeliga, *Access 2003 Pl. Kurs*, Helion, 2003
7. Helen Feddema, *Microsoft Access wersja 2002 dla ekspertów*, Read Me 2003
8. Hugh Darwen, C. J. Date, *SQL. Omówienie standardu języka*. Wydawnictwo Naukowo-Techniczne, 2000
9. John Colby, Paul Wilton, *SQL od podstaw*. Helion 2005
10. Marek Jeznach, *Visual Basic w Accessie od podstaw*. Translator S.C. 2004
11. Michael R. Irwin, Jennifer Reardon, Cary N. Prauge, *Access 2003 Pl. Biblia*, Helion 2004, tłum. Grzegorz Werner i inni.
12. P. Cardoza, *Access 2003 Programmer's Reference*, John Wiley & Sons, 2004
13. Paul Benon-Davies, *Systemy baz danych*, Wydawnictwo Naukowo-Techniczne, 2003, tłum. Marcin Bałachowski, Lech Banachowski
14. Piotr Stokłosa, *Microsoft Office Access 2003 krok po kroku*, Read Me, 2004
15. Rick Dobson, *Programowanie Access 2000*, Read Me, 2000, tłum. Janusz Machowski
16. *SQL Access to SQL Server*, APress, 2002
17. Stephen Forte, *Access 2000. Księga eksperta*, Helion 2001
18. Steven Roman, *Access. Baza danych – projektowanie i programowanie*. Helion 2001

9. Indeks

A

ADO, 108
 Connection, 108
 Recordset, 108
 ANSI, 81

B

Baza
 kartotekowa, 8
 plik MDE, 366
 projektowanie, 11
 relacyjna, 9
 rozdzielacz bazy danych, 347

C

Connection
 CurrentProject, 106, 110, 114, 161, 203,
 212, 235, 241-242, 252, 261, 265-267,
 270, 279, 283, 285, 293, 299, 308, 319,
 325, 329
 metoda Close, 106, 120, 235, 241, 243,
 252, 261, 265-267, 268, 272, 280, 284,
 286, 295, 299, 309, 319, 325, 330
 metoda Execute, 119, 203, 235, 241-242,
 266-268, 271, 279, 284, 286, 294, 299
 możliwości, 109
 przykład, 110, 114, 116

D

DAO, 108
 DDL, 81
 Delete, *Patrz* Polecenia SQL
 DML, 81
 Dodatkowe formanty, 303

E

Encja, 7

F

Formularz złożony
 formularz główny, 46
 odwołanie do kontrolki podformularza, 58,
 60, 62, 171, 234, 282
 podformularz, 46
 pola łączące, 154
 pola wiązające, 48
 poziom zagnieżdżenia, 46
 projektowanie, 150, 220
 projektowanie podformularza, 150
 umieszczanie podformularza, 47, 153
 widok podformularza, 52
 widok projektu, 154, 233

Formularze

obliczenia w formularzach, 54
 proste, 38
 złożony, 46

Funkcje

CCur, 26, 60, 79, 236, 279, 288, 299, 301,
 311, 316, 326
 CDate, 305
 CStr, 278
 CurrentUser, 362
 Dlookup, 59
 Format, 148, 222
 IIf, 59, 148
 IsNull, 100, 107, 245, 247, 249, 251, 254,
 265, 280-281, 289, 293, 299
 LTrim, 76, 182
 Now, 76, 222
 Round, 214, 271, 301, 311, 316, 331
 Str, 76, 103, 182, 214, 238, 259, 267, 279,
 280, 290, 298, 318, 330
 Suma, 56, 79, 211
 Year, 76, 87

Funkcje Accessa

Suma, 281, 316

I

Insert into, *Patrz* Polecenia SQL

K

- Klasa
 - metoda, 100
 - nowa instancja, 293
 - procedury Let i Get, 99
 - właściwości, 99
 - zmienne prywatne, 99
- Konstruktor wyrażeń, 71
- Kontrolka kalendarza, 303
- Kwerendy
 - akcyjna, 20
 - aktualizująca, 30, 188, 194
 - definiowanie kryteriów, 24
 - dodającąca, 34, 63
 - funkcja Suma, 28
 - grupowanie, 28
 - grupująca, 79, 200, 237, 312
 - ikony akcyjnych, 37
 - krzyżowa, 29, 209
 - operator And, 27
 - operator Between, 27
 - operator Or, 25
 - parametry kwerendy, 184, 264
 - parametryczna, 37, 183, 237, 263, 289, 310, 320
 - pole wyliczane, 22, 26, 147, 157, 182, 259, 288-289, 311
 - projektowanie, 21, 143, 147, 152, 156
 - rekordy unikatowe, 24
 - składająca, 20
 - sortowanie, 23
 - uruchamianie aktualizującej, 31
 - ustawienia systemowe, 32
 - usuwająca, 33, 55
 - wartości unikatowe, 73, 276
 - wybierająca, 20, 65, 73, 193, 236, 287, 316

M

- Makropolecenie
 - projektowanie, 205
 - widok projektu, 207
- Menedżer tabel połączonych, 349
- Menu użytkownika
 - dodawanie polecen, 335
 - funkcje obsługujące, 338

- funkcje udostępniające, 362
- projektowanie, 334
- przypisywanie akcji, 341
- standardowe przyciski, 336
- właściwości, 335
- MS Access
 - uruchamianie, 218, 345, 366

N

- Normalizacja, 16

O

- Operator
 - And, 25, 86
 - Between, 87
 - In, 88
 - Like, 25, 86, 206
 - Or, 86

P

- Pasek narzędziowy
 - dodawanie przycisków, 216
 - właściwości, 216
- Pętle
 - Do ... Loop, 101, 105, 115, 294
 - For ... Next, 104, 160, 235, 241-243, 265, 286, 309
 - For Each, 106, 114, 362
 - licznik pętli, 104
 - rodzaje pętli, 104
 - wyjście warunkowe, 105
- Pole kombi
 - instrukcja SQL, 61
 - właściwości, 145, 148, 152, 157, 183, 186, 193, 224-225, 230-231, 277-278, 308
 - właściwość Column, 276, 279, 298-299, 321
- Pole tekstowe
 - właściwości, 124
- Polecenia SQL
 - alias pola, 85
 - As, 85
 - delete, 90, 203, 241-242, 284, 286
 - Distinct, 83

-
- insert into, 89, 235, 241, 266, 271, 286, 294
 - Join, 83
 - konstruktor kwerend, 259, 276, 297
 - łączenie tekstów, 85
 - Order by, 88
 - Right Join, 85
 - select, 82, 224, 226-227, 242, 247, 252, 265, 270, 277-278, 281, 283, 294, 298, 308, 316, 326, 330
 - update, 243, 266-268, 272, 279, 283, 299
 - Where, 86
 - Polecenie DoCmd
 - metoda Close, 96, 166, 176-177, 189, 195, 241, 245, 248, 252, 255, 272, 286, 296, 299, 305, 309, 342
 - metoda OpenForm, 163, 173, 175, 187, 246, 248, 252-253, 256, 294, 338, 339, 340, 342
 - Procedura
 - argumenty, 96
 - Projektowanie formularza
 - formant karta, 314
 - formanty, 39
 - kolejność dostępu, 45
 - konstruktor makr, 58
 - kreator instrukcji SQL, 44
 - lista pól, 143
 - makropolecenie, 57, 61, 77
 - nowy formularz, 139, 142, 157
 - okno właściwości, 39
 - pola wyliczane, 56
 - pole kombi, 43, 143
 - pole podsumowania, 232, 276, 281
 - przelaczniki, 205
 - przybornik, 39
 - sekcje, 39
 - umieszczanie kontrolek, 41
 - właściwości pola kombi, 44
 - właściwość Cykliczny, 60
 - zdarzenia, 160
 - Projektowanie raportu
 - nowy raport, 210
 - numeracja stron, 71
 - numerowanie rekordów, 74
 - pole podsumowania, 312
 - raport dynamiczny, 212
 - raport prosty, 67
 - raport złożony, 72
 - sortowanie i grupowanie, 68, 317, 331
 - suma bieżąca, 69
 - widok projektu, 312, 317, 321, 323, 327, 331
 - Przekazanie argumentów, 97
 - Przycisk polecenia
 - właściwość Przy kliknięciu, 164, 174, 175

R

- Raport, *Patrz* Projektowanie raportu
- Raport złożony
 - projektowanie podraportu, 237, 288
 - widok projektu, 239, 290
- RDO, 108
- RecordCount, *Patrz* Właściwość
- Recordset
 - aktualizacja danych, 112
 - BOF, 112
 - EOF, 112, 115
 - kursor, 111
 - metoda AddNew, 117, 161
 - metoda CancelUpdate, 120
 - metoda Close, 106, 117, 120, 161, 243, 252, 261, 265, 271, 283-284, 286, 293, 295, 309, 319, 325, 330
 - metoda Delete, 119
 - metoda Filter, 119
 - metoda MoveNext, 115-116, 243, 261, 271, 286, 294, 309, 319, 325, 330
 - metoda Open, 106, 110, 113-117, 119, 161, 171, 185, 212, 242, 252, 261, 265, 270, 283, 286, 294, 309, 319, 325, 330
 - metoda Requery, 120
 - metoda Update, 116, 118
 - nawigacja, 112
- Relacje
 - edytowanie, 19
 - edytowanie relacji, 129
 - jeden-do-wielu, 84, 127, 129, 222
 - kaskadowe usuwanie, 33
 - przykład, 132, 136, 138, 220, 223
 - tworzenie relacji, 128
 - typy relacji, 17
 - więzy integralności, 18, 33
 - właściwości sprzężenia, 73, 201, 202

S

Select, *Patrz* Polecenia SQL
 SQL, 81
 Struktury warunkowe
 Choose, 108, 289
 If, 107, 165, 170, 245, 254, 294
 Select case, 107, 114, 116, 187, 215, 271,
 340, 342, 363

T

Tabela, 12
 czwarta postać normalna, 16
 druga postać normalna, 13
 klucz, 8
 klucz prosty, 12, 123
 klucz złożony, 14–15, 135
 pierwsza postać normalna, 11
 pole, 7
 połączona, 348
 projektowanie nowej, 122
 przykładowy projekt, 127, 180, 292
 redundancja danych, 11
 rekord, 7
 relacja, 9
 trzecia postać normalna, 13
 właściwości pól, 126
 zależność funkcyjonalna, 15
 zależność wielowartościowa, 15
 zapisywanie, 124

U

Update, *Patrz* Polecenia SQL

V

VBA, 91
 analiza błędu, 249
 deklaracja stałej, 95, 246
 deklaracja zmiennych, 93
 dwukropki, *Patrz* separator instrukcji
 funkcja, 98
 kolekcja, 106, 362
 moduł, 92
 moduł klasy, 98

okna edytora, 92
 On Error Resume Next, 166, 342
 On Error Goto, 245, 247, 251, 254
 procedura, 96
 Redim, 261
 Run-time error, 248
 separator instrukcji, 295
 struktury warunkowe, 107
 uruchomienie edytora, 91
 zasięg zmiennych, 93
 zmienna obiektowa, 94
 zmienna tablicowa, 93
 zmienna użytkownika, 95

W

Właściwości
 formantu ActiveX, 304
 RowSource, *Patrz* źródło wierszy
 typ źródła wierszy, 308, 319, 325
 wartość domyślna, 296
 źródło formantu, 238, 240, 259
 źródło wierszy, 308, 316, 319
 Właściwości formularza
 cykliczny, 61, 181
 filtr, 205
 okno właściwości, 141
 paski przewijania, 148, 151
 przyciski Min Max, 148
 rekord bieżący, 232
 widok domyślny, 151
 wprowadzanie danych, 61, 181
 źródło rekordów, 151
 Właściwości pól tabeli
 reguła poprawności, 134
 typ danych, 126
 wartość domyślna, 134, 137, 222
 wymagane, 126, 127, 137

Właściwość

 paska narzędzi, 344
 RecordCount, 110, 117, 119, 120, 243,
 252, 261, 270–272, 286, 309, 319, 325,
 330
 suma bieżąca, 70

Z

Zabezpieczenie bazy
dodanie nowego użytkownika, 359
dodanie nowych grup, 358
formularz logowania, 356
grupa Użytkownicy, 353
konta grupowe, 352
kreator zabezpieczeń, 351
nadawanie uprawnień, 360-361
nowi użytkownicy, 353
plik grupy roboczej, 350
raport kreatora, 355
skrót ekranowy, 356, 367

zmiana hasła logowania, 357
Zdarzenia formularza
po wstawieniu, 283
przed aktualizacją, 231, 242, 249, 280
przy bieżącym, 241, 281
przy otwarciu, 162, 165, 168, 170, 176,
187, 215, 245, 253-254, 261, 363
Zdarzenia raportu
przy otwarciu, 326, 330
Zdarzenie pola kombi
After update, Patrz po aktualizacji
po aktualizacji, 185, 194, 197, 226, 231,
260, 265, 277, 279, 299, 308
przed aktualizacją, 279