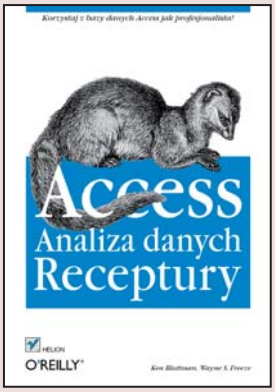
Access. Analiza danych. Receptury



Autor: Ken Bluttman, Wayne Freeze
Tłumaczenie: Grzegorz Kowalczyk
ISBN: 978-83-246-1285-7
Tytuł oryginału: Access Data Analysis Cookbook
Format: B5, stron: 360

Korzystaj z bazy danych Access jak profesjonalista!

- Jak stosować wskaźniki statystyczne do analizy danych biznesowych?
- Jak rozszerzać funkcjonalność zapytań SQL, stosując skrypty VBA?
- Jak przetwarzać dane i przenosić je między bazami Access?

Access to znane już narzędzie służące do wszechstronnego przetwarzania i analizy danych. Posiada sporo ukrytych mechanizmów, pozwalających efektywnie wykonywać zadania, które początkowo mogą wydawać się skomplikowane. Książka przedstawia przykłady kwerend, metody przenoszenia danych pomiędzy bazami Access, obliczania wielu wskaźników finansowo-biznesowych i sporo innych zagadnień – wszystko pod kątem analizy i przetwarzania danych. Każda zaprezentowana receptura jest opatrzona kompletnym opisem rozwiązania problemu wraz ze szczegółowym omówieniem metody postępowania oraz analizą kodu.

Access. Analiza danych. Receptury to uniwersalny podręcznik przeznaczony zarówno dla początkujących użytkowników bazy danych Access, jak i doświadczonych. Dzięki przejrzystemu językowi i mnogości poruszonych zagadnień każdy, niezależnie od stopnia zaawansowania, może poszerzyć swoją wiedzę. Zawiera mnóstwo ciekawych wskazówek i technik ułatwiających codzienną pracę z bazami danych, co czyni ją atrakcyjną nawet dla osób doskonale posługujących się bazą Access. Jest to także kompendium wiedzy niezbędnej każdemu, kto chce wyciągać ze zbiorów danych naprawdę cenne informacje.

- Tworzenie kwerend różnych typów
- Wstawianie, aktualizacja i usuwanie danych
- Przetwarzanie tekstu i liczb zapisanych w formie łańcucha znaków
- Zastosowanie tabel, modyfikacja zawartości systemu Windows, szyfrowanie danych
- Wykorzystanie obiektu FileSystemObject, przetwarzanie danych XML oraz XSLT, komunikacja z bazami SQL
- Rozwiązywanie problemów biznesowych
- Obliczanie wskaźników statystycznych

Baza danych to fundament biznesu – zobacz, jak efektywnie nią zarządzać!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 3.

Kwerendy funkcjonalne

Kwerendy dzielimy na pasywne i funkcjonalne. *Kwerendy pasywne*, takie jak standardowe kwerendy wybierające, zwracają zestawy rekordów spełniających kryteria wyszukiwania, ale w żaden sposób nie modyfikują danych (kwerendy pasywne nie modyfikują danych w tabelach źródłowych ani nie przechowują zwracanych rekordów dłużej niż dana kwerenda jest aktywna).

Kwerendy aktywne mogą modyfikować dane źródłowe, a zestawy rekordów zwracanych przez takie kwerendy mogą być dostępne w nieskończoność. Na przykład kwerenda usuwająca, jak sama nazwa wskazuje, usuwa rekordy z tabel źródłowych bazy danych — jest to procedura całkowicie niszcząca dane. Jeżeli takie usunięte informacje nie zostały wcześniej zapisane w kopii bezpieczeństwa, to ich odzyskanie nie będzie możliwe. Kwerendy aktualizujące również mogą modyfikować dane źródłowe, zmieniając informacje przechowywane w poszczególnych rekordach — podobnie jak w przypadku kwerend usuwających, jeżeli aktualizowane rekordy nie zostały wcześniej zapisane w kopii bezpieczeństwa, to po ich aktualizacji przywrócenie poprzednich wartości nie będzie możliwe.

Kwerendy dołączające oraz kwerendy tworzące tabele należą do grupy kwerend funkcjonalnych, które nie modyfikują danych źródłowych, ale za to powodują, że zwracane rekordy są przechowywane dłużej, niż wynosi czas aktywności kwerendy. Kwerendy dołączające dodają zwracane rekordy do istniejących tabel, a kwerendy tworzące tabele umieszczają zwracane rekordy w nowych tabelach. W tym rozdziale będziemy szczegółowo omawiali wszystkie cztery rodzaje kwerend funkcjonalnych.

3.1. Kwerendy aktualizujące

Opis problemu

Mamy daną tabelę, której rekordy wymagają modyfikacji. Pole *Nazwa stanu* przechowuje dwuliterowe skróty nazw stanów. Naszym zadaniem jest zamiana tych skrótów na pełne nazwy stanów. Jak tego dokonać?

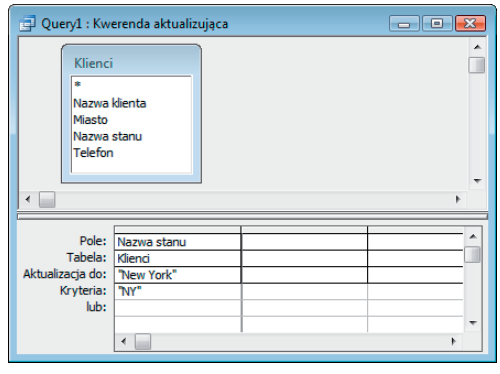
Rozwiązanie

Rozwiązaniem problemu będzie zastosowanie kwerendy aktualizującej, którą możemy w prosty sposób utworzyć z poziomu widoku projektu kwerendy. Aby tego dokonać, należy po przełączeniu do widoku projektu kwerendy wybrać z menu głównego polecenie *Kwerenda/Kwerenda aktualizująca*, co spowoduje odpowiednie przygotowanie siatki projektu kwerendy.

W wersji Access 2007 należy najpierw przy użyciu *Wstążki* utworzyć pusty projekt kwerendy, a następnie na karcie *Projektowanie* w grupie *Typ kwerendy* kliknąć polecenie *Aktualizuj*.

Po utworzeniu kwerendy aktualizującej w siatce projektu kwerendy pojawi się dodatkowy wiersz o nazwie *Aktualizacja do*, natomiast znikną wiersze *Sortuj* i *Pokaż* znane z kwerend wybierających.

Na rysunku 3.1 przedstawiono projekt kwerendy aktualizującej, której zadaniem jest zastąpienie w polu *Nazwa stanu* wszystkich wystąpień akronimu `NY` na pełną nazwę `New York`.

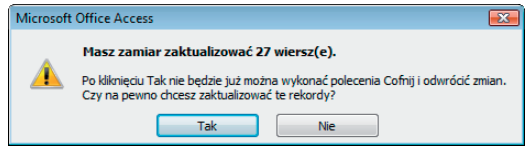


Rysunek 3.1. Prosta kwerenda aktualizująca

Zwróćmy uwagę, że w wierszu *Kryteria* znajduje się wartość `NY` — jest to wartość, która będzie poszukiwana i zamieniana na nową. Wartość wpisana w wierszu *Aktualizacja do* — `New York` — to nowa wartość, która zastąpi wartość poszukiwaną. Odpowiednie użycie kryteriów ma tutaj ogromne znaczenie, bo przecież chcemy zmodyfikować tylko rekordy, które w polu *Nazwa stanu* mają wartość `NY`.

Kwerenda aktualizująca nie musi posiadać klauzuli `Where` (która odpowiednio filtruje rekordy i pozwala na aktualizację tylko tych, które są zgodne z podanym wzorcem wyszukiwania) — kwerenda aktualizująca może aktualizować wszystkie rekordy w bazie. W pewnych sytuacjach taki zakres działania kwerendy może być bardzo pożądany, ale w naszym przykładzie aktualizacja wszystkich rekordów w tabeli klientów byłaby prawdziwą katastrofą (jak pamiętamy, jeżeli nie mamy kopii zapasowej tabeli, to nie możemy anulować zmian wprowadzonych przez kwerendę aktualizującą i przywrócić poprzedniej zawartości tabeli).

Po uruchomieniu kwerendy na ekranie pojawi się okno dialogowe z prośbą o potwierdzenie zamiaru wykonania takiej operacji, przedstawione na rysunku 3.2. Aby kontynuować, należy nacisnąć przycisk *Tak*.



Rysunek 3.2. Okno dialogowe z prośbą o potwierdzenie zamiaru wykonania aktualizacji

Przedstawiona powyżej kwerenda aktualizuje nazwę tylko jednego stanu. Jest to rozwiązanie do przyjęcia, aczkolwiek uruchamianie podobnej kwerendy aktualizującej 50 razy (po jednej kwerendzie aktualizującej dla każdego stanu) może być dosyć czasochłonne i nużące. Na szczęście istnieją również bardziej efektywne metody.

Jednym z możliwych rozwiązań jest wywołanie z poziomu kwerendy własnej funkcji przetwarzającej dane. W naszym przypadku taka funkcja powinna pobierać akronim nazwy stanu i zwracać jego pełną nazwę. Poniżej przedstawiamy przykładowy kod takiej funkcji:

```
Function new_state_name(current_state_name As String) As String
 new_state_name = current_state_name
   If current_state_name = "NY" Then new_state_name = "New York"
   If current_state_name = "CT" Then new_state_name = "Connecticut"
   If current_state_name = "MA" Then new_state_name = "Massachusetts"
   If current_state_name = "CA" Then new_state_name = "California"
End Function
```

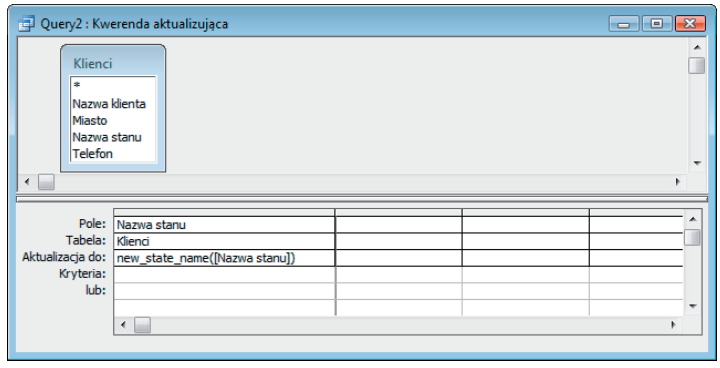
Jak widać, dla uproszczenia w kodzie naszej przykładowej funkcji zakodowaliśmy zaledwie kilka akronimów nazw stanów, ale oczywiście nic nie stoi na przeszkodzie, aby umieścić tam wszystkie 50 nazw stanów (bądź równie dobrze tylko niektóre, wybrane nazwy stanów). Kodowanie wszystkich 50 stanów może być nieco żmudnym zajęciem, ale przynajmniej w efekcie otrzymamy dosyć uniwersalną, w pełni użyteczną funkcję.

Pierwszy wiersz kodu funkcji ustawia wynik działania funkcji na wartość będącą argumentem funkcji (bieżący akronim nazwy stanu), dzięki czemu jeżeli pełna nazwa stanu nie zostanie odnaleziona, funkcja zwróci akronim nazwy stanu i w zasadzie w takim rekordzie nic się nie zmieni. Jeżeli jednak akronim będący argumentem funkcji pasuje do któregoś z poleceń `If`, funkcja jako wynik swojego działania zwraca pełną nazwę stanu.

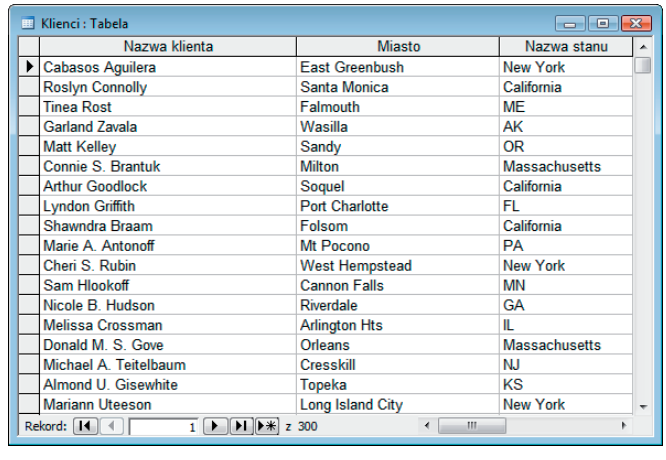
Na rysunku 3.3 przedstawiono wygląd siatki projektu takiej kwerendy aktualizującej. Zwróćmy uwagę na fakt, że tym razem nie mamy ustawionych żadnych kryteriów. Brak kryteriów wyszukiwania wynika z prostego faktu, że chcemy przetwarzać wszystkie rekordy tabeli. Sama funkcja wywoływana jest w wierszu *Aktualizacja do*, a jako argumentu wywołania funkcji używamy wartości pola *Nazwa stanu*.

Wyniki działania kwerendy przedstawiono na rysunku 3.4. Jak widać, dla tych stanów, które zostały w funkcji odpowiednio zakodowane przy użyciu poleceń `If`, akronimy zostały zastąpione pełnymi nazwami stanów; dla wszystkich pozostałych stanów oryginalne akronimy nie zostały zaktualizowane.

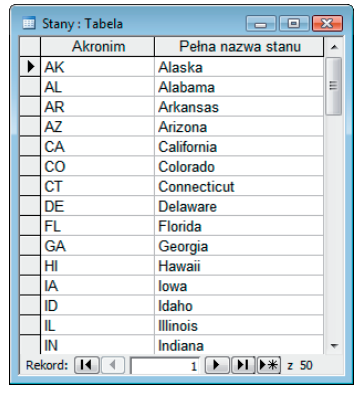
Jeszcze innym sposobem dokonania konwersji akronimów na pełne nazwy stanów jest zastosowanie wbudowanej funkcji `Dlookup`. W naszym przykładzie użyjemy tabeli *Stany*, przechowującej nazwy stanów. Tabela posiada dwa pola: *Akronim* oraz *Pełna nazwa stanu*. Wygląd tej tabeli został przedstawiony na rysunku 3.5.



Rysunek 3.3. Zastosowanie własnej funkcji w kwerendzie aktualizującej

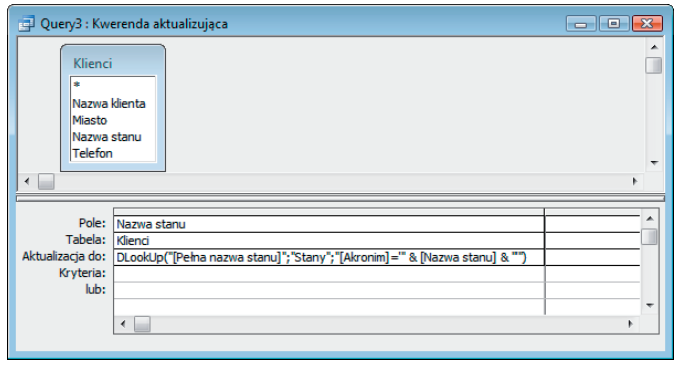


Rysunek 3.4. Wyniki działania kwerendy aktualizującej



Rysunek 3.5. Tabela przechowująca pełne nazwy stanów i ich akronimy

Wygląd siatki projektu kwerendy został przedstawiony na rysunku 3.6.



Rysunek 3.6. Kwerenda aktualizująca z funkcją DlookUp

Wywołanie funkcji `DlookUp` następuje w wierszu *Aktualizacja do* i wygląda następująco:

```
DLookUp("[Pełna nazwa stanu]";"Stany";"[Akronim]='" & [Nazwa stanu] & "'")
```

Pełna nazwa stanu oraz *Akronim* to nazwy dwóch pól tabeli *Stany*, a *Nazwa stanu* to pole w tabeli *Klienci*. W tabeli *Stany* każdy stan posiada swój rekord przechowujący pełną nazwę stanu oraz jej akronim. Dzięki takiemu rozwiązaniu po uruchomieniu nasza kwerenda dokona zamiany akronimów na pełne nazwy dla wszystkich 50 stanów (pod warunkiem oczywiście, że w tabeli *Stany* nie ma żadnych błędów).

Omówienie

Wszystkie kwerendy aktualizujące, o których mówiliśmy do tej pory, pracowały tylko na jednym polu tabeli. Kod SQL kwerendy przedstawionej na rysunku 3.1 jest krótki i wygląda następująco:

```
UPDATE Klienci SET Klienci.[Nazwa stanu] = "New York"
WHERE (((Klienci.[Nazwa stanu])="NY"));
```

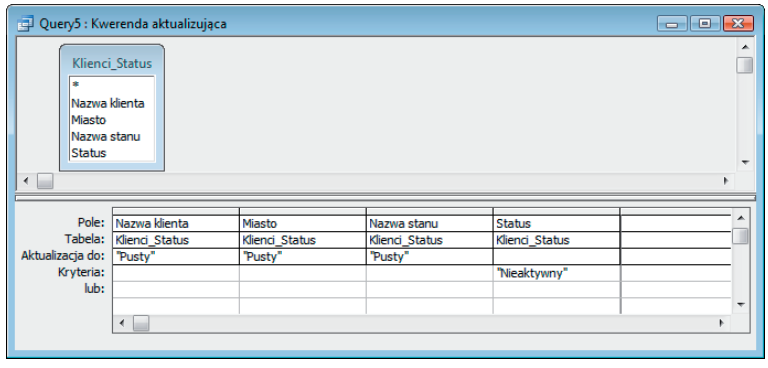
Kod SQL kwerendy aktualizującej zawsze rozpoczyna się od słowa kluczowego `UPDATE`, po którym następuje nazwa tabeli i klauzula `SET` wskazująca pole, które będzie aktualizowane (jednocześnie możemy aktualizować więcej niż jedno pole, o czym będziemy mówić już za chwilę). Dowolne kryteria wyszukiwania używane do ograniczenia liczby aktualizowanych rekordów są definiowane w klauzuli `WHERE`.

Kryteria wyszukiwania nie muszą bazować na polu, które jest aktualizowane. W poprzednim przykładzie kryteria wyszukiwania operowały co prawda na aktualizowanym polu, ale porównajmy kod poprzedniej kwerendy z następującym zapytaniem SQL:

```
UPDATE Klienci SET Klienci.[Nazwa stanu] = "New York"
WHERE (((Klienci.Miasto)="New York City"));
```

W tym przykładzie pole *Nazwa stanu* jest aktualizowane tylko dla tych rekordów, w których pole *Miasto* ma wartość `New York City`.

Jedna kwerenda może aktualizować dowolną ilość pól, aczkolwiek należy pamiętać o tym, że dla wszystkich pól obowiązują te same kryteria wyszukiwania zdefiniowane w kwerendzie. Na rysunku 3.7 przedstawiono projekt kwerendy, która dla wszystkich klientów mających status `Nieaktywny` ustawia wartość trzech pól na `Pusty`.



Rysunek 3.7. Kwerenda aktualizująca kilka pól jednocześnie

Kod SQL kwerendy przedstawionej powyżej wygląda następująco:

```
UPDATE Klienci_Status SET
Klienci_Status.[Nazwa klienta] = "Pusty",
Klienci_Status.Miasto = "Pusty",
Klienci_Status.[Nazwa stanu] = "Pusty"
WHERE (((Klienci_Status.Status)="Nieaktywny"));
```

Zgodnie z wymogami składni języka SQL kwerenda rozpoczyna się od słowa kluczowego `UPDATE`, po którym następuje nazwa tabeli i klauzula `SET`. Następnie wymieniane są wszystkie pola, którym przypisywane są nowe wartości. Kod kwerendy kończy klauzula `WHERE` definiująca kryteria wyszukiwania (o ile oczywiście takie kryteria zostały zdefiniowane).

3.2. Dołączanie danych

Opis problemu

Chcemy, aby rekordy zwracane przez daną kwerendę były dołączane do innej tabeli. Jak można tego dokonać?

Rozwiązanie

Bardzo często spotykamy się z koniecznością archiwizacji starszych danych, zakończonych transakcji i innych tego typu informacji. Zazwyczaj takie operacje są realizowane poprzez przeniesienie odpowiednich rekordów do innych tabel przechowujących zarchiwizowane czy też historyczne dane. Takie tabele mają zazwyczaj identyczną strukturę jak tablice źródłowe, dzięki czemu przenoszenie rekordów pomiędzy nimi jest bardzo proste i wygodne. Warto jednak pamiętać o tym, że nie jest to żaden twardy wymóg — tabele przechowujące dane archiwalne mogą mieć

dodatkowe pola, w których umieszczane są takie informacje jak data przeniesienia rekordu do archiwum, kto zatwierdził archiwizację danego rekordu i tak dalej.

Prawdziwym wołem roboczym takich rozwiązań jest jedna z kwerend funkcjonalnych — kwerenda dołączająca (ang. *append query*). Jak sama nazwa wskazuje, kwerenda dołączająca dodaje rekordy do istniejącej tabeli. Bardzo często dołączane rekordy są pobierane z innej tabeli, ale równie dobrze dołączane rekordy mogą być generowane przez jakiś proces, wartości poszczególnych pól mogą być wyliczane bądź nawet mogą być pobierane z tej samej tabeli.

Dołączając do tabeli rekordy pobierane z tej samej tabeli, należy uważać, aby nie doszło do dublowania wartości unikalnego klucza tabeli.

Na rysunku 3.8 przedstawiono tabelę *Transakcje* przechowującą informacje o transakcjach dokonywanych przez poszczególnych klientów. Rekordy opisują transakcje przeprowadzane w różnych dniach, opiewające na różne kwoty i posiadające różne statusy.

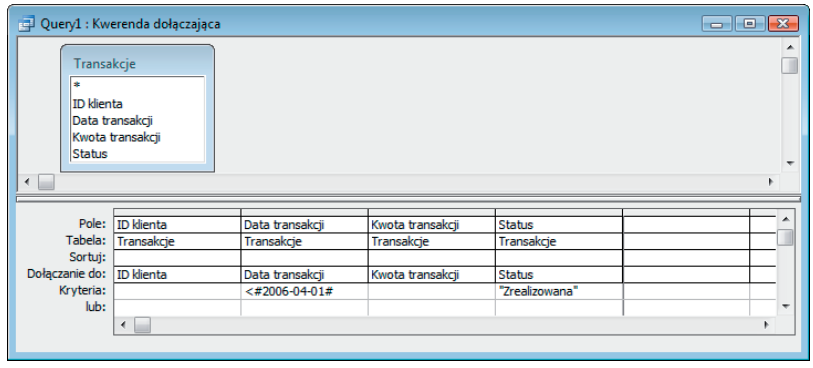
Transakcje : Tabela

ID klienta	Data transakcji	Kwota transakcji	Status
100	2006-06-18	224	Anulowana
100	2006-06-27	245	Zrealizowana
101	2006-04-03	227	Zrealizowana
101	2006-05-11	383	Anulowana
101	2006-05-11	71	Anulowana
101	2006-06-18	225	Anulowana
101	2006-03-31	249	Zrealizowana
101	2006-03-23	334	Zrealizowana
102	2006-05-30	375	Zrealizowana
102	2006-04-04	229	Zrealizowana
102	2006-04-04	230	Zrealizowana
102	2006-06-15	69	Zrealizowana
103	2006-05-13	76	Częściowa
103	2006-06-23	72	Zrealizowana
104	2006-06-27	390	Zrealizowana
104	2006-04-23	226	Wstrzymana
104	2006-06-27	391	Zrealizowana
104	2006-04-05	235	Zrealizowana
104	2006-04-05	234	Zrealizowana
104	2006-03-31	80	Zrealizowana
104	2006-05-17	232	Zrealizowana
105	2006-04-24	227	Wstrzymana

Rekord: 1 z 1000

Rysunek 3.8. Tabela przechowująca informacje o transakcjach

Jednym z zadań, które są często wykonywane w takich sytuacjach, jest wyczyszczenie tabeli ze starych rekordów opisujących zakończone transakcje. Załóżmy, że mamy do dyspozycji drugą tabelę, *ArchiwumTransakcji*, która przechowuje takie rekordy. W prosty sposób możemy teraz utworzyć projekt kwerendy, która odfiltruje zakończone transakcje i umożliwi przeniesienie ich do archiwum. Na rysunku 3.9 przedstawiono wygląd projektu kwerendy dołączającej, wyszukującej rekordy mające datę transakcji wcześniejszą niż `2006-04-01` i status `Zrealizowana`. Uruchomienie tej kwerendy spowoduje umieszczenie takich rekordów w tabeli *ArchiwumTransakcji* (na rysunku 3.9 nie widać tego wprost, ale tabela *ArchiwumTransakcji* została wybrana z listy *Nazwa tabeli* podczas tworzenia kwerendy dołączającej w oknie dialogowym *Dołączanie*).



Rysunek 3.9. Projekt kwerendy dołączającej

Jeżeli struktura tabeli *ArchiwumTransakcji* jest identyczna jak tabeli *Transakcje*, to w wierszu *Dołączanie do* nazwy odpowiednich pól pojawią się automatycznie. Jeżeli dołączamy rekordy do tabeli o innej strukturze, to musimy ręcznie wybrać odpowiednie pola źródłowe i dopasować do nich pola docelowe.

Kod SQL kwerendy dołączającej przedstawionej na rysunku 3.9 wygląda następująco:

```
INSERT INTO ArchiwumTransakcji
( [ID klienta], [Data transakcji], [Kwota transakcji], Status )
SELECT Transakcje.[ID klienta],
Transakcje.[Data transakcji],
Transakcje.[Kwota transakcji],
Transakcje.Status
FROM Transakcje
WHERE (((Transakcje.[Data transakcji])<#2006-04-01#)
AND ((Transakcje.Status)="Zrealizowana"));
```

Kod SQL kwerendy dołączającej rozpoczyna się od słów kluczowych `INSERT INTO`, po których następuje nazwa tabeli docelowej oraz umieszczona w nawiasach lista pól tabeli docelowej. Następnie w składni kwerendy pojawia się wyrażenie `SELECT`, które pobiera odpowiednie rekordy z tabeli źródłowej (w naszym przypadku jest to tabela *Transakcje*), zgodnie ze zdefiniowanymi kryteriami wyszukiwania. Warto zwrócić uwagę na fakt, że ilość pól wybranych z tabeli źródłowej musi odpowiadać ilość pól w tabeli docelowej. Kwerenda dołączająca nie musi operować na wszystkich polach tabeli źródłowej.

Po uruchomieniu naszej kwerendy rekordy są dodawane do tabeli *ArchiwumTransakcji*. Ale to dopiero połowa całej historii — wybrane rekordy zostały skopiowane do tabeli archiwalnej, ale ich oryginały nadal istnieją w tabeli źródłowej. Aby je usunąć, musimy użyć odpowiedniej kwerendy usuwającej. Kwerendy usuwające zostaną szczegółowo omówione w podrozdziale 3.3.

Omówienie

Powyższy przykład dobrze ilustruje typowy sposób działania kwerend dołączających: podkwerenda wybiera z tabeli źródłowej odpowiednie rekordy, wyszukiwane w zależności od zdefiniowanych (bądź nie) kryteriów wyszukiwania i następnie wyszukane rekordy są dołączane do tabeli docelowej.

Teraz przyjrzymy się innej metodzie wstawiania rekordów do tabeli, w której zamiast pól z tabeli źródłowej do pól tabeli docelowej będziemy bezpośrednio wstawiali odpowiednie wartości. W przedstawionym poniżej kodzie SQL kwerendy dołączającej rolę tabeli docelowej nadal spełnia tabela *ArchiwumTransakcji*, ale tym razem do tabeli docelowej wstawiamy zakodowane „na sztywno" wartości zdefiniowane w klauzuli `VALUES`. Warto zauważyć, że wstawiane wartości muszą być umieszczone w nawiasach.

```
INSERT INTO ArchiwumTransakcji
VALUES (2000, #2006-04-10#, 35.25, "Zrealizowana");
```

Tworząc takie kwerendy dołączające, musimy pamiętać o następujących kluczowych elementach:

- `INSERT INTO <nazwa tabeli>` to prawidłowy sposób rozpoczynania wyrażenia SQL.
- Za pomocą klauzuli `VALUES` można „na sztywno" zakodować wartości wstawiane do tabeli. Poszczególne wartości muszą być odpowiednio dopasowane do typu poszczególnych pól w tabeli, do których są wstawiane. Wartości numeryczne nie potrzebują żadnych kwalifikatorów i mogą być wstawiane bezpośrednio w kodzie SQL kwerendy jako liczby całkowite, rzeczywiste itd. (na przykład `2000` czy `35.25`). Daty muszą być ujęte w znaki krzyżyka # (ang. *hash*), a tekst (ciągi alfanumeryczne) musi być ujęty w znaki apostrofu lub cudzysłowu (oba warianty są dopuszczalne).

Cztery wartości użyte w kodzie SQL poprzedniego przykładu odpowiadają pod względem typu i kolejności czterem polom w tabeli docelowej, stąd nie istnieje tutaj konieczność definiowania listy pól. Warto jednak powiedzieć, że zdefiniowanie w takim przypadku listy pól nie jest żadnym błędem, a co więcej, może być nawet rozwiązaniem preferowanym ze względu na większą przejrzystość kodu. W takim przypadku kod SQL kwerendy dołączającej wyglądałby następująco:

```
INSERT INTO ArchiwumTransakcji
([ID klienta], [Data transakcji], [Kwota transakcji], Status)
VALUES (2000, #2006-04-10#, 35.25, "Zrealizowana");
```

Wyniki działania obu przedstawionych kwerend są identyczne.

Zdefiniowanie nazw pól tabeli docelowej jest wymagane w sytuacji, kiedy wstawiane wartości nie są ułożone w odpowiednim porządku bądź jeżeli niektóre pola tabeli zostają pominięte. Przykładowo: możemy wstawić do tabeli nowy rekord, w którym ustawimy tylko wartości pól *ID klienta* oraz *Kwota transakcji*; wartości innych pól mogą nie być jeszcze znane i dlatego zostały pominięte. Kod SQL kwerendy dołączającej może wyglądać w takiej sytuacji następująco:

```
INSERT INTO ArchiwumTransakcji ([ID klienta], [Kwota transakcji])
VALUES (2000, 35.25);
```

W tym przypadku ustawiamy jedynie wartości dwóch pól. Taki sposób postępowania jest najzupełniej prawidłowy, a wykonanie kwerendy zakończy się powodzeniem, pod warunkiem że pola, których wartości nie ustawiamy, mogą przyjmować wartości puste.

Dołączanie rekordów ze zbioru rekordów

Polecenie `INSERT INTO` jest często wykorzystywane w procedurach przetwarzających dane VBA/ADO, gdzie tabela docelowa jest wypełniana w miarę przechodzenia pętli przez kolejne rekordy zbioru rekordów (ang. *recordset*), przykładowo:

```
Sub append_routine()
  Dim conn As ADODB.Connection
  Set conn = CurrentProject.Connection
  Dim rs_transactions As New ADODB.Recordset
```

```
  Dim ssql As String
  ' Pobieramy wszystkie rekordy z tabeli Transakcje
  ssql = "Select * From Transakcje"
  rs_transactions.Open ssql, conn, adOpenKeyset, adLockOptimistic
  Do Until rs_transactions.EOF
    ' Jeżeli data transakcji to 1 kwietnia,
    ' wstawiamy rekord do archiwum i ustawiamy kwotę transakcji na 0
    If rs_transactions.Fields("Data transakcji") = #2006-04-01# Then
      ssql = "Insert Into ArchiwumTransakcji Values ("
      ssql = ssql & rs_transactions.Fields("ID klienta") & ", "
      ssql = ssql & "#" & rs_transactions.Fields("Data transakcji") & "#, "
      ssql = ssql & 0 & ", "
      ssql = ssql & "'April''s Fools Day - wszystko za darmo!')"
      conn.Execute ssql
    End If
  rs_transactions.MoveNext
  Loop
  ' usuwamy wszystkie rekordy z 2006-04-01 z tabeli Transakcje
  ssql = "Delete * From Transakcje Where "
  ssql = ssql & " Transakcje.[Data transakcji]=#2006-04-01#"
  conn.Execute ssql
  rs_transactions.Close
  Set rs_transactions = Nothing
  conn.Close
  MsgBox "Gotowe!"
End Sub
```

W tym przykładzie zbiór rekordów (`rs_transactions`) zawiera wszystkie rekordy z tabeli *Transakcje*. Podczas przetwarzania w pętli kolejnych rekordów ze zbioru procedura sprawdza, czy data transakcji to `2006-04-01`. Jeżeli tak, tworzone jest odpowiednie polecenie `INSERT INTO` języka SQL. Przykładowo:

```
INSERT INTO ArchiwumTransakcji Values
VALUES (106, #2006-04-01#, 0, 'April''s Fools Day - wszystko za darmo!');
```

Uważni Czytelnicy z pewnością zwrócili uwagę na użycie podwójnego znaku apostrofu w ciągu znaków `April''s`. Taki zapis zapobiega wystąpieniu błędu, który mógłby się w przeciwnym razie pojawić podczas próby wstawienia ciągu znaków zawierającego apostrof.

Procedura przedstawiona powyżej kopiuje wszystkie rekordy z 1 kwietnia 2006 do tabeli archiwalnej i jako kwotę transakcji ustawia wartość `0`. Po przejściu pętli przez wszystkie rekordy zbioru wykonywana jest kwerenda usuwająca wszystkie rekordy z 1 kwietnia 2006 z tabeli Transakcje. Poniżej przedstawiamy fragment kodu odpowiedzialny za usuwanie rekordów:

```
' usuwamy wszystkie rekordy z 2006-04-01 z tabeli Transakcje
ssql = "Delete * From Transakcje Where "
ssql = ssql & " Transakcje.[Data transakcji]=#2006-04-01#"
conn.Execute ssql
```

Użycie odpowiedniej procedury VBA do przechodzenia przez kolejne rekordy i podejmowanie odpowiednich decyzji o dołączaniu poszczególnych rekordów do innej tabeli jest świetnym rozwiązaniem zwłaszcza w sytuacji, kiedy warunki określające sposób przetwarzania stają się złożone. Bo jak inaczej znaleźć na przykład rekordy, w których musimy zredukować kwotę transakcji do `0`, jeżeli możemy to zrobić tylko dla ściśle wybranych klientów, dla transakcji wykonanych tylko w kilku określonych dniach i tylko wtedy, gdy całkowite saldo transakcji takiego klienta jest mniejsze niż `100`, a ostatnie zlecenie zostało złożone nie wcześniej niż 30 dni temu?

Zdefiniowanie takich warunków w siatce projektu kwerendy może być niezłym wyzwaniem, stąd znajomość sposobu połączenia w procedurze VBA poleceń języka SQL z instrukcjami warunkowymi może być bezcenną pozycją w zestawie umiejętności każdego użytkownika bazy danych Microsoft Access.

3.3. Usuwanie danych

Opis problemu

Musimy usunąć z tabeli określone dane. Rekordy, które mają być usunięte, muszą spełniać określone kryteria wyszukiwania, a pozostałe rekordy muszą pozostać nienaruszone. W jaki sposób można bezpiecznie wykonać taką operację?

Rozwiązanie

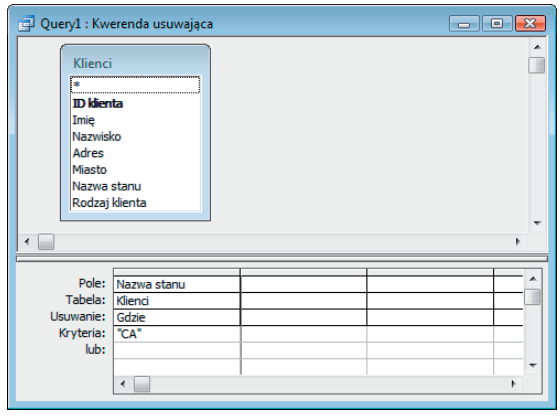
Aby usunąć z tabeli rekordy spełniające określone kryteria wyszukiwania, należy użyć *kwerendy usuwającej* (ang. *delete query*). Jeżeli w takiej kwerendzie nie zastosujemy żadnych kryteriów wyszukiwania, to używając jej, musimy zachować daleko idącą ostrożność — kwerenda usuwająca uruchomiona bez kryteriów wyszukiwania może całkowicie wyczyścić zawartość tabeli.

Kwerendy usuwające powodują usunięcie danych, ale pozostawiają tabele. Tabele nie są usuwane.

Aby usunąć wybrane dane z tabeli, musimy utworzyć odpowiednią kwerendę usuwającą, która wybierze tylko rekordy przeznaczone do skasowania. Na rysunku 3.10 przedstawiono projekt kwerendy, która usuwa rekordy klientów pochodzących ze stanu CA (California). Oznacza to, że usunięte zostaną tylko takie rekordy, dla których pole *Nazwa stanu* ma wartość `CA`; inne rekordy tabeli pozostaną nienaruszone. Zawsze musimy pamiętać, że mimo iż w siatce projektu kwerendy na rysunku 3.10 umieszczone zostało tylko jedno pole, uruchomienie kwerendy nie usuwa danych tylko z tego pola — zamiast tego w całości zostają usunięte wszystkie rekordy spełniające podane kryterium wyszukiwania. W siatce projektu kwerendy nie musimy umieszczać wszystkich pól rekordu; wystarczy umieścić tam pola, dla których definiujemy kryteria wyszukiwania. Kiedy używamy kwerendy usuwającej wszystkie rekordy z tabeli (czyli kwerendy bez kryteriów wyszukiwania), wystarczy z okna tabeli przeciągnąć gwiazdkę na siatkę projektu kwerendy — gwiazdka oznacza po prostu wszystkie pola tabeli.

Aby utworzyć kwerendę usuwającą, należy po przejściu na siatkę projektu kwerendy wybrać z menu głównego polecenie *Kwerendy/Kwerenda usuwająca*. W wersji Access 2007 wystarczy w tym celu skorzystać z odpowiedniego przycisku na *Wstążce*. Kod SQL kwerendy przedstawionej na rysunku 3.10 wygląda następująco:

```
DELETE [Klienci].[Nazwa stanu]
FROM Klienci
WHERE ((([Klienci].[Nazwa stanu])="CA"));
```



Rysunek 3.10. Kwerenda usuwająca ze zdefiniowanymi kryteriami wyszukiwania

Jak widać, kod SQL kwerendy jest relatywnie prosty. Jego składnia jest nieco zbliżona do składni kwerendy wybierającej `SELECT`, z wyjątkiem tego, że kod kwerendy usuwającej rozpoczyna się od słowa kluczowego `DELETE`. Interesujący jest fakt, że „przesłanie" czy też „wiadomość" wynikające ze składni powyższego kodu SQL mogą być nieco mylące. Jak już wspominaliśmy wcześniej, wykonanie takiej kwerendy usuwa całe rekordy, a nie tylko wartości pola *Nazwa stanu*. Zdecydowanie lepszym „składniowo" zapisem takiej kwerendy będzie następujący kod SQL:

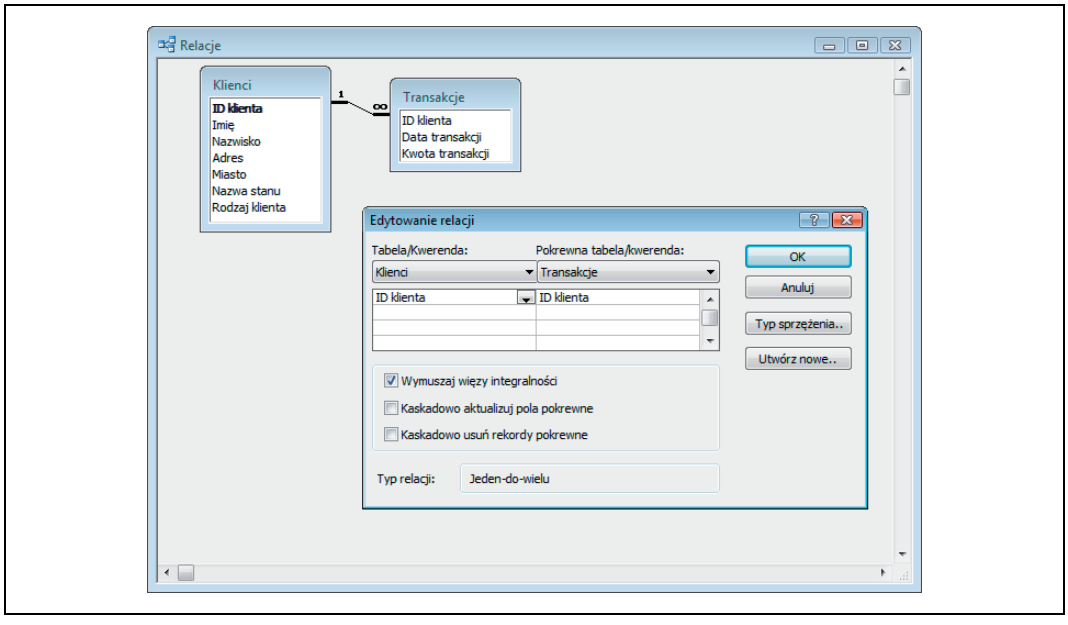
```
DELETE [Klienci].*
FROM Klienci
WHERE ((([Klienci].[Nazwa stanu])="CA"));
```

Różnica polega na tym, że zamiast nazwy pojedynczego pola użyliśmy gwiazdki, która symbolizuje wszystkie pola tabeli. Jak widać, czasami sposób, w jaki Access tworzy kod SQL kwerend, nie jest najlepszą reprezentacją zamierzonej akcji. Oczywiście kod generowany przez Accessa jest poprawny i działa, ale może być nieco mylący.

Omówienie

Kiedy usuwamy rekordy z tabeli, która jest powiązana relacjami z innymi tabelami, musimy wziąć pod uwagę kilka dodatkowych elementów. Ponieważ tabela nadrzędna jest połączona z tabelą podrzędną relacją jeden do wielu, usunięcie rekordów z tabeli nadrzędnej spowodowałoby naruszenie więzów integralności i pozostawienie „osieroconych" rekordów w tabeli podrzędnej.

Access posiada mechanizm pozwalający na sprawne rozwiązanie takiego dylematu, co nie zmienia faktu, że zrozumienie istoty problemu jest niezmiernie ważne. Aby zilustrować całe zagadnienie, posłużymy się przykładem. Na rysunku 3.11 przedstawiono relację ustanowioną pomiędzy tabelami *Klienci* i *Transakcje*. Zwróćmy uwagę, że w oknie dialogowym *Edytowanie relacji* zaznaczona została opcja *Wymuszaj więzy integralności* (aby wyświetlić to okno dialogowe, należy dwukrotnie kliknąć lewym przyciskiem myszy linię łączącą obie tabele bądź z menu głównego wybrać polecenie *Relacje/Edytuj relację*). Taka relacja oznacza, że rekordy w tabeli *Transakcje* muszą być dopasowane do odpowiednich rekordów z tabeli *Klienci*, a dokładniej, że każdy rekord w tabeli *Transakcje* musi posiadać w polu *ID klienta* wartość, która odpowiada wartości pola *ID klienta* jakiegoś rekordu w tabeli *Klienci*.



Rysunek 3.11. Przeglądanie relacji pomiędzy dwiema tabelami

Rekordy w tabeli *Klienci* muszą posiadać unikatowe wartości w polu *ID klienta*, stąd ilość rekordów w tabeli *Klienci* jest taka sama, jak ilość unikatowych identyfikatorów klientów (ilość unikatowych wartości pola *ID klienta*). W taki właśnie sposób tabela *Klienci* spełnia rolę tabeli nadrzędnej w relacji jeden do wielu.

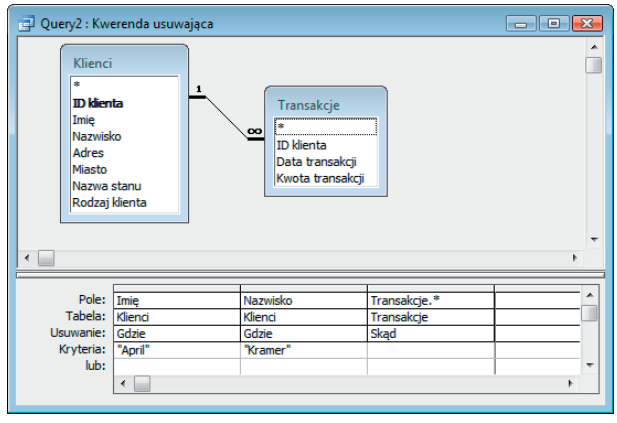
Pole *ID klienta* w tabeli *Transakcje* nie musi posiadać unikatowych wartości. W praktyce sytuacja wygląda tak, że niemal każdy rekord z tabeli *Klienci* będzie posiadał wiele odpowiadających mu rekordów w tabeli podrzędnej *Transakcje* — dobrze prowadzony biznes powoduje, że stali, lojalni klienci ciągle wracają i składają nowe zlecenia.

I znów, jedynym wymaganiem dla rekordów z tabeli *Transakcje* jest to, że wartość pola *ID klienta* musi odpowiadać wartości tego pola w jednym z rekordów tabeli *Klienci*.

Teraz załóżmy, że chcemy usunąć danego klienta z tabeli *Klienci*. Ponieważ pomiędzy tabelami istnieją więzy integralności, ale kaskadowe usuwanie rekordów pokrewnych nie jest dozwolone (opcja *Kaskadowo usuń rekordy pokrewne* jest wyłączona, jak to zostało zilustrowane na rysunku 3.11), to jeżeli dany klient będzie posiadał powiązane rekordy w tabeli podrzędnej, Access nie pozwoli na proste usunięcie klienta. Więzy integralności pomiędzy tabelami nie pozwolą na utworzenie „osieroconych" rekordów. Klienci nie muszą mieć żadnych rekordów opisujących transakcje, więc usunięcie klientów bez transakcji jest możliwe, ale jeżeli dla danego klienta istnieją w tabeli podrzędnej jakiekolwiek rekordy opisujące jego transakcje, to usunięcie takiego klienta nie będzie możliwe.

Jeżeli klient posada jakieś powiązane z nim transakcje, to przed usunięciem rekordu klienta musimy usunąć wszystkie rekordy opisujące jego transakcje. Usuwanie rekordów transakcji nie podlega żadnym ograniczeniom i w żaden sposób nie możemy utworzyć „osieroconego" rekordu klienta — „osierocone" rekordy mogą się teoretycznie pojawić jedynie w tabeli podrzędnej.

A zatem w jaki sposób usunąć wszystkie transakcje danego klienta? Kwerenda usuwająca przedstawiona na rysunku 3.12 usuwa wszystkie rekordy transakcji dla klienta `April Kramer`. W rekordzie opisującym tego klienta znajduje się odpowiednie pole *ID klienta*, które jest wykorzystywane przez kwerendę do identyfikacji usuwanych rekordów. Zwróćmy uwagę na fakt, że w wierszu *Usuwanie* w pierwszych dwóch kolumnach umieszczono klauzule `Where`, spełniające rolę kryteriów wyszukiwania. Trzecia kolumna identyfikuje tabelę, z której będą usuwane odnalezione rekordy (*Transakcje*); w wierszu *Usuwanie* tej kolumny umieszczono klauzulę *Skąd*.



Rysunek 3.12. Usuwanie rekordów z jednej tabeli w oparciu o kryteria z innej tabeli

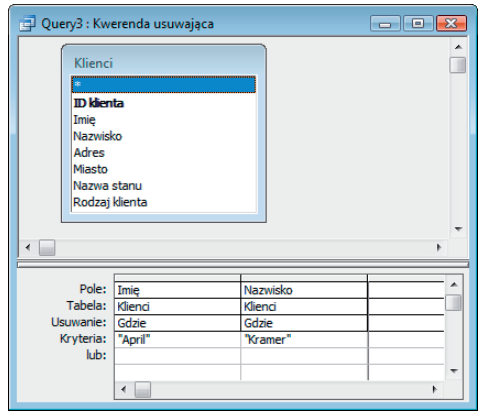
Kod SQL kwerendy przedstawionej na rysunku 3.12 wygląda następująco:

```
DELETE Klienci.Imię, Klienci.Nazwisko, Transakcje.*
FROM Klienci INNER JOIN Transakcje ON
Klienci.[ID klienta] = Transakcje.[ID klienta]
WHERE (((Klienci.Imię)="April") AND
((Klienci.Nazwisko)="Kramer"));
```

Jest to nieco mylące, ponieważ po słowie kluczowym `DELETE` występują nazwy pól z tabeli *Klienci* (*Imię* i *Nazwisko*) — można stąd wysnuć mylny wniosek, że usuwane będą rekordy z tabeli *Klienci*. Oczywiście nie jest to prawdą — usuwane są tylko rekordy transakcji z tabeli podrzędnej, podczas gdy tabela klientów pozostaje nienaruszona. Rekord opisujący klienta zostaje usunięty przez kolejną kwerendę, przedstawioną na rysunku 3.13.

Usunięcie w ten sposób rekordu klienta `April Kramer` może być niebezpieczne, ponieważ teoretycznie może istnieć więcej niż jeden klient o takim imieniu i nazwisku. Imienia i nazwiska klienta użyto w tej kwerendzie tylko na potrzeby lepszego zilustrowania zasady usuwania takich rekordów. W praktyce jedynym sposobem gwarantującym, że usunięty zostanie właściwy rekord klienta, jest posłużenie się polem *ID klienta*.

Jak widać, jeżeli kaskadowe usuwanie rekordów pokrewnych nie jest dozwolone (na przykład kiedy w oknie *Edytowanie relacji* opcja *Kaskadowo usuń rekordy pokrewne* została wyłączona), przed usunięciem rekordu klienta musimy usunąć wszystkie odpowiadające mu rekordy z tabeli podrzędnej. Jeżeli jednak ta opcja została włączona, to usunięcie wybranych rekordów z tabeli nadrzędnej spowoduje automatyczne usunięcie wszystkich pokrewnych rekordów z tabeli podrzędnej. W takiej sytuacji usunięcie `April Kramer` z tabeli *Klienci* spowodowałoby automatycznie usunięcie wszystkich związanych z nią rekordów z tabeli *Transakcje*.



Rysunek 3.13. Kwerenda usuwająca z tabeli nadrzędnej rekord klienta

Takie rozwiązanie potrafi zaoszczędzić masę czasu, ale nie ma nic za darmo. Kaskadowe usuwanie rekordów pokrewnych może w prosty sposób spowodować niezamierzone usunięcie z tabel cennych informacji. Jeżeli chcemy skorzystać z tej opcji, powinniśmy upewnić się, że kopie zapasowe danych są tworzone odpowiednio często, w regularnych odstępach czasu. Usunięcie danych jest nieodwołalne i jeżeli nie posiadamy odpowiedniej kopii zapasowej, anulowanie takiej operacji i przywrócenie poprzednich danych nie będzie możliwe. Jeżeli nie posiadamy odpowiedniego mechanizmu tworzenia kopii zapasowych, ryzyko związane z użyciem możliwości kaskadowego usuwania rekordów może przeważyć nad wszystkimi zaletami płynącymi z zastosowania tego mechanizmu. Zanim zdecydujemy się na jego zastosowanie, musimy starannie rozważyć wszystkie argumenty za i przeciw.

3.4. Kwerendy tworzące tabele

Opis problemu

W jaki sposób można utworzyć tabelę przechowującą rekordy będące rezultatem działania kwerendy?

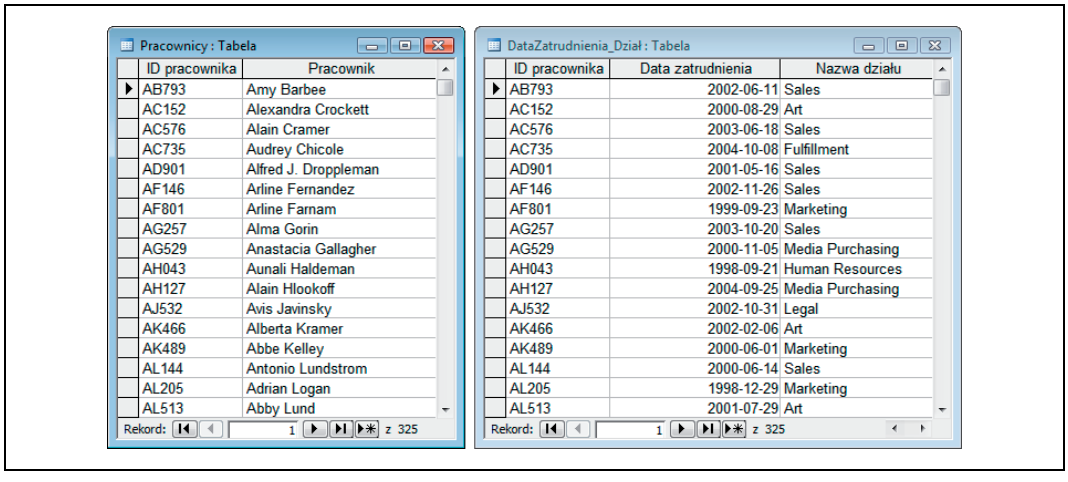
Rozwiązanie

W pewnych sytuacjach bardzo użyteczna może być możliwość umieszczenia rekordów zwracanych przez kwerendę bezpośrednio w nowej tabeli. Aby tego dokonać, musimy skorzystać z *kwerendy tworzącej tabelę* (ang. *make-table query*).

W zasadzie możemy sobie teraz zadać pytanie, po co mamy zadawać sobie trud tworzenia nowej tabeli, skoro tabele przechowujące takie dane już istnieją? Oto kilka powodów:

- Aby połączyć w jednej tabeli powiązane ze sobą dane nieposiadające struktury hierarchicznej.
- Aby podzielić dane z jednej wielkiej tabeli na kilka mniejszych tabel. Takiego podziału dokonujemy zazwyczaj w oparciu o wartości jednego lub kilku pól kluczowych tabeli źródłowej.

Rysunek 3.14 dobrze ilustruje pierwszą sytuację. Mamy tutaj dwie tabele, które najwyraźniej są ze sobą powiązane — mają wspólne pole *ID pracownika*, aczkolwiek nie istnieje tutaj relacja jeden do wielu. W każdej z tabel jednemu pracownikowi odpowiada tylko jeden rekord. Utrzymywanie nazwisk pracowników w jednej tabeli, a informacji o dacie zatrudnienia i dziale w innej tabeli być może ma jakieś uzasadnienie biznesowe, ale nie ma żadnego sensu z punktu widzenia projektowania bazy danych. Połączenie danych z tych dwóch tabel w jedną wydaje się być jak najbardziej sensownym posunięciem. Tabela będąca rezultatem takiej operacji będzie miała jedno pole *ID pracownika* oraz trzy dodatkowe pola opisujące dane pracownika.



Pracownicy : Tabela

ID pracownika	Pracownik
AB793	Amy Barbee
AC152	Alexandra Crockett
AC576	Alain Cramer
AC735	Audrey Chicole
AD901	Alfred J. Droppleman
AF146	Arline Fernandez
AF801	Arline Farnam
AG257	Alma Gorin
AG529	Anastacia Gallagher
AH043	Aunali Haldeman
AH127	Alain Hlookoff
AJ532	Avis Javinsky
AK466	Alberta Kramer
AK489	Abbe Kelley
AL144	Antonio Lundstrom
AL205	Adrian Logan
AL513	Abby Lund

Rekord: 1 z 325

DataZatrudnienia_Dział : Tabela

ID pracownika	Data zatrudnienia	Nazwa działu
AB793	2002-06-11	Sales
AC152	2000-08-29	Art
AC576	2003-06-18	Sales
AC735	2004-10-08	Fulfillment
AD901	2001-05-16	Sales
AF146	2002-11-26	Sales
AF801	1999-09-23	Marketing
AG257	2003-10-20	Sales
AG529	2000-11-05	Media Purchasing
AH043	1998-09-21	Human Resources
AH127	2004-09-25	Media Purchasing
AJ532	2002-10-31	Legal
AK466	2002-02-06	Art
AK489	2000-06-01	Marketing
AL144	2000-06-14	Sales
AL205	1998-12-29	Marketing
AL513	2001-07-29	Art

Rekord: 1 z 325

Rysunek 3.14. Dwie tabele połączone relacją jeden do jednego

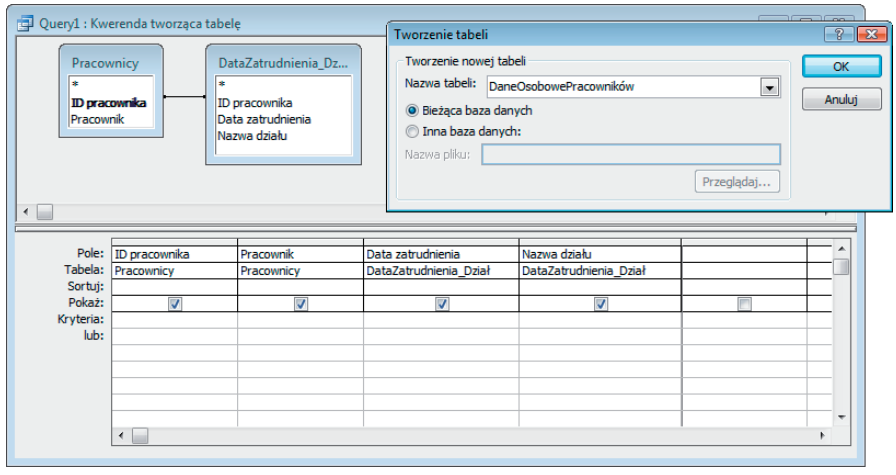
Aby utworzyć taką tabelę, musimy umieścić dwie istniejące tabele w widoku projektu kwerendy, a w siatce projektu umieścić wszystkie pola tych tabel (ale tylko jedno pole *ID pracownika*). Następnie musimy poinformować Accessa, że tworzymy kwerendę tworzącą tabele. Aby tego dokonać, wybieramy z menu głównego polecenie *Kwerendy/Kwerenda tworząca tabele* (w wersji Access 2007 używamy odpowiedniego przycisku na *Wstążce*). Kiedy wybierzemy z menu kwerendę tworzącą tabele, na ekranie pojawi się okno dialogowe *Tworzenie tabeli*, w którym możemy wpisać nazwę nowej tabeli lub wybrać z listy rozwijanej nazwę istniejącej tabeli.

Na rysunku 3.15 przedstawiono projekt kwerendy, która łączy rekordy z dwóch tabel i zapisuje je w jednej, nowej tabeli o nazwie *DaneOsobowePracowników*.

Kod SQL kwerendy przedstawionej na rysunku 3.15 wygląda następująco:

```
SELECT Pracownicy.[ID pracownika], Pracownicy.Pracownik,
DataZatrudnienia_Dział.[Data zatrudnienia],
DataZatrudnienia_Dział.[Nazwa działu] INTO DaneOsobowePracowników
FROM Pracownicy INNER JOIN DataZatrudnienia_Dział ON
Pracownicy.[ID pracownika] = DataZatrudnienia_Dział.[ID pracownika];
```

Zwróćmy uwagę, że kluczowym elementem takiej kwerendy jest polecenie `SELECT <lista pól> INTO <nazwa nowej tabeli>`, po którym następują: klauzula `FROM` oraz ewentualne sprzężenia i kryteria wyszukiwania.



Rysunek 3.15. Projekt kwerendy tworzącej tabele

Na rysunku 3.16 przedstawiono wyniki działania kwerendy tworzącej tabele. Wszystkie informacje o pracowniku zostały teraz umieszczone w jednej tabeli. Żadne relacje pomiędzy danymi pracowników nie zostały utracone, dla każdego pracownika mamy podany jego identyfikator, nazwisko, datę zatrudnienia oraz nazwę działu, w którym pracuje.

DaneOsobowePracowników : Tabela

ID pracownika	Pracownik	Data zatrudnienia	Nazwa działu
AB793	Amy Barbee	2002-06-11	Sales
AC152	Alexandra Croc	2000-08-29	Art
AC576	Alain Cramer	2003-06-18	Sales
AC735	Audrey Chicole	2004-10-08	Fulfillment
AD901	Alfred J. Droppl	2001-05-16	Sales
AF146	Arline Fernande	2002-11-26	Sales
AF801	Arline Farnam	1999-09-23	Marketing
AG257	Alma Gorin	2003-10-20	Sales
AG529	Anastacia Galla	2000-11-05	Media Purchasing
AH043	Aunali Haldema	1998-09-21	Human Resources
AH127	Alain Hlookoff	2004-09-25	Media Purchasing
AJ532	Avis Javinsky	2002-10-31	Legal
AK466	Alberta Kramer	2002-02-06	Art
AK489	Abbe Kelley	2000-06-01	Marketing
AL144	Antonio Lundst	2000-06-14	Sales
AL205	Adrian Logan	1998-12-29	Marketing
AL513	Abby Lund	2001-07-29	Art

Rekord: 1 z 325

Rysunek 3.16. Tabela będąca wynikiem działania kwerendy łączącej tabele

Jak już wspomniano wcześniej, innym zastosowaniem kwerend tworzących tabele jest dzielenie wielkich tabel na mniejsze. Przykładem takiego zadania będzie podzielenie tabeli przedstawionej na rysunku 3.16 na szereg tabel przechowujących informacje o pracownikach poszczególnych działów. Przykładowo: aby utworzyć nową tabelę przechowującą informacje o pracownikach działu sprzedaży, należałoby utworzyć następującą kwerendę tworzącą tabele:

```
SELECT DaneOsobowePracowników.[ID pracownika],
DaneOsobowePracowników.Pracownik,
DaneOsobowePracowników.[Data zatrudnienia],
DaneOsobowePracowników.[Nazwa działu]
```

```
INTO DziałSprzedaży
FROM DaneOsobowePracowników
WHERE (((DaneOsobowePracowników.[Nazwa działu])="Sales"));
```

Pamiętajmy, że utworzenie nowej tabeli zawierającej informacje o pracownikach działu sprzedaży nie modyfikuje w żaden sposób naszej tabeli źródłowej. Jeżeli chcemy takie dane usunąć, musimy skorzystać z osobnej kwerendy usuwającej rekordy.

Omówienie

Kwerendy tworzącej tabele możemy użyć do nadpisania istniejącej tabeli. Jeżeli po słowie kluczowym `INTO` umieścimy nazwę istniejącej tabeli, to jej struktura oraz dane zostaną całkowicie zastąpione strukturą i danymi będącymi rezultatem działania kwerendy tworzącej tabele. Nazwa nowej tabeli pozostaje taka sama jak starej tabeli, ale jej stan przed i po wykonaniu kwerendy już wcale taki sam być nie musi.

Innym sposobem wykorzystania kwerendy tworzącej tabele jest użycie w składni polecenia SQL pól tymczasowych, co spowoduje utworzenie nowej tabeli posiadającej jeden rekord. Nazwy pól i ich wartości są zapisane bezpośrednio w kodzie SQL kwerendy. A oto przykład:

```
SELECT 123 AS [Moja wartość], 'Adam' AS [Moje imię] INTO MojaNowaTabela;
```

Wykonanie tego prostego zapytania SQL powoduje utworzenie nowej tabeli o nazwie *MojaNowaTabela* zawierającej jeden rekord składający się z dwóch pól: pierwsze pole, o nazwie *Moja wartość*, ma wartość `123`, natomiast drugie pole, o nazwie *Moje imię*, ma wartość `Adam`. Zwróćmy uwagę na fakt, iż w kodzie kwerendy nie występuje ani słowo kluczowe `FROM`, ani sekcja klauzuli `Where` — dzieje się tak, ponieważ nasza kwerenda nie odwołuje się do żadnej tabeli źródłowej.