Część 2

Tworzenie stron i aplikacji internetowych oraz baz danych i administrowanie nimi

INF.03

Podręcznik do nauki zawodu

TECHNIK INFORMATYK
TECHNIK PROGRAMISTA

WSiP

Autorzy: **Agnieszka Klekot, Tomasz Klekot**

Typ szkoły: **technikum**
Zawód: **technik informatyk, technik programista**
Kwalifikacja: INF.03. **Tworzenie i administrowanie stronami i aplikacjami internetowymi oraz bazami danych**

© Copyright by Wydawnictwa Szkolne i Pedagogiczne
Warszawa 2020

Wydanie IV, 2024

ISBN 978-83-02-22261-0 – podręcznik w postaci elektronicznej
ISBN 978-83-02-19077-3 (część 2) – podręcznik w postaci papierowej
ISBN 978-83-02-19078-0 (całość) – podręcznik w postaci papierowej

Opracowanie merytoryczne i redakcyjne: **Tomasz Kompanowski** (redaktor koordynator),
Sebastian Jabłoński (redaktor merytoryczny)
Redakcja językowa: **Tomasz Karpowicz**
Redakcja techniczna: **Elżbieta Walczak**
Projekt okładki: **Dominik Krajewski**
Ilustracje: **Roman Kirilenko**
Skład i łamanie: **Ledor**

Wydawnictwa Szkolne i Pedagogiczne Spółka Akcyjna
00-807 Warszawa, Aleje Jerozolimskie 96
KRS 0000595068
Infolinia: 801 220 555
www.wsip.pl
Wyprodukowano w Polsce

I. Podstawy baz danych

1. Wstęp do baz danych 6
2. Relacyjne bazy danych 12

II. Zasady projektowania baz danych

3. Model baz danych 22
4. Zasady tworzenia diagramów ER 34

III. Systemy zarządzania bazami danych

5. Rodzaje systemów zarządzania bazami danych 40
6. Instalacja i konfiguracja MS Access 44
7. Zarządzanie bazą danych w pakiecie XAMPP 54
8. Microsoft SQL Server 62

IV. Tworzenie baz danych w MS Access

9. Tworzenie tabel w MS Access 78
10. Relacje – powiązania między tabelami 99
11. Wprowadzanie danych do bazy. Operacje na danych 103
12. Formularze 109
13. Kwerendy (zapytania) w MS Access 116
14. Raporty 125
15. Programowanie w MS Access 129

V. SQL – strukturalny język zapytań

16. Instrukcje języka SQL 138
17. Język definiowania struktur danych – DDL 141
18. Język do wybierania i manipulowania danymi – DML 146
19. Język definiowania zapytań – DQL 149
20. Funkcje agregujące 157
21. Grupowanie danych 162
22. Łączenie tabel 165
23. Więzy integralności 170
24. Łączenie zapytań 172
25. Użytkownicy i ich uprawnienia w bazie danych 178

VI. Algorytmika

26. Podstawy stosowania algorytmów 184

VII. Zasady programowania strukturalnego

27. Środowisko programistyczne 198
28. Struktura prostego programu w C++ 205
29. Typy zmiennych w C++ 208
30. Instrukcje we / wy oraz operatory i ich działanie 212
31. Instrukcje sterujące 220
32. Funkcje 238
33. Tablice 250
34. Tablice znakowe i stringi 262
35. Wskaźniki 270
36. Operacje na plikach 276

VIII. Skrypty po stronie klienta – JavaScript

37. Podstawy JavaScript 288
38. Wyświetlanie komunikatów na ekranie. Okna dialogowe 291
39. Zmienne i ich typy 294
40. Operatory 298
41. Instrukcje sterujące 305
42. Funkcje 318
43. Obiekty w JavaScript 323
44. Zdarzenia 336
45. Obsługa formularzy 341

IX. Skrypty po stronie serwera – PHP

46. Struktura języka PHP 358
47. Typy zmiennych i rodzaje operatorów 362
48. Instrukcje sterujące 377
49. Funkcje i ich zastosowanie 389
50. Obsługa formularzy 409
51. Obsługa bazy danych w PHP 418
52. Pliki cookies i sesje 431
53. Obsługa plików w PHP 439

X. Projektowanie i dokumentowanie oprogramowania

54. Projektowanie programów 446
55. Dokumentowanie tworzonych aplikacji 449

Załącznik 1 455
Załącznik 2 459
Załącznik 3 467

Wykaz podstawowych pojęć w językach: polskim, angielskim i niemieckim 470

I. Podstawy baz danych

- Podstawowe pojęcia dotyczące baz danych
- Modele baz danych
- Podstawy relacyjnych baz danych
- Elementy relacyjnej bazy danych

1 Wstęp do baz danych

ZAGADNIENIA

- Definicje bazy danych, SZBD oraz systemu baz danych
- Cechy systemów baz danych
- Modele baz danych
- Bazy danych w chmurze

Podstawowe pojęcia dotyczące baz danych

Gromadzenie i wykorzystywanie informacji to kluczowe zadania systemów informatycznych. Nowoczesne rozwiązania w systemach informatycznych opierają się na digitalizacji informacji, szybkim dostępie do danych i ich analizie oraz automatyzacji operacji. Standardem jest tworzenie dynamicznych witryn i aplikacji internetowych korzystających ze zbiorów danych, zlokalizowanych na serwerach lub w chmurze obliczeniowej. Tworzenie baz na potrzeby systemów informatycznych lokalnych i internetowych jest istotną umiejętnością przyszłego technika informatyka / technika programisty.

Aby zrozumieć rozwiązania dotyczące obsługi danych przez systemy informatyczne, należy zdefiniować kilka kluczowych pojęć.

Baza danych (ang. *database*) – zbiór danych uporządkowany zgodnie z określoną charakterystyką i w określonej strukturze.
Dane (ang. *data*) – cyfrowa forma informacji mająca interpretację binarną, odpowiednia do transmisji i przetwarzania.
System zarządzania bazą danych, **SZBD** (ang. *Database Management System*) – system programów i narzędzi (może występować jako jedna aplikacja) umożliwiających tworzenie, przechowywanie, przetwarzanie i zarządzanie bazami danych.
System bazy danych – system zarządzania bazą danych obsługujący określoną bazę danych.

Podstawowe operacje na danych w bazie obejmują:

- wprowadzanie,
- zapamiętywanie i przechowywanie,
- wyszukiwanie, analizowanie i prezentowanie,
- dodawanie i usuwanie,
- aktualizowanie.

Ze względu na fizyczne umiejscowienie systemu bazy danych rozróżnia się **lokalne systemy baz danych** (aplikacyjne lub desktopowe), **sieciowe systemy baz danych** (serwerowe) oraz **chmurowe systemy baz danych**.

Serwerowe bazy danych najczęściej implementuje się w architekturze klient–serwer. System zarządzania bazą danych jest zainstalowany jako usługa serwerowa. Na serwerze przechowuje się również samą bazę danych. Do serwera może być podłączonych wiele stacji roboczych, które korzystają z bazy danych, wykonując na nich podstawowe operacje w zależności od przyznanych uprawnień.

Do najpopularniejszych SZBD w architekturze klient–serwer zalicza się:

- Microsoft SQL Server,
- MySQL,
- Oracle,
- PostgreSQL.

Istnieją również systemy baz danych pracujące jako lokalne aplikacje. Nazywa się je lokalnymi lub desktopowymi systemami baz danych. Do najpopularniejszych SZBD tego typu należą MS Access i Kexi. Te systemy umożliwiają tworzenie nowych, samodzielnych aplikacji bazodanowych oraz współdzielenie danych w sieci lokalnej.

Lokalnie można również wykorzystać pakiety serwerowe pełniące funkcje serwerowych baz danych, takich jak XAMPP.

Cechy systemu baz danych

Każdy system baz danych powinien charakteryzować się cechami takimi jak:

- określona struktura systemu bazy danych;
- trwałość danych, czyli możliwość przechowywania danych w pamięci masowej przez długi czas;
- możliwość realizacji określonych operacji na danych; dotyczy to edycji, realizacji zapytań, edycji bądź usuwania danych;
- niezależność danych, czyli oddzielenie struktury przechowywania danych od systemów zarządzania danymi oraz platformy sprzętowej;
- bezpieczeństwo danych, czyli zapewnienie mechanizmów kontroli dostępu do danych;
- integralność danych przez zapewnienie zgodności danych z rzeczywistością;
- spójność danych określana przez poprawność danych oraz odporność na anomalie;
- skalowalność, czyli możliwość rozbudowy.

Modele baz danych

Można je określić jako logiczny sposób organizacji danych złożony ze zbioru reguł opisujących dane, wzajemne zależności między nimi, dozwolone operacje na danych oraz zasady dostępu do systemu baz danych.

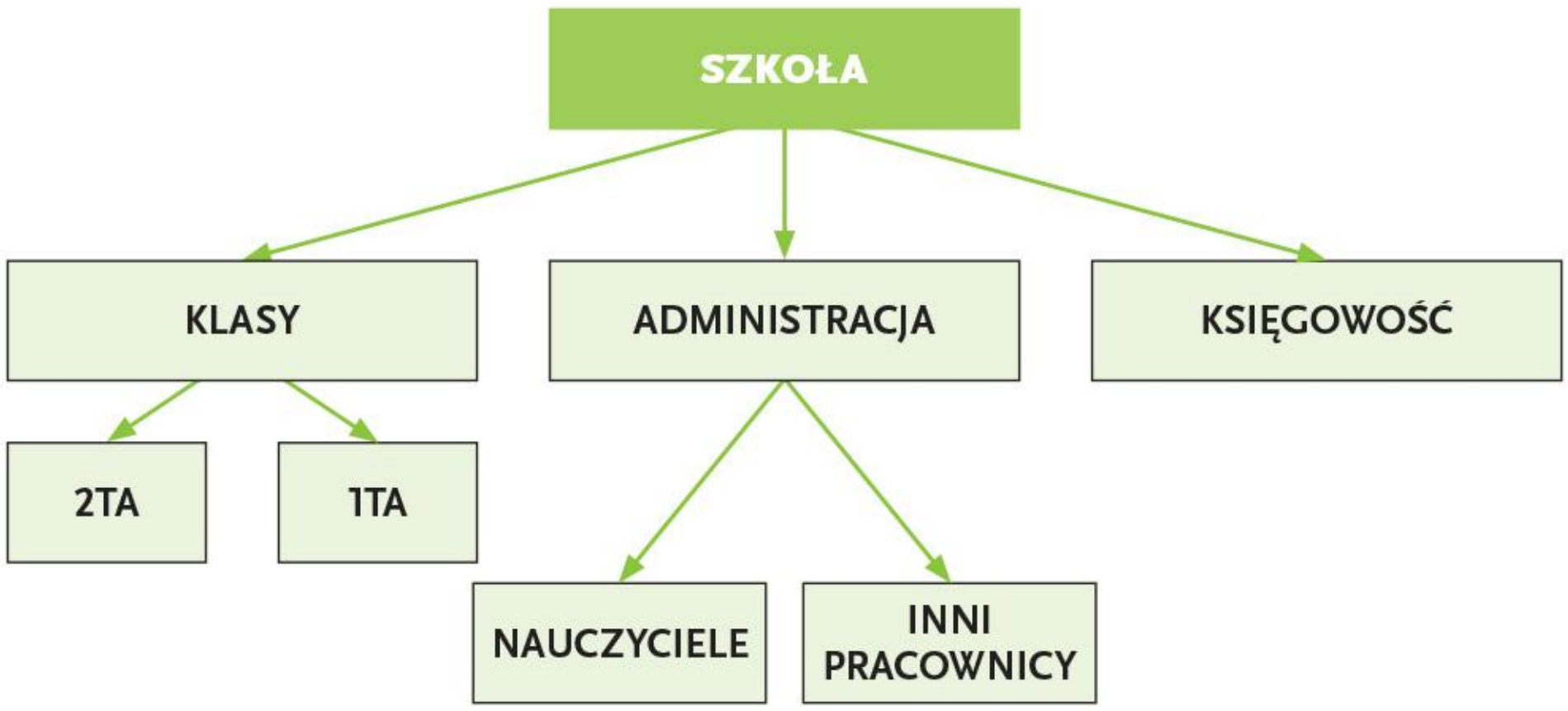
Wyróżnia się następujące modele danych:

- **model jednorodny** (rys. 1.1) – proste bazy danych oparte na jednej tabeli; mogą się charakteryzować nadmiarowością danych i utrudnionym wyszukiwaniem informacji;

Abacki Grzegorz, Legnica, ul. Sarmacka 34 tel. 4539878234
Aczecki Marcin, Legnica, ul. Piotrkowska 3 tel.987345098
Agowska Agata, Łódź, ul. Biała 23 tel.342198342
Baczaski Zenon, Gdańsk, ul. Długa 33 tel. 594322234

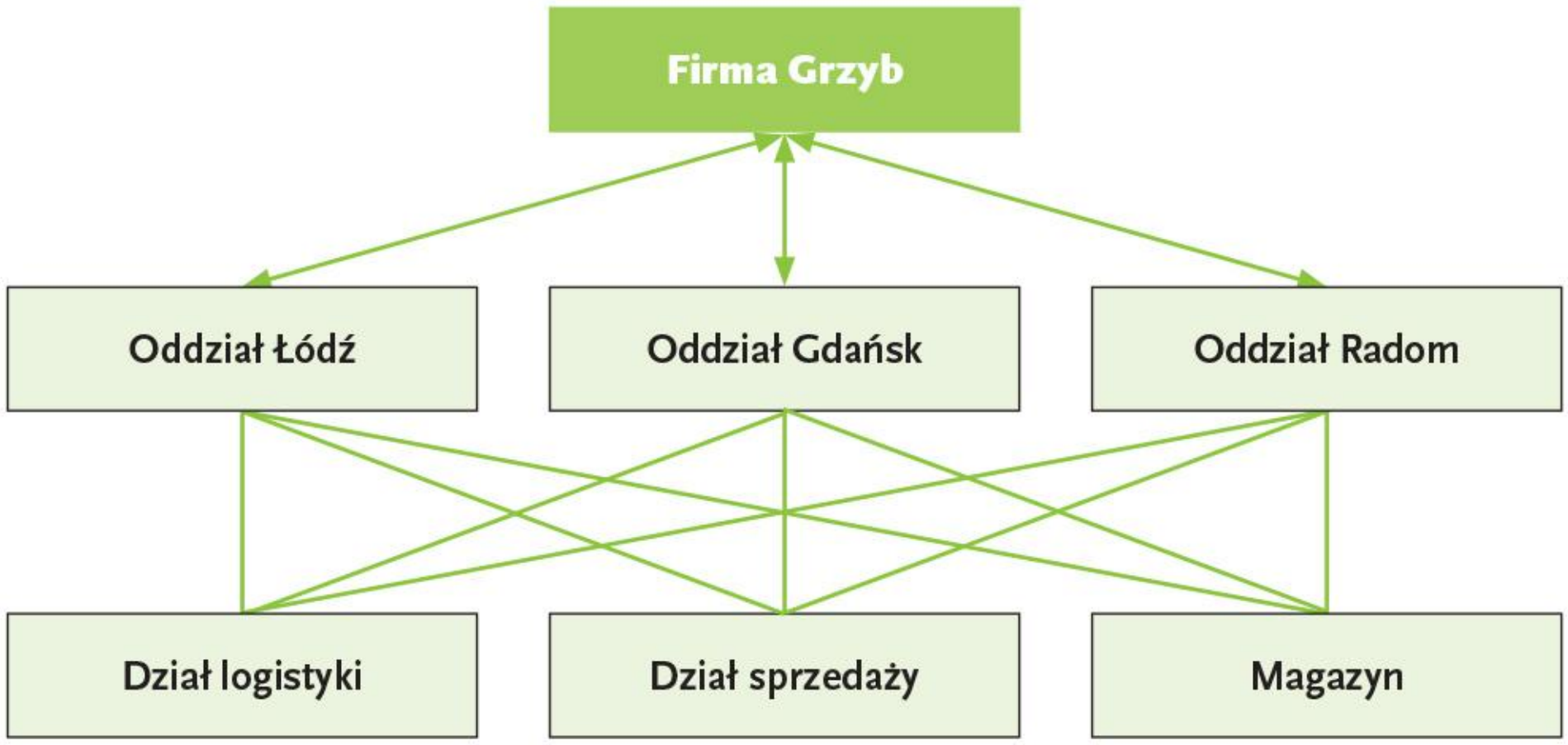
Rys. 1.1. Przykład modelu jednorodnego

- **model hierarchiczny** (rys. 1.2) – oparty na strukturze odwróconego drzewa; dla tego modelu jest charakterystyczna zależność typu podrzędny-nadrzędny;



Rys. 1.2. Przykład modelu hierarchicznego

- **model sieciowy** (lub **rozproszony**) (rys. 1.3) – poszczególne elementy mogą łączyć się z innymi i tworzyć kratownicę; może być wizualizacją organizacji obiegu dokumentów;

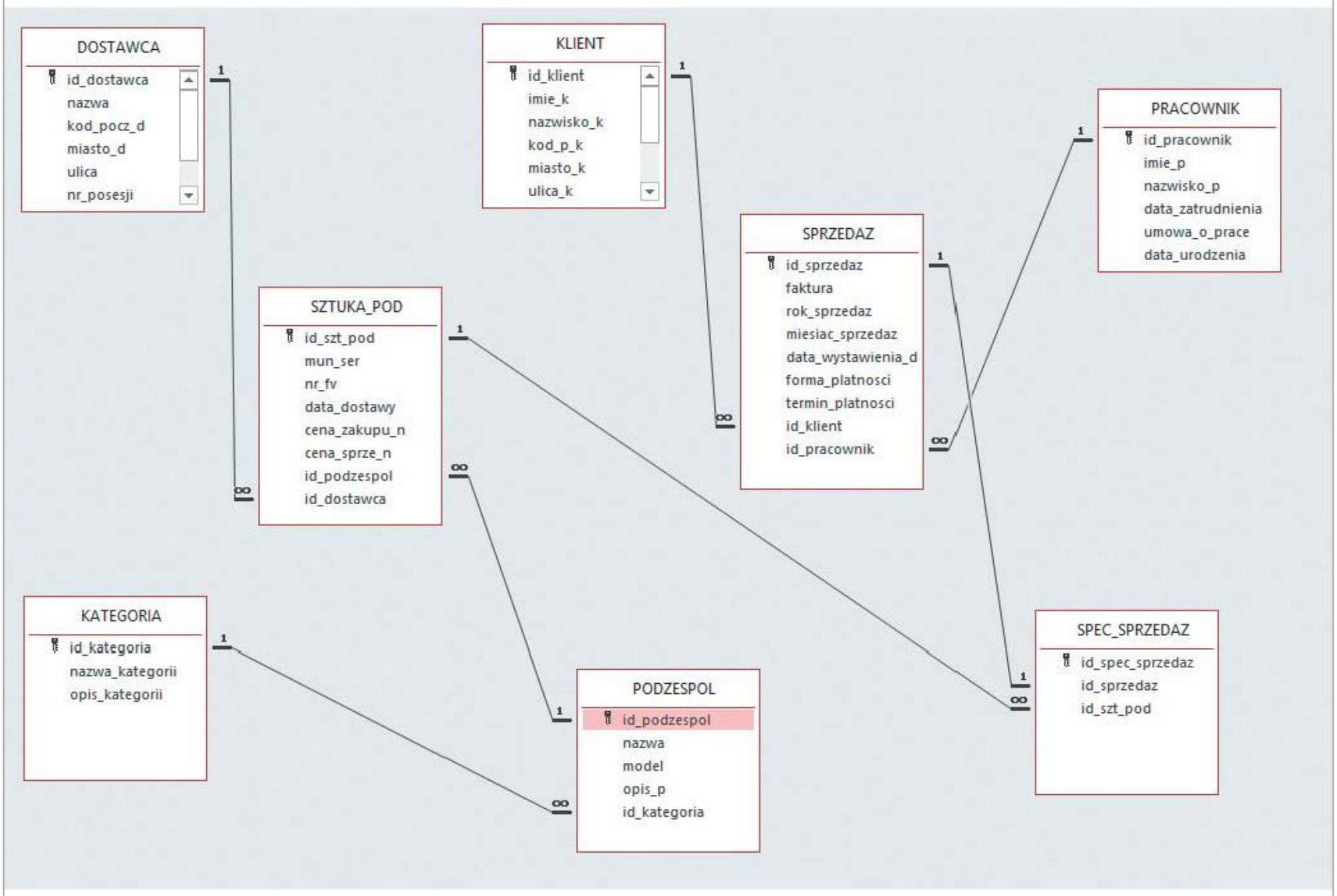


Rys. 1.3. Przykład modelu sieciowego

- **model relacyjny** (rys. 1.4 i 1.5) – jego podstawą jest przechowywanie danych w postaci relacyjnej; w praktyce dane w tym modelu są grupowane w relacje, które prezentuje się w formie tabel; istotną cechą tego modelu jest zależność między polami opisującymi cechy poszczególnych obiektów oraz tworzącymi wiersze a kolumnami definiującymi typ danych umieszczanych w tych polach; najprostsze relacyjne bazy danych mogą składać się z jednej tabeli, ale znacznie częściej są budowane bazy reprezentowane przez wiele tabel, między którymi zachodzą powiązania (ang. *relationship*);

Imię	Nazwisko	Wiek	Wzrost
Andrzej	Abacki	12 lat	145 cm
Jan	Babacki	13 lat	133 cm
Zenon	Cabacki	14 lat	156 cm

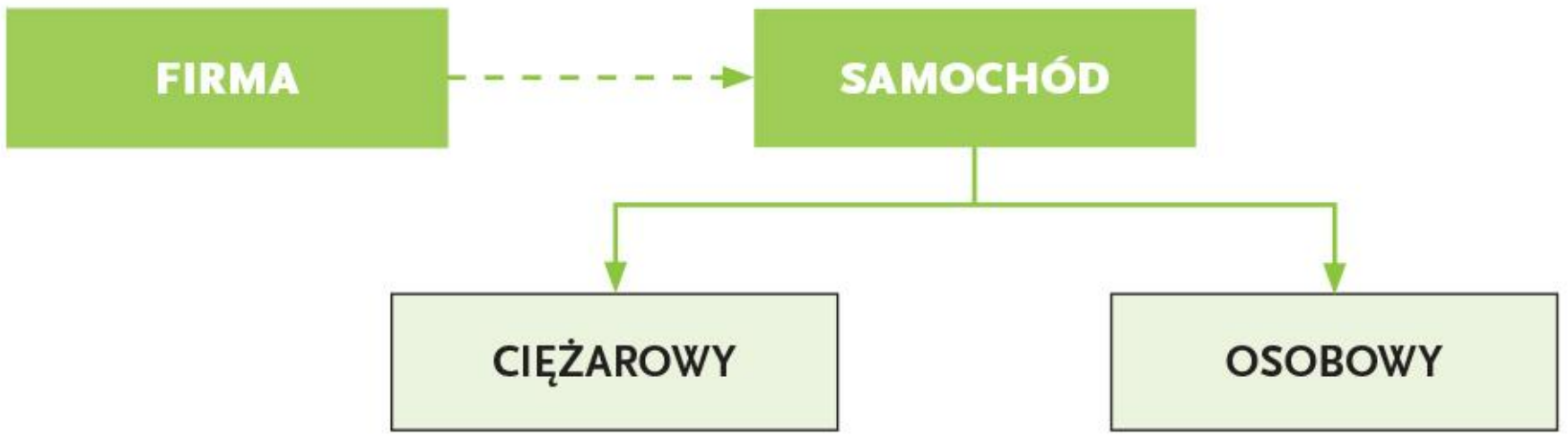
Rys. 1.4. Przykład modelu relacyjnego opartego na jednej tabeli



Rys. 1.5. Przykład modelu relacyjnego opartego na wielu tabelach ze sobą powiązanych

- **model postrelacyjny (relacyjno-obiektowy)** – w strukturze relacyjnej bazy danych mogą być przechowywane dane typu obiektowego; model ten umożliwia tworzenie w tabelach kolumn, w których są przechowywane dane typu obiektowego; umożliwia to definiowanie zmiennych oraz metod, które będą wykonywane na wartościach wprowadzanych do obiektu;

- **model obiektowy** (rys. 1.6) – opiera się na koncepcji obiektów, w których obiekt jest odwzorowaniem rzeczywistości lub abstrakcji; w obiektowej bazie danych obiekty są reprezentacją typów obiektowych (klas) z metodami oraz dziedziczeniem.



Rys. 1.6. Przykład modelu obiektowego

W realiach dynamicznego rozwoju baz danych można spotkać się z określeniami charakteryzującymi jeszcze inne modele. Do takich przykładów zalicza się:

- **hurtownie danych OLAP** (ang. *Online Analytical Processing*) – określają struktury danych do szybkiego wykonywania zapytań i analiz;
- **transakcyjne bazy danych OLTP** (ang. *Online Transaction Processing*) – bazy optymalizowane pod kątem maksymalnej wydajności transakcyjnej, wysokiej równoległości i dostępności;
- **grafowe bazy danych** – wykorzystują struktury grafów z węzłami, krawędziami i własnościami do przedstawiania i przechowywania danych oraz do obsługi zapytań;
- **bazy danych dokumentowe JSON** (ang. *JavaScript Object Notation*) – przeznaczone do przechowywania i pobierania informacji w postaci dokumentów JSON zamiast tabel;
- **bazy danych w chmurze** – zbiór danych, uporządkowanych lub nie, przechowywany w prywatnej, publicznej lub hybrydowej chmurze;
- **autonomiczne bazy danych** – działają w chmurze i wykorzystują sztuczną inteligencję do automatyzacji wszystkich operacji na danych.

Bazy danych w chmurze

Przyszłością baz danych może być ich migracja do chmur obliczeniowych. Obecnie istnieją dwa główne modele chmurowych baz danych:

- **model tradycyjny**, w którym zarówno system zarządzania bazą danych, jak i baza danych są wdrażane w chmurze; klient korzystający z usługi w chmurze jest odpowiedzialny również za nadzór i zarządzanie bazą;
- **baza danych jako usługa**, **DBaaS** (ang. *Database-as-a-Service*), w której klient zawiera umowę z dostawcą usług w chmurze w modelu płatnej subskrypcji; dostawca w ramach subskrypcji oferuje również wiele usług operacyjnych, konserwacyjnych i administracyjnych w czasie rzeczywistym.

Do najistotniejszych cech decydujących o migracji systemów baz danych do chmury zalicza się:

- swobodę eksploatacji i rozwoju bazy danych;
- dostępność usług programistycznych i testowych;
- większe bezpieczeństwo danych oraz cesję odpowiedzialności za ochronę danych osobowych i wrażliwych na dostawcę usługi (zwłaszcza w modelach DBaaS);
- niższe koszty przeznaczane przez użytkownika bazy na infrastrukturę sprzętową;
- łatwiejsze usuwanie awarii;
- możliwość zautomatyzowania operacji na bazie danych lub wykorzystania rozwiązań dotyczących samouczenia się maszyn wirtualnych.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Opisz różnicę między modelem hierarchicznym a modelem relacyjnym bazy danych.
2. Co oznacza skrót SZBD?
3. Jakie znasz modele chmurowych baz danych?
4. Jakie cechy ma każdy system baz danych?
5. Czym charakteryzuje się baza danych OLTP?

2 Relacyjne bazy danych

ZAGADNIENIA

- Elementy relacyjnej bazy danych
- Klucz
- Związki między tabelami
- Normalizacja

W systemach informatycznych opartych na bazach danych najbardziej rozpowszechniony jest model relacyjny bazy danych. Podstawą relacyjnych baz danych jest teoria, którą opublikował Edgar Frank „Ted" Codd. Według niej cały model relacyjny jest oparty na matematycznym pojęciu relacji. Relacją nazywa się dowolny podzbiór iloczynu kartezjańskiego pewnych zbiorów. W naszych rozważaniach relacje będą reprezentowane przez **tabele**. Pozwoli to na łatwiejsze zrozumienie teorii relacyjnych baz danych i odróżnienie relacji jako tabeli od relacji jako związku między tabelami. Jeżeli tabela jest typową relacją, to jej kolumny są atrybutami, a wiersze – krotkami.

Krotka to niepowtarzalny zbiór wartości o określonych typach danych umieszczonych w polach opisujący pojedynczy element tabeli bazy danych. Z tej definicji wyłączono pierwszy wiersz tabeli, który jest wierszem nagłówkowym zawierającym nazwy kolumn (atrybutów). Krotki nazywa się też rekordami tabeli bazy danych.

Atrybut to kolumna tabeli mająca określoną nazwę. Atrybuty określają zbiór cech elementu bazy danych, następnie opisywany wartościami. Zbiór wartości atrybutu nazywa się dziedziną lub – inaczej – typem danych.

W projektowaniu relacyjnych baz danych stosuje się dodatkowo pojęcie 'encja'. **Encja** to reprezentacja zarówno obiektu materialnego, jak i niematerialnego (rzecz, osoba, miejsce, zdarzenie, pojęcie) będącego elementem dającym się odróżnić przez określone cechy. Encje o podobnych cechach mogą być grupowane w zbiory encji. Encje są opisane atrybutami. Między nimi mogą zachodzić związki zwane relacjami. Na podstawie projektu zdefiniowanych encji lub grup encji tworzy się tabele relacyjnej bazy danych. Atrybuty encji są podstawą do tworzenia kolumn tych tabel.

Każda właściwie zdefiniowana tabela musi spełniać następujące warunki:

- mieć jednoznaczną nazwę (np. Auto, Pracownik, Pacjent, Faktura);
- każda kolumna (atrybut) tabeli musi mieć jednoznaczną nazwę, która określa cechę elementu bazy (np. imię, kolor oczu, cena), oraz określony typ danych do opisu wartości cechy;

- każde pole tabeli musi zawierać tzw. atomową wartość, czyli wartość niepodzielną zgodną z ustalonym typem danych;
- opis tego samego elementu bazy danych w postaci zdublowanego rekordu nie może się powtarzać;
- nie powinny pojawiać się pola tabeli opisujące cechy nieużyteczne z punktu widzenia funkcjonalności bazy danych;
- każda tabela musi mieć przynajmniej jeden atrybut, który będzie jednoznacznie identyfikował każdy rekord bazy danych; taki atrybut nazywa się **kluczem**.

Rys. 2.1. Elementy tabeli relacyjnej bazy danych

Rodzaje kluczy

Każda tabela musi mieć atrybut jednoznacznie identyfikujący każdy rekord tabeli bazy danych. Taki atrybut nazywa się **kluczem głównym (podstawowym)**. Klucz główny musi spełniać następujące warunki:

- być unikatowy;
- mieć niepodzielną wartość (wartość atomową);
- nie może przyjmować wartości NULL;
- wartość klucza nie może podlegać zmianom.

W bazach danych występują następujące rodzaje kluczy:

- **klucz prosty** – klucz jednoelementowy (jedna kolumna, np. numer PESEL);
- **klucz złożony** – klucz kilkuelementowy (kilka kolumn, np. IMIĘ i NAZWISKO);
- **klucz sztuczny** – dodatkowa kolumna identyfikująca każdy rekord utworzona w sposób sztuczny; z klucza sztucznego korzysta się wtedy, gdy nie można z cech opisujących obiekt wyodrębnić takiej, która spełnia warunki dla klucza głównego (np. ID_KLIENT);
- **klucz obcy** – klucz główny jednej tabeli wykorzystany do tworzenia związków między tabelami, występujący w innej, powiązanej tabeli; klucz obcy nie może być kluczem głównym;
- **klucz kandydujący** – kolumna brana pod uwagę jako przyszły klucz główny w danej tabeli podczas projektowania bazy danych.

Związki między tabelami – relacje

Relacja jako zdefiniowany logiczny związek między tabelami bazy danych może być określona ze względu na liczność (krotność) jako jeden z następujących typów:

- **relacja jeden do jednego** (1:1) (rys. 2.2) – jednemu rekordowi z tabeli A można przyporządkować tylko jeden rekord z tabeli B;

Rys. 2.2. Przykład relacji jeden do jednego

- **relacja jeden do wielu** (1:N) (rys. 2.3) – wiele rekordów z tabeli A można przyporządkować do jednego rekordu w tabeli B;

Rys. 2.3. Przykład relacji jeden do wielu

- **relacja wiele do wielu** (M:N) (rys. 2.4) – wiele rekordów z tabeli A może być przyporządkowanych do wielu rekordów w tabeli B.

Rys. 2.4. Przykład relacji wiele do wielu

Relacje mogą być opcjonalne lub wymuszone. Aby poprawnie zdefiniować relacje między tabelami, stosuje się klucz obcy.

Normalizacja relacyjnej bazy danych

Normalizacja to proces występujący podczas projektowania i optymalizowania relacyjnej bazy danych polegający na analizie encji i wartości atrybutów encji, ma na celu zmniejszenie liczby atrybutów i tworzenie mniejszych encji o dobrze zdefiniowanych związkach. Normalizacja ma na celu wyeliminowanie potencjalnych anomalii, nadmiarowości oraz niespójnych zależności.

Efektem normalizacji bazy danych jest usunięcie redundantnych informacji, co niestety zawsze skutkuje wzrostem złożoności struktury.

Normalizacja jest zbiorem reguł opisujących kolejne poziomy zgodności. Wyróżnia się następujące postaci normalne: 1NF, 2NF, 3NF, 4NF, 5NF, 6NF. W tej części podręcznika skupimy się przede wszystkim na opisaniu przykładów baz danych normalizowanych do 3NF, czyli spełniających kolejno następujące reguły:

- **pierwsza postać normalna (1NF)** – tabela musi być relacją; każda tabela przechowuje wartości elementarne (atomowe); jeśli normalizuje się tabele, czyli rozdzieli wartości atrybutów na dane atomowe, jest konieczne określenie typu tych danych (przykład 2.1);
- **druga postać normalna (2NF)** – tabele bazy muszą spełniać warunki 1NF oraz warunek, że każda tabela powinna przechowywać dane dotyczące tylko konkretnego obiektu; podczas normalizowania do 2NF należy pozostawić w tabeli zbiór atrybutów (kolumn), który jest zależny tylko od klucza głównego (przykład 2.1);
- **trzecia postać normalna (3NF)** – tabele bazy muszą spełnić warunki 1NF i 2NF oraz warunek, że każda kolumna (atrybut) nienależąca do klucza nie zależy też od innej kolumny (atrybutu), nienależącej do klucza; każdy atrybut niebędący kluczem jest bezpośrednio zależny tylko od klucza głównego, a nie – od innej kolumny.

Znormalizowanie bazy danych do 3NF uznaje się za wystarczające dla większości projektów.

PRZYKŁAD 2.1

W tabeli przedstawiono tabelę bazy danych stacji benzynowej. Uwzględniono w niej wystawione faktury i dane odbierających je klientów. Musimy przeprowadzić analizę struktury tabeli i wykonać normalizację do drugiej postaci normalnej 2NF.

Tabela 2.1. Przykład tabeli bazy danych przed normalizacją

Id_klient	Imię	Nazwisko	Miasto	Numer Faktury	Data
1	Adam	Nogawski	Łomża	FV345/2016 FV678/2016	2016-09-12 2016-11-11
2	Cecylia	Madoń	Łask	FV34/2017	2017-01-09
3	Zenon	Kosicki	Ustka	FV222/2016 FV12/2017	2016-02-23 2017-04-23

Druga postać normalna wymaga spełnienia przez tabelę warunków 1NF. Po analizie tabeli musimy doprowadzić do sytuacji, w której w każdym polu będzie atomowa wartość atrybutu.

Tabela 2.2. Przykład tabeli po spełnieniu warunków 1NF

Id_klient	Imię	Nazwisko	Miasto	Numer Faktury	Data
1	Adam	Nogawski	Łomża	FV345/2016	2016-09-12
2	Cecylia	Madoń	Łask	FV34/2017	2017-01-09
3	Zenon	Kosicki	Ustka	FV222/2016	2016-02-23
4	Adam	Nogawski	Łomża	FV678/2016	2016-11-11
5	Zenon	Kosicki	Ustka	FV12/2017	2017-04-23

Druga postać normalna wymaga, aby każda tabela przechowywała dane dotyczące tylko konkretnego obiektu. Nieuniknione są zatem podział przykładowej tabeli na dwie – Klient i Faktura – oraz utworzenie związku między nimi.

Tabela 2.3. Nowa tabela Klient powstała po podziale tabeli stacja benzynowa w 2NF

Id_klient	Imię	Nazwisko	Miasto
1	Adam	Nogawski	Łomża
2	Cecylia	Madoń	Łask
3	Zenon	Kosicki	Ustka

Tabela 2.4. Nowa tabela Faktura powstała po podziale tabeli stacja benzynowa w 2NF

Id_faktura	Numer Faktury	Data	Id_klient
1	FV345/2016	2016-09-12	1
2	FV34/2017	2017-01-09	2
3	FV222/2016	2016-02-23	3
4	FV678/2016	2016-11-11	1
5	FV12/2017	2017-04-23	3

Po użyciu klucza obcego (id_klient) w tabeli Faktura jest możliwe zbudowanie relacji 1N. Jeżeli do numeracji faktur użyjemy typu danych, który zapewni niepowtarzalność tej wartości pola, możemy użyć numeru faktury jako klucza głównego zamiast pola id_faktura.

Pozostałe postacie normalizacji

Czwartą postać normalną 4NF nazywa się też postacią BCNF (ang. *Boyce Codd Normal Form*). PNBC jest następną wersją 3NF. Każda tabela zgodna z PNBC jest także zgodna z 3NF, ale wynikanie odwrotne nie musi w niej zachodzić. Postaci 5NF i 6NF w praktyce rzadko są brane pod uwagę w rzeczywistych projektach. Zignorowanie tych postaci może skutkować projektem niedoskonałym, ale nie powinno wpływać na działanie bazy danych.

Do najważniejszych zalet normalizacji baz danych zalicza się:

- zmniejszenie liczby danych przechowywanych w tabelach bazy;
- rozwiązanie problemów występujących podczas dodawania, modyfikacji i usuwania danych w bazie;
- wydajniejsze zarządzanie transakcjami.

Największą wadą stosowania normalizacji baz danych jest możliwość spowolnienia wykonywania zapytań ze względu na rozrost struktury i większą liczbę danych.

Typy danych

Typ danych określa rodzaj i zakres wartości, jakie mogą przyjmować dane wpisywane do relacyjnej bazy danych.

Najczęściej wykorzystywane typy w tworzeniu struktury relacyjnych baz danych przedstawiono w tabeli 2.5.

Tabela 2.5. Typy danych

Typ danych	Opis
INTEGER	4-bajtowy typ całkowity z zakresem: –231 (–2 147 483 648) do 231–1 (2 147 483 647)
FLOAT	typ zmiennoprzecinkowy z zakresem: –1.79E + 308 do –2.23E – 308, 0 = 2.23E – 308 do 1.79E + 308; liczba bajtów zależy od precyzji. 1–24 – 4 bajty (7 cyfr), 25–53 – 8 bajtów (15 cyfr)
DECIMAL	wielkość typu zależy od precyzji: 1–9 – 5 bajtów, 10–19 – 9 bajtów, 20–28 – 13 bajtów, 29–38 – 17 bajtów
CHAR	pole tekstowe o stałej długości znaków, może przechowywać od 0 do 255 znaków
VARCHAR	pole tekstowe o zmiennej długości od 0 do 255 znaków
DATE	10-znakowy typ zapisu daty z precyzją: 10.0; domyślnym formatem jest YYYY-MM-DD, natomiast domyślna wartość to 1900-01-01
TIME	3–5-bajtowy typ do zapisu czasu z dokładnością do 100 ns; liczba bajtów zależy od skali precyzji
DATETIME	8-bajtowy typ do zapisu daty z zakresu 1 stycznia 1753 do 31 grudnia 9999 z dokładnością do 0,00333 sekundy
ENUM	wyliczanie; każdy rekord ENUM musi być unikatowy w danej kolumnie
TEXT	tekst o maksymalnej długości 65 535 znaków
TINYINT	liczby typu całkowitego; zakres wynosi od –128 do 127 lub od 0 do 255
BINARY	typ o wielkości 1–8000 bajtów, przechowywany najczęściej jako stały strumień bajtów
BIT	wartości 0, 1 lub NULL

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jakie atrybuty przypiszesz do cech elementów należących do encji SAMOCHÓD?
2. Jakie krotności relacji występują między encjami:
   a) ZOO – ZWIERZĘ?
   b) UCZEŃ – KLASA?
   c) WYPOŻYCZENIE – KSIĄŻKA?
   d) KSIĄŻKA – AUTOR?
3. Jakie cechy powinna mieć właściwie utworzona tabela w relacyjnej bazie danych?
4. Jakie są zalety i wady stosowania normalizacji w bazach danych?
5. Zdefiniuj typ danych VARCHAR.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Określ, której postaci nie spełnia ta tabela.

Klient	Nr zamówienia	Nazwa towaru
Adam Nowak	2	Karta sieciowa, HDD 1TB
Andrzej Kowalik	1	CPU i5, Obudowa ATX
Zenon Jarocki	3	Pamięć RAM, Karta graficzna

2. Przeanalizuj strukturę tabeli i określ, który z jej atrybutów (kolumn) mógłby zostać kluczem głównym. Uzasadnij swój wybór.

Imię	Nazwisko	Pesel	Nr_telefonu	Nr_legitymacji	Miasto
Tomasz	Gniew	98121209023	604 345 543	123/2017	Olsztyn
Jan	Caban	99030190923	501 223 332	123/2018	Gniezno

3. Wykonaj normalizację tabeli relacyjnej bazy danych do 3NF.

Nr. pozycji	Nr zamówienia	Nazwa klienta	Adres	Podzespół	Cena	Kategoria	Dostawca
1	1101	Jan Matejka	Warszawa, ul. Sabały 8	HDD	234,00	Napędy masowe	Megor S.A.
2	999	Jeremi Harman	Szczecin, ul. Mirabella 12	Karta dźwiękowa	199,00	Podzespoły bazowe	ABC sp. z o.o.
3	1101	Jan Matejka	Warszawa, ul. Sabały 8	Klawiatura	34,70	Urządzenia wskazujące	ASIS-TEK
4	998	Adam Trzek	Zgierz, ul. Znienacka 2	Napęd DVD	112,00	Napędy masowe	Megor S.A.
5	997	Zenon Kosior	Opole, ul. Zapole 34	Karta graficzna	443,00	Podzespoły bazowe	Tesla

4. Dopasuj odpowiedni typ danych do poszczególnych atrybutów encji PACJENT.

Atrybut	Typ danych
Imię	
Nazwisko	
PESEL	
Data urodzenia	
Wzrost	
Waga	
Ubezpieczenie	

II. Zasady projektowania baz danych

- Modelowanie konceptualne
- Etapy i strategie projektowania modelu bazy danych
- Diagramy ER i UML
- Programy typu CASE
- Tworzenie diagramów ER

3 Model baz danych

ZAGADNIENIA

- Zasady projektowania baz danych
- Etapy projektowania baz danych
- Notacje diagramów ER i UML
- Rodzaje strategii projektowania baz danych

Model bazy danych

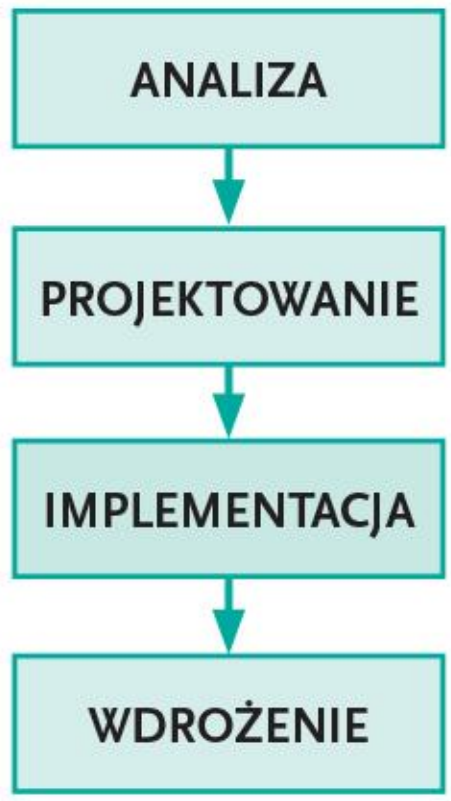
Każda baza danych jest cyfrową reprezentacją określonego obszaru świata rzeczywistego. Należy zatem budować taki model rzeczywistości, który łatwo i logicznie można przekształcić w system bazy danych. Projektowanie baz danych polega właśnie na budowaniu modeli struktur, wykorzystywanych później do tworzenia fizycznych baz danych. Najważniejsze procesy każdego cyklu projektowego systemu baz danych pokazano na rys 3.1.

Modelowanie można podzielić na konceptualne i implementacyjne. Modelowanie konceptualne polega na reprezentacji poszczególnych obiektów tworzących daną rzeczywistość w uniwersalnym modelu. Modelowanie implementacyjne odzwierciedla model konceptualny w konkretnych strukturach danych (relacyjne, obiektowe itp.).

Do najważniejszych zalet tworzenia modelu bazy danych zalicza się:

- logiczne odzwierciedlenie rzeczywistości;
- czytelną prezentację projektu bazy;
- tworzenie platformy komunikacji między twórcą a odbiorcą (użytkownikiem);
- umożliwienie zapanowania nad złożonością całego projektu;
- podnoszenie jakości procesu tworzenia bazy danych;
- ułatwianie znalezienia błędów dotyczących koncepcji struktury bazy lub problemów z tym związanych.

Proces projektowania bazy danych opiera się na teoretycznej (konceptualnej) analizie środowiska, którego ma dotyczyć tworzona baza danych. Analizowany obszar rzeczywistości opisuje się za pomocą określonej notacji. Najczęściej wykorzystuje się notacje modeli **ER** (ang. *Entity Relationship Model*) oraz **UML** (ang. *Unified Modeling Language*). Modelowanie ER wywodzi się z podejścia strukturalnego, natomiast model UML wspiera podejście obiektowe. Do naszych zastosowań będziemy wykorzystywać notację według modelu ER. Zależnie od rodzaju modelowania oraz zastosowania modelu używa się różnych pojęć opisujących obiekty struktury. Podstawowe pojęcia dotyczące projektowania baz danych według różnych modeli zebrano w tabeli 3.1.



Rys. 3.1. Cykl projektowy systemu baz danych

Tabela 3.1. Podstawowe pojęcia w projektowaniu baz danych

Obiekty lub **encje** bądź **instancje**	dające się zidentyfikować elementy rzeczywistości najczęściej opisywane za pomocą rzeczowników lub nazw własnych
Atrybuty lub **własności**	cechy obiektów mające wartość i należące do określonego zbioru; najczęściej opisywane za pomocą przymiotników i liczebników
Związki lub **relacje**	elementy występujące między dwoma lub większą liczbą obiektów
Klasy	zbiory encji lub obiektów w diagramach UML mających wspólne cechy, należących do jednego rodzaju opisywane najczęściej za pomocą rzeczowników pospolitych w liczbie mnogiej

Cały proces projektowania relacyjnych baz danych można podzielić na następujące etapy:

1. zdefiniowanie przeznaczenia bazy danych;
2. zebranie informacji dotyczących przetwarzania danych, obiegu i wyglądu dokumentacji oraz opisu podmiotów lub zdarzeń składających się na dany system informacyjny;
3. zdefiniowanie poszczególnych encji i ich atrybutów; nadanie jednoznacznych nazw encjom;
4. sprawdzenie poprawności struktury encji pod względem redundancji danych i normalizacji bazy danych;
5. określenie typu danych dla poszczególnych atrybutów encji;
6. określenie kluczy podstawowych dla każdej encji;
7. skonfigurowanie związków między encjami, charakteru wystąpień oraz ich liczności;
8. wykonanie diagramu w aplikacji typu CASE;
9. walidacja i ewentualna ewaluacja projektu;
10. implementacja projektu na kod SQL lub strukturę tabel w MS Access.

Model ER

W **modelu ER** obiekty świata rzeczywistego są reprezentowane za pomocą **encji** (ang. *entities*), a powiązania między obiektami świata rzeczywistego – za pomocą **związków** (ang. *relationships*) między encjami. Cechy encji opisuje się za pomocą atrybutów encji.

W projektowaniu encję określa się jako:

- kategorię rzeczy,
- rzecz lub podmiot,
- zdarzenie,
- miejsce.

Atrybut encji określa się jako:

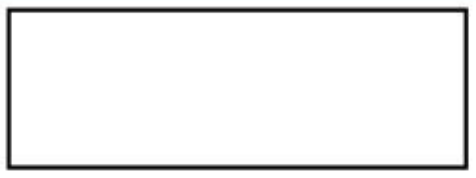
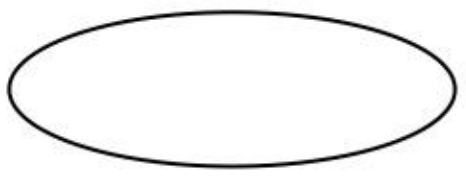
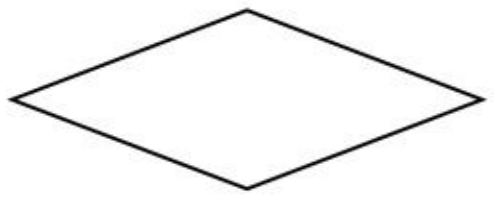
- rzeczownik używany do opisania encji;
- informację, którą trzeba znać w odniesieniu do encji;
- cechę opisującą obiekt.

Związek definiuje się jako:

- sposób zależności jednej encji z innymi;
- reguły biznesowe;
- opis zdarzenia zachodzącego między encjami.

Modele konceptualne ER przedstawia się za pomocą diagramów ERD (ang. *Entity Relationship Diagram*). Można je tworzyć z wykorzystaniem różnych notacji. Najpopularniejsze są diagramy w notacjach Martina, Chena, IDEF1X oraz Barkera.

W tworzeniu diagramów wykorzystuje się poniższe piktogramy.

Element diagramu ER	Opis elementu
	encja
	atrybut encji
	związek między encjami

Rys. 3.2. Elementy diagramu ER

W diagramie **Chena** elementy diagramu łączy się za pomocą linii prostych. Przy połączeniach encji ze związkami określa się ich liczność.

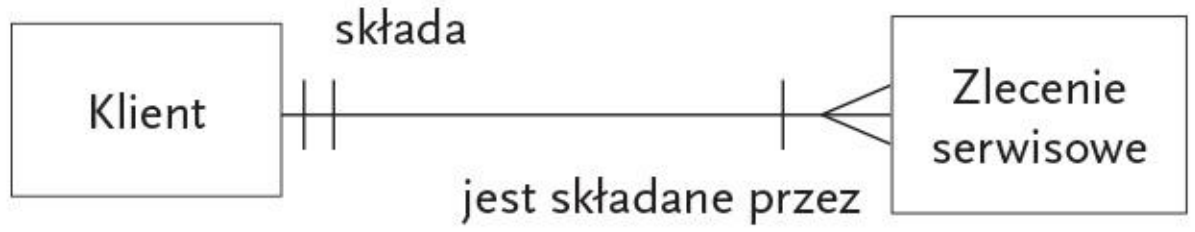
Rys. 3.3. Diagram Chena

W notacji **Martina** encje łączy się liniami prostymi z symboliką określającą liczność związku oraz jego opcjonalność. W tabeli 3.2 zamieszczono wykaz oznaczeń powiązań.

Tabela 3.2. Rodzaje oznaczeń powiązań w notacji Martina

powiązanie określające liczność relacji 1 do 1	
powiązanie określające liczność relacji 1 do N	
powiązanie określające liczność relacji N do M	

Przykład zastosowania tych oznaczeń przedstawiono na rys. 3.4.

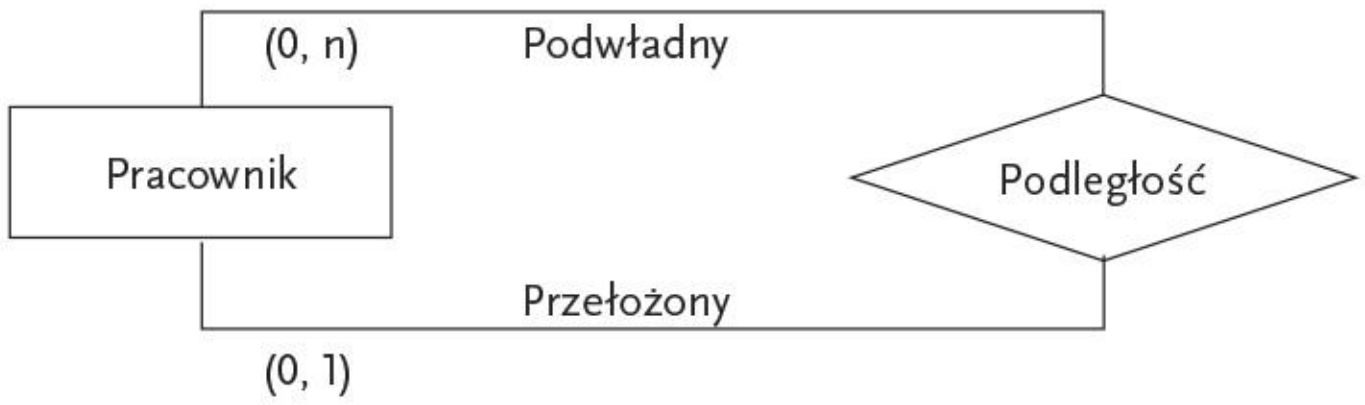


Rys. 3.4. Przykład zastosowania oznaczeń w notacji Martina

Opcjonalność związku między encjami określa się na diagramie ustalonymi symbolami. Przykład opcjonalności w notacji Martina pokazano na rys. 3.5.

Rys. 3.5. Przykład opcjonalności w notacji Martina

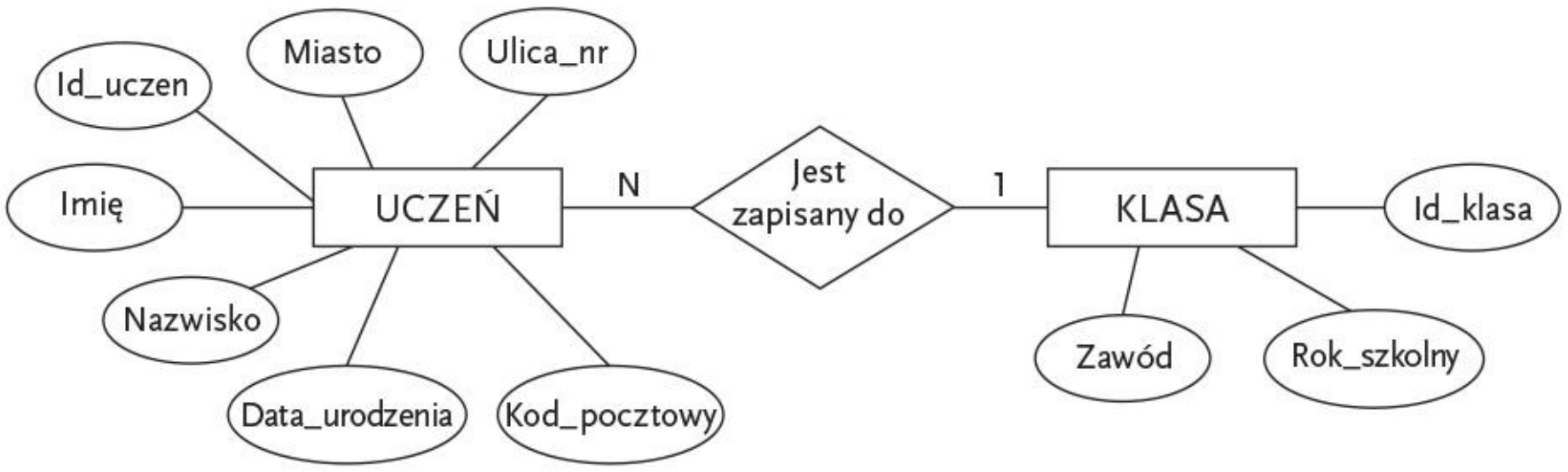
W diagramach modelu ER mogą również zachodzić **związki rekurencyjne**. Jest to szczególny związek, który zachodzi między tą samą encją.



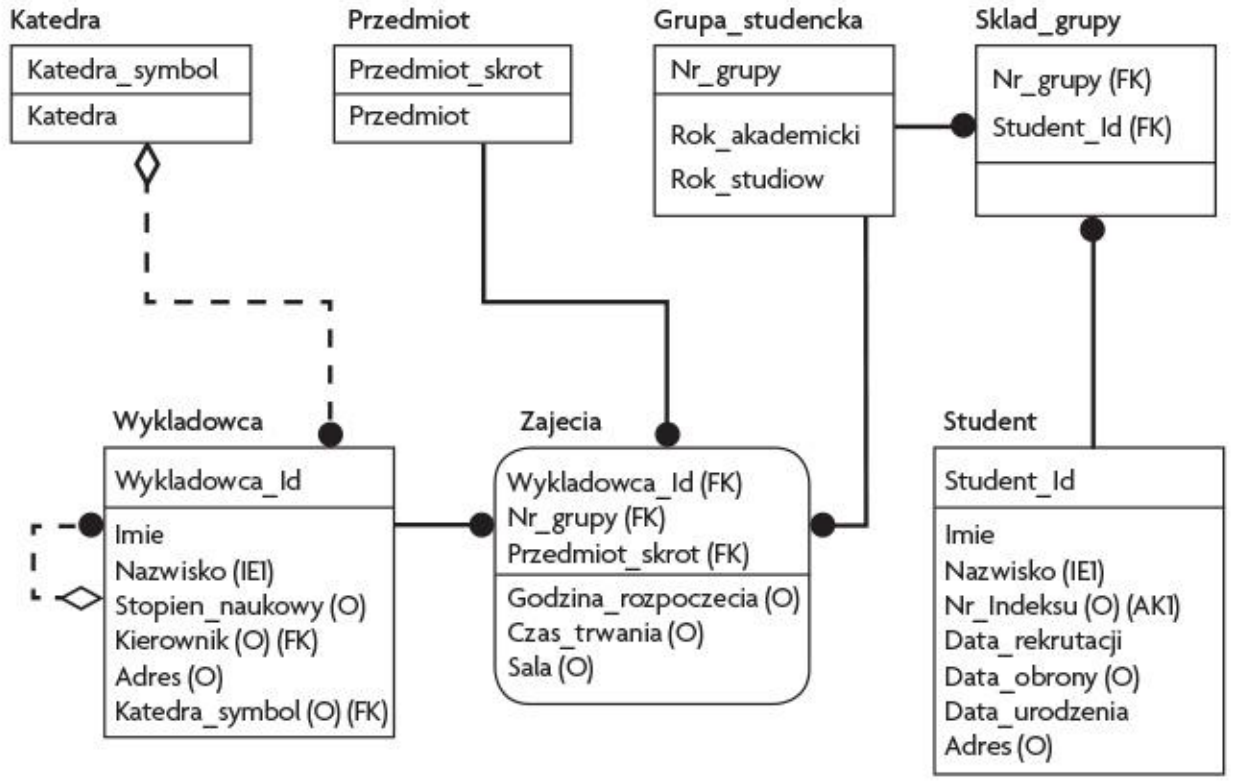
Rys. 3.6. Związek rekurencyjny w diagramie modelu ER

PRZYKŁAD 3.1

Diagram ER opisujący encje Uczeń i Klasa wraz z opisującymi je atrybutami zapisany w notacji Chena pokazano na rys. 3.7.



Rys. 3.7. Diagram ER opisujący encje Uczeń i Klasa

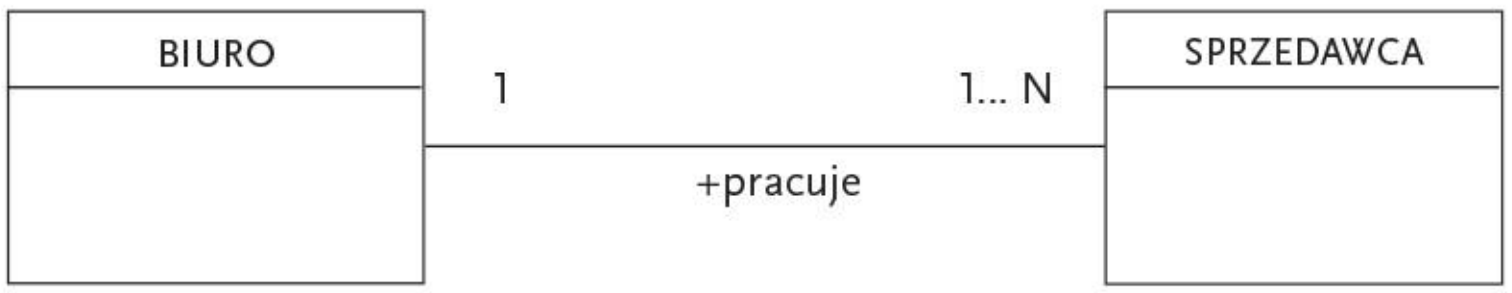


Rys. 3.8. Przykład diagramu IDEF1X

Model UML

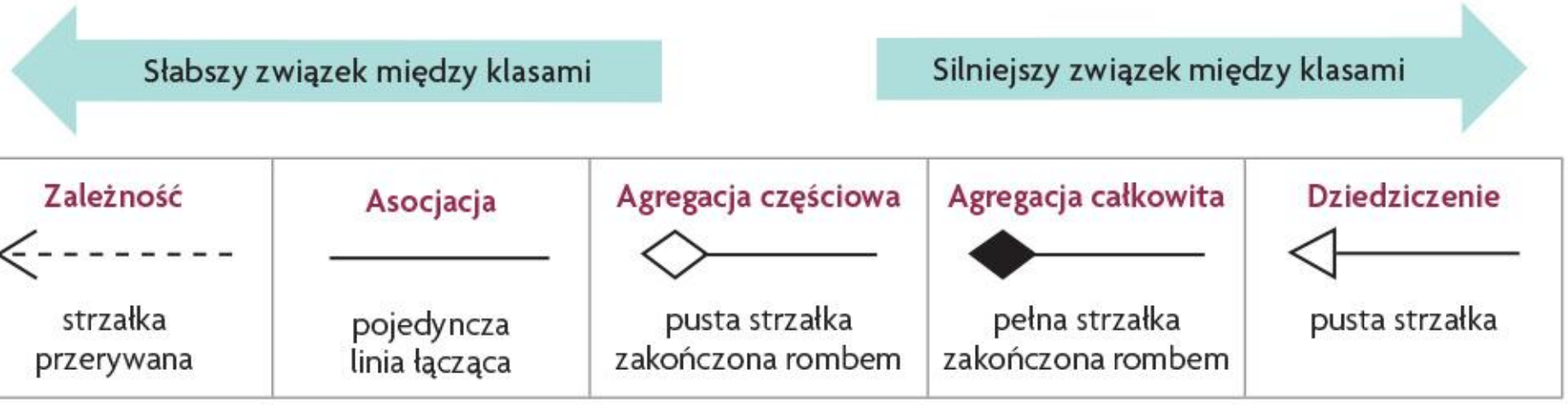
Model **UML** (ang. *Unified Modeling Language*) to jasno wyspecyfikowany graficzny język modelowania składający się z kilkunastu typów diagramów. Te diagramy pozwalają na formalne opisywanie i modelowanie struktur bądź procesów. Modelowanie UML wykorzystuje się w wielu dziedzinach projektowania konceptualnego związanego z tworzeniem baz danych, kompleksowych systemów informatycznych czy oprogramowania.

Notacja w języku UML jest dostępna m.in. w programie MS Visio. Model UML wykorzystuje się do modelowania systemów informatycznych przez stosowanie symboli graficznych i odpowiedniego opisu. Najpopularniejszym diagramem UML jest diagram klas.



Rys. 3.9. Model UML

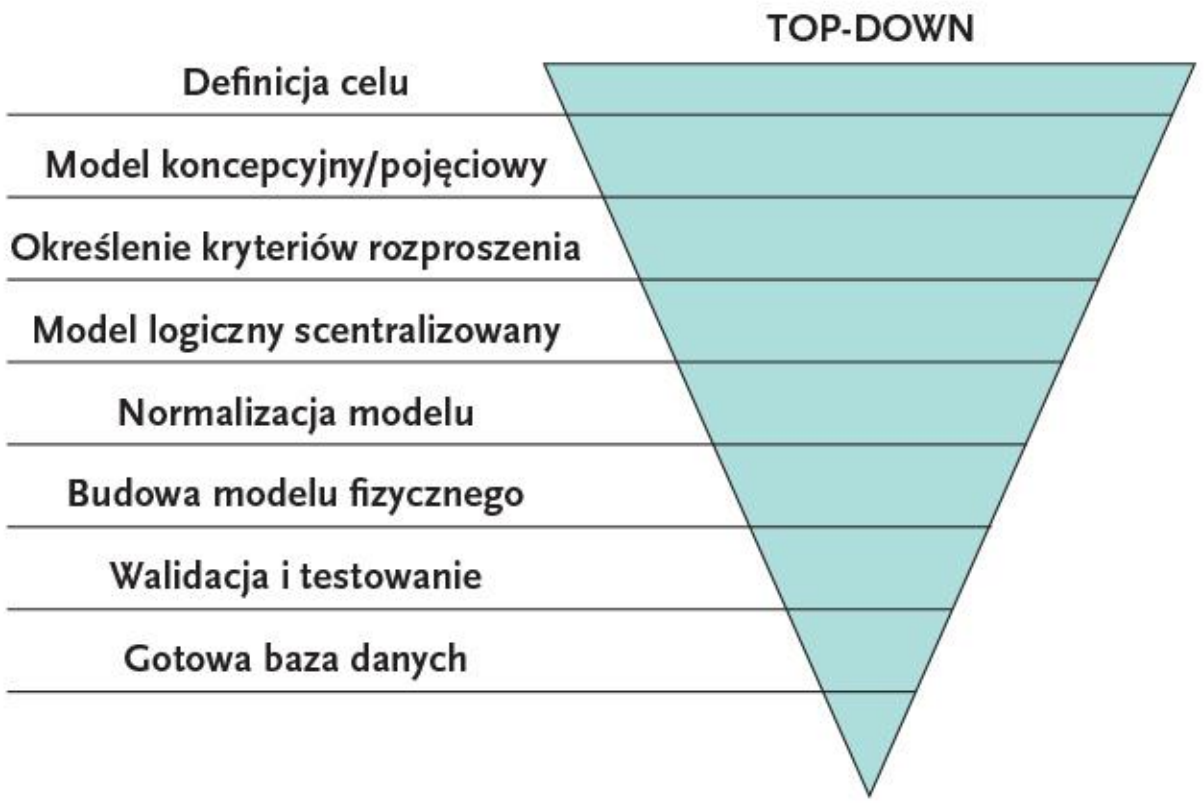
W diagramie UML związki między klasami określa się za pomocą łączników. Przykładowo: agregację oznacza się za pomocą linii zakończonej pustym rombem, natomiast asocjację – za pomocą linii, która może być zakończona strzałką (oznaczającą kierunek powiązania klas). Nazwę cechy wraz z krotnością umieszcza się w punkcie docelowym asocjacji. Na rys. 3.10 pokazano przykłady wszystkich możliwych oznaczeń związków między klasami.



Rys. 3.10. Rodzaje oznaczeń związków między klasami w diagramach UML

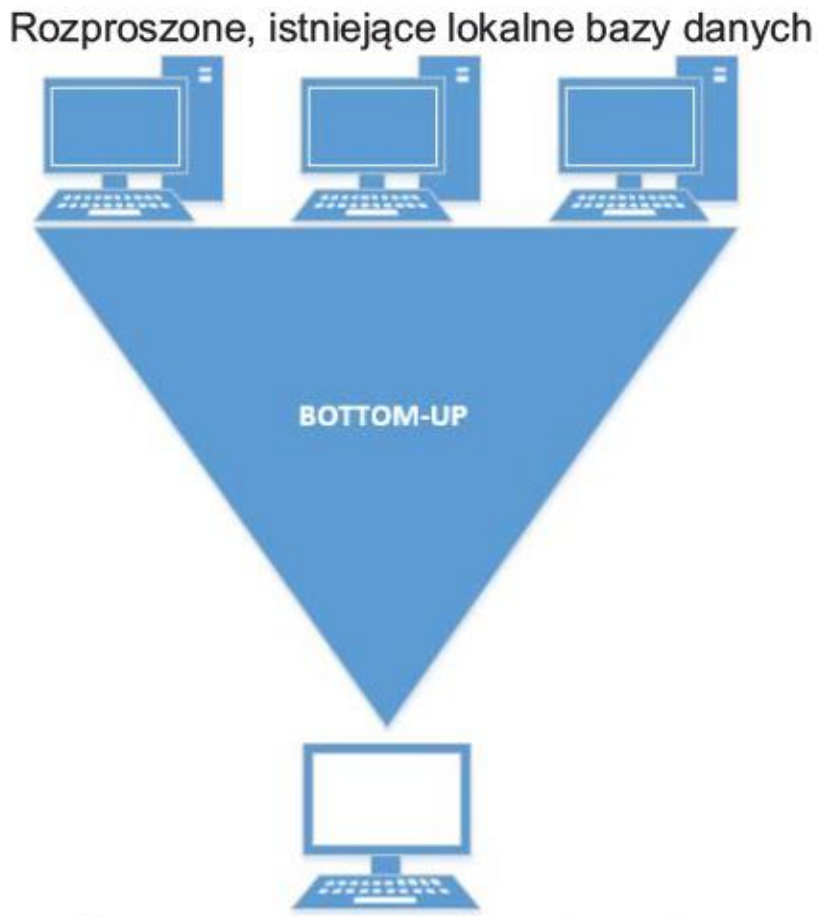
Strategie projektowania baz danych

Budowanie diagramu może odbywać się według strategii przyjętych na początku projektowania. Najczęściej stosuje się strategię **TOP-DOWN**. Definiuje ona analizę obszaru od ogółu do pojęć szczegółowych. Ta strategia daje najlepszy obraz ogólny struktury projektu oraz gwarantuje odpowiednią jakość.



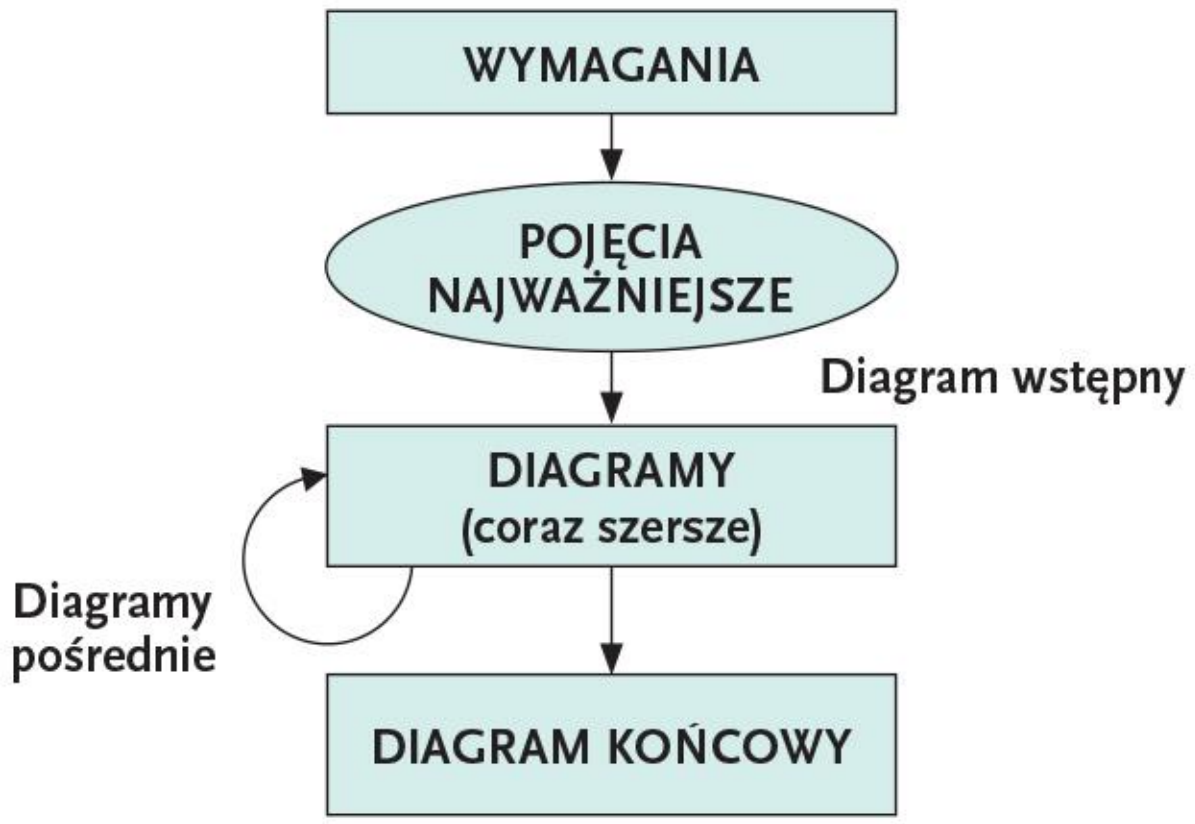
Rys. 3.11. Przykład procesu projektowania bazy danych według strategii TOP-DOWN

Innym rodzajem strategii projektowania jest podejście **BOTTOM-UP**. Zgodnie z nim analizę obszaru rozpoczyna się od pojęć elementarnych do ogółu. Tego rodzaju strategię wykorzystuje się dla już istniejących struktur danych. Jej wadami są brak jednoznacznych odniesień do potrzeb użytkowników oraz niepewna jakość.



Rys. 3.12. Przykład procesu projektowania bazy danych według strategii BOTTOM-UP

Istnieją jeszcze strategie typu **INSIDE-OUT**, w których analizę rozpoczyna się od definiowania encji najważniejszych, a potem rozwija się projekt o kolejne elementy, oraz **MIXED**, w której wykorzystuje się elementy wszystkich wcześniej wymienionych strategii.



Rys. 3.13. Przykład procesu projektowania bazy danych według strategii INSIDE-OUT

Narzędzia do modelowania konceptualnego

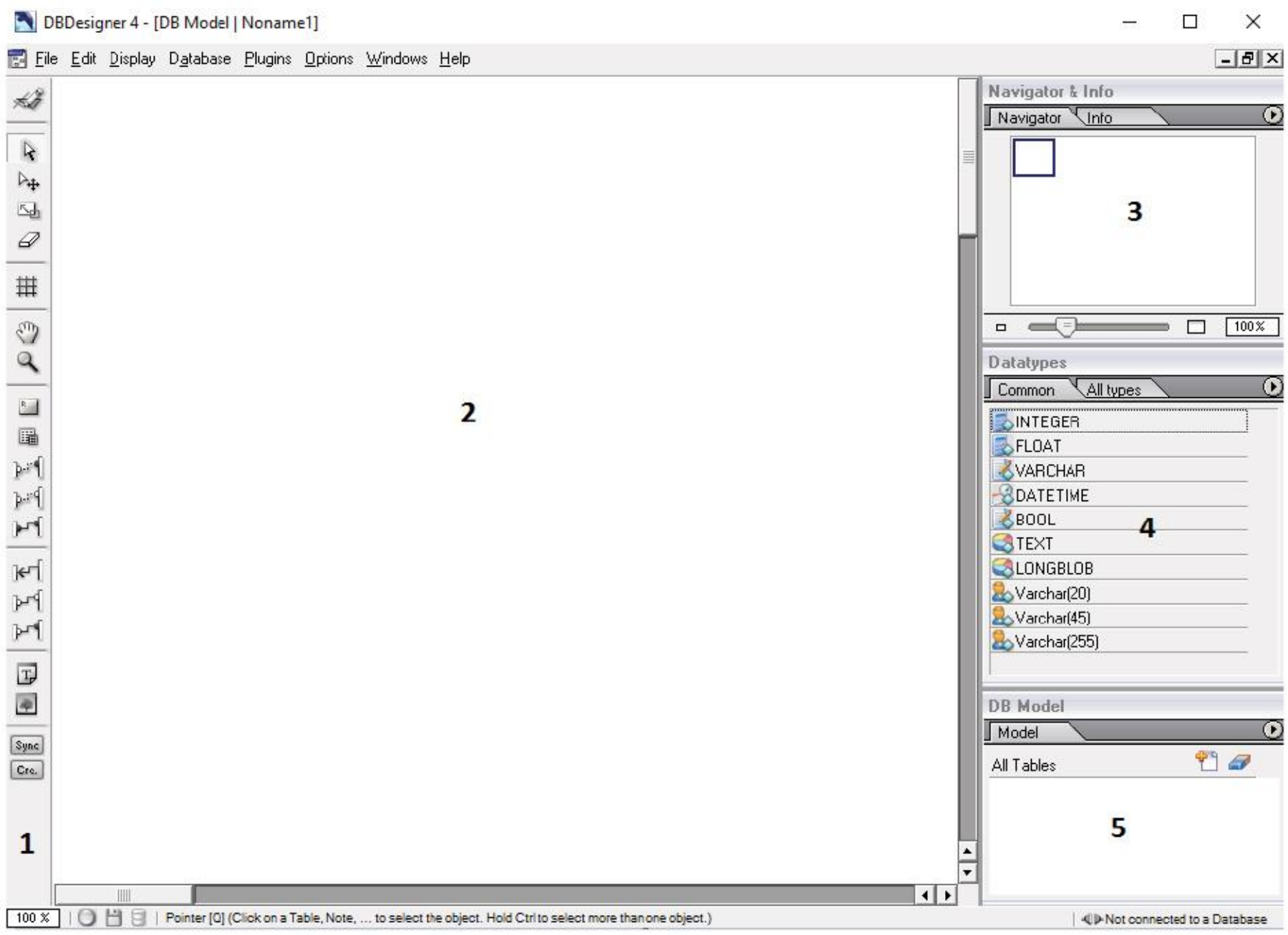
Modelowanie konceptualne jest szkicem przyszłej fizycznej struktury bazy danych. Forma prezentacji modelu za pomocą diagramu pozwala na wykorzystanie dowolnych narzędzi, które umożliwiają prezentację obiektów i związków między nimi w określonej notacji. Dlatego dopuszcza się nawet formę odręcznego szkicu na kartce lub w najprostszym edytorze graficznym (tak jak wykonano rysunek w przykładzie 3.1). Taka prezentacja jednak ma duże wady. Wszelkie zmiany będą pogarszać ogólną czytelność diagramu lub wymuszą wykonanie nowego rysunku. Co więcej, są pozbawione użytecznych narzę-

dzi (takich jak biblioteki piktogramów), typów danych lub automatyzacji w tworzeniu powiązań. Dlatego dobrą praktyką jest korzystanie z odpowiednich edytorów.

Do tworzenia diagramów stosuje się oprogramowanie typu **CASE** (ang. *Computer Aided Software Engineering*). Jest to typ aplikacji używany do komputerowego wspomagania projektowania konceptualnego oprogramowania. Zawiera narzędzia do tworzenia diagramów w notacji ER lub UML. Przykładami takich aplikacji są: MS Visio, StarUML i DBDesigner4. Oprócz możliwości tworzenia diagramów programy tego typu mogą być wyposażone w opcje generowania skryptów w Visual Basic-u, SQL-u czy ODBC.

DBDesigner4

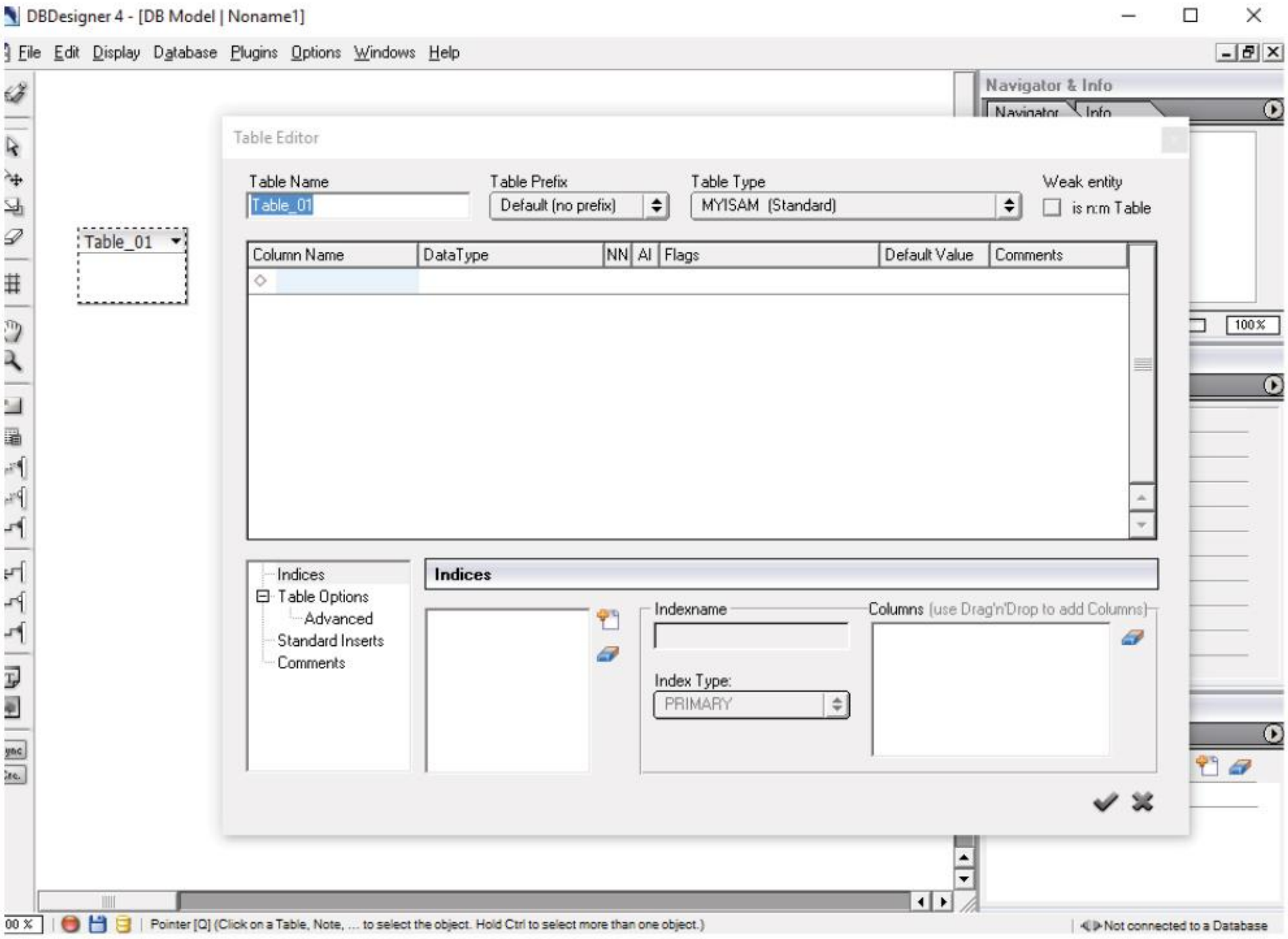
Program DBDesigner4 to aplikacja typu **CASE** do wizualnego modelowania i tworzenia diagramów ERD relacyjnych baz danych. Program jest darmowy i można go pobrać ze strony http://fabforce.eu/downloads.php. Aplikacja jest dostępna na systemy Windows oraz Linux.



Rys. 3.14. Budowa okna programu DBDesigner4

Główne okno programu DBDesigner4 (rys. 3.14) składa się z 5 obszarów: 1 – pasek narzędzi, 2 – obszar roboczy, 3 – okno nawigacji i informacji, 4 – okno typów danych, 5 – okno bieżącego modelu bazy danych.

Pracę z DBDesigner4 zaczyna się od utworzenia nowego projekty opcją **`File/New`**. W celu dodania nowej tabeli do projektu wybiera się z paska narzędzi ikonę tabeli i wskazuje miejsce umieszczenia tabeli na obszarze roboczym. Edycja nowo utworzonej tabeli następuje po dwukrotnym naciśnięciu lewego klawisza myszy na nowym elemencie. Pojawi się wówczas okno edytora tabeli (rys. 3.15).



Rys. 3.15. Tworzenie tabel w programie DBDesigner4

Następnie nadaje się nazwę tabeli (encji) oraz wprowadza nazwy poszczególnych atrybutów encji. W celu uporządkowania struktury projektu pierwszy atrybut powinien być również kluczem podstawowym (głównym). Warunkiem koniecznym do określenia klucza głównego jest zaznaczenie w wierszu opisującym właściwości atrybutu pól **`NN`** (*not null*) oraz **`AI`** (*auto increment*). Pierwsze informuje nas, że pole atrybutu nie może być puste, natomiast przez drugie zostaje wymuszone autonumerowanie wierszy w kolumnie opisanej tym atrybutem.

Kolejna operacja to zdefiniowanie typu danych dla każdego atrybutu encji. W programie DBDesigner4 do wyboru mamy wiele typów, których podgląd znajduje się w oknie **`Typy danych`**. Wszystkie te typy są dostępne również w języku SQL.

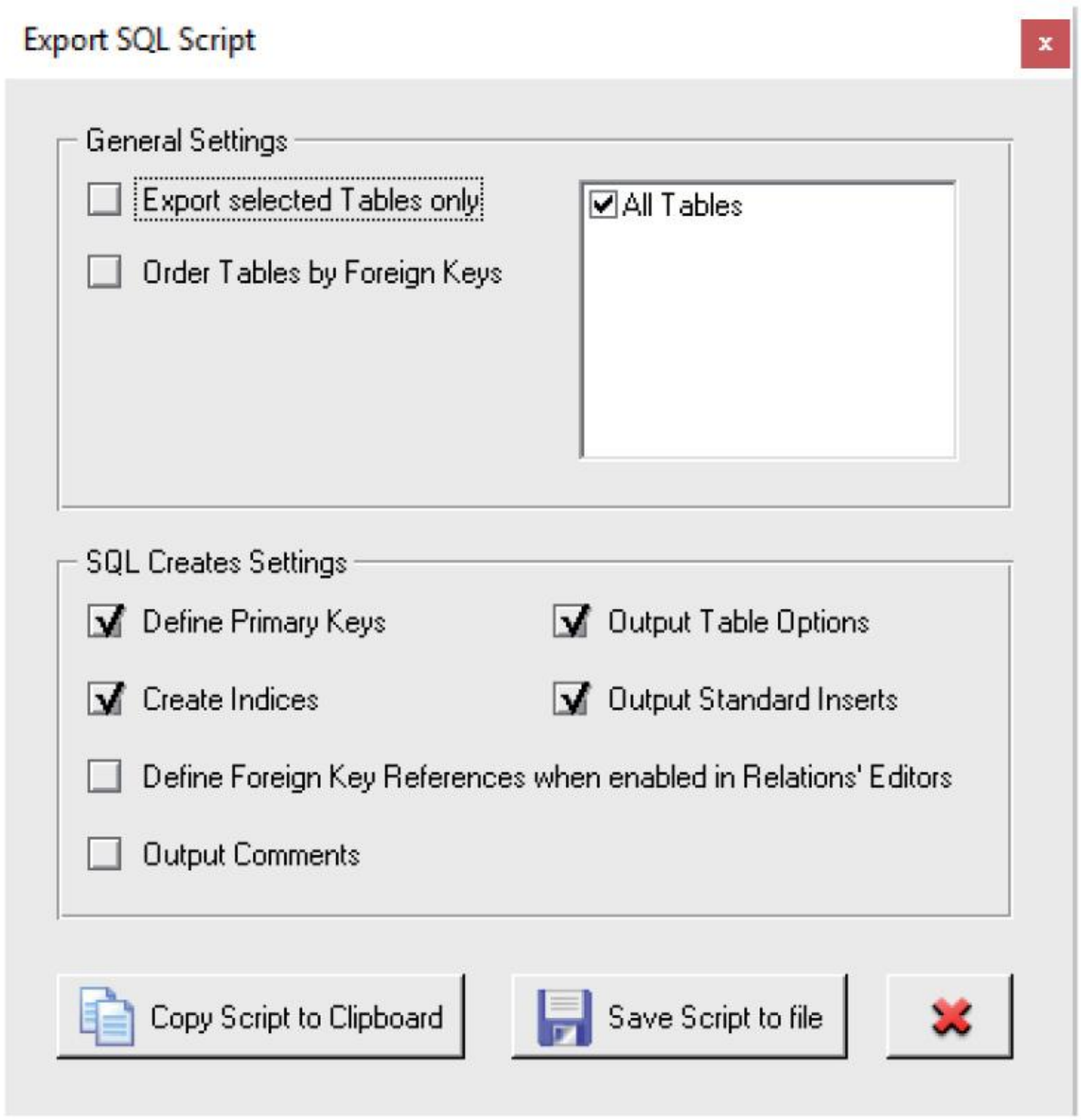
Tworzone tabele naszego projektu bazy danych można grupować w **obszary** (**regiony**). Pozwala to na utworzenie czytelniejszego projektu struktury bazy danych.

Między tabelami zachodzą określone relacje. Definiuje się je dzięki wybraniu właściwego rodzaju relacji z paska narzędziowego i wskazaniu dwóch tabel, między którymi zachodzi związek.

Podczas łączenia tabel relacją program automatycznie tworzy klucze obce. Edytor relacji uruchamia się po podwójnym naciśnięciu lewego klawisza myszy w linię reprezentującą relację. Edytor relacji np. umożliwia zmianę nazwy relacji oraz nazwy klucza obcego.

Po zakończeniu etapu modelowania można wygenerować kod w języku SQL, pozwalający na implementację fizycznej struktury tabel. W tym celu należy skorzystać z menu

File / Export / SQL Create Script oraz określić parametry eksportu w oknie pokazanym na rys. 3.16.



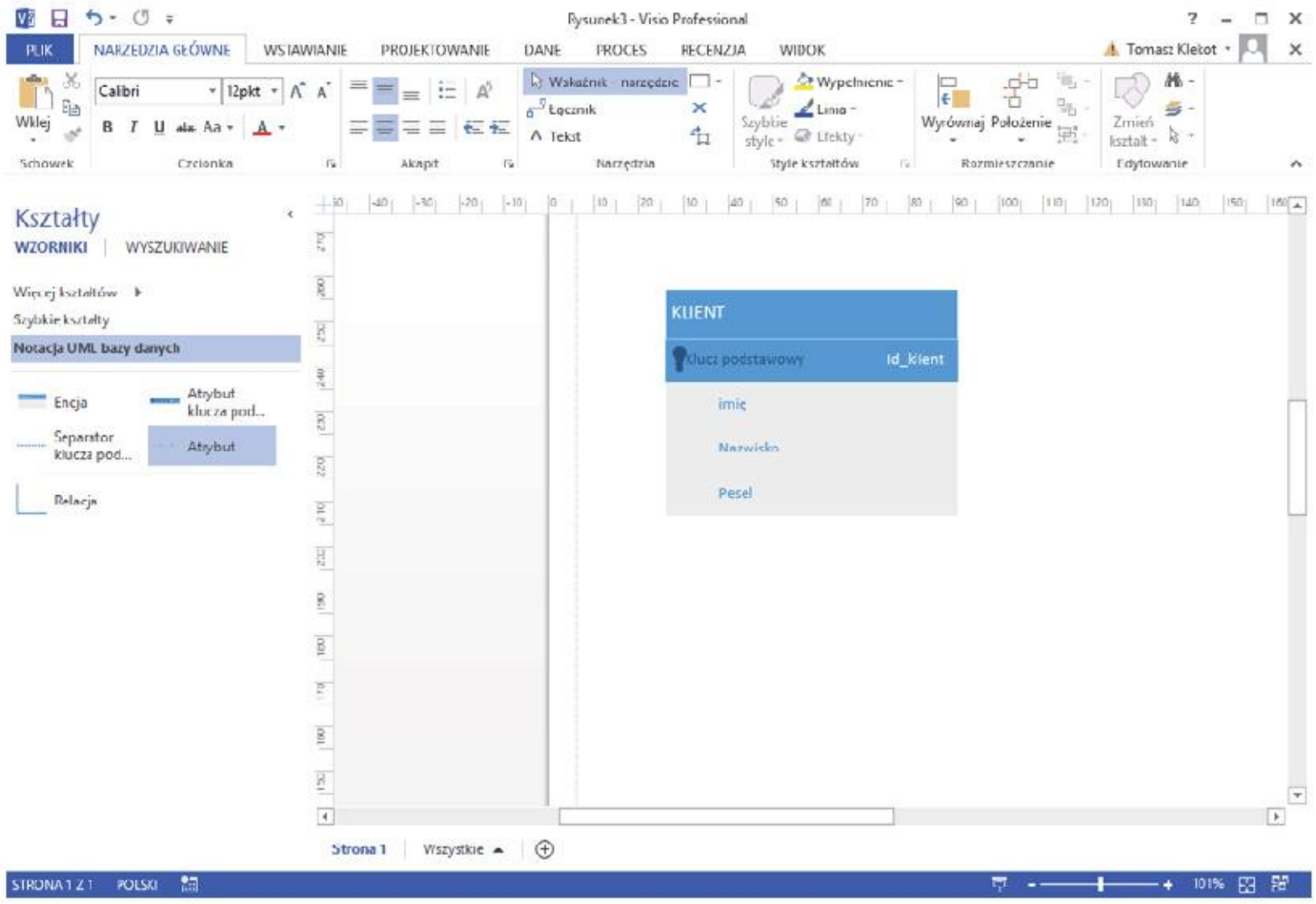
Rys. 3.16. Okno eksportu modelu bazy danych do pliku SQL

Gotowy projekt bazy danych można również zapisać jako plik XML oraz grafikę.

MS Visio

Program **MS Visio** jest składową pakietu Office. Aplikacja umożliwia tworzenie diagramów, wykresów i schematów blokowych z różnych dziedzin. Można tworzyć m.in. diagramy algorytmów czy modele struktur baz danych w notacji UML. Okno programu MS Visio pokazano na rys. 3.17.

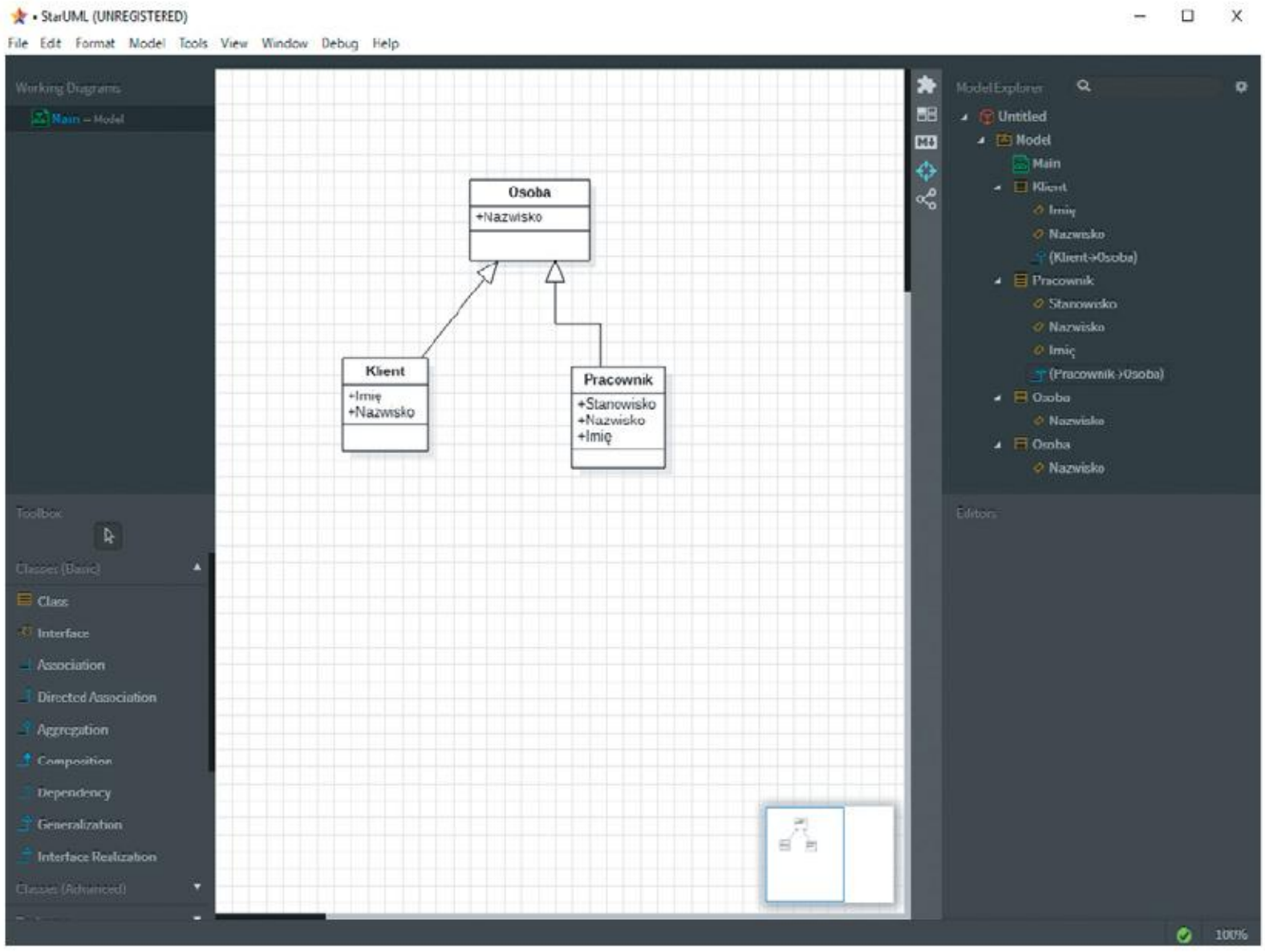
Niestety, program MS Visio jest płatny i stanowi poszerzenie pakietu MS Office. Można jednak wykorzystać wersję próbną. Program jest również dostępny w usłudze Azure for Education dla uczniów technikum informatycznego.



Rys. 3.17. Okno programu MS Visio

StarUML

StarUML jest aplikacją wspierającą język graficznego modelowania UML. To narzędzie multiplatformowe, które opiera się na wersji UML 2.0. Dostarcza użytkownikowi kilkanaście różnych diagramów z wybranymi elementami notacji UML 2.0. Aplikację w wersji testowej można pobrać ze strony http://staruml.io.



Rys. 3.18. Okno programu StarUML

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co powinien przedstawiać diagram ER?
2. Jakie funkcje pełnią aplikacje typu CASE?
3. Czym różni się struktura logiczna (konceptualna) od struktury fizycznej bazy danych?
4. Na jakie etapy można podzielić proces projektowania relacyjnych baz danych?
5. Opisz różnice między strategią TOP-DOWN a INSIDE-OUT.
6. Zaznacz, czy zdanie jest prawdziwe czy fałszywe.

W modelu ER	Prawda	Fałsz
encję określa się jako przymiotnik opisujący cechę obiektu.		
związek definiuje się jako sposób zależności jednej encji z innymi.		
liczebność związku oznacza się za pomocą linii zakończonej pustym rombem oraz wpisaną w niego cyfrą.		

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Zainstaluj na komputerze aplikację DBDesigner. Zapoznaj się z interfejsem programu oraz dokumentacją dostępną na stronie http://fabforce.eu/downloads.php.
2. Wyszukaj w zasobach sieci internet przykłady programów typu CASE, które nie zostały opisane w podręczniku. Przedstaw ich krótką charakterystykę.
3. Narysuj w zeszycie diagram modelu ER zgodnego z przykładem 3.1, który przedstawi strukturę bazy danych szkolnej biblioteki.

4 Zasady tworzenia diagramów ER

ZAGADNIENIA

- Określenie przeznaczenia przykładowej bazy danych
- Proces definiowania encji i relacji
- Modelowanie przykładowej struktury bazy danych w programie DBDesigner4
- Implementacja skryptu SQL

Aby zrozumieć poszczególne etapy projektowania bazy danych, trzeba przeanalizować przykład pokazujący, w jaki sposób powinno się przygotowywać model bazy danych przed implementacją do postaci struktury określonego SZBD.

Określenie przeznaczenia bazy danych

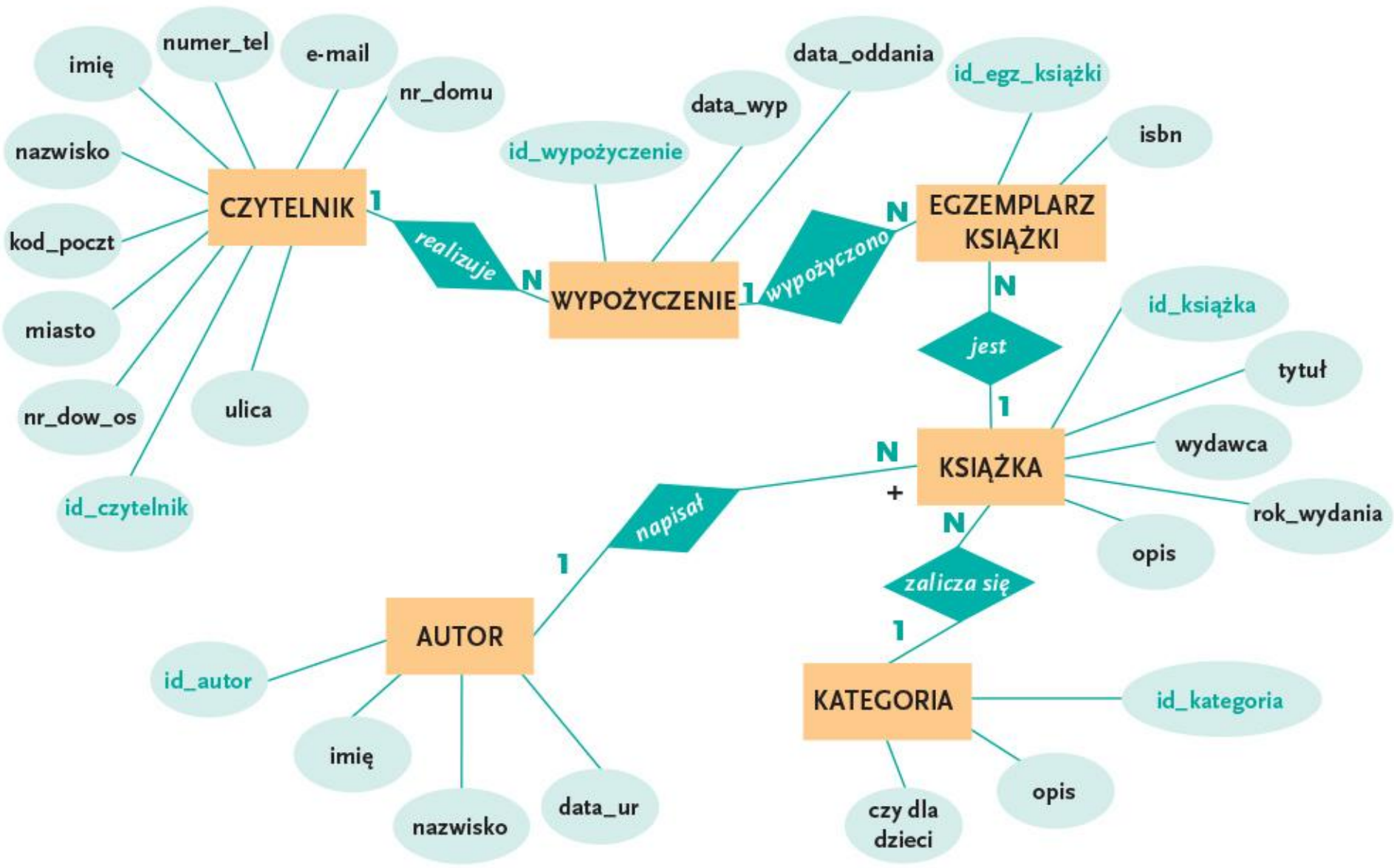
Elementem koniecznym do utworzenia całego modelu bazy danych dla określonego środowiska obiegu informacji jest zdefiniowanie jej przeznaczenia. W danym zbiorze obiektów i opisujących ich cech oraz strukturze zależności między nimi system informatyczny może dotyczyć tylko pewnej części z nich. Jak obszerny ma to być zbiór, decyduje zleceniodawca projektu. To on określa wytyczne, które będą definiowały przeznaczenie i funkcjonalność projektu i przyszłej fizycznej postaci bazy danych.

Załóżmy, że zlecono nam wykonanie projektu bazy danych osiedlowej biblioteki. Zleceniodawca, w tym wypadku biblioteka, określił wymagania dotyczące przeznaczenia bazy danych i elementów decydujących o jej funkcjonalności. Przekazane wytyczne, poddane szczegółowej analizie, definiują zestaw następujących cech:

- Baza danych ma zbierać informacje o książkach wypożyczanych w bibliotece. Te informacje muszą wystarczyć do wykonania inwentaryzacji według przepisów wewnętrznych.
- Biblioteka ma po kilka egzemplarzy książek i należy umożliwić ewidencjonowanie wypożyczeń każdego egzemplarza.
- Baza danych powinna zbierać dane o czytelnikach, ale tylko te, które umożliwią jego identyfikację oraz kontakt z nim.
- Czytelnik lub pracownik biblioteki powinni mieć możliwość wyszukiwania książek określonego autora według kategorii oraz szybkiego wyszukania książek dla dzieci.
- Najważniejszą funkcją bazy danych ma być ewidencjonowanie każdego wypożyczenia z określeniem daty wypożyczenia, oddania, autora książki, tytułu książki i identyfikatora egzemplarza oraz danych czytelnika.

Zdefiniowanie encji i ich atrybutów

Po analizie obiektów systemu informacyjnego, który istnieje w obszarze biblioteki, należy wyodrębnić i zdefiniować encje – opisać ich cechy odpowiednimi atrybutami. Wykonamy to za pomocą diagramu w notacji Chena (rys. 4.1).



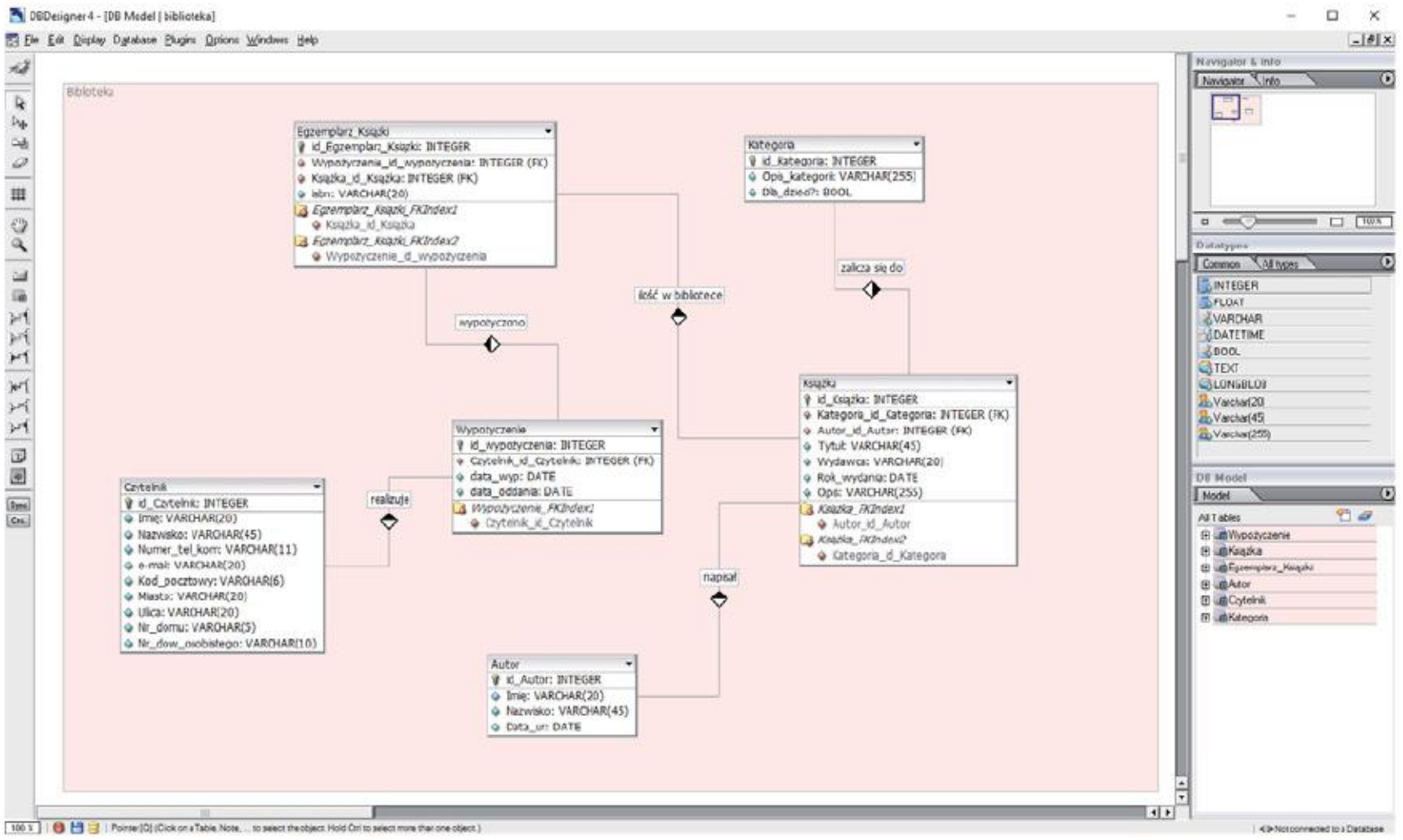
Rys. 4.1. Przykład diagramu ER w notacji Chena

Modelowanie struktury w programie typu CASE

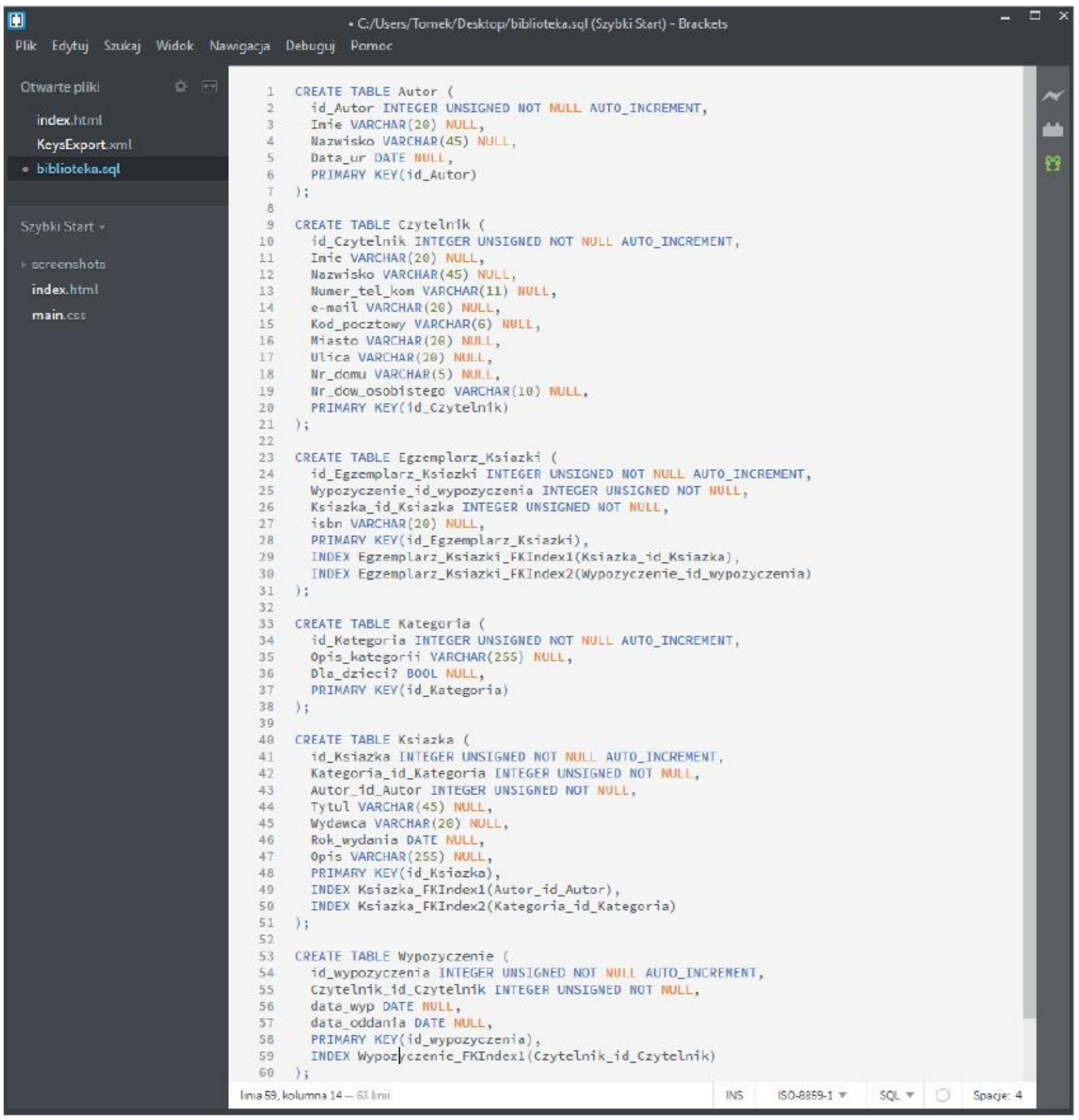
Po wizualizacji analizy rzeczywistości, jaką daje nam diagram ER, możemy przystąpić do utworzenia modelu bazy danych w programie DBDesigner4. Podczas tworzenia tabel zostaną zdefiniowane typy danych oraz klucze obce. Gotowy model struktury przedstawiono na rys. 4.2.

Implementacja w postaci kodu skryptu SQL

Po wykonaniu projektu struktury tabel i powiazań między nimi w programie DBDesigner4 możemy implementować skrypt pozwalający utworzyć strukturę fizyczną na dowolnym serwerze SQL. Przykład kodu wygenerowanego w programie DBDesigner4 przedstawiono na rys. 4.3.

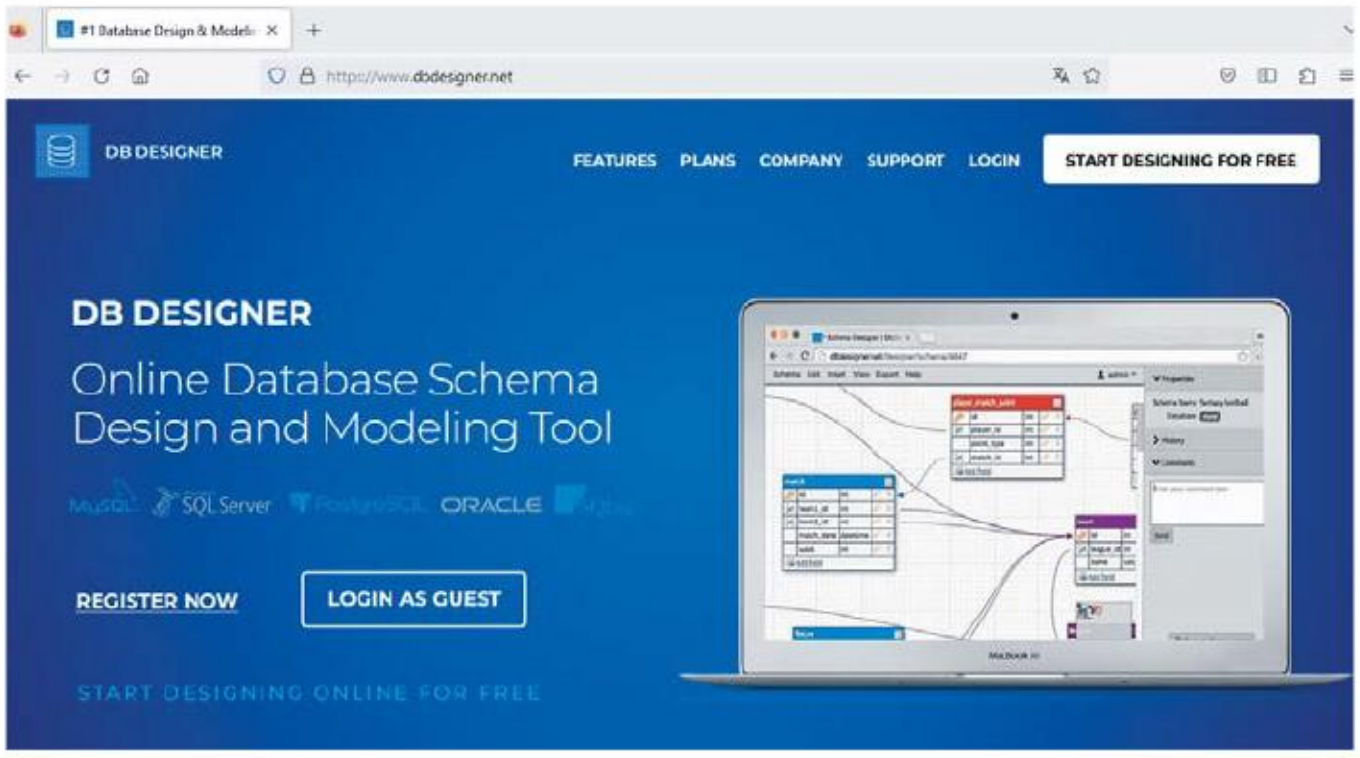


Rys. 4.2. Przykład modelu struktury bazy danych w programie DBDesigner4



Rys. 4.3. Wygenerowany skrypt SQL

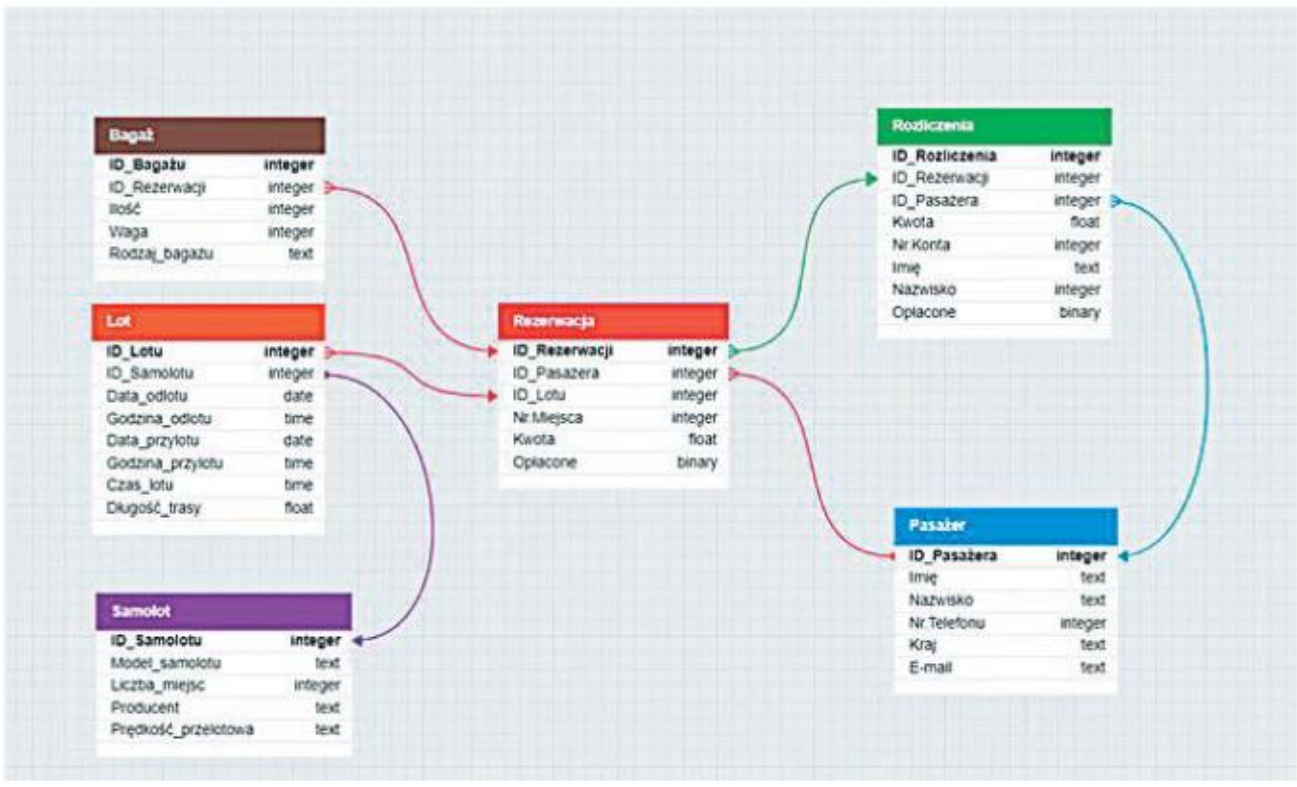
Oprócz programów desktopowych istnieją zaawansowane rozwiązania do projektowania schematów baz danych online. Jednym z rozwiązań jest platforma https://www.dbdesigner.net. Pozwala ona na tworzenie diagramu ER od początku lub zaimportowanie już istniejącej struktury bazy danych.



Rys. 4.4. DB Designer

Po zakończeniu projektowania użytkownik może wygenerować skrypt SQL dla wybranego z dostępnych silników baz danych, takich jak MySQL, MSSqI, PostgreSQL, Oracle czy SQLite.

Na rys. 4.5 przedstawiono przykład diagramu wykonanego w aplikacji internetowej DBDesigner.



Rys. 4.5. Schemat diagramu ER wykonany w aplikacji DBDesigner

UWAGA

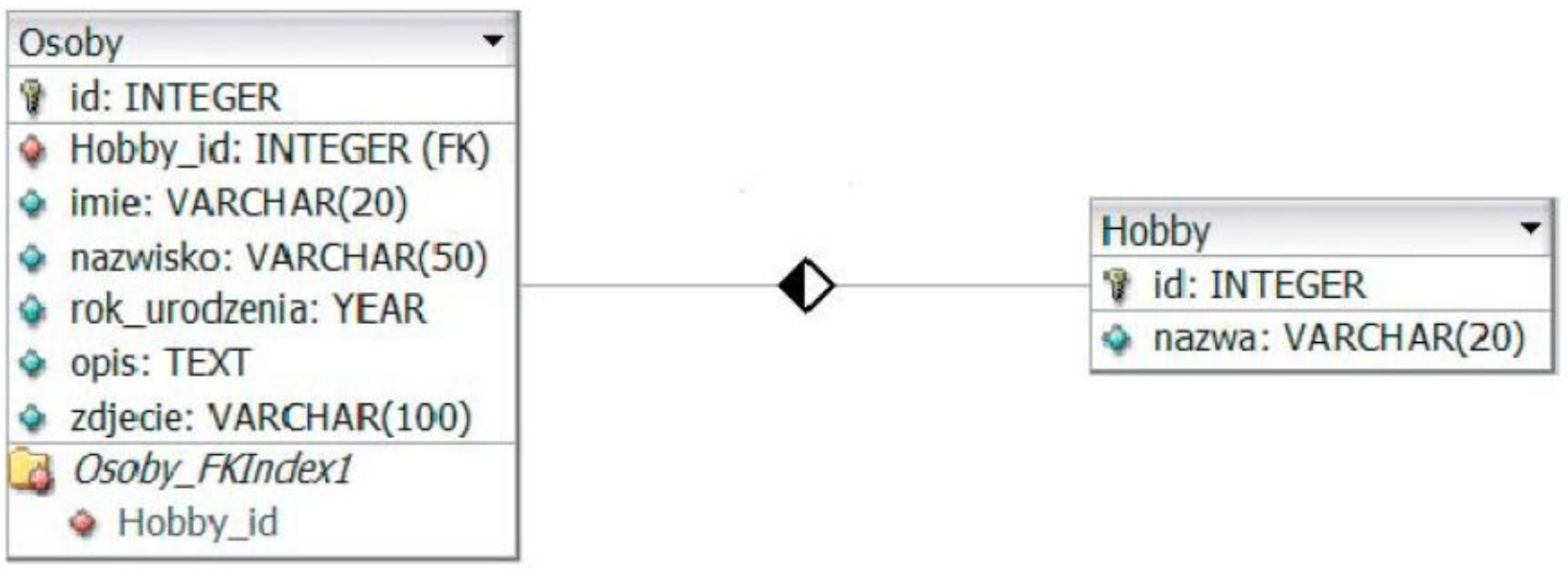
W celu lepszego zrozumienia przykładu w modelowaniu diagramów i tworzeniu tabel w DBDesigner4 użyto w nazwach polskich znaków. Niestety, nie są one zawsze właściwie kodowane w trakcie procesu implementacji skryptu SQL – dlatego lepiej od samego początku nie stosować polskich nazw i opisów w strukturze ERD.

SPRAWDŹ SWOJĄ WIEDZĘ

1. W jakiej notacji wykonano diagram z rys. 4.1?
2. Co oznacza skrót ERD?
3. Wymień etapy tworzenia diagramu modelu ER dowolnej bazy danych.
4. W jakim celu implementuje się model struktury bazy danych do pliku sql?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Zdefiniuj cechy encji Klient w dwóch wariantach:
   a) dla zakładu fryzjerskiego,
   b) dla internetowej apteki sprzedającej również leki refundowane przez NFZ.
2. Zaprojektuj diagram ER bazy danych stacji benzynowej w notacji Chena.
3. Zaprojektuj model ERD struktury bazy danych szkolnego dziennika ocen uczniów z przedmiotów ogólnokształcących i zawodowych. Zdefiniuj niezbędne encje i atrybuty. Przyporządkuj wszystkim atrybutom właściwe typy danych. Model wykonaj w programie DBDesigner4 lub z wykorzystaniem innego edytora typu CASE. Gotowy projekt przedstaw nauczycielowi do oceny.
4. Zaprojektuj model ERD struktury bazy danych do zarządzania szafkami uczniów. Model wykonaj w dowolnym programie typu CASE. W strukturze mogą wystąpić wyłącznie cztery encje: Uczeń, Klasa, Szafka i Kaucja. Zdefiniuj dla każdej encji atrybuty oraz przyporządkuj im właściwe typy danych. Gotowy projekt przedstaw nauczycielowi do oceny.
5. Wykonaj interpretację rysunku przedstawiającego strukturę bazy danych. Opisz nazwy tabel, liczność relacji oraz scharakteryzuj cechy poszczególnych atrybutów.



III. Systemy zarządzania bazami danych

- Rodzaje SZBD
- Instalacja i konfiguracja MS Access
- Zarządzanie bazą danych w pakiecie XAMPP
- Microsoft SQL Server

5 Rodzaje systemów zarządzania bazami danych

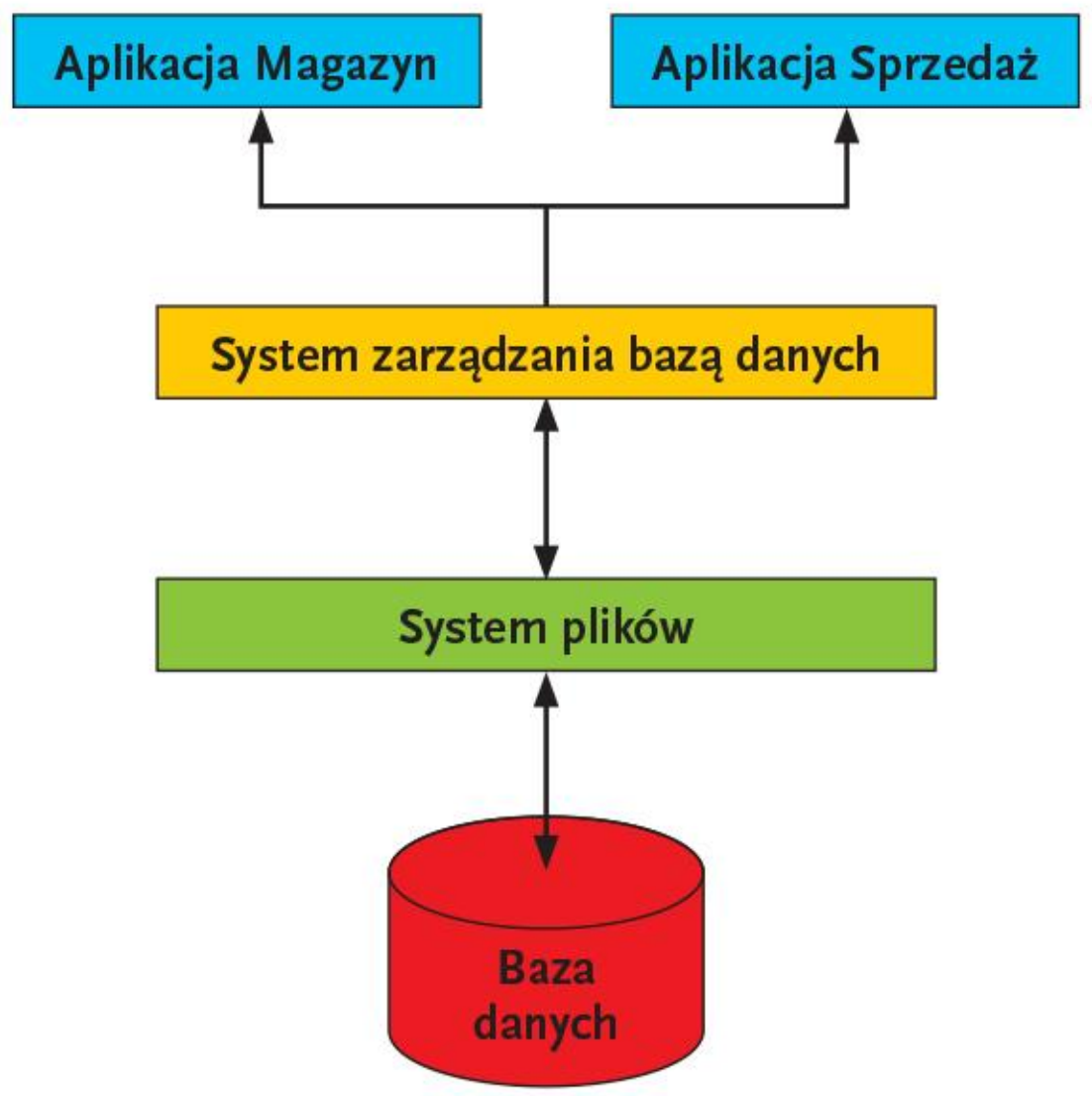
ZAGADNIENIA

- Rodzaje systemów zarządzania bazami danych
- Dobór SZBD do określonych zastosowań
- Zarządzanie SZBD

System zarządzania bazami danych – SZBD (ang. *Database Management System*, DBMS) – to system aplikacji i narzędzi do zarządzania bazami danych. Jak pisaliśmy w pierwszym rozdziale, SZBD może pracować lokalnie, sieciowo w architekturze klient–serwer oraz w chmurze obliczeniowej.

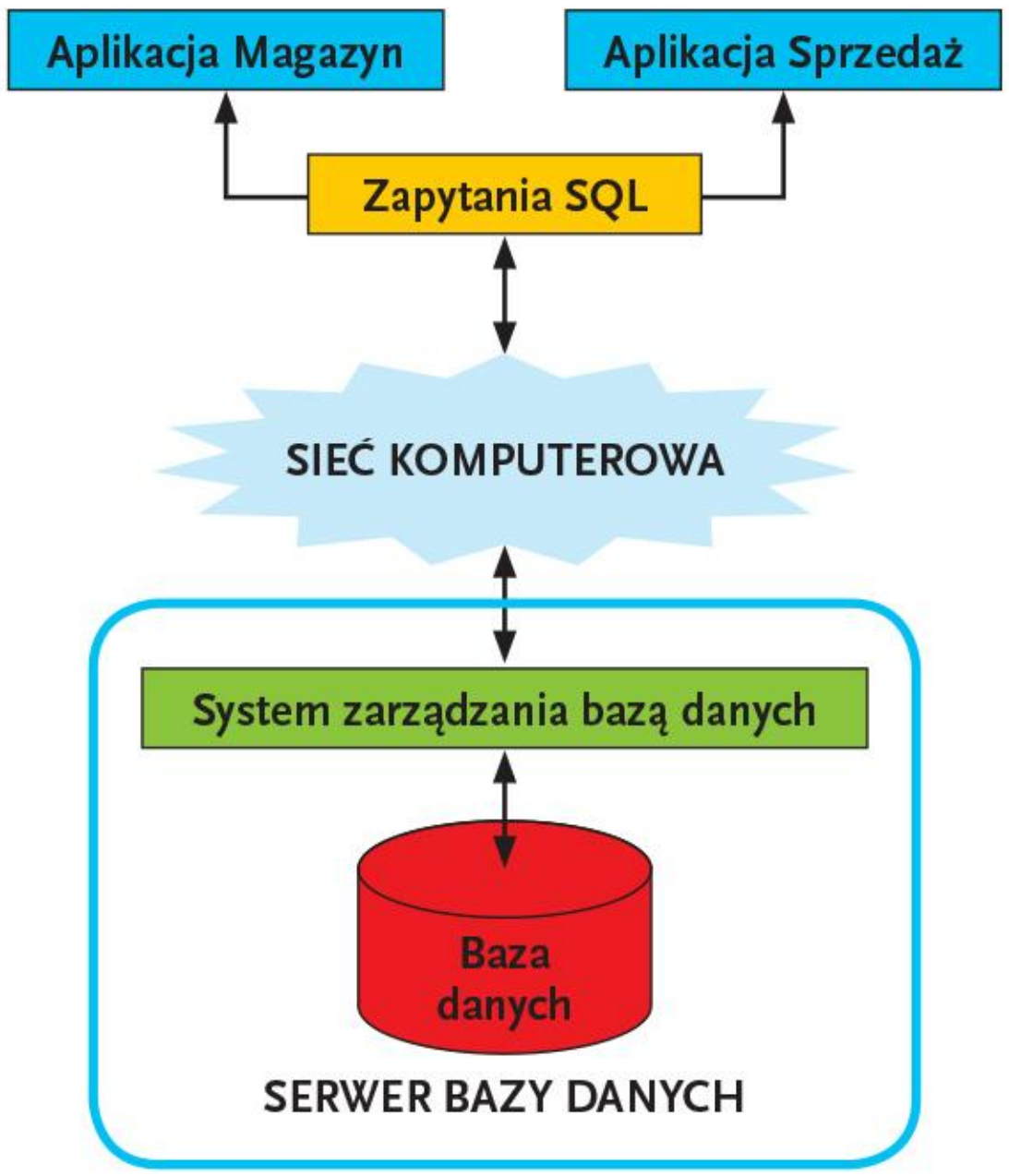
Ze względu na sposób dostępu do baz danych rozróżnia się:

- SZBD mające dostęp do danych przez wbudowane w aplikację biblioteki (system plikowy). Spotykane przede wszystkim w lokalnych systemach baz danych. Wszystkie operacje na danych wykonuje się na lokalnym komputerze, dlatego wydajność i skalowalność są ograniczone jego możliwościami technicznymi. Przykładami SZBD tego typu są MS Access oraz Kexi. Schemat działania tego rodzaju SZBD pokazano na rys. 5.1;



Rys. 5.1. System plikowy SZBD

- SZBD mające dostęp do danych przez komunikację z serwerem bazy danych (system serwerowy). Ten rodzaj dostępu do danych występuje w strukturach sieciowych klient–serwer oraz chmurowych. Charakterystyczne dla niego jest przesyłanie przez sieć tylko zapytań i ich wyników. Zwiększa to wydajność pracy na bazie danych. Dzięki zastosowaniu architektury klient–serwer jest możliwa bezproblemowa skalowalność całej struktury systemu bazy danych. Przykładami SZBD tego typu są: Firebird, MSSQL, MySQL, MariaDB, PostgreSQL i Oracle. Schemat działania tego rodzaju SZBD pokazano na rys. 5.2.



Rys. 5.2. System sieciowy SZBD

Dobór SZBD do określonych zastosowań

Do najważniejszych kryteriów doboru SZBD należą:

- środowisko i wymagania techniczne – SZBD musi być zgodny z systemami operacyjnymi i standardami sieciowymi funkcjonującymi w organizacji lub firmie;
- liczba składowanych i przetwarzanych danych – w tym miejscu należy się również zastanowić nad skalowalnością SZBD, czyli rozważyć, jak zmieni się działanie systemu (jego wydajność), jeśli wzrośnie liczba użytkowników lub danych;
- liczba użytkowników jednorazowo korzystających z bazy danych;
- rozproszenie klientów bazy danych – inaczej trzeba rozpatrywać przypadek systemu baz danych, który działa na jednym komputerze w szkolnej księgowości, niż system złożony z kilku tysięcy stacji roboczych koncernu prowadzącego działalność na dwóch kontynentach;

- bezpieczeństwo danych – ważne jest, jakie narzędzia oferuje SZBD w zakresie zarządzania użytkownikami, wykonywania kopii zapasowych i zabezpieczeń przed atakami hakerskimi;
- funkcjonalność, czyli ogół funkcji dostępnych w SZBD – warto zwrócić uwagę zarówno na funkcje wykorzystywane przez użytkownika, jak i administratora bądź projektanta takiego systemu;
- wspomaganie, które oznacza zapewnienie odpowiedniej pomocy przez dostawcę systemu – niestety często jest to dodatkowo płatna usługa;
- cena – nie tylko koszt zakupu SZBD, lecz także wszystkie pozostałe koszty związane z wdrożeniem tego systemu oraz przewidywanymi kosztami jego eksploatacji; koszt zakupu jest często tylko elementem składowym ogólnej ceny systemu.

Zarządzanie SZBD

To ważna funkcja w każdym systemie bazy danych. Sprowadza się do nadzorowania pracy sprzętowych elementów systemu oraz do realizowania zadań administracyjnych względem oprogramowania SZBD. Do najważniejszych zadań administratora SZBD należy zaliczyć:

- instalację, konfigurację oraz aktualizację systemu;
- zarządzanie systemami komputerowymi i infrastrukturą sieciową oraz nadzór nad ich prawidłowym funkcjonowaniem;
- zarządzanie użytkownikami i ich uprawnieniami do SZBD oraz samej bazy danych;
- wykonywanie systemu kopii zapasowych i odtwarzanie bazy z kopii w razie awarii;
- optymalizację aplikacji do wymagań użytkowników końcowych;
- zabezpieczanie dostępu do danych oraz zdefiniowanie procedur bezpieczeństwa dla użytkowników;
- porządkowanie, normalizowanie i aktualizowanie danych;
- planowanie i organizowanie zgodnie z harmonogramem prac konserwacyjnych;
- wybór metod pomiarowych i narzędzi testujących sprawność działania programów i sprzętu komputerowego.

Użytkownicy baz danych i ich rola w SZBD

Wśród najważniejszych grup użytkowników SZBD można wyróżnić:

- administratora systemu (ang. *system administrator*) – jego zadaniem jest nadzór nad całym systemem zarządzania bazami danych; definiuje on bazy danych w systemie oraz zakłada użytkowników;
- administratora bazy danych (ang. *database administrator*) – administruje jedynie bazą lub bazami, które ma pod swoją opieką; przysługują mu uprawnienia do zakładania i administrowania obiektami bazy, ale tylko w obrębie danych bazy danych; wykonuje również kopie bezpieczeństwa;
- programistę aplikacji (ang. *application programmer*) – jego rolą jest pisanie kodu, który będzie wspomagał użytkowanie bazy (np. makropoleceń lub aplikacji klienckich);
- użytkownika standardowego (ang. *user*) – wykonuje czynności na danych w bazie danych.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jakie grupy użytkowników występują najczęściej w systemach zarządzania bazami danych?
2. Jakie kryteria warunkują wybór określonego systemu zarządzania bazami danych?
3. Opisz strukturę sieciowego SZBD opartego na architekturze klient–serwer.
4. Zaproponuj rodzaj SZBD do utworzenia systemu baz danych w szkolnej księgowości.
5. Miarą złożoności systemów informatycznych jest liczba punktów funkcyjnych. Jakie systemy zarządzania bazami danych będą najlepsze do obsługi baz danych w systemach informatycznych wysokiej złożoności?

6 Instalacja i konfiguracja MS Access

ZAGADNIENIA

- Funkcje i budowa programu MS Access
- Instalacja MS Access
- Zarządzanie MS Access

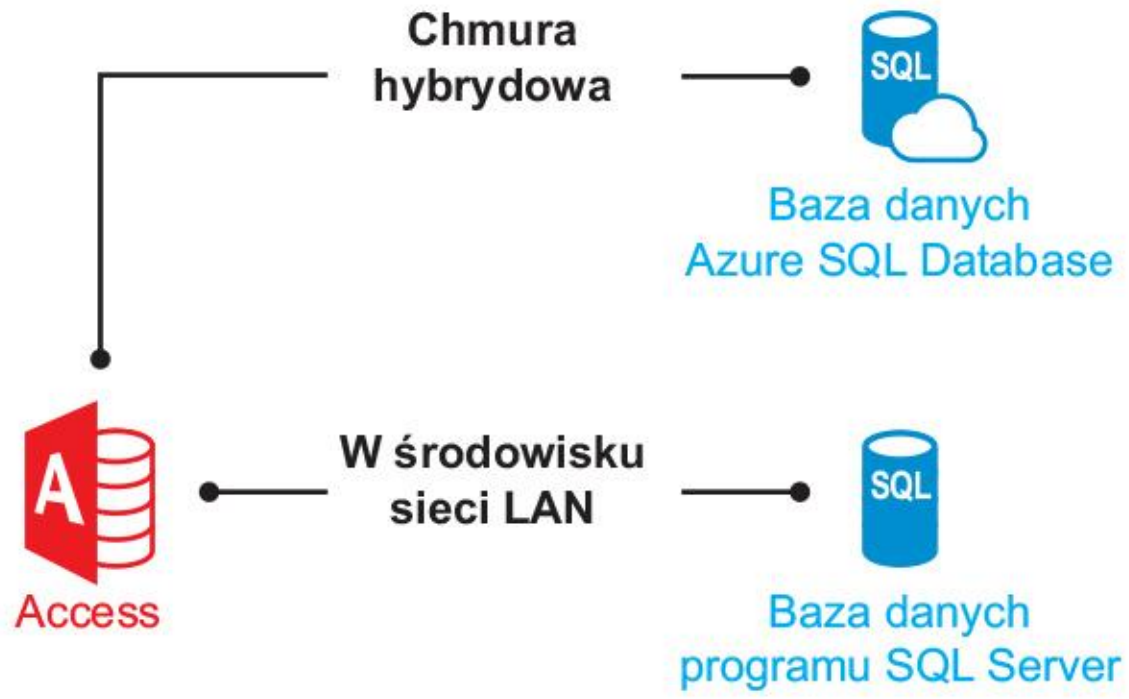
Microsoft Access to plikowy system zarządzania relacyjnymi bazami danych. Program ten jest przeznaczony przede wszystkim dla środowiska Windows. Od niedawna jest też dostępna wersja dla użytkowników MAC OS. Program Access pozwala na zarządzanie wieloma bazami danych, które zapisuje się w oddzielnych plikach. W MS Access można również pracować na zewnętrznych bazach danych oraz współpracować z innymi aplikacjami firmy Microsoft (np. MS Excel, SharePoint czy MS Outlook).

MS Access można stosować do:

- tworzenia lokalnych baz danych;
- lokalnego SZBD do zbierania danych i prezentacji danych, które są przechowywane w dużych sieciowych bazach SQL;
- tworzenia baz danych udostępnianych w lokalnych sieciach komputerowych za pomocą serwera IIS (ang. *Internet Information Server*);
- analizy dużej liczby danych;
- tworzenia aplikacji baz danych dostępnych z poziomu witryny SharePoint.

Baz danych MS Access nie powinno się wykorzystywać w sytuacji, gdy zależy nam na wydajnych systemach, są wymagane zaawansowane zasady poufności danych oraz gdy baza danych ma pracować w sieci rozległej. Każda baza danych programu MS Access ma ograniczenie rozmiaru wynoszące 2 GB i nie może obsługiwać więcej niż 255 użytkowników jednocześnie. Bazy danych MS Access interpretują język SQL, dlatego w razie braku możliwości rozwoju na lokalnym SZBD można przeprowadzić migrację bazy na sieciową lub chmurową wersje SZBD. Taką sytuację pokazano na rys. 6.1.

Bazy danych w programie MS Access zapisuje się w pojedynczych plikach z rozszerzeniem **mdb** lub **accdb** (od wersji Access 2007), w których są zawarte obiekty odpowiedzialne za przechowywanie danych, operacje na danych, prezentacje danych i szablony wydruków.



Rys. 6.1. Przykład migracji bazy danych MS Access

Do podstawowych obiektów MS Access zalicza się:

- **tabele**, które służą do przechowywania danych o określonym typie i formacie; tabele są najczęściej fizyczną notacją encji, zdefiniowanej w procesie projektowania bazy danych;
- **relacje**, które określają, jakie są powiązania między poszczególnymi tabelami; za pomocą odpowiednio zaprojektowanych tabel i relacji następuje odwzorowanie rzeczywistości w strukturze bazy danych;
- **kwerendy**, które pozwalają na wybieranie określonych danych z jednej lub kilku tabel;
- **formularze**, które tworzą interfejs między końcowym użytkownikiem bazy danych a jej strukturą tabel i kwerend; służą do wpisywania i edycji danych;
- **raporty**, które pozwalają na tworzenie szablonów wydruków; raporty są wyświetlane również na ekranie monitora lub można je dołączyć jako załącznik do poczty elektronicznej;
- **makra i moduły**, które pozwalają na automatyzację prac; różnica między nimi polega na sposobie ich tworzenia: makra to ciągi predefiniowanych czynności zawartych na liście, a moduły zawierają funkcje napisane w języku programowania VBA (ang. *Visual Basic for Applications*).

Utworzenie pliku bazy danych w formacie MS Access wymaga zainstalowania aplikacji dostępnej w wersji na komputery PC. Program MS Access jest elementem składowym MS Office, ale nie występuje we wszystkich wersjach tego pakietu. Obecnie umieszcza się go w następujących produktach i usługach firmy Microsoft:

- Office 365 dla Użytkowników Domowych,
- Office 365 Personal,
- Office 365 Business,
- Office 365 Business Premium,
- Platforma Azure for Education dla wyższych uczelni i szkół kształcących w zawodzie technik informatyk i technik programista.

Niestety, wszystkie wymienione pakiety są płatne. Do nauki można wykorzystać bezpłatną wersję próbną dostępną na stronie https://products.office.com/pl-pl/access.

Jeżeli jednak baza danych jest już utworzona i zapisana w pliku, użytkownik niemający pełnej wersji MS Access może ją uruchomić i użytkować w tzw. środowisku uruchomieniowym (ang. *Microsoft Access 2013 Runtime*). W środowisku uruchomieniowym programu MSAccess wszystkie związane z projektowaniem elementy interfejsu użytkownika są usunięte lub wyłączone. Środowisko uruchomieniowe można dowolnie redystrybuować i udostępniać użytkownikom w postaci łącza do strony pobierania: https://www.microsoft.com/pl-pl/download/details.aspx?id=39358. Dla programu MS Access 2016 środowisko uruchomieniowe jest dostępne pod adresem https://www.microsoft.com/pl-pl/download/details.aspx?id=50040.

Jedynie aplikacje oparte na sieci Web dla programu SharePoint utworzone przy użyciu programu MS Access nie wymagają środowiska uruchomieniowego.

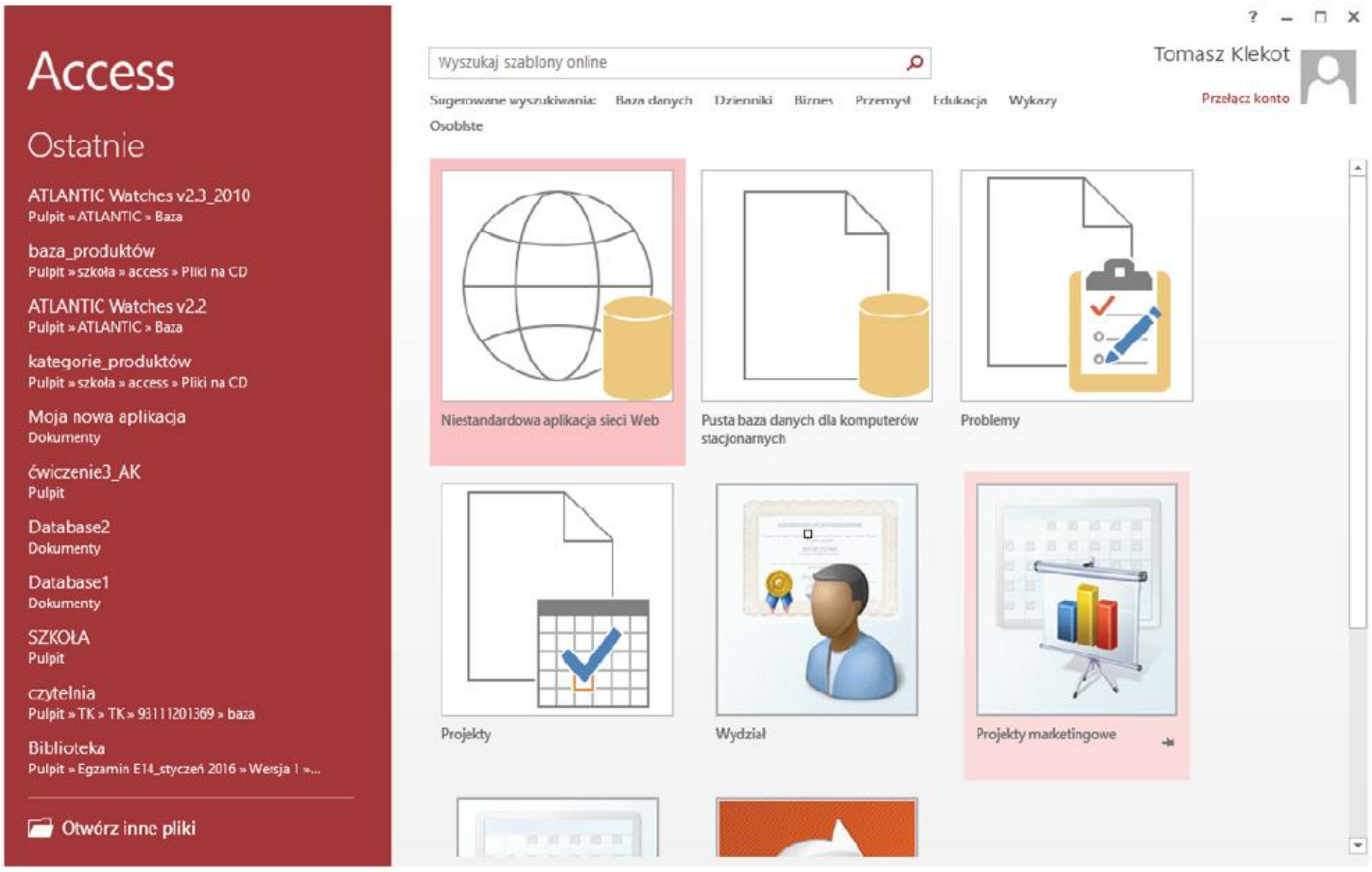
Instalacja zarówno pełnej wersji programu, jak i okrojonego środowiska uruchomieniowego jest intuicyjna i powinna przebiegać bezproblemowo. Aktywacja programu może wymagać połączenia z internetem. Należy jednak pamiętać, że przed instalacją aplikacji warto sprawdzić określone dla niej wymagania systemowe (tabela 6.1). Instalację i opis budowy przedstawiono na przykładzie wersji MS Access 2013.

Tabela 6.1. Wymagania systemowe dla MS Access

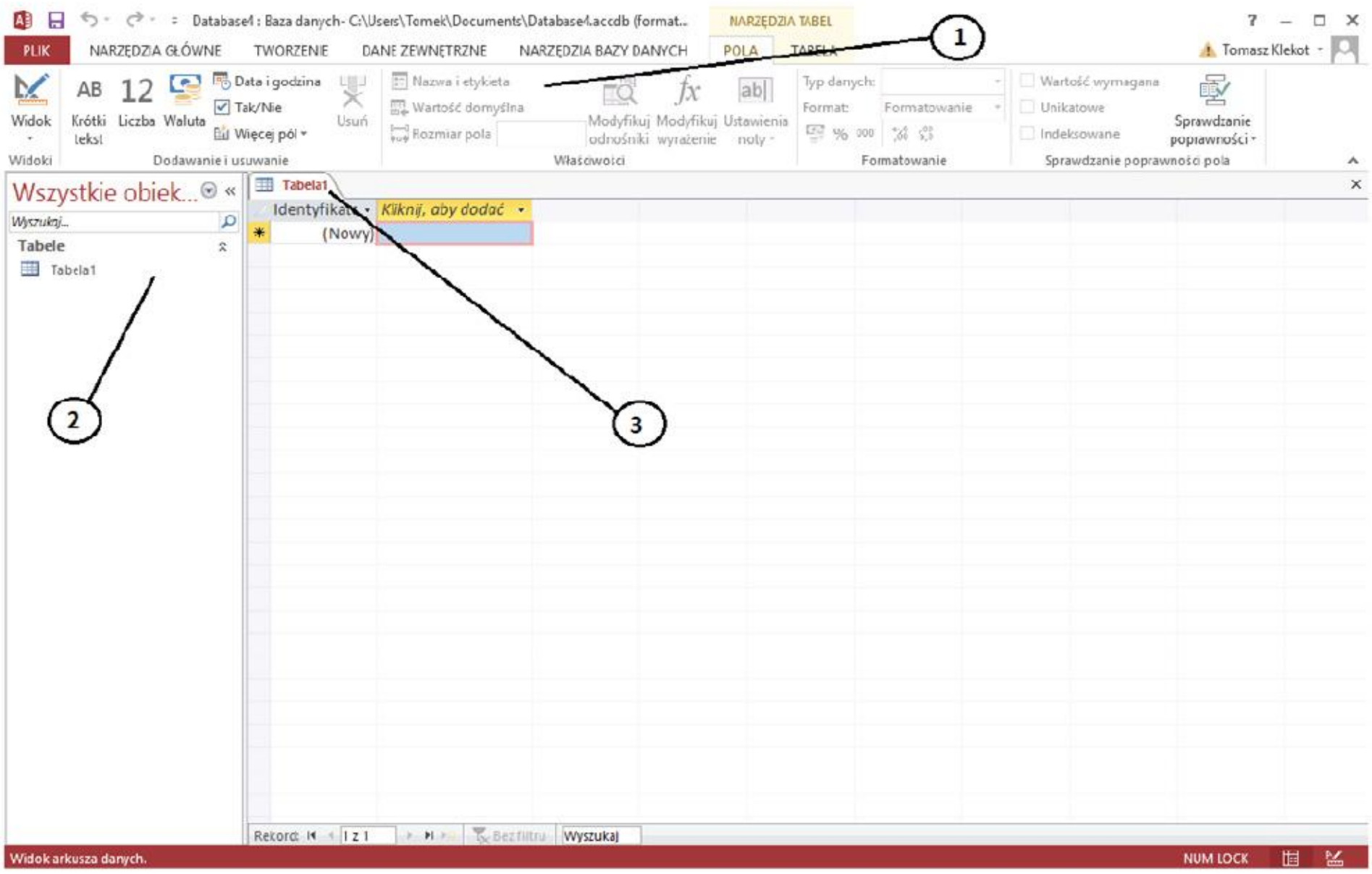
Wersja	MS Access 2013	MS Access 2019
procesor	częstotliwość co najmniej 1 GHz	częstotliwość co najmniej 1,6 GHz
pamięć RAM	2 GB	4 GB
miejsce na dysku twardym	ok. 3 GB	ok. 4 GB
system operacyjny	32-bitowy lub 64-bitowy Windows 7, 8 lub 10	64-bitowy Windows 10

Po instalacji aplikacji uruchamia się ją i wybiera typ tworzonego pliku. Jeżeli tworzy się nową bazę danych, wybiera się opcję **`Pusta baza danych dla komputerów stacjonarnych`** (rys. 6.2) oraz podaje lokalizację pliku bazy danych i jego nazwę.

Po wybraniu opcji **`Niestandardowa aplikacja sieci Web`** można utworzyć bazę danych dla usługi Office 365 z funkcją SharePoint Online. Po wyborze przycisku **`Utwórz`** otworzy się okno nowego pliku bazy danych.



Rys. 6.2. Okno tworzenia nowej bazy danych



Rys. 6.3. Okno bazy danych programu MS Access

Okno programu MS Access nawiązuje interfejsem użytkownika do pozostałych aplikacji znanych z pakietu Office. Składa się ono z trzech zasadniczych bloków (rys. 6.3):

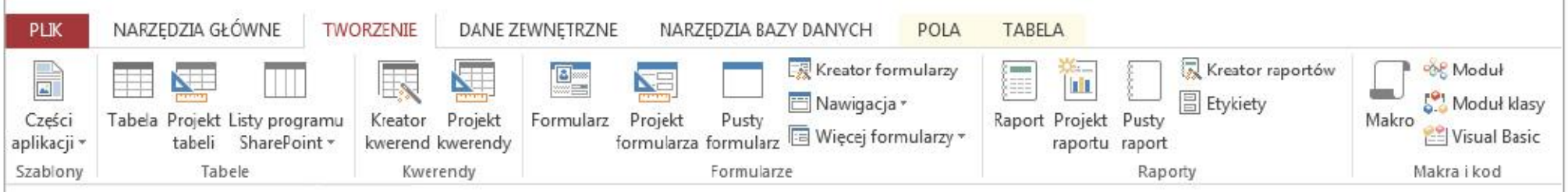
1. standardowej wstążki dla programów pakietu Office, która zawiera zakładki pasków narzędziowych programu Access;
2. okna nawigacyjnego do przeglądania i wyboru określonych obiektów otwartej bazy danych;
3. okna roboczego z zakładkami aktywnych obiektów.

Wśród zakładek pasków narzędziowych umieszczonych na wstążce znajduje się sześć pasków.

- Pasek narzędziowy zakładki **`Narzędzia główne`** (rys. 6.4). Mieści się tutaj narzędzie przełączania widoku. Do dyspozycji mamy **`Widok projektu`** oraz **`Widok arkusza danych`**. Dostępne są również opcje schowka, narzędzie filtrowania i wyszukiwania oraz formatowania tekstu.

Rys. 6.4. Pasek **`Narzędzia główne`**

- Pasek narzędziowy zakładki **`Tworzenie`** (rys. 6.5). Znajdują się tutaj wszystkie narzędzia pozwalające utworzyć podstawowe obiekty bazy danych oraz skorzystać z gotowych szablonów.

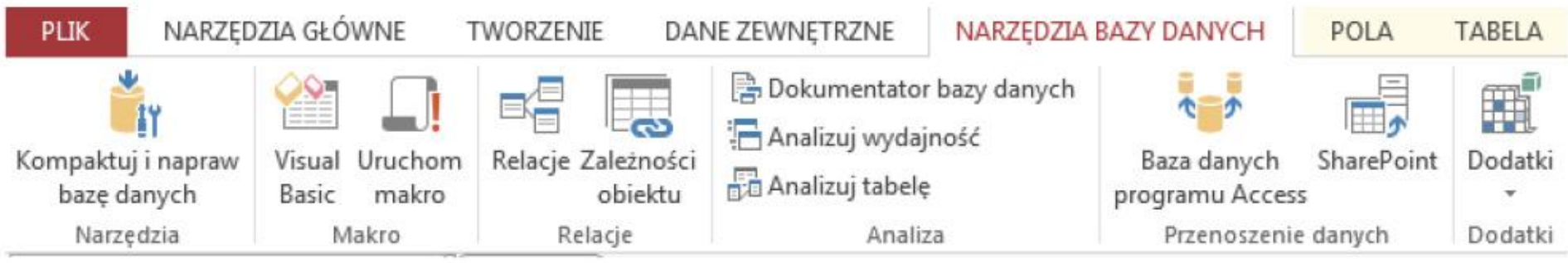


Rys. 6.5. Pasek **`Tworzenie`**

- Pasek narzędziowy zakładki **`Dane zewnętrzne`** (rys. 6.6). Na tym pasku są dostępne narzędzia pozwalające na importowanie i eksportowanie danych oraz łączenie plików zawierających dane odczytywane przez MS Access.

Rys. 6.6. Pasek **`Dane zewnętrzne`**

- Pasek narzędziowy zakładki **`Narzędzia bazy danych`** (rys. 6.7). Za pomocą narzędzi znajdujących się na tym pasku można przeprowadzać diagnostykę i naprawę plików bazy danych, definiować relacje, udostępniać w usłudze SharePoint czy tworzyć makra w języku VisualBasic.



Rys. 6.7. Pasek **Narzędzia bazy danych**

- Pasek narzędziowy zakładki **Pola** (rys. 6.8). Znajdują się tutaj wszystkie narzędzia do formatowania i sprawdzania poprawności właściwości pól bazy danych.

Rys. 6.8. Pasek **Pola**

- Pasek narzędziowy zakładki **Tabela** (rys. 6.9). Zawiera narzędzia do wykonywania wszelkich operacji na tabelach bazy danych.

Rys. 6.9. Pasek **Tabela**

Po utworzeniu pliku bazy danych program jest gotowy do budowy struktury tabel i pozostałych elementów tworzących bazę w MS Access. Podczas ich tworzenia należy pamiętać, że nazwy w bazach danych Microsoft Access muszą spełniać następujące warunki:

- mogą zawierać do 64 znaków;
- mogą zawierać dowolną kombinację liter, cyfr, spacji i znaków specjalnych z wyjątkiem kropki (.), wykrzyknika (!), akcentu słabego (`) i nawiasów kwadratowych ([ ]);
- nie mogą zaczynać się od spacji;
- nie mogą zawierać znaków kontrolnych (wartości kodu ASCII od 0 do 31);
- nie mogą zawierać cudzysłowu podwójnego („).

Ze względu na możliwość ciągłej rozbudowy i automatyzacji operacji wykonywanych na bazie danych należy korzystać z dobrych praktyk dotyczących tworzenia struktury bazy danych. Zalicza się do nich następujące zalecenia:

- nazwy tabel, kwerend oraz innych obiektów nie powinny mieć takich samych nazw;
- w nazwach obiektów oraz pól należy unikać stosowania znaków diakrytycznych;
- nazwy powinny być jednowyrazowe lub rozdzielane znakiem „_”.

Po utworzeniu nowego pliku bazy danych program jest gotowy do planowania struktury tabel i relacji między nimi oraz edycji innych obiektów tworzących system bazy danych.

Konfiguracja i zarządzanie MS Access

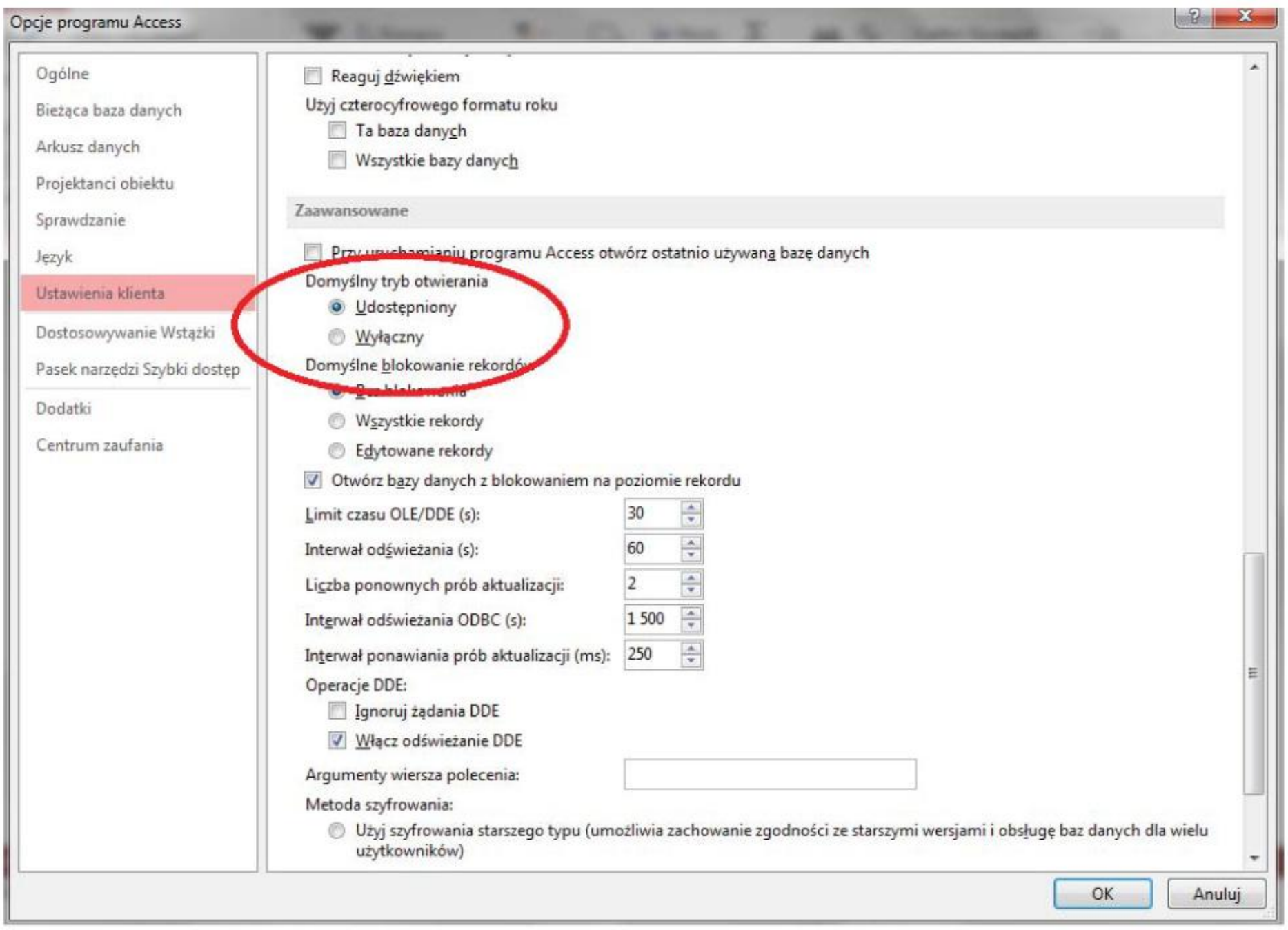
Każdy system bazy danych powinien być zarządzany i kontrolowany przez administratora. MS Access udostępnia odpowiednie narzędzia i usługi ułatwiające zarządzanie zasobami bazy danych oraz pozwalające na zabezpieczenie lub naprawę danych. Opiszemy tutaj niektóre z ważniejszych usług i narzędzi.

Ważną funkcją MS Access jest możliwość **współdzielenia bazy danych w sieci lokalnej**. Jedna z metod udostępniania bazy danych w sieci to łączenie z listami programu **SharePoint**. Ta metoda daje te same korzyści, co używanie podzielonej bazy danych. Użytkownicy mogą modyfikować własną kopię bazy danych, ponieważ dane są udostępniane za pośrednictwem witryny programu SharePoint. Dzięki umieszczeniu danych na listach programu SharePoint można oddzielnie udostępniać dane w sieci przy użyciu funkcji tego programu.

Metoda udostępniania przez SharePoint wymaga wykonania trzech podstawowych czynności:

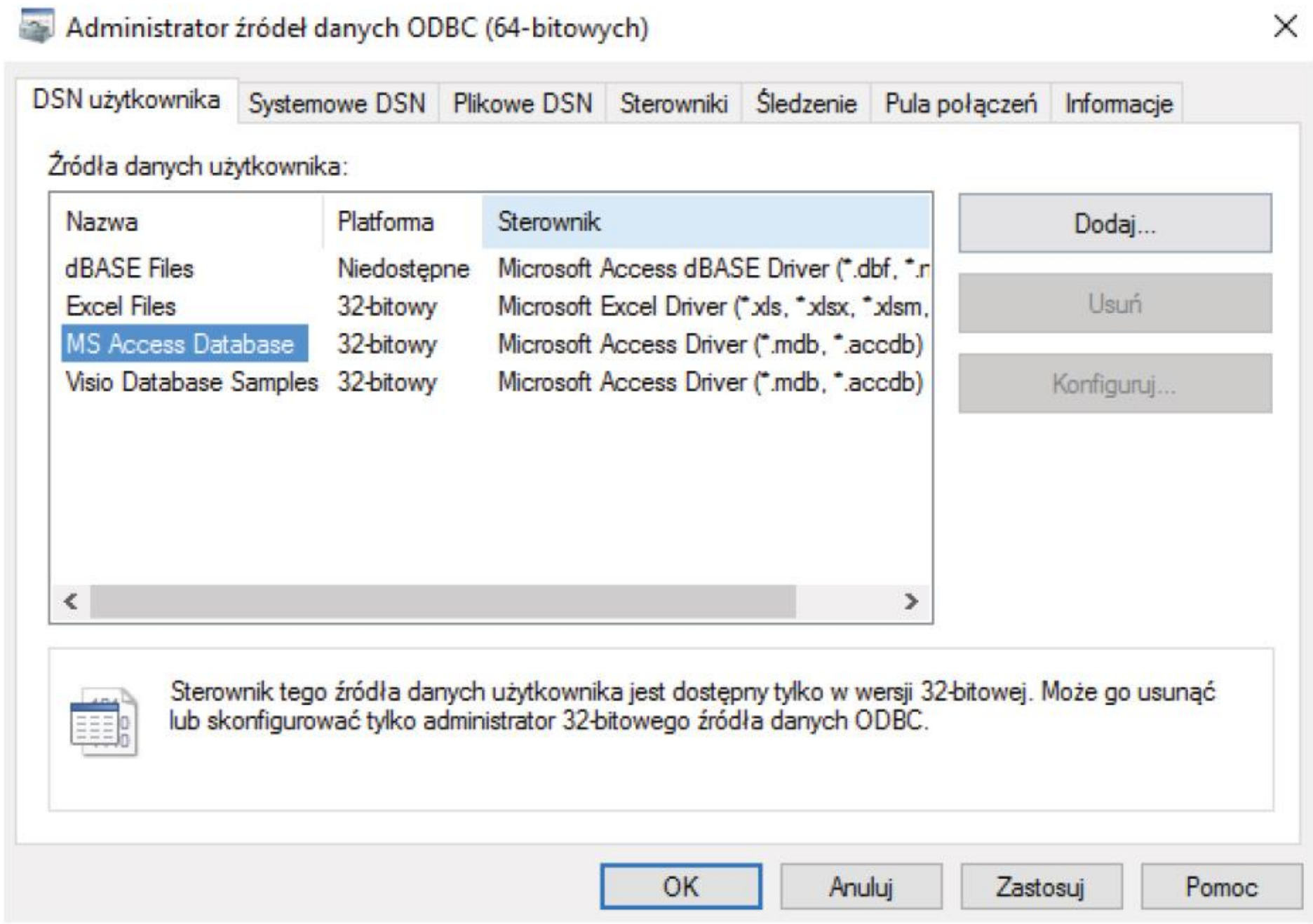
- przeniesienia danych na listy programu SharePoint;
- utworzenia łączy do tych list;
- rozpowszechnienia pliku bazy danych.

Najprostszą metodą jest jednak **udostępnianie danych z MS Access jako zasobów sieci lokalnej**. Trzeba jednak pamiętać o właściwej konfiguracji opcji programu, pozwalającej na udostępnienie możliwości otwierania danych z innych lokalizacji (rys. 6.10).



Rys. 6.10. Konfiguracja udostępniania bazy danych w sieci lokalnej

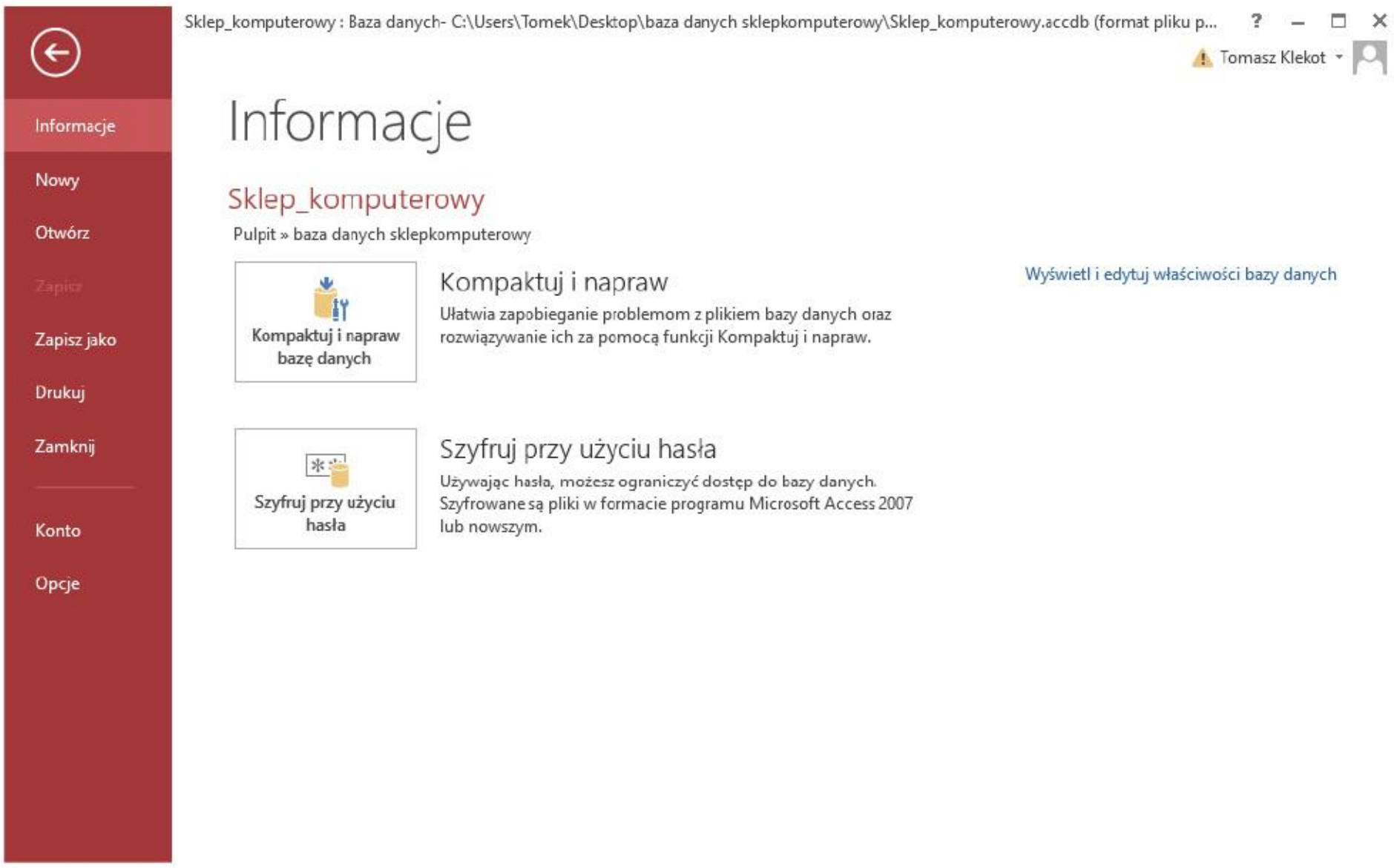
ODBC (ang. *Open Database Connectivity*) to protokół umożliwiający połączenie bazy danych programu Microsoft Access z zewnętrznym źródłem danych, takim jak program MS SQL. Zewnętrzne źródło danych to dane wraz z informacjami o połączeniu, które jest wymagane, aby można było uzyskać dostęp do tych danych. Przykładowe źródła danych to: Server SQL, Oracle RDBMS, arkusz kalkulacyjny oraz plik tekstowy. W architekturze ODBC MS Access łączy się z **Administratorem źródła danych ODBC**, który następnie łączy się ze źródłem danych za pomocą określonego sterownika ODBC. Aby dodać zewnętrzne źródło danych, należy skonfigurować ustawienia Administratora źródła danych ODBC w **`Narzędziach administracyjnych`** systemu operacyjnego Windows w **`Panelu sterowania`**.



Rys. 6.11. Okno Administrator źródeł danych ODBC

Jednym elementem zarządzania bazą danych jest **zapewnienie bezpieczeństwa**. Aby pomóc chronić pliki bazy danych programu MS Access przed nieautoryzowanym użyciem, można ją zaszyfrować – ustawić hasło. Ta opcja jest dostępna w opcji **`Informacje`** menu **`Plik`** (rys. 6.12). Trzeba jednak pamiętać o tym, aby baza danych była otwarta w trybie wyłączności. Ustawienie trybu uruchamiania domyślnego bazy danych znajduje się w opcjach programu w menu **`Ustawienia klienta`**.

Po uruchomieniu opcji szyfrowania pojawi się okno dialogowe, w którym należy wprowadzić hasło zabezpieczające.



Rys. 6.12. Grupa **Informacje** w menu **Plik**

W tym samym menu jest dostępne narzędzie **Kompaktuj i napraw**. Ta opcja pozwala na zmniejszenie rozmiaru pliku bazy danych przez wyeliminowanie nieużywanego miejsca. Polecenie **Kompaktuj i napraw bazę danych** może również poprawić wydajność bazy danych. Jest to niezbędne w razie udostępniania bazy danych w sieci. Należy jednak pamiętać, aby nikt nie korzystał z plików bazy danych podczas naprawy.

Tworzenie struktury fizycznej bazy danych za pomocą MS Access opiszemy w następnym rozdziale.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jaką funkcję w MS Access pełni protokół ODBC?
2. Jakie rozszerzenia mają pliki bazy danych MS Access?
3. Scharakteryzuj funkcje zakładki **Dane zewnętrzne** i **Tabela programu MS Access**.
4. Czy można szyfrować pliki bazy danych MS Access? Jeżeli tak, to w jaki sposób można to robić?
5. Jakie warunki muszą spełniać nazwy w obiektach tworzonej struktury bazy danych w MS Access?
6. Czy można migrować z bazami danych MS Access do chmurowych systemów bazodanowych? Uzasadnij swoją odpowiedź.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Sprawdź wymagania systemowe swojej wersji programu MS Access.
2. Zainstaluj (lub uruchom) na komputerze MS Access. Zapoznaj się z interfejsem programu.
3. Pobierz ze strony https://pl.libreoffice.org pakiet biurowy LibreOffice. Zainstaluj na swoim komputerze program Base. Opisz w zeszycie przedmiotowym różnice w budowie interfejsu oraz funkcjach między programem Base a MS Access.

7 Zarządzanie bazą danych w pakiecie XAMPP

ZAGADNIENIA

- Zarządzanie bazą danych w pakiecie serwerowym XAMPP
- Podstawowe funkcje phpMyAdmin
- Zarządzanie bazą danych za pomocą phpMyAdmin

W pierwszej części podręcznika opisaliśmy instalację i budowę pakietu serwerowego XAMPP, dzięki któremu mogliśmy publikować witryny CMS. Jednym ze składników tego pakietu jest sieciowy SZBD MariaDB (darmowa wersja serwera MySQL). Aby można było konfigurować bazy danych serwera MariaDB (MySQL) i nimi zarządzać, musimy korzystać z dodatkowego narzędzia administracyjnego o nazwie phpMyAdmin. Jest to aplikacja internetowa napisana w językach PHP i HTML. Standardowo wchodzi w skład pakietu XAMPP; phpMyAdmin można również pobrać jako oddzielny moduł ze strony https://www.phpmyadmin.net.

! UWAGA

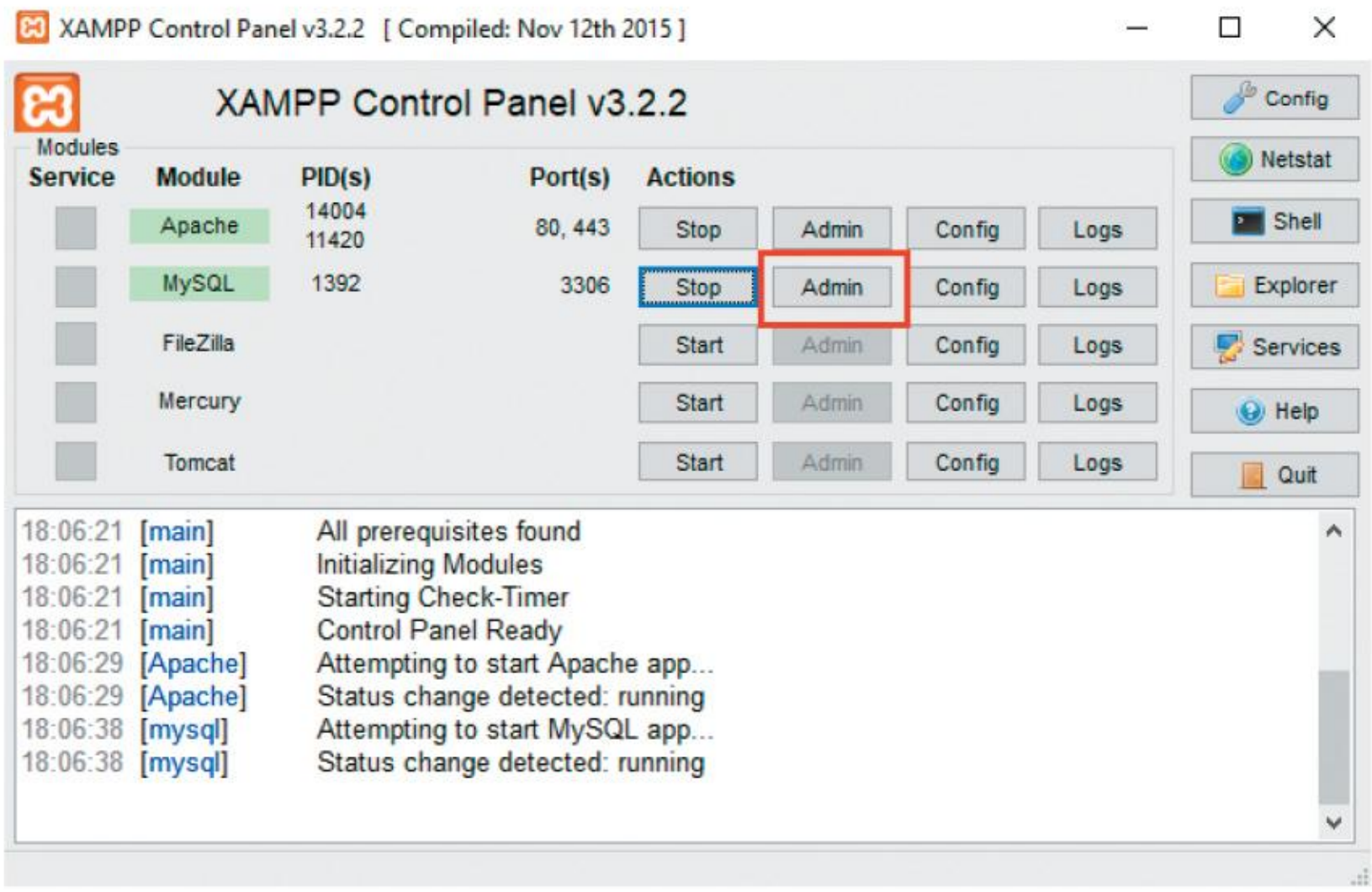
W najnowszych wersjach XAMPP zamiast serwera bazy danych MySQL jest dostarczany serwer MariaDB. Pomimo innej nazwy ten serwer SQL zachowuje się identycznie jak MySQL. Powodem zmiany są sprawy dotyczące licencji MySQL, która od 2010 r. jest własnością firmy Oracle.

Pozwala ona wykonywać czynności administracyjne za pomocą graficznego interfejsu aplikacji uruchamiającej się w przeglądarce internetowej.

Aby zarządzać bazami danych serwera MariaDB, trzeba na początku uruchomić pakiet XAMPP. Po otwarciu okna Panelu kontrolnego pakietu XAMPP należy uruchomić serwer Apache i serwer MySQL. Serwer WWW Apache jest niezbędny do działania skryptów PHP, a – jak pisaliśmy wcześniej – w tym języku napisano phpMyAdmin. Dopiero teraz można uruchomić phpMyAdmin za pomocą przycisku **`Admin`** (rys. 7.1).

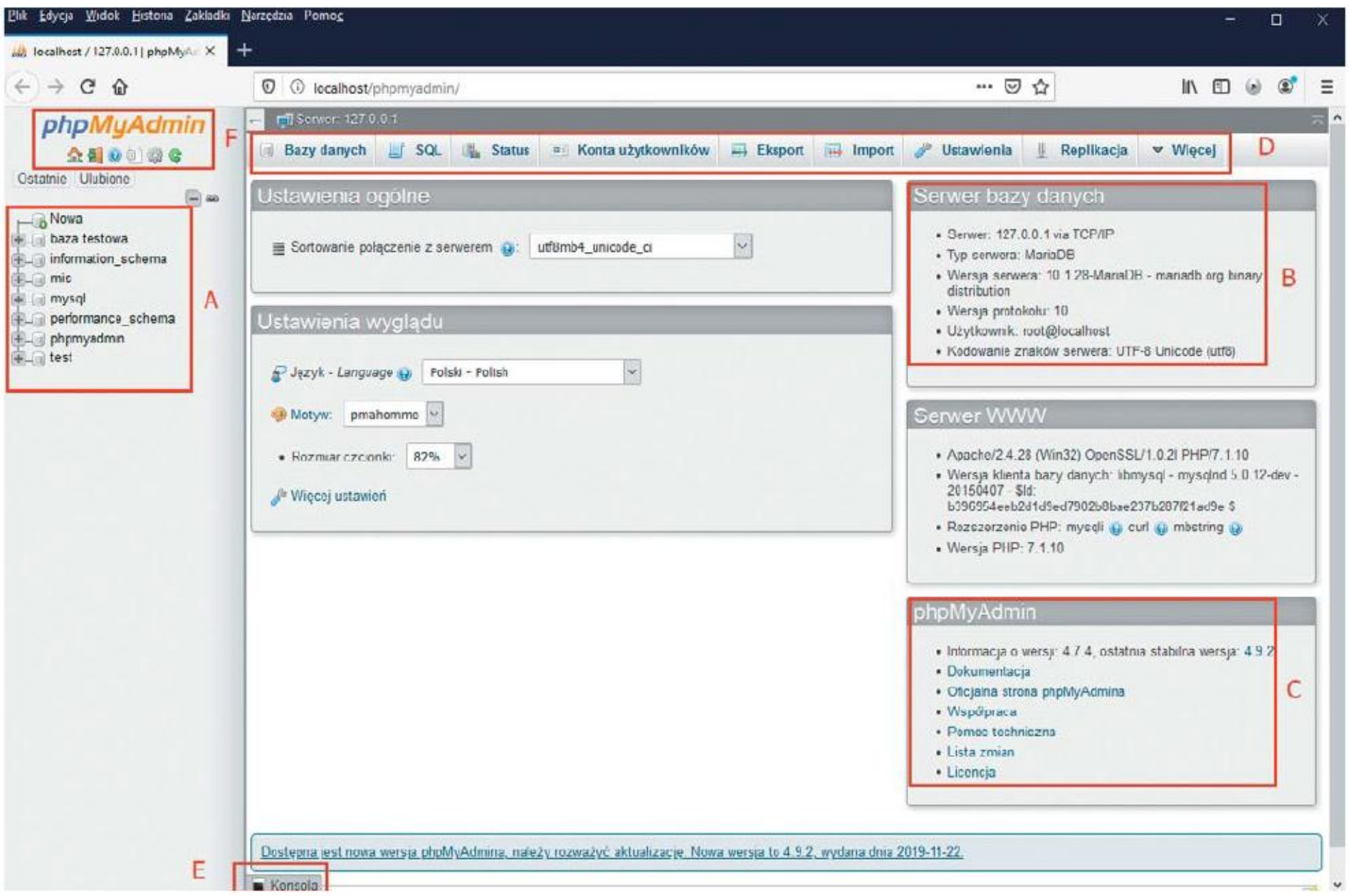
! UWAGA

Jeżeli pakiet XAMPP pracuje w tle, cały czas możemy uruchomić phpMyAdmin – wystarczy wpisać w przeglądarkę internetową adres http://localhost/phpmyadmin.



Rys. 7.1. Uruchomienie phpMyAdmin

Po uruchomieniu aplikacji wyświetli się w przeglądarce okno główne phpMyAdmin (rys. 7.2). Po lewej stronie jest wyświetlana lista dostępnych baz danych na naszym serwerze (rys. 7.2A). Pozwala ona szybko i wygodnie przełączać się między kolejnymi bazami i ich tabelami.



Rys. 7.2. Budowa okna phpMyAdmin

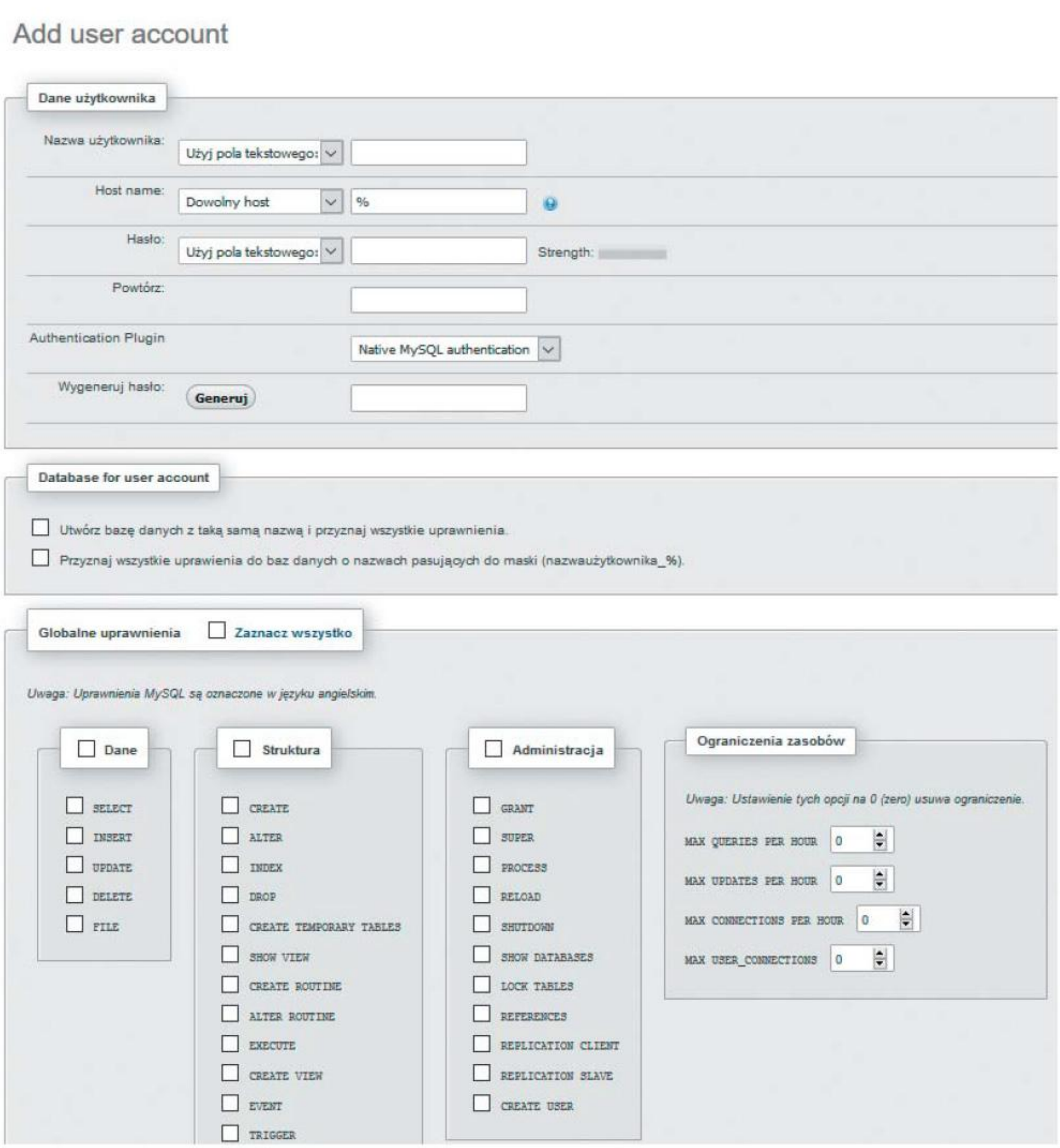
Przyciski (+) i (-) umożliwiają rozwijanie i zwijanie gałęzi tabel bazy danych. Powyżej znajduje się główny panel nawigacyjny (rys. 7.2F). Po wybraniu lewym klawiszem myszy napisu **phpMyAdmin** lub ikony **dom** zawsze wrócimy do widoku okna głównego aplikacji (opuścimy tym samym wszystkie otwarte bazy danych lub tabele). Ikona **wyjście** pozwala na wylogowanie się z serwera bazy danych. Jest to ważne zwłaszcza wtedy, gdy pracujemy na zdalnym serwerze. Ikona **znak zapytania** jest linkiem do dokumentacji phpMyAdmin, a ikona **dokument** przenosi do dokumentacji używanej bazy danych – MariaDB. Ikona **koło zębate** otwiera okno ustawień panelu nawigacyjnego, a ikona **zielona strzałka** wymusza odświeżenie całego panelu nawigacyjnego.

W głównym oknie programu jest wyświetlony ekran startowy. Ukazują się na nim najważniejsze informacje o naszym serwerze oraz wiele opcji pozwalających na zapoznanie się ze statystykami naszego serwera czy też na zmianę jego poszczególnych ustawień. Przykładowo: w jednej części są wyświetlone najważniejsze informacje na temat serwera bazy danych MariaDB (rys. 7.2B), natomiast w innej (rys. 7.2C) – informacje na temat wersji PHP oraz link do jego dokumentacji. Po lewej stronie mamy podgląd na informacje ogólne oraz ustawienia wyglądu.

W górnej części ekranu startowego znajduje się **menu zakładek** (rys. 7.2D), które jest głównym zbiorem narzędzi administracyjnych aplikacji phpMyAdmin. Zakładka **Bazy danych** pozwala na przeglądanie istniejących baz danych, na ich usuwanie, dodawanie nowych oraz sprawdzanie uprawnień użytkowników baz danych. Zakładka **SQL** daje użytkownikowi dostęp do pola tekstowego, w którym można wpisywać polecenia języka SQL. Zakładka **Status** zawiera statystyki całego serwera baz danych. Dzięki nim można dowiedzieć się o aktualnie zalogowanych użytkownikach czy najczęściej wykonywanych poleceniach. Zakładka **Konta użytkowników** pozwala na dodawanie i przeglądanie użytkowników serwera baz danych oraz na ich zarządzanie. Zmienić uprawnienia dla użytkowników może wyłącznie administrator.

Na rys. 7.3 pokazano okno dodawania użytkownika bazy danych. W pierwszej części należy podać nazwę użytkownika. Jeżeli zmienimy opcję z **Użyj pola tekstowego** na **Dowolny użytkownik**, to prawa, które nadamy dla tego konta, będą obowiązywały każdego użytkownika w bazie. Opcja **Host name** pozwala ograniczyć dostęp dla tworzonego konta do wskazanego adresu IP, lokalnego hosta lub nie ograniczać dostępu (gdy podamy znak `%` w polu bądź wybierzemy opcję **Dowolny host**). Po ustaleniu nazwy hosta możliwe staje się ograniczenie wykonywania operacji na bazie nie tylko do konta użytkownika, lecz nawet do adresu, z którego użytkownik jest zalogowany. Następne dwa pola wymagają podania i powtórzenia hasła dla tworzonego użytkownika. Poniżej znajduje się opcja pozwalająca na wygenerowanie hasła automatycznie.

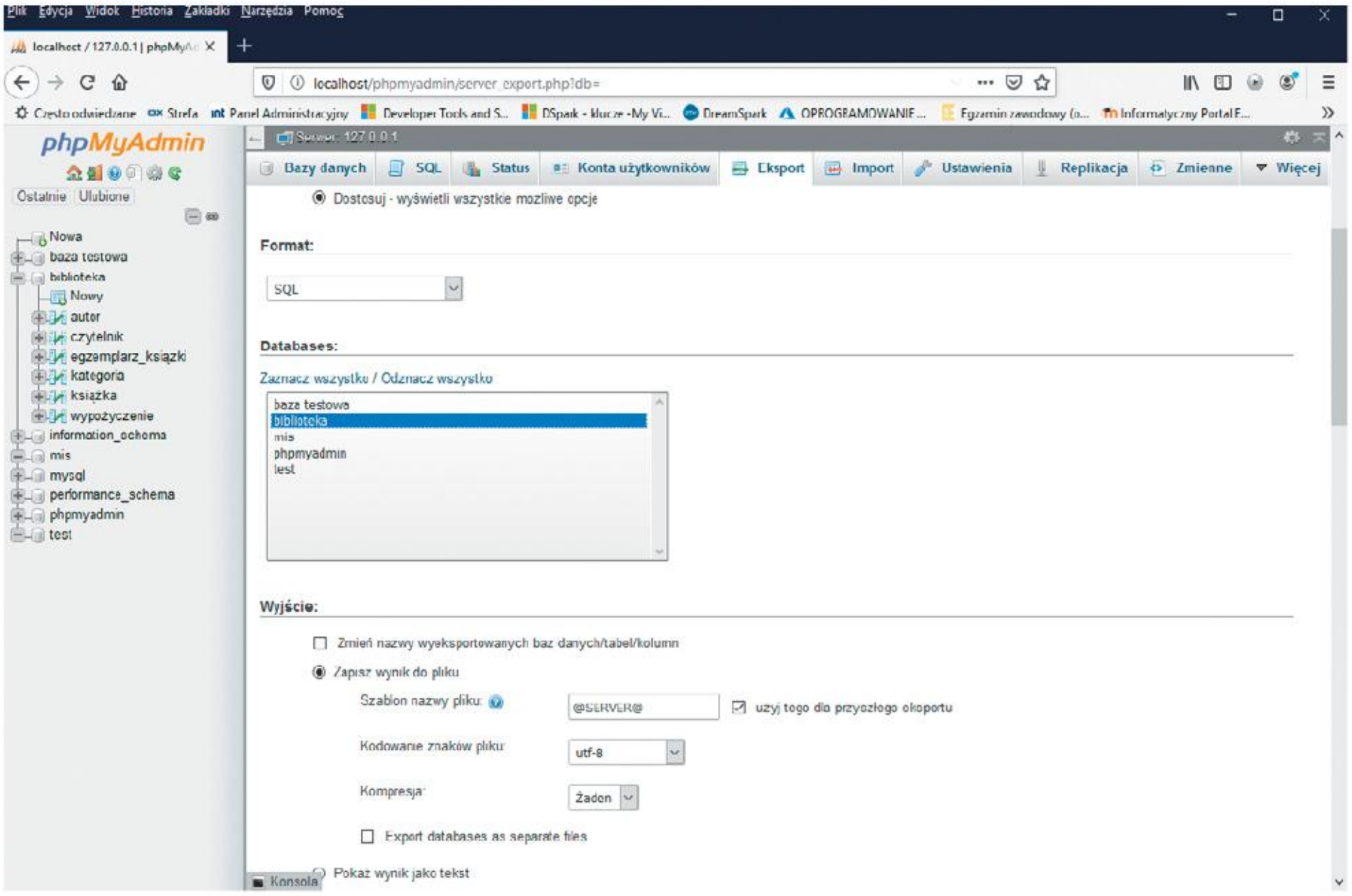
Dodatkowo mamy opcję **Utwórz bazę danych z taką samą nazwą i przyznaj wszystkie uprawnienia** lub też, jeżeli mamy już wcześniej utworzone bazy, możemy nadać do nich uprawnienia użytkownikowi, korzystając z opcji **Przyznaj wszystkie uprawienia do baz danych o nazwach pasujących do maski**.



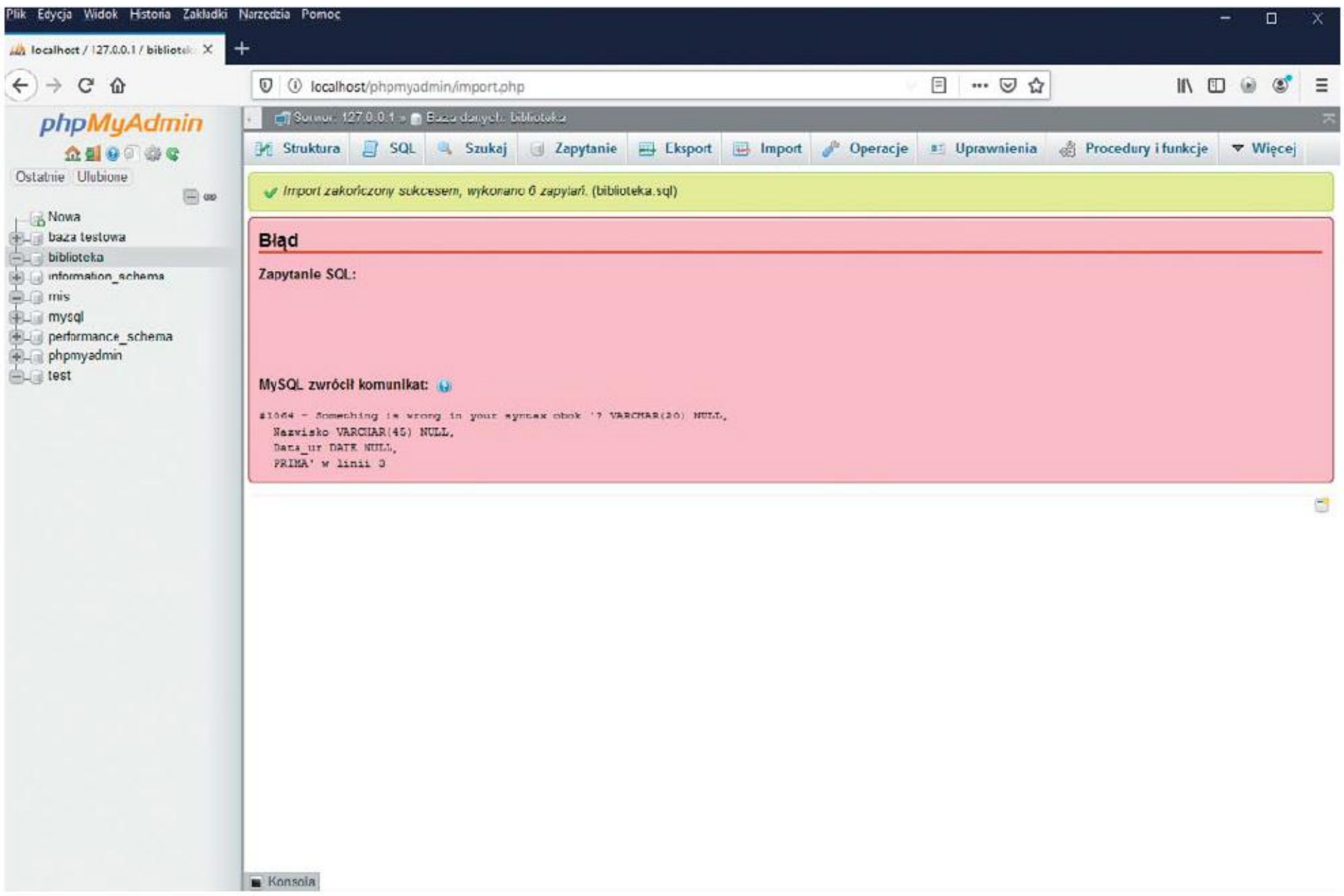
Rys. 7.3. Okno dodawania użytkownika SZBD

Formularz **Globalne uprawnienia** dotyczy uprawnień do całego SZBD bazy danych. Grupa **Dane** pozwala na określenie poleceń SQL, które użytkownik będzie mógł wykonywać na wszystkich bazach danych. Grupa **Struktura** nadaje uprawnienia do poleceń tworzących lub modyfikujących strukturę tabel. **Administracja** zawiera uprawnienia użytkownika do wykonywania takich operacji, jak nadawanie i odbieranie uprawnień, możliwość blokady baz danych lub tabel, tworzenie replikacji bazy czy też dodanie nowych użytkowników. Grupa **Ograniczenia zasobów** pozwala na ustawienie limitu wykonywania poszczególnych operacji przez wskazanego użytkownika.

Eksport umożliwia wykonanie kopii bezpieczeństwa bazy danych. Możemy wyeksportować strukturę i zawartość bazy do pliku tekstowego arkusza kalkulacyjnego oraz JSON.



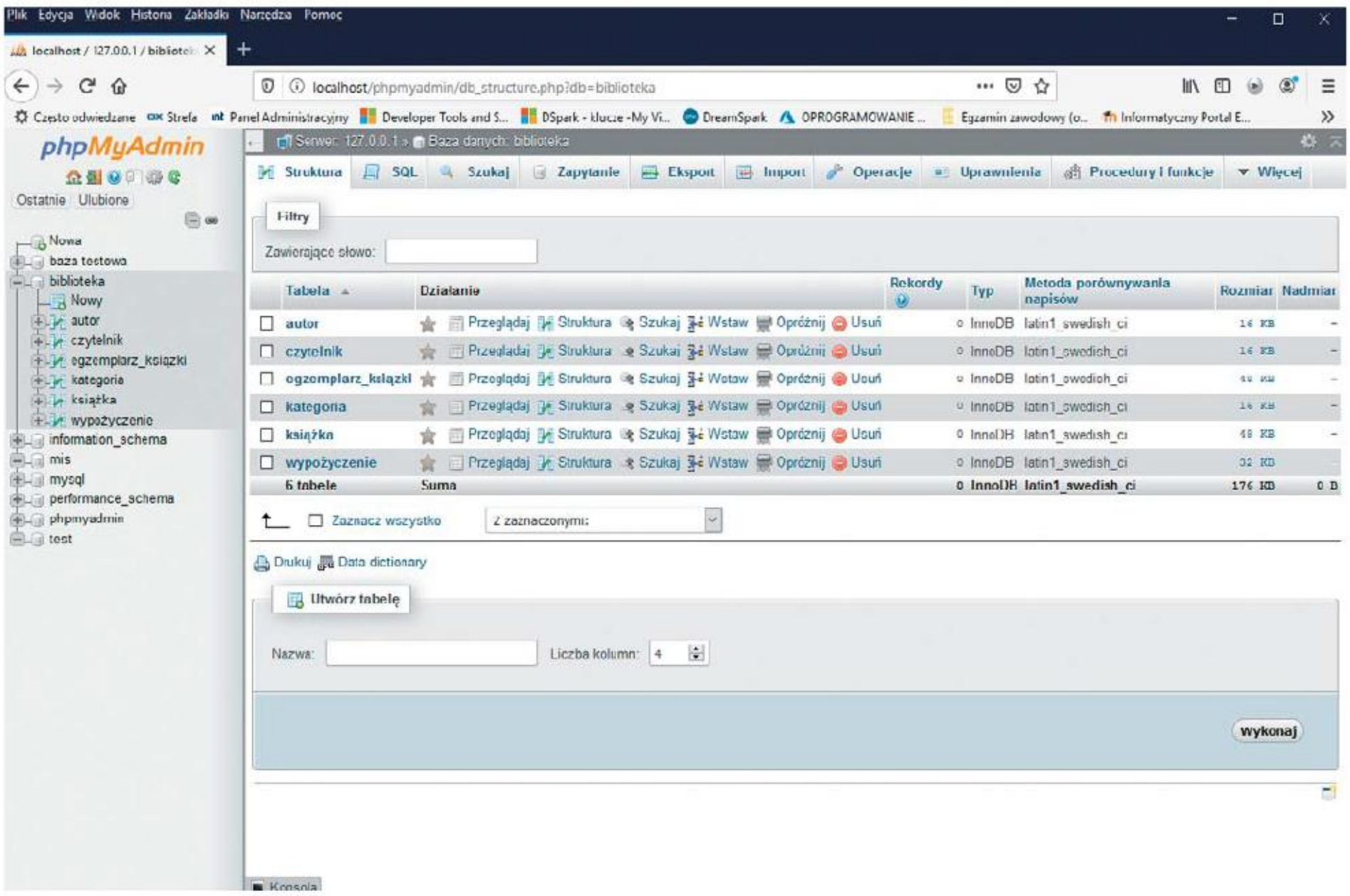
Rys. 7.4. Okno zakładki **`Eksport`**



Rys. 7.5. Okno importu zapytań i danych

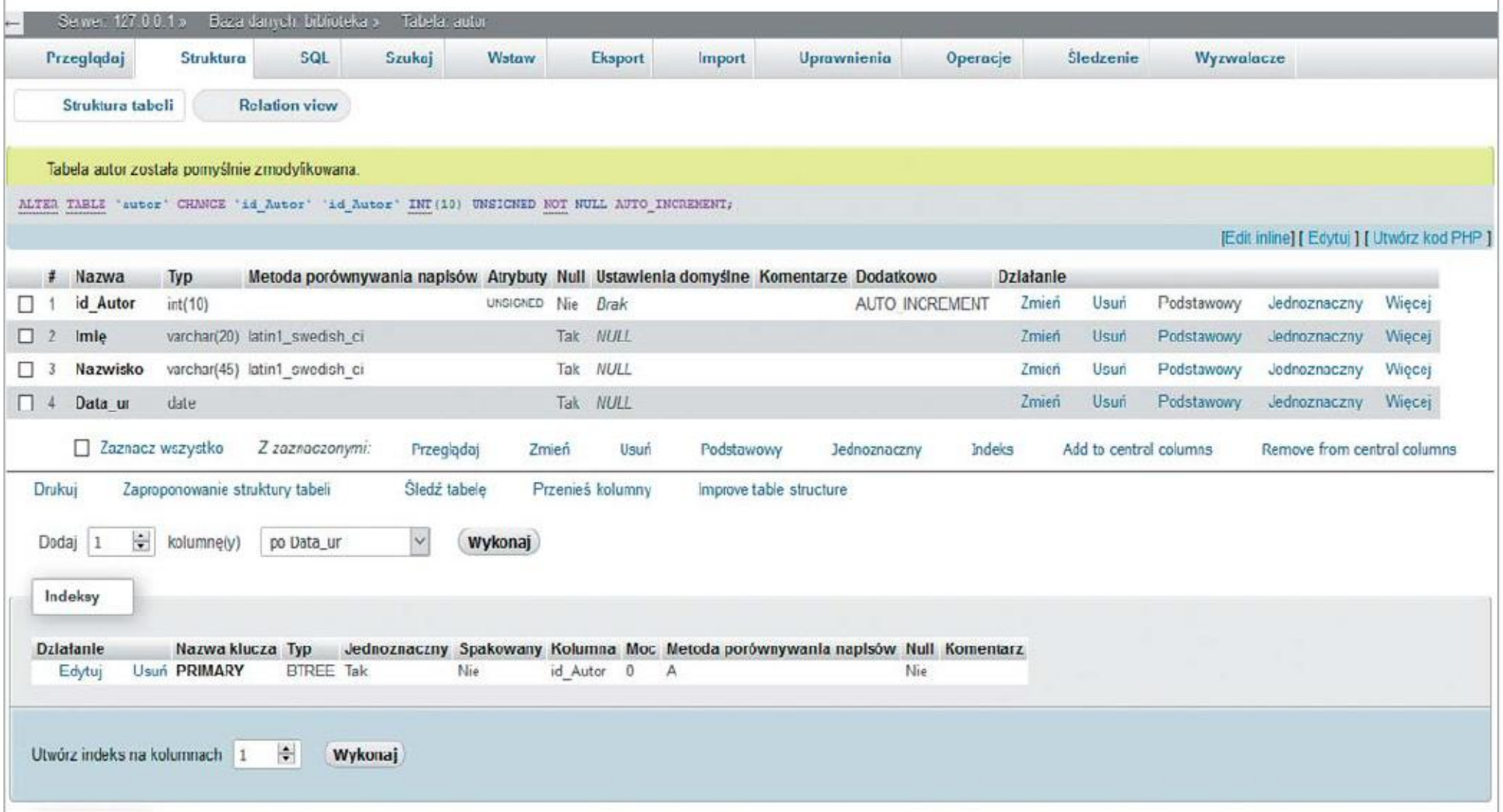
Do istotnych czynności wykonywanych w systemach zarządzania bazami danych należy wczytywanie danych z plików. Mogą to być pliki kopii zapasowych, które wczytuje się, żeby odtworzyć bazę po awarii, albo skrypty tworzące strukturę danych lub wprowadzające dane do tabel. Wykonanie tego typu operacji umożliwia zakładka **`Import`**.

Import danych odbywa się z plików tekstowych zapisanych w formacie sql, csv lub xml. Jeżeli mamy plik z zapytaniami tworzącymi strukturę bazy, który został wygenerowany z programów typu CASE, program będzie wykonywał zapisany skrypt i wyświetli na koniec operacji komunikat o jej wyniku. Jeżeli skrypt nie będzie zawierał błędów, pojawi się komunikat na zielonym tle o sukcesie operacji. Jeżeli program napotka błędy, wyświetli ich listę na czerwonym tle i zakończy proces niepowodzeniem. Należy wtedy usunąć błędy skryptu i powtórzyć import pliku. Po zakończonym imporcie na drzewie baz danych powinna pojawić się lista nowo utworzonych tabel. Jeżeli skrypt nie będzie miał polecenia tworzącego bazę danych, należy ją wcześniej utworzyć ręcznie i wybrać ją przed importem pozostałej struktury.



Rys. 7.6. Zaimportowana struktura tabel

Użytkownik mający uprawnienia do modyfikacji struktury tabel może w tym widoku wybrać określoną tabelę i przeprowadzić zmiany właściwości poszczególnych pól. Może również dowolnie rozbudować istniejącą strukturę tabel, a także powiązania między nimi. Każda operacja wykonana w trybie graficznym spowoduje wygenerowanie skryptu SQL oraz jego wykonanie, co jest widoczne w postaci pola w górnej części okna z wyświetloną składnią zapytania (rys. 7.7).



Rys. 7.7. Okno edycji pól w tabeli

Zakładka **Ustawienia** umożliwia konfigurację ustawień związanych bezpośrednio z samym phpMyAdmin. Większość ustawień będzie zapisana tylko dla bieżącej sesji.

W zakładce **Replikacja** istnieje możliwość zarządzania kopiami naszego systemu zarządzania bazą danych. Replikacje są kluczową operacją administracji bazami danych MySQL, ponieważ zapewniają ciągłość dostępu do danych nawet w razie awarii serwera (przez przywrócenie utraconej zawartości bazy danych).

W kolejnych opcjach phpMyAdmin można zobaczyć wszystkie zmienne serwera SQL (zakładka **Zmienne**), wyświetlić wszystkie kodowania znaków dostępne na serwerze i opis związanych z nimi metod porównywania ciągów znakowych (zakładka **Kodowania znaków**), dowiedzieć się o wszystkich dostępnych na serwerze mechanizmach składowania danych w bazach (**Mechanizmy**) oraz o zainstalowanych wtyczkach (zakładka **Wtyczki**).

Ostatnim elementem istotnym w zarządzaniu bazą danych jest uruchomienie konsoli znakowej do bezpośredniego administrowania bazą danych MySQL za pomocą poleceń (rys. 7.2E).

Do bezpośredniego zarządzania strukturą każdej bazy oraz operacjami na danych służą polecenia i zapytania tworzone w języku SQL. Poznaniem tego języka zajmiemy się w jednym z następnych rozdziałów.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jaką funkcję pełni w phpMyAdmin replikacja?
2. Z jakich formatów plików tekstowych najczęściej importuje się dane do bazy danych MySQL w aplikacji phpMyAdmin?
3. Wymień rodzaje uprawnień możliwych do przyznania użytkownikom do struktury bazy danych.
4. Jakie uprawnienia powinien mieć użytkownik, aby mógł wyłącznie wyszukiwać informacje z tabel jednej bazy danych?
5. Które uprawnienia użytkownika mogą naruszać ochronę danych (RODO) i należy je przydzielać ze szczególną ostrożnością?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Zainstaluj pakiet serwerowy XAMPP i uruchom phpMyAdmin. Zapoznaj się z informacjami na temat zastosowanego silnika bazy danych, kodowania znaków oraz wersji serwera bazy danych, serwera WWW i PHP. Zapisz uzyskane informacje w zeszycie przedmiotowym.
2. Utwórz w phpMyAdmin bazę danych PRZYKŁAD. Zaimportuj do utworzonej bazy danych strukturę tabel z pliku SQL, zaprojektowaną przez Ciebie w poprzednich ćwiczeniach w programie typu CASE.
3. Utwórz użytkownika o nazwie EDYTOR z dowolnym hasłem, który będzie miał dostęp do hosta lokalnego oraz prawa wyłącznie do wprowadzania, modyfikacji i usuwania danych.
4. Wykonaj eksport dowolnej bazy danych z poziomu phpMyAdmin do pliku w formacie sql.

8 Microsoft SQL Server

ZAGADNIENIA

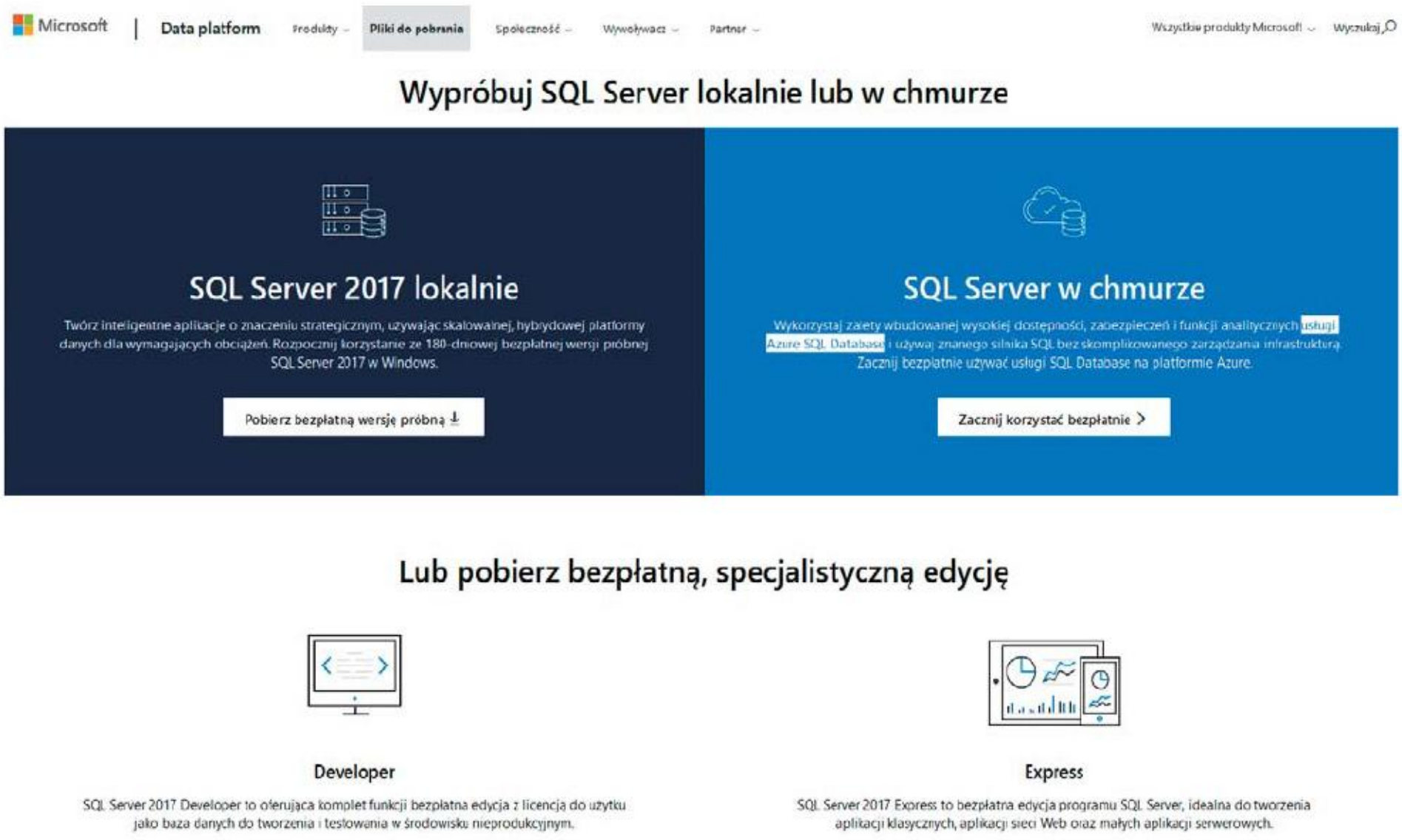
- Opis MSSQL
- Instalacja MSSQL
- Podstawowe funkcje MSSQL

Alternatywą dla MySQL jest MSSQL (Microsoft SQL Server) stworzony przez firmę Microsoft. W odróżnieniu od lokalnego Microsoft Access jest on systemem opartym na architekturze klient–serwer. Ponieważ producentem obu systemów jest ta sama firma, zastosowano mechanizmy pozwalające na migrację baz z MS Access do sieciowej lub chmurowej wersji MSSQL.

Microsoft SQL Server dostarcza zaawansowane środowisko zarządzania bazami danych przede wszystkim przeznaczone do profesjonalnych zastosowań. Oferuje wysoką zgodność ze standardem SQL, chociaż jest obsługiwany dzięki rozszerzonej wersji tego języka – Transact-SQL. MSSQL jest bardzo dobrym serwerem baz danych do zastosowania w systemach sieciowych z rodziny Windows Serwer.

Na rynku są dostępne wersje: Microsoft SQL Server 2016, Microsoft SQL Server 2017, Microsoft SQL Server 2019. Spośród różnych edycji najbardziej godna polecenia jest wersja Express, ponieważ jest darmowa i oferuje pełną funkcjonalność podczas tworzenia i obsługi niewielkich baz danych. Poza tą edycją są również dostępne wersje: Standard, Enterprise i Developer. Edycja Standard ma rozbudowane możliwości programistyczne, innowacje w zakresie zabezpieczeń oraz większą wydajność aplikacji. Enterprise poszerza dostęp do analiz biznesowych i zaawansowanych funkcji analitycznych, a także szyfrowania danych i mechanizmów przydatnych w strukturze chmurowej. Edycja developer jest przeznaczona do testowania i demonstracji serwera w środowisku nieprodukcyjnym. Pakiet instalacyjny można pobrać z witryny **`https://www.microsoft.com/pl-pl/sql-server/sql-server-downloads`**.

Warto pamiętać, że z aplikacji Microsoft SQL Server można skorzystać również w chmurowej usłudze Azure.



Rys. 8.1. Witryna do pobrania pliku instalacyjnego MS SQL Server

PRZYKŁAD 8.1

Instalację Microsoft SQL Serwer przedstawimy na przykładzie wersji 2017 Express. Przebieg opiszemy w sześciu krokach.

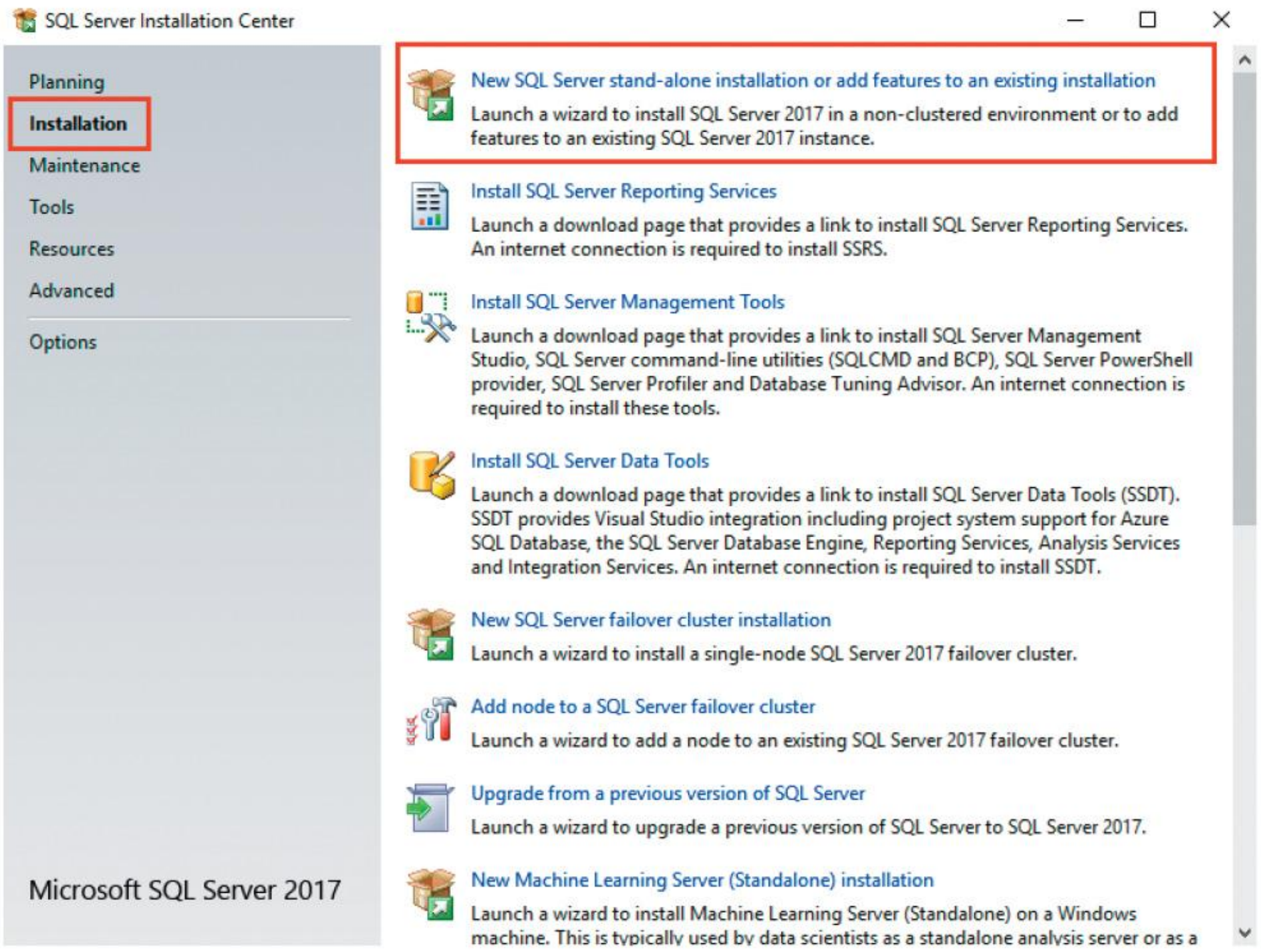
Krok 1

Po pobraniu pliku z podanego adresu uruchamiamy instalację. Po uruchomieniu okna centrum instalacji dobrze jest rozpocząć od sprawdzenia parametrów systemu i niezbędnego oprogramowania. W tym celu w menu **`Planning`** wybieramy opcję System **`Configuration Checker`**. Uruchomimy w ten sposób test kompatybilności naszego systemu komputerowego do współpracy z instalowaną wersją serwera. W razie wystąpienia błędów usuwamy ich przyczynę przed przejściem do następnych działań.

Jeżeli nasz komputer poprawnie przeszedł test, możemy przejść do menu **`Installation`** i wybrać opcję **`New SQL Server stand-alone installation or add feature to an existing installation`** (rys. 8.2). Rozpoczynamy w ten sposób instalację komponentów serwera. W oknie dialogowym **`Installation Type`** możemy zdecydować, czy chcemy zainstalować nową instancję SQL Server, czy też doinstalować dodatkowe komponenty do istniejącej instancji (jeżeli taką mamy). Jeżeli jest to nasza pierwsza instalacja, wybieramy **`Perform a new installation of SQL Server 2017`** i naciskamy przycisk **`Next`**.

Krok 2

W oknie dialogowym **`Produkt Key`** musimy określić typ naszej instalacji. Jeżeli mamy płatną edycję (Standard lub Enterprise), zaznaczamy opcję **`Enter the product key`** i wpisujemy klucz produktu. Jeżeli będziemy korzystać z bezpłatnej wersji (Express lub Developer), wybieramy właściwą nazwę w opcji **`Specify a free edition`** i przechodzimy dalej.

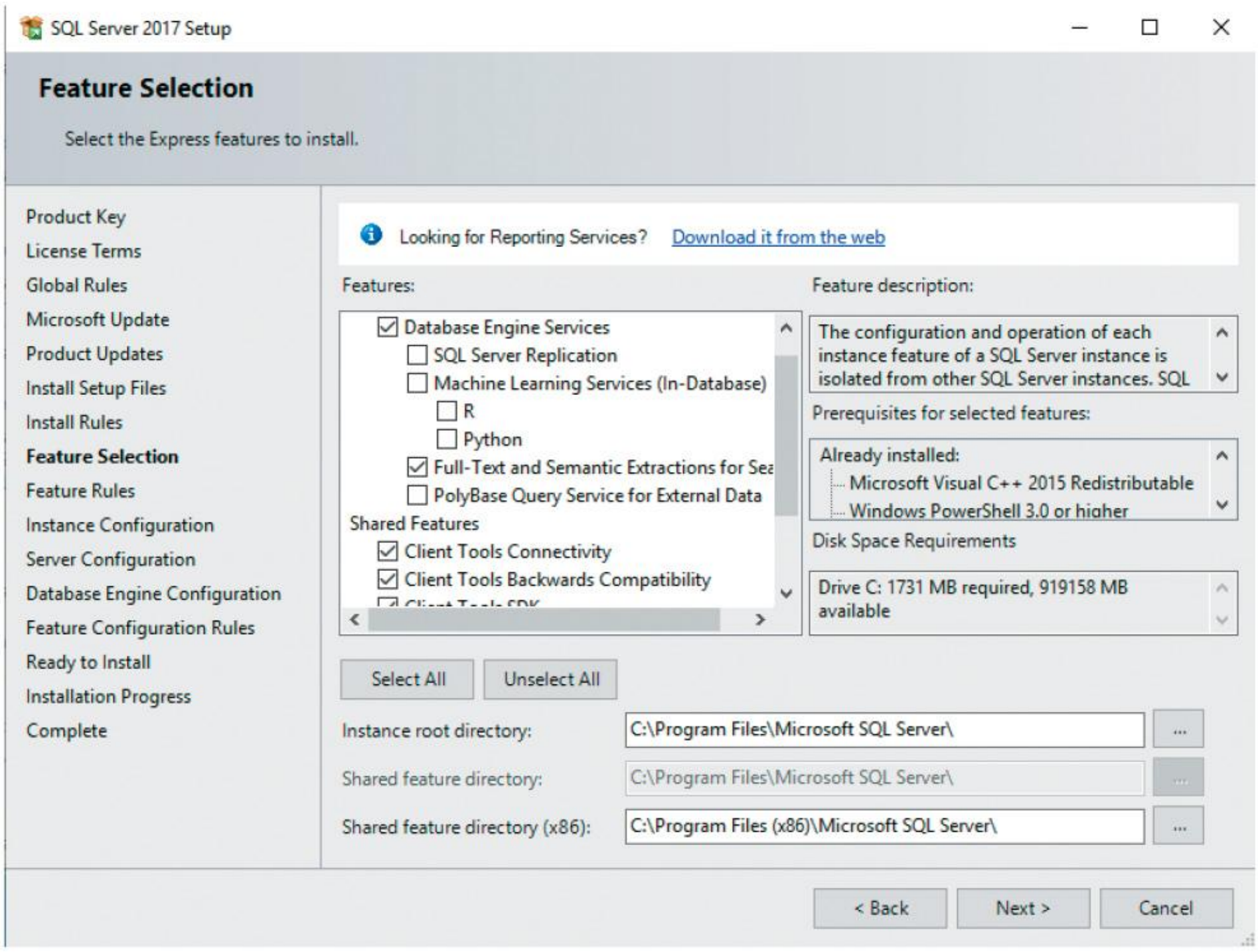


Rys. 8.2. Rozpoczęcie instalacji SQL Serwera

Krok 3

Akceptujemy licencję na produkt i przechodzimy do okna dialogowego **`Feature Selection`**. Wybieramy komponenty do instalacji serwera (rys. 8.3). Aby korzystać z pełnej funkcjonalności aplikacji, wybieramy następujące opcje:

- Database Engine Services,
- SQL Server Replication,
- Full-Text and Semantic Extractions for Search,
- Client Tools Connectivity,
- Client Tools Backward Compatibility,
- Client Tools SDK,
- SQL Client Connectivity SDK.



Rys. 8.3. Okno wyboru komponentów do instalacji

Krok 4

W następnym oknie dialogowym – **`Instance Configuration`** w polu tekstowym **`Named instance`** wpisujemy nazwę, jaką będzie miał nasz serwer. Kolejne okno – **`Database Engine Configuration`** – jest bardzo istotne w procesie instalacji. Wybieramy tutaj rodzaj autoryzacji dostępu do serwera. Możemy wybrać opcję dostępu zgodnej z profilami systemu Windows lub podać nowe hasło administratora MSSQL przy opcji **`Mixed Mode`** (rys. 8.4).

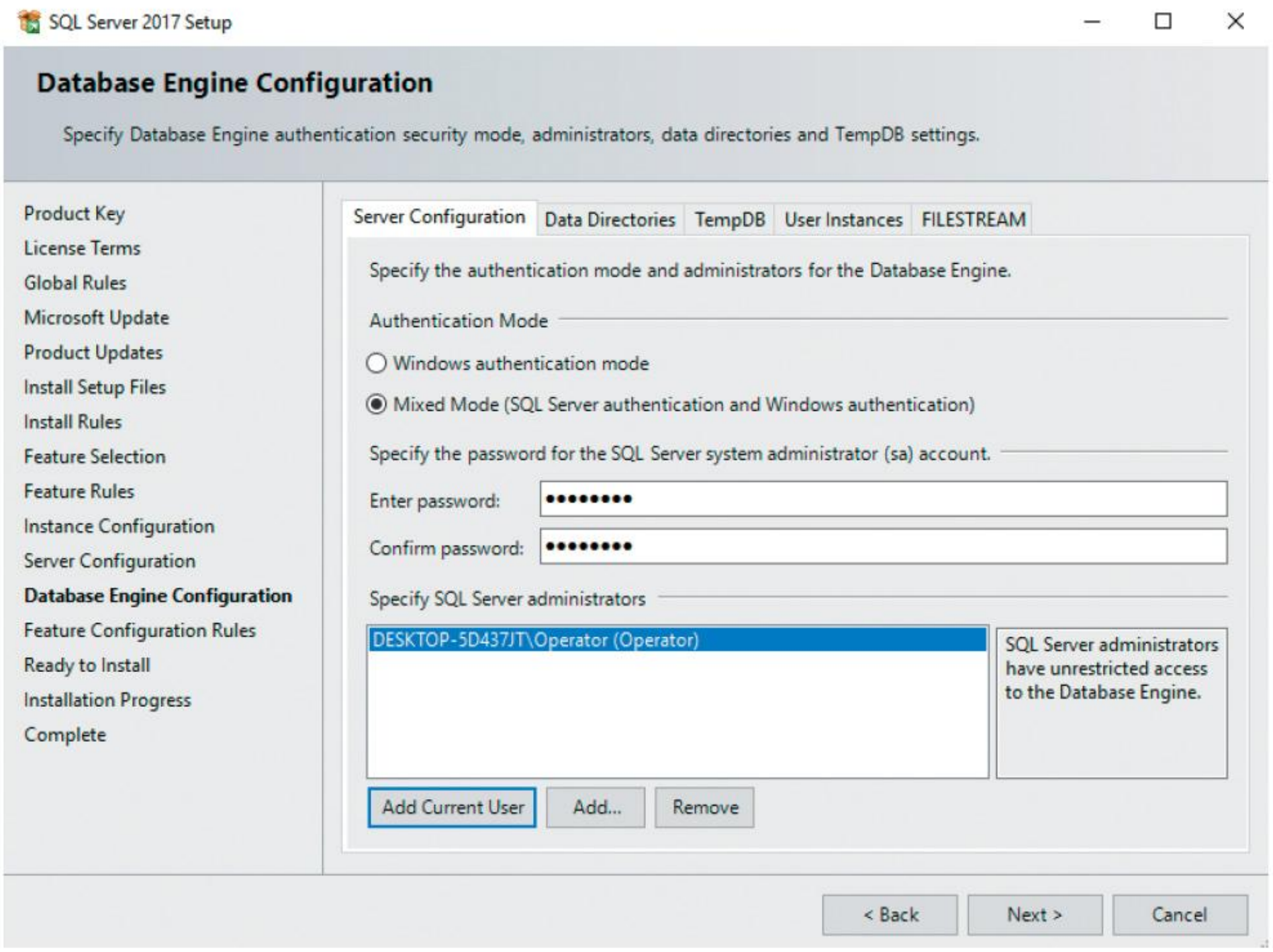
Teraz rozpoczną się instalowanie kolejnych komponentów i wstępna konfiguracja aplikacji. Po zakończeniu procesu instalacji zostanie wyświetlone okno **`Complete`**.

Krok 5

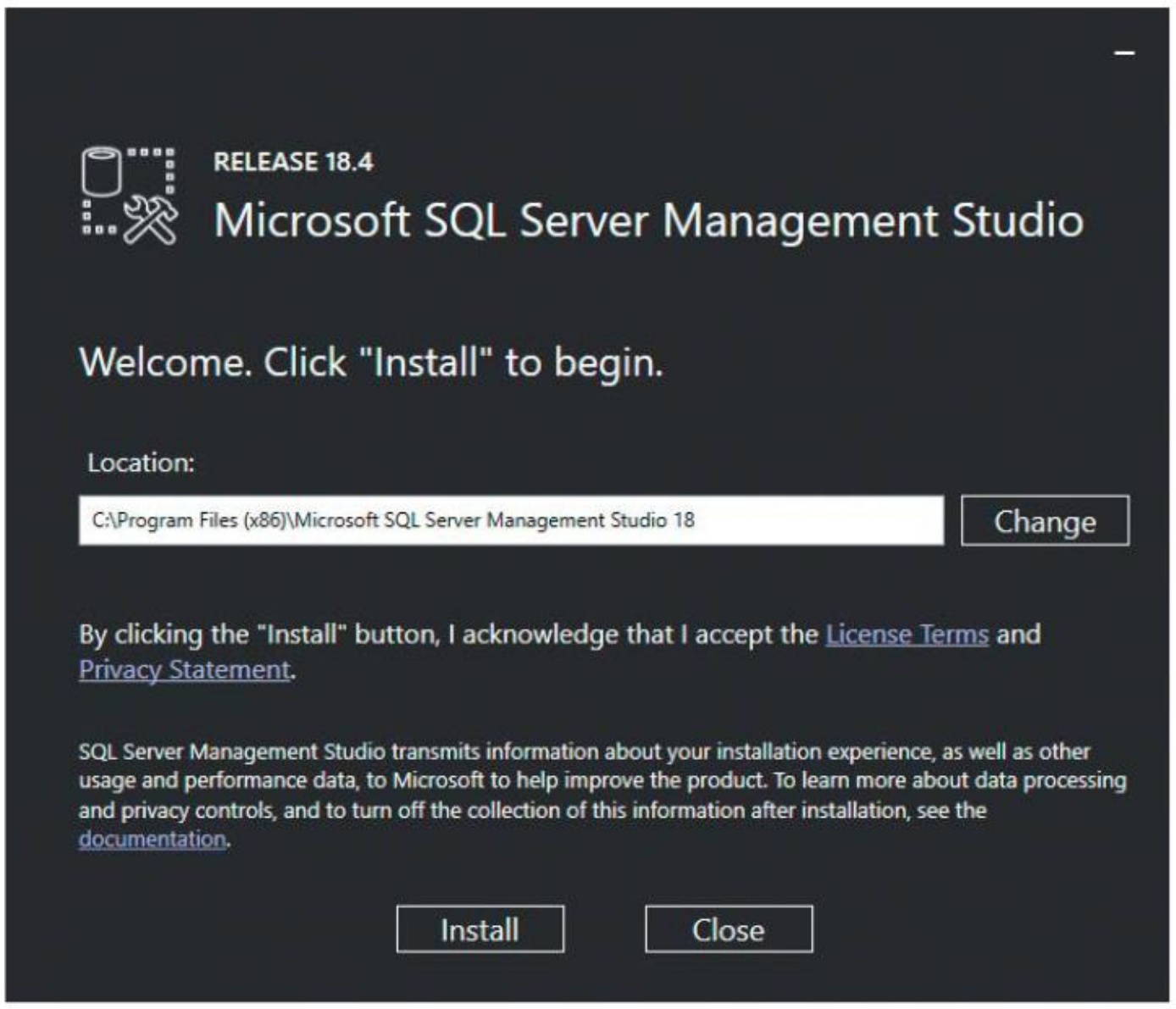
W tym momencie przechodzimy ponownie do **`SQL Server Installation Center`**. Wybieramy **`Install SQL Server Management Tools`**. Spowoduje to przeniesienie nas na stronę, z której można pobrać plik instalacyjny **SSMS** (ang. *SQL Server Management Studio*).

Podobnie jak w pakiecie XAMPP z MySQL wszelkie operacje administrator może wykonywać z poziomu aplikacji phpMyAdmin, tak w MSSQL musimy doinstalować **`SQL Server Management Studio`**.

Na stronie z dostępną dla naszego serwera wersją SSMS podajemy ścieżkę instalacji pakietu i wybieramy przycisk **`Install`**.



Rys. 8.4. Okno konfiguracji silnika bazy danych



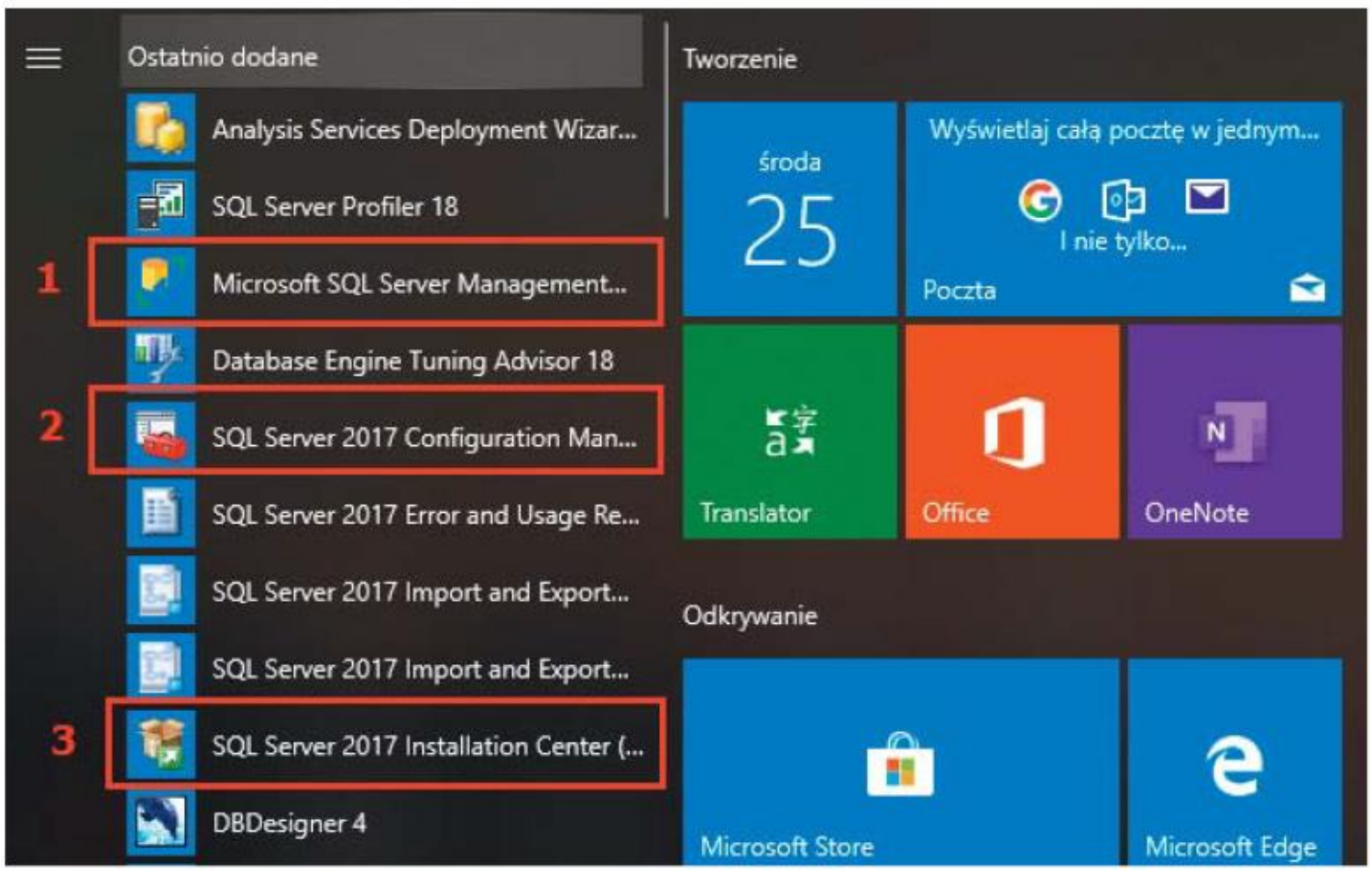
Rys. 8.5. Okno instalacji pakietu SSMS

Krok 6

Po zainstalowaniu SSMS program poprosi nas o restart komputera. Po tej operacji serwer SQL jest gotowy do pracy. Po wyświetleniu listy aplikacji z poziomu paska zadań zobaczymy ostatnio dodane do niego programy związane z Microsoft SQL Server. Do najważniejszych dla nas w pracy z systemem baz danych należą:

- Microsoft SQL Serwer Management Studio (rys. 8.6, nr 1);
- SQL Server 2017 Configuration Manager (rys. 8.6, nr 2);
- SQL Server 2017 Installation Center (rys. 8.6, nr 3).

Tak przygotowany SZBD MSSQL jest gotowy do pracy.



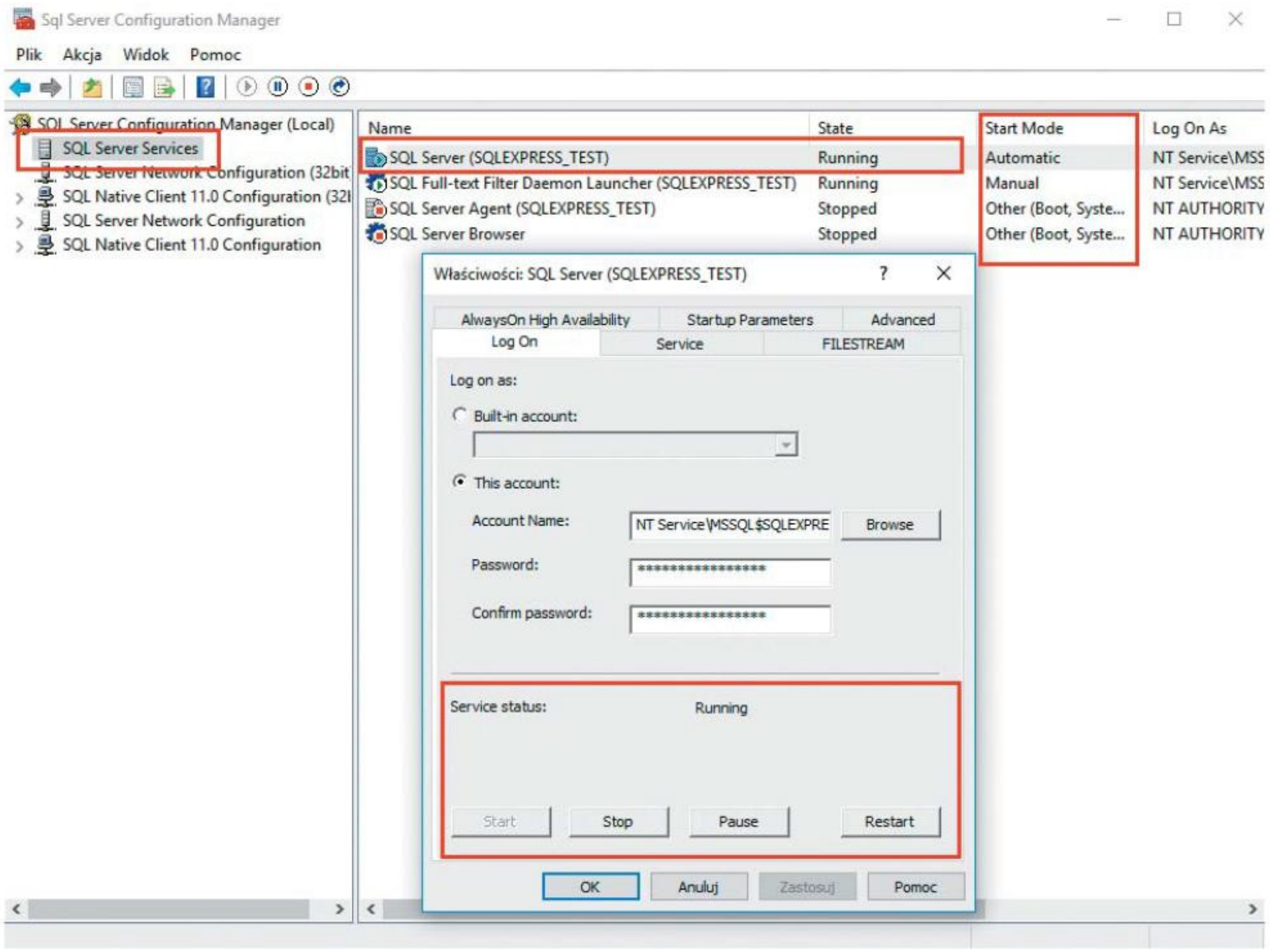
Rys. 8.6. Zainstalowane komponenty SQL Server 2017 Express

Zarządzanie SQL Server 2017 Express

Jeżeli mamy już zainstalowany serwer bazy danych oraz komponenty do zarządzania całym środowiskiem bazodanowym, możemy przystąpić do założenia bazy danych i do jej obsługi. Nie opiszemy wszystkich działań administracyjnych tego potężnego serwera bazodanowego, lecz skupimy się na tych, które umożliwiają pracę z bazą danych początkującemu użytkownikowi.

Pierwszym krokiem jest sprawdzenie, czy nasz SQL Server jest uruchomiony. Możemy to zrobić za pomocą aplikacji SQL Server Configuration Manager. Ten program służy do sprawdzania stanu usług, ich konfigurowania oraz zarządzania protokołami sieciowymi obsługiwanymi przez serwer. Na rys. 8.7 pokazano okno menedżera konfiguracji z zaznaczonymi obszarami, które informują nas o stanie usług serwera. Po włączeniu prawym klawiszem menu kontekstowego można dostać się do właściwości naszego serwera i zmienić jego parametry konfiguracyjne. Tutaj również można zatrzymać jego pracę lub zrestartować serwer. Za pomocą kolumny `Start mode` w głównym oknie konfiguruje się metodę uruchamiania serwera względem systemu operacyjnego.

W gałęzi SQL Server Network Configuration, znajdującej się po lewej stronie głównego okna, można skonfigurować pracę protokołu TCP/IP. Jest to przydatne np. w aspekcie konfigurowania zapory sieciowej. Sprawdza się tu również, jakie porty są wykorzystywane przez serwer do komunikacji w sieci.

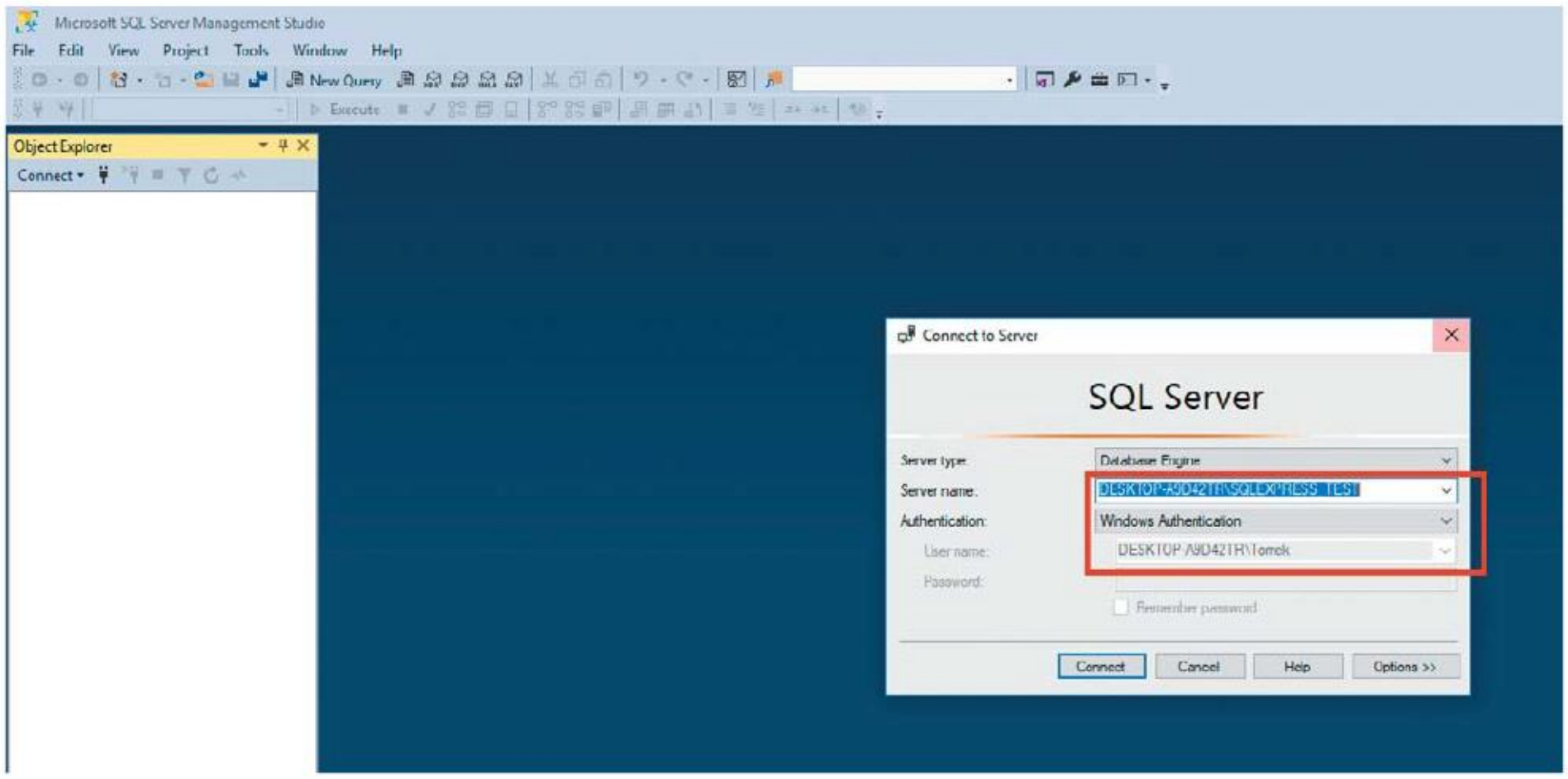


Rys. 8.7. Okno SQL Server Configuration Manager

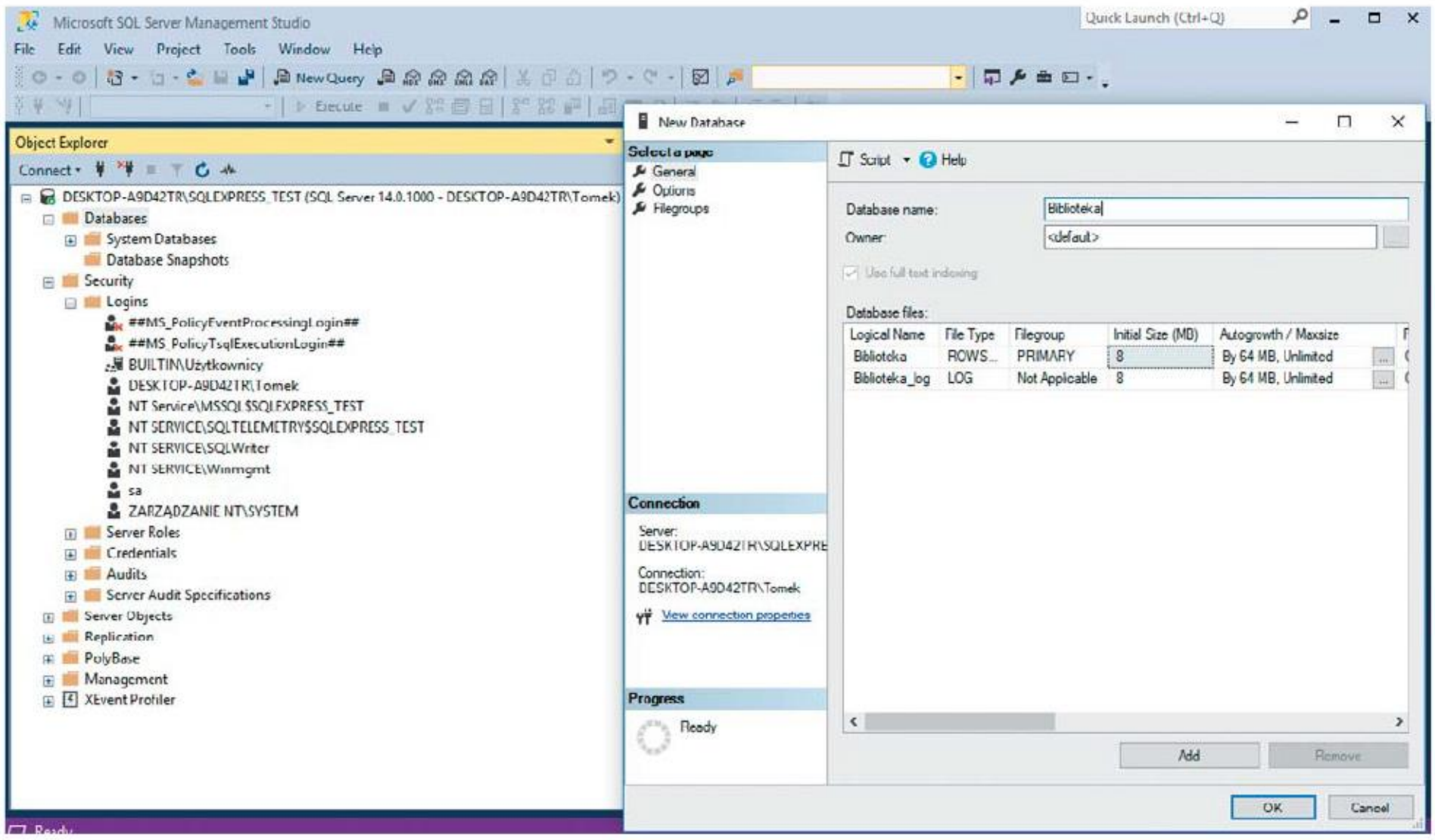
Drugim istotnym programem w zarządzaniu systemem bazy danych opartym na SQL Server 2017 jest Microsoft SQL Serwer Management Studio.

Po uruchomieniu tej aplikacji pojawi się okno logowania do systemu bazy danych (instancji serwera), utworzonego w trakcie instalacji serwera. Każdy serwer MS SQL może obsługiwać wiele zdefiniowanych instancji. Natomiast w jednej instancji może być wiele baz danych. W oknie logowania wybieramy użytkownika mającego prawa administratora do danej instancji i wpisujemy hasło. Jeżeli podczas instalacji serwera do listy administratorów w oknie uwierzytelniania dodaliśmy bieżące konto użytkownika, to nie musimy podawać hasła, ponieważ dane autoryzacyjne zostaną pobrane automatycznie z systemu operacyjnego Windows (pole zaznaczone na rys. 8.8).

Po połączeniu się z daną instancją widać po lewej stronie drzewo obiektów (ang. *Object Explorer*; rys. 8.9). Najbardziej będą nas interesowały gałęzie o nazwach **`Databases`** oraz **`Security`**.



Rys. 8.8. Okno łączenia się z instancją serwera SQL

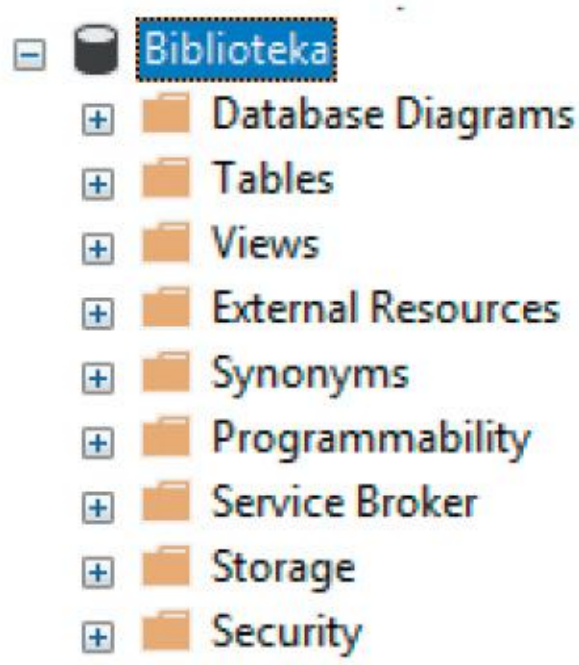


Rys. 8.9. Tworzenie nowej bazy danych

Tworzenie nowej bazy danych i struktury tabel

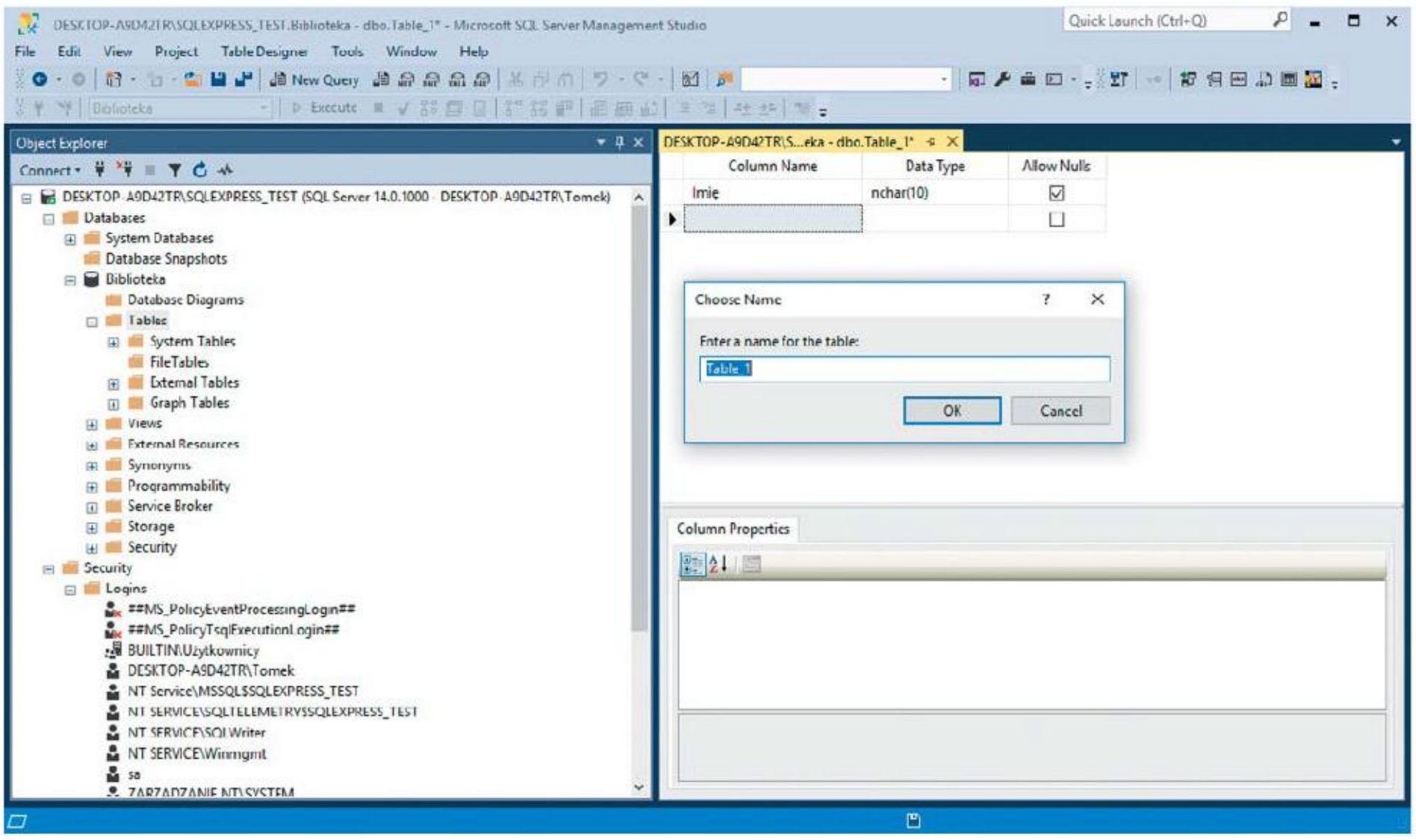
Obiekt **`Databases`** pozwala na dodawanie baz danych w serwerze i na ich konfigurowanie. Każda baza danych w MS SQL składa się z plików rozmieszczonych w określonych lokalizacjach. Plik bazy można podzielić na pliki danych (np. Biblioteka) i pliki dziennika transakcji (np. Biblioteka_log). W plikach danych są przechowywane obiekty baz danych, takie jak tabele wraz z danymi, indeksy czy procedury składowane. W plikach dziennika transakcji system zapisuje strony w blokach o określonym rozmiarze, zawierające modyfikacje danych.

Jeśli naciśniemy prawym klawiszem myszy na gałąź **Databases**, możemy utworzyć nową bazę danych – po wybraniu opcji **New Database**. W otwartym oknie (rys. 8.9) wpisujemy jej nazwę oraz rozmiary początkowe plików bazy (można pozostawić wartości domyślne). W trakcie pracy rozmiary bazy można zmienić zgodnie ze zwiększaniem się liczby danych. Po zatwierdzeniu utworzenia nowej bazy pojawi się ona w drzewie **Databases** z domyślnie utworzonymi folderami (obiektami), widocznymi na rys. 8.10.



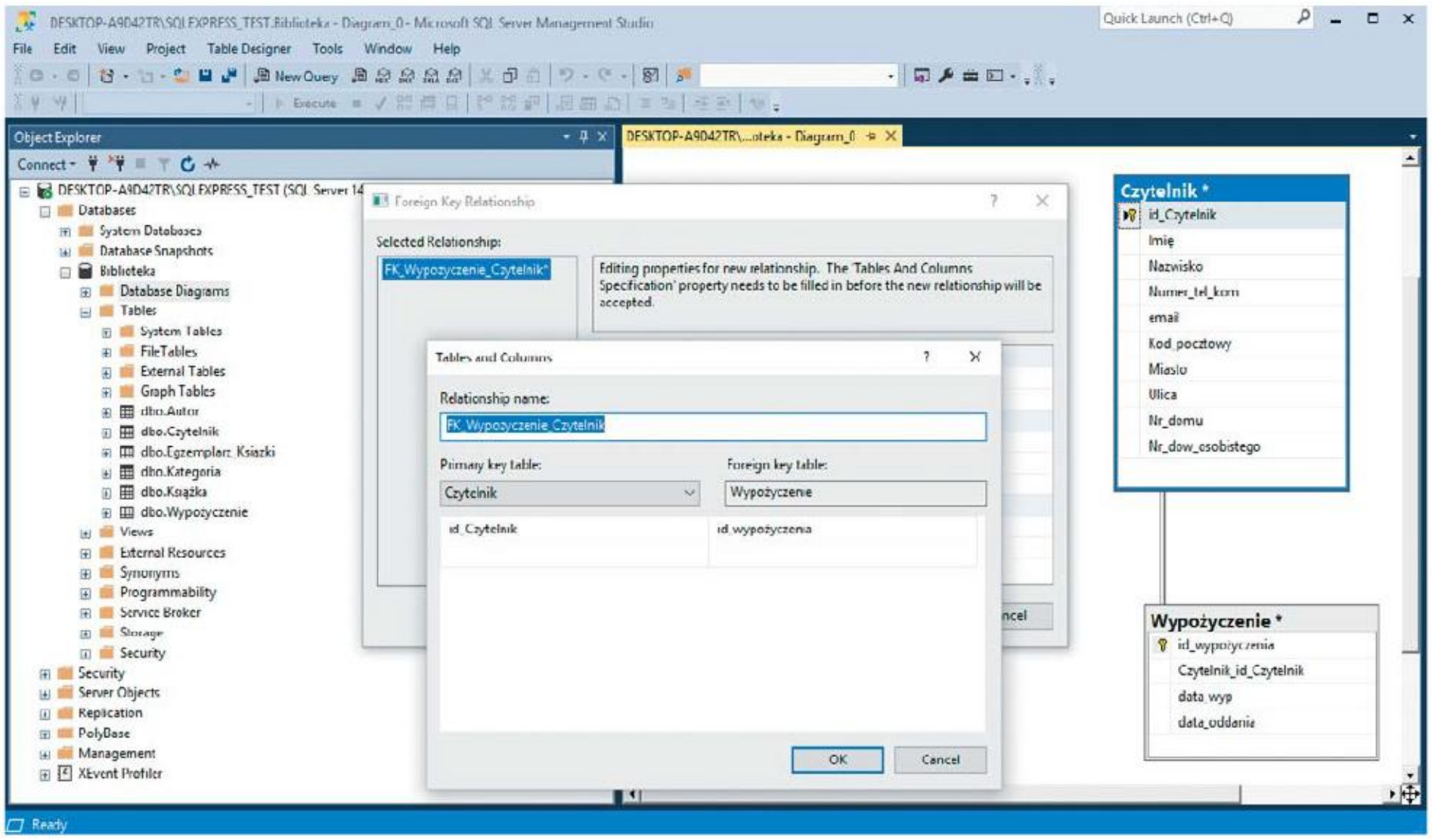
Rys. 8.10. Domyślna struktura folderów nowej bazy danych

Budowa struktury tabel naszej bazy Biblioteka jest możliwa w obiekcie **Tables**. Tutaj, po wyborze z menu kontekstowego polecenia **New/Table**, możemy tworzyć poszczególne kolumny tabeli. Podczas zamykania zakładki z oknem do tworzenia pól tabeli jesteśmy w stanie nadać jej nazwę (rys. 8.11).



Rys. 8.11. Tworzenie tabel

Po utworzeniu tabel możemy utworzyć związki między tabelami za pomocą obiektu **`Database Diagrams`**. Po uruchomieniu widoku diagramu struktury bazy dodajemy kolejne tabele i tworzymy związki, przeciągając kursor między określonymi polami (rys. 8.12). Należy pamiętać, że struktura bazy musi być odpowiednio przygotowana – chodzi o nadanie właściwych nazw pól oraz zdefiniowanie kluczy głównych.



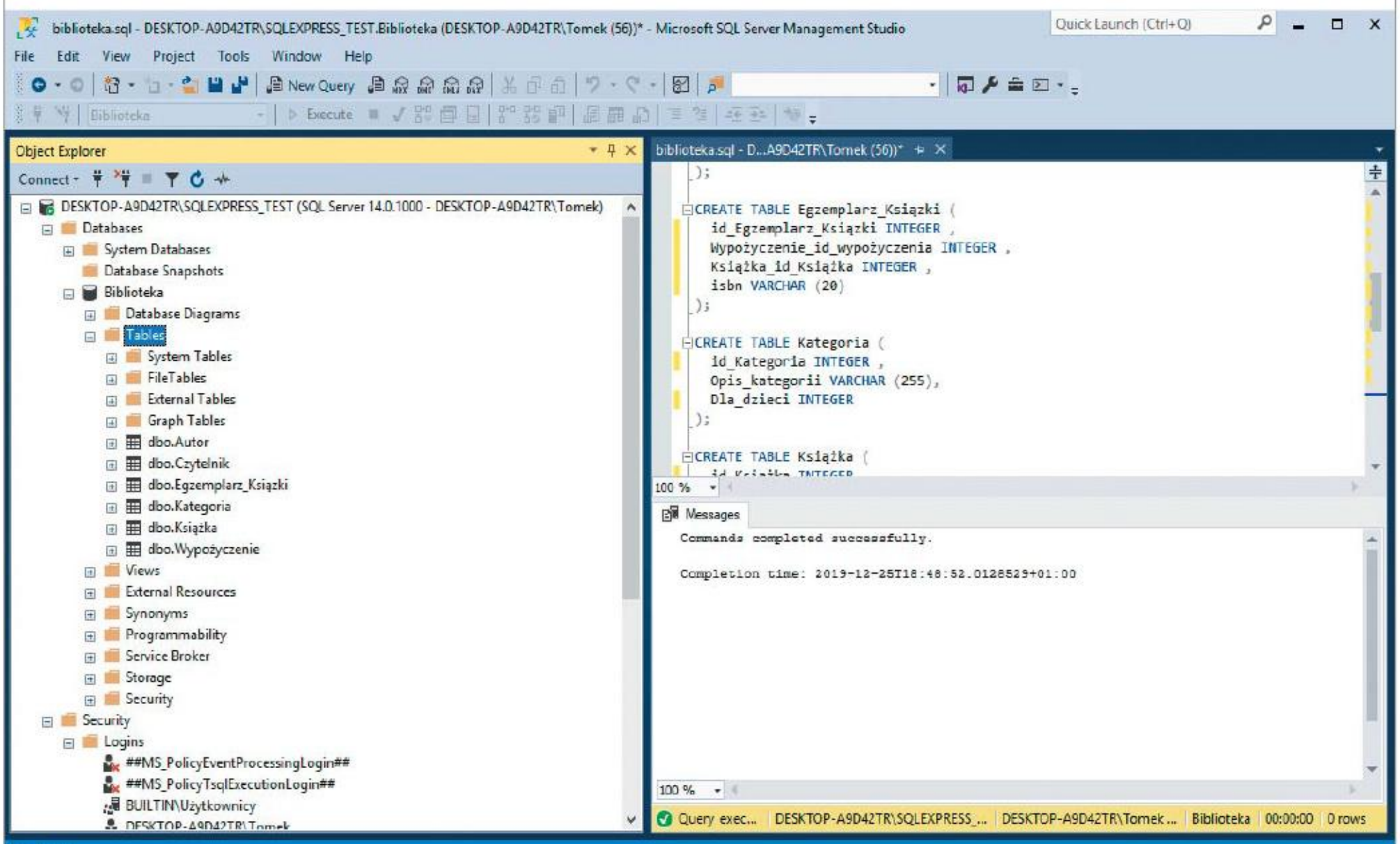
Rys. 8.12. Tworzenie diagramu bazy danych

Importowanie gotowej struktury tabel z pliku sql jest możliwe dzięki uruchomieniu zapytania tworzącego tabelę. Aby to wykonać, można po prostu uruchomić plik sql za pomocą polecenia **`Uruchom za pomocą`** i wybrać wtedy aplikację SSMS, lub skopiować zawartość pliku do okna zapytań w tej samej aplikacji. Po wyborze bazy, której dotyczy zapytanie, oraz jej uruchomieniu zostanie utworzona cała struktura tabel (rys. 8.13).

Do najczęstszych zadań administratorów SZBD należy tworzenie kopii zapasowych. W bardzo rozbudowanych bazach danych wykonuje się różne rodzaje kopii zapasowych o różnym zastosowaniu i harmonogramie tworzenia. Narzędzia MSSQL umożliwiają utworzenie m.in.:

- **pełnej kopii** zapasowej bazy danych (ang. *full database backup*), która zawiera wszystkie strony bazy danych oraz odpowiednią część logu transakcyjnego;
- **różnicowej kopii** zapasowej bazy danych (ang. *differential database backup*), która obejmuje wyłącznie dane zmodyfikowane od czasu ostatniej pełnej kopii zapasowej;
- **częściowej kopii** zapasowej (ang. *partial backup*), która obejmuje pliki dziennika transakcji.

Podczas obsługi SZBD administratorzy wykonują pełną kopię raz w tygodniu, kopie różnicowe – co dwa dni lub codziennie, natomiast kopie dziennika transakcji – np. co godzinę.

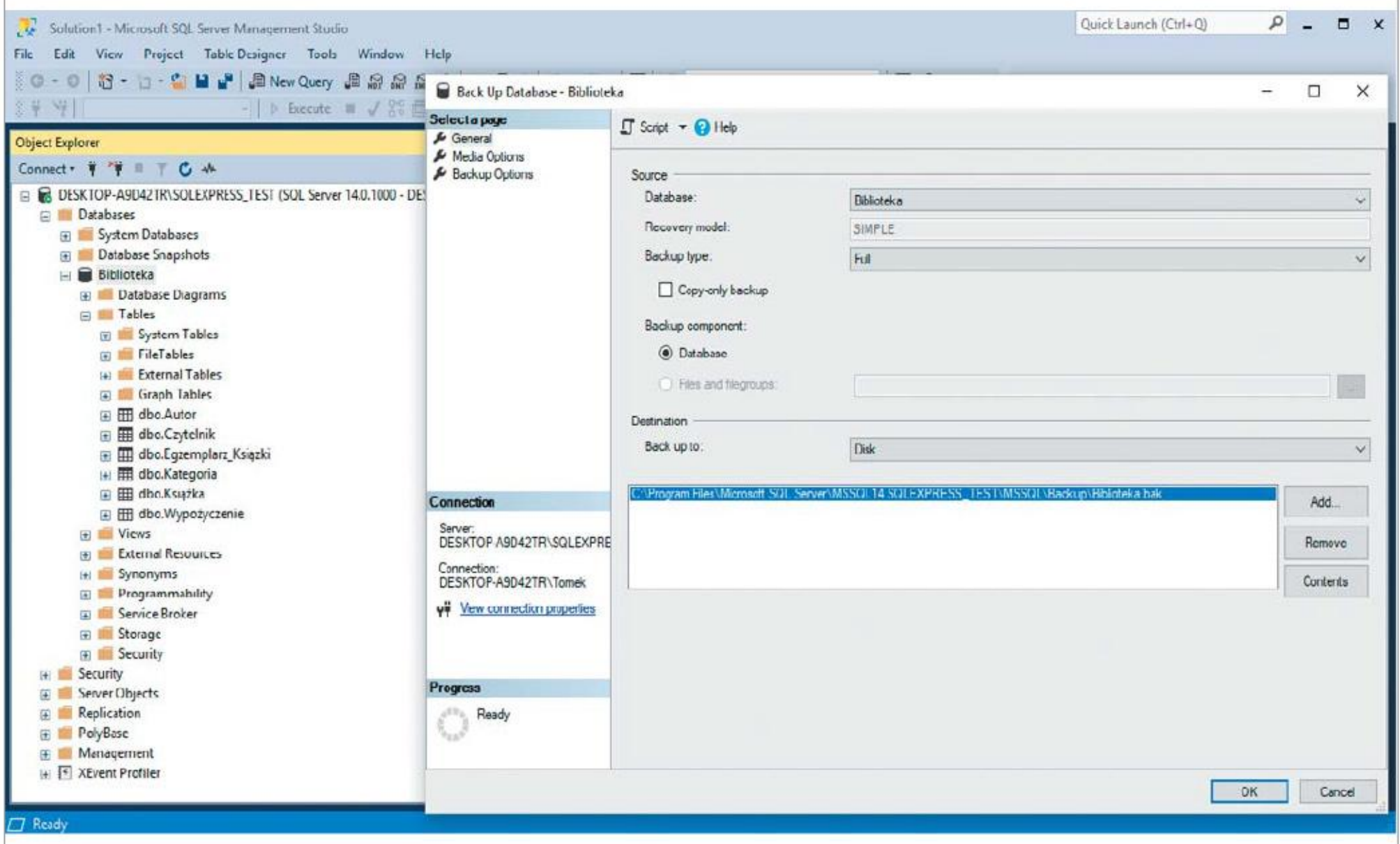


Rys. 8.13. Importowanie skryptów sql

Tworzenie kopii zapasowej w MSSQL

Jest kilka metod wykonywania kopii zapasowych. Najprostsza metoda to wykorzystanie do tego celu narzędzia SSMS. Żeby wykonać jedną z opisanych kopii bazy danych, należy postępować według następującej listy kroków.

1. Logujemy się do programu **SQL Server Management Studio**.
2. W oknie **`Eksplorator obiektów`** (ang. *Object Explorer*) wybieramy określoną bazę danych.
3. W menu kontekstowym wybieramy menu **`Zadania`** (ang. *Tasks*) i wskazujemy polecenie **`Kopia zapasowa`** (ang. *Back Up*). Pojawi się wtedy okno dialogowe wykonania kopii zapasowej bazy danych (rys. 8.14).
4. W obszarze docelowym wybieramy przycisk **`Dodaj`** (ang. *Add*). W wyświetlonym oknie dialogowym wskazujemy miejsce docelowe kopii zapasowej.
5. W polu **`Rodzaj kopii`** (ang. *Backup type*) określamy, jaką kopię chcemy wykonać.
6. Po zakończeniu procesu tworzenia kopii zapasowej pojawi się komunikat o pomyślnym zakończeniu operacji.



Rys. 8.14. Tworzenie kopii zapasowej bazy danych

Dodawanie użytkowników

Inną ważną operacją administracyjną jest tworzenie użytkowników i nadawanie im uprawnień do określonych czynności, wykonywanych na naszej bazie.

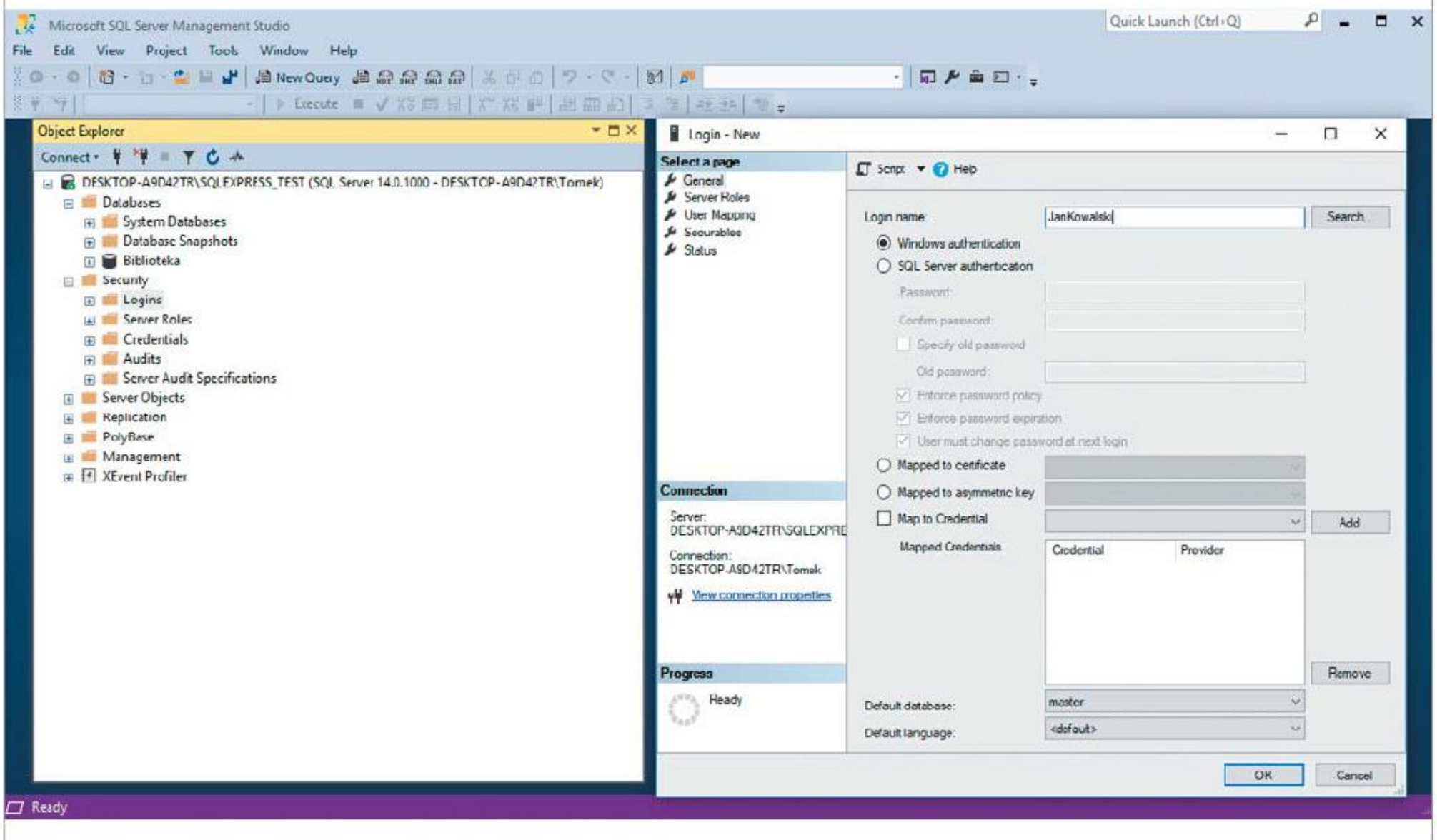
Po uruchomieniu aplikacji SSMS i zalogowaniu się do instancji założonej podczas instalacji otworzy się dobrze znane okno Eksplorator obiektów. W obiekcie **`Security`** w gałęzi **`Logins`** są widoczne wszystkie konta naszego serwera. Po rozwinięciu menu kontekstowego na nazwie Logins możemy wybrać opcję **`New Login`**, która umożliwia dodanie nowego użytkownika (rys. 8.15). W oknie **`General`** określa się podstawowe dane użytkownika, takie jak nazwa, rodzaj autoryzacji, domyślny język czy nazwa domyślnej bazy danych. Domyślnie ustawioną nazwą bazy jest **`master`**. Jest to najważniejsza baza, w której zapisuje się konfiguracja serwera SQL, czyli informacje o obiektach systemowych, loginach, lokalizacji poszczególnych plików czy podpiętych bazach użytkownika. Można również wybrać konkretną nazwę bazy, np. Biblioteka, którą wcześniej się utworzyło.

W opcji **`Serwer Roles`** ustala się uprawnienia użytkownika do wykonywania określonych operacji na serwerze SQL. Ze względów bezpieczeństwa użytkownik standardowy powinien mieć tylko rolę o nazwie **`public`**. W następnej opcji **`User Mapping`** określa się role użytkownika dla poszczególnych baz danych. Dla pojedynczej bazy danych (np. Biblioteka) do dyspozycji są następujące role:

- db_accessadmin – zarządzanie dostępem kont logowania do baz danych;
- db_backupoperator – wykonywanie kopii zapasowych;
- db_ddladmin – wykonywanie dowolnych instrukcji języka DDL w danej bazie;

- db_datawriter – wykonywanie dowolnych instrukcji języka DML w danej bazie,
- db_datareader – wykonywanie dowolnej instrukcji SELECT w danej bazie;
- db_denydatawriter – przeciwieństwo db_datawriter;
- db_denydatareader – przeciwieństwo db_datareader;
- db_owner – członkowie są administratorami baz danych;
- db_securityadmin – członkowie zarządzają uprawnieniami, członkostwem w rolach;
- public – członkowie widzą bazy danych w instancji.

W opcji **`Status`** można zarządzać aktualnym stanem konta użytkownika, zarówno gdy chodzi o prawa, jak i o logowanie do serwera.



Rys. 8.15. Tworzenie nowego użytkownika

Wykonywanie operacji na danych w bazie oraz zaawansowanych skryptów administracyjnych wymaga znajomości języka SQL, który opiszemy w następnych rozdziałach podręcznika.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Opisz różnice między poszczególnymi edycjami Microsoft SQL Serwer 2017.
2. Co określa pojęcie 'instancja'?
3. Czym różni się uwierzytelnienie Windows od SQL Server?
4. Jakie rodzaje ról można przydzielić nowemu użytkownikowi do pracy z określoną bazą danych w MSSQL 2017 Express?
5. Jaką funkcję w pracy z MSSQL realizuje aplikacja SQL Server Management Studio?
6. Czym charakteryzuje się kopia różnicowa bazy danych w MSSQL?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Pobierz plik instalacyjny serwera MSSQL oraz przeprowadź instalację wersji Express zgodnie z opisem zamieszczonym w podręczniku. Ustal z nauczycielem parametry niezbędne podczas instalacji (np. sposób uwierzytelniania, nazwę instancji). Po zainstalowaniu serwera wykonaj instalację odpowiedniej wersji SSMS.
2. Zaloguj się do zainstalowanego serwera za pomocą SSMS. Utwórz bazę danych SZKOŁA. Ustaw początkowe wielkości plików na 4 MB dla pliku danych i 1 MB dla pliku dziennika transakcji. W bazie danych utwórz tabelę UCZNIOWIE, w której wykonaj następujące czynności:
   - wprowadź nazwę dla nowej kolumny **`ID_Ucznia`**;
   - ustal typ danych dla tej kolumny jako **`smallint`**;
   - wprowadź nazwę dla następnej kolumny jako **`Imie`**;
   - ustal typ danych jako **`varchar(50)`**;
   - wprowadź nazwę dla trzeciej kolumny jako **`Nazwisko`**;
   - jako typ danych ustal **`varchar(50)`**;
   - wprowadź nazwę dla ostatniej kolumny jako **`Nr_Indeksu`**;
   - jako typ danych ustal **`varchar(10)`**;
   - dla każdej kolumny odznacz opcję **`Allow null`**;
   - ustal klucz główny dla **`ID_Ucznia`**, wybierając opcję **`Set Primary Key`** w menu kontekstowym.
3. Dodaj do bazy danych SZKOŁA tabelę definiującą KLASY według własnego projektu. Projekt musi zawierać pole ID_Ucznia.
4. Skorzystaj z obiektu Database Diagrams w bazie danych SZKOŁA i dodaj do diagramu tabele KLASY i UCZNIOWIE oraz utwórz związek między nimi. Liczebność i pozostałe cechy związku ustal z nauczycielem.
5. Wykonaj pełną kopię zapasową plików bazy danych SZKOŁA oraz zapisz ją na pulpicie komputera.
6. Utwórz użytkownika o nazwie Nauczyciel z autoryzacją SQL Server (hasło: zaq1@WSX) oraz nadaj mu uprawnienia tylko do wykonywania instrukcji SELECT oraz tworzenia kopii bazy danych SZKOŁA.
7. Zaimportuj do MSSQL strukturę bazy danych z pliku SQL, zaprojektowaną przez Ciebie w poprzednich ćwiczeniach w programie typu CASE.

IV. Tworzenie baz danych w MS Access

- Tworzenie tabel
- Operacje na danych
- Relacje między tabelami
- Tworzenie kwerend
- Projektowanie formularzy
- Generowanie raportów
- Zastosowanie makr

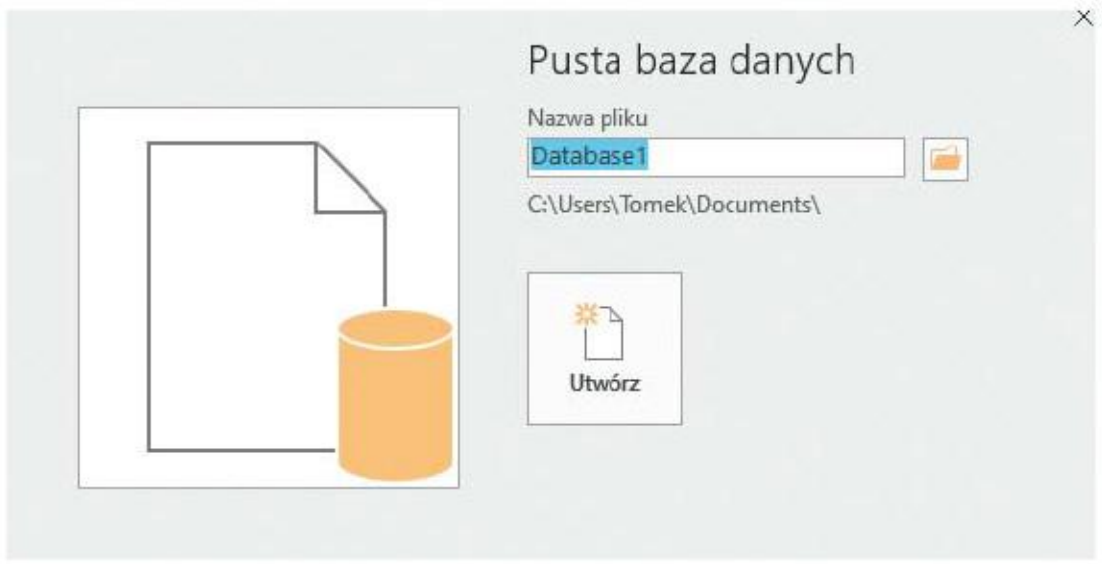
9 Tworzenie tabel w MS Access

ZAGADNIENIA

- Zasady tworzenia nowej bazy danych
- Pola i ich właściwości
- Maski wprowadzania i reguły poprawności
- Tworzenie tabel według projektu

Podczas tworzenia baza danych według zasad „dobrych praktyk" należy wstępnie zaprojektować jej strukturę w postaci diagramu ER. Pozwoli to na poprawne znormalizowanie tabel oraz określenie właściwych relacji. Trzeba jednak pamiętać, że dostępne typy danych MS Access nieznacznie różnią się od stosowanych w języku zapytań SQL z poziomu takich programów, jak DBDesigner lub MySQL Workbench.

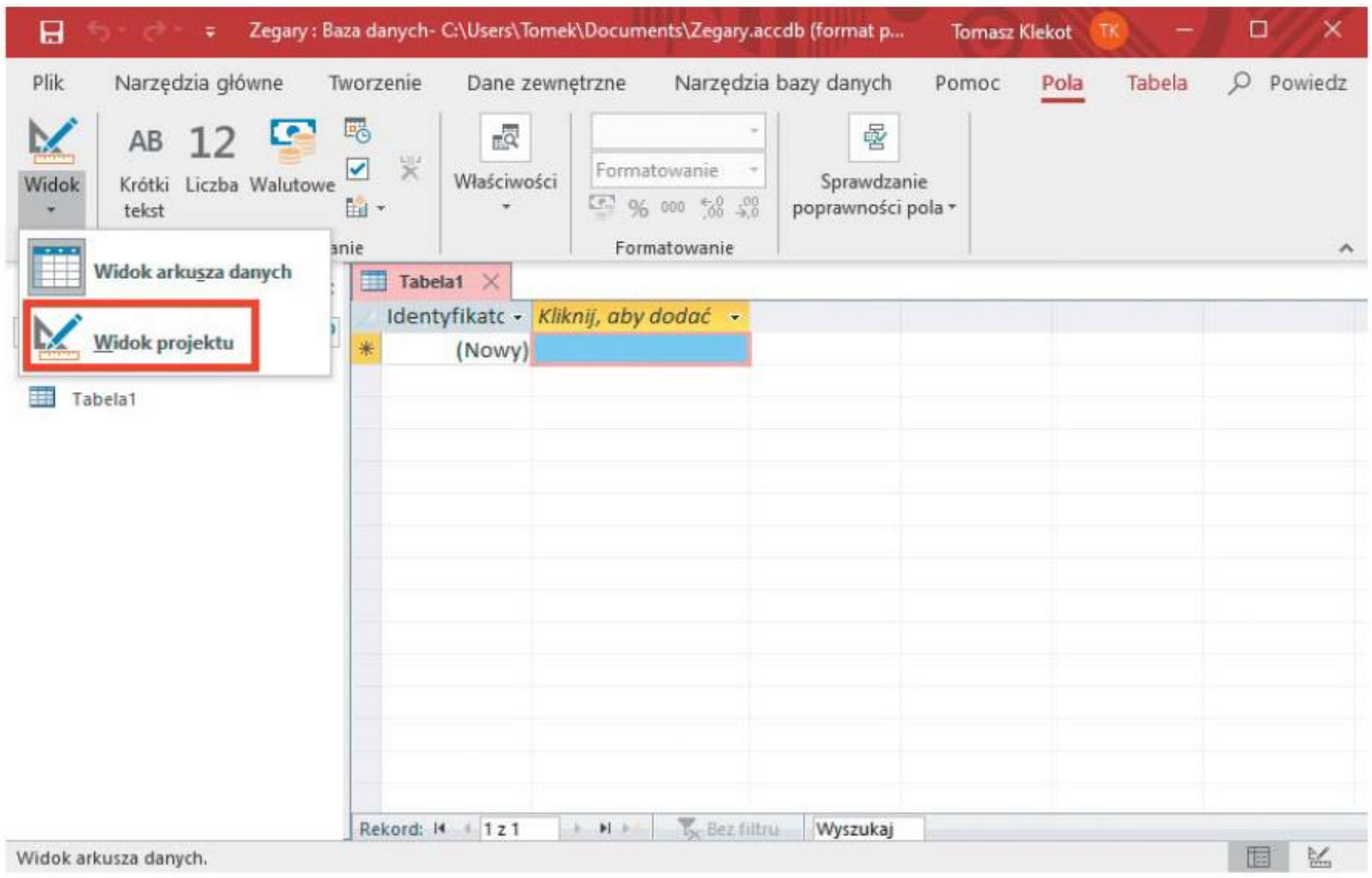
Na podstawie gotowego projektu można tworzyć strukturę fizyczną bazy w postaci tabel ich powiazań oraz innych obiektów. Po uruchomieniu MS Access należy utworzyć nową pustą bazę danych. Po wyborze ikony o tej funkcji wprowadza się nazwę bazy oraz wskazuje ścieżkę dostępu do miejsca przechowywania plików bazy.



Rys. 9.1. Tworzenie nowej bazy danych

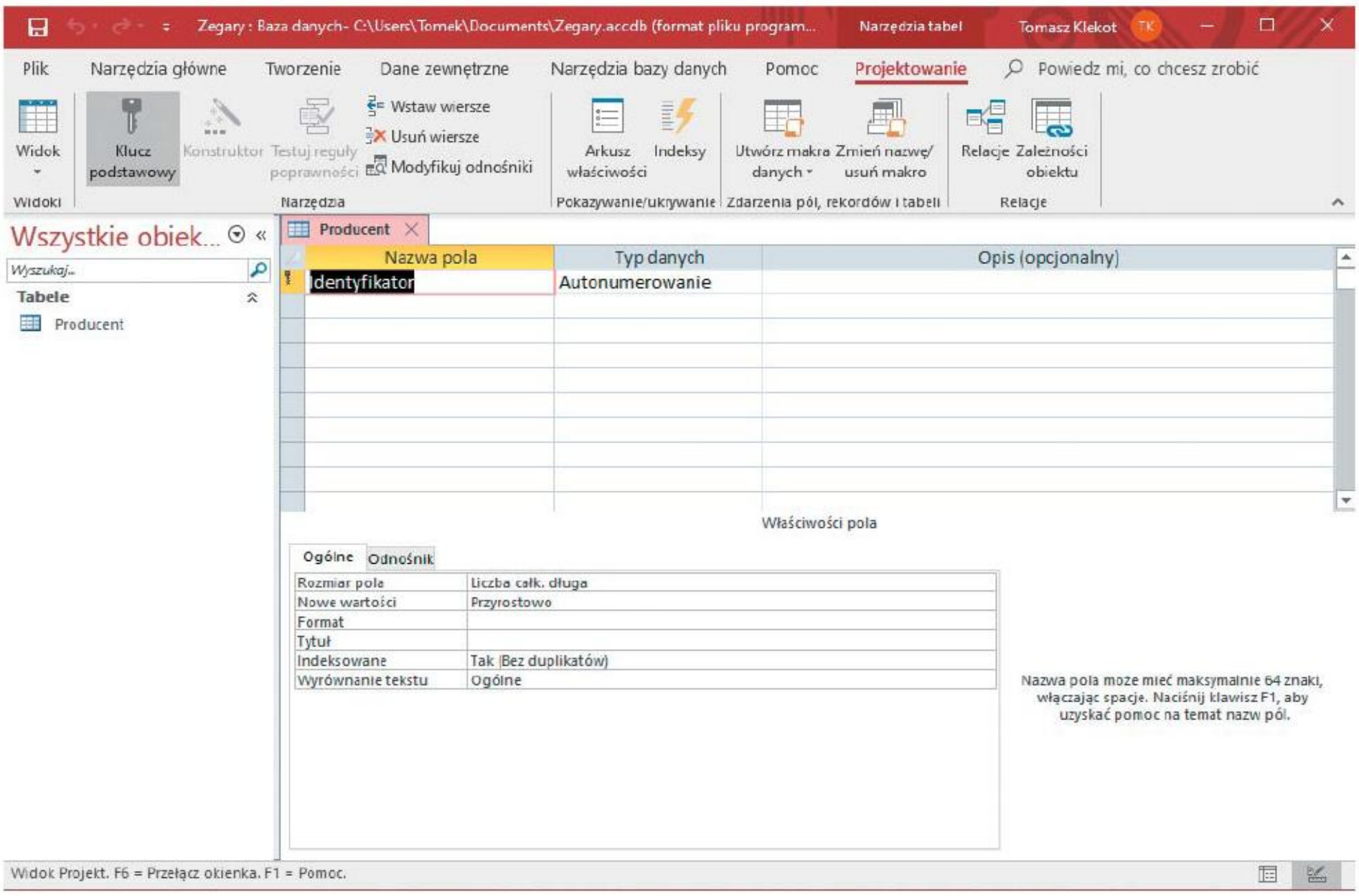
Ścieżkę domyślną, w której będą zapisywane wszystkie pliki bazy danych, można ustawić w **`Opcjach programu MS Access`** w menu **`Plik`**. Folder domyślny, zawierający bazę danych, definiuje się jako zaufaną lokalizację za pomocą opcji **`Centrum zaufania / Zaufane lokalizacje`**.

Po utworzeniu pliku bazy danych pierwszą nową tabelę o domyślnej nazwie **`Tabela1`** program tworzy automatycznie. Otrzymany obszar o strukturze tabeli jest gotowy do edycji danych. Nie ma jednak jeszcze zdefiniowanych pól tabeli i ich właściwości. Dlatego najpierw należy przełączyć widok okna roboczego na **`widok projektu`** (rys. 9.2).



Rys. 9.2. Widok projektu

Po przełączeniu widoku otrzyma się „obróconą" tabelę, w której wiersze będą określały kolejne pola w wierszu nagłówkowym tabeli bazy danych (rys. 9.3). Podczas zmiany widoku program poprosi o zapisanie zmian w pierwszej tabeli. Jest to moment, w którym można zmienić domyślną nazwę tabeli.



Rys. 9.3. Okno nowej tabeli w widoku projektu

Podczas przygotowywania się do tworzenia tabeli w MS Access trzeba dostosować nazwy pól i ich właściwości do rodzaju przechowywanych w nich danych oraz zadbać o to, aby dane były wprowadzane zgodnie ze stanem faktycznym oraz miały narzędzia pozwalające na eliminację błędów.

Podstawowe zasady podczas definiowania pól w tabelach to:

- **`Nazwa pola`** powinna być zgodna z nazwą atrybutu encji z projektu bazy danych.
- **`Nazwa pola`** musi być nazwą unikatową, która nie może się powtórzyć w obrębie tej samej tabeli.
- **`Typ danych`** pola musi być dostosowany do rodzaju informacji w nim przechowywanych.
- Należy zablokować możliwość niewypełniania pól danymi atrybutem NN (ang. *Not Null*); MS Access można to wykonać przez ustawienie we właściwościach pola wartości parametru **`Wymagane`** na **`Tak`**.
- Trzeba zapobiec wprowadzaniu takich samych danych w różnych polach tabeli; w tym celu należy we właściwościach pola ustawić wartość parametru **`Indeksowanie`** na **`Tak (bez duplikatów)`**.
- Należy korzystać z wartości domyślnych, wprowadzanych automatycznie do pól.

Nazwy pól wprowadza się kolejno – zaczyna się od pierwszego pola będącego zazwyczaj kluczem podstawowym (głównym). Po zdefiniowaniu nazwy określonego pola definiuje się typ danych za pomocą rozwijanej listy (tabela 9.1). Ostatnim elementem jest **`Opis pola`**, który można wykorzystać jako komentarz tworzący dokumentację projektu tabeli. Opis pola jest wartością opcjonalną i nie wpływa na funkcjonalność tabeli w strukturze bazy danych.

Tabela 9.1. Podstawowe typy danych w MS Access

Typ danych	Opis
Krótki tekst	wartości alfanumeryczne do 255 znaków
Długi tekst	tekst lub połączenie tekstu i liczb, które można formatować za pomocą kontrolek czcionek i kolorów maksymalnie do 1 GB tekstu, mimo że w formularzach i raportach może wyświetlać tylko pierwszych 64 000 znaków; zastąpił wcześniejszy typ o nazwie „nota”
Liczba	wartości liczbowe o określanym we właściwościach rozmiarze (np.: bajt, liczba całkowita, liczba całkowita długa, podwójna precyzja)
Duża liczba	wartości liczbowe zgodne z typem danych SQL_BIGINT w technologii ODBC; za pomocą tego typu danych można wydajnie obliczać duże liczby (od -2^{63} do $2^{63}-1$)
Waluta	wartości pieniężne
Tak / Nie	wartość logiczna Tak lub Nie (prawda / fałsz)

Typ danych	Opis
Data / Godzina	wartość daty lub godziny z przedziału lat od 100 do 9999
Obliczeniowy	wyniki obliczeń; obliczenia muszą zawierać odwołania do innych pól w tej samej tabeli; do tworzenia obliczeń służy Konstruktor wyrażeń
Załącznik	dołączone obrazy, pliki arkuszy kalkulacyjnych, dokumenty, wykresy i inne typy obsługiwanych plików dołączanych do rekordów bazy danych o rozmiarze do 2 GB
Hiperlink	adres linku do dokumentu lub pliku w internecie, intranecie, sieci lokalnej lub na komputerze lokalnym
Obiekt OLE	obiekt wstawiany za pomocą techniki kopiuj / wklej; mogą to być wykresy, obrazy lub inne obiekty ActiveX o rozmiarze do 2 GB
Autonumerowanie	unikatowa kolejna liczba 4-bajtowa (16 bajtów dla identyfikatora replikacji); najczęściej wykorzystywana do kluczy głównych
Kreator odnośników / Odnośnik	wyświetlana lista wartości pobieranych z tabeli lub kwerendy albo zestaw wartości określony podczas tworzenia pola; powoduje uruchomienie Kreatora odnośników, umożliwiającego utworzenie pola odnośnika; typem danych pola odnośnika jest Tekst lub Liczba, zależnie od opcji wybranych w kreatorze

Właściwości tabeli i pól tabeli

Podczas konfigurowania właściwości tabeli są dostępne opcje (w dolnej części okna widoku projektu). Do najważniejszych z nich należą:

- **rozmiar** – maksymalna liczba znaków, które mogą być wpisane w pole danego typu;
- **format** – służy do wyświetlania danych w jednolitej notacji;
- **maska wprowadzania** – określa sposób wprowadzania przez użytkowników danych do pól w tabeli; można przez nie wymusić wprowadzanie danych w określonym zapisie, łącznie z zastosowaniem norm językowych; aby zapewnić poprawność wprowadzania danych, należy używać masek wprowadzania, które stosuje się dla pól typu: `Liczba`, `Data/Godzina` i `Tekst`.

Do tworzenia masek wprowadzania oraz formatu można korzystać z symboli określonych w MS Access, wyjaśnionych w tabeli 9.2.

Tabela 9.2. Symbole maski wprowadzania

Symbol	Znaczenie
0	Cyfra od 0 do 9. Pozycja wymagana
9	Cyfra lub spacja. Pozycja opcjonalna
#	Cyfra lub spacja. Pozycja opcjonalna. Znaki plus i minus dozwolone
L	Litera od A do Z. Pozycja wymagana

Symbol	Znaczenie
?	Litera od A do Z. Pozycja opcjonalna
A	Litera lub cyfra. Pozycja wymagana
a	Litera lub cyfra. Pozycja opcjonalna
&	Dowolny znak lub spacja. Pozycja wymagana
C	Dowolny znak lub spacja. Pozycja opcjonalna
. , : ; - /	Znak użyty jako separator
<	Wszystkie następujące po nim znaki są konwertowane na małe znaki
>	Wszystkie następujące po nim znaki są konwertowane na duże znaki
!	Powoduje wypełnienie maski od prawej do lewej strony
\	Powoduje, że kolejny znak jest wyświetlany jako znak literałowy
Hasło	Dowolny znak wpisany w tym polu jest przechowywany jako znak, ale wyświetlany jako gwiazdka (*)

Definicja maski wprowadzania może zawierać maksymalnie trzy sekcje rozdzielone średnikami. Pierwsza sekcja (rys. 9.4, część A) zawiera trzon maski wprowadzania. Druga (rys. 9.4, część B) określa, czy mają być przechowywane znaki literałowe (0 lub 1). Jeżeli w tej sekcji zostanie użyta wartość 0, to wszystkie znaki literałowe (np. nawiasy użyte w masce) będą przechowywane razem z wprowadzoną wartością. Jeżeli zostanie wprowadzona wartość 1, to będą przechowywane tylko znaki wpisane do formatu. Ostatnia sekcja (rys. 9.4, część C) definiuje znak wyświetlany zamiast pustych miejsc. Domyślnie jest wykorzystywany znak „_”.

Rys. 9.4. Struktura budowy maski wprowadzania

PRZYKŁAD 9.1

Maska	Przykładowe dane
(#00) 000-000-000	(+48) 612-511-248
>L0LL 0L0	P2MS 8M2
>L< ??????????	Płyta główna
ISBN 0-&&&&&&&&&-0	ISBN 1-446155009-7
#999	-12

Tytuł to etykieta pola na formularzu pozwalająca na jej identyfikację i mogąca wystąpić w wyrażeniach obliczających.

Wartość domyślna występuje jako ciąg znaków pojawiający się automatycznie dla nowych rekordów.

Reguła poprawności to samoczynna walidacja poprawności danych. Są dwa rodzaje reguł poprawności:

- reguła sprawdzania pola,
- reguła sprawdzania rekordu.

Reguły poprawności pola używa się do sprawdzania wprowadzanej wartości z chwilą przejścia do następnego pola. Reguła sprawdzania rekordu służy do sprawdzenia, czy cały rekord może być zapisany. W razie naruszenia zasad wprowadzania danych oba rodzaje reguł będą wyświetlać komunikaty z informacją, jak poprawnie wprowadzić dane.

PRZYKŁAD 9.2

< >0	Należy wprowadzić wartość różną od zera
18 or >18	Wartość musi być równa 18 lub większa od 18
Like „EE??”	Wartość musi składać się z czterech znaków i zaczynać się do liter EE, np. EE09
<#68-1-1#	Należy wprowadzić datę sprzed roku 1968

Komunikat o błędzie to informacja, która pojawia się po wprowadzeniu danych niezgodnie z regułą poprawności.

Wymagane wartości **`Tak`** lub **`Nie`** oznaczają, że pole nie może być puste, jeśli użyto wartości **`Tak`**, lub że wpisanie informacji nie jest wymagane, jeśli użyto wartości **`Nie`**.

Zerowa długość dozwolona określa, czy dopuścić ciąg znaków o zerowej długości.

Indeksowanie to funkcja przyśpieszająca wyszukiwanie i sortowanie danych. Nie mogą być indeksowane pola **`Memo`** i **`Obiekt OLE`**. Indeks jest tworzony automatycznie dla pola klucza głównego. W tym polu ustala się, czy indeksowanie ma dopuszczać powtórzenia informacji w polu, czy – nie.

Kompresja UNICODE pozwala na ograniczenie miejsca w pamięci dla pól typu **`Tekst`**, **`Memo`** lub **`Hiperłącze`**.

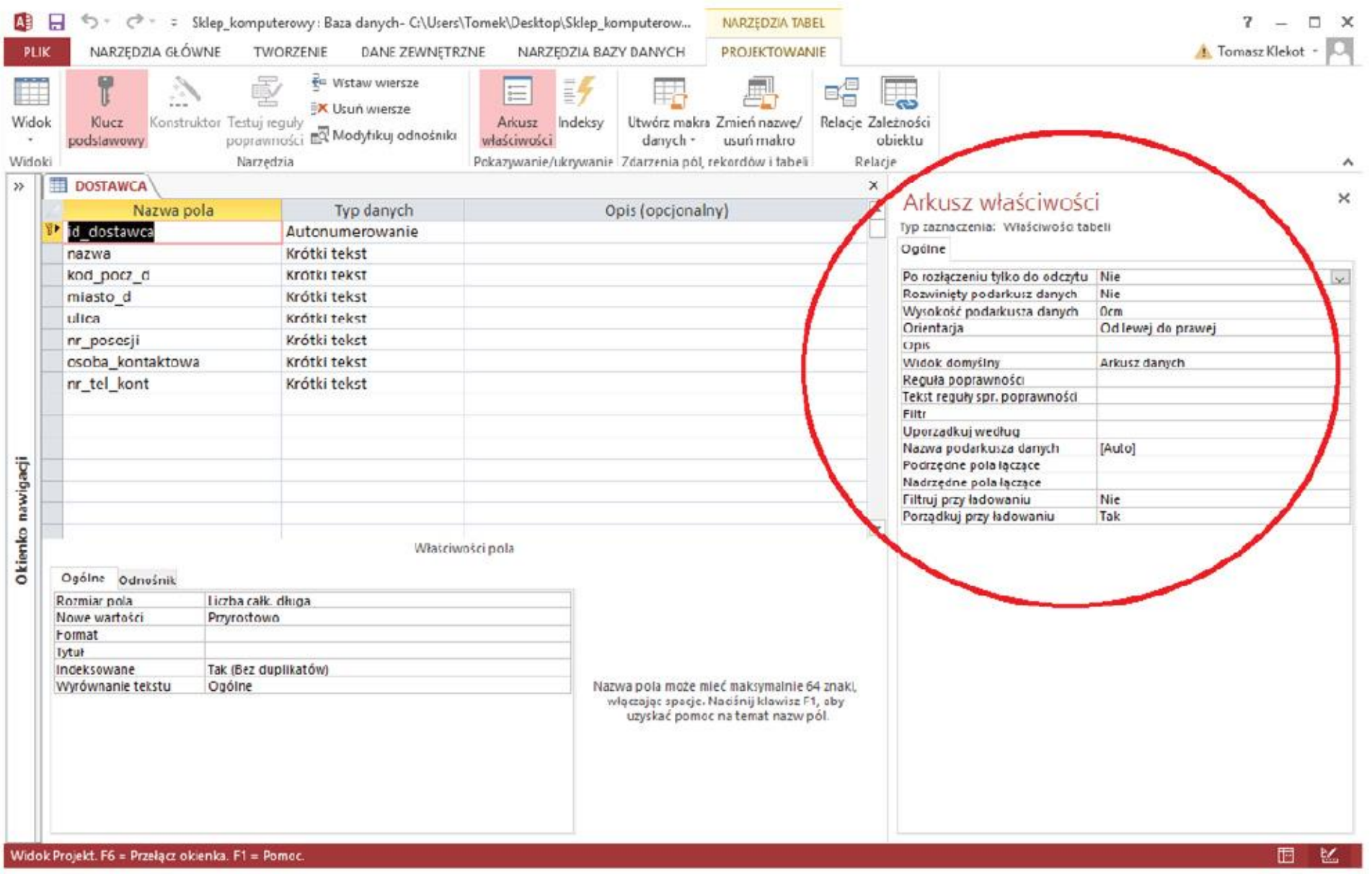
Wyrównanie tekstu określa domyślne wyrównanie tekstu w obrębie formantu.

Podpis to tekst etykiety wyświetlany domyślnie dla tego pola w formularzach, raportach i zapytaniach. Jeśli ta właściwość jest pusta, zostanie użyta nazwa pola. Dozwolony jest dowolny ciąg tekstowy.

Miejsca dziesiętne określają liczbę miejsc dziesiętnych używanych do wyświetlania liczb.

Tryb edytora IME steruje konwersją znaków we wschodnioazjatyckich wersjach systemu Windows.

W bazie danych programu MS Access można również zdefiniować właściwości tabeli. Właściwości tabeli są cechami tabeli, które mają wpływ na wygląd lub „zachowanie" tabeli jako całości. Właściwości tabeli formatuje się w arkuszu właściwości danej tabeli w **`widoku projektu`** (rys. 9.5).



Rys. 9.5. Arkusz właściwości tabeli

Klucze główne i obce

Podczas tworzenia tabel kluczowym etapem pracy jest zdefiniowanie **klucza podstawowego** (ang. *primary key*). Każda tabela może mieć tylko jeden klucz podstawowy. W MS Access automatycznie tworzą się pola klucza podstawowego zaraz po utworzeniu tabeli w pierwszym wierszu. Można również określić samodzielnie pola, które mają być użyte jako klucz główny – za pomocą ikony klucza na pasku narzędziowym (rys. 9.5).

Jeżeli tabela uczestniczy w relacji z inną tabelą, powinna mieć pola o funkcji klucza obcego. Pole klucza obcego musi mieć taką samą nazwę jak w tabeli macierzystej. Każda tabela może mieć wiele kluczy obcych.

Pola obliczeniowe

W programie MS Access można utworzyć pola tabeli służące do wykonywania obliczeń na wprowadzonych wartościach. Obliczenia mogą zawierać wartości z pól w tej samej tabeli, a także wbudowanych funkcji programu MS Access. Edycję wyrażeń obliczeniowych przeprowadza się za pomocą **`Konstruktora wyrażeń`**. Obliczenie nie może obejmować pól z innych tabel ani zapytań, a wyniki obliczeń są tylko do odczytu.

UWAGA

Należy pamiętać, że pola obliczeniowe są dostępne w plikach bazy danych o formacie accdb w wersjach Access 2010 i nowszych.

PRZYKŁAD 9.3

Pole obliczeniowe, które oblicza **wartość_zakupu** z komórek **ilość** i **cena_jednostkowa**, może wyglądać następująco:

```
wartość_zakupu: [ilość]*[cena_jednostkowa]
```

Konstruktor wyrażeń

Wyrażenie w MS Access to formuła składająca się z operatorów matematycznych i logicznych, stałych, funkcji, pól tabel, kontrolek i właściwości, której wynikiem jest pojedyncza wartość. Wyrażenia w programie Access mogą służyć do obliczania wartości w polach obliczeniowych, do sprawdzania poprawności danych i do ustawiania wartości domyślnej.

Konstruktor wyrażeń to narzędzie pozwalające na edycję wyrażeń oraz wykorzystanie wbudowanych funkcji. Na rys. 9.6 pokazano okno konstruktora wyrażeń. Do jego podstawowych elementów zalicza się:

- **pole wyrażenia** (1) – edytuje się w nim wyrażenie;
- **listę elementów wyrażeń** (2) – należą do nich identyfikatory pól tabel lub formanty w formularzach, funkcje, stałe oraz operatory;
- **listę kategorii wyrażeń** (3) – lista poszczególnych składowych wybranego elementu zawierających określone wartości;
- **listę wartości wyrażeń** (4) – po wyborze konkretnej wartości znajdzie się ona w konstruowanym wyrażeniu.

W wyrażeniach odnoszących się do tabeli identyfikuje się ją przez wpisanie jej nazwy w nawias kwadratowy. Jeżeli w wyrażeniu odnosi się do pola w tabeli, nazwy tych dwóch obiektów rozdziela się kropką.

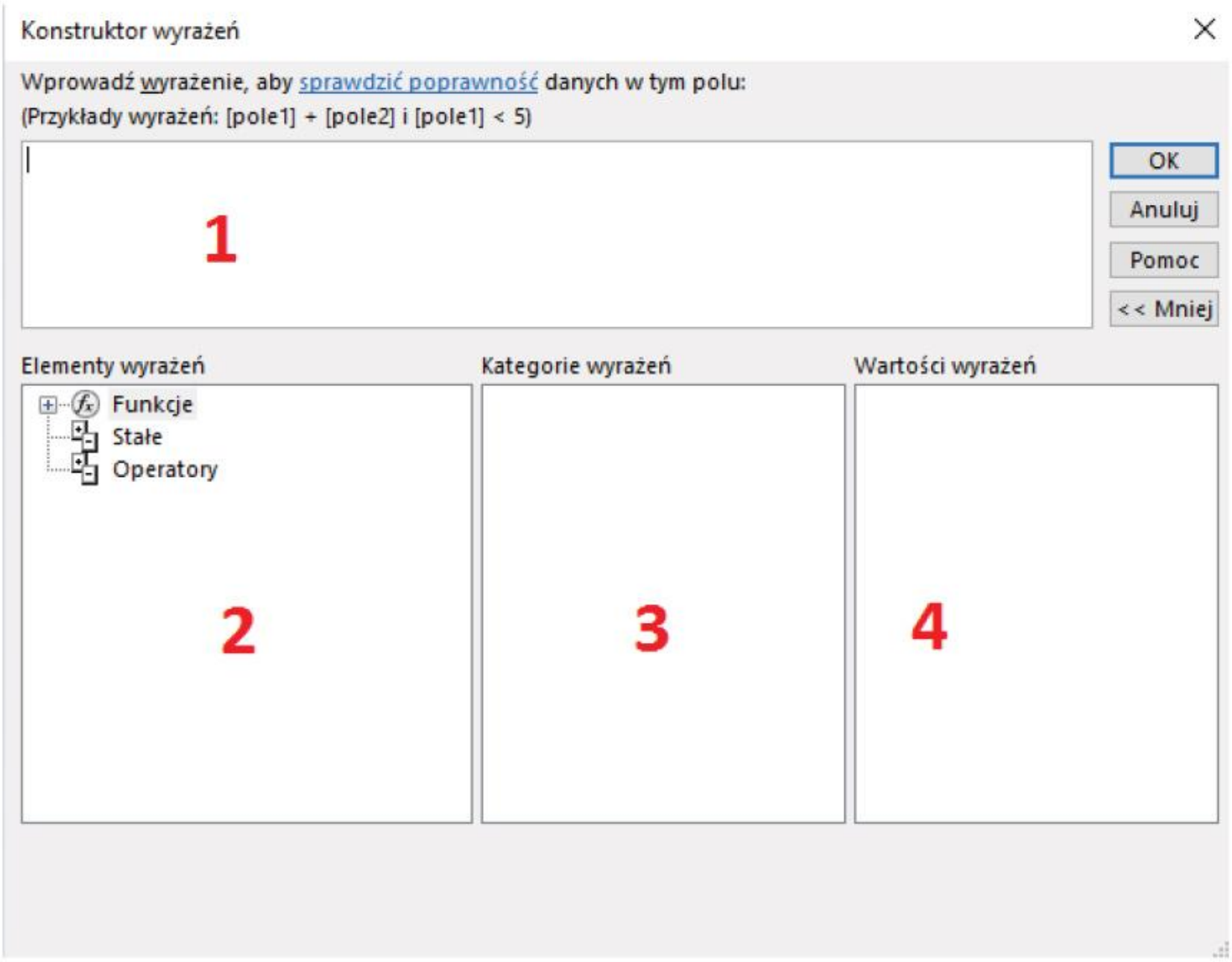
```
= [nazwa tabeli].[nazwa pola]
```

Wartości w wyrażeniach używa się w postaci liczb, tekstu lub typu data / godzina i zapisuje je według przykładu:

```
1392   'Warszawa'   "Łódź"   #03.11.2017#
```

Do tworzenia wyrażeń korzysta się z operatorów. W MS Access są do dyspozycji:

- operatory arytmetyczne (`+, -, *, /`),
- operatory porównań (`<, >, =`),
- operatory tekstowe (`&`),
- operatory logiczne (`not`, `or`, `like` itp.).



Rys. 9.6. Okno konstruktora wyrażeń

Dodatkowo można korzystać z wbudowanych funkcji o określonych zastosowaniach. Po nazwie funkcji w nawiasach zwykłych należy podać określone argumenty funkcji. Są też funkcje niemające argumentów, np. `Date()`.

```
= Nazwa funkcji(argument)
```

PRZYKŁAD 9.4

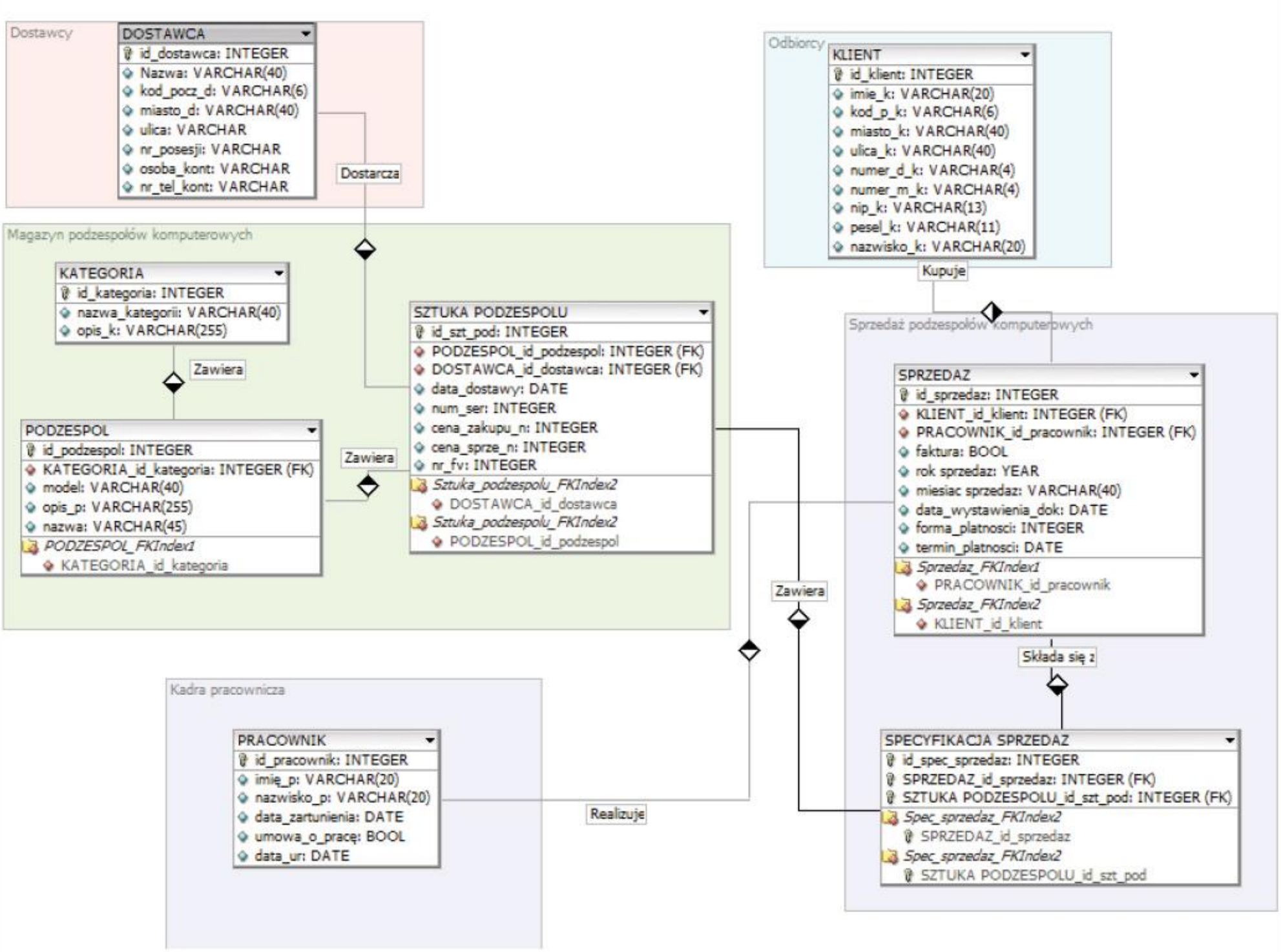
Wyrażenie	Wynik działania
`LIKE „K*”`	Wyświetla wszystkie nazwy z danego pola zaczynające się na literę K o dowolnej długości
`=[Cena]*1.23`	Zwróci cenę brutto (przy VAT = 23 %)
`=Avg ([wiek])`	Funkcja wyświetla średnią pola **Wiek**
`=Date()`	Funkcja wyświetla bieżącą datę
`Not „Warszawa”`	Wyświetla wszystkie nazwy miast z danego pola inne niż **Warszawa**
`=[Imie]&[Nazwisko]`	Wyświetla wartości znajdujące się w polach **Imię** i **Nazwisko**

Tworzenie tabel według projektu

Punktem wyjściowym do fizycznego tworzenia tabel w MS Access jest projekt struktury, w którym często zaplanowano już nazwy tabel, liczbę oraz nazwy pól, a także ich podstawowe właściwości. Przebieg tych czynności prześledzimy na przykładzie bazy danych **`Sklep_komputerowy`**.

PRZYKŁAD 9.5

Na rys. 9.7 przedstawiono projekt koncepcyjny bazy danych Sklep_komputerowy, który powstał za pomocą programu DBDesigner. Naszym zadaniem będzie utworzenie na jego podstawie fizycznej struktury bazy danych w MS Access.



Rys. 9.7. Diagram ER bazy danych Sklep_komputerowy

Na podstawie diagramu utworzono specyfikacje tabel, w których zdefiniowano typy danych dostępne w MS Access oraz określono inne właściwości. Jest to konieczne ze względu na różnice między nazewnictwem stosowanym w języku SQL a nazewnictwem MS Access. Każdą encję i opisujące ją atrybuty zamieszczono w oddzielnej tabeli o określonej, unikatowej nazwie zgodnej z projektem.

Tabela 9.3. Właściwości tabeli KLIENT

KLIENT							
Nazwa pola	Właściwości pól w MS Access						
	klucz główny	typ danych	rozmiar	maska	format	wymagane / indeksowanie	opis
id_klient	tak	autonumero-wanie	domyślny	domyślna	domyślny	wymagane, indeksowane bez powtórzeń	
imie_k	nie	krótki tekst	20 znaków	>L<??????????????????	domyślny	wymagane, indeksowane z powtórzeniami	
nazwisko_k	nie	krótki tekst	20 znaków	>L<??????????????????	domyślny	wymagane, indeksowane z powtórzeniami	
kod_p_k	nie	krótki tekst	6 znaków	00-000	domyślny	wymagane, indeksowane z powtórzeniami	kod pocztowy
miasto_k	nie	krótki tekst	40 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	
ulica_k	nie	krótki tekst	40 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	
numer_d_k	nie	krótki tekst	4 znaki	&CCC	domyślny	nie	numer domu
numer_m_k	nie	liczba	4 znaki	0999	domyślny	nie	numer mieszkania
nip_k	nie	krótki tekst	13 znaków	000-00-00-000	domyślny	nie	NIP, jeżeli jest to firma
pesel_k	nie	krótki tekst	11 znaków	00000000000	domyślny	nie	PESEL, jeżeli jest to osoba fizyczna

Tabela 9.4. Właściwości tabeli PRACOWNIK

PRACOWNIK							
Nazwa pola	Właściwości pól w MS Access						
	klucz główny	typ danych	rozmiar	maska	format	wymagane / indeksowanie	opis
id_ pracownik	tak	autonumero-wanie	domyślny	domyślna	domyślny	wymagane, indeksowane bez powtórzeń	
imie_p	nie	krótki tekst	20 znaków	>L<??????????????????	domyślny	wymagane, indeksowane z powtórzeniami	
nazwisko_p	nie	krótki tekst	20 znaków	>L<??????????????????	domyślny	wymagane, indeksowane z powtórzeniami	
data_ zatrudnienia	nie	data / godzina	brak	99/99/9999;0;_	dd-MM-rrrr	wymagane, indeksowane z powtórzeniami	
umowa_o_ pracę	nie	tak / nie	brak	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	informacja o rodzaju umowy
data_ur	nie	krótki tekst	brak	99/99/9999;0;_	dd-MM-rrrr	wymagane, indeksowane z powtórzeniami	data urodzenia

Tabela 9.5. Właściwości tabeli KATEGORIA

KATEGORIA							
Nazwa pola	Właściwości pól w MS Access						
	klucz główny	typ danych	rozmiar	maska	format	wymagane / indeksowanie	opis
id_kategoria	tak	autonumero-wanie	domyślny	domyślna	domyślny	wymagane, indeksowane bez powtórzeń	
nazwa_ kategorii	nie	krótki tekst	40 znaków	domyślna	domyślny	nie	
opis_k	nie	długi tekst	domyślny	domyślna	domyślny	nie	charakterystyka danej kategorii

Tabela 9.6. Właściwości tabeli PODZESPOL

PODZESPOL							
Nazwa pola	Właściwości pól w MS Access						
	klucz główny	typ danych	rozmiar	maska	format	wymagane / indeksowanie	opis
id_podzespol	tak	autonumero-wanie	domyślny	domyślna	domyślny	wymagane, indeksowane bez powtórzeń	
nazwa	nie	krótki tekst	45 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	
model	nie	krótki tekst	40 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	
opis_p	nie	długi tekst	domyślny	domyślna	domyślny	nie	opis parametrów technicznych
id_kategoria	klucz obcy	liczba	domyślny	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	klucz główny z tabeli KATEGORIA

Tabela 9.7. Właściwości tabeli SZTUKA_PODZESPOLU

SZTUKA_PODZESPOLU							
Nazwa pola	Właściwości pól w MS Access						
	klucz główny	typ danych	rozmiar	maska	format	wymagane / indeksowanie	opis
id_szt_pod	tak	autonumero-wanie	domyślny	domyślna	domyślny	wymagane, indeksowane bez powtórzeń	
num_ser	nie	krótki tekst	20 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	numer seryjny pojedynczej sztuki
nr_fv	nie	krótki tekst	10 znaków	&&&&&&&&&&	domyślny	wymagane, indeksowane z powtórzeniami	numer faktury dostawcy
data_ dostawy	nie	data / godzina	brak	99/99/9999;0;_	dd-MM-rrrr	wymagane, indeksowane z powtórzeniami	data przyjęcia na magazyn
cena_ zakupu_n	nie	waluta	domyślny	domyślna	walutowy	wymagane, indeksowane z powtórzeniami	cena zakupu netto w złotych
cena_ sprzedaży_n	nie	waluta	domyślny	domyślna	walutowy	wymagane, indeksowane z powtórzeniami	cena sprzedaży netto w złotych
id_podzespol	klucz obcy	liczba	domyślny	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	klucz główny z tabeli PODZESPOL
id_dostawca	klucz obcy	liczba	domyślny	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	klucz główny z tabeli DOSTAWCA

Tabela 9.8. Właściwości tabeli SPRZEDAZ

SPRZEDAZ							
Nazwa pola	Właściwości pól w MS Access						
	klucz główny	typ danych	rozmiar	maska	format	wymagane / indeksowanie	opis
id_sprzedaz	tak	autonumero-wanie	domyślny	domyślna	domyślny	wymagane, indeksowane bez powtórzeń	jednocześnie numer fv
faktura	nie	tak / nie	brak	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	informacja, czy wystawia się fakturę
rok_sprzedaz	nie	liczba	4 znaki	0000	domyślny	wymagane, indeksowane z powtórzeniami	sam rok sprzedaży
miesiac_ sprzedaz	nie	krótki tekst	11	LLL????????	domyślny	wymagane, indeksowane z powtórzeniami	sam miesiąc sprzedaży
data_wysta-wienia_dok	nie	data/godzina	brak	99/99/9999;0;_	dd-MM-rrrr	wymagane, indeksowane z powtórzeniami	inaczej data sprzedaży
forma_ platnosci	nie	krótki tekst	20 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	opis typu: gotówka, przelew itp.
termin_ platnosci	nie	liczba	2 znaki	99	domyślny	wymagane, indeksowane z powtórzeniami	liczba dni na zapłatę
id_klient	klucz obcy	liczba	domyślny	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	klucz główny z tabeli KLIENT
id_pracownik	klucz obcy	liczba	domyślny	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	klucz główny z tabeli PRACOWNIK

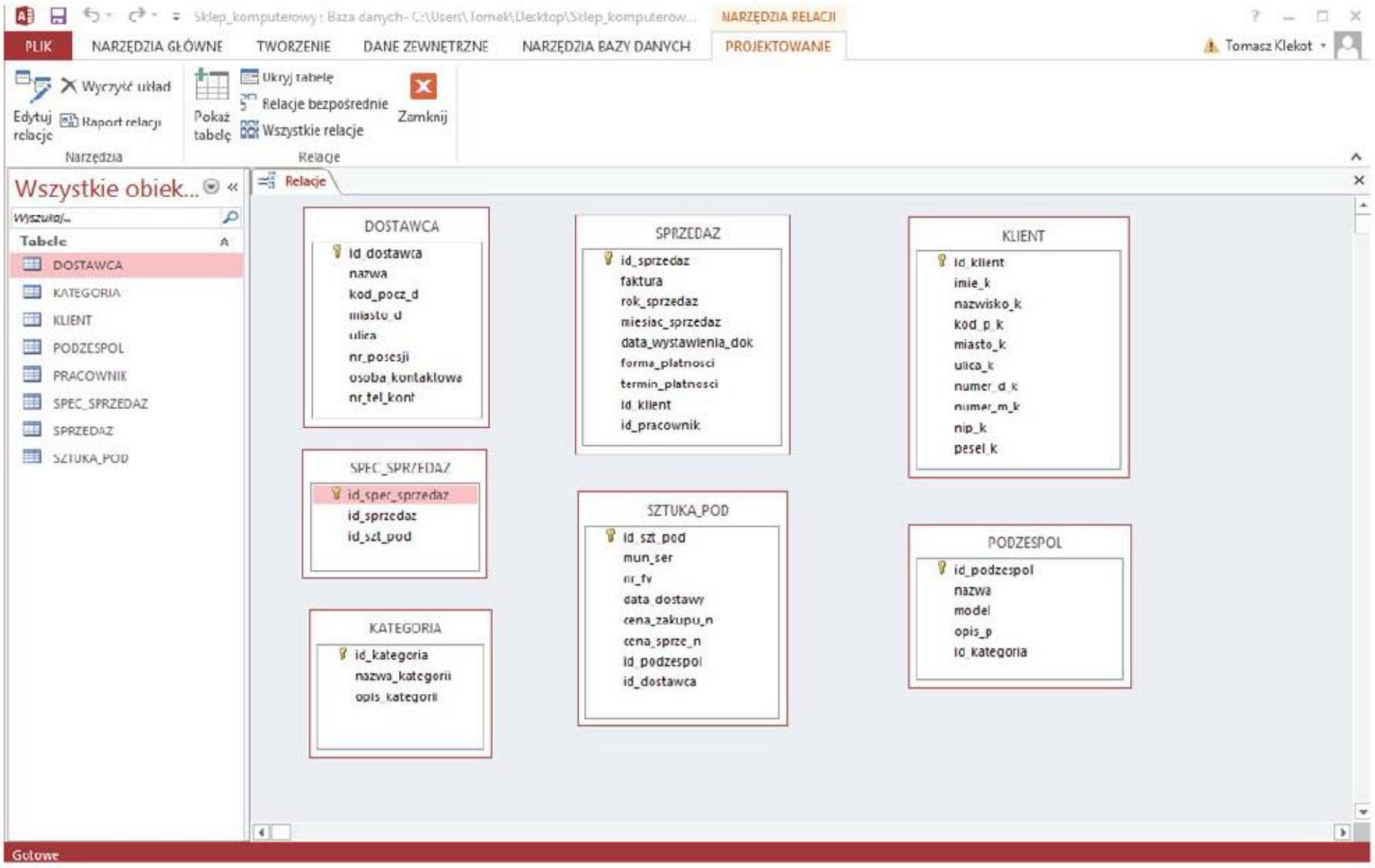
Tabela 9.9. Właściwości tabeli SPEC_SPRZEDAZ

SPEC_SPRZEDAZ							
Nazwa pola	Właściwości pól w MS Access						
	klucz główny	typ danych	rozmiar	maska	format	wymagane / indeksowanie	opis
id_spec_sprzedaz	tak	autonumero-wanie	domyślny	domyślna	domyślny	wymagane, indeksowane bez powtórzeń	
id_sprzedaz	klucz obcy	liczba	domyślny	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	klucz główny z tabeli SPRZEDAZ
id_szt_podzespolu	klucz obcy	liczba	domyślny	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	klucz główny z tabeli SZTUKA_PODZESPOLU

Tabela 9.10. Właściwości tabeli DOSTAWCA

DOSTAWCA							
Nazwa pola	Właściwości pól w MS Access						
	klucz główny	typ danych	rozmiar	maska	format	wymagane / indeksowanie	opis
id_dostawca	tak	autonumero-wanie	domyślny	domyślna	domyślny	wymagane, indeksowane bez powtórzeń	
nazwa	nie	krótki tekst	40 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	
kod_pocz_d	nie	krótki tekst	6 znaków	00-000	domyślny	wymagane, indeksowane z powtórzeniami	kod pocztowy
miasto_d	nie	krótki tekst	40 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	
ulica	nie	krótki tekst	40 znaków	domyślna	domyślny	wymagane, indeksowane z powtórzeniami	
nr_posesji	nie	liczba	4 znaki	9999	domyślny	wymagane, indeksowane z powtórzeniami	numer domu
osoba_ kontaktowa	nie	krótki tekst	40 znaków	domyślna	domyślny	nie	imię i nazwisko osoby do kontaktu
nr_tel_ kontaktowy	nie	krótki tekst	16 znaków	(#00)000-000-000	domyślny	nie	numer telefonu osoby do kontaktu

Na podstawie informacji zawartych w diagramie ER oraz tabelach przystępujemy do tworzenia struktury fizycznej tabel. Otwieramy nowy plik bazy danych Access dla komputerów stacjonarnych i nadajemy mu nazwę **`Sklep_komputerowy`**. Następnie tworzymy kolejne tabele oraz pola z własnościami określonymi w poszczególnych wzorach. Po zakończeniu edycji powinniśmy otrzymać strukturę tabel przedstawioną na rys. 9.8.



Rys. 9.8. Struktura fizyczna tabel w MS Access

Po utworzeniu struktury tabel możemy przystąpić do kolejnych etapów, które zostaną opisane w dalszej części podręcznika.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jaką rolę w tworzeniu bazy danych odgrywa maska wprowadzania?
2. Wymień symbole stosowane do budowy maski wprowadzania i scharakteryzuj ich działanie.
3. W jakim celu stosuje się pola obliczeniowe w tabelach bazy danych Access?
4. Kiedy wykorzystuje się narzędzie Konstruktor wyrażeń?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz folder na pulpicie o nazwie zgodnej z twoim imieniem i nazwiskiem oraz ustaw go jako domyślną lokalizację, w której będą przechowywane pliki bazy danych. Dodaj go również do zaufanych lokalizacji programu MS Access.
2. Zaproponuj składnię maski wprowadzania dla następujących pól tabeli:
   - PESEL,
   - numer telefonu komórkowego,
   - kod pocztowy,
   - numer dowodu osobistego.
3. Na rysunku (s. 98) przedstawiono informacje na temat płyt analogowych różnych wykonawców. Na podstawie tych informacji utwórz bazę danych płyt analogowych o nazwie **Vinyle** oraz wykonaj następujące czynności:
   - wykonaj normalizację tabeli oraz diagram ER pokazujący projekt struktury znormalizowanej bazy danych wraz z ustalonymi powiazaniami;
   - utwórz strukturę fizyczną tabel – zaprojektuj właściwości pól oraz maski wprowadzania i reguły poprawności;
   - przedstaw nauczycielowi efekt pracy do oceny.
4. Zaprojektuj strukturę pól dla tabeli opisującej dyski twarde HDD i SSD tak, aby opisać jak najwięcej cech tych podzespołów komputerowych. Zwróć szczególną uwagę na maski wprowadzania.
5. Utwórz strukturę tabel i ich właściwości z przykładu 9.4 w programie MS Access.

Artysta	Kraj pochodzenia	Numer katalogowy	Tytuł płyty	Format	Rok wydania	Wydawca	Cena	Liczba sztuk	Kategoria
KORN	USA	MOVLP109	Issues	2LP	10-01-2010	Epic	124,00	6	Metal
KORN	USA	MOVLP1157	Korn	2LP	01-03-2014	Music on Vinyl	129,50	23	Metal
BLACK SABBATH	Wielka Brytania	LP414939, MOVLP223	Sabbath Bloody Sabbath	LP	01-12-1973, 20-02-1974	Sanctuary Records, Epic	64,99	12	Rock
BURNING LOVE	Kanada	ANX009	Black Widow	7EP	20-09-2012	High Anxiety	30,00	43	Punk
BLACK SABBATH	Wielka Brytania	LPA541493	Sabotage	LP	12-07-1975	Sanctuary Records	85,50	1	Rock
THE POLICE	Wielka Brytania	LP3947531	Outlandos D'amour	LP	08-12-2014	A&M Records	112,20	9	Rock
ADELE	Wielka Brytania	XLCD 520	21	LP	20-01-2011	XL Recordings	143,23	2	Pop

10 Relacje – powiązania między tabelami

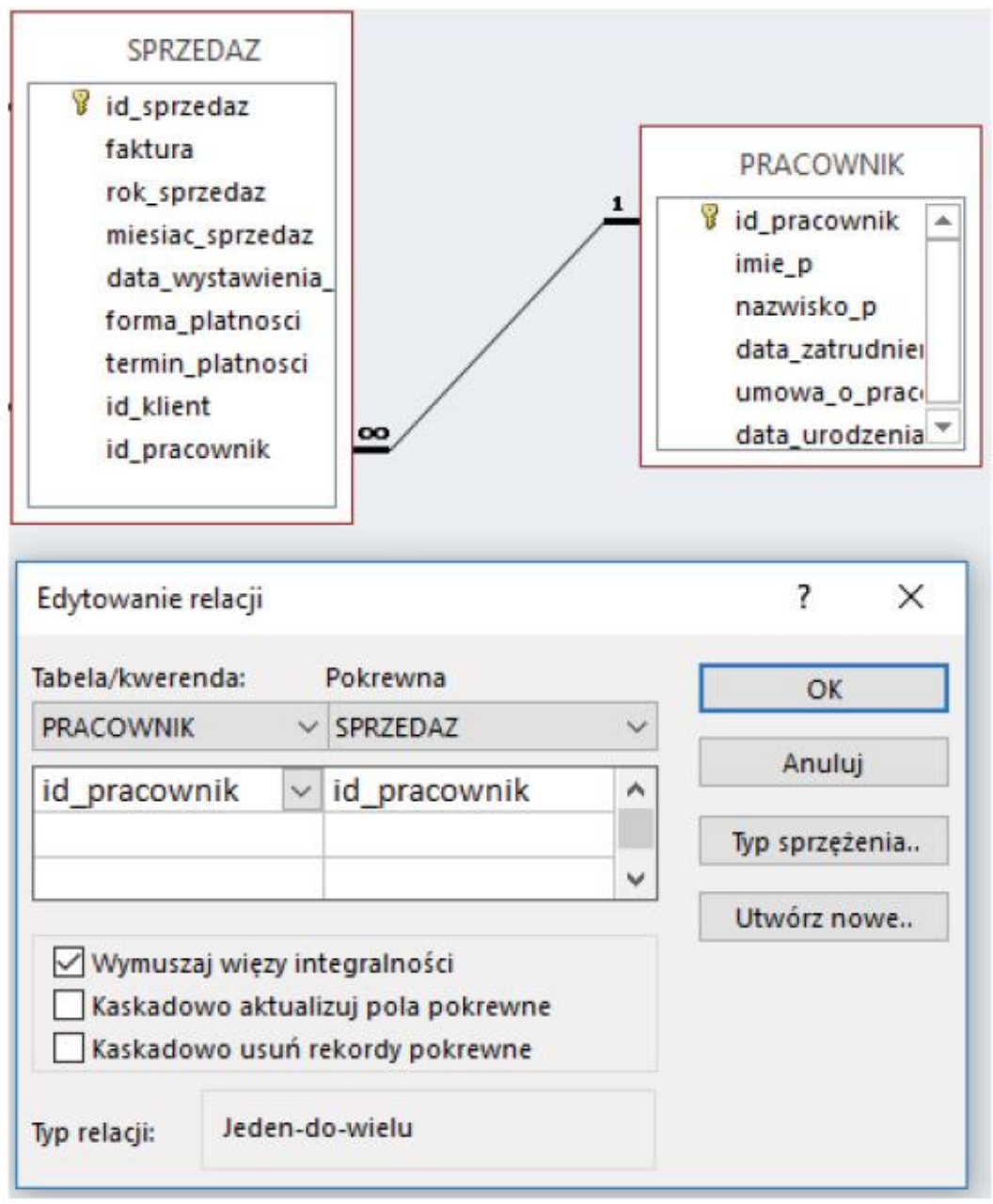
ZAGADNIENIA

- Tworzenie relacji między tabelami
- Edycja utworzonych relacji

Aby baza danych działała w sposób spójny i efektywny, między utworzonymi tabelami należy utworzyć relacje. Relacje definiuje się zgodnie z założeniami zawartymi w projekcie.

W MS Access są dostępne następujące liczności relacji: jeden do jednego, jeden do wielu i wiele do wielu. Relacje definiuje się w zakładce **`Narzędzia bazy danych / Relacje`**.

Relacje tworzy się przez dodanie – do obszaru roboczego relacji – utworzonych wcześniej tabel, między którymi mają powstać powiązania. Następnie rysuje się linie powiązań – przeciąga się kursor myszy między polami dwóch tabel. Po naciśnięciu lewego kursora myszy na linii powiązania można otworzyć okno **`Edytowanie relacji`**.



Rys. 10.1. Przykład widoku relacji z zaznaczoną opcją wymuszania więzów integralności

Okno **Edytowanie relacji** umożliwia zmianę relacji między tabelami. Dzięki niemu można zmienić tabele, kwerendy lub pola po dowolnej stronie relacji. Ponadto można ustawić typ sprzężenia lub wymusić więzy integralności i wybrać opcję operacji kaskadowych. Typ sprężenia określa rodzaj powiązania rekordów tabel zawierających tak samo zdefiniowane wartości. Więzy integralności to reguły wymuszające właściwe związki między rekordami powiązanych tabel. Uniemożliwiają one przypadkowe usunięcie powiązanych rekordów. Wpływa to na spójność bazy danych.

Aby zapobiec powstawaniu rekordów odłączonych, czyli takich, które pozostają po usunięciu powiązanych rekordów w innych tabelach, zaznacza się opcję **Wymuszaj więzy integralności**. Jeżeli zaznaczy się tę opcję, linia relacji będzie grubsza na obu końcach. Ponadto nad grubszym odcinkiem po jednej stronie linii relacji zostanie wyświetlona liczba 1, a nad grubszym odcinkiem po drugiej stronie linii – symbol nieskończoności „∞” (rys. 10.1).

Trzeba jednak pamiętać, że podczas tworzenia relacji jeden do jednego oba pola wspólne (zazwyczaj pola klucza podstawowego w jednej tabeli i klucza obcego w tabeli powiązanej) muszą mieć unikatowy indeks. Oznacza to, że właściwość pola **Indeksowane** powinna być ustawiona na **Tak (Bez duplikatów)**.

Inaczej będzie podczas tworzenia relacji jeden do wielu. Przy takiej liczności relacji w tabeli powiązanej (z kluczem obcym) należy ustawić właściwość pola **Indeksowane** na **Tak (z duplikatami)**.

Pole **Typ sprzężenia** w oknie **Edytowanie relacji** określa, które rekordy mają być uwzględnione w wyniku wykonania dowolnej kwerendy odnoszącej się do powiązanych tabel. Kwerendy zostaną dokładniej omówione w dalszej części podręcznika.

Aby utworzyć relację wiele-do-wielu, należy utworzyć dodatkową tabelę, tzw. tabelę skrzyżowań lub tabelę łączącą, która dzieli relację wiele-do-wielu na dwie relacje jeden-do-wielu. Do tej dodatkowej tabeli wstawia się klucze podstawowe z obu pierwotnych tabel. W efekcie trzecia tabela rejestruje każde wystąpienie tej relacji.

! UWAGA

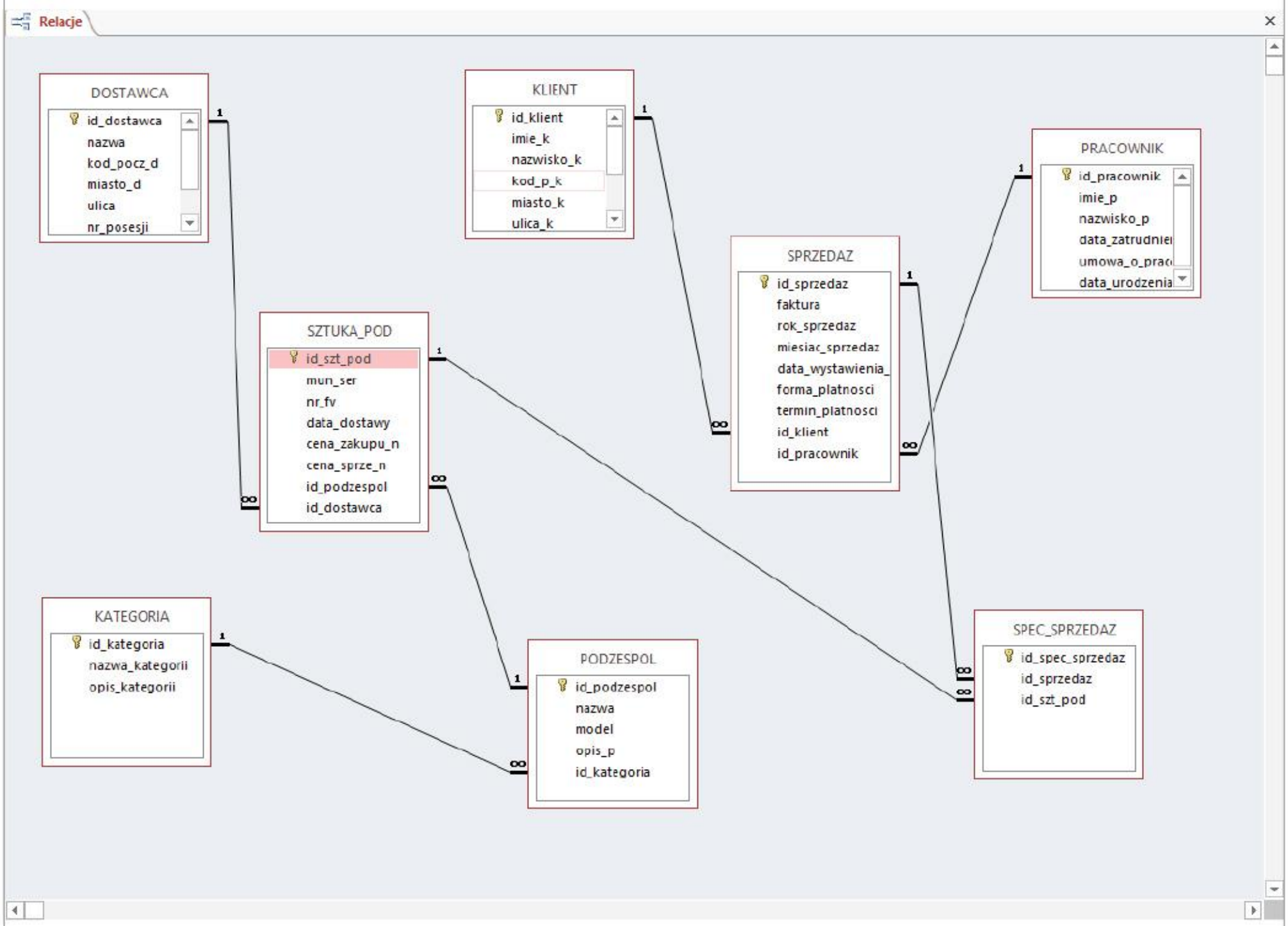
Okno **Relacje** nie jest dostępne w aplikacji sieci Web programu MS Access. Zamiast relacji w aplikacji sieci Web programu Access należy utworzyć pole odnośnika, które pobiera wartości z pola pokrewnego w innej tabeli.

PRZYKŁAD 10.1

Utworzymy relacje między tabelami bazy danych **Sklep_komputerowy**. Podczas ich tworzenia musimy pamiętać o następujących zależnościach:

1. Jedno zdarzenie sprzedaży może zawierać wiele rekordów specyfikacji sprzedaży, a dana specyfikacja – dotyczyć tylko jednej sprzedaży. Utworzymy tutaj relację 1 : N.

2. Jedna sprzedaż jest związana z jednym klientem, ale jeden klient może dokonać zakupu wiele razy. Oznacza to relację 1 : N.
3. Jeden podzespół może należeć tylko do jednej kategorii, natomiast do danej kategorii może należeć wiele podzespołów. Oznacza to relację 1 : N.
4. Możemy mieć w magazynie wiele sztuk danego podzespołu z indywidualnym numerem seryjnym. Utworzymy relację 1 : N.
5. W każdej specyfikacji może wystąpić wiele sztuk różnych podzespołów, ale konkretna sztuka może należeć tylko do jednej specyfikacji sprzedaży. Oznacza to relację 1 : N.
6. Każda sztuka pochodzi od konkretnego dostawcy, ale dostawca może dostarczać wiele sztuk różnych podzespołów. Oznacza to relację 1 : N.
7. Dana sprzedaż jest prowadzona przez jednego pracownika, ale może on realizować wiele sprzedaży w danym czasie. Utworzymy tutaj relację 1 : N.



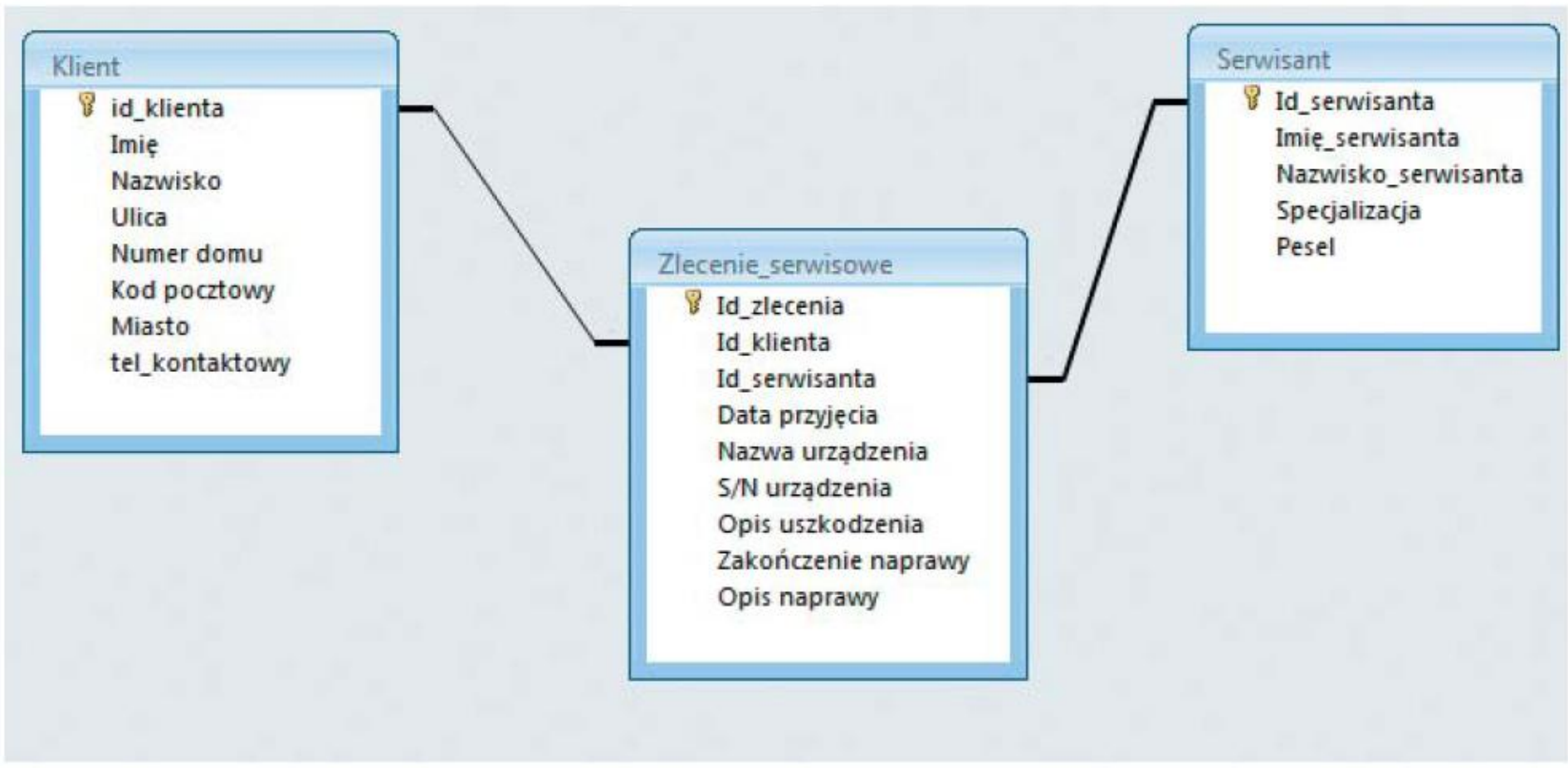
Rys. 10.2. Relacje między tabelami w bazie danych **`Sklep_komputerowy`**

SPRAWDŹ SWOJĄ WIEDZĘ

1. Na podstawie konkretnych przykładów opisz rodzaje relacji w MS Access.
2. Jaką funkcję podczas tworzenia relacji pełni tabela skrzyżowań?
3. Podaj nazwę relacji, która powinna wystąpić między tabelami o następujących nazwach:
   - Książka – Wydawnictwo,
   - Uczeń – Klasa,
   - Obywatel – Numer PESEL.
4. Co to są więzy integralności?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Sprawdź na dowolnej relacji między tabelami, jakie są typy sprzężenia relacji.
2. Utwórz bazę danych **`Serwis`** z tabelami według schematu. Zdefiniuj samodzielnie właściwości poszczególnych pól tak, aby zachodziły wskazane powiązania. Określ liczebność relacji między tabelami. Efekt pracy przedstaw nauczycielowi do oceny.



11 Wprowadzanie danych do bazy. Operacje na danych

ZAGADNIENIA

- Importowanie danych do bazy
- Łączenie bazy danych z zewnętrznym źródłem danych
- Eksportowanie danych do plików zewnętrznych

Wprowadzanie danych do tabeli

Po stworzeniu struktury pól w tabeli bazy danych MS Access oraz zdefiniowaniu relacji można przejść do edycji danych. Wykonuje się to przez zapisanie zmian w tabeli i wybór opcji **`Widok arkusza danych`**.

Najczęściej jednak dane są importowane z zewnętrznych źródeł danych lub wprowadzane za pomocą specjalnie do tego celu tworzonych **formularzy**. Zasady projektowania, tworzenia oraz wykorzystania formularzy do wprowadzania danych zostaną opisane w następnej lekcji.

Import danych

Dane można nie tylko edytować, lecz także pobierać ze źródeł zewnętrznych. W programie MS Access istnieje możliwość dostępu do danych, które pochodzą z innych baz danych lub są zapisane w plikach. Można w takich sytuacjach wykonać operacje importowania danych. Importowanie nie powoduje zastąpienia ani zmodyfikowania istniejących tabel i obiektów. Jeżeli w docelowej bazie danych istnieje już obiekt o takiej samej nazwie jak obiekt źródłowy, program MS Access dołącza do nazwy importowanego obiektu odpowiedni, kolejny numer.

Można importować dane z plików:

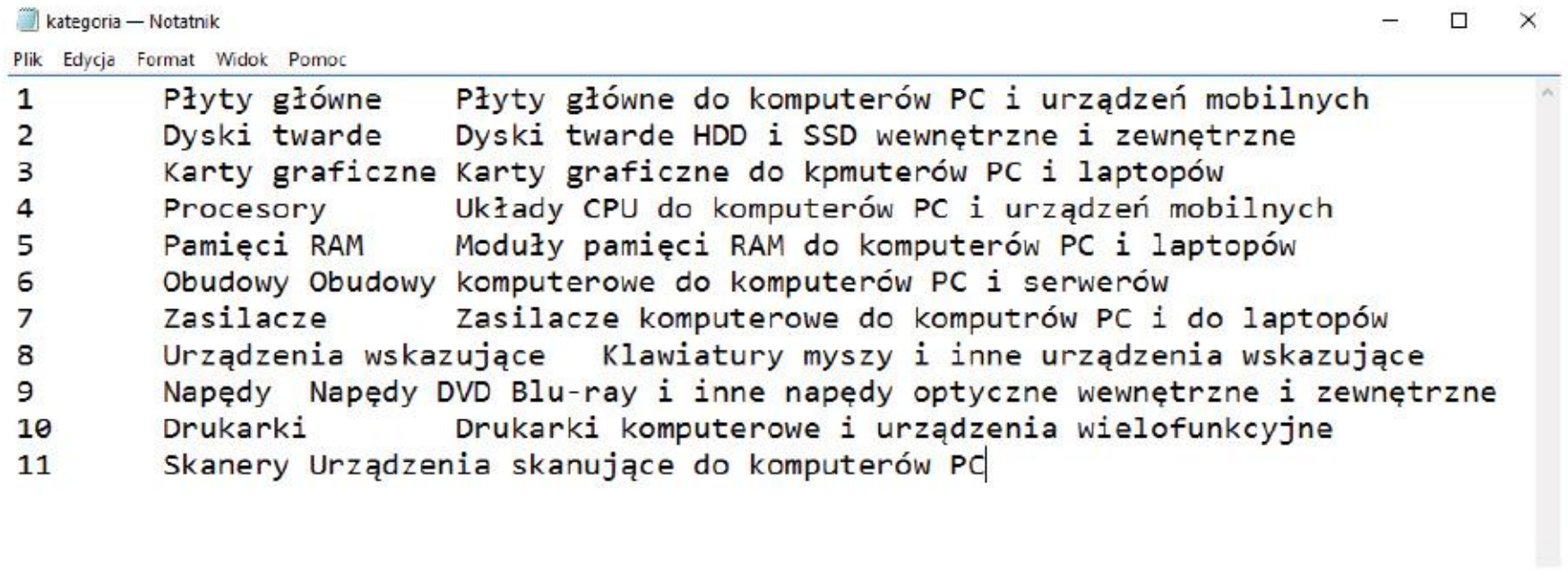
- MS Excel,
- MS Access,
- tekstowych z wartościami oddzielonymi określonymi ogranicznikami,
- list programu SharePoint,
- XML,
- HTML,
- formatu Outlook.

Aby importować dane z pliku tekstowego, trzeba zacząć od odpowiedniego przygotowania pliku źródłowego. Wyróżnia się **pliki rozdzielane** oraz **stałej szerokości**. W plikach rozdzielanych stosuje się **ograniczniki** rozdzielające kolejne wartości. Ogranicznikiem mogą

być: tabulator, średnik, przecinek lub spacja. Należy jednak pamiętać, że nie można użyć ogranicznika, którego symbol jest wykorzystywany w polach bazy danych jako separator wartości. W plikach stałej szerokości każdy rekord występuje w osobnym wierszu, a szerokość każdego pola jest stała we wszystkich rekordach.

PRZYKŁAD 11.1

W naszym przykładzie przygotujemy plik tekstowy zawierający wartości pól tabeli **`Kategoria`** z bazy danych **`Sklep_komputerowy`**. Zastosujemy plik rozdzielany, a jako ogranicznik wykorzystamy tabulację. Wynik naszej pracy przedstawiono na rys. 11.1.



Rys. 11.1. Przygotowany plik ze źródłem danych

Inną metodą jest zastosowanie plików tekstowych o stałej szerokości. Wtedy każdy rekord zapisuje się w oddzielnym wierszu. Pola w każdym rekordzie muszą mieć stałą szerokość, która pozwala na wpisanie danej wartości. Jeżeli szerokość pola nie jest wystarczająca, zwiększa się ją przez wstawienie spacji (rys. 11.2).

Rys. 11.2. Przygotowany plik o stałej szerokości

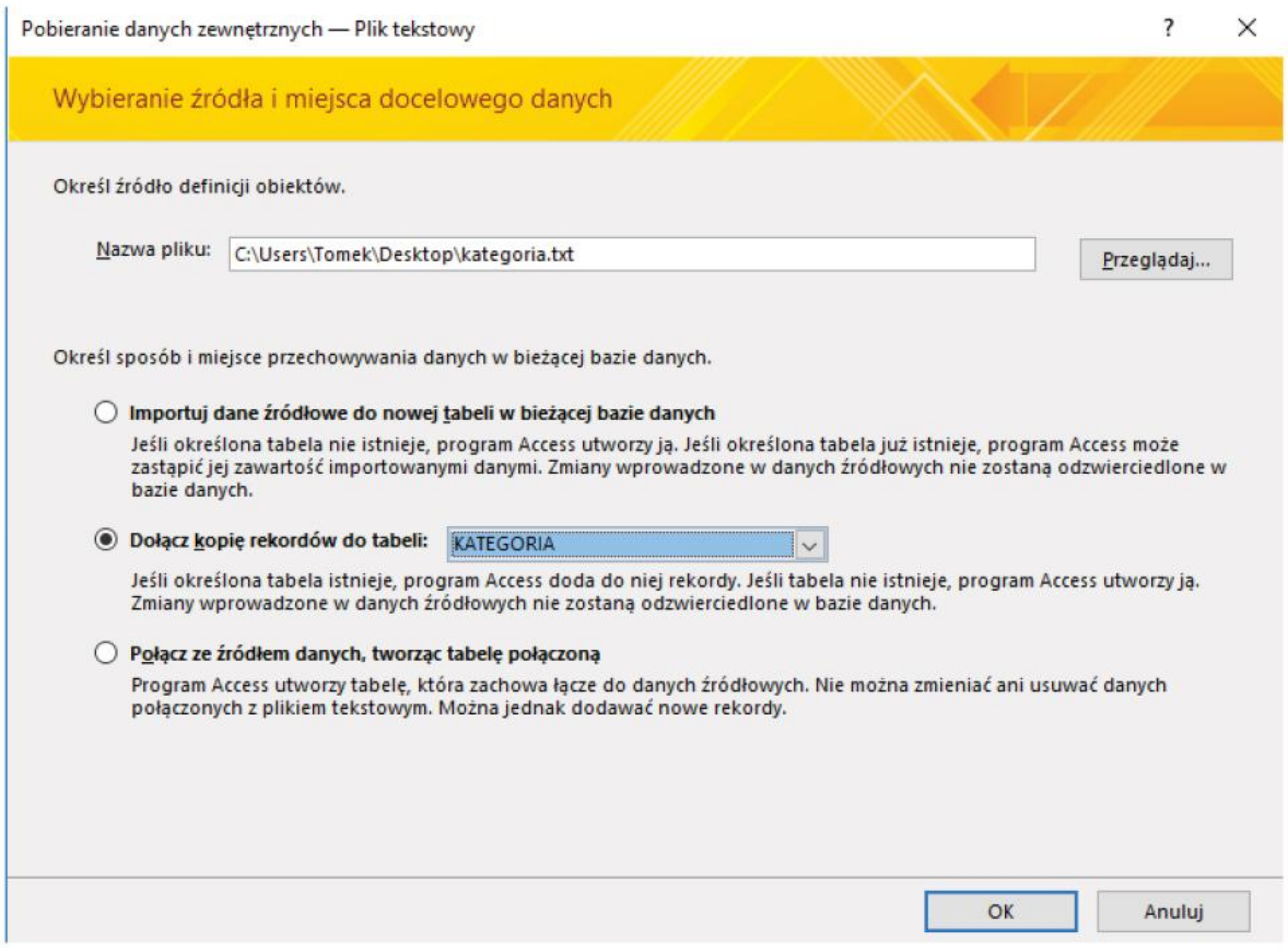
Podczas tworzenia plików źródłowych do importowania danych w programie MS Access należy pamiętać o następujących zasadach:

- pliki rozdzielane mogą zawierać wartości pól ujęte w pojedyncze lub podwójne cudzysłowy, np. ”Tomasz Klekot”,2019-03-05,”Warszawa”. Cudzysłów jest nazywany kwalifikatorem tekstu. Kwalifikatory tekstu nie są ogólnie wymagane, ale niezbędne w następujących przypadkach:

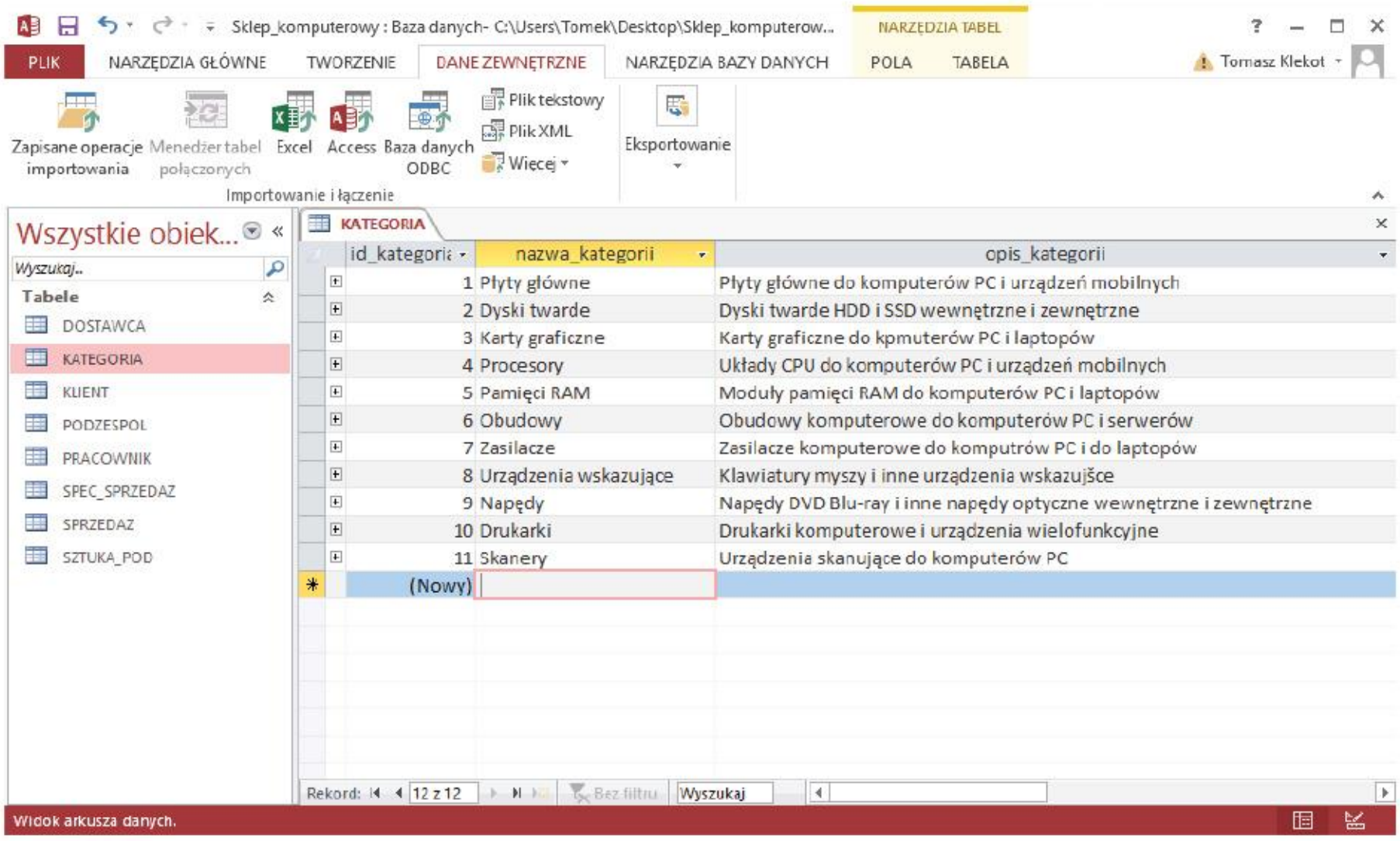
  - ogranicznik wartości pola występuje jako jego część, np. "Warszawa, Łódź";
  - MS Access ma traktować wartości nietekstowe (np. liczby) jak wartości tekstowe i przechowywać je w polu tekstowym, np. "00323000445".

  Podczas operacji importowania należy określić, czy w pliku są używane kwalifikatory, a jeśli tak, to jaki znak pełni funkcję kwalifikatora;
- liczba pól źródłowych nie może przekroczyć 255 sztuk;
- z pliku należy usunąć wszystkie zbędne puste wiersze oraz pola; puste pola będą przez MS Access akceptowane tylko w razie zdefiniowania właściwości typu NULL (niewymagane).

Po przygotowaniu pliku źródłowego przystępuje się do importowania danych. W tym celu wybiera się zakładkę **Dane zewnętrzne** w grupie **Importowanie i łączenie** oraz naciska lewym klawiszem myszy pozycję **Plik tekstowy**. Otworzy się okno dialogowe przedstawione na rys. 11.3. W polu **Nazwa pliku** podaje się ścieżkę do pliku źródłowego oraz określa sposób i miejsce przechowywania danych w bazie danych. W naszym przykładzie zaznaczamy opcję **Dołącz kopię rekordów do tabeli** i wybieramy tabelę **Kategoria**. Po zatwierdzeniu tego kroku klawiszem OK przechodzimy do dalszej części kreatora. W kolejnych etapach sprawdzamy poprawność wyboru ogranicznika oraz zatwierdzamy rozpoczęcie importu. Jeżeli kreator nie napotka błędów, poszczególne wiersze z pliku źródłowego zostaną dołączone do tabeli bazy danych. Efekt importu przedstawiono na rys. 11.4.



Rys. 11.3. Okno pobierania danych zewnętrznych



Rys. 11.4. Okno tabeli Kategoria po zaimportowaniu danych

Łączenie bazy danych z zewnętrznym plikiem

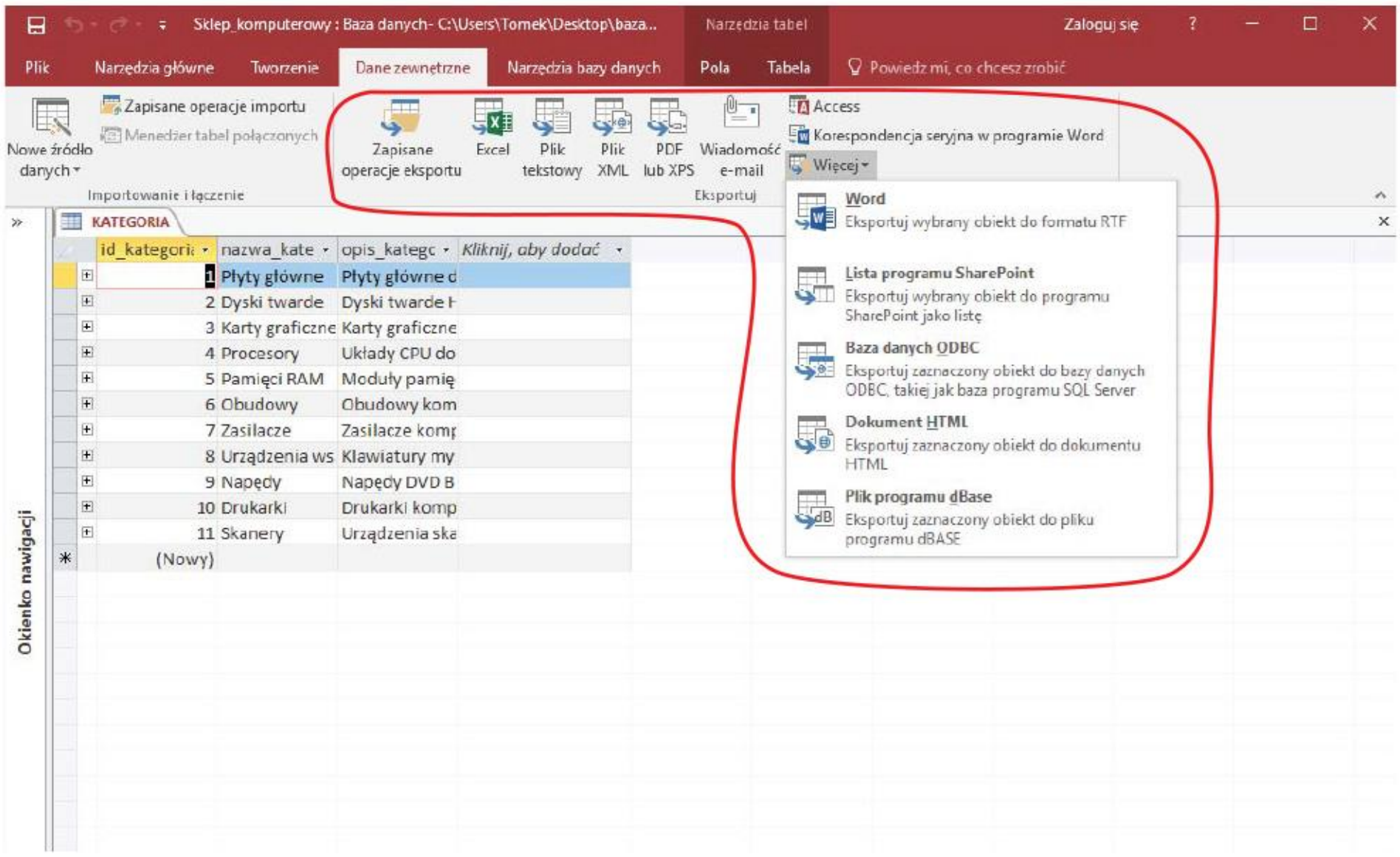
Łączenie pozwala połączyć się z danymi w innej bazie danych bez ich importowania. Umożliwia to przeglądanie i modyfikowanie danych zarówno w źródłowej, jak i w docelowej bazie danych bez tworzenia i utrzymywania dwóch kopii tych samych danych. Można również wykonać połączenie w postaci linku z zewnętrznym plikiem tekstowym, który ma być naszym źródłem danych. Ta operacja powoduje utworzenie specjalnej tabeli, zwanej tabelą połączoną. Wtedy każda zmiana przeprowadzona w pliku źródłowym jest odzwierciedlana w tabeli połączonej, ale nie można edytować zawartości tej tabeli w programie MS Access. Aby zmienić zawartość lub strukturę danych, trzeba otworzyć plik źródłowy i w nim wprowadzić odpowiednie zmiany.

W tym celu korzysta się z tej samej procedury, którą opisano w rozdziale poświęconym importowaniu danych z pliku tekstowego. W oknie przedstawionym na rys. 11.3 należy jednak wybrać opcję **`Połącz ze źródłem danych, tworząc tabelę połączoną`**. Podczas łączenia program MS Access skanuje zawartość pliku i sugeruje wybór sposobu zorganizowania pliku. Należy wówczas określić, czy plik źródłowy jest rozdzielany ogranicznikiem, czy też ma stałą szerokość.

Eksport danych

MS Access umożliwia również eksportowanie danych do innych baz danych, aplikacji, strony internetowej oraz pliku.

Operacja eksportowania danych jest podobna do operacji importu. Kluczowy element to określenie rodzaju dokumentu wynikowego oraz miejsca docelowego eksportu danych. Na rys. 11.5 przedstawiono dostępne opcje eksportu danych w programie MS Access.



Rys. 11.5. Narzędzia eksportu danych

SPRAWDŹ SWOJĄ WIEDZĘ

1. Czym różni się importowanie danych od łączenia?
2. Jakie są rodzaje struktury plików źródłowych wykorzystywanych do importu danych do MS Access z plików tekstowych?
3. Wymień możliwości eksportowania danych w MS Access.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz oddzielne pliki tekstowe ze źródłem danych do trzech tabel bazy Serwis, wykonanych w poprzednim rozdziale (zadanie 2.). Zastosuj metodę plików rozdzielanych. Zaimportuj utworzone pliki do bazy i sprawdź poprawność wykonania operacji.
2. Utwórz pliki tekstowe według specyfikacji do zaimportowania danych do bazy danych Sklep_komputerowy. Zastosuj metodę plików ze stałą szerokością. Zaimportuj utworzone pliki do bazy i sprawdź poprawność wykonania operacji.

   Dane do tabeli DOSTAWCA:

   1,ABCData,00-234,Warszawa,Zgierska,23,Jan Kiel,601234543

   2,Komputerlander,34-234,Łódź,Strykowska,12/23,Tomasz Celeban,602335456

   3,Techmagic,32-098,Wrocław,Psie pole,10,Janusz Nakieł,505234543

   Dane do tabeli KATEGORIA:

   1,Płyty główne,Płyty główne do komputerów PC i urządzeń mobilnych

   2,Dyski twarde,Dyski twarde HDD i SSD wewnętrzne i zewnętrzne

3,Karty graficzne,Karty graficzne do komputerów PC i laptopów
4,Procesory,Układy CPU do komputerów PC i urządzeń mobilnych
5,Pamięci RAM,Moduły pamięci RAM do komputerów PC i laptopów
6,Obudowy,Obudowy komputerowe do komputerów PC i serwerów
7,Zasilacze,Zasilacze komputerowe do komputerów PC i do laptopów
8,Urządzenia wskazujące,Klawiatury myszy i inne urządzenia wskazujące
9,Napędy,Napędy DVD Blu-ray i inne napędy optyczne wewnętrzne i zewnętrzne
10,Drukarki,Drukarki komputerowe i urządzenia wielofunkcyjne
11,Skanery,Urządzenia skanujące do komputerów PC

Dane do tabeli KLIENT:

3,Janusz,Mateja,90-234,Wrocław,Senatorska,12,3,1253340908,78102301098
4,Tomasz,Nakiel,89-098,Łódź,Sagały,3,1,2987450912,70120301023
5,Andrzej,Mazur,23-098,Olsztyn,Masaka,123,23,4092348567,89010292034

Dane do tabeli PODZESPOL:

1,Karta graficzna PCI-E,Asus Nvidia Geforce GTX 1060,Nowoczesna karta PCI-E x16 1582MHz 3GB GDDR5,3
2,Karta graficzna PCI-E,MSI Radeon RX 580,GAMING X 8GB GDDR5,3
3,Dysk twardy HDD,WD 1TB WD10EZEX,Dysk HDD wewnętrzny 3,5" 1000 GB SATA III,2
4,Dysk twardy SSD,Kingston 240GB SA400S37/240G,Dysk SSD wewnętrzny 2,5" SATA SSD A400 240 GB,2
5,Dysk twardy USB,Seagate 2TB Expansion Portable STEA2000400 Dysk zewnętrzny 2,5"USB 3.0,2
6,Drukarka laserowa HP HP LaserJet Pro M12a T0L45A,Laserowa monochromatyczna A4 USB 18 str./min,10

3. Wprowadź ręcznie do tabeli SZTUKA_PODZESPOLU bazy danych Sklep_komputerowy 30 rekordów z przykładowymi danymi. Skorzystaj z widoku arkusza danych tabeli.

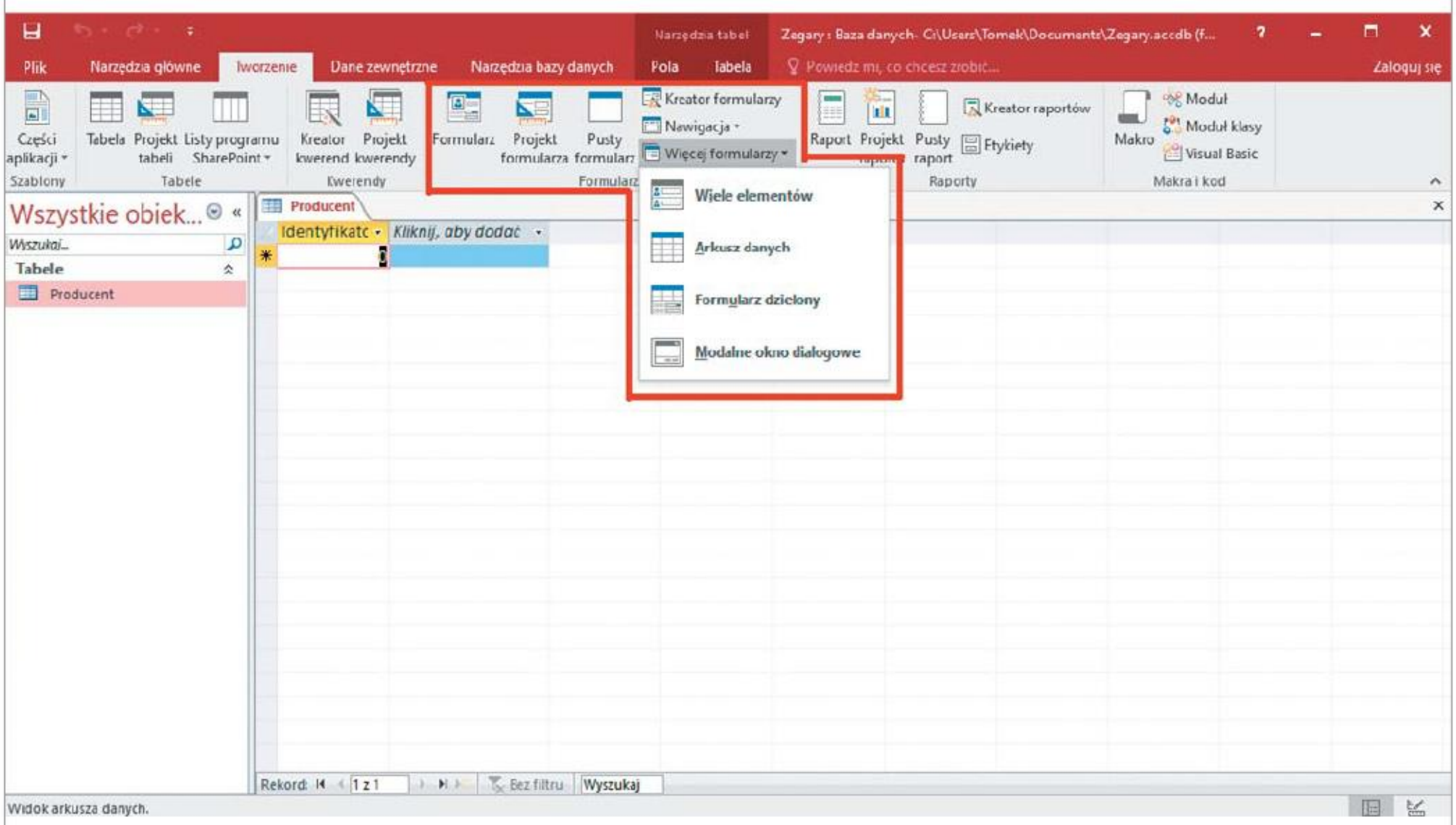
12 Formularze

ZAGADNIENIA

- Funkcje formularzy w MS Access
- Zasady projektowania i tworzenia formularzy

Formularze są obiektami bazy i odgrywają rolę interfejsu użytkownika bazy danych MS Access. Rozróżnia się formularze powiązane i niepowiązane. Źródłem danych dla formularzy powiązanych są inne obiekty bazy, takie jak tabele i kwerendy. Formularz umożliwia wprowadzanie, edytowanie i wyświetlanie danych. Odpowiednio zaprojektowane i wykonane formularze mogą służyć do nawigacji między elementami bazy danych. Tego typu formularze nazywa się formularzami niepowiązanymi.

Formularze można tworzyć za pomocą kreatora, gotowych formularzy oraz dzięki sformatowaniu własnego układu w widoku projektu. Formularze tworzy się za pomocą zakładki **`Tworzenie`** i grupy **`Formularze`** (rys. 12.1).



Rys. 12.1. Opcje tworzenia formularzy w MS Access

W grupie **Formularze** jest do dyspozycji kilka opcji, które pozwalają tworzyć następujące obiekty:

- formularz powiązany, tworzony z istniejącej tabeli lub zapytania;
- formularz pusty, który nie zawiera żadnych kontrolek ani wstępnie zdefiniowanych elementów;
- formularz dzielony, który udostępnia jednocześnie dwa widoki danych – formularza i arkusza danych;
- formularz ciągły o nazwie „Wiele elementów", który stosuje się do wyświetlania wielu rekordów w formularzu o większej możliwości dostosowania niż w arkuszu danych;
- formularz zawierający podformularz, przydatny podczas wyświetlania danych z wielu tabel lub zapytań;
- formularz nawigacyjny, który ma kontrolki nawigacyjne pozwalające uruchamiać inne obiekty bazy danych oraz wykonywać makropolecenia do obsługi całej bazy.

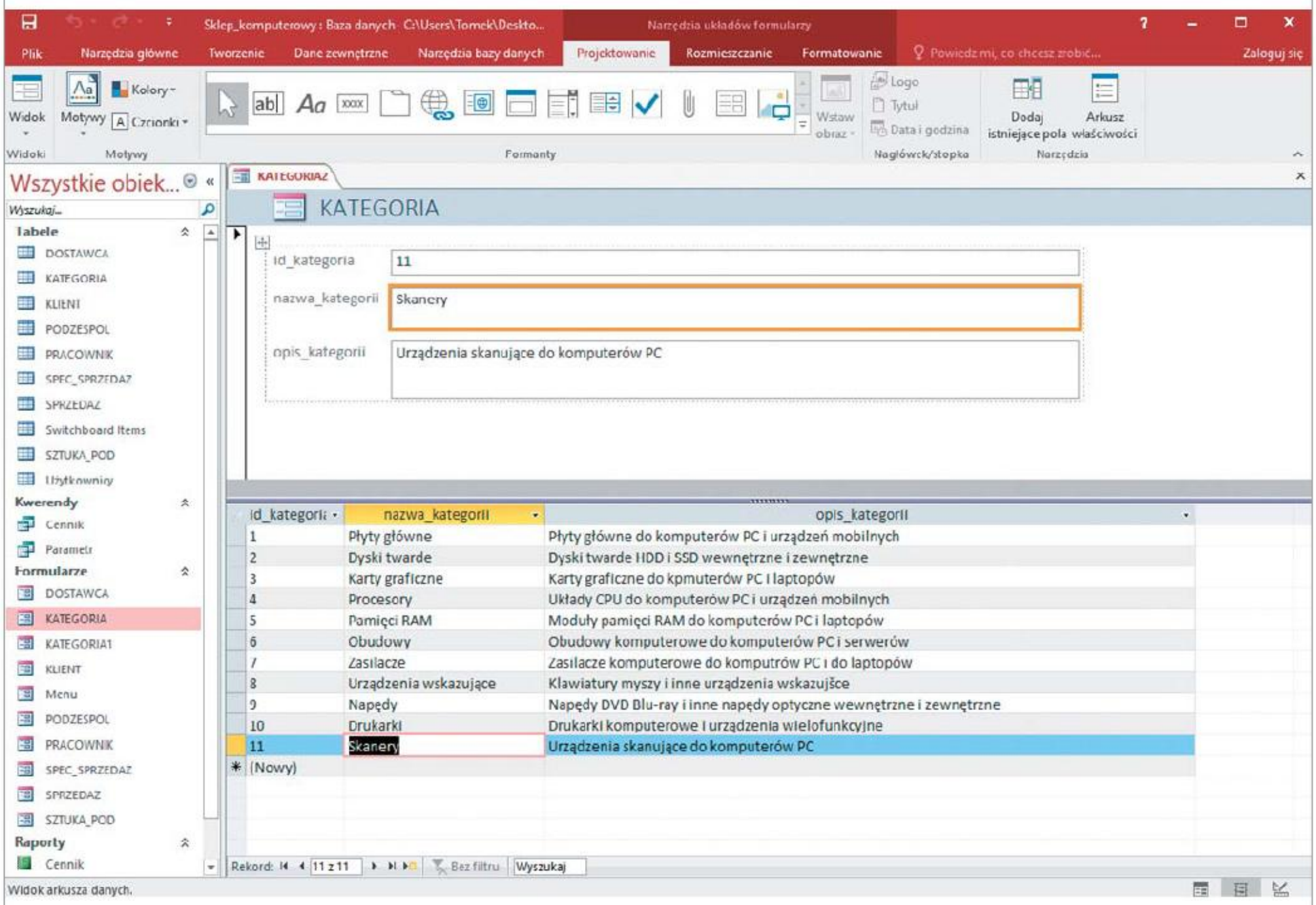
Można również skorzystać z dostępnego kreatora formularzy, który pozwala utworzyć jeden z dostępnych rodzajów formularzy:

- kolumnowy,
- tabelaryczny,
- arkusz danych,
- wyjustowany.

Jeżeli korzysta się z tworzenia formularzy w widoku projektu, można dowolnie pogrupować poszczególne pola tabel i kwerend, opisać poszczególne elementy etykietami bądź wstawić przyciski wykonujące akcje na wszystkich elementach bazy danych. Wszystkie te obiekty zwiększające funkcjonalność formularzy nazywa się **formantami**. Grupa formantów jest widoczna w pasku projektowania formularza. Formanty mogą być **powiązane**, **niepowiązane** i **obliczeniowe**. Każdy formant ma swoją grupę właściwości, dzięki którym można go dokładnie dostosować do wymagań użytkownika. Najlepiej do projektowania nadaje się formularz pusty, ale w każdym tego typu obiekcie można zastosować dodatkowe formanty.

PRZYKŁAD 12.1

Ciekawym przykładem jest **formularz dzielony**. Podział na widok formularza i widok arkusza danych daje wiele możliwości użytecznych w obsłudze bazy danych. Można jednocześnie wprowadzać lub edytować dane do tabeli oraz przeglądać jej zawartość. Poza tym dwa widoki są połączone z tym samym źródłem danych, więc zawsze są ze sobą zsynchronizowane. Tworzenie tego typu formularza, powiązanego z istniejącą tabelą, jest szybkie i bardzo proste. Wystarczy z listy tabel bazy danych zaznaczyć wybraną tabelę oraz wskazać na wstążce **Tworzenie** ikony **Formularz dzielony**. Przykład takiej operacji przedstawiono na rys. 12.2.

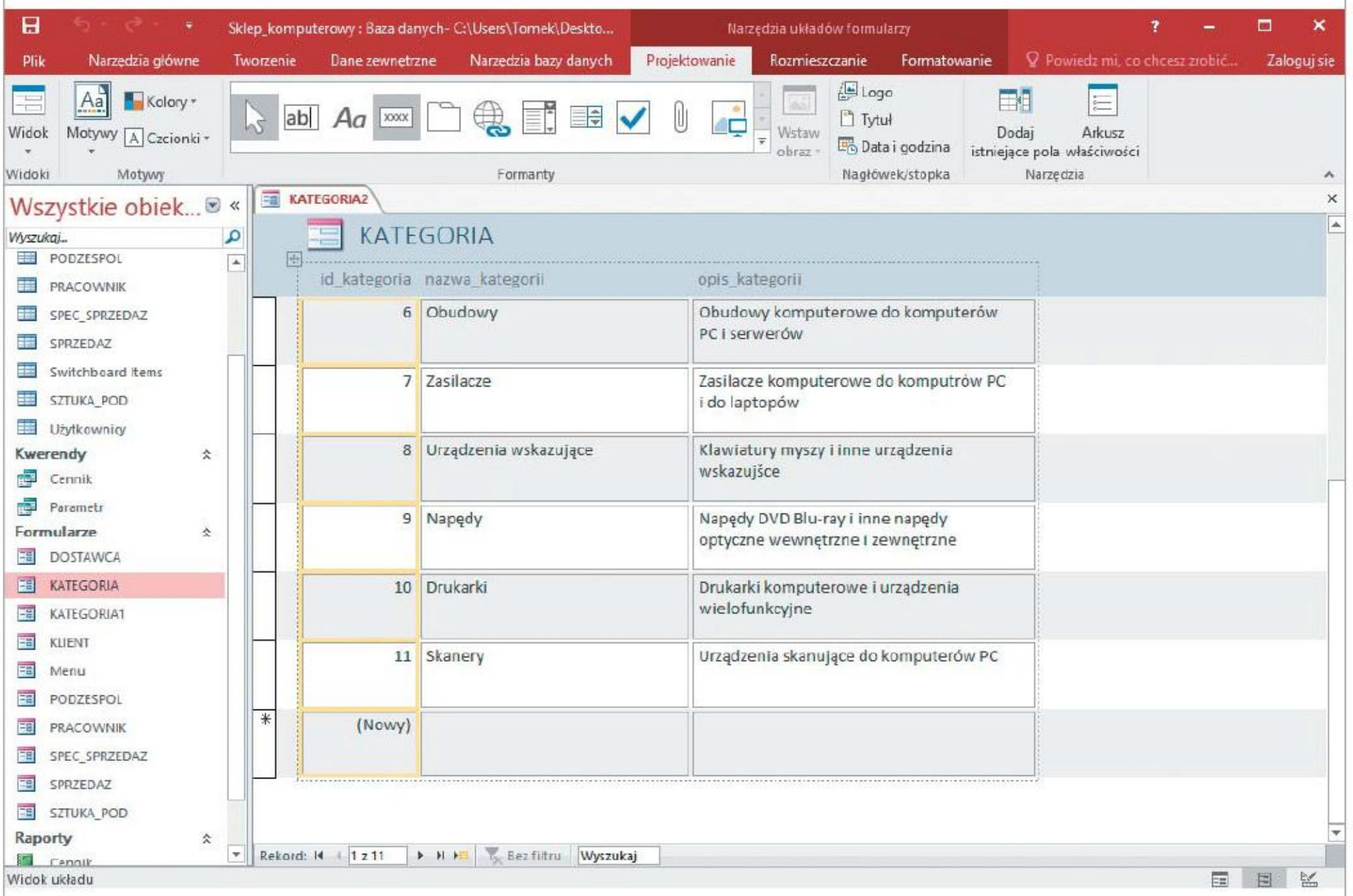


Rys. 12.2. Przykład formularza dzielonego

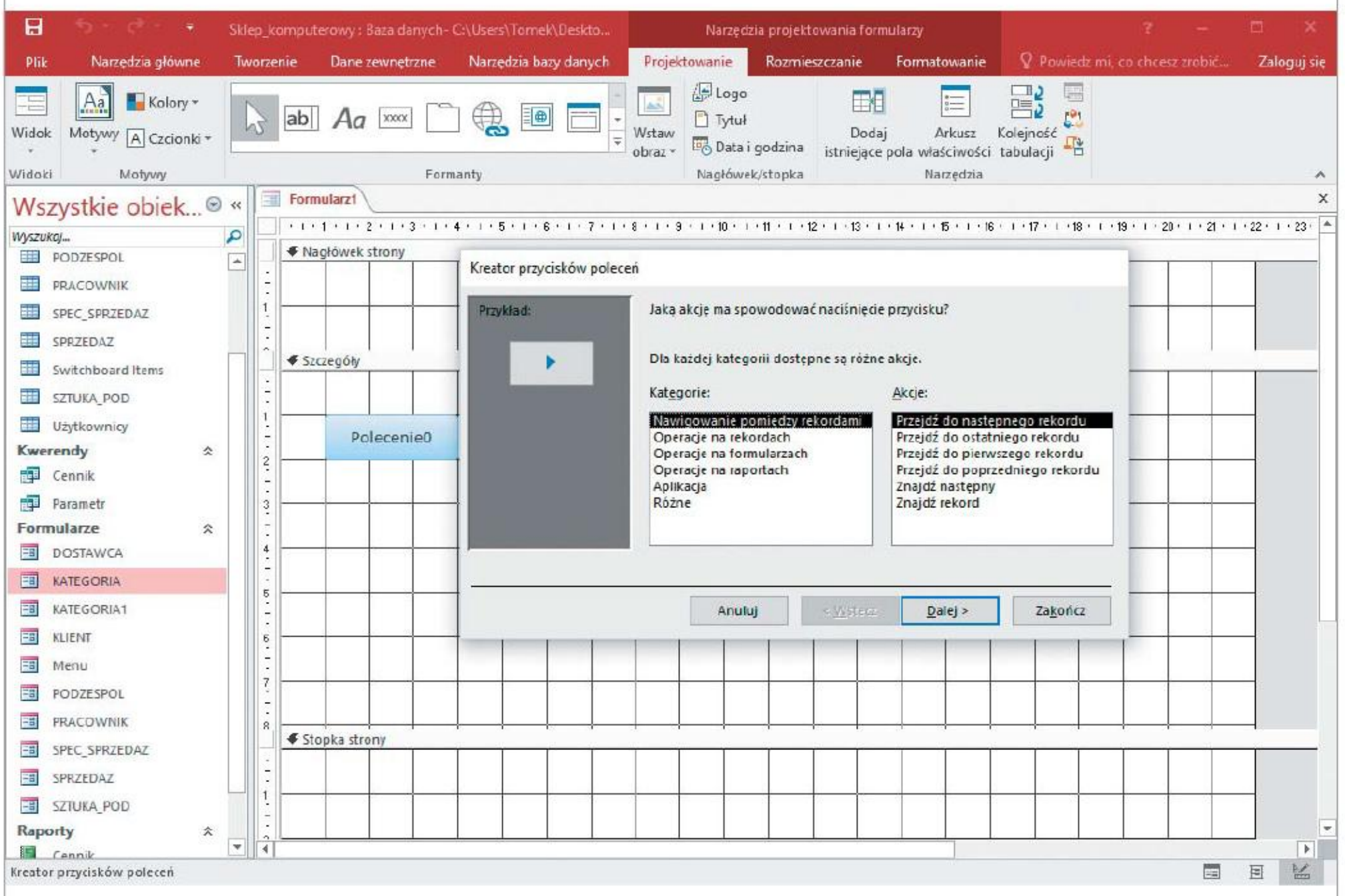
PRZYKŁAD 12.2

Innym przykładem formularza jest **formularz wieloelementowy** (**ciągły**). Jest on idealnym rozwiązaniem dla tabel i kwerend z niewielką liczbą pól, ponieważ oferuje widok arkusza danych w układzie tabelarycznym. Można jednocześnie przeglądać i edytować wszystkie rekordy oraz dodawać nowe. Żeby stworzyć tego typu formularz, należy na liście tabel bazy danych zaznaczyć wybraną tabelę oraz wskazać na wstążce **Tworzenie** ikony formularza **Wiele elementów**. Przykład takiej operacji przedstawiono na rys. 12.3.

Przykładem zastosowania pustego formularza jest menu bazy danych. Jest to obiekt niepowiązany bezpośrednio z jedną konkretną tabelą lub zapytaniem. Tworzy się pusty obszar, który w widoku projektu można dowolnie wypełnić. Do nawigacji wszystkimi elementami tworzącymi bazę danych można użyć np. przycisków, w których zdefiniuje się proste makropolecenia z biblioteki przygotowanej przez MS Access. W tworzeniu formularza pomaga kreator, dzięki któremu sprawnie i szybko modernizuje się jego poszczególne elementy. Na rys. 12.4 przedstawiono przykład pustego formularza z rozpoczętą operacją tworzenia nowego przycisku.

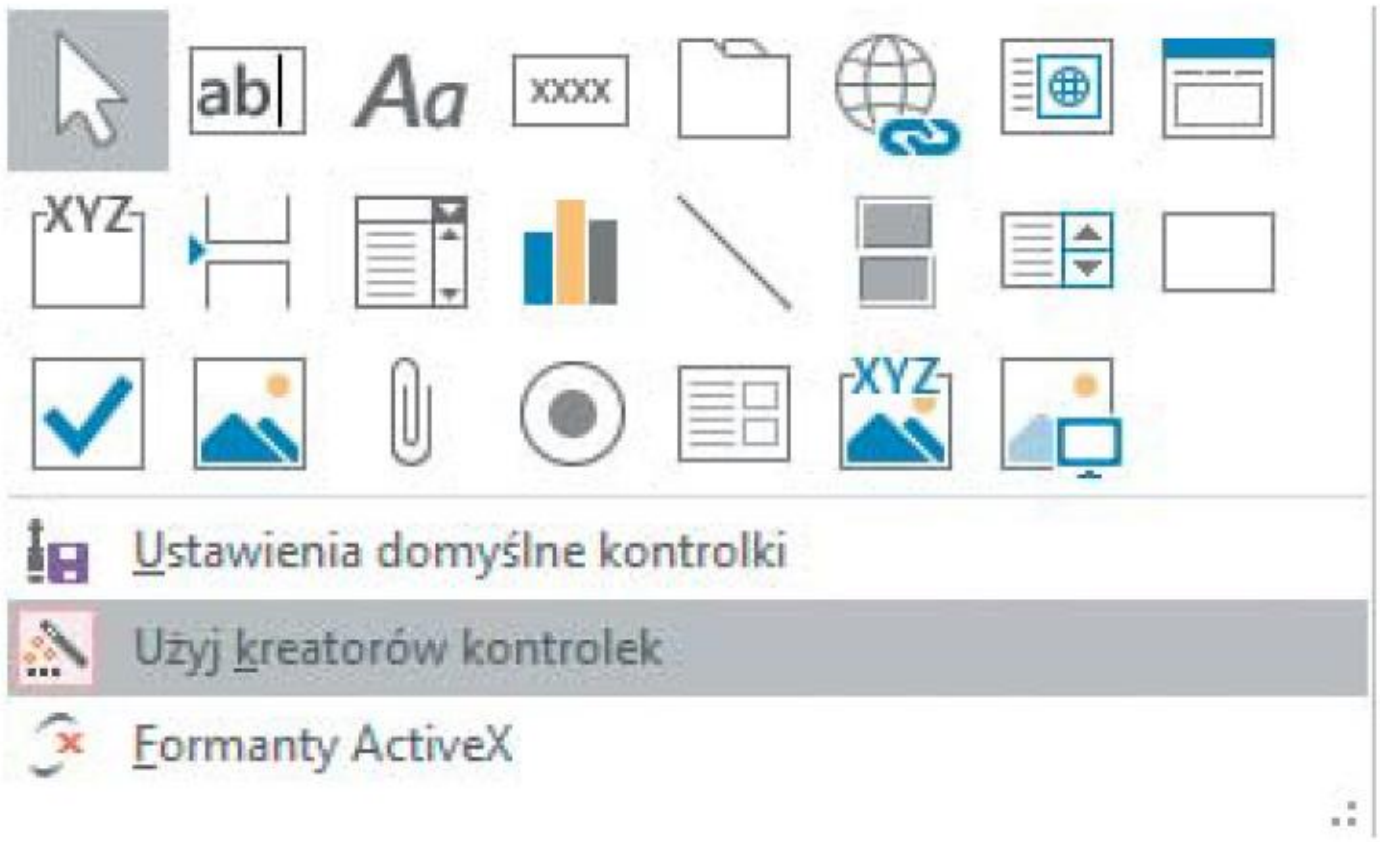


Rys. 12.3. Przykład formularza wieloelementowego



Rys. 12.4. Projektowanie pustego formularza

W widoku projektu pustego formularza uaktywnia się grupa zakładek **Narzędzia projektowania formularzy**. Są w niej wstążki o nazwach **Projektowanie**, **Rozmieszczenie** oraz **Formatowanie**. We wstążce **Projektowanie** jest do dyspozycji grupa formantów nazywanych kontrolkami, które pozwalają realizować najbardziej potrzebne elementy nawigacyjne, edycyjne, graficzne i opisowe. Na rys. 12.5 przedstawiono okno wyboru podstawowych kontrolek. W obszarze roboczym jest siatka, na której można umieszczać poszczególne formanty. Siatka ułatwia ich dokładne rozmieszczenie. Obszar roboczy można dodatkowo podzielić na: nagłówek, szczegóły i stopkę formularza.



Rys. 12.5. Kontrolki projektowania formularzy

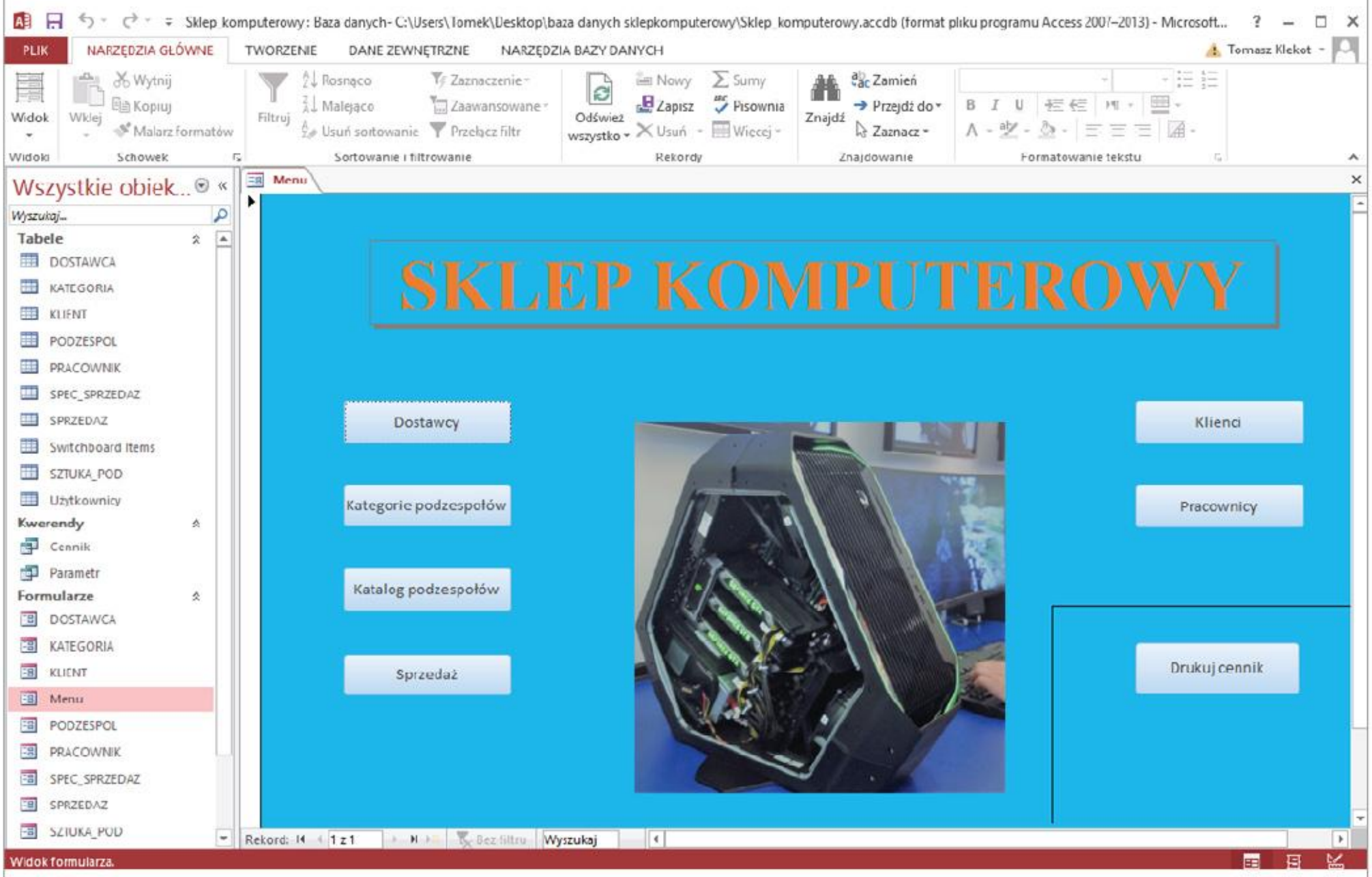
W tabeli 12.1 przedstawiono opis najczęściej wykorzystywanych formantów.

Tabela 12.1. Opis formantów (kontrolek)

Nazwa formantu	Opis zastosowania
pole tekstowe	Wyświetlanie pola z tekstem, liczbą, datą lub godziną. Pole tekstowe można powiązać z jednym z pól w tabeli lub kwerendy źródłowej
etykieta	Nazwy, tytuły i opisy elementów formularza. Tekst stały, bez powiązań
przycisk	Przycisk polecenia, którym można aktywować makra lub procedury języka Visual Basic
hiperlink	Adres URL do zasobu lub strony w sieci Internet lub sieci lokalnej
kontrolka nawigacji	Umożliwia przechodzenie do formularzy i raportów w bazie danych
pole kombi	Zawiera listę potencjalnych wartości dla kontrolki i pola tekstowego

Nazwa formantu	Opis zastosowania
linia, prostokąt	Narzędzia do rysowania elementów graficznych na siatce formularza
przycisk przełącznika, pole wyboru, pole opcji	Umożliwia graficzną edycję pola tak / nie. Można też dołączyć przycisk przełącznika do grupy opcji i przypisać mu unikatowe wartości liczbowe
pole listy	Umożliwia zawieranie listy potencjalnych wartości dla tej kontrolki
obraz, wykres	Pozwala na wstawianie grafiki statycznej oraz wykresów

Na rys. 12.6 przedstawiono przykład projektu formularza pełniącego funkcję menu bazy danych. Są na nim widoczne przyciski poleceń uruchamiające poszczególne formularze, które służą do edycji danych w tabelach, oraz przycisk do drukowania raportu o nazwie „cennik”.



Rys. 12.6. Przykład formularza realizującego funkcję menu

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jaką funkcję w systemach baz danych pełnią formularze?
2. Co to są formanty?
3. Czym charakteryzuje się formularz dzielony?
4. Wymień nazwy kontrolek, które można wykorzystać do wprowadzania wartości logicznych prawda / fałsz w formularzach MS Access.
5. Jaka jest różnica między formatkami: pole tekstowe i etykieta?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz formularz dzielony do wprowadzania i modyfikacji danych dla tabel Klient i Serwisant bazy danych Serwis z poprzedniego rozdziału. Dla tabeli Zlecenie serwisowe utwórz formularz kolumnowy za pomocą kreatora.
2. Zaprojektuj formularz główny dla bazy danych Serwis. Jego zadaniem ma być:
   a) uruchomienie wszystkich tabel z bazy za pomocą przycisków;
   b) zamknięcie bazy danych.
3. Zaprojektuj formularz dla bazy danych Sklep_komputerowy, który pozwoli na przeglądanie danych z tabeli Klient bez pól NIP i PESEL.

13 Kwerendy (zapytania) w MS Access

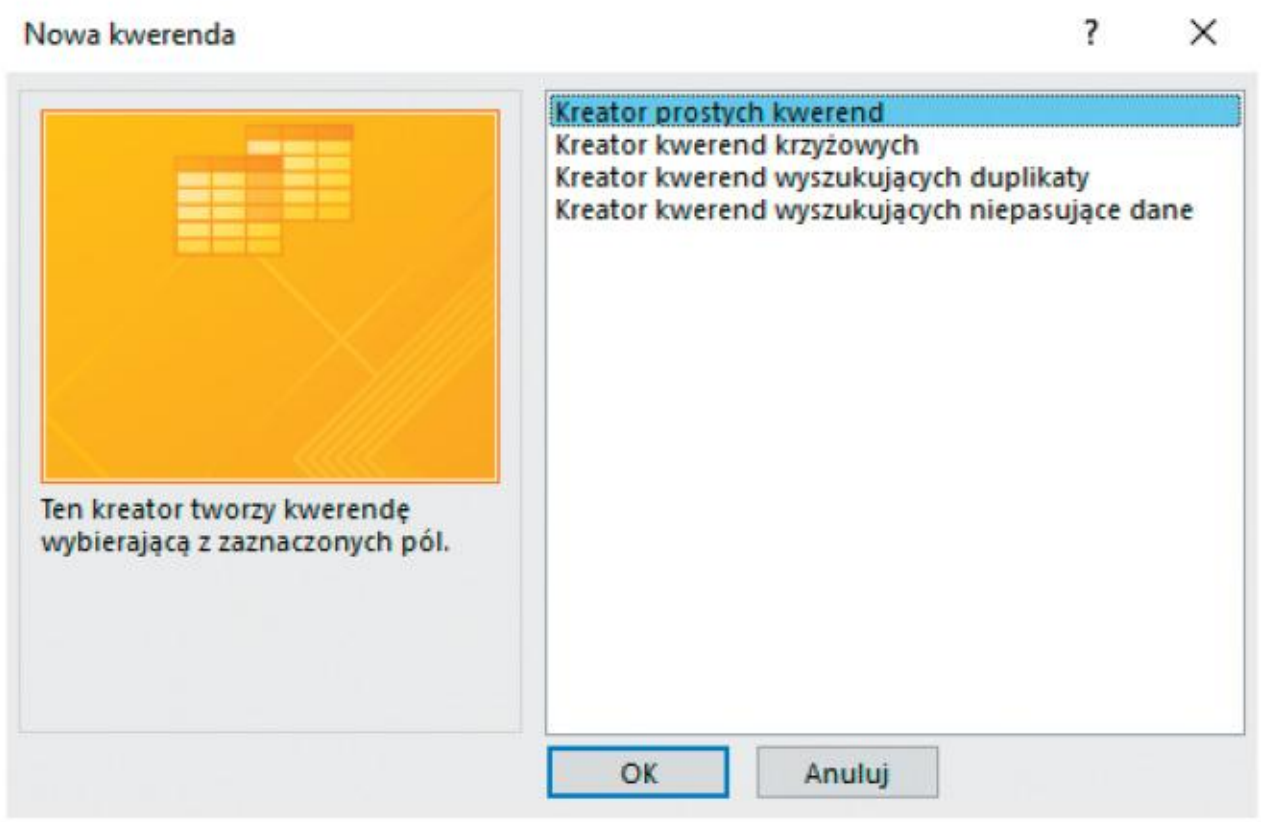
ZAGADNIENIA

- Rodzaje kwerend
- Zasady tworzenia kwerend

Kwerenda jest zapytaniem, które kieruje się do tabel lub innych kwerend. Kwerendy pozwalają na różne sposoby przeglądać, modyfikować i analizować dane. Można ich również używać jako źródeł rekordów dla formularzy, raportów i stron dostępu do danych. W programie MS Access można tworzyć kwerendy za pomocą widoku projektu, kreatora kwerend lub skryptów pisanych w języku **SQL** (ang. Structured Query Language). Kreator kwerend pozwala tworzyć zapytania za pomocą graficznego formularza. W kreatorze można skorzystać z jednej z opcji do tworzenia kwerend:

- prostych,
- krzyżowych,
- wyszukujących duplikaty,
- wyszukujących niepasujące dane.

Na rys. 13.1 przedstawiono okno kreatora kwerend z menu wyboru rodzaju zapytania.



Rys. 13.1. Okno kreatora kwerend

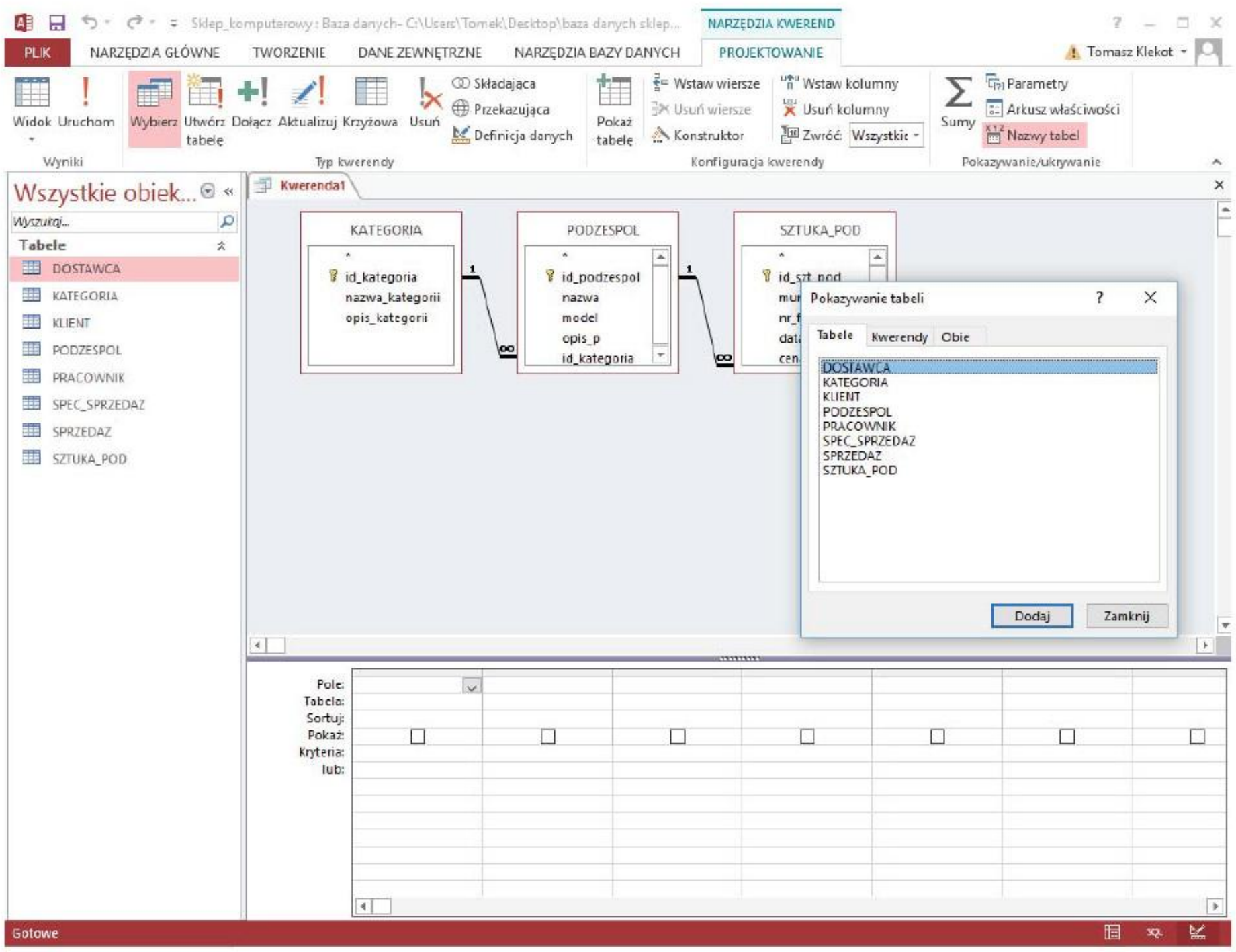
Ze względu na rolę realizowanego zapytania kwerendy dzieli się na dwie grupy:

- **kwerendy wybierające**,
- **kwerendy funkcjonalne**.

Zadaniem kwerendy wybierającej jest wyszukiwanie danych zgodnie z określonymi kryteriami oraz ich prezentacja. Szczególnym rodzajem kwerendy wybierającej jest **kwerenda parametryczna**. Wykonanie kwerendy parametrycznej skutkuje wyświetleniem okna dialogowego, w którym należy podać informacje, takie jak kryteria pobierania rekordów czy wartości, które mają być wstawione w polach.

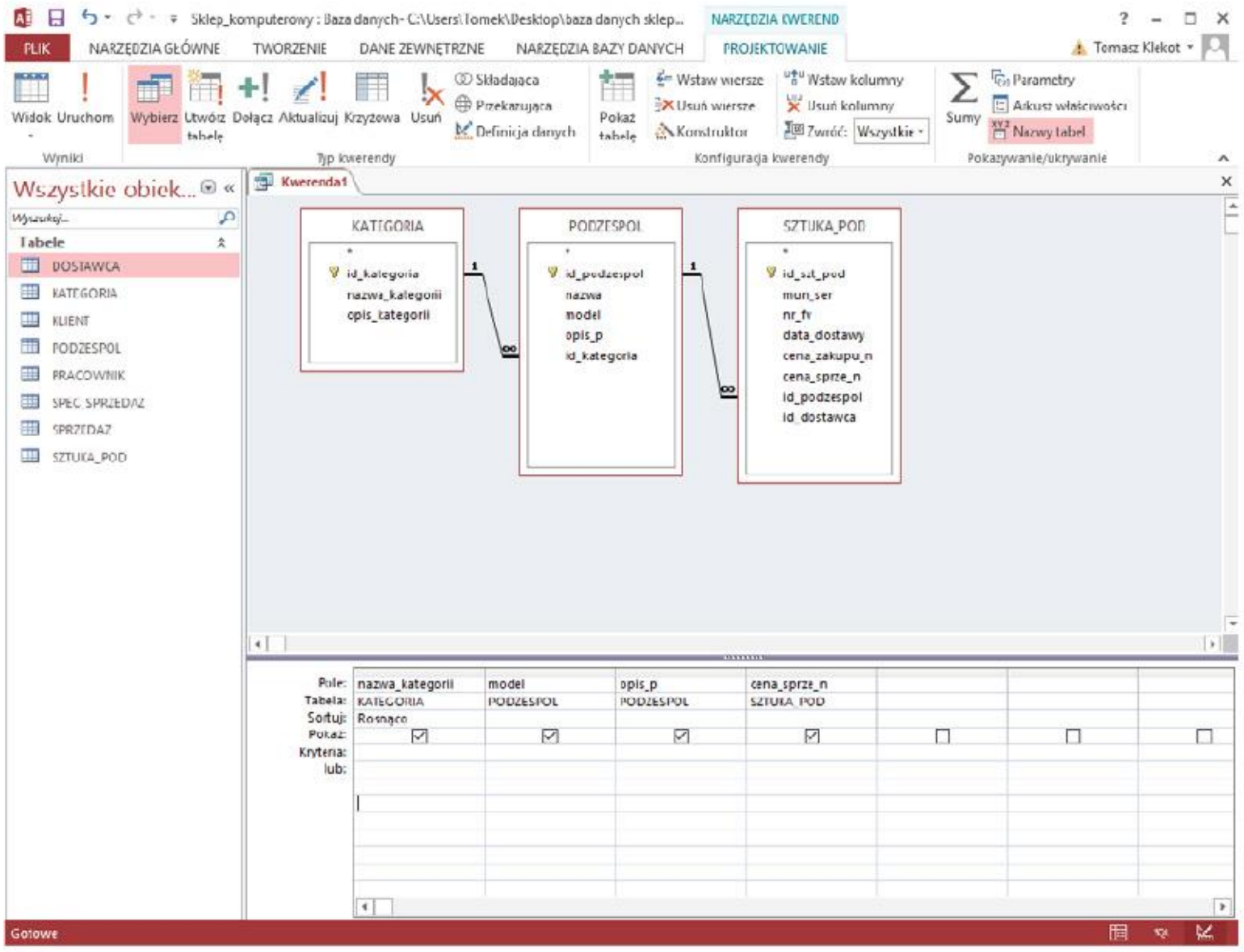
PRZYKŁAD 13.1

Jedną z najczęściej stosowanych kwerend jest **kwerenda wybierająca**. Zwraca ona wyszukane dane zgodnie z określonymi kryteriami. Wygenerujmy kwerendę, która będzie naszą bazą do przygotowania raportu w postaci cennika. Tworzenie kwerendy rozpoczynamy od wyboru z zakładki **`Tworzenie`** opcji **`Projekt kwerendy`**. W górnym oknie tworzenia kwerendy dodajemy tabele, z których będziemy czerpać dane do naszego cennika (rys. 13.2).



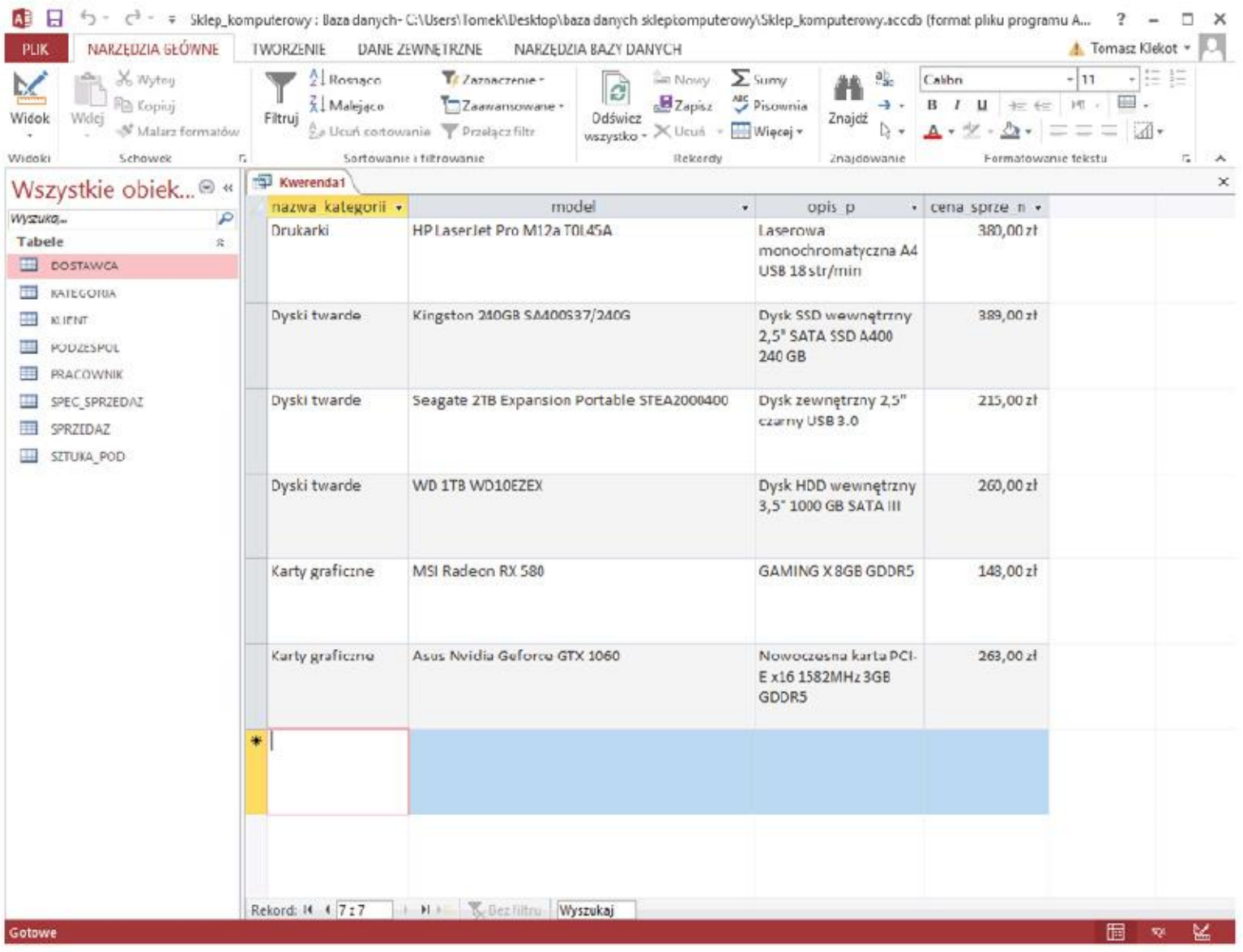
Rys. 13.2. Dodawanie tabel do zapytania

Następnie w dolnym oknie kryteriów wybieramy w pierwszym wierszu nazwy pól, które nas interesują. Dołączamy do kryteriów pola: **`nazwa_kategorii (tabela kategoria)`**, **`model (tabela podzespol)`**, **`opis_p (tabela podzespol)`**, **`cena_sprze_n (tabela sztuka_pod)`**. Aby wynik zapytania został wyświetlony alfabetycznie według kategorii, w wierszu **`Sortuj`** zaznaczamy opcję **`Rosnąco`**. Zakończoną edycję naszej kwerendy przedstawiono na rys. 13.3.



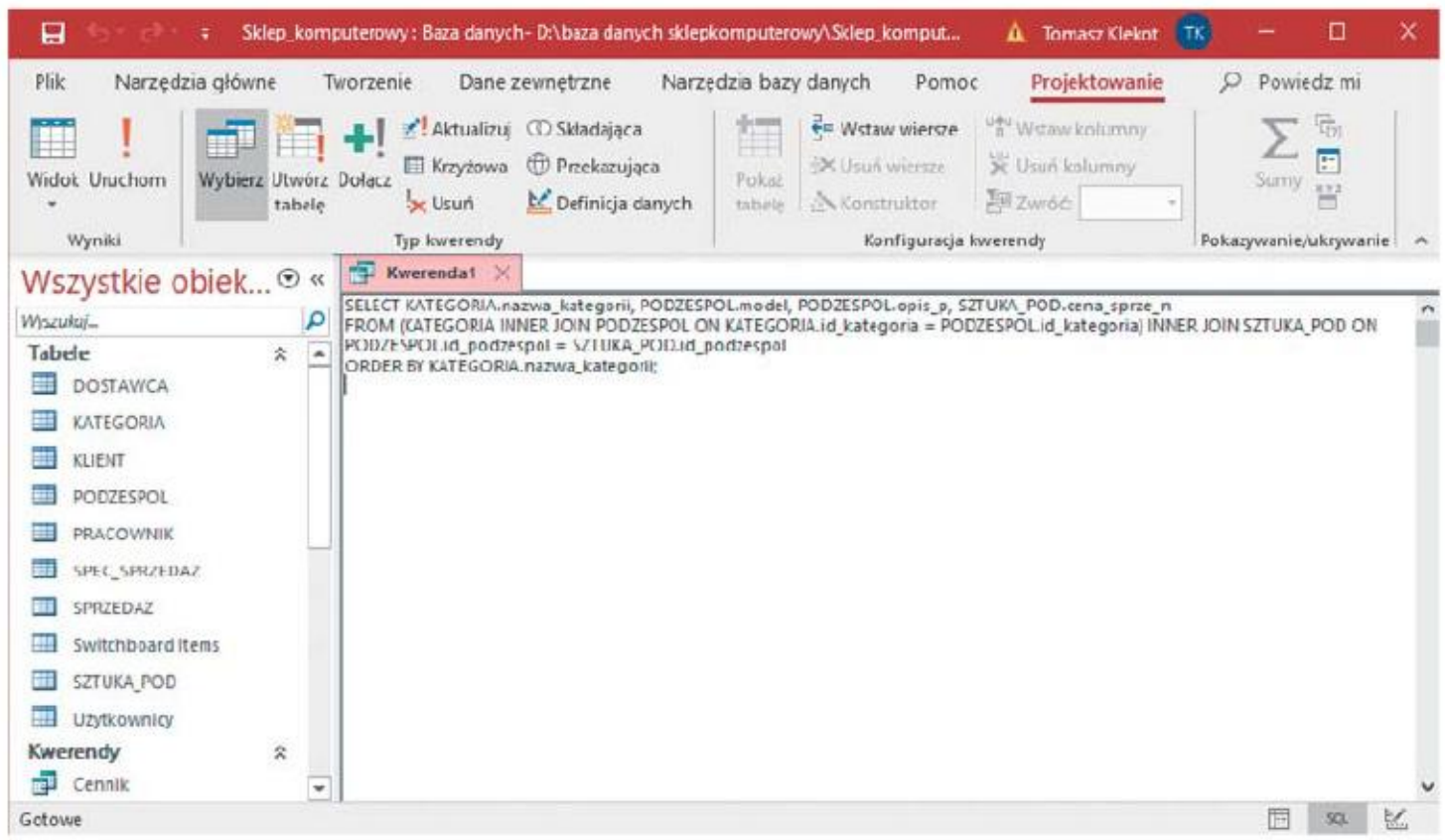
Rys. 13.3. Utworzona kwerenda wybierająca

Po wyborze opcji **`Uruchom`**, widocznej po lewej stronie paska narzędziowego **`Narzędzia kwerend/Projektowanie`**, możemy zobaczyć efekt działania naszego zapytania i ewentualnie wprowadzić zmiany przed zapisaniem (rys. 13.4).



Rys. 13.4. Efekt działania zapytania

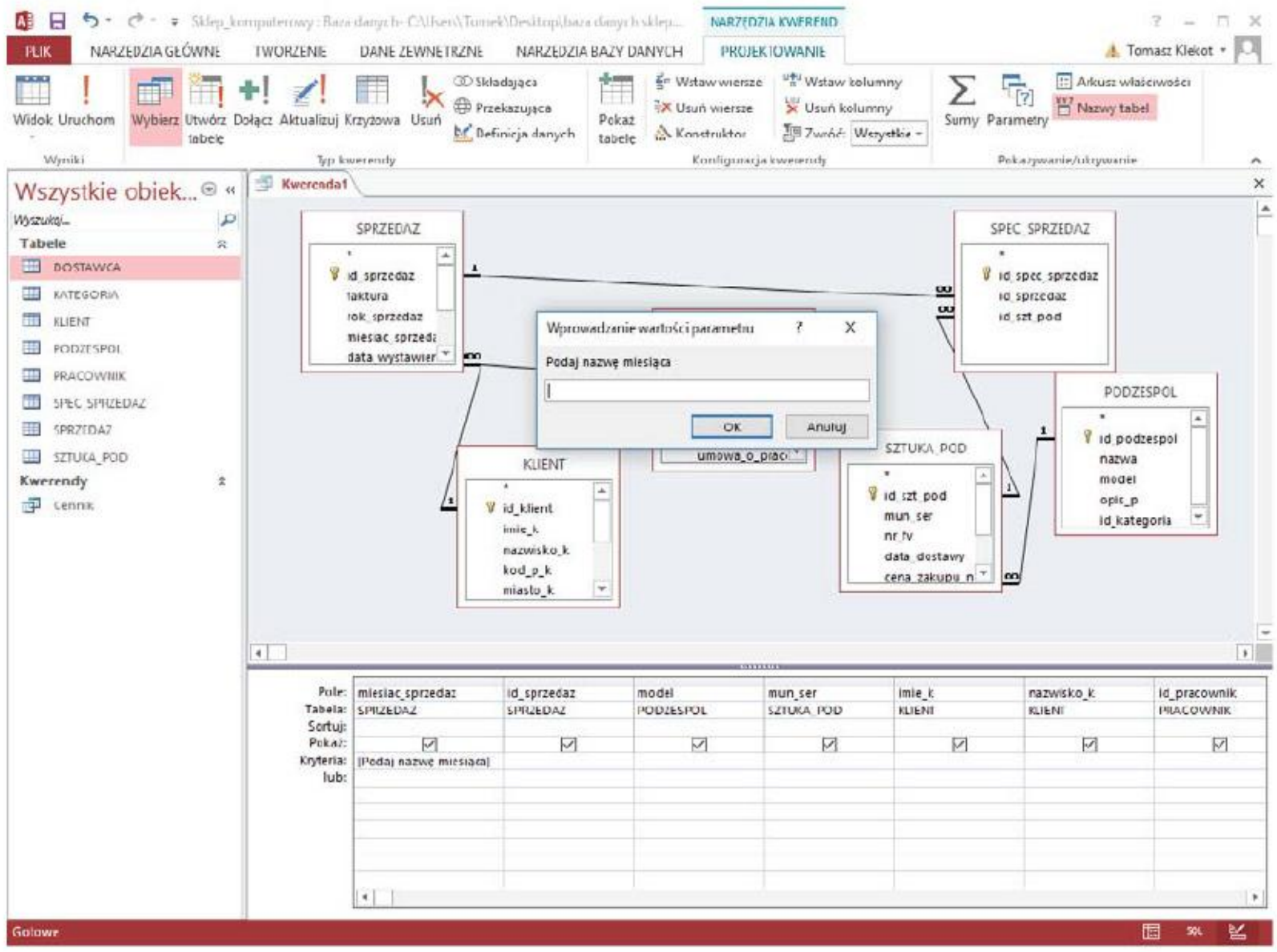
Po zdefiniowaniu zapytania w widoku projektu możemy wygenerować jego postać w języku SQL. Do tego celu służy opcja **SQL** w prawym dolnym rogu okna aplikacji. Na rys. 13.5 przedstawiono składnię zapytania wybierającego z naszego przykładu w postaci kodu SQL.



Rys. 13.5. Okno generatora kodu SQL

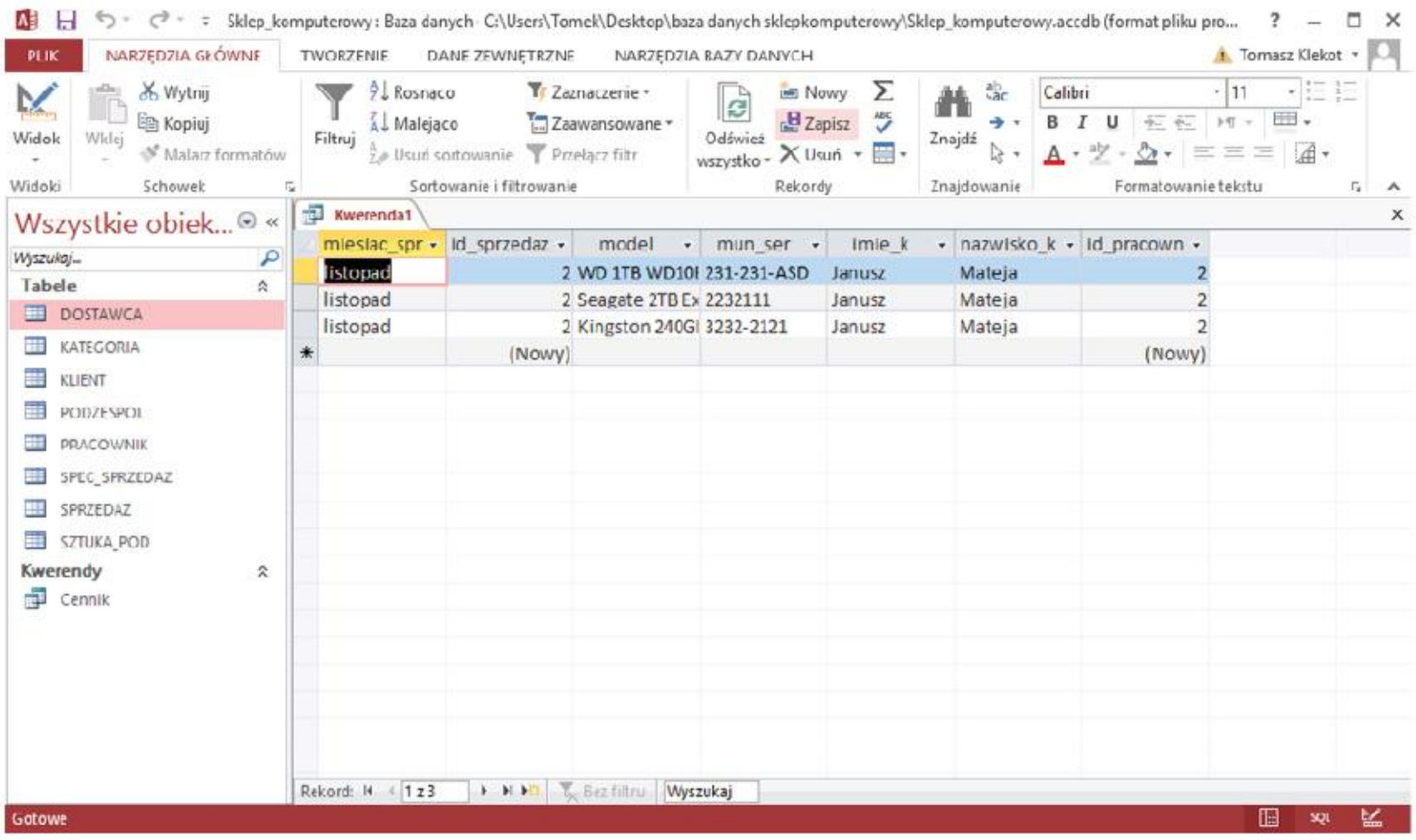
PRZYKŁAD 13.2

Specyficzną odmianą kwerendy wybierającej jest **kwerenda parametryczna**. Jeżeli chcemy – przykładowo – zobaczyć, jakie operacje sprzedaży wystąpiły w miesiącu wybranym przez użytkownika, w wierszu **`Kryteria`** wpisujemy tekst **`[Podaj nazwę miesiąca]`** (rys. 13.6).



Rys. 13.6. Edycja kwerendy parametrycznej

Pozwoli to na wyświetlenie okna dialogowego po uruchomieniu kwerendy z etykietą o treści wpisanej przez nas w kryterium. Po wpisaniu nazwy miesiąca przez użytkownika i zatwierdzeniu wyboru uruchomi się zapytanie – wyświetli określone dane zgodne z wprowadzonym wcześniej parametrem. Efekt działania kwerendy przedstawiono na rys. 13.7.



Rys. 13.7. Efekt działania zapytania parametrycznego

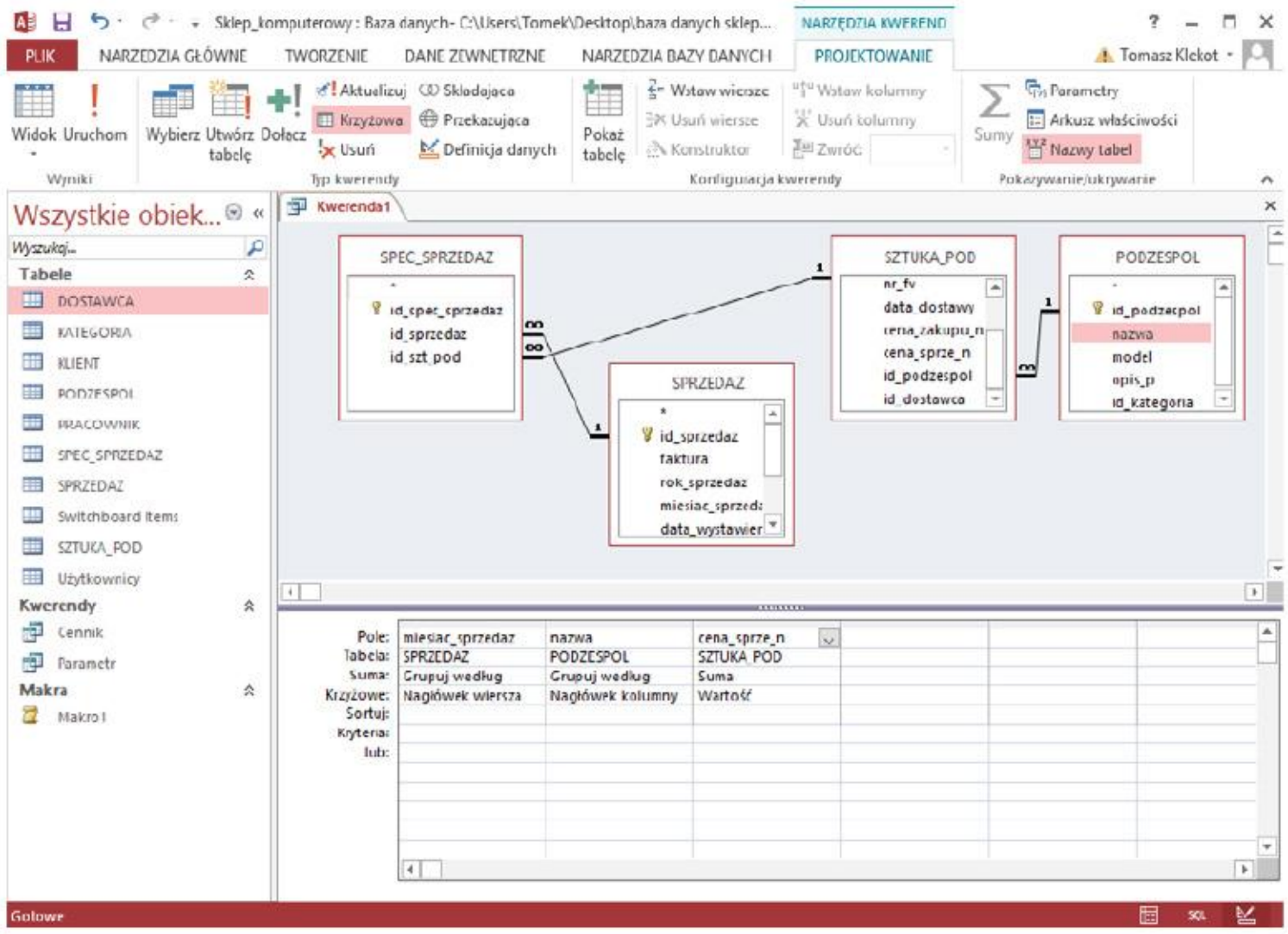
Drugą grupą kwerend są kwerendy funkcjonalne. Do kwerend funkcjonalnych zalicza się:

- **kwerendy modyfikujące** – pozwalają na wprowadzenie modyfikacji w rekordach tabel; wyróżnia się cztery rodzaje kwerend modyfikujących:
- **kwerendy usuwające** – usuwają rekordy z tabel według podanych kryteriów;
- **kwerendy aktualizujące** – pozwalają na modyfikacje istniejących danych w jednej tabeli lub kilku tabelach;
- **kwerendy dołączające** – powodują dołączenie do wybranej tabeli rekordów z innej tabeli;
- **kwerendy tworzące tabelę** – tworzą nową tabelę zawierającą rekordy określone w zapytaniu;
- **kwerendy krzyżowe** – powodują prezentację zliczonych wartości z pola i porządkują je w wiersze i kolumny.

PRZYKŁAD 13.3

Podczas tworzenia kwerendy krzyżowej określamy pola, które zawierają nagłówki wierszy, nagłówki kolumn oraz wartości do podsumowania. W celu zdefiniowania nagłówków kolumn i wartości do podsumowania możemy użyć tylko jednego pola. W celu określenia nagłówków wierszy możemy zastosować maksymalnie trzy pola. W bazie danych Sklep_komputerowy zrealizujemy kwerendę krzyżową,

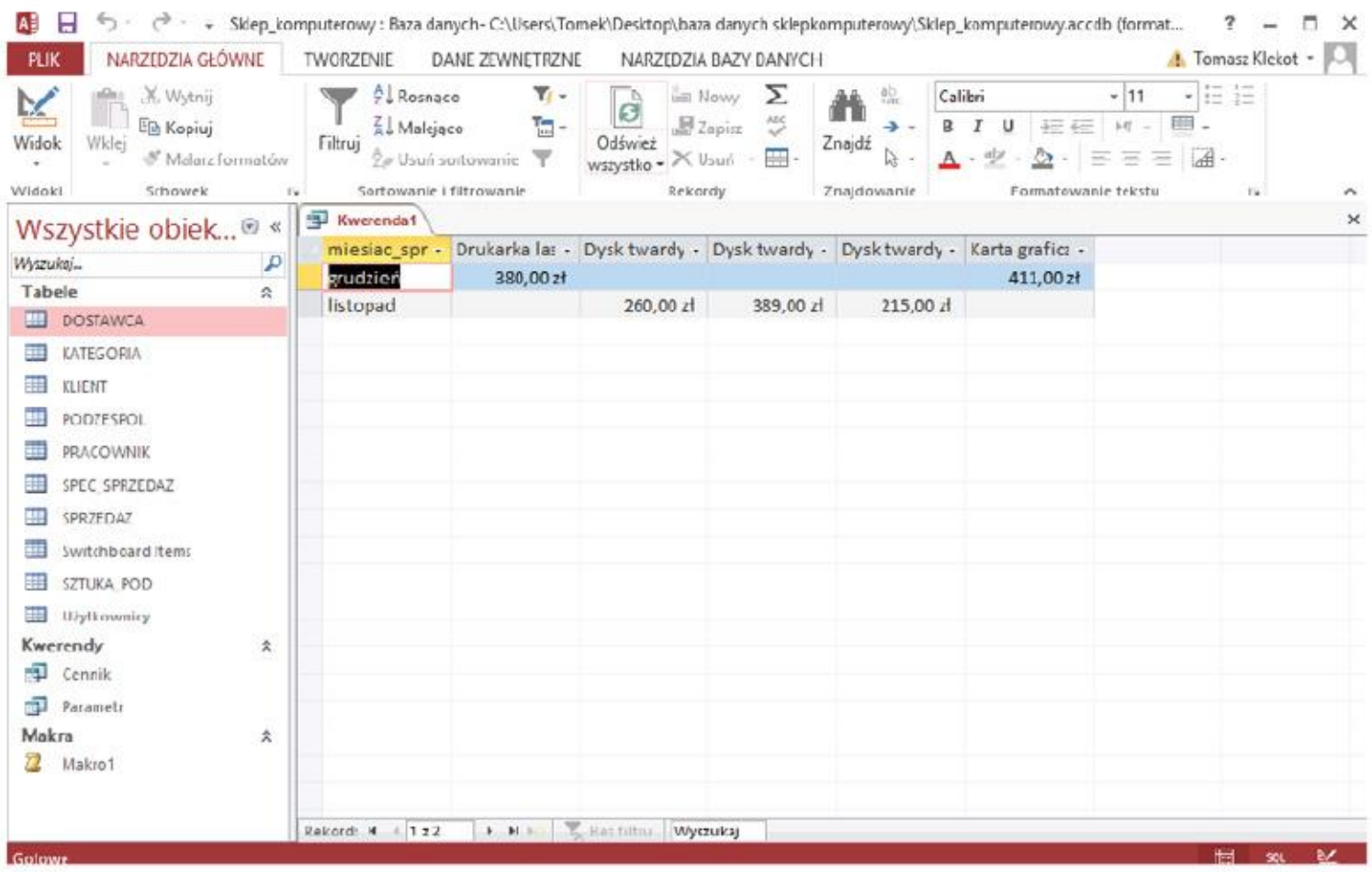
która wyświetli ceny netto sprzedaży podzespołów, które znalazły nabywców w poszczególnych miesiącach. Projekt takiej kwerendy przedstawiono na rys. 13.8.



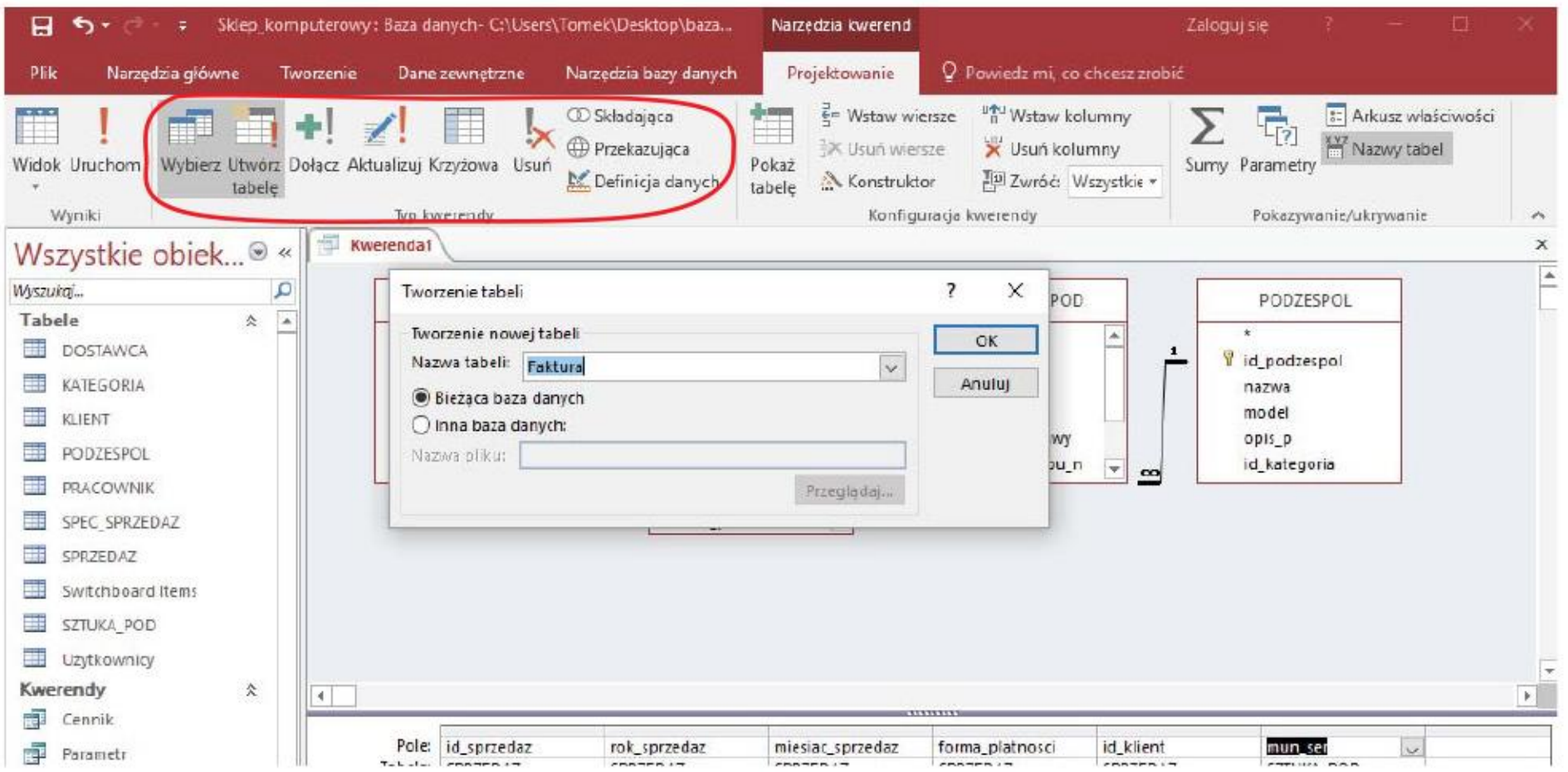
Rys. 13.8. Projekt kwerendy krzyżowej

Po uruchomieniu kwerendy otrzymamy zestawienie miesięczne sprzedaży poszczególnych podzespołów według cen netto (rys. 13.9).

W oknie tworzenia kwerend możemy wybrać określony typ tworzonej kwerendy. Na rys. 13.10 przedstawiono narzędzia pozwalające na utworzenie dowolnej kwerendy funkcjonalnej.



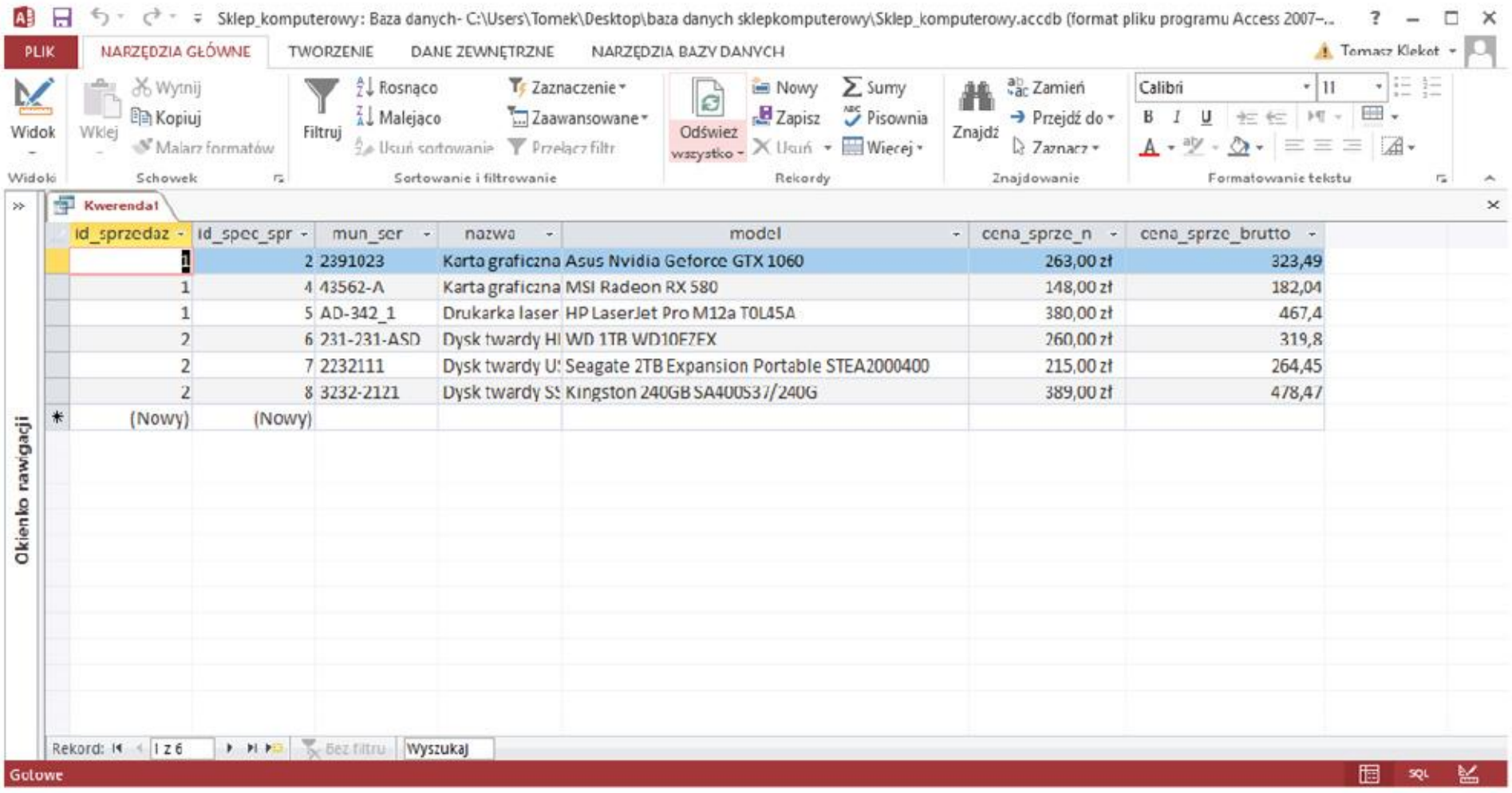
Rys. 13.9. Wynik kwerendy krzyżowej



Rys. 13.10. Okno tworzenia kwerend w MS Access

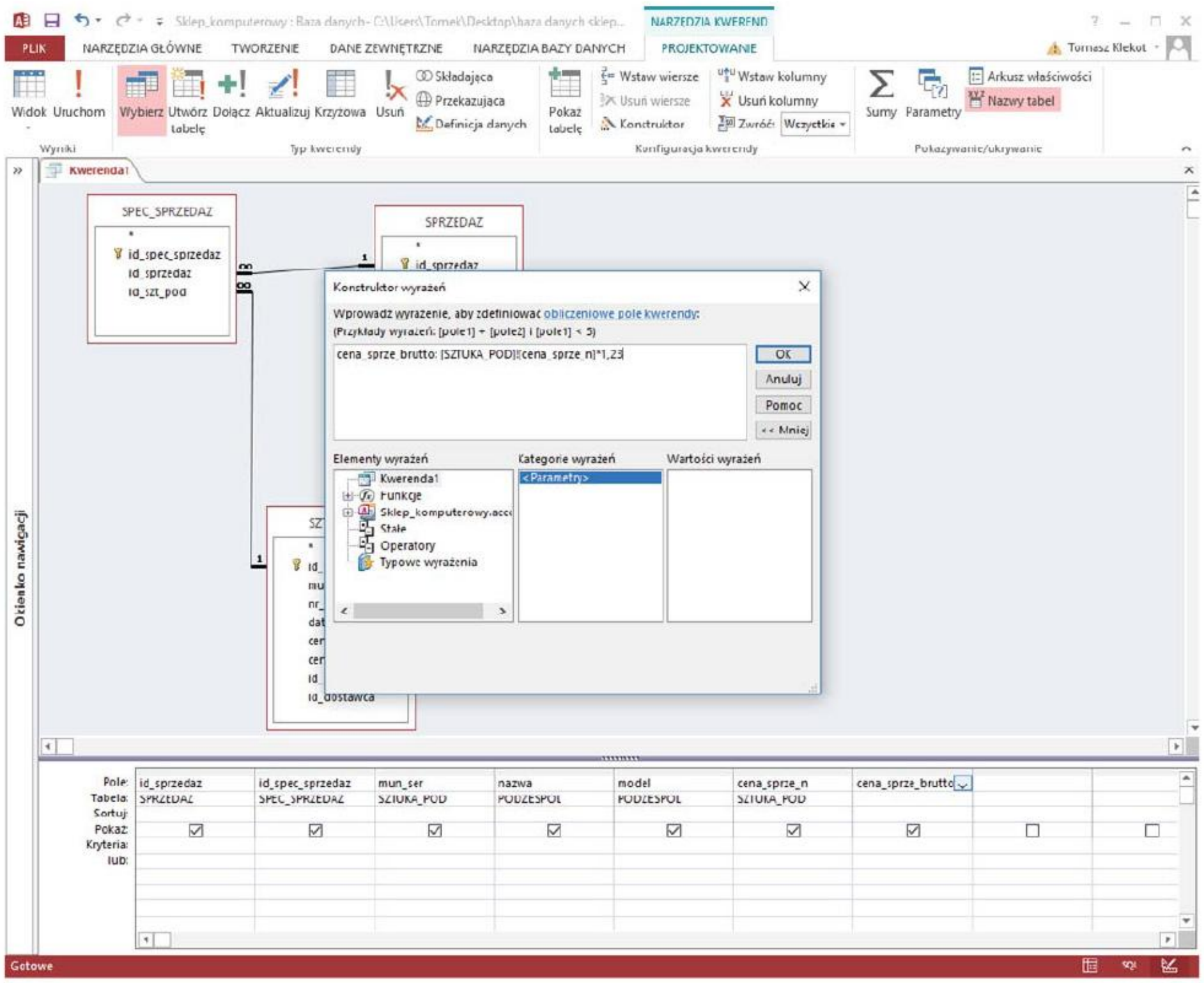
PRZYKŁAD 13.4

W polach kwerendy możemy również korzystać z konstruktora wyrażeń. Za jego pomocą uda nam się stworzyć najróżniejsze zapytania filtrujące najbardziej rozbudowane bazy danych i wyświetlić interesujące nas informacje. Te informacje możemy następnie zwizualizować w postaci raportów lub wyeksportować w celu dalszej analizy do innych programów pakietu Office bądź udostępnić w sieci. Konstruktora wyrażeń możemy również użyć do tworzenia pól obliczeniowych w kwerendach. Na rys. 13.11 przedstawiono kwerendę wybierającą z utworzonym polem, w którym automatycznie jest wyliczana cena sprzedaży podzespołu brutto (wraz z VAT).



Rys. 13.11. Okno kwerendy z polem obliczeniowym

Aby utworzyć taką kwerendę, należy w widoku projektu na wolnym polu uruchomić konstruktora wyrażeń (skorzystać z ikony **Konstruktor**). Następnie tworzymy formułę, która przemnoży wartość z pola cena_sprze_n z tabeli sztuka_pod przez 1,23. Wynik zastosowania formuły utworzonej za pomocą konstruktora wyrażeń przedstawiono na rys. 13.12.



Rys. 13.12. Projekt kwerendy z polem obliczeniowym

Jak już wspomniano, w MS Access można wykorzystać zapytania tworzone w języku SQL. Podstawy tego strukturalnego języka zapytań będą tematem jednego z następnych rozdziałów podręcznika.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Scharakteryzuj rodzaje kwerend funkcjonalnych.
2. Omów na dowolnym przykładzie działanie kwerendy krzyżowej.
3. Jakie narzędzie umożliwia automatyzację tworzenia kwerend?
4. Co oznacza skrótowiec SQL i w jakich sytuacjach się go stosuje?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Wykonaj w bazie danych Sklep_komputerowy następujące zapytania wybierające:
   - wyświetl imiona i nazwiska klientów mieszkających w Łodzi;
   - wyświetl listę podzespołów z tabeli Sztuka_podzespolu z ceną netto mniejszą niż 200 zł;
   - wyświetl z tabeli Dostawca dane osoby do kontaktu z firmy ABCData oraz jego numer telefonu.
2. Wykonaj samodzielnie zapytanie opisane w przykładzie 13.3.
3. Wykonaj samodzielnie zapytanie opisane w przykładzie 13.4.
4. Wykonaj zapytanie parametryczne na podstawie danych Serwis. Zapytanie ma wyświetlić zlecenia serwisowe z datą przyjęcia wpisywaną przez użytkownika w polu o nazwie „Podaj datę zlecenia”.

14 Raporty

ZAGADNIENIA

- Rola raportów w bazach danych
- Zasady tworzenia raportów w MS Access

Raporty służą do tworzenia czytelnych i estetycznych wydruków z przygotowanych danych. Umożliwiają one również przygotowywanie podsumowań, zestawień i etykiet adresowych, które można przekonwertować na różne formaty (m.in. PDF i XPS) w celu publikacji w sieci.

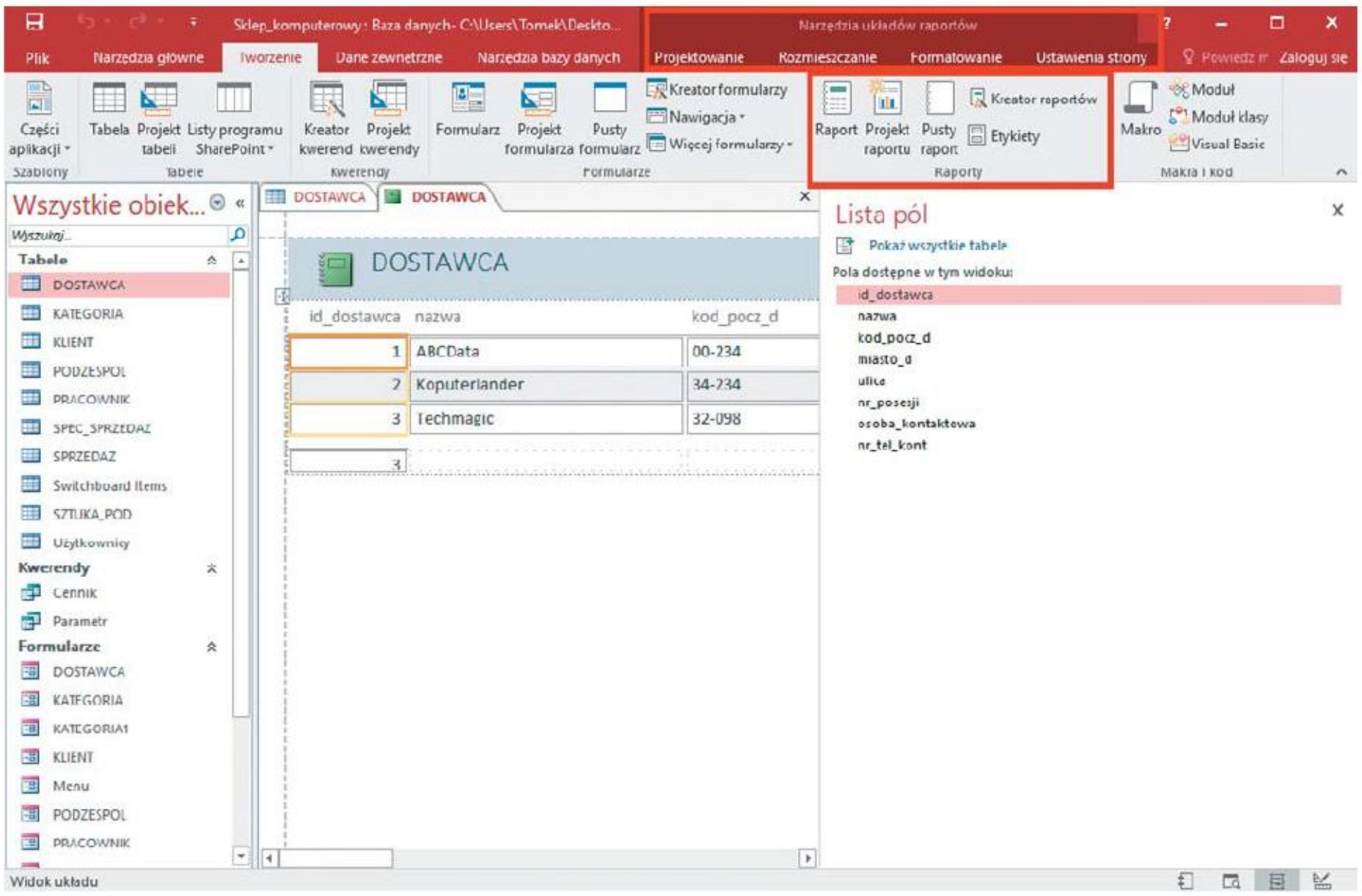
Wyróżnia się następujące rodzaje raportów:

- raport szczegółowy – przedstawia listę danych z tabeli lub kwerendy;
- raport pogrupowany – dane są w nim zorganizowane w grupach, np. według kategorii produktów lub numeru zlecenia serwisowego;
- raport podsumowujący – zawiera sumy całkowite, lecz nie drukuje szczegółów;
- raport z etykietami adresowymi;
- raport wielokolumnowy – wyświetla informacje w kilku kolumnach.

Raporty można stworzyć za pomocą kreatora lub widoku projektu. Projektowanie raportów jest niezbędne, gdy trzeba szczegółowo grupować dane lub wykonywać na nich dodatkowe obliczenia (np. sumowanie). Wszystkie raporty są obiektami związanymi z określonym źródłem danych. Można też tworzyć raporty niezwiązane, lecz ze względu na funkcję tego obiektu nie będziemy wykorzystywać takiej możliwości.

Raporty można drukować z poziomu podglądu wydruku lub widoku projektu. Wykorzystuje się również makropolecenia do stworzenia przycisku automatycznie uruchamiającego wydruk.

Na rys. 14.1 przedstawiono okno tworzenia raportów w MS Access. Po wyborze tworzenia raportu pojawia się nowa wstążka o nazwie **`Narzędzia układów raportów`** zawierająca zakładki: **`Projektowanie`**, **`Rozmieszczanie`**, **`Formatowanie`** i **`Ustawienia strony`** (zaznaczone czerwonym prostokątem).



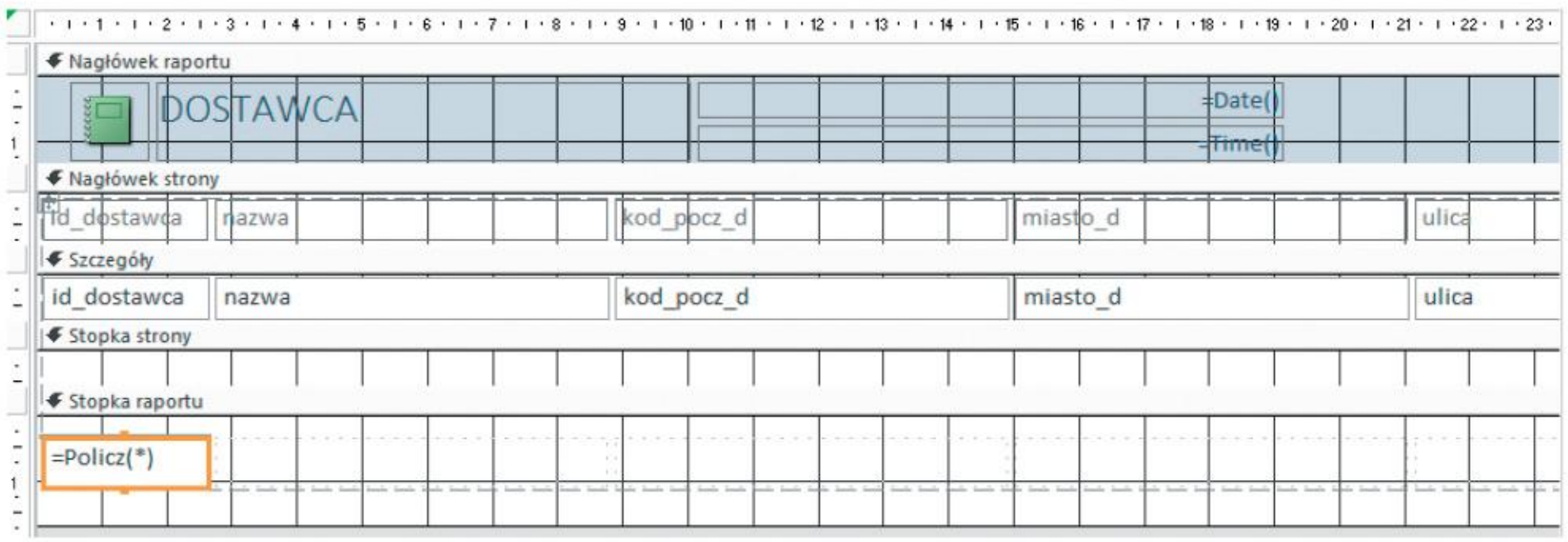
Rys. 14.1. Okno tworzenia raportów

Każdy raport powinien spełniać następujące kryteria:

- mieć estetyczny wygląd i być czytelny;
- mieć pogrupowane i posortowane dane;
- potwierdzać zgodność nagłówka i stopki na każdej stronie wydruku;
- być zgodny z obowiązującym wzorem określonego druku (np. faktura VAT).

Pola wyliczeniowe można tworzyć z użyciem konstruktora wyrażeń lub w kreatorze raportów.

Raport utworzony za pomocą kreatora można zmodyfikować za pomocą arkusza właściwości raportu dostępnego w widoku projektu. Na rys. 14.2 przedstawiono widok projektu raportu z rys. 14.1.



Rys. 14.2. Widok projektu raportu

Każdy raport w MS Access składa się z następujących sekcji:

- **nagłówek raportu** – sekcja drukowana tylko raz, na początku raportu, tuż przed nagłówkiem strony;
- **nagłówek strony** – sekcja drukowana u góry każdej strony wydruku; można tu umieścić informację na temat nazwy bazy bądź raportu, dane typu data / godzina lub dane dotyczące osoby generującej raport;
- **nagłówek grupy** – sekcja drukowana na początku każdej grupy rekordów; umożliwia drukowanie nazwy grupy w nagłówku grupy;
- **szczegóły** – sekcja drukowana raz dla każdego wiersza w źródle rekordów; jest to miejsce, w którym umieszcza się większość kontrolek, które tworzą główną część raportu;
- **stopka grupy** – sekcja drukowana na końcu każdej grupy rekordów; w stopce grupy można drukować informacje podsumowujące dla grupy;
- **stopka strony** – sekcja drukowana na końcu każdej strony; może być wykorzystana do drukowania numerów stron lub informacji dotyczących poszczególnych stron;
- **stopka raportu** – sekcja drukowana tylko raz, na końcu raportu; można w niej umieszczać pola sumujące oraz inne informacje podsumowujące cały raport.

Gotowy raport można zobaczyć za pomocą podglądu wydruku (druga ikona na pasku stanu okna MS Access). Na rys. 14.3 przedstawiono raport gotowy do wydruku.

DOSTAWCA

piątek, 31 stycznia 2020 21:21:36

id_dostawca	nazwa	kod_pocz_d	miasto_d
1	ABCData	00-234	Warszawa
2	Koputerlander	34-234	Łódź
3	Techmagic	32-098	Wrocław

3

Rys. 14.3. Podgląd wydruku raportu

UWAGA

Program MS Access umożliwia również tworzenie raportów zgodnych z siecią WEB, które można publikować w usługach Access Services. Będą one interpretowane przez przeglądarkę internetową.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jaką funkcję w strukturze raportu pełni nagłówek raportu?
2. Jakie cechy powinien mieć raport bazy danych?
3. Wymień dwa formaty plików, do których można przekonwertować plik raportu MS Access.
4. Opisz rodzaje raportów w MS Access.
5. Co należy zrobić, by przed wydrukiem zobaczyć wygląd raportu?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz raport o nazwie Magazyn sumujący wartość wszystkich podzespołów komputerowych wpisanych w bazie danych Sklep_komputerowy w cenach sprzedaży netto.
2. Napisz raport dla wybranego zdarzenia sprzedaży jednej sztuki podzespołów z bazy danych Sklep_komputerowy. Utwórz pole w raporcie obliczające kwotę brutto ceny sprzedaży tego podzespołu.
3. Napisz raport, który będzie zestawieniem wszystkich zrealizowanych zleceń serwisowych bazy danych Serwis.
4. Utwórz raport drukujący zestawienie z zapytania opisanego w przykładzie 13.3.

15 Programowanie w MS Access

ZAGADNIENIA

- Zastosowanie makropoleceń w MS Access
- Skrypty VBA w MS Access

Programowanie w programie MS Access polega na dodawaniu funkcji do tworzonej bazy danych za pomocą **makr** lub skryptów w języku **VBA** (ang. *Visual Basic for Applications*). Głównie jest to potrzebne do obsługi określonych zdarzeń realizowanych w obiektach takich jak formularze czy raporty. W tym rozdziale omówimy tylko podstawy tych zagadnień.

Makra

Makro jest zbiorem akcji wykonywanych jako odpowiedź na określone zdarzenie lub po spełnieniu zdefiniowanych warunków. W czasie tworzenia formularzy, raportów i kontrolek najczęściej korzysta się z makr bez potrzeby pisania kodu w module języka VBA. Makra stanowią podzestaw poleceń dostępnych w języku VBA. Tworzenie makr odbywa się przez wybór akcji z dostępnej listy, a następnie uzupełnienie informacji wymaganych dla każdej akcji. Z makr korzysta się przede wszystkim ze względu na prostotę ich tworzenia oraz ze względu na większe bezpieczeństwo tego rozwiązania w porównaniu do kodu VBA.

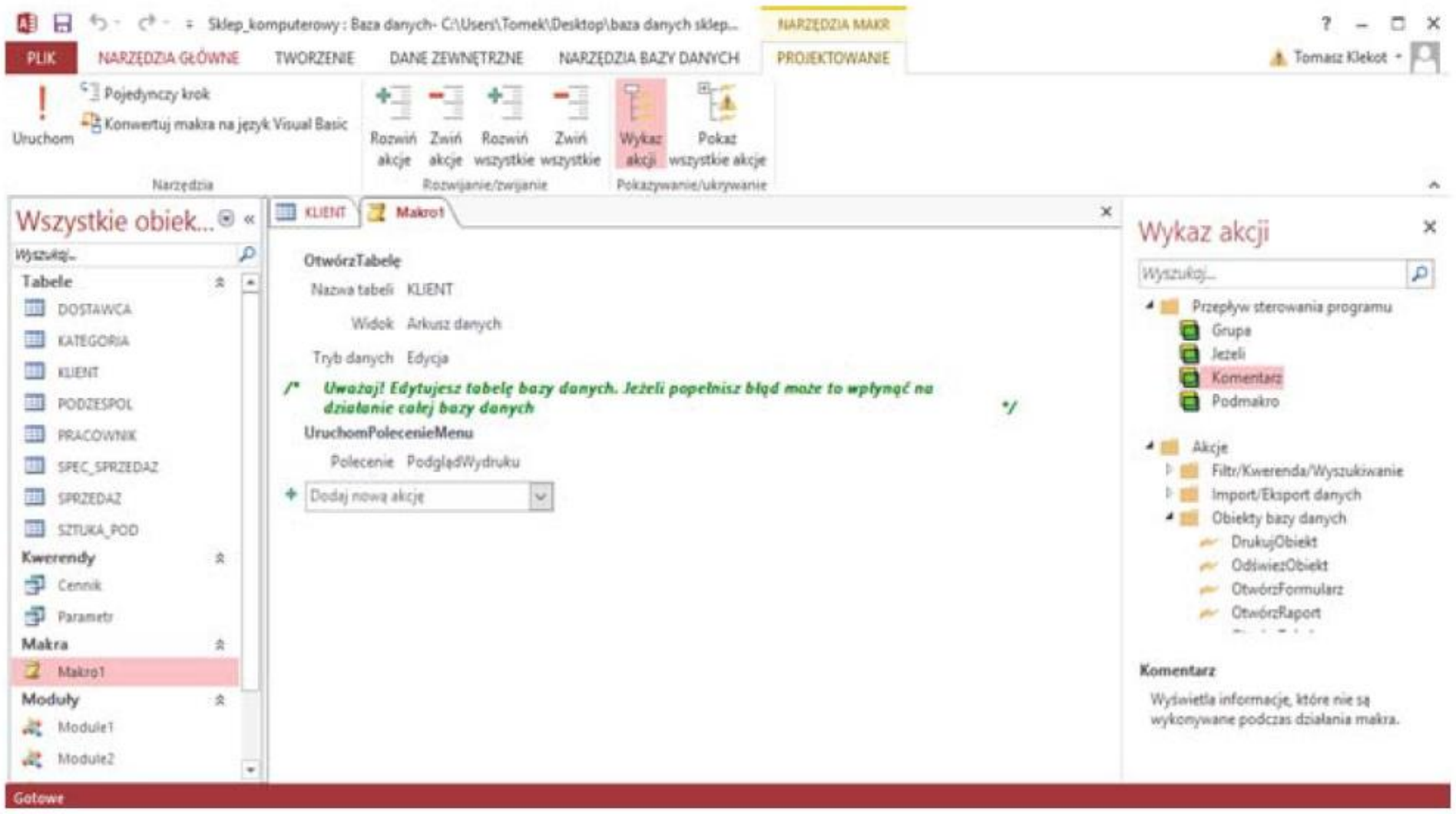
W programie MS Access można wykonywać akcje tuż po pierwszym otwarciu bazy danych. Wymaga to utworzenia makra o nazwie **AutoExec**.

Do tworzenia makr w MS Access wykorzystuje się **konstruktor makr**, który jest dostępny w zakładce **`Tworzenie`** w grupie **`makra i kod`**. Okno konstruktora makr przedstawiono na rys. 15.1.

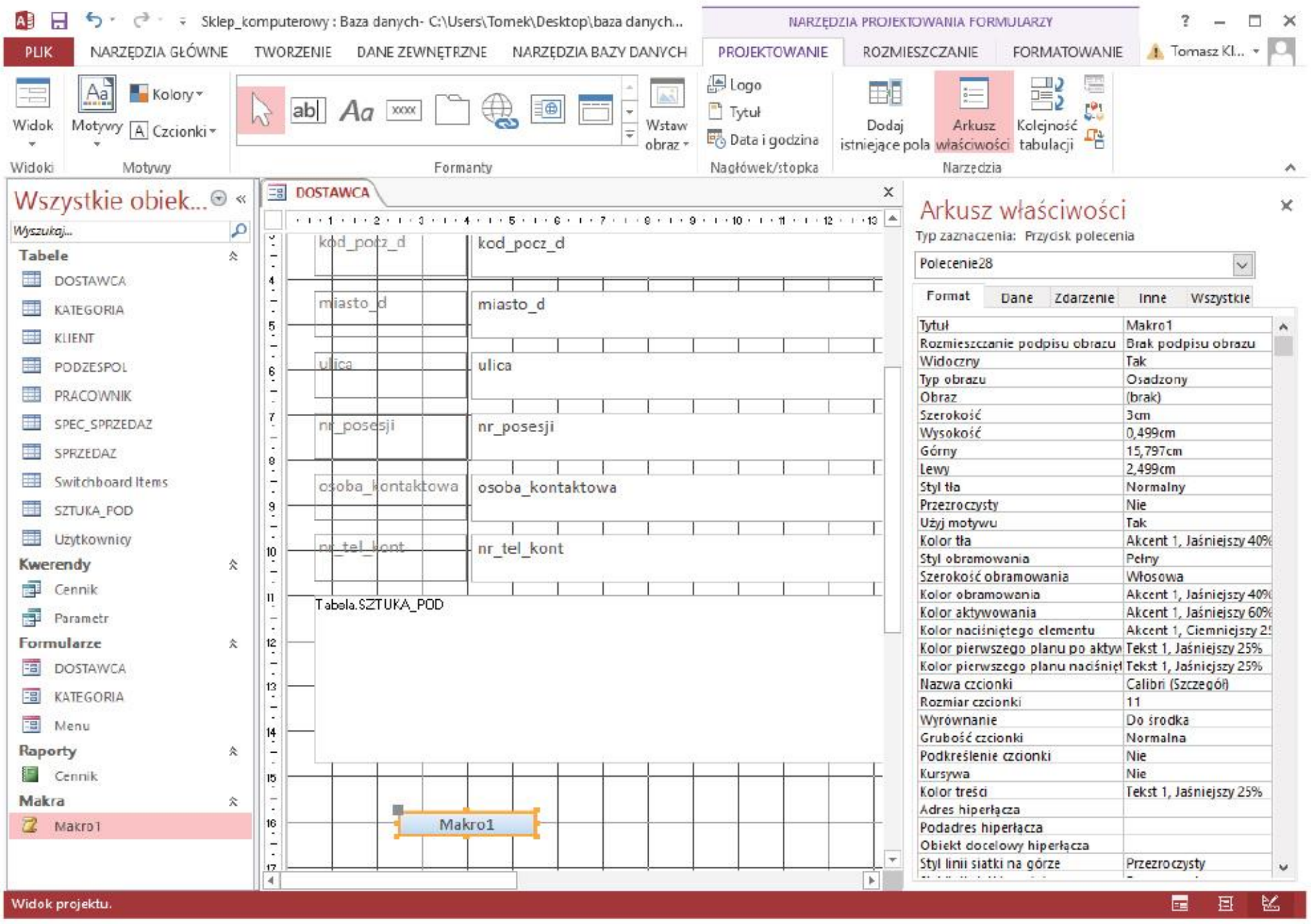
Utworzone makra można uruchamiać z poziomu okna nawigacyjnego bazy danych lub powiązać je z określonymi przyciskami na formularzu lub raporcie. Aby utworzyć makro powiązane z formularzem, należy postępować według następujących kroków:

1. Otwieramy określony formularz w widoku projektu.
2. Przenosimy wybrane makro z okna nawigacyjnego do obszaru projektu formularza. Zostanie wówczas utworzony przycisk powiązany z makrem.
3. Modyfikujemy wygląd oraz pozostałe właściwości przycisku.

Przykład zastosowania makra przedstawiono na rys. 15.2.



Rys. 15.1. Okno konstruktora makr



Rys. 15.2. Przykład powiązania makra z przyciskiem

Skrypty VBA

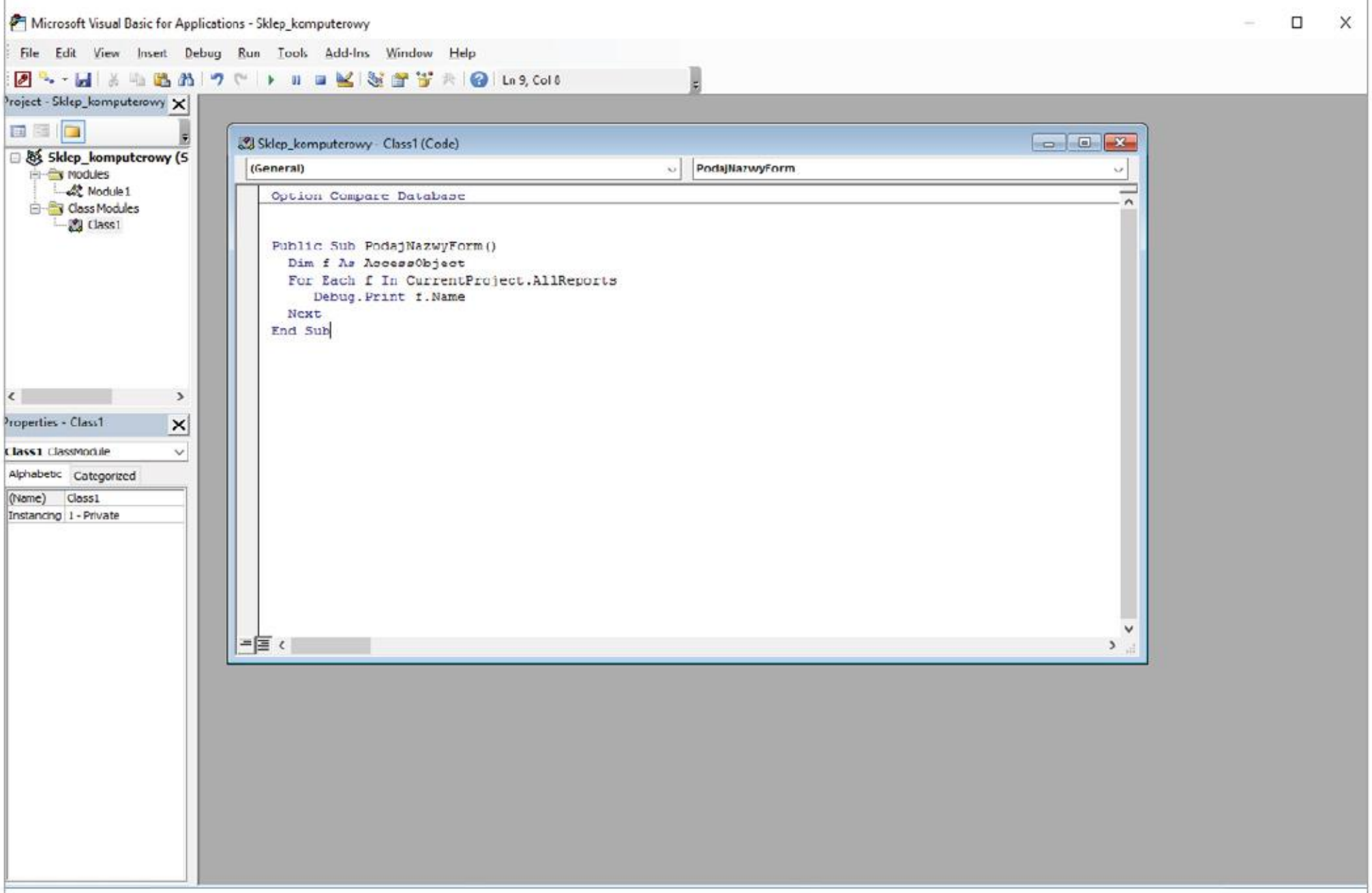
VBA jest językiem programowania zorientowanym obiektowo. Każdy obiekt może mieć określony zbiór metod na nim wykonywanych. Praca w środowisku VBA przebiega z użyciem dwóch rodzajów procedur. Do pierwszego rodzaju należą procedury niezwracające wartości (procedury Sub). Drugim rodzajem procedur, które zwracają wartość, są **funkcje** (ang. *function*). Mogą one być deklarowane jako **procedury publiczne** (ang. *public*) lub **procedury prywatne** (ang. *private*). Procedury mogą wykonywać określone zdarzenie lub uruchomić formułę obliczeniową. Procedury tworzy się według narzuconej składni. Składnię skryptu VBA przedstawiono w przykładzie 15.1.

PRZYKŁAD 15.1

```
Sub PoleKwadratu()
    Dim bok, pole
       Bok= InputBox("podaj długość boku kwadratu")
    If bok > 0 Then
       pole= bok * bok
       MsgBox "pole kwadratu = " & pole
    Else
       MsgBox "niepoprawna wartość"
    End If
End Sub
```

Zbiór deklaracji, instrukcji i procedur przechowywanych razem pod jedną nazwą, który służy organizacji kodu języka Microsoft Visual Basic, nazywa się **modułem**. W programie Microsoft Access są dwa typy modułów: **moduły standardowe** i **moduły klasy**. Kod umieszczony w **module standardowym** jest dostępny z każdego poziomu bazy danych. Kod umieszczony w **module klasy** jest natomiast związany z konkretnym obiektem bazy danych (np. formularzem lub raportem). Do tworzenia programów w VBA służy narzędzie **Microsoft Visual Basic for Applications** (rys. 15.3).

Zbiór modułów tworzy **projekt**. Projekt ma przeważnie tę samą nazwę co baza danych. Można też tworzyć projekty niezwiązane z daną bazą – wtedy ma się do czynienia z uniwersalnym dodatkiem do MS Access.



Rys. 15.3. Okno Microsoft Visual Basic for Applications

Tworzenie modułu standardowego rozpoczyna się od wybory rodzaju modułu w grupie **Moduły**. Okno modułu otwiera się z dwoma liniami kodu:

```
Option Compare Database
Option Explicit
```

Pierwsza linia służy do określenia domyślnej metody porównywania ciągów znaków. Druga wymusza zdefiniowanie zmiennych. Tworzenie nowej procedury należy zacząć od wyboru polecenia **Procedure** z menu **Insert** bądź do wpisania instrukcji **Sub** lub **Function**. W oknie **Add procedure** określa się nazwę procedury, jej typ i zakres.

Tworzenie nowej procedury **Sub** o nazwie **Magazyn**:

```
Sub Magazym ( )
End Sub
```

Tworzenie nowej funkcji o nazwie **LiczMagazyn**:

```
Function LiczMagazyn ( )
End Function
```

Jeżeli procedura zawiera argumenty, należy je wpisać w nawiasy.

PRZYKŁAD 15.2

Moduły klasy mogą mieć przypisane zestawy zdarzeń. Klasy użytkownika mają tylko dwa zdarzenia: **konstruktor** i **destruktor**. Pierwsze uruchamia się w momencie inicjalizacji klasy, drugie – w momencie jej deinicjalizacji.

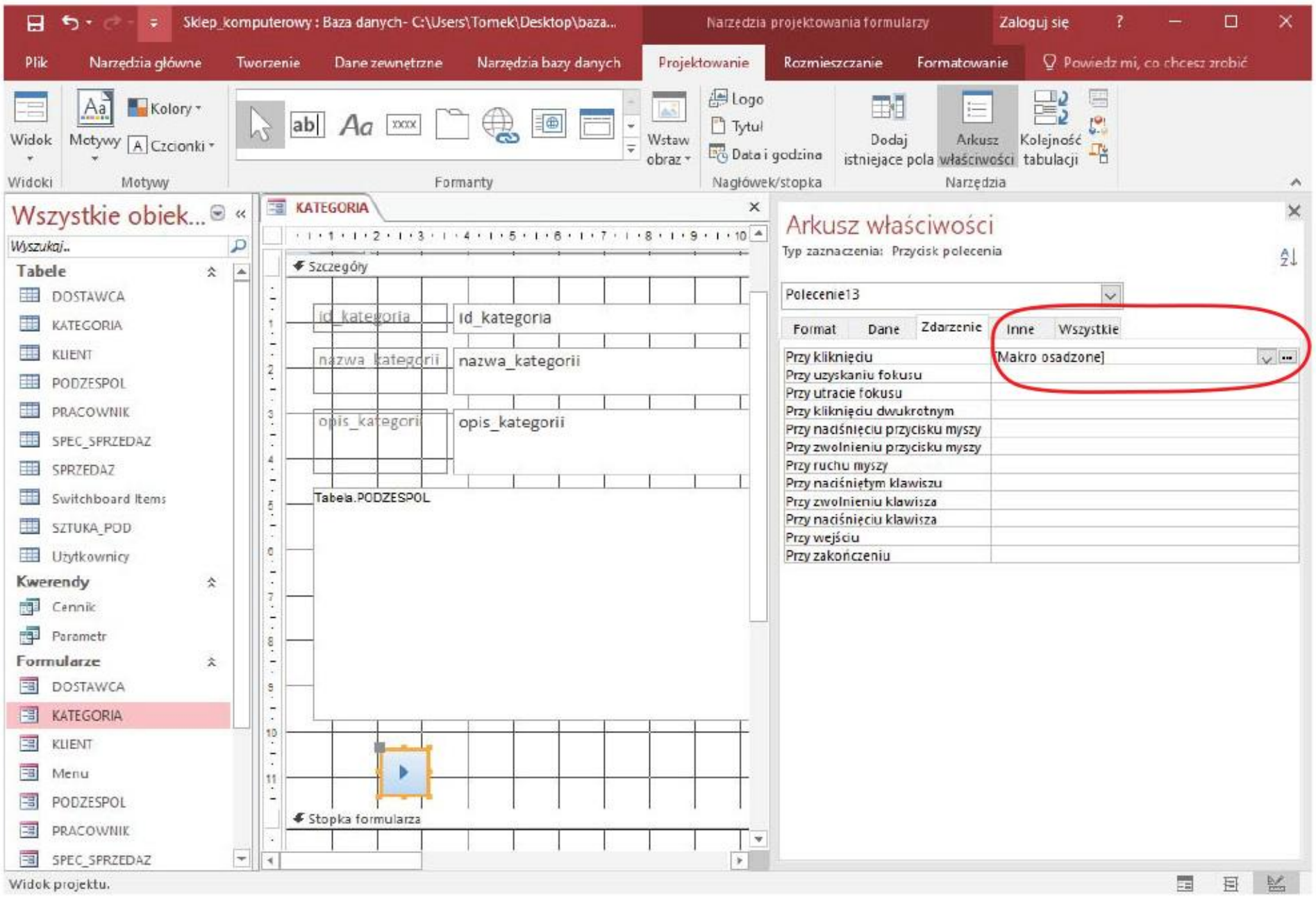
```
Option Compare Database
Option Explicit
Private Sub Class_Initialize ( )
    MsgBox "Rozpoczynasz edycję tabeli"
End Sub
Private Sub Class_Terminate ( )
    MsgBox "Koniec pracy"
End Sub
```

Innym przykładem zastosowania skryptów VBA może być procedura realizująca dodawanie nowego rekordu w istniejącej tabeli bazy danych. Wtedy wykorzystuje się metodę zapisu / odczytu Recordset (zestaw rekordów) modelu DAO (*Database object*). Trzeba również pamiętać o tym, żeby otworzyć źródło danych, czyli tabelę, zaktualizować rekordy (*Update*) oraz zamknąć tabelę (przykład 15.3).

PRZYKŁAD 15.3

```
Sub DodajRekord()
    Dim db As DAO.Database
    Dim rs As DAO.Recordset

    Set db = CurrentDb
    Set rs = db.OpenRecordset("tbKlient")

    rs.AddNew
    rs!imie_k = "Celina"
    rs!nazwisko_k = "Cabacka"
    rs.Update

    rs.Close
    Set rs = Nothing
    Set db = Nothing
End Sub
```

Moduły klasy są najczęściej związane z wykonaniem zdarzenia w określonym obiekcie bazy danych. Aby utworzyć moduł klasy, należy wyświetlić dany formularz lub raport w widoku projektu, a następnie otworzyć arkusz właściwości przycisku, z którym ma być powiązany skrypt, i uruchomić opcję **Buduj** z zakładki **Zdarzenie**. W oknie modułu z rozwijanych list można wybrać obiekty i zdarzenia, które mogą zachodzić względem tego formularza. Po wykonaniu edycji skryptu zapisuje się zmiany i sprawdza poprawność działania przycisku w widoku formularza (rys. 15.4).



Rys. 15.4. Tworzenie modułu klasy

Deklaracja zmiennych

Każda zmienna powinna być zadeklarowana. W trakcie deklarowania zmiennej podaje się jeden z wbudowanych typów dostępnych w języku VBA. Każdy z typów zajmuje odpowiednią ilość pamięci i jest przeznaczony do przechowywania danego typu informacji. Typy zmiennych, które udostępnia VBA, opisano w tabeli 15.1.

Za pomocą kodów w VBA można budować aplikacje oparte na MS Access. VBA nie pozwala jednak na tworzenie skompilowanych aplikacji typu EXE. Makra napisane w VBA zawsze są związane z bazą danych, dla której zostały stworzone.

Tabela 15.1. Typy zmiennych VBA

Typ	Rozmiar w pamięci	Opis	Zakres wartości
Byte	1 bajt	wartości dodatnie	0 do 255
Boolean	zależnie od systemu	dwie wartości	True lub False
Integer	2 bajty	wartości całkowite	–32,768 do +32,767
Currency	8 bajtów	liczby rzeczywiste o stałej liczbie miejsc po przecinku	–922,337,203,685,477.5808 do +922,337,203,685,477.5807
Long	4 bajty	wartości całkowite	–2,147,483,648 do +2,147,483,647
Single	4 bajty	liczby rzeczywiste	–3.4e38 do +3.4e38
Double	8 bajtów	liczby rzeczywiste	–1.8e308 do +1.8e308
String	w zależności od zawartości	zestaw znaków o zmiennej długości	zmienny – do 2 bilionów znaków
Date	8 bajtów	typ przeznaczony dla dat	1 stycznia 100 do 31 grudnia 9999
Object	4 bajty	wskaźnik na obiekt	wskaźnik na dowolny obiekt

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co oznacza skrótowiec VBA?
2. Co to jest makro?
3. Wyjaśnij, w jaki sposób można tworzyć skrypty VBA w MS Access.
4. Co to jest projekt VBA?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. W bazie danych Serwis utwórz makro autonomiczne o nazwie AutoExec, które podczas uruchomiania programu MS Access posłuży do otwarcia jednego z utworzonych wcześniej formularzy. Powiększ maksymalnie rozmiar okna formularza. Dodatkowo zablokuj okno nawigacji tak, aby nie było możliwości usunięcia z niego któregokolwiek z obiektów.
2. Utwórz makro danych, które podczas dodawania nowego rekordu do tabeli Klient w bazie danych Serwis nie pozwoli wprowadzić wartości większej niż 300 do pola Numer domu.

V. SQL – strukturalny język zapytań

- Instrukcje języka SQL
- Język definiowania struktur danych – DDL
- Język do wybierania i manipulowania danymi – DML
- Język definiowania zapytań – DQL
- Funkcje agregujące
- Grupowanie danych
- Łączenie tabel
- Więzy integralności
- Łączenie zapytań, podzapytania i operatory zapytań wewnętrznych
- Użytkownicy i ich uprawnienia w bazie danych

16 Instrukcje języka SQL

ZAGADNIENIA

- Składnia języka SQL
- Wyrażenia i klauzule
- Kategorie pojęć składniowych
- Identyfikatory i operatory
- Typy danych

Język SQL (ang. *Structured Query Language*) jest uniwersalnym językiem zapytań stosowanym w systemach relacyjnych baz danych w celu komunikowania się z bazą. Służy do manipulowania danymi umieszczonymi w relacyjnych bazach danych. Dzięki niemu praca na różnych systemach baz danych sprowadza się do wydawania tych samych lub podobnych komend, tzw. **zapytań SQL**.

Język SQL zaimplementowano w większości relacyjnych systemów baz danych, takich jak: DB2, Oracle, InterBase, MySQL, dBase, Paradox.

Składnię języka SQL można podzielić na cztery części:

- **język definiowania struktur danych**, **DDL** (ang. *Data Definition Language*) – jest wykorzystywany do wszelkiego rodzaju operacji na tabelach, takich jak tworzenie, modyfikacja oraz usuwanie;
- **język do wybierania i manipulowania danymi**, **DML** (ang. *Data Manipulation Language*) – służy do manipulowania danymi umieszczonymi w tabelach, pozwala na wstawienie danych, ich prezentację, modyfikowanie oraz usuwanie;
- **język do zapewnienia bezpieczeństwa dostępu do danych**, **DCL** (ang. *Data Control Language*) – jest używany głównie przez administratorów systemu baz danych do nadawania odpowiednich uprawnień do korzystania z bazy danych;
- **język definiowania zapytań**, **DQL** (ang. *Data Query Language*) – używany do formułowania zapytań do baz danych. Do zakresu tego języka wchodzi polecenie **`SELECT`**.

Instrukcje w SQL składają się z wyrażeń oraz klauzuli. Wyrażenie jest poleceniem (instrukcją), które ustala, co należy zrobić. Klauzula służy do określenia ograniczeń dla danego polecenia i jest zapisywana w formie warunków.

PRZYKŁAD 16.1

```
SELECT nazwisko, imie
FROM studenci
WHERE miasto="Kielce";
```

Po słowie **SELECT** umieszcza się nazwy kolumn, których wartości mają być wyświetlone. W przykładzie 16.1 to pola **nazwisko** oraz **imie**. Po słowie **FROM** wpisuje się nazwę tabeli lub tabel, w których są zawarte poszukiwane informacje. Klauzula **WHERE** pozwala na określenie warunku wybierania danych. Polecenie kończy się średnikiem.

W języku SQL stosuje się następujące kategorie pojęć składniowych:

- identyfikatory,
- literały,
- operatory.

Identyfikator definiuje obiekt bazy danych, tj. tabelę lub kolumnę. Każdy obiekt ma unikatową nazwę. Identyfikatory muszą spełniać następujące reguły:

- nie mogą być dłuższe niż 128 znaków;
- mogą zawierać litery, cyfry oraz znaki @, $, #;
- nie mogą mieć spacji;
- muszą zaczynać się literą;
- nie mogą być słowem kluczowym języka SQL.

Wszystkie wartości liczbowe, ciągi znaków oraz daty są traktowane jako stałe, czyli literały.

Operatory dzieli się na:

- arytmetyczne: suma **+**, różnica **–**, iloczyn *****, iloraz **/**, dzielenie modulo **%**;
- znakowe: konkatenacja **+**, symbol zastępujący dowolny ciąg znaków **%**, symbol zastępujący jeden znak **_**;
- logiczne: koniunkcja **AND**, alternatywa **OR**, negacja **NOT**;
- porównania: **<**, **>**, **<=**, **=>**, **<>**;
- operatory specjalne: **BETWEEN...AND**, **LIKE**, **IN**.

Tabela 16.1. Typy danych w SQL

Typ	Opis
CHAR[Length]	pole o stałej długości, przechowuje od 0 do 255 znaków
VARCHAR[Length]	pole tekstowe o zmiennej długości
TINYTEXT	łańcuch o maksymalnej długości 255 znaków
TEXT	łańcuch o maksymalnej długości 65 535 znaków

Typ	Opis
`MEDIUMTEXT`	łańcuch o maksymalnej długości 16 777 215 znaków
`LONGTEXT`	łańcuch o maksymalnej długości 4 294 967 295 znaków
`TINYINT[Length]`	liczby z zakresu od –128 do 127 lub liczby dodatnie od 0 do 255
`SMALLINT[Length]`	liczby z zakresu od –32 768 do 32 767 lub liczby dodatnie od 0 do 65 535
`MEDIUMINT[Length]`	liczby z zakresu od –8 388 608 do 8 388 607 lub liczby dodatnie od 0 do 16 777 215
`INT[Length]`	liczby z zakresu od –2 147 483 648 do 2 147 483 647 lub liczby dodatnie od 0 do 4 294 967 295
`BIGINT[Length]`	liczby z zakresu od –9 223 372 036 854 775 808 do 9 223 372 036 854 775 807 lub liczby dodatnie od 0 do 18 446 744 073 709 551 615
`FLOAT`	mała liczba rzeczywista, zmiennoprzecinkowa
`DOUBLE[Length, Decimals]`	duża liczba rzeczywista, zmiennoprzecinkowa
`DATE`	data w formacie YYYY-MM-DD
`DATETIME`	data w formacie YYYY-MM-DD HH:MM:SS
`TIME`	data w formacie HH:MM:SS
`MONEY`	typ danych służący do przechowywania wartości pieniężnych (dokładność do jednej dziesięciotysięcznej jednostki, przechowuje liczby z zakresu: od –263 do 263–1)

SPRAWDŹ SWOJĄ WIEDZĘ

1. Omów kategorie pojęć składniowych w języku SQL.
2. Co to jest klauzula?
3. Wymień i omów operatory stosowane w SQL.
4. Przedstaw zasady stosowane podczas tworzenia identyfikatorów.
5. Omów typy danych w SQL.

17 Język definiowania struktur danych – DDL

ZAGADNIENIA

- Tworzenie i usuwanie baz danych oraz tabel
- Modyfikowanie tabel
- Atrybuty pól w tabelach

Język **DDL** służy do tworzenia, modyfikacji i usuwania bazy danych oraz jej tabel. Do instrukcji DDL należą następujące polecenia:

- **`CREATE nazwa_obiektu`** – tworzenie obiektu (bazy, tabeli);
- **`ALTER nazwa_obiektu`** – modyfikacja obiektu;
- **`DROP nazwa_obiektu`** – usuwanie obiektu.

Aby zacząć pracę z bazą danych, należy ją najpierw utworzyć. Na potrzeby ćwiczeń w tej części podręcznika będzie wykorzystana baza o nazwie **`SZKOLA`** o strukturze przedstawionej na rys. 17.1.

Rys. 17.1. Schemat bazy szkola

Tworzenie i usuwanie baz danych oraz tabel

Aby utworzyć nową bazę danych, należy użyć polecenia:

```
CREATE DATABASE nazwa_bazy;
```

PRZYKŁAD 17.1

```
CREATE DATABASE szkola;
```

Równie prostym poleceniem jest usunięcie bazy danych:

```
DROP DATABASE nazwa_bazy;
```

PRZYKŁAD 17.2

```
DROP DATABASE szkola;
```

Usunięcie bazy danych jest możliwe tylko wtedy, jeśli dysponuje się odpowiednim uprawnieniem do bazy i nikt nie jest z nią połączony w danym momencie.

Po utworzeniu bazy danych można przystąpić do tworzenia tabel oraz do określenia połączeń między nimi. Dane w typowych bazach danych są przechowywane w tzw. relacjach, czyli tabelach. Tabele składają się z rekordów (wierszy) i określonego typu pól.

Aby utworzyć tabelę, należy użyć polecenia **CREATE TABLE**:

```
CREATE TABLE nazwaTabeli(
nazwaKolumny1 TypDanych,
nazwaKolumny2 TypDanych,
.....
);
```

PRZYKŁAD 17.3

Tabelę książki utworzy następujące polecenie:

```
CREATE TABLE ksiazki (
 idksiazki int(11),
 imieautora text,
 nazwiskoautora text,
 tytul text,
 cena float
);
```

Podczas tworzenie tabel należy pamiętać o następujących zasadach:

- każda tabela powinna mieć unikatową nazwę;
- każda kolumna w tabeli powinna mieć unikatową nazwę;
- każda kolumna w tabeli musi mieć określony typ;
- kolumna znakowa musi mieć określoną długość maksymalną.

Należy pamiętać, że utworzone tabele nie są wypełnione danymi.

Aby usunąć tabelę, należy użyć instrukcji:

```
DROP TABLE nazwa_tabeli;
```

PRZYKŁAD 17.4

```
DROP TABLE ksiazki;
```

Modyfikowanie tabel

Podczas modyfikowania utworzonej tabeli należy zastosować instrukcję ALTER TABLE. Umożliwia ona m.in. dodanie nowej kolumny, skasowanie już utworzonej kolumny bądź dodanie lub usunięcie danego ograniczenia.

Aby dodać nową kolumnę do już istniejącej tabeli, należy użyć polecenia:

```
ALTER TABLE nazwaTabeli
ADD nazwaKolumny TypDanych;
```

PRZYKŁAD 17.5

```
ALTER TABLE ksiazki
ADD wydawnictwo varchar(15);
```

Powyższe polecenie utworzy dodatkową kolumnę **wydawnictwo**, będzie to ostatnia kolumna w tabeli **ksiazki**.

Aby utworzyć kolumnę **rokwydania** za kolumną **cena**, należy użyć polecenia:

```
ALTER TABLE ksiazki
ADD rokwydania int(4) after cena;
```

Aby usunąć kolumnę z istniejącej tabeli, należy użyć polecenia:

```
ALTER TABLE nazwaTabeli
DROP COLUMN nazwaKolumny;
```

PRZYKŁAD 17.6

```
ALTER TABLE ksiazki
DROP COLUMN wydawnictwo;
```

Atrybuty pól w tabelach

W tabelach tworzonych w bazach danych można narzucić pewne ograniczenia określające dokładnie strukturę danych, jakie mogą zostać zapisane w danej kolumnie. Ograniczenia dotyczące kolumn mogą być definiowane w trakcie tworzenia tabeli lub jej modyfikowania.

Atrybut **NOT NULL** oznacza, że w danej kolumnie nie mogą znajdować się wartości puste. Aby zabronić wstawiania do kolumny wartości **NULL**, podczas tworzenia tabeli należy po nazwie kolumny wpisać **NOT NULL**. Można również jawnie zezwolić na wprowadzanie do kolumny wartości **NULL** przez wpisanie po jej nazwie słowa **NULL**.

PRZYKŁAD 17.7

```
CREATE TABLE klient(
 Idklienta int(11) NOT NULL,
 imie text NOT NULL,
 nazwisko text NOT NULL,
 miejscowosc text
);
```

Klucz podstawowy (**PRIMARY KEY**) to kolumna lub grupa kolumn, które w sposób jednoznaczny definiują wiersz w tabeli. Kolumna z tym atrybutem jest unikatowa i automatycznie indeksowana. Klucz podstawowy w tabelach odgrywa bardzo ważną rolę, ponieważ służy do jednoznacznego identyfikowania rekordów w danej tabeli. Logiczne jest zatem to, że dane w tej kolumnie (lub kolumnach) nie mogą się powtarzać (lub nie mogą występować w takim samym układzie wielokrotnie).

Tworzenie nowej tabeli z kluczem podstawowym:

```
CREATE TABLE zamowienie(
 idzamowienie int(11) PRIMARY KEY,
 Idklienta int(11) NOT NULL,
 idksiazki int(11) NOT NULL,
 data date NOT NULL,
 status text
);
```

Aby w utworzonej wcześniej tabeli ustalić klucz podstawowy, należy użyć polecenia **ALTER**.

PRZYKŁAD 17.8

```
ALTER TABLE ksiazki
ADD CONSTRAINT PRIMARY KEY(idksiazki);
```

Podczas wstawiania nowych danych do tabeli dobrze jest nie podawać wartości dla klucza, ponieważ użytkownik bazy po prostu może nie wiedzieć, jakie ID ma ostatnio dodany rekord. Najlepiej, żeby wartość tej kolumny była automatycznie uzupełniana przez serwer bazodanowy.

Autoinkrementację uzyskuje się za pomocą komendy **AUTO_INCREMENT**.

PRZYKŁAD 17.9

```
CREATE TABLE osoba(
id INT NOT NULL AUTO_INCREMENT,
Imie Varchar(200),
```

```
Nazwisko VARCHAR(200),
PRIMARY KEY (id)
);
```

UWAGA

Dodanie tej własności dla kolumny jest możliwe tylko podczas jej tworzenia, nie ma możliwości zaktualizowania utworzonej już kolumny.

Aby używać danych, które nie są automatycznie generowane przez serwer bazodanowy, jako klucza podstawowego, trzeba zagwarantować niepowtarzalność danych (unikatowość), tak aby można było jednoznacznie zidentyfikować poszczególny rekord w tabeli. W bazach MySQL wykonuje się to za pośrednictwem słowa kluczowego **UNIQUE**.

PRZYKŁAD 17.10

```
CREATE TABLE nazwaTabeli (
ID INT PRIMARY KEY UNIQUE,
Imie VARCHAR(200),
Nazwisko VARCHAR(200)
);
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Do czego służy polecenie **CREATE**?
2. Za pomocą jakiego polecenia usuwa się tabelę z bazy?
3. Wymień zasady obowiązujące podczas tworzenia tabeli.
4. Wymień i omów atrybuty pól tabeli.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Nawiąż połączenie z bazą MySQL i utwórz bazę o nazwie **SZKOŁA**. Zaprojektuj w niej tabelę **UCZEN** oraz **PRZEDMIOT**. Tabela **UCZEN** ma mieć następujące pola: **IdUcznia**, **Imie**, **Nazwisko**, **Data_urodzenia**, **Miejsce_ur**, **Plec**, **Klasa**, a tabela **PRZEDMIOT**: **IdPrzedmiotu, Nazwa_przedmiotu**.
   Samodzielnie określ typy poszczególnych pól oraz ustaw klucz podstawowy dla każdej z tabel. Przedyskutuj w klasie, jaką tabelę dodatkowo powinna mieć baza, aby można było każdemu uczniowi wpisać oceny z poszczególnych przedmiotów.
2. Utwórz bazę o nazwie KSIĘGARNIA, a w niej tabele zgodnie z projektem przedstawionym na rys. 17.1.

18 Język do wybierania i manipulowania danymi – DML

ZAGADNIENIA

- Instrukcja `INSERT`
- Instrukcja `UPDATE`
- Instrukcja `DELETE`

Język manipulacji danymi służy do pobierania i modyfikowania danymi. Do instrukcji DML należą następujące polecenia:

- **`INSERT`** – wstawianie nowych wierszy w tabeli;
- **`UPDATE`** – zmiana zawartości istniejącego wiersza;
- **`DELETE`** – usuwanie wierszy w tabeli.

Instrukcja INSERT

Polecenie **`INSERT`** służy do wstawiania nowego rekordu do tabeli i ma następującą postać:

```
INSERT INTO nazwa_tabeli(pole1, pole2, …)
VALUES (wart1, wart2, …);
```

Jeśli pominie się nazwy kolumn w tabeli, to dane podane po słowie VALUES zostaną wprowadzone do kolejnych kolumn tabeli.

PRZYKŁAD 18.1

```
INSERT INTO klient(Idklienta,imie,nazwisko,miejscowosc) VALUES
(15,'Adam', 'Trzeciak', 'Zgierz');
```

Polecenie z przykładu 18.1 wstawiło dane nowego klienta w tabeli **`Klient`**.

Gdyby kolumna **`Idklienta`** była typu **`AUTO_INCREMENT`**, wartość 15 można pominąć w klauzuli **`VALUES`**.

Jeżeli podczas definiowania tabeli dla określonej kolumny zostało zabronione zapisywanie wartości nieznanej (**`NULL`**), to próba zapisania nowego wiersza bez podania wartości dla tej kolumny zakończy się niepowodzeniem.

PRZYKŁAD 18.2

```
INSERT INTO klient(imie,nazwisko,miejscowosc) VALUES ('Joanna',
'Pacak', 'Sopot'), ('Jerzy', 'Kupiec', 'Szczecin'), ('Anna',
'Mateja', 'Sopot');
```

Do tabeli **Klient** wprowadzono trzy nowe rekordy. Jak widać, pominięto pole **Idklienta**, ponieważ nadano mu atrybut autoinkrementacji.

Instrukcja UPDATE

Do aktualizowania danych w bazie służy polecenie **UPDATE** następującej postaci:

```
UPDATE... SET...
```

Po słowie **UPDATE** należy podać tabelę, w której będą wprowadzane zmiany, natomiast po klauzuli **SET** podaje się nazwę kolumny i jej nową wartość. Wykonanie tej instrukcji spowoduje zmianę we wszystkich wierszach danej kolumny.

PRZYKŁAD 18.3

```
UPDATE klient SET miejscowosc="Katowice";
```

Polecenie to spowoduje, że w tabeli **Klient** w kolumnie **miejscowosc** zostanie wpisana wartość „Katowice".

Jeśli modyfikacja ma dotyczyć wybranych wierszy, to do instrukcji należy dołączyć klauzulę **WHERE**.

PRZYKŁAD 18.4

```
UPDATE klient SET miejscowosc="Katowice" WHERE imie='Agnieszka' and nazwisko='Nowak';
```

Za pomocą klauzuli **WHERE** uszczegółowiono działanie instrukcji **UPDATE**. Nazwa miejscowości zostanie zmieniona tylko dla Agnieszki Nowak. W poleceniu zastosowano operator iloczynu logicznego **AND**, którego zadaniem jest wybranie osoby o podanym nazwisku i imieniu. W przypadku zastosowania operatora alternatywy logicznej **OR** miejscowość zostałaby zmieniona dla wszystkich Agnieszek (niezależnie od ich nazwiska) oraz dla wszystkich Nowaków (niezależnie od ich imienia).

PRZYKŁAD 18.5

```
UPDATE ksiazki SET cena=cena-5 WHERE cena>55;
```

Polecenie spowoduje obniżenie cen tylko tych książek, które kosztują powyżej 55 złotych.

Instrukcja DELETE

Polecenie **DELETE** pozwala na usuwanie informacji z wybranej tabeli bazy danych. Ma ono następującą postać:

```
DELETE FROM nazwa_tabeli [WHERE warunek] [LIMIT liczba];
```

Użycie polecenia bez klauzuli **WHERE** oraz **LIMIT** poskutkuje usunięciem wszystkich rekordów znajdujących się w danej tabeli.

Dodanie klauzuli **WHERE** pozwala określić warunek, który muszą spełnić wiersze przeznaczone do usunięcia.

Klauzula **LIMIT** ogranicza liczbę usuwanych wierszy z danej tabeli.

PRZYKŁAD 18.6

```
DELETE FROM klient WHERE nazwisko="Rutkowski";
```

Polecenie usunie dane wszystkich klientów o nazwisku Rutkowski.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Omów instrukcję służącą do tworzenia nowych rekordów tabeli.
2. Do czego służy instrukcja **UPDATE**? Podaj przykład zastosowania.
3. Opisz składnię instrukcji **DELETE**. Podaj przykład zastosowania.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Do bazy **SZKOLA** utworzonej w rozdziale 17 wprowadź informacje o pięciu uczniach. Następnie wpisz do tabeli **PRZEDMIOTY** nazwy przedmiotów: matematyka, fizyka, chemia, geografia, biologia oraz przyporządkuj każdemu uczniowi oceny z tych przedmiotów.

19 Język definiowania zapytań – DQL

ZAGADNIENIA

- Instrukcja `SELECT`
- Klauzula `DISTINCT`
- Klauzula `WHERE`
- Operator `LIMIT`
- Operator `BETWEEN...AND`

Instrukcja `SELECT`

Instrukcja **`SELECT`** pozwala na określenie tego, jakie dane zostaną wyświetlone z danej tabeli (lub tabel). Instrukcja jest równoważna kwerendzie wybierającej tworzonej w programie MS Access.

Postać instrukcji jest następująca:

```
SELECT nazwa_kolumny1, nazwa_kolumny FROM nazwa_tabeli;
```

UWAGA

Wszystkie ćwiczenia dotyczące tworzenia zapytań będą przedstawione na bazie danych **`księgarnia`** (Załącznik 1).

PRZYKŁAD 19.1

Polecenie:

```
SELECT imie, nazwisko FROM klient;
```

wyświetli następujące dane:

imie	nazwisko
Piotr	Lewandowski
Jan	Nowak
Maciej	Nowak
Agnieszka	Jankowska
Tomasz	Mazur
Marek	Kowalski
Artur	Rutkowski
Mateusz	Skorupa
Jerzy	Rutkowski
Anna	Karenina
Maria	Lewandowska
Henryk	Kowalski
Agnieszka	Nowak
Piotr	Krupski

Rys. 19.1. Wynik działania prostej instrukcji **`SELECT`**

Aby wyświetlić wszystkie kolumny danej tabeli, należy użyć symbolu *****.

PRZYKŁAD 19.2

```
SELECT * FROM ksiazka;
```

Efekt polecenia jest następujący:

idksiazki	imieautora	nazwiskoautora	tytul	cena
1	Jan	Michalak	Zaawansowane programowanie w PHP	47.29
2	Andrzej	Krawczyk	Windows 8 PL. Zaawansowana administracja systemem	49.99
3	Piotr	Jakubowski	HTML5. Tworzenie witryn	53.65
4	Tomasz	Kowalski	Urządzenia techniki komputerowej	34.15
5	Jan	Pasternak	PHP. Tworzenie nowoczesnych stron WWW	29.99
6	Robin	Nixon	PHP, MySQL i JavaScript	99
7	Jolanta	Pokorska	Tworzenie stron internetowych	23.95
8	Krzysztof	Pytel	Pracownia aplikacji internetowych	35.4

Rys. 19.2. Efekt zastosowania symbolu ***** w instrukcji **SELECT**

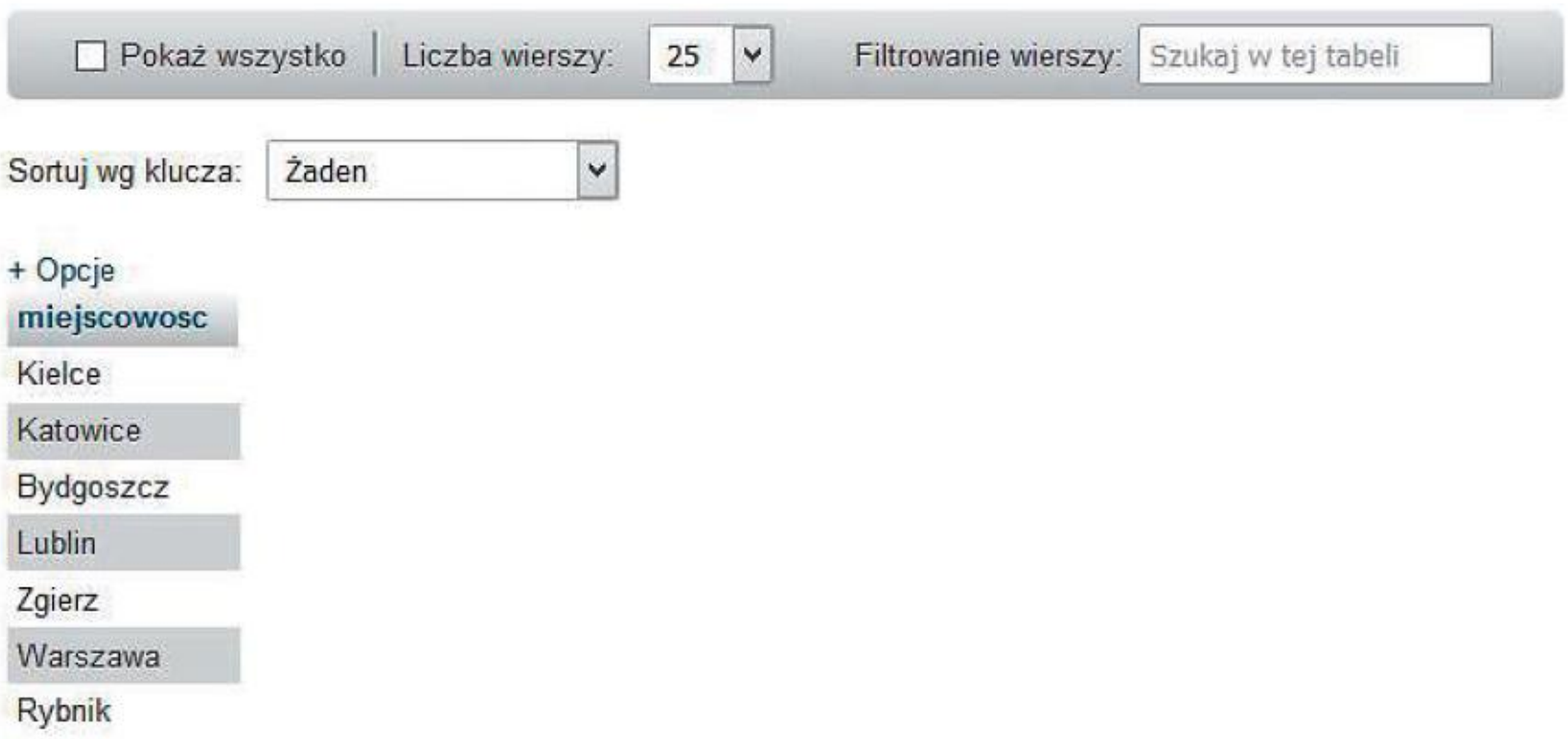
Klauzula DISTINCT

Klauzula **DINSTINCT** pozwala na wyświetlanie niepowtarzających się danych. Przykładem może być wyświetlenie nazwy miejscowości, w których mieszkają klienci księgarni.

PRZYKŁAD 19.3

```
SELECT DISTINCT miejscowosc FROM klient;
```

Otrzymamy następujące wyniki:



Rys. 19.3. Efekt użycia klauzuli **DISTINCT**

Obliczanie wyrażeń w instrukcji SELECT

Oprócz wyświetlania określonych informacji z bazy danych, można także tworzyć wyrażenia, które będą wykonywały obliczenia na danych z kolumn tabeli.

PRZYKŁAD 19.4

```
SELECT tytul, 0.8*cena as 'Cena promocyjna' FROM ksiazka;
```

Instrukcja wyświetli tytuły dostępne w księgarni wraz z ceną promocyjną książek (po odliczeniu 20 % ceny). Kolumna z nową ceną nazywa się **Cena promocyjna**. Otrzymamy następujące wyniki:

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Szukaj w tej tabeli

Sortuj wg klucza: Żaden

+ Opcje

tytul	Cena promocyjna
Zaawansowane programowanie w PHP	37.832000732421875
Windows 8 PL. Zaawansowana administracja systemem	39.99200134277344
HTML5. Tworzenie witryn	42.92000122070313
Urządzenia techniki komputerowej	27.320001220703126
PHP. Tworzenie nowoczesnych stron WWW	23.991999816894534
PHP, MySQL i JavaScript	79.2
Tworzenie stron internetowych	19.160000610351563
Pracownia aplikacji internetowych	28.320001220703126

Rys. 19.4. Tworzenie wyrażeń w instrukcji **SELECT**

Jak widać, wyświetlone wartości liczbowe mają zbyt dużo liczb po przecinku. Są to ceny książek, dlatego należy zaokrąglić wartości do dwóch miejsc po przecinku. W tym celu stosuje się funkcję **ROUND**.

```
SELECT tytul, round(0.8*cena,2) as 'Cena promocyjna' FROM ksiazka
```

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

tytul	Cena promocyjna
Zaawansowane programowanie w PHP	37.83
Windows 8 PL. Zaawansowana administracja systemem	39.99
HTML5. Tworzenie witryn	42.92
Urządzenia techniki komputerowej	27.32
PHP. Tworzenie nowoczesnych stron WWW	23.99
PHP, MySQL i JavaScript	79.20
Tworzenie stron internetowych	19.16
Pracownia aplikacji internetowych	28.32

Rys. 19.5. Zastosowanie funkcji **ROUND**

W celu połączenia ze sobą kliku pól w bazie danych MySQL należy użyć polecenia **CONCAT()**. Ta funkcja łączy ze sobą ciągi znaków, które oddziela się przecinkami.

PRZYKŁAD 19.5

```
SELECT tytul as 'Tytuł książki',
CONCAT('imieautora', 'nazwiskoautora') as 'Imię i nazwisko
autora' FROM ksiazka;
```

```
SELECT tytul as 'Tytuł książki', CONCAT(imieautora,' ',nazwiskoautora) as 'Imię i nazwisko autora' FROM ksiazka
```

☐ Profilowanie [Edytuj w linii] [Edytuj]

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę Sortuj wg klucza: Żaden

+ Opcje

Tytuł książki	Imię i nazwisko autora
Zaawansowane programowanie w PHP	Jan Michalak
Windows 8 PL. Zaawansowana administracja systemem	Andrzej Krawczyk
HTML5. Tworzenie witryn	Piotr Jakubowski
Urządzenia techniki komputerowej	Tomasz Kowalski
PHP. Tworzenie nowoczesnych stron WWW	Jan Pasternak
PHP, MySQL i JavaScript	Robin Nixon
Tworzenie stron internetowych	Jolanta Pokorska
Pracownia aplikacji internetowych	Krzysztof Pytel

Rys. 19.6. Zastosowanie funkcji **CONCAT**

Sortowanie danych w klauzuli SELECT

Aby posortować wyświetlane dane, należy użyć klauzuli **ORDER BY**. Po klauzuli umieszcza się nazwy lub numery kolumn, według których będzie wykonywane sortowanie. Domyślnie dane są sortowane rosnąco. Do tego rodzaju sortowania służy słowo kluczowe **ASC**. Aby posortować dane malejąco, należy użyć słowa **DESC**.

PRZYKŁAD 19.6

```
SELECT nazwisko, imie, miejscowosc FROM klient ORDER BY
nazwisko;
```

```
SELECT nazwisko, imie, miejscowosc FROM klient ORDER BY nazwisko
```

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

nazwisko ▲ 1	imie	miejscowosc
Jankowska	Agnieszka	Lublin
Karenina	Anna	Katowice
Kowalski	Marek	Lublin
Kowalski	Henryk	Bydgoszcz
Krupski	Piotr	Zgierz
Lewandowska	Maria	Bydgoszcz
Lewandowski	Piotr	Kielce
Mazur	Tomasz	Zgierz
Nowak	Jan	Katowice
Nowak	Maciej	Bydgoszcz
Nowak	Agnieszka	Warszawa
Rutkowski	Artur	Kielce
Rutkowski	Jerzy	Rybnik
Skorupa	Mateusz	Warszawa

Rys. 19.7. Zastosowanie klauzuli **ORDER BY**

Po słowie „nazwisko" nie użyto ani słowa **ASC**, ani **DESC**. Dlatego też domyślnie nazwiska posortowano rosnąco. Zamiast użycia nazwy kolumny **nazwisko** można również napisać numer wyświetlanej kolumny, czyli wartość **1**.

Dane mogą być posortowane według wielu kolumn. Klauzule określające sposób sortowania są od siebie niezależne. Oprócz tego można odwoływać się do kolumn niewymienionych w klauzuli **SELECT**.

PRZYKŁAD 19.7

```
SELECT nazwisko, imie, miejscowosc FROM klient ORDER BY
nazwisko, miejscowosc;
```

```
SELECT nazwisko, imie, miejscowosc FROM klient ORDER BY nazwisko, miejscowosc
```

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

nazwisko ▲ 1	imie	miejscowosc ▲ 2
Jankowska	Agnieszka	Lublin
Karenina	Anna	Katowice
Kowalski	Henryk	Bydgoszcz
Kowalski	Marek	Lublin
Krupski	Piotr	Zgierz
Lewandowska	Maria	Bydgoszcz
Lewandowski	Piotr	Kielce
Mazur	Tomasz	Zgierz
Nowak	Maciej	Bydgoszcz
Nowak	Jan	Katowice
Nowak	Agnieszka	Warszawa
Rutkowski	Artur	Kielce
Rutkowski	Jerzy	Rybnik
Skorupa	Mateusz	Warszawa

Rys. 19.8. Sortowanie według wielu kolumn

Wybieranie wierszy – klauzula WHERE

W klauzuli **WHERE** formułuje się warunek, który odpowiada warunkowi wyboru w algebrze relacyjnej. Określa on ograniczenia, jakie mają spełniać rekordy, aby zostać wybrane w danym zapytaniu.

Wybieranie pewnych wierszy z tabeli nazywa się **selekcją rekordów**.
Klauzula **WHERE** występuje bezpośrednio po klauzuli **FROM**.

PRZYKŁAD 19.8

```
SELECT imie, nazwisko FROM klient WHERE miejscowosc="Bydgoszcz";
```

Polecenie wyświetli imiona i nazwiska klientów z Bydgoszczy:

```
SELECT imie, nazwisko FROM klient WHERE miejscowosc="Bydgoszcz"
```

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

imie	nazwisko
Maciej	Nowak
Maria	Lewandowska
Henryk	Kowalski

Rys. 19.9. Zastosowanie klauzuli **WHERE**

Wyrażenie **miejscowosc="Bydgoszcz"** można zastąpić wyrażeniem:

miejscowosc LIKE "Bydgoszcz".

Efekt będzie identyczny. Słowo **LIKE** oznacza po prostu „podobny do". Operator **LIKE** sprawdza, czy wartość tekstowa odpowiada podanemu wzorcowi. Umożliwia więc wykonywanie częściowych porównań, takich jak „zaczynający się od tekstu", „kończący się na tekście" lub „zawierający tekst". Podczas tworzenia wzorca stosuje się znaki wieloznaczne: % (zastępuje sekwencję dowolnej długości) oraz _ (odpowiada jednemu znakowi w przeszukiwanym tekście).

PRZYKŁAD 19.9

SELECT nazwisko, imie, miejscowosc FROM klient WHERE nazwisko LIKE "K%";

Polecenie wyświetli klientów księgarni, których nazwisko rozpoczyna się na literę **K**:

```
SELECT nazwisko, imie, miejscowosc FROM klient WHERE nazwisko LIKE "K%"
```

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

nazwisko	imie	miejscowosc
Kowalski	Marek	Lublin
Karenina	Anna	Katowice
Kowalski	Henryk	Bydgoszcz
Krupski	Piotr	Zgierz

Rys. 19.10. Zastosowanie operatora **LIKE**

Operator LIMIT

Klauzula **LIMIT** pozwala ograniczyć liczbę wybieranych wierszy. Umieszcza się ją na końcu zapytania. Polecenie z zastosowaniem klauzuli **LIMIT** ma postać:

```
SELECT lista_kolumn FROM tabela ORDER BY kolumna LIMIT liczba;
```

PRZYKŁAD 19.10

Polecenie:

```
SELECT tytul, cena FROM ksiazka ORDER BY cena DESC LIMIT 1;
```

wyświetli tytuł oraz cenę najdroższej książki istniejącej w bazie. Przed klauzulą **LIMIT** wstawiono klauzulę **ORDER BY**, której zadaniem jest posortowanie listy malejąco według ceny. Dopiero po tym wybiera się pierwszy rekord wyświetlanej tabeli.

```
SELECT tytul, cena FROM ksiazka ORDER BY cena DESC LIMIT 1
```

+ Opcje

tytul	cena
PHP, MySQL i JavaScript	99

Rys. 19.11. Zastosowanie klauzuli **LIMIT**

Aby wyświetlić tytuły trzech najdroższych książek w bazie, wystarczy po słowie **LIMIT** umieścić liczbę 3.

```
SELECT tytul, cena FROM ksiazka ORDER BY cena DESC LIMIT 3
```

☐ Profilowanie

+ Opcje

tytul	cena ▼ 1
PHP, MySQL i JavaScript	99
HTML5. Tworzenie witryn	53.65
Windows 8 PL. Zaawansowana administracja systemem	49.99

Rys. 19.12. Efekt zastosowania klauzuli **LIMIT 3**

Operator BETWEEN...AND

Operator **BETWEEN...AND** służy do określenia w kryterium poszukiwania wartości granicznych danego pola. Zamiast stosować operatory porównania **<** lub **>** do określenia przedziału, łatwiej jest stosować operator **BETWEEN...AND**.

PRZYKŁAD 19.11

Zapytanie:

```
SELECT * FROM ksiazka WHERE cena BETWEEN 25 AND 40;
```

wyświetli dane o książkach istniejących w bazie, których cena mieści się w przedziale <25,40>.

Rys. 19.13. Zastosowanie operatora **BETWEEN...AND**

Zapytanie można zapisać za pomocą operatora porównania, dzięki czemu otrzyma się ten sam wynik polecenia:

```
SELECT * FROM ksiazka WHERE cena>=25 AND cena<=40;
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Do czego służy symbol `%`?
2. Jaka jest funkcja klauzuli **DISTINCT**?
3. Do czego służy klauzula **LIMIT**?
4. W jaki sposób zaokrąglić wynik w wyświetlanej tabeli?
5. Opisz zastosowanie funkcji **CONCAT**.
6. Jaką funkcję pełni klauzula **WHERE**?
7. Omów podstawową składnię polecenia **SELECT**.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Skorzystaj z bazy **SZKOLA** utworzonej w rozdziale 18 i wykonaj następujące polecenia:
   - wyświetl listę uczniów alfabetycznie według nazwisk;
   - wyświetl listę uczniów, których nazwiska rozpoczynają się na literę N, a imię kończy na literę A;
   - wyświetl imiona i nazwiska dziewcząt, których nazwiska rozpoczynają się na literę B.

20 Funkcje agregujące

ZAGADNIENIA

- Funkcja `COUNT`
- Funkcja `SUM`
- Funkcja `AVG`
- Funkcje `MIN` oraz `MAX`

Funkcje agregujące działają na wartościach wybranego pola (kolumny) w grupie wierszy. Zwracają one pojedyncze wartości. Stosuje się je w instrukcji **`SELECT`**.

Funkcje agregujące są jednym z najważniejszych narzędzi relacyjnych baz danych. Podstawowe funkcje agregujące to:

- **`COUNT`** – zwraca liczbę wierszy w grupie;
- **`SUM`** – zwraca sumę wartości w grupie dla wybranej kolumny;
- **`AVG`** – zwraca średnią wartość w grupie dla wybranej kolumny;
- **`MAX`** – zwraca największą wartość w grupie dla wybranej kolumny;
- **`MIN`** – zwraca najmniejszą wartość w grupie dla wybranej kolumny.

Funkcja `COUNT`

Postać funkcji:

```
COUNT(nazwa_kolumny)
```

Aby wyświetlić liczbę wierszy zwróconych przez zapytanie, najprościej użyć funkcji w postaci **`COUNT(*)`** (gwiazdka – wszystkie kolumny tabeli). W ten sposób:

- umożliwia się optymalizatorowi bazy danych wybranie kolumny do wykonania obliczeń, co czasem nieznacznie podnosi wydajność zapytania;
- nie trzeba się martwić o wartości **`NULL`** zawarte w kolumnie oraz o to, czy kolumna o podanej nazwie w ogóle istnieje.

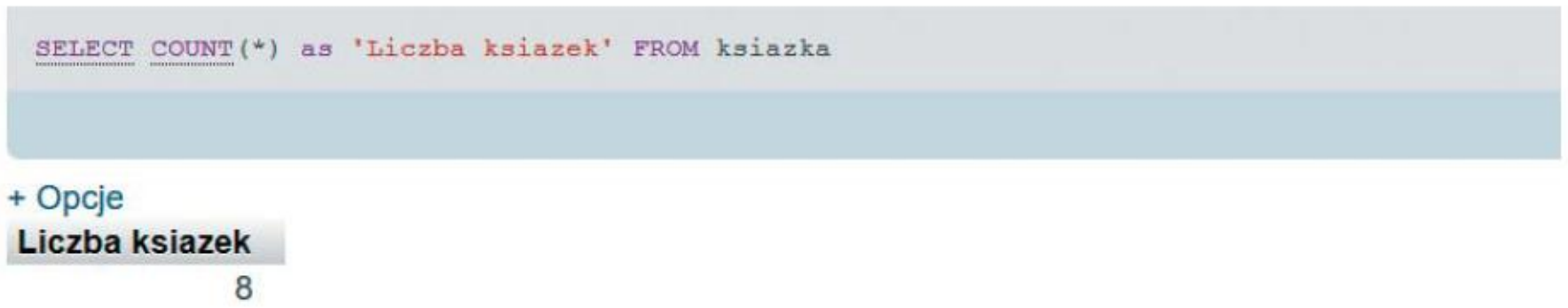
UWAGA

Funkcja w postaci **`COUNT(nazwa_kolumny)`** nie uwzględnia pól z wartościami **`NULL`**. Użycie funkcji w postaci **`COUNT(*)`** zlicza wszystkie wiersze bez względu na ich zawartość.

PRZYKŁAD 20.1

```
SELECT COUNT(*) as 'Liczba ksiazek' FROM ksiazka;
```

Polecenie wyświetli liczbę wierszy w tabeli **ksiazka**:



Rys. 20.1. Zastosowanie funkcji **COUNT**

Funkcja SUM

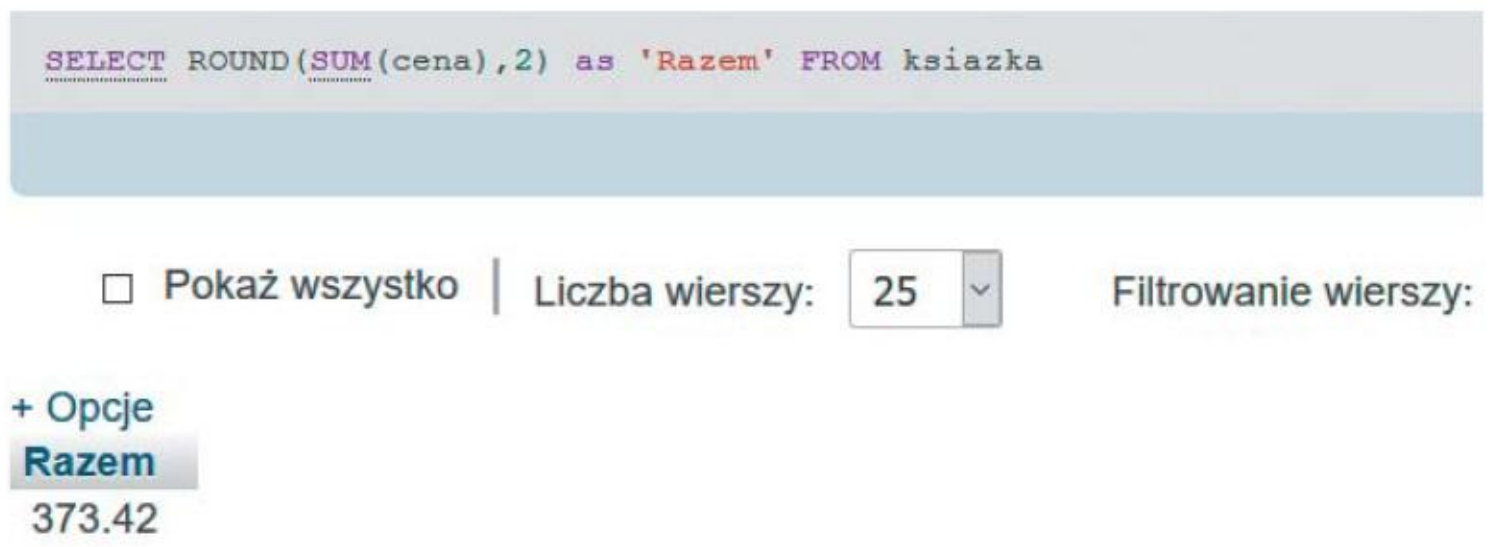
Postać funkcji:

```
SUM(nazwa_kolumny)
```

W przeciwieństwie do funkcji **COUNT()**, która działa dla wszystkich typów danych, funkcja **SUM()** działa tylko dla argumentów liczbowych – dodaje wszystkie wartości z wybranej kolumny i zwraca pojedynczy wynik. W działaniu tej funkcji wartości **NULL** są ignorowane (nie są uwzględniane w obliczeniach). Ta funkcja może być zastosowana w połączeniu z operatorem **DISTINCT**, który spowoduje, że obliczenia będą przeprowadzane tylko dla wartości unikalnych (niepowtarzających się).

PRZYKŁAD 20.2

```
SELECT ROUND(SUM(cena),2) as 'Razem' FROM ksiazka;
```



Rys. 20.2. Zastosowanie funkcji **SUM**

Oprócz funkcji **SUM** zastosowano funkcję **ROUND**, która zaokrągla wynik do dwóch miejsc po przecinku.

Funkcja AVG

Postać funkcji:

```
AVG(nazwa_kolumny)
```

Ta funkcja działa tylko na argumentach liczbowych, wylicza wartość średnią dla danej kolumny. Także w przypadku tej funkcji wartości **NULL** są ignorowane. Może być również zastosowana w połączeniu z operatorem **DISTINCT**.

PRZYKŁAD 20.3

```
SELECT ROUND(AVG(cena),2) as 'Średnia' FROM ksiazka;
```

```
SELECT ROUND(AVG(cena),2) as 'Średnia' FROM ksiazka
```

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

Średnia
46.68

Rys. 20.3. Zastosowanie funkcji **AVG**

W zapytaniach z użyciem funkcji agregujących można także zastosować klauzulę **WHERE**, aby zawęzić wartości liczbowe, na których mają zostać wykonane obliczenia.

PRZYKŁAD 20.4

```
SELECT ROUND(AVG(cena),2) as 'Średnia - PHP' FROM ksiazka
WHERE tytul LIKE "PHP%";
```

```
SELECT ROUND(AVG(cena),2) as 'Średnia - PHP' FROM ksiazka WHERE tytul LIKE "PHP%"
```

☐ Profilowanie

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

Średnia - PHP
64.49

Rys. 20.4. Zastosowanie funkcji AVG z klauzulą **WHERE**

Funkcje MIN oraz MAX

Postać funkcji:

MAX(nazwa_kolumny) lub **MIN(nazwa_kolumny)**

Służą one do znajdowania wartości najmniejszej i największej w zbiorze wartości. Obie funkcje mogą być użyte dla rożnych typów danych.

Funkcja **MAX()** znajduje największy łańcuch danych (zgodnie z regułami porównywania łańcuchów), najnowszą datę (lub najodleglejszą w przyszłości) oraz największą liczbę w zestawieniu. Funkcja **MIN()** znajduje analogiczne wartości najmniejsze.

Wartość **NULL** jest traktowana jako nieokreślona i nie można jej porównywać z innymi (te wartości są ignorowane).

Zarówno funkcja **MAX**, jak i funkcja **MIN** mogą być stosowane z operatorem **DISTINCT**, ale nie ma to większego znaczenia, gdyż i tak zwracają tylko jedną wartość z zestawienia.

PRZYKŁAD 20.5

Najwyższa cena książki zostanie wyświetlona za pomocą polecenia:

```
SELECT MAX(cena)as "Najwyższa cena" FROM ksiazka;
```

Otrzymamy następujący rezultat:

Rys. 20.5. Zastosowanie funkcji **MAX**

PRZYKŁAD 20.6

Aby wyświetlić datę najstarszego zamówienia, należy użyć polecenia:

```
SELECT MIN(data) as "Najstarsze zamówienie" FROM zamowienie;
```

Wynik będzie następujący:

Rys. 20.6. Zastosowanie funkcji **MIN**

SPRAWDŹ SWOJĄ WIEDZĘ

1. Do czego służy funkcja **AVG**? Podaj przykład jej zastosowania.
2. Opisz działanie funkcji **MIN** oraz **MAX**.
3. Jaka jest różnica między funkcją **COUNT** a funkcją **SUM**?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Skorzystaj z bazy **SZKOLA** utworzonej w rozdziale 17 i wykonaj następujące polecenia:
   - wyświetl liczbę uczniów zapisanych do bazy;
   - wyświetl ogólną średnią ocen uczniów;
   - wyświetl najniższą ocenę z chemii; w zapytaniu użyj pola **IdPrzedmiotu** zamiast **Nazwa_przedmiotu**.

21 Grupowanie danych

ZAGADNIENIA

- Klauzula `GROUP BY`
- Klauzula `HAVING`

Przed zastosowaniem funkcji agregującej jest czasami potrzebne podzielenie rekordów na grupy. Do jednej grupy należą te rekordy relacji, dla których tzw. wyrażenie grupujące zwraca tę samą wartość. Wyrażeniem grupującym jest najczęściej pojedynczy atrybut relacji. Po podziale rekordów na grupy w każdej z grup zostaje zastosowana funkcja agregująca, która wylicza pojedynczą wartość dla grupy. Z tego powodu w wyniku zapytania otrzymuje się tyle rekordów, ile grup utworzono w wyniku operacji grupowania. Do grupowania wierszy służy klauzula **`GROUP BY`**.

PRZYKŁAD 21.1

Aby policzyć liczbę egzemplarzy w tabeli **`ksiazka`** i pogrupować je według roku wydania, należy zastosować polecenie:

```
SELECT rokwydania, COUNT(*) as "Liczba egzemplarzy" FROM
ksiazka GROUP BY rokwydania;
```

Wynik będzie następujący:

```
SELECT rokwydania, COUNT(*) as "Liczba egzemplarzy" FROM ksiazka GROUP BY rokwydania
```

☐ Profilowanie

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

rokwydania	Liczba egzemplarzy
2000	4
2001	1
2002	1
2015	2

Rys. 21.1. Rezultat zastosowania klauzuli **`GROUP BY`**

PRZYKŁAD 21.2

Aby policzyć klientów mieszkających w tej samej miejscowości i posortować ich wyniki według liczby mieszkańców w danym mieście, należy użyć polecenia:

```
SELECT miejscowosc, COUNT(miejscowosc) AS "Liczba klientów"
FROM klient GROUP BY miejscowosc ORDER BY 2 DESC;
```

Efekt będzie następujący:

```
SELECT miejscowosc, COUNT(miejscowosc) AS "Liczba klientów" FROM klient GROUP BY miejscowosc ORDER BY 2 DESC
```

☐ Profilowanie [Edytuj w linii]

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

miejscowosc	Liczba klientów
Bydgoszcz	3
Kielce	2
Katowice	2
Lublin	2
Zgierz	2
Warszawa	2
Rybnik	1

Rys. 21.2. Zastosowanie klauzul **GROUP BY** i **ORDER BY**

Klauzula HAVING

Klauzula **HAVING** jest ściśle powiązana z klauzulą **GROUP BY**. Określa, które wiersze mają być zwrócone przez klauzulę **GROUP BY**. Jest to tzw. selekcja pozioma grup rekordów. Warunki, które wpisuje się po słowie **WHERE**, są traktowane jako selekcja pozioma pojedynczych wierszy.

PRZYKŁAD 21.3

```
SELECT rokwydania, COUNT(tytul) AS "Liczba egzemplarzy" FROM
ksiazka GROUP BY rokwydania HAVING rokwydania>=2010;
```

Polecenie wyświetli informację o liczbie egzemplarzy wydanych w danym roku, ale tylko od roku 2010 (rys. 21.3).

```
SELECT rokwydania, COUNT(tytul) AS "Liczba egzemplarzy" FROM ksiazka GROUP BY rokwydania HAVING rokwydania>=2010
```

☐ Profilowanie [Edytuj w linii]

+ Opcje

rokwydania	Liczba egzemplarzy
2015	2

Rys. 21.3. Zastosowanie klauzuli **HAVING**

SPRAWDŹ SWOJĄ WIEDZĘ

1. Do czego służy klauzula **`GROUP BY`**?
2. Opisz działanie klauzuli **`HAVING`**.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Skorzystaj z bazy **`SZKOLA`** utworzonej w rozdziale 17 i wykonaj następujące polecenia:
   - wyświetl liczbę chłopców i dziewcząt znajdujących się w bazie;
   - wyświetl średnią ocen z poszczególnych przedmiotów;
   - wyświetl liczbę uczniów w poszczególnych klasach.

22 Łączenie tabel

ZAGADNIENIA

- Połączenie wewnętrzne `INNER JOIN`
- Połączenie zewnętrzne `OUTER JOIN`
- Połączenie krzyżowe `CROSS JOIN`
- Połączenie tabeli z samą sobą `SELF JOIN`

W klauzuli **FROM** określa się przede wszystkim źródła (zbiory), z których zamierza się pobierać dane. W zapytaniu SQL można odwoływać się do jednego lub wielu zbiorów. Aby wybrać przynajmniej z dwóch, należy określić sposób ich połączenia oraz warunki.

W języku SQL połączenie między tabelami jest definiowane w sekcji **FROM**. Słowem kluczowym **INNER JOIN** określa się połączenie między tabelami. Polecenie realizujące połączenie wewnętrzne ma następującą postać:

```
tabela_1 INNER JOIN tabela_2
ON tabela1.kolumna1=tabela2.kolumna2;
```

Jedną z podstawowych i najczęściej wykorzystywanych cech języka SQL jest możliwość łączenia tabel.

Połączenie wewnętrzne – INNER JOIN

Jeżeli nie sprecyzowano tego inaczej, to domyślnie złączenie tabel w zapytaniu SQL jest traktowane jako złączenie wewnętrzne (**INNER JOIN** lub po prostu **JOIN**). Złączenie wewnętrzne kojarzy ze sobą rekordy występujące w dwóch lub więcej tabelach i zwraca w wyniku tylko pasujące rekordy. Złączenie wewnętrzne może być zdefiniowane w sposób jasno określony (wyraźny) lub domniemany. Złączenie domniemane jest bardziej intuicyjne i najczęściej spotykane.

Postać połączenia wewnętrznego:

```
SELECT * FROM tab1, tab2
WHERE tab1.ID = tab2.ID
```

lub

```
SELECT * FROM tab1
INNER JOIN tab2
ON tab1.ID = tab2.ID
```

PRZYKŁAD 22.1

Aby wyświetlić tytuł oraz imię i nazwisko autora książek, które zamówiono, należy napisać polecenie:

```
SELECT DISTINCT tytul,imieautora, nazwiskoautora FROM ksiazka,
zamowienie where zamowienie.Idksiazki=ksiazka.Idksiazki;
```

Trzeba pobrać informacje z dwóch tabel, dlatego też jest potrzebne połączenie ich za pomocą pola **Idksiazki**. Wynik będzie następujący:

```
SELECT DISTINCT tytul,imieautora, nazwiskoautora FROM ksiazka, zamowienie where zamowienie.Idksiazki=ksiazka.Idksiazki
```

☐ Profilowanie [Edytuj w linii] [Edytuj]

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

tytul	imieautora	nazwiskoautora
Urządzenia techniki komputerowej	Tomasz	Kowalski
Zaawansowane programowanie w PHP	Jan	Michalak
Windows 8 PL. Zaawansowana administracja systemem	Andrzej	Krawczyk
PHP. Tworzenie nowoczesnych stron WWW	Jan	Pasternak
HTML5. Tworzenie witryn	Piotr	Jakubowski

Rys. 22.1. Zastosowanie połączenia wewnętrznego

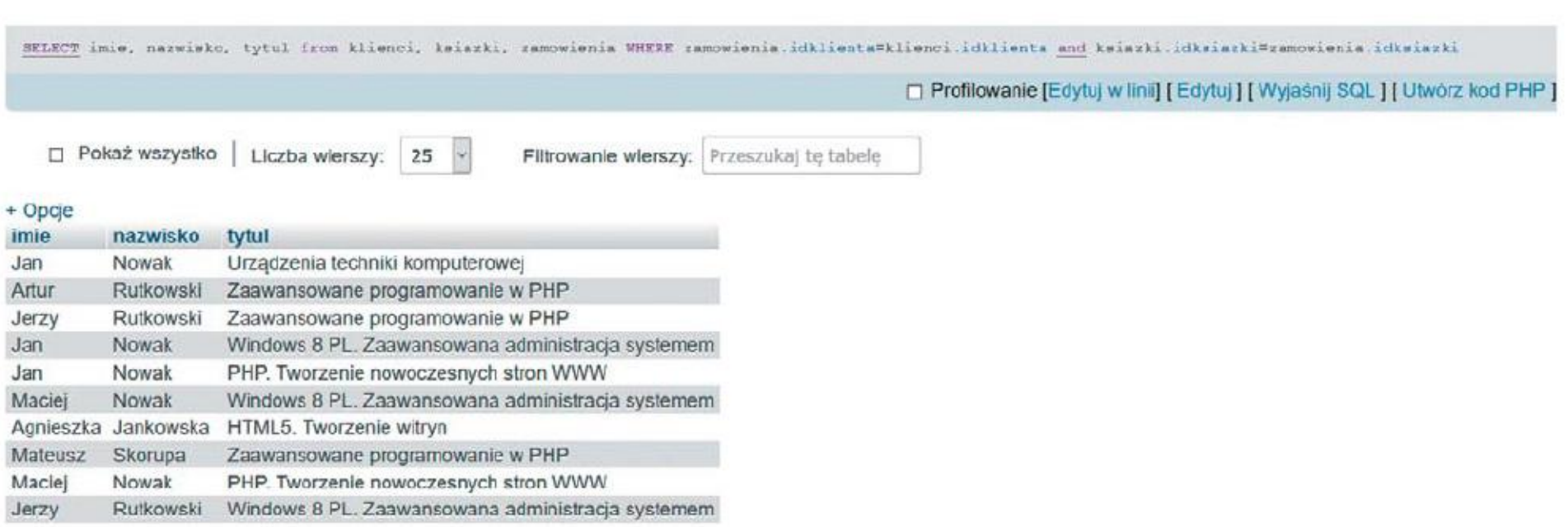
Ten sam wynik można otrzymać po uruchomieniu polecenia:

```
SELECT DISTINCT tytul, imieautora, nazwiskoautora FROM ksiazka
INNER JOIN zamowienie
ON zamowienie.Idksiazki=ksiazka.Idksiazki
```

PRZYKŁAD 22.2

Aby wyświetlić imiona i nazwiska klientów oraz tytuł książek, które zamówili, należy użyć polecenia:

```
SELECT imie, nazwisko, tytul from klient, ksiazka, zamowienie
WHERE zamowienie.Idklienta =klient.Idklienta and ksiazka.
Idksiazki=zamowienie.Idksiazki;
```



```
SELECT imie, nazwisko, tytul from klienci, ksiazki, zamowienia WHERE zamowienia.idklienta=klienci.idklienta and ksiazki.idksiazki=zamowienia.idksiazki
```

☐ Profilowanie [Edytuj w linii] [Edytuj] [Wyjaśnij SQL] [Utwórz kod PHP]

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

imie	nazwisko	tytul
Jan	Nowak	Urządzenia techniki komputerowej
Artur	Rutkowski	Zaawansowane programowanie w PHP
Jerzy	Rutkowski	Zaawansowane programowanie w PHP
Jan	Nowak	Windows 8 PL. Zaawansowana administracja systemem
Jan	Nowak	PHP. Tworzenie nowoczesnych stron WWW
Maciej	Nowak	Windows 8 PL. Zaawansowana administracja systemem
Agnieszka	Jankowska	HTML5. Tworzenie witryn
Mateusz	Skorupa	Zaawansowane programowanie w PHP
Maciej	Nowak	PHP. Tworzenie nowoczesnych stron WWW
Jerzy	Rutkowski	Windows 8 PL. Zaawansowana administracja systemem

Rys. 22.2. Połączenie trzech tabel w poleceniu **SELECT**

Ten sam wynik można otrzymać po wykonaniu polecenia:

```
SELECT imie, nazwisko, tytul from klient INNER JOIN zamowienie
ON zamowienie.Idklienta =klient.Idklienta INNER JOIN ksiazka on
ksiazka.Idksiazki=zamowienie.Idksiazki
```

Połączenie zewnętrzne – OUTER JOIN

Połączenie zewnętrzne tabel jest rozszerzeniem połączenia **INNER JOIN**. Natomiast **OUTER JOIN** pozwala na włączenie do tabeli wynikowej rekordów, które kwalifikują się do tabeli wynikowej na podstawie polecenia **INNER JOIN**, i dodatkowo umożliwia włączenie wybranych rekordów niespełniających warunków zapytania.

Połączenie zewnętrzne może przybrać następujące postaci:

- złączenie lewe – **LEFT OUTER JOIN**
  W złączeniu zewnętrznym lewym są zwracane wszystkie wiersze występujące w tabeli z lewej strony, a wiersze z prawej tabeli, które nie zostały znalezione, są wypełnione wartościami **NULL**. Postać polecenia jest następująca:

```
SELECT *
FROM Tab1
LEFT OUTER JOIN Tab2 ON Tab1.ID = Tab2.ID;
```

- złączenie prawe – **RIGHT OUTER JOIN**
  W złączeniu zewnętrznym prawym są zwracane wszystkie wiersze występujące w tabeli z prawej strony, a wiersze z lewej tabeli, które nie zostały znalezione, są wypełnione wartościami **NULL**. Postać polecenia jest następująca:

```
SELECT *
FROM Tab1
RIGHT OUTER JOIN Tab2 ON Tab1.ID = Tab2.ID;
```

- złączenie pełne – **FULL OUTER JOIN**
  W rezultatach zapytania SQL są obecne wiersze złączone wewnętrznie, rozszerzone o rekordy niezwrócone z obu tabel, wypełnione **NULL**.

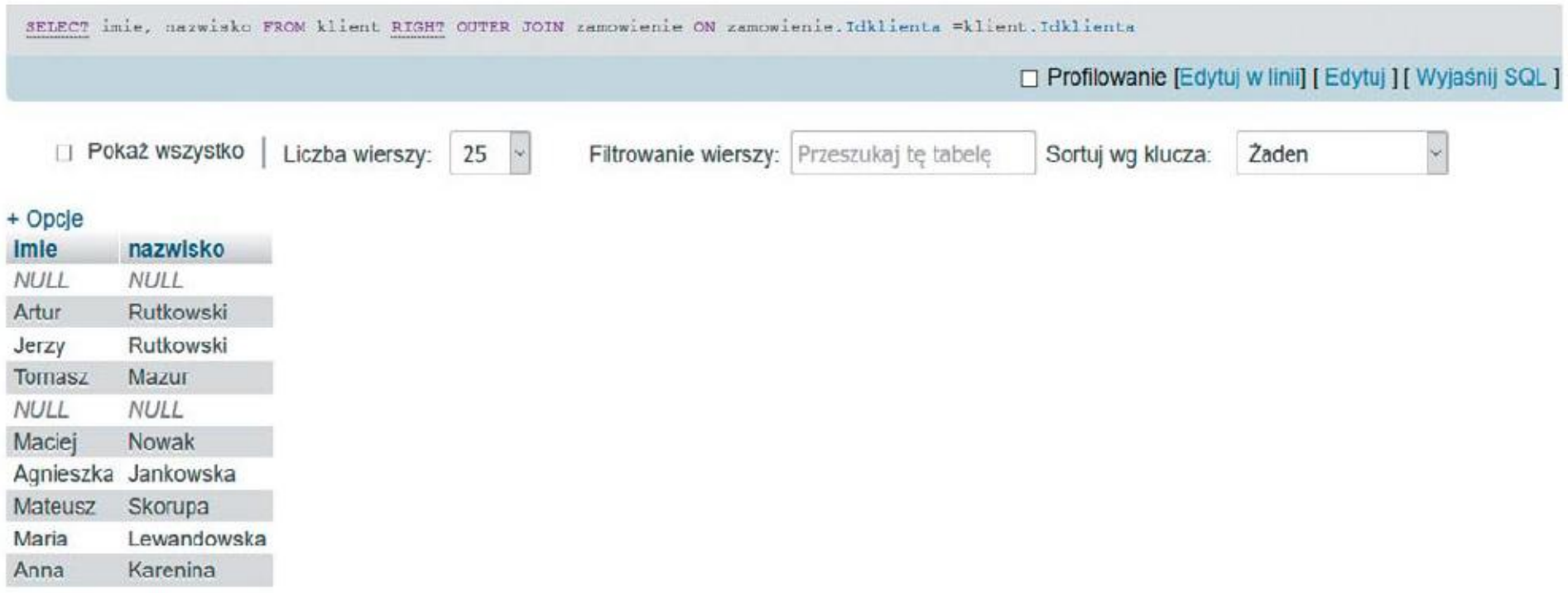
```
SELECT *
FROM Tab1
FULL OUTER JOIN Tab2 ON Tab1.ID = Tab2.ID;
```

PRZYKŁAD 22.3

Polecenie:

```
SELECT imie, nazwisko FROM klient RIGHT OUTER JOIN zamowienie
ON zamowienie.Idklienta =klient.Idklienta;
```

wyświetli rekordy z tabeli **zamówienia**, a rekordy z tabeli **klient**, które nie występują w tabeli zamówienia, zostaną wypełnione wartością **NULL** (rys. 22.3).



Rys. 22.3. Zastosowanie połączenia zewnętrznego

Połączenie krzyżowe – CROSS JOIN

Połączenie krzyżowe (iloczyn kartezjański) to połączenie wierszy dwóch tabel. Można nim połączyć dwie dowolne tabele. Tego typu połączenia są rzadko stosowane w relacyjnych bazach danych – łączą one każdy wiersz tabeli A z każdym wierszem tabeli B.

PRZYKŁAD 22.4

```
SELECT klient.nazwisko, klient.imie , ksiazka.tytul FROM
klient CROSS JOIN ksiazka;
```

Tabela **klient** zawiera 14 rekordów, tabela **ksiazka** – 8 rekordów. Polecenie spowoduje wyświetlenie 112 rekordów (jako iloczyn 14 · 8).

Połączenie tabeli z samą sobą – SELF JOIN

Definiuje się je podobnie jak połączenia różnych tabel. W razie połączenia z tą samą tabelą w klauzuli **FROM** trzeba podać dwie takie same nazwy. Aby je odróżnić, należy nadać tym tabelom aliasy.

SPRAWDŹ SWOJĄ WIEDZĘ

1. W jaki sposób można połączyć tabele w zapytaniach?
2. Jakie postaci może przybrać połączenie zewnętrzne?
3. Omów składnię połączenia wewnętrznego tabel w zapytaniach.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Skorzystaj z bazy **SZKOLA** utworzonej w rozdziale 17 i wykonaj następujące polecenia:
   - wyświetl imiona i nazwiska uczniów oraz ich oceny z geografii;
   - wyświetl nazwiska uczniów, którzy otrzymali z fizyki ocenę 4. Wyniki posortuj malejąco według nazwisk;
   - wyświetl informacje o uczniach, którzy mają z matematyki ocenę niższą niż 3. Informacje posortuj rosnąco według nazwisk.

23 Więzy integralności

ZAGADNIENIA

- Klucz obcy
- Kaskadowe usuwanie i aktualizowanie danych

Klucz obcy

Sprawdzenie spójności w bazie danych odbywa się za pomocą zdefiniowania klucza obcego. W naszej bazie w tabeli **zamówienia** pole **Idklienta** powinno być kluczem obcym pochodzącym z kolumny **Idklienta** w tabeli **klient**.

Klauzulę tworzącą klucz obcy należy umieścić za definicją kolumn w poleceniu **CREATE TABLE** lub **ALTER TABLE**.

PRZYKŁAD 23.1

Aby ustawić klucz obcy w utworzonej wcześniej tabeli **zamówienie**, należy zastosować polecenie:

```
ALTER TABLE zamowienie ADD CONSTRAINT Idklienta _FK FOREIGN
KEY (Idklienta ) REFERENCES klient(Idklienta );
```

Aby wstawić nowe pole **Idksiazka** (jako klucz obcy) do istniejącej już tabeli **zamówienie**, należy zapisać polecenie:

```
ALTER TABLE zamowienie ADD Idksiazka INT REFERENCES klient
(Idklienta );
```

Podczas tworzenia nowej tabeli można ustawić zarówno klucz podstawowy, jak i klucz obcy.

PRZYKŁAD 23.2

```
CREATE TABLE zamówienie
(
idzamowienie INT NOT NULL PRIMARY KEY;
data_zamowienie DATA;
Idksiazka INT REFERENCES książki (Idksiazka),
.....
);
```

Kaskadowe usuwanie i aktualizowanie danych

Kaskadowe usuwanie i aktualizowanie danych definiuje się w klauzulach **ON UPDATE** oraz **ON DELETE** z następującymi wartościami:

- **NO ACTION** – wartość domyślna, dane w powiązanych tabelach nie będą modyfikowane automatycznie;
- **CASCADE** – modyfikowanie będzie powtórzone automatycznie we wszystkich powiązanych tabelach;
- **SET NULL** – zmodyfikowane wartości klucza podstawowego będą zastąpione wartością **NULL** w powiązanych kolumnach klucza obcego;
- **SET DEFAULT** – zmodyfikowane wartości klucza podstawowego będą zastąpione wartością domyślną w powiązanych kolumnach klucza obcego.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Do czego służy klucz obcy?
2. Wyjaśnij, na czym polega kaskadowe usuwanie i aktualizowanie danych.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Która z tabel utworzonych w bazie **SZKOLA** powinna mieć klucz obcy? Do wykonania tego zadania zastosuj odpowiednie polecenie.

24 Łączenie zapytań

ZAGADNIENIA

- Instrukcja `UNION`
- Podzapytania klauzuli `WHERE` oraz `FROM`
- Operatory zapytań wewnętrznych

Instrukcja `UNION`

Do połączenia zapytań używa się instrukcji **`UNION`**, która służy do złączenia wyników zapytań. Ma ona następującą postać:

```
zapytanie1 UNION zapytanie2
```

UWAGA

Aby instrukcja została wykonana, w łączonych zapytaniach muszą być ta sama liczba i ten sam typ kolumn.

PRZYKŁAD 24.1

```
SELECT imie, nazwisko, miejscowosc FROM klient
WHERE miejscowosc="Bydgoszcz"
UNION
SELECT imie, nazwisko, miejscowosc FROM klient
WHERE miejscowosc="Katowice";
```

Pierwsze zapytanie wyświetli klientów z Bydgoszczy, drugie – z Katowic. Wynikiem połączenia tych dwóch zapytań jest lista klientów z Bydgoszczy i Katowic (rys. 24.1).

Rys. 24.1. Zastosowanie instrukcji **`UNION`**

Instrukcja **UNION** domyślnie powoduje usunięcie powtarzających się wierszy. Aby tego uniknąć, należy użyć instrukcji **UNION ALL**.

Instrukcja **UNION** lub **UNION ALL** może zastąpić operator **OR**, zatem polecenie z przykładu 24.1 może zostać zastąpione:

```
SELECT imie, nazwisko, miejscowosc from klient
where miejscowosc="Bydgoszcz" or miejscowosc="Katowice";
```

Podzapytania klauzuli WHERE

Podzapytania, jak sama nazwa wskazuje, są częścią podrzędną innego zapytania.

Jest to zapytanie umieszczone wewnątrz innego zapytania, tzw. zapytania zewnętrznego. Podzapytania najczęściej umieszcza się w warunkach w klauzulach **WHERE** i **HAVING** zapytania zewnętrznego. Niektóre SZBD dopuszczają również stosowanie podzapytań w klauzulach **SELECT** i **FROM**.

Ogólny schemat stosowania podzapytań jest następujący:

```
SELECT kolumna1, kolumna2,... FROM Tabela1
WHERE kolumna1=
(SELECT kolumnaA FROM Tabela2 WHERE warunek AS nazwa)
```

Zapytania wewnętrzne w klauzuli **WHERE** mogą zwracać pojedynczą wartość, listę wartości lub dane tabelaryczne.

PRZYKŁAD 24.2

Polecenie:

```
SELECT imie, nazwisko FROM klient where Idklienta =
(SELECT zamowienie.Idklienta FROM zamowienie
WHERE zamowienie.Idksiazka=3)
```

wyświetli imię i nazwisko klienta, który zamówił książkę o identyfikatorze równym 3. Wynik zapytania będzie następujący:

Rys. 24.2. Podzapytanie klauzuli **WHERE**

Wynikiem zapytania wewnętrznego jest **Idklienta**, który zamówił książkę o id=3. Zapytanie zewnętrzne natomiast działa tak, że zwraca imię i nazwisko klienta, którego **Idklienta** jest wynikiem zapytania wewnętrznego.

PRZYKŁAD 24.3

Aby wyświetlić tytuły książek, których cena jest większa od średniej ceny książek znajdujących się w bazie, oraz personalia ich autorów, należy napisać polecenie:

```
SELECT tytul, imieautora, nazwiskoautora, cena
FROM ksiazka
WHERE cena> (SELECT avg(cena) FROM ksiazka);
```

Otrzymamy następujący wynik:

tytul	imieautora	nazwiskoautora	cena
Zaawansowane programowanie w PHP	Jan	Michalak	47.29
Windows 8 PL. Zaawansowana administracja systemem	Andrzej	Krawczyk	49.99
HTML5. Tworzenie witryn	Piotr	Jakubowski	53.65
PHP, MySQL i JavaScript	Robin	Nixon	99

Rys. 24.3. Podzapytanie z funkcją agregującą

Podzapytania klauzuli FROM

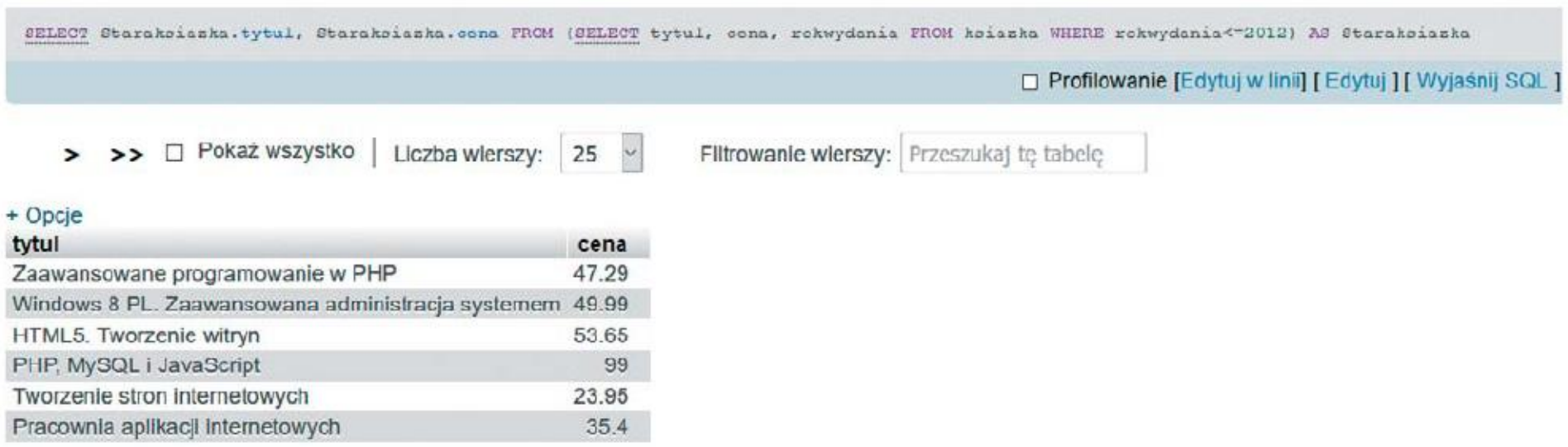
Wynikiem klauzuli **SELECT** jest najczęściej tabela zawierająca kolumny określone w zapytaniu. Dlatego też jest możliwe wykonanie dla niej kolejnego zapytania. Tabeli zwróconej przez podzapytanie użyte w klauzuli **FROM** należy przypisać nazwę.

Jeśli zapytanie wewnętrzne zwraca wartości w postaci tabeli, to tworzą się tzw. **tabele pochodne**. Są one dostępne tylko w zapytaniu, w którym je utworzono.

PRZYKŁAD 24.4

```
SELECT Staraksiazka.tytul, Staraksiazka.cena
FROM
(SELECT tytul, cena, rokwydania FROM ksiazka WHERE rokwyda-
nia<=2012) AS Staraksiazka;
```

Efektem zapytania będą następujące informacje:



tytul	cena
Zaawansowane programowanie w PHP	47.29
Windows 8 PL. Zaawansowana administracja systemem	49.99
HTML5. Tworzenie witryn	53.65
PHP, MySQL i JavaScript	99
Tworzenie stron internetowych	23.95
Pracownia aplikacji internetowych	35.4

Rys. 24.4. Podzapytanie klauzuli **FROM**

Operatory zapytań wewnętrznych

Podzapytania, które zwracają w zapytaniu wewnętrznym listę wartości, wymagają stosowania w klauzuli **WHERE** operatorów: **IN**, **EXISTS**, **ANY**, **SOME**, **ALL**.

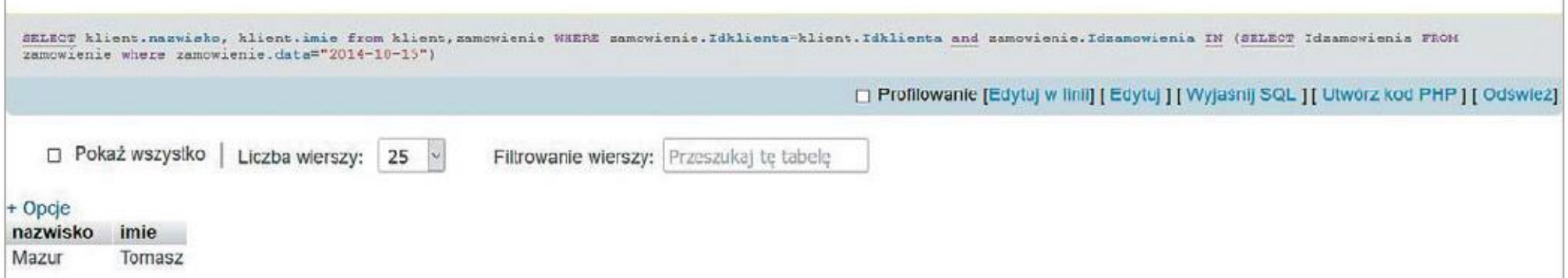
Operator **IN** służy do sprawdzenia, czy wartość poszukiwana w bazie należy do pewnego zbioru danych.

PRZYKŁAD 24.5

Przykład zapytania z zastosowaniem operatora **IN**, które wyświetli imię i nazwiska klienta składającego zamówienie 15 października 2014 r.

```
SELECT klient.nazwisko, klient.imie from klient,zamowienie
WHERE zamowienie.Idklienta=klient.Idklienta and zamowienie.
Idzamowienia IN (SELECT Idzamowienia FROM zamowienie where
zamowienie.data="2014-10-15");
```

Wynikiem zapytania będzie:



Rys. 24.5. Efekt zastosowania operatora **IN**

Operator **EXISTS** służy do sprawdzenia, czy w wyniku wewnętrznego zapytania zostały wyświetlone jakiekolwiek wartości. Zwraca wartość **TRUE** lub **FALSE**. Jeśli zapytanie wewnętrzne wyświetli jakiekolwiek wartości, to **EXISTS** zwróci prawdę. W przeciwnym przypadku zwróci fałsz.

PRZYKŁAD 24.6

```
SELECT klient.nazwisko, klient.imie from klient
WHERE EXISTS
(SELECT * FROM zamowienie where klient.Idklienta =zamowienie.
Idklienta );
```

Zapytanie wyświetli klientów, którzy złożyli zamówienia w księgarni. Otrzymamy następującą tabelę:

```
SELECT klient.nazwisko, klient.imie from klient WHERE EXISTS (SELECT * FROM zamowienie WHERE klient.IdKlienta =zamowienie.IdKlienta )
```

☐ Profilowanie [Edytuj w linii] [Edytuj] [Wyjaśnij SQL]

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę Sortuj wg klucza: Żaden

+ Opcje

nazwisko	imie
Nowak	Maciej
Jankowska	Agnieszka
Mazur	Tomasz
Rutkowski	Artur
Skorupa	Mateusz
Rutkowski	Jerzy
Karenina	Anna
Lewandowska	Maria

Rys. 24.6. Efekt zastosowania operatora **EXISTS**

Operatory **ANY** i **SOME** sprawdzają wartość dowolnego wiersza zwróconego przez podzapytanie. Operator **ANY** zwraca prawdę, jeżeli którakolwiek wartość będąca wynikiem zapytania wewnętrznego spełnia poprzedzające kryterium.

Operator **ALL** sprawdza wartość wszystkich wierszy wyniku zapytania. Zwraca wartość **PRAWDA**, jeśli wszystkie zwrócone wartości przez zapytanie wewnętrzne spełniają poprzedzający warunek.

PRZYKŁAD 24.7

Przykładem zastosowania operatora ANY jest zapytanie wyświetlające informacje o zamówieniach, których cena jest większa od ceny minimalnej i jednocześnie nie przekracza wartości 40.

```
SELECT * FROM zamowienie,ksiazka WHERE zamowienie.Idksiazki=k-
siazka.Idksiazki AND cena<40 AND cena> ANY
(SELECT MIN(cena) FROM ksiazka);
```

Wynikiem będą następujące informacje:

```
SELECT * FROM zamowienie,ksiazka WHERE zamowienie.Idksiazki=ksiazka.Idksiazki AND cena<40 AND cena> ANY (SELECT MIN(cena) FROM ksiazka)
```

☐ Profilowanie [Edytuj w linii] [Edytuj] [Wyjaśnij SQL] [Utwórz kod PHP] [Odśwież]

☐ Pokaż wszystko | Liczba wierszy: 25 Filtrowanie wierszy: Przeszukaj tę tabelę

+ Opcje

Idzamowienia	Idklienta	Idksiazki	data	status	Idksiazki	imieautora	nazwiskoautora	tytul	cena	rokwydania	wydawnictwo
1	2	4	2014-10-08	oczekiwanie	4	Tomasz	Kowalski	Urządzenia techniki komputerowej	34.15	2015	Komp2002
5	2	5	2014-08-12	oczekiwanie	5	Jan	Pasternak	PHP. Tworzenie nowoczesnych stron WWW	29.99	2015	EdukacjaPl
9	11	5	2014-11-19	wyslano	5	Jan	Pasternak	PHP. Tworzenie nowoczesnych stron WWW	29.99	2015	EdukacjaPl

Rys. 24.7. Efekt zastosowania operatora **ANY**

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień instrukcje służące do łączenia zapytań. Omów ich zastosowanie.
2. Do czego służą podzapytania?
3. Omów funkcje operatorów wewnętrznych.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Skorzystaj z bazy **SZKOLA** utworzonej w rozdziale 17 i wykonaj następujące polecenia:
   - wyświetl nazwiska chłopców urodzonych w Łodzi;
   - wyświetl imiona i nazwiska uczniów, którzy otrzymali z geografii ocenę 4 lub 5;
   - wyświetl imiona i nazwiska uczniów, którzy otrzymali z matematyki ocenę 4, ale nie zdali z geografii.

25 Użytkownicy i ich uprawnienia w bazie danych

ZAGADNIENIA

- Tworzenie i usuwanie użytkowników w bazie
- Nadawanie uprawnień użytkownikom bazy

Tworzenie i usuwanie użytkowników w bazie

Podczas instalacji i konfiguracji MySQL utworzono konto administratora – root. Ten użytkownik ma pełny dostęp do wykonywania działań związanych z zarządzaniem istniejących baz danych. Byłoby dobrze, aby pewne uprawnienia nie były dostępne dla wszystkich użytkowników – dzięki temu zmniejsza się ryzyko awarii.

Należy zatem stworzyć konta dla poszczególnych użytkowników oraz przydzielić im odpowiednie uprawnienia.

Aby utworzyć nowego użytkownika, który będzie się logował z komputera lokalnego, trzeba zastosować polecenie:

```
CREATE USER 'uzytkownik'@'localhost' [IDENTIFIED BY 'haslo'];
```

Parametr ujęty w nawias kwadratowy jest opcjonalny, czyli można utworzyć konto bez hasła.

PRZYKŁAD 25.1

```
CREATE USER 'mod'@'localhost' IDENTIFIED BY 'zaq1@WSX';
```

Utworzono hasło o nazwie **mod** zabezpieczone hasłem **zaq1@WSX**.
Hasła użytkowników są przechowywane w postaci zaszyfrowanej. Można je uzyskać za pomocą polecenia:

```
SELECT PASSWORD('haslo');
```

PRZYKŁAD 25.2

```
SELECT PASSWORD('zaq1@WSX');
```

Polecenie to wyświetli następującą informację:

Rys. 25.1. Otrzymanie zaszyfrowanego hasła

Aby dany użytkownik mógł logować się z dowolnego komputera w wybranej domenie, należy użyć symbolu `%`:

```
CREATE USER 'user'@'%.nazwa_domeny' IDENTIFIED BY 'haslo';
```

Jeśli w nazwie użytkownika bądź w nazwie domeny nie występują znaki specjalne – lub `%`, to w poleceniu można opuścić apostrofy.

Zastosowany symbol % jest oznaczeniem dowolnego komputera:

```
CREATE USER uzytkownik2@'%' IDENTIFIED BY 'haslo';
```

Nazwa hosta może być również podawana w postaci adresu IP:

```
CREATE USER 'user'@'213.186.88.112' IDENTIFIED BY 'haslo';
```

Do usunięcia użytkownika służy polecenie:

```
DROP USER nazwa_uzytkownika;
```

PRZYKŁAD 25.3

```
DROP USER 'mod'@'localhost';
```

Nadawanie uprawnień użytkownikom bazy

Aby nadać utworzonemu użytkownikowi uprawnienia do danej bazy danych, należy zastosować polecenie **GRANT** w następującej postaci:

```
GRANT lista_uprawnień ON obiekt TO użytkownik;
```

Obiektami mogą być pojedyncza tabela lub wszystkie tabele w wybranej bazie, a lista uprawnień przyjmuje wartości przedstawione w tabeli 25.1.

Tabela 25.1. Wartości parametru GRANT

Nazwa uprawnienia	Opis uprawnienia
`ALL [PRIVILEGES]`	nadanie wszystkich uprawnień oprócz `grant option`
`ALTER`	nadanie uprawnień do modyfikacji struktury tabeli
`CREATE`	nadanie uprawnień do tworzenia obiektów, czyli baz, tabel, kolumn
`DELETE`	nadanie uprawnień do usuwania wierszy w wybranej tabeli
`DROP`	nadanie uprawnień do usuwania tabel
`INSERT`	nadanie uprawnień do wstawiania wierszy do tabel
`SELECT`	nadanie uprawnień do zadawania zapytań
`SHOW DATABASES`	nadanie uprawnień do wyświetlania listy baz danych
`UPDATE`	nadanie uprawnień do modyfikacji zawartości tabel
`USAGE`	nadanie jedynie dostępu do bazy, bez żadnych dodatkowych uprawnień

PRZYKŁAD 25.4

```
GRANT SELECT ON samochody TO 'user1'@'localhost';
```

Polecenie nadało użytkownikowi **user1** uprawnienia do zadawania zapytań do tabeli **samochody**.
Aby temu samemu użytkownikowi nadać uprawnienia do zmiany struktury tabel oraz usuwania wierszy w tabelach dla wszystkich tabel w bazie **komis**, należy użyć polecenia:

```
GRANT ALTER, DELETE ON komis.* TO 'user1'@'localhost';
```

Skoro można nadać uprawnienia danemu użytkownikowi, można także je odebrać. Służy do tego polecenie **REVOKE** w następującej postaci:

```
REVOKE lista_uprawnień ON obiekt FROM użytkownik;
```

PRZYKŁAD 25.5

```
REVOKE DELETE ON komis.* FROM 'user1'@'localhost';
```

Polecenie poskutkuje odebraniem użytkownikowi **user1** uprawnień do usuwania wierszy w tabelach bazy **komis**.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Omów składnię polecenia służącego do tworzenia nowego użytkownika.
2. Jaki polecenie służy do nadawania uprawnień do bazy? Omów składnię tego polecenia.
3. W jakiej postaci przechowuje się hasła utworzonych użytkowników?
4. Czy można odebrać uprawnienia danemu użytkownikowi? Jeśli tak, to omów postać tego polecenia.
5. Wymień podstawowe właściwości polecenia **GRANT**.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Skorzystaj z bazy **SZKOLA** utworzonej w rozdziale 17 i wykonaj następujące polecenia:
   - utwórz użytkownika **Dyrektor** na localhost z hasłem **Dyrektor18!**;
   - nadaj użytkownikowi **Dyrektor** prawo do usuwania, dodawania oraz przeglądania danych tabeli **UCZEN**;
   - nadaj użytkownikowi **Dyrektor** wszystkie uprawnienia do tabeli **PRZEDMIOT**;
   - odbierz użytkownikowi **Dyrektor** uprawnienia do usuwania danych z tabeli **PRZEDMIOT**.

VI. Algorytmika

- Definicja algorytmów
- Rodzaje algorytmów i ich reprezentacja
- Stosowanie algorytmów do rozwiązywania problemów programistycznych

26 Podstawy stosowania algorytmów

ZAGADNIENIA

- Co to jest algorytm
- Reprezentacje algorytmów
- Klasyfikacja algorytmów

Zanim programista przystąpi do napisania kodu programu, tworzy algorytm, który pozwoli na czytelne określenie działania i funkcjonalności programu bądź systemu. Tworzenie algorytmu może przyjmować różną postać, ale najczęściej zaczyna się od dyskusji o danym problemie lub spisaniu na kartce papieru listy poszczególnych działań.

Algorytm opisuje sposób przekształcania danych wejściowych w dane wyjściowe zgodnie z wyznaczonym celem. Jest to po prostu przepis czy też instrukcja rozwiązania danego problemu. Na co dzień stosuje się wiele algorytmów (np. przepis na pieczenie ciasta czy instrukcja korzystania z pralki automatycznej). Wszystkie instrukcje obsługi urządzeń elektrycznych są *de facto* algorytmami.

Cechy algorytmu:

- poprawność – oznacza, że algorytm przynosi oczekiwane wyniki;
- jednoznaczność – polega na tym, że zawsze przy tych samych danych wejściowych otrzymuje się te same wyniki;
- skończoność – oznacza, że algorytm wykonuje się w skończonej liczbie kroków;
- efektywność – polega na tym, że rozwiązanie zadania jest następuje w jak najmniejszej liczbie kroków.

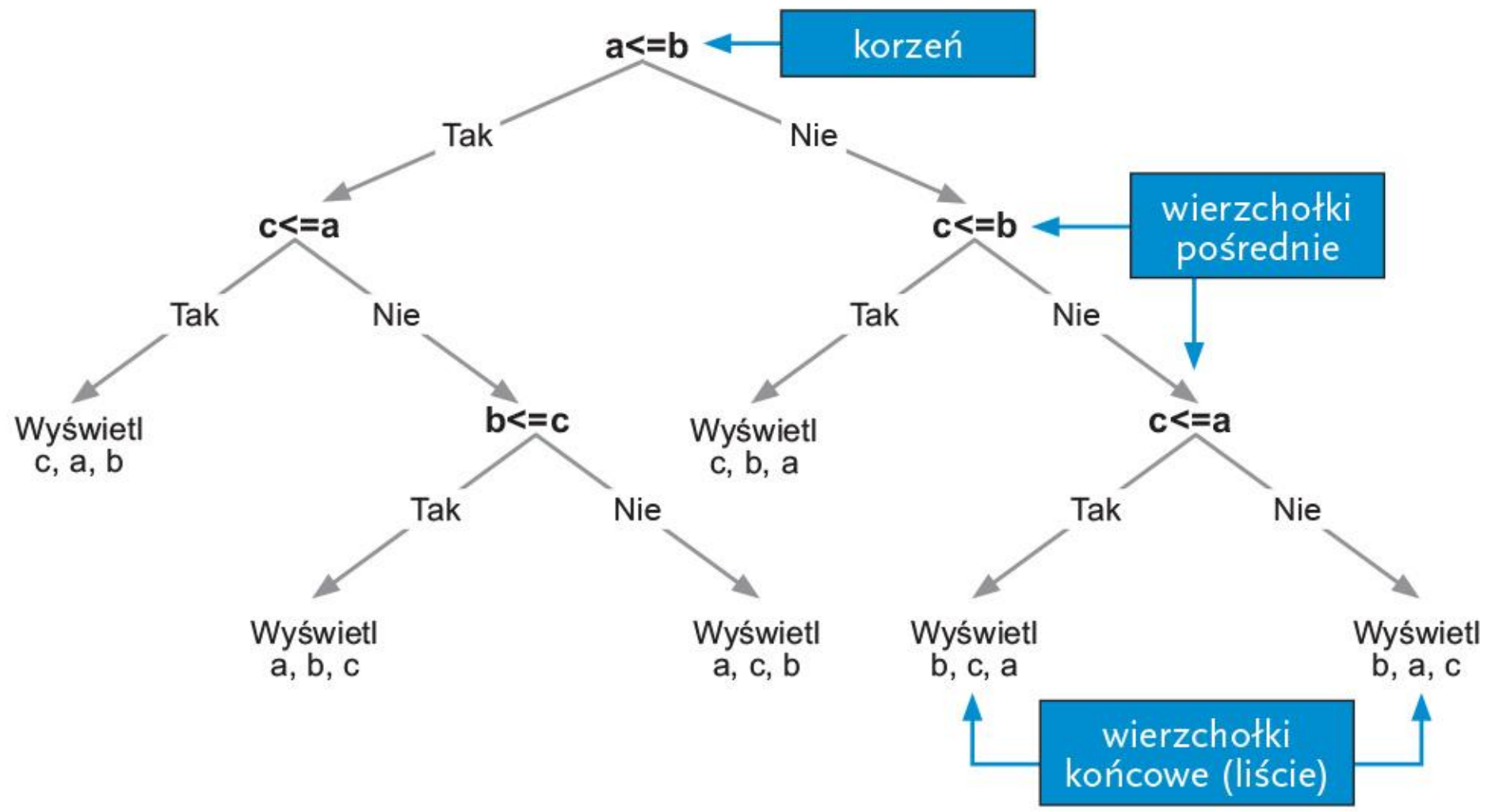
Zanim przejdziemy do utworzenia algorytmu, musimy zrealizować kolejne etapy. Na początku należy sformułować zadanie, czyli określić, jaki konkretny problem mamy do rozwiązania. Następnie trzeba określić dane wejściowe, które będą potrzebne do rozpoczęcia algorytmu. Określamy, jakiego typu danych będziemy potrzebować, czy będą to liczby całkowite, rzeczywiste, czy też może tablice znakowe. Kolejnym krokiem jest określenie wyniku i jego typu. Po poznaniu ogólnego zarysu problemu należy przystąpić do wybrania metody rozwiązania danego zadania. Jeśli okaże się, że jest kilka sposobów, trzeba wybrać optymalną metodę pod względem liczby wykonywanych instrukcji, szybkości działania czy oszczędności pamięci komputera. Po wybraniu sposobu rozwiązania zapisujemy algorytm. Następnie analizujemy poprawność jego działania, testujemy dla różnych danych wejściowych w celu sprawdzenia, czy nie generuje on błędów.

Istnieją następujące metody zapisu algorytmu:

- **słowny opis** – mało konkretny sposób opisu algorytmu; rozpoczyna się dyskusją, w jaki sposób można rozwiązać dane zadanie; pozwala na wybór metody rozwiązania danego problemu;
- **lista kroków** – dokładny sposób opisywania obliczeń i ich kolejności; poszczególne kroki zawierają opis operacji, które mają być wykonane przez algorytm; kroki są ponumerowane zgodnie z kolejnością ich wykonywania;
- **schemat blokowy** – najpopularniejszy sposób, który opiera się na blokach oraz połączeniach między nimi; są tu zapisane operacje, które mają być wykonane, a połączenia za pomocą strzałek wyznaczają kolejność ich wykonywania;
- **drzewo algorytmiczne** (drzewo obliczeń) – w drzewie algorytmu można wyróżnić korzeń – wierzchołek, w którym rozpoczynają się działania algorytmu, wierzchołki pośrednie, w których są umieszczane operacje wykonywane w algorytmie, oraz wierzchołki końcowe (tzw. liście), które odpowiadają różnym wynikom zakończenia obliczeń w algorytmie;

PRZYKŁAD 26.1

Rozważmy zadanie polegające na wyświetleniu w postaci ciągu rosnącego trzech liczb: a, b, c. Drzewo algorytmiczne tego zadania będzie miało postać jak na rys. 26.1.



Rys. 26.1. Przykład algorytmu w postaci drzewa algorytmicznego

- **pseudokod** – przy wykorzystaniu tego sposobu rezygnujemy z reguł składniowych danego języka programowania na rzecz czytelności instrukcji;
- **język algorytmiczny** (język programowania) – najbardziej ścisły i zrozumiały dla komputera opis algorytmu; algorytm zapisany w języku programowania nazywamy programem.

PRZYKŁAD 26.2

Weźmy pod uwagę zadanie mające na celu obliczenie wartości wyrażenia w = 2 * a – 4 * b. Lista kroków będzie miała następującą postać:

1. pobierz liczbę a;
2. pobierz liczbę b;
3. oblicz 2 * a – 4 * b, a obliczoną wartość przypisz do zmiennej w;
4. wyświetl wartość zmiennej w;
5. zakończ algorytm.

PRZYKŁAD 26.3

Weźmy pod uwagę zadanie z przykładu 26.2. Algorytm utworzony za pomocą pseudojęzyka może mieć postać:

1. start,
2. wprowadź liczby rzeczywiste a i b,
3. oblicz w = 2 * a – 4 * b,
4. wypisz w,
5. end.

Najczęściej wykorzystywanym sposobem przedstawiania algorytmów jest schemat blokowy, stosowany przede wszystkim przez programistów. Jest to graficzny zapis algorytmu rozwiązania danego zadania, który przedstawia opis i kolejność wykonywania czynności.

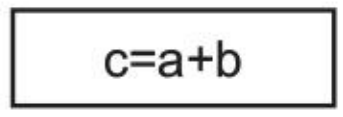
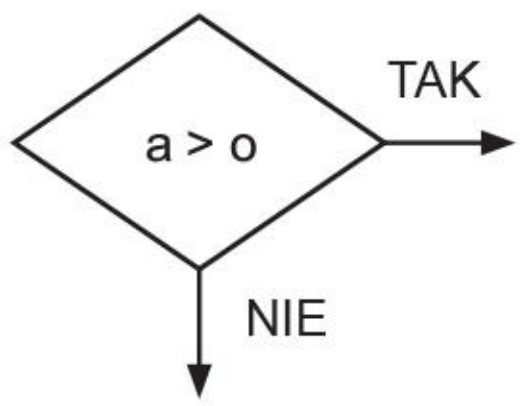
Schematy blokowe zwykle zapisuje się na kartce papieru. Można jednak skorzystać z edytora tekstu MS Word bądź MS Visio. Mają one wbudowane opcje do tworzenia figur schematu, ale nie są zbyt wygodne w użyciu.

Alternatywą dla nich są programy specjalnie przeznaczone do opisywania algorytmów. Jednym z nich są Magiczne Bloczki. Ten program umożliwia testowanie działania algorytmu. Ta funkcja jest szczególnie przydatna dla początkujących programistów, którzy mają problemy z wyszukiwaniem błędów w programie.

Zależnie od przedstawianego algorytmu stosuje się różne zestawy figur geometrycznych zwanych blokami, których kształty umownie reprezentują rodzaje elementów składowych. Bloki wykorzystywane do tworzenia schematów blokowych przedstawiono w tabeli 26.1.

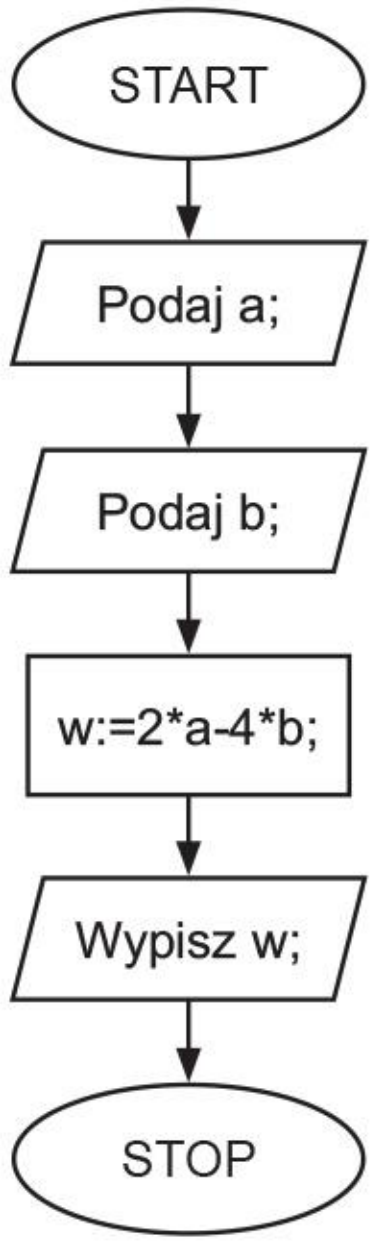
Tabela 26.1. Bloki wykorzystywane do tworzenia schematów blokowych

Schemat graficzny	Nazwa skrzynki (bloku)	Funkcja
START	skrzynka graficzna	początek algorytmu
STOP		koniec algorytmu

Schemat graficzny	Nazwa skrzynki (bloku)	Funkcja
	skrzynka operacyjna (blok operacyjny)	obliczanie wartości wyrażeń
	skrzynka wejścia (blok wejścia)	wprowadzanie danych
	skrzynka wyjścia (blok wyjścia)	wyprowadzanie danych
	skrzynka warunkowa (blok decyzyjny)	sprawdzanie warunku

PRZYKŁAD 26.4

Rozważmy zadanie z przykładu 26.2. Schemat blokowy tego algorytmu ma blok startowy, dwa bloki wejścia pozwalają na wprowadzenie danych wejściowych a i b. Dalej znajduje się blok operacyjny przypisujący zmiennej w wartość wyrażenia 2 * a –4 * b. Przedostatnim blokiem jest blok wyjścia wyświetlający wartość zmiennej w. Ostatni blok to skrzynka końca algorytmu. Bloki są połączone strzałkami określającymi kolejność wykonywanych instrukcji w algorytmie.



Rys. 26.2. Przykład schematu blokowego

Przed każdym algorytmem powinna zostać umieszczona specyfikacja problemu algorytmicznego, w którym opisujemy:

- problem algorytmiczny,
- dane wejściowe,
- dane wyjściowe.

Zarówno dane wejściowe, jak i wyjściowe są oznaczone za pomocą pojedynczych liter lub nazw.

PRZYKŁAD 26.5

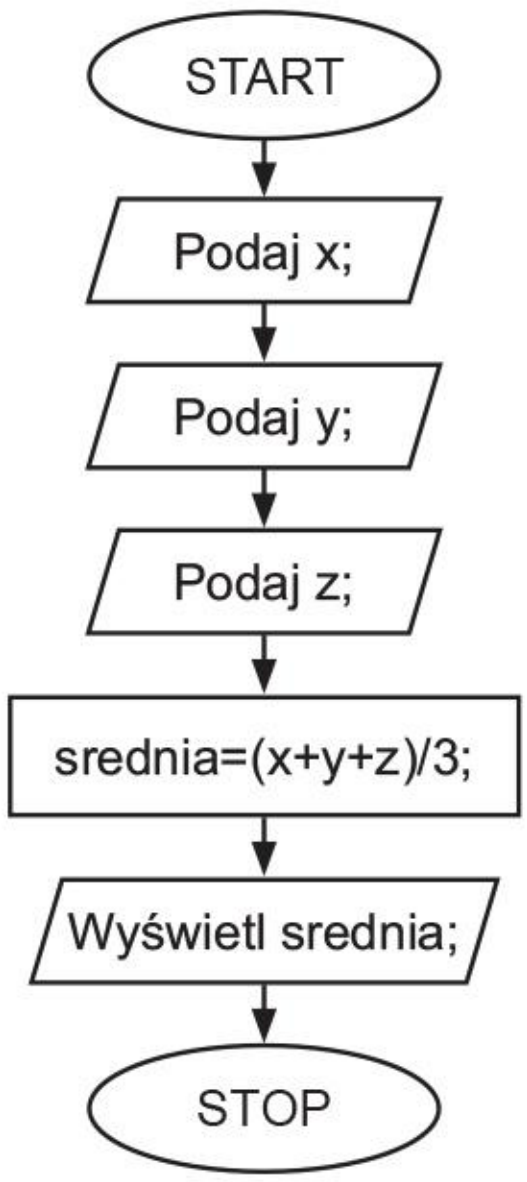
Specyfikacja problemu algorytmicznego z przykładu 26.3 ma postać:
Problem algorytmiczny: obliczenie wartości wyrażenia 2 * a – 4 * b
Dane wejściowe: a, b ∈ R
Dane wyjściowe: w ∈ R, w – wartość wyrażenia 2 * a – 4 * b

Klasyfikacja algorytmów

Algorytmy można podzielić ze względu na kolejność wykonywania działań. Należy tu wymienić algorytmy: sekwencyjne, z rozgałęzieniami, cykliczne i mieszane.

Algorytmy sekwencyjne, inaczej – proste lub liniowe, polegają na tym, że kolejne kroki algorytmu są zawsze wykonywane w tej samej kolejności, w jakiej je zapisano, niezależnie od danych wejściowych. Żaden krok nie może zostać pominięty ani powtórzony. Algorytmy tego rodzaju są zapisem obliczeń matematycznych, które będą wykonywane bez sprawdzania jakichkolwiek warunków.

PRZYKŁAD 26.6

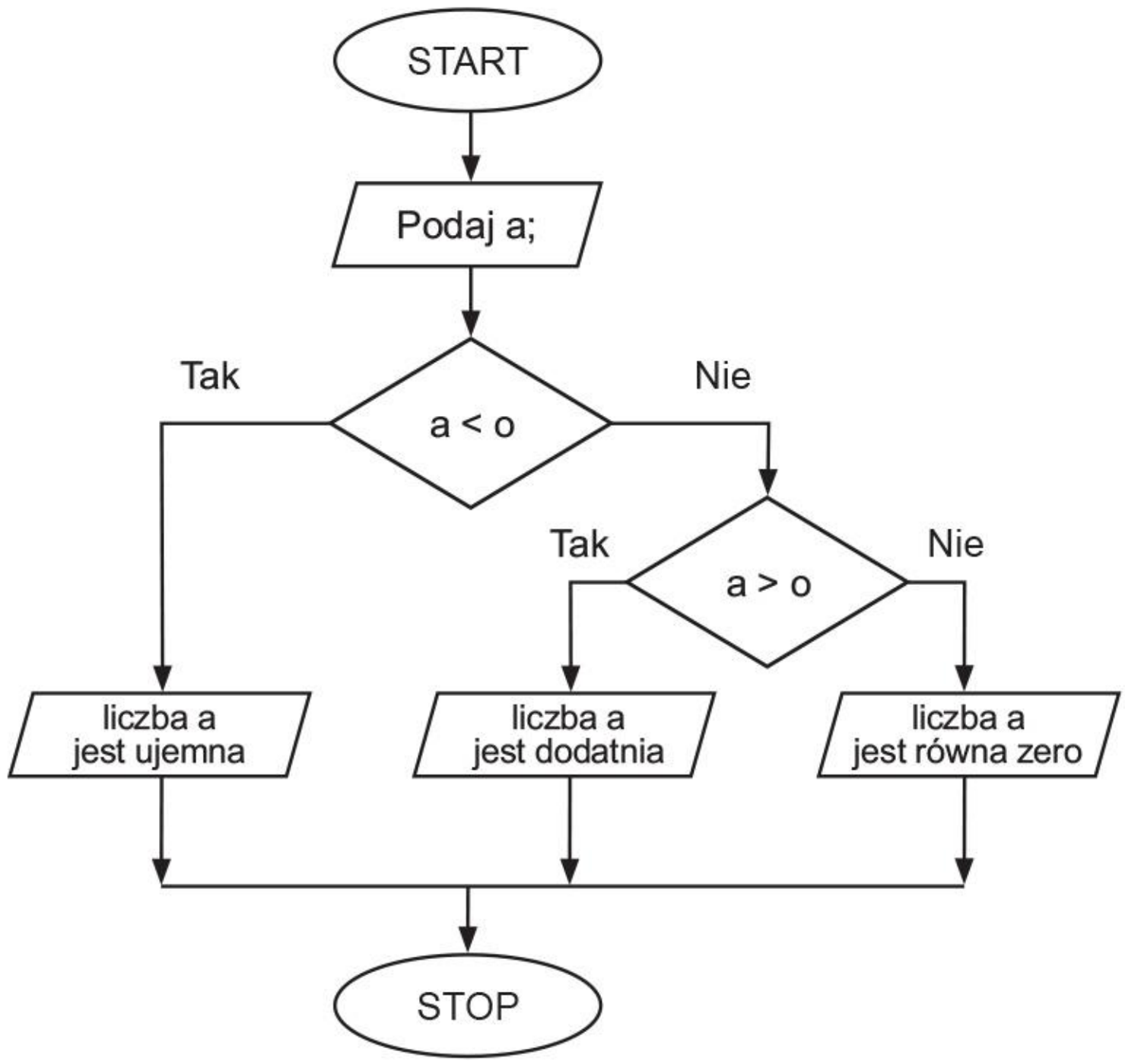


Rys. 26.3. Algorytm sekwencyjny

Algorytm oblicza średnią arytmetyczną trzech liczb: **x**, **y**, **z**. Wartość wyrażenia jest przypisana zmiennej **srednia**. Przed zakończeniem algorytmu wartość średniej jest wyświetlana na ekranie monitora.

Algorytmy z rozgałęzieniami charakteryzują się tym, że kolejność wykonywania niektórych kroków może się zmienić zależnie od wartości danych wejściowych lub wartości danych pomocniczych, ponieważ istnieje przynajmniej jeden blok warunkowy. Sprawdzanie warunków występuje w wielu wzorach matematycznych, np. podczas obliczania ilorazu dwóch liczb bądź obliczania pierwiastka kwadratowego podanej liczby. Aby zapewnić niezawodność algorytmu, należy uwzględnić bloki warunkowe umożliwiające zabezpieczenie przez zabronionymi działaniami.

PRZYKŁAD 26.7

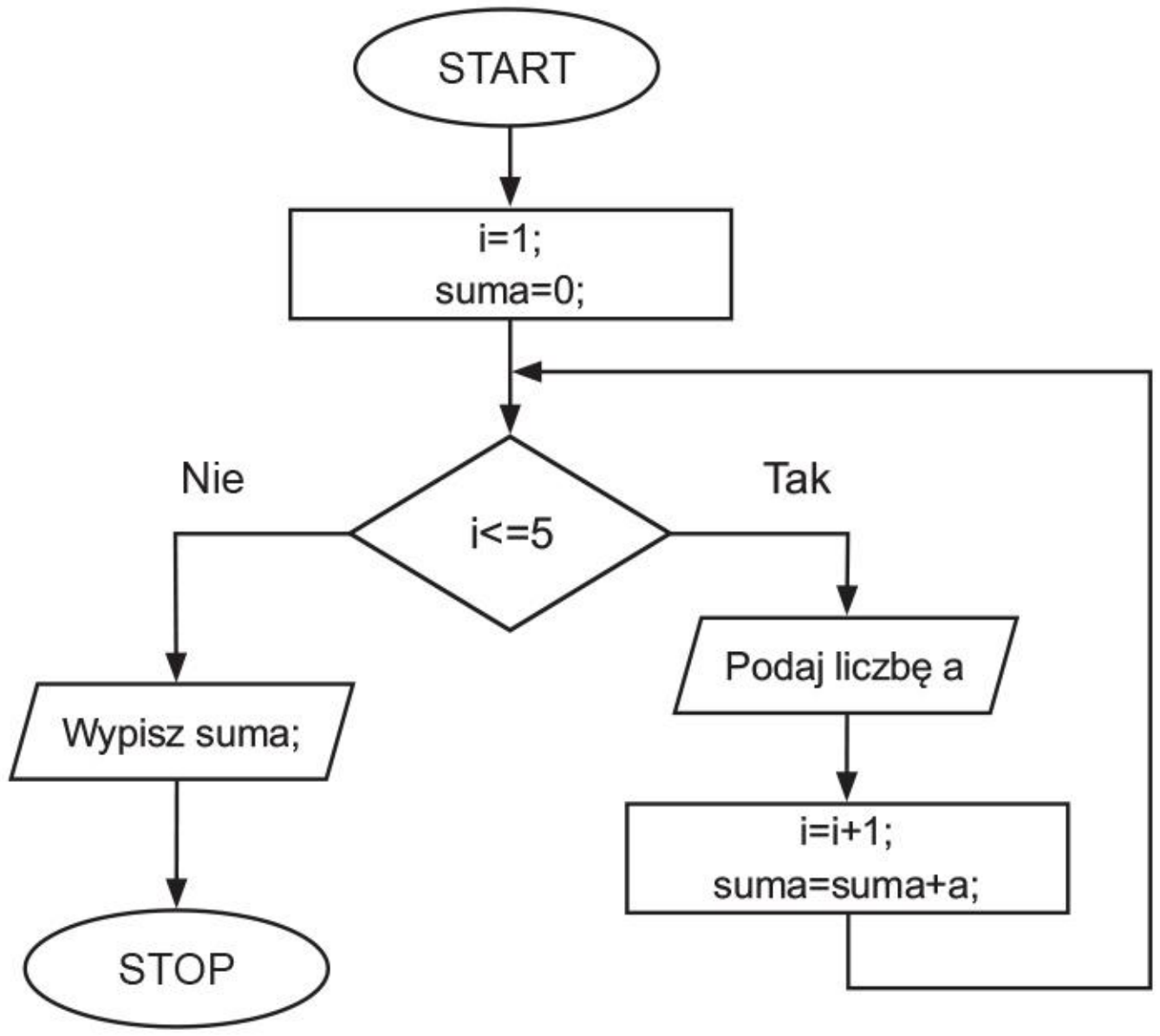


Rys. 26.4. Algorytmy z rozgałęzieniami

Algorytm z przykładu 26.6 pobiera wartość dla zmiennej a, następnie za pomocą pierwszego bloku warunkowego sprawdza się, czy podana wartość jest liczbą ujemną. Jeśli tak, to zostaje wypisany komunikat „Liczba a jest ujemna". W przeciwnym przypadku sprawdzamy, czy liczba a jest dodatnia. Jeśli warunek jest spełniony, to zostaje wyświetlany komunikat „Liczba a jest dodatnia". W przeciwnym razie otrzymujemy informację, że liczba a jest zerem.

Algorytmy cykliczne (**z pętlą**) to algorytmy, w których występuje powtarzanie określonych instrukcji przy zmienionych danych wejściowych. Pętla to część algorytmu, która jest powtarzalna.

PRZYKŁAD 26.8



Rys. 26.5. Algorytm z pętlą

Schemat blokowy z przykładu 26.4 oblicza sumę pięciu liczb podanych przez użytkownika. Instrukcjami powtarzanymi w pętli są prośba o podanie liczby oraz dodanie jej do poprzedniej sumy. Po wyjściu z pętli wynik sumy jest wyświetlany na ekranie monitora.

Algorytmy mieszane to kombinacja algorytmów prostych z rozgałęzieniami lub z pętlą.

Ze względu na sposób wykonywania operacji algorytmy dzieli się na: sekwencyjne, iteracyjne i rekurencyjne.

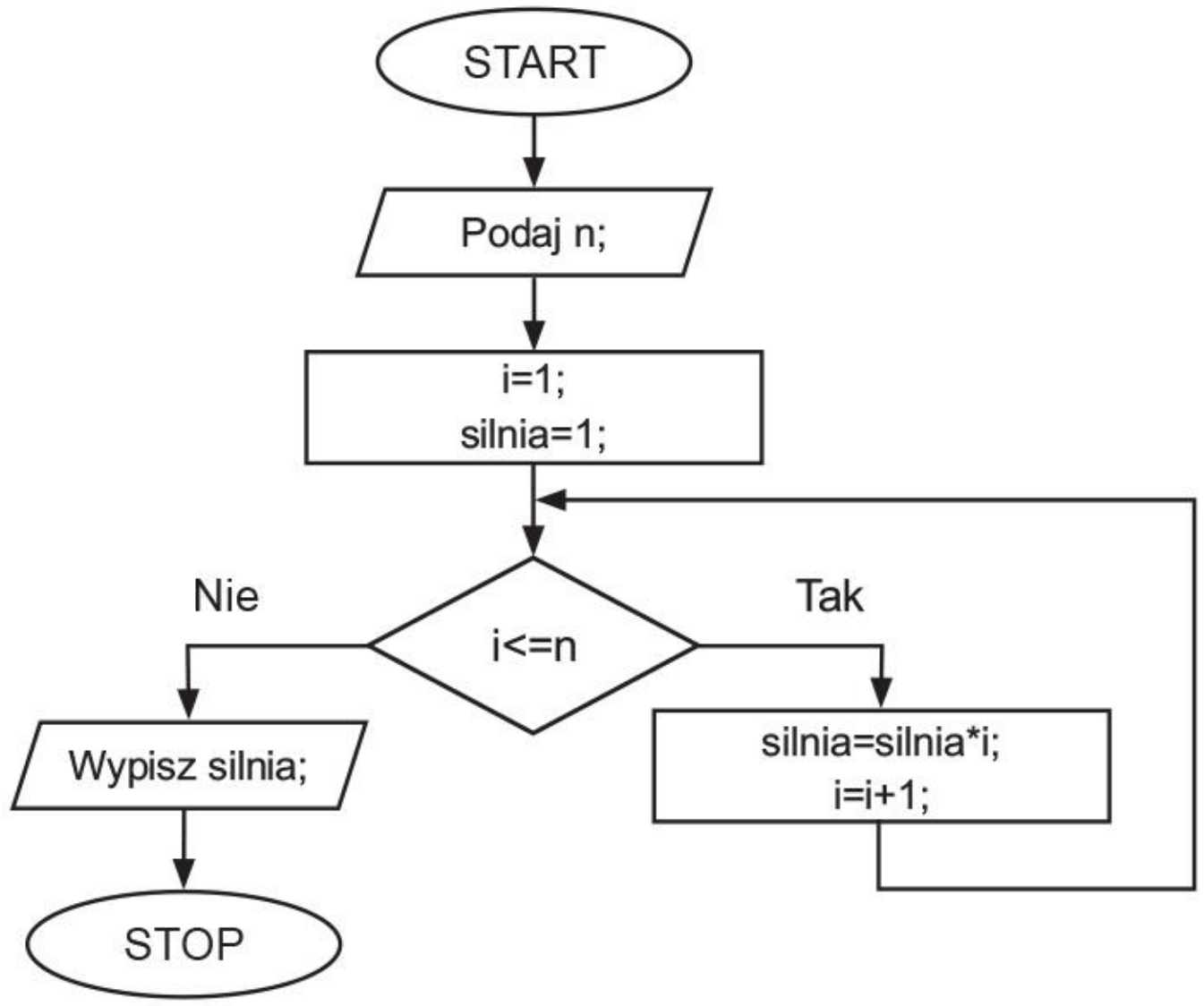
Algorytmy sekwencyjne zostały opisane podczas pierwszego podziału algorytmów ze względu na kolejność wykonywania działań, co znaczy, że instrukcje opisane w algorytmie są wykonywane w kolejności, w jakiej zostały wprowadzone.

Algorytmy iteracyjne charakteryzują się tym, że uzyskujemy wynik przez **iterację**, czyli powtarzanie danych instrukcji określoną liczbę razy lub do momentu spełnienia pewnego warunku.

Algorytmy rekurencyjne polegają na stosowaniu funkcji wywołujących same siebie. Rekurencja polega na tym, że rozwiązanie danego zadania wyraża się za pomocą tego samego zadania, ale dla danych o mniejszych wartościach. Niestety, należy pamiętać, że rekurencja zwiększa ilość potrzebnej pamięci na program.

PRZYKŁAD 26.9

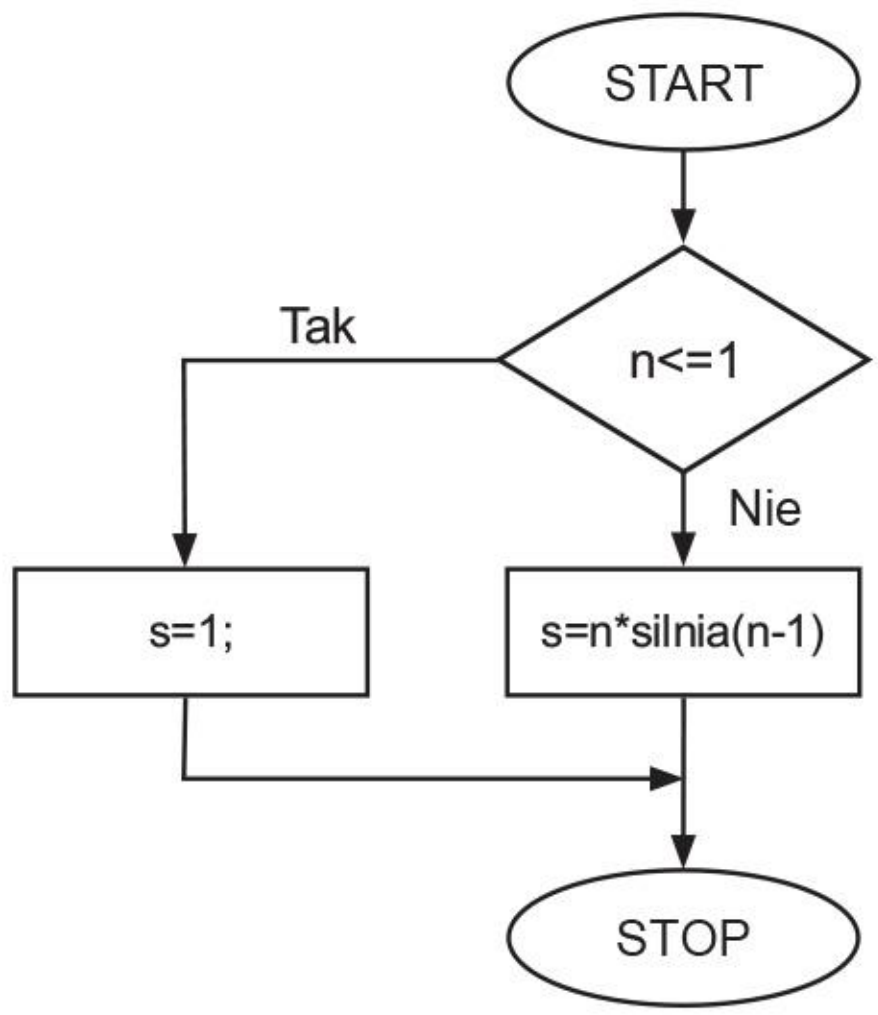
Przykładem algorytmu iteracyjnego jest obliczanie wartości silni liczby naturalnej n, czyli iloczynu kolejnych liczb naturalnych.



Rys. 26.6. Algorytm iteracyjny

PRZYKŁAD 26.10

Obliczenie silni dla zadanej liczby naturalnej n może odbywać się również za pomocą rekurencji.



Rys. 26.7. Algorytm rekurencyjny

Rekurencyjna definicja silni dla podanej liczby naturalnej n ma postać:

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n \geq 1 \end{cases}$$

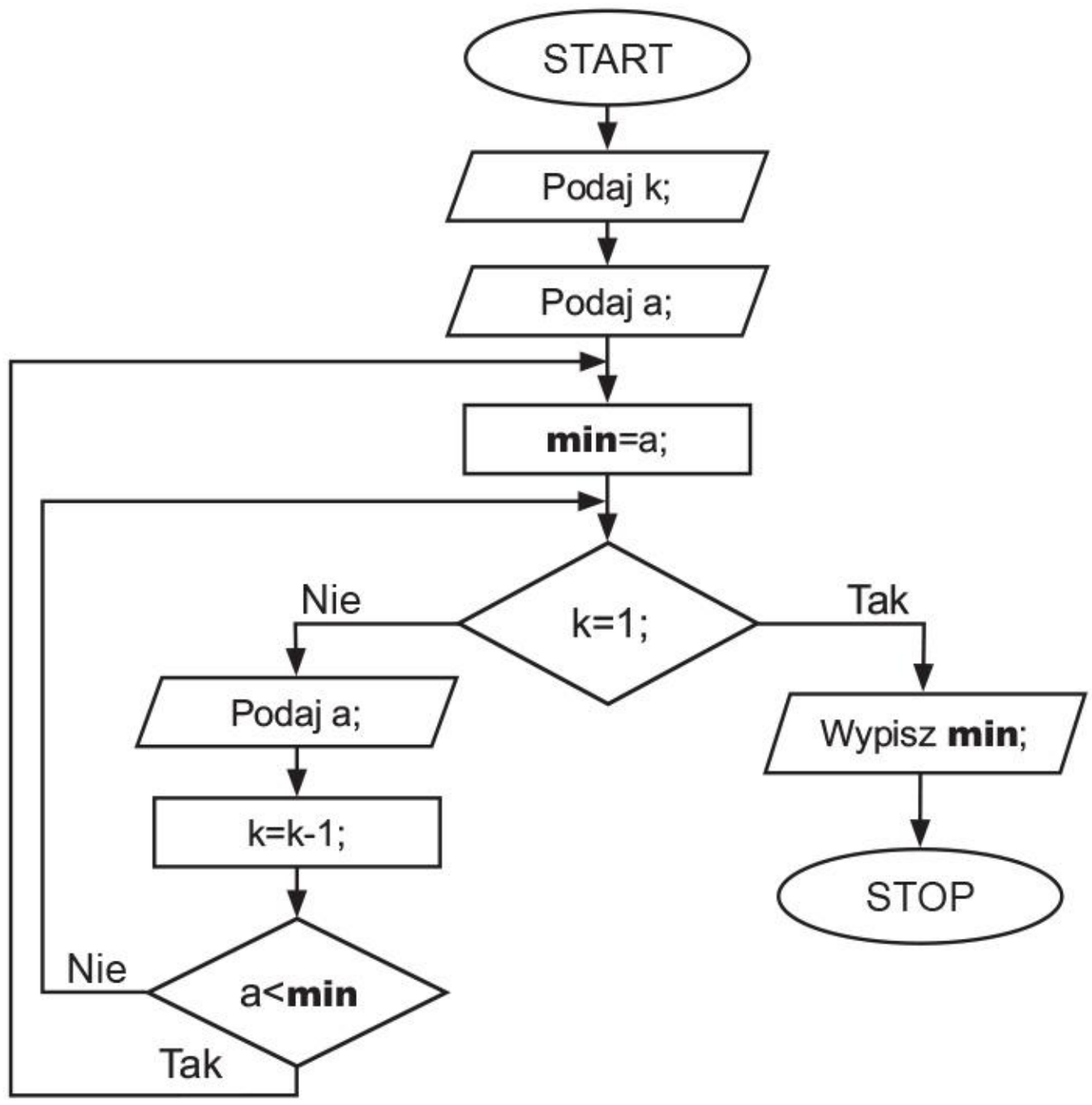
W przypadku podania n = 3 schemat działania jest następujący:

1. n = 3 i jest liczbą większą od 1 lub jej równą, zatem s = 3 · silnia(2) = 3 · 2!
2. n = 2 i jest liczbą większą od 1 lub jej równą, zatem s = 2 · silnia(1) = 3 · 2 · 1!
3. n = 1, zatem s = 3 · 2 · 1, co kończy algorytm.

Podział algorytmów ze względu na przeznaczenie:

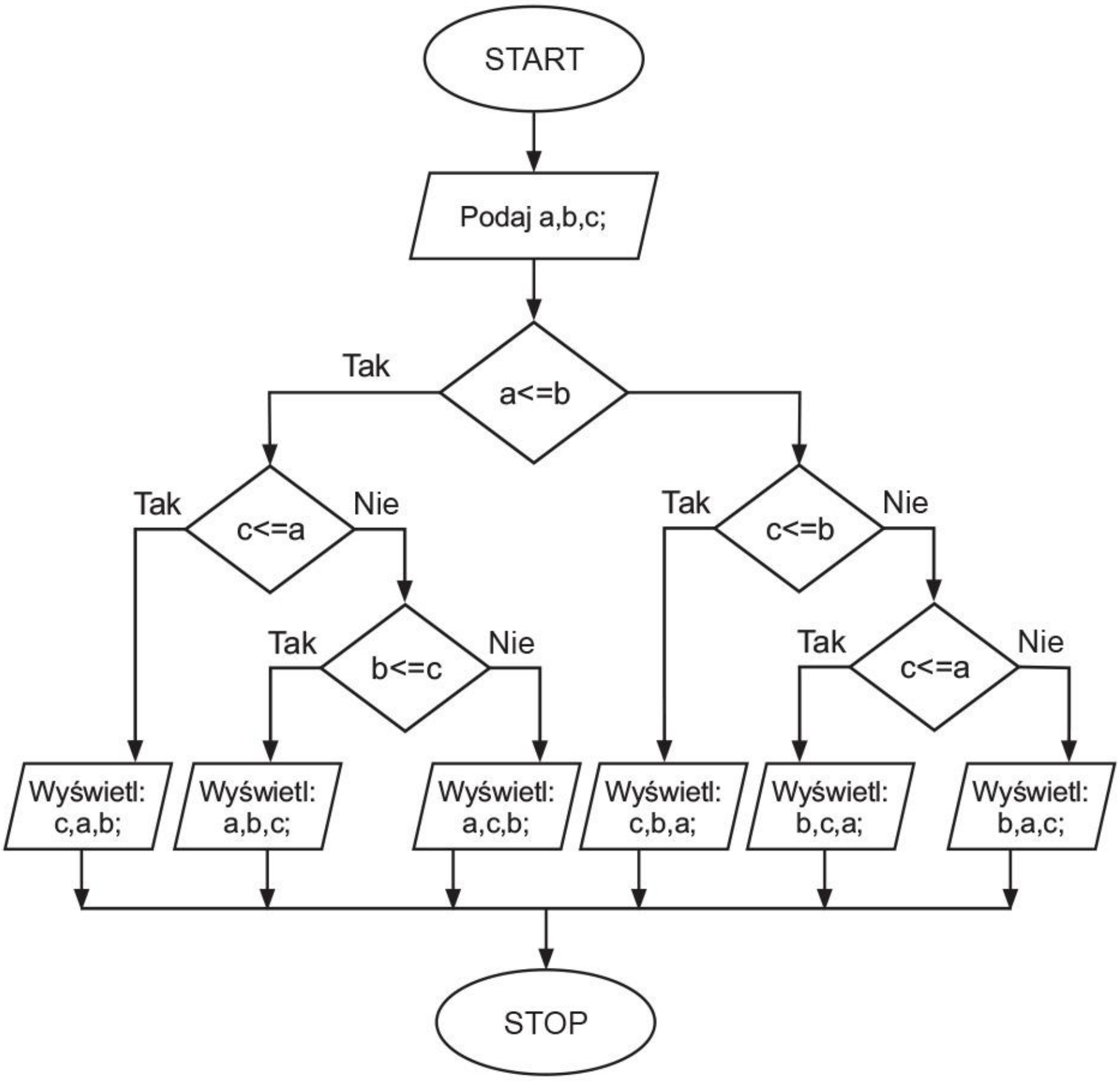
- algorytmy numeryczne (matematyczne) – wykonują obliczenia numeryczne;
- algorytmy przeszukujące – badają określony zbiór danych w celu wyszukania określonego elementu lub elementów;
- algorytmy porządkujące – ustawiają wybrane dane według określonego kryterium;
- algorytmy rekurencyjne – rozwiązują zadanie, które można podzielić na mniejsze części będące kopią wzorca;
- algorytmy szyfrujące / deszyfrujące – służą do szyfrowania danych;
- algorytmy kompresji danych – określają optymalny sposób kompresji zapisu danych.

PRZYKŁAD 26.11



Rys. 26.8. Algorytm wyszukujący najmniejszą liczbę spośród podanych

PRZYKŁAD 26.12



Rys. 26.9. Algorytm porządkujący trzy liczby w postaci ciągu rosnącego

Oprócz poprzednich kategorii algorytmy można również sklasyfikować ze względu na sposób ich konstruowania:

- **„dziel i zwyciężaj”** – problem jest dzielony na kilka mniejszych problemów, które znów są dzielone na mniejsze – aż do momentu, gdy otrzymamy problemy łatwe do rozwiązania; mniejsze problemy zostają rozwiązywane rekurencyjnie, a następnie ich rozwiązania są łączone, aby otrzymać rozwiązanie zadania początkowego; to jedna z najlepszych metod rozwiązywania zadań; przykładem może być tu znajdowanie najmniejszego elementu zbioru czy potęgowanie liczb;

PRZYKŁAD 26.13

Weźmy pod uwagę zbiór liczb: 2, 5, 4, 8, 16, 7. Załóżmy, że chcemy znaleźć element najmniejszy w tym zbiorze.
Metoda „dziel i zwyciężaj" polega tutaj na tym, że najpierw wyszukiwanie elementu następuje przez porównywanie parami, czyli: 2<5, 4<8, 16>7.
Z tych porównywanych par otrzymujemy pretendentów na minimum zbioru: 2, 4, 7.
Na tym podzbiorze znowu przeprowadzamy porównywanie parami i otrzymujemy: 2<4, 7.
Stąd otrzymujemy mniejszy podzbiór: 2, 7, na którym przeprowadzamy porównywanie, czy 2<7, i ostatecznie rozwiązaniem jest liczba 2.

- **programowanie dynamiczne** – metoda podobna do poprzedniej, czyli problem jest dzielony na kilka mniejszych, a ich rozwiązania wykorzystuje się do rozwiązania problemu wyjściowego; można je stosować, gdy podproblemy nie są niezależne, czyli kiedy podproblemy mogą zawierać te same podproblemy;
- **metoda zachłanna** – wykonuje takie działanie, które w danej chwili wydaje się najkorzystniejsze; ten rodzaj algorytmu nie ocenia, czy w kolejnych krokach jest sens wykonywać dane działanie, tylko podejmuje decyzje lokalnie optymalne;
- **poszukiwanie i wyliczanie** – zbiór danych jest przeszukiwany aż do znalezienia rozwiązania;
- **algorytmy heurystyczne** – algorytm, który nie zapewnia pewności rozwiązania zadania optymalnego, tylko daje dość dobre rozwiązanie w określonym czasie.

Gdy tworzymy algorytm, a potem na jego podstawie – program komputerowy, musimy pamiętać o oszczędności pamięci i czasu wykonywania obliczeń. W celu optymalizacji algorytmu wprowadzono pojęcie **złożoności obliczeniowej algorytmu**, która jest ilością czasu niezbędnego do rozwiązywania problemu w zależności od liczby danych wejściowych. To pojęcie dotyczy złożoności pamięciowej i czasowej.

Złożoność pamięciowa zależy od wielkości pamięci komputerowej potrzebnej do wykonania algorytmu w postaci programu komputerowego.

Złożoność czasowa pozwala na określenie czasu potrzebnego do wykonania algorytmu.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest algorytm?
2. Omów podział algorytmów ze względu na przeznaczenie.
3. Omów podział algorytmów ze względu na kolejność wykonywania działań.
4. Omów podział algorytmów ze względu na sposób wykonywania operacji.
5. Z czego składa się każdy algorytm?
6. Wymień cechy algorytmu.
7. Wymień i omów poszczególne bloki służące do tworzenia schematów blokowych.
8. Na czym polega zasada działania algorytmu „dziel i zwyciężaj"?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz drzewo algorytmiczne i schemat blokowy algorytmu rozwiązującego dowolne równanie kwadratowe.
2. Utwórz algorytm w postaci listy kroków i schematu blokowego wyznaczający n-tą potęgę liczby x (n i x są liczbami naturalnymi).
3. Utwórz schemat blokowy algorytmu wyznaczającego największy wspólny dzielnik dwóch liczb całkowitych.
4. Utwórz schemat blokowy algorytmu wypisującego liczby dwucyfrowe.
5. Utwórz drzewo algorytmiczne oraz listę kroków algorytmu pobierającego kolejne liczby jednocyfrowe, aż ich suma przekroczy 30.
6. Utwórz schemat blokowy algorytmu wypisujący n elementów ciągu Fibonacciego (wzór na kolejne wyrazy ciągu znajdź w zasobach internetowych).
7. Utwórz schemat blokowy algorytmu wyszukujący trójki pitagorejskie z podanego przedziału.

VII. Zasady programowania strukturalnego

- Środowiska programistyczne języka C++
- Struktura prostego programu w C++
- Dyrektywy preprocesora
- Typy zmiennych w C++
- Operatory w C++
- Instrukcje we/wy

27 Środowisko programistyczne

ZAGADNIENIA

- Środowiska programistyczne
- Struktura prostego programu w C++
- Schemat kompilacji

Jednym z najpopularniejszych języków programowania, przeznaczonych do tworzenia programów i gier, jest C++. Umożliwia on pisanie aplikacji pod Windowsa, Linuksa i inne mniej popularne systemy operacyjne. Łatwość wykorzystywania możliwości sprzętowych komputera w połączeniu z tzw. przenośnością kodu źródłowego między platformami stwarza bardzo duże możliwości programistom z całego świata.

Na rynku jest dostępnych kilka różnych środowisk umożliwiających programowanie w C++.

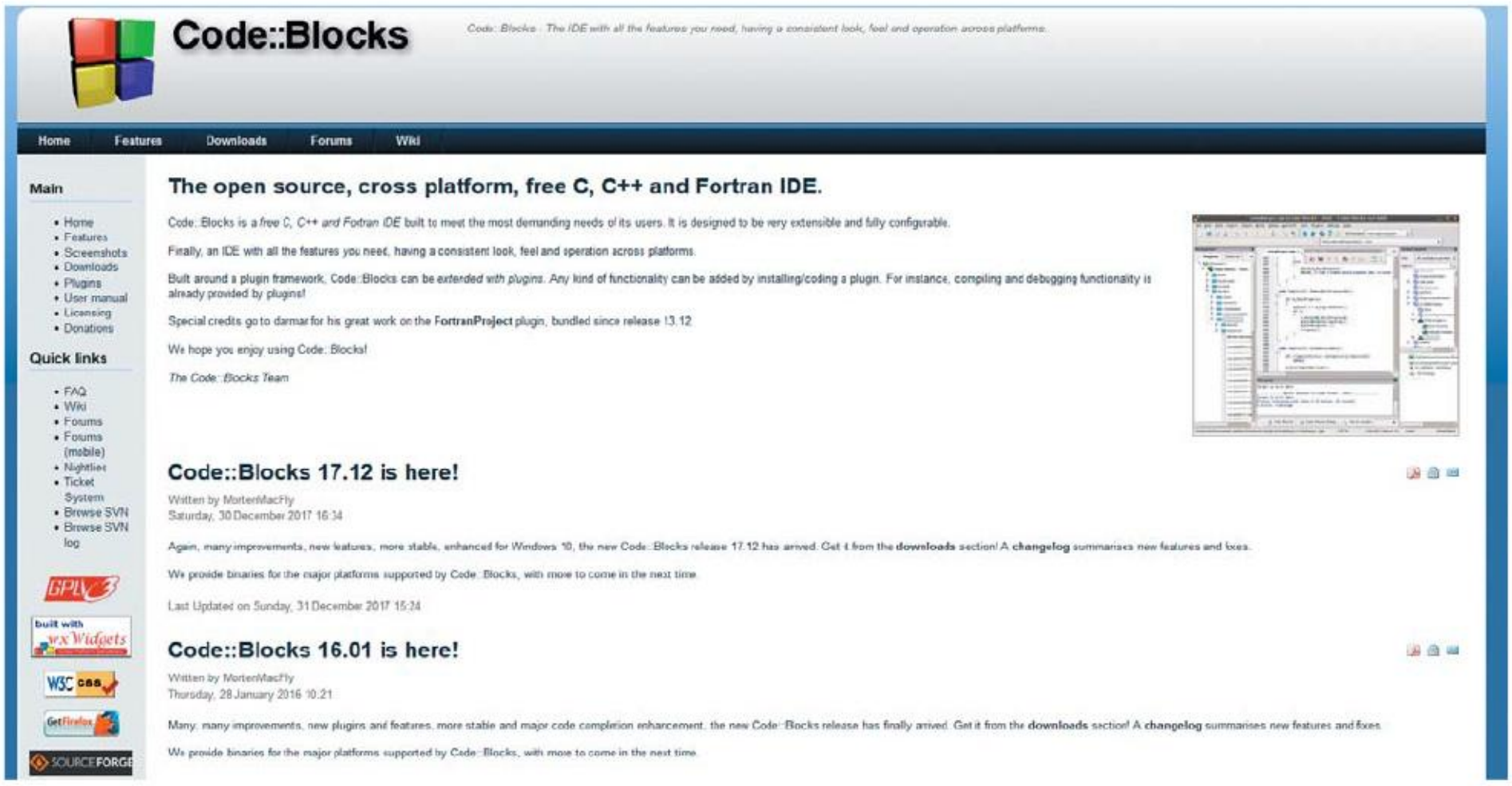
Lista dostępnych środowisk

- Dev-C++ – darmowe środowisko programistyczne C/C++, które zawiera wielookienkowy edytor kodu źródłowego z podświetlaniem składni, kompilator, debbuger oraz linker. Od kilku lat nie jest on już rozwijany. Edytor tego środowiska jest prosty w użyciu.
- Code::Blocks – darmowe środowisko programistyczne umożliwiające tworzenie aplikacji w języku C i C++. Ten program jest stale rozwijany. Dużą zaletą tego środowiska jest wieloplatformowość, co znaczy, że to środowisko jest dostępne zarówno pod Windowsa, jak i Linuksa. Zaraz po zainstalowaniu tego programu edytor jest skonfigurowany do pracy, dlatego od razu można przystępować do kodowania. To środowisko można pobrać bez kompilatora lub z kompilatorem GCC od *MinGW*. Program ma opcję zaimportowania projektów utworzonych w innych środowiskach programowania.
- Microsoft Visual C++ – płatne, rozbudowane środowisko programistyczne, które umożliwia tworzenie aplikacji dla systemu, ale jego złożoność może okazać się zbyt trudna dla początkujących programistów.

Podręcznik napisano z myślą o środowisku Code::Blocks, ponieważ:

- jest ono darmowe;
- możliwości programu są zbliżone do Visual C++;
- tego programu można używać zarówno pod Windows, jak i pod Linuksem;
- jest to odpowiednie środowisko do edukacji.

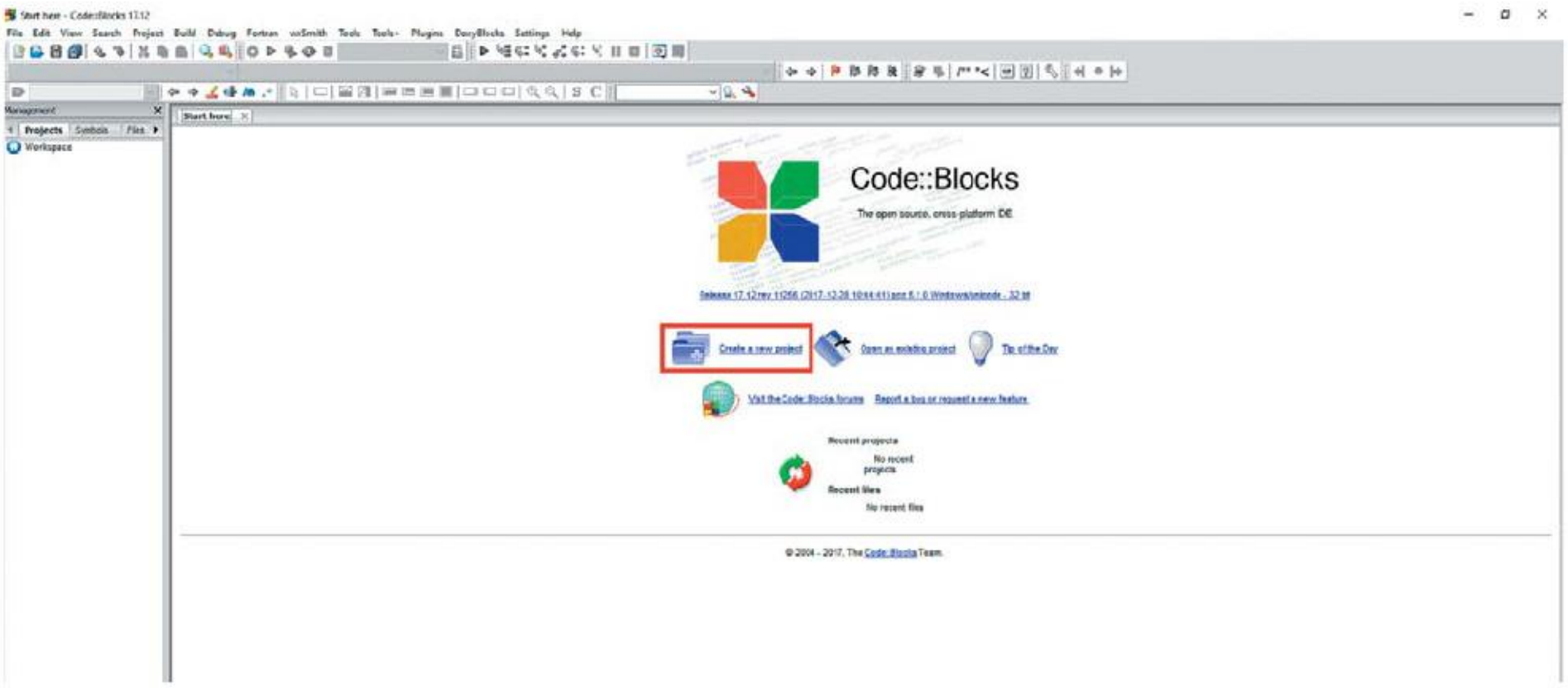
Instalacja Code::Blocks nie należy do skomplikowanych. Wystarczy je pobrać ze strony http://www.codeblocks.org/ i uruchomić plik instalacyjny. Należy pamiętać, aby w pobieranym pliku instalacyjnym zawierał się pakiet MinGW (kompilator).



Rys. 27.1. Strona, z której można pobrać Code::Blocks

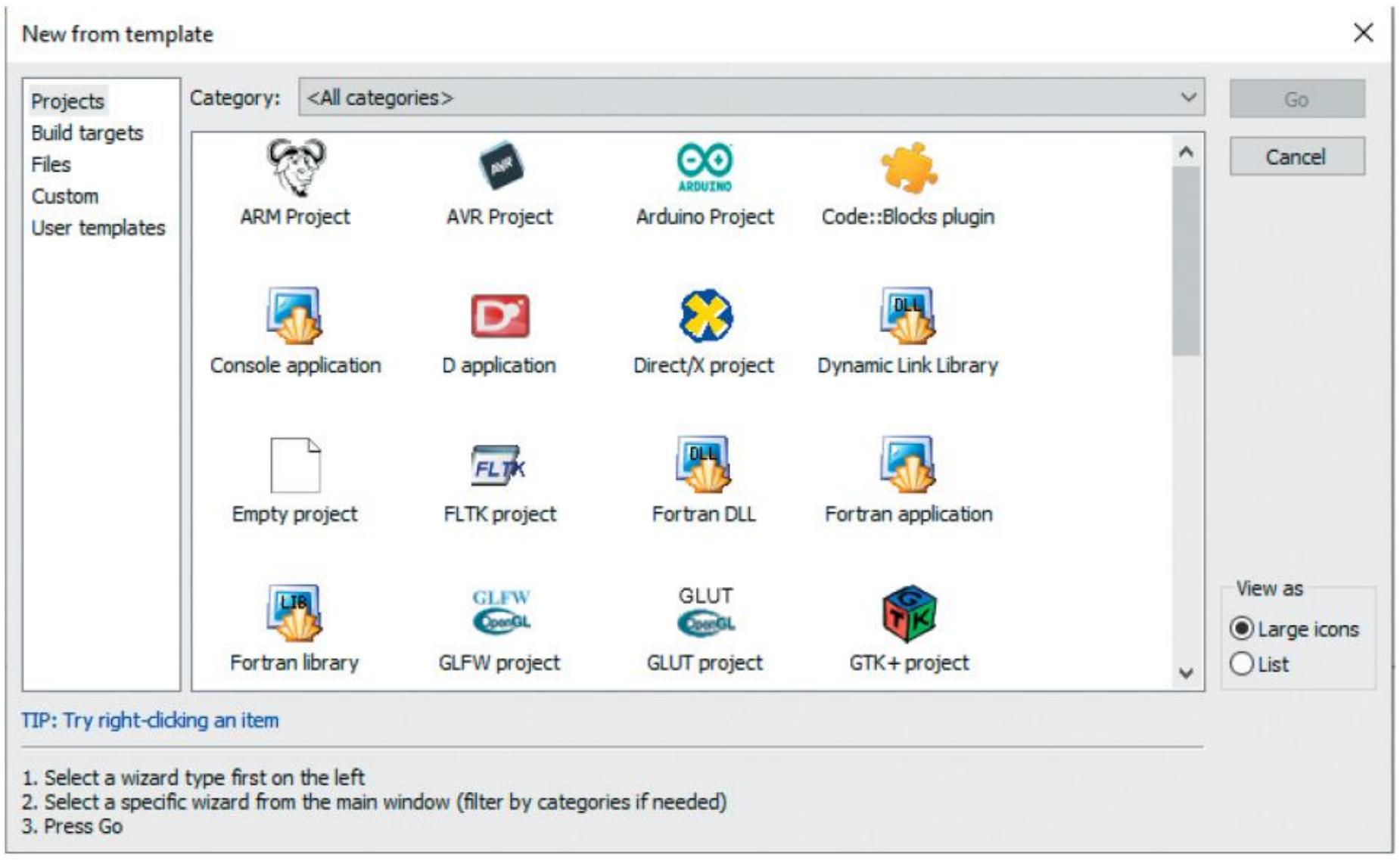
Kolejnym etapem instalacji jest pierwsze uruchomienie Code::Blocks, podczas którego należy wybrać kompilator, który będzie wykorzystywany do tłumaczenia aplikacji z języka C++ do postaci kodu wykonywalnego, tj. aplikacji w postaci pliku exe. Do paczki instalacyjnej Code::Blocks dołączono kompilator GNU GCC Compiler (to on powinien zostać wybrany).

Aby utworzyć nowy program w Code::Blocks, należy kliknąć: **FILE→NEW→PROJECT** lub bezpośrednio na **`Create a new project`**.



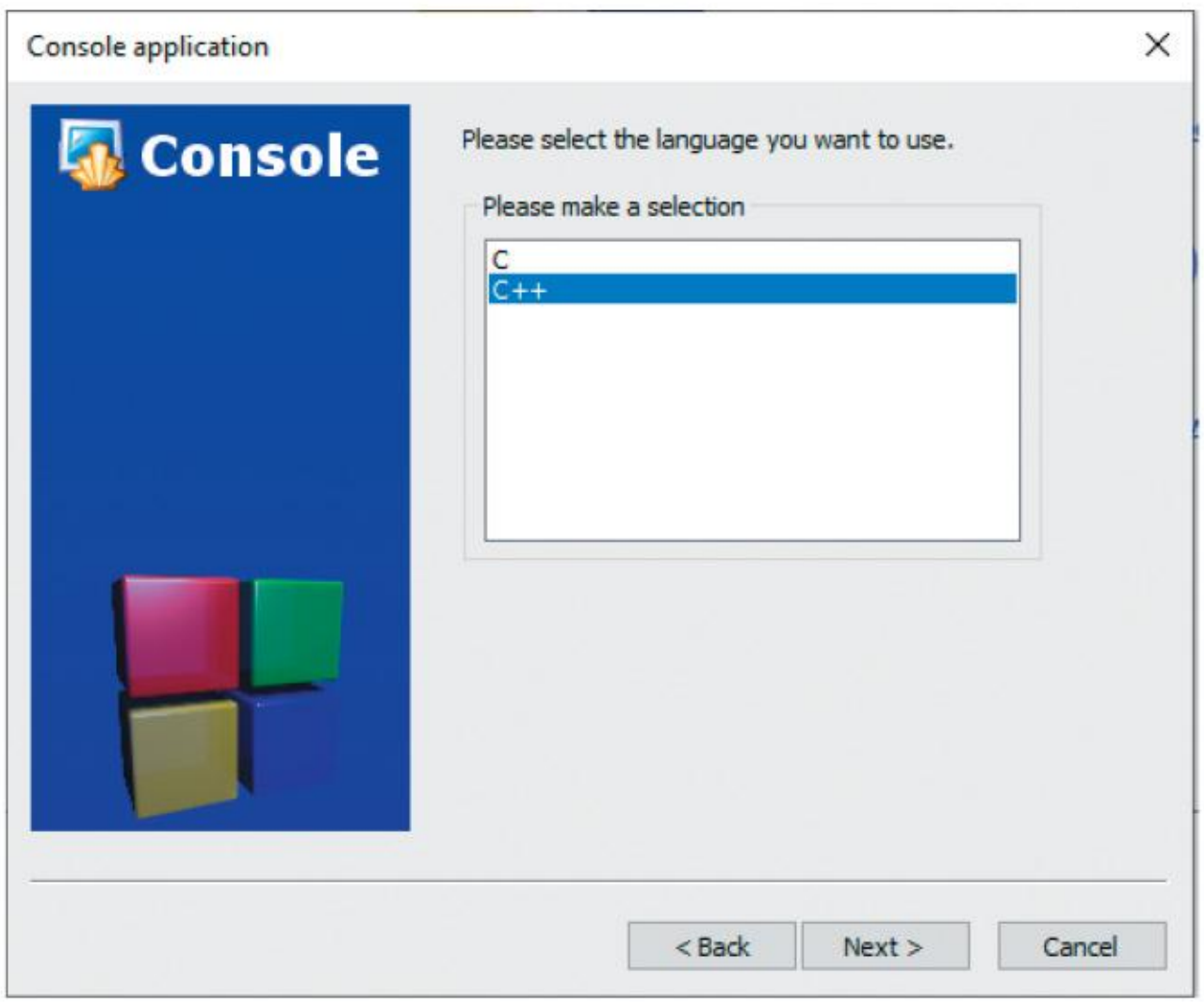
Rys. 27.2. Tworzenie nowego projektu w C++

Następnie trzeba wybrać **`Console application`** i **`Go`**.



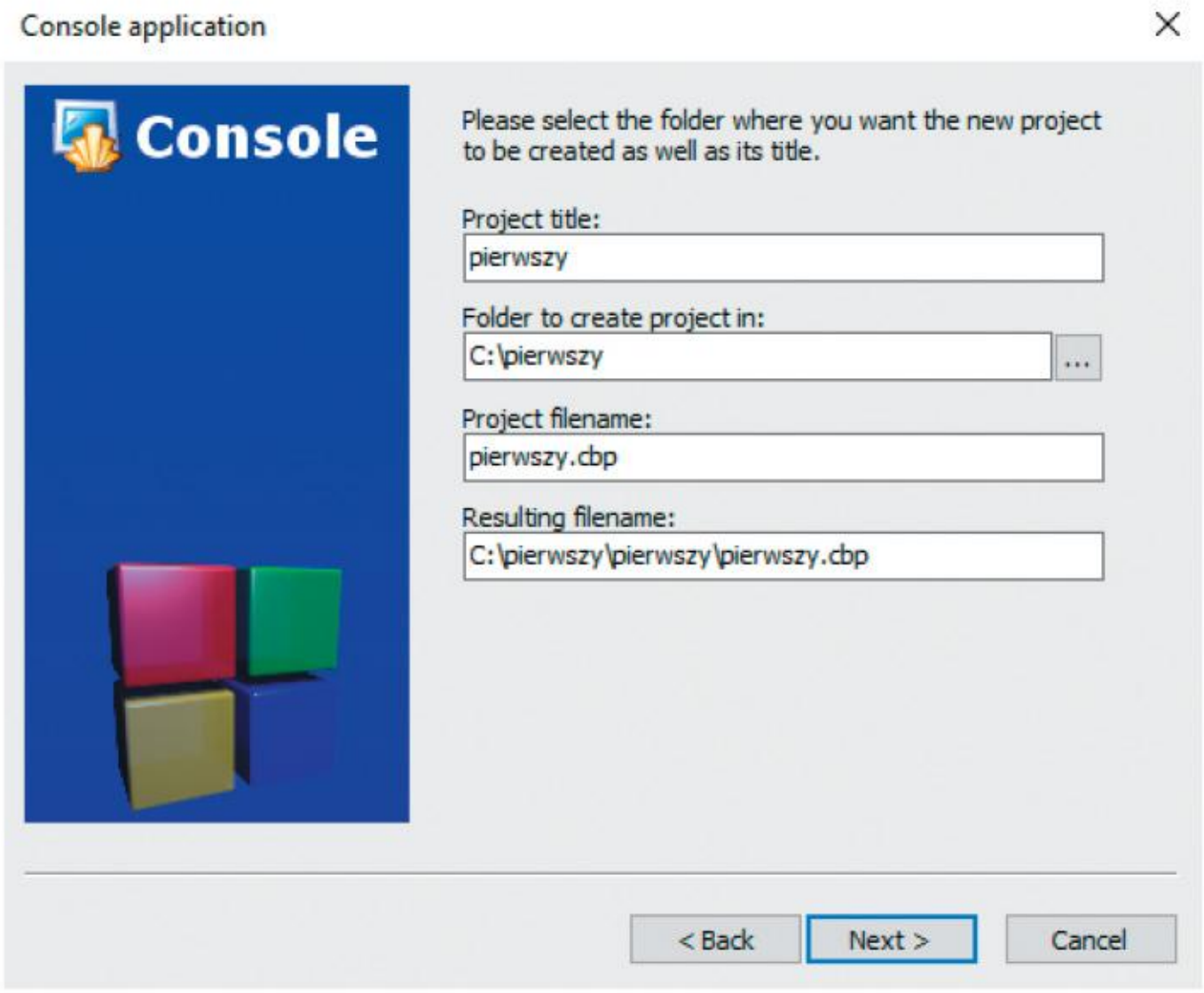
Rys. 27.3. Ciąg dalszy tworzenia nowego projektu w C++

W kolejnym kroku należy wybrać język C lub C++.



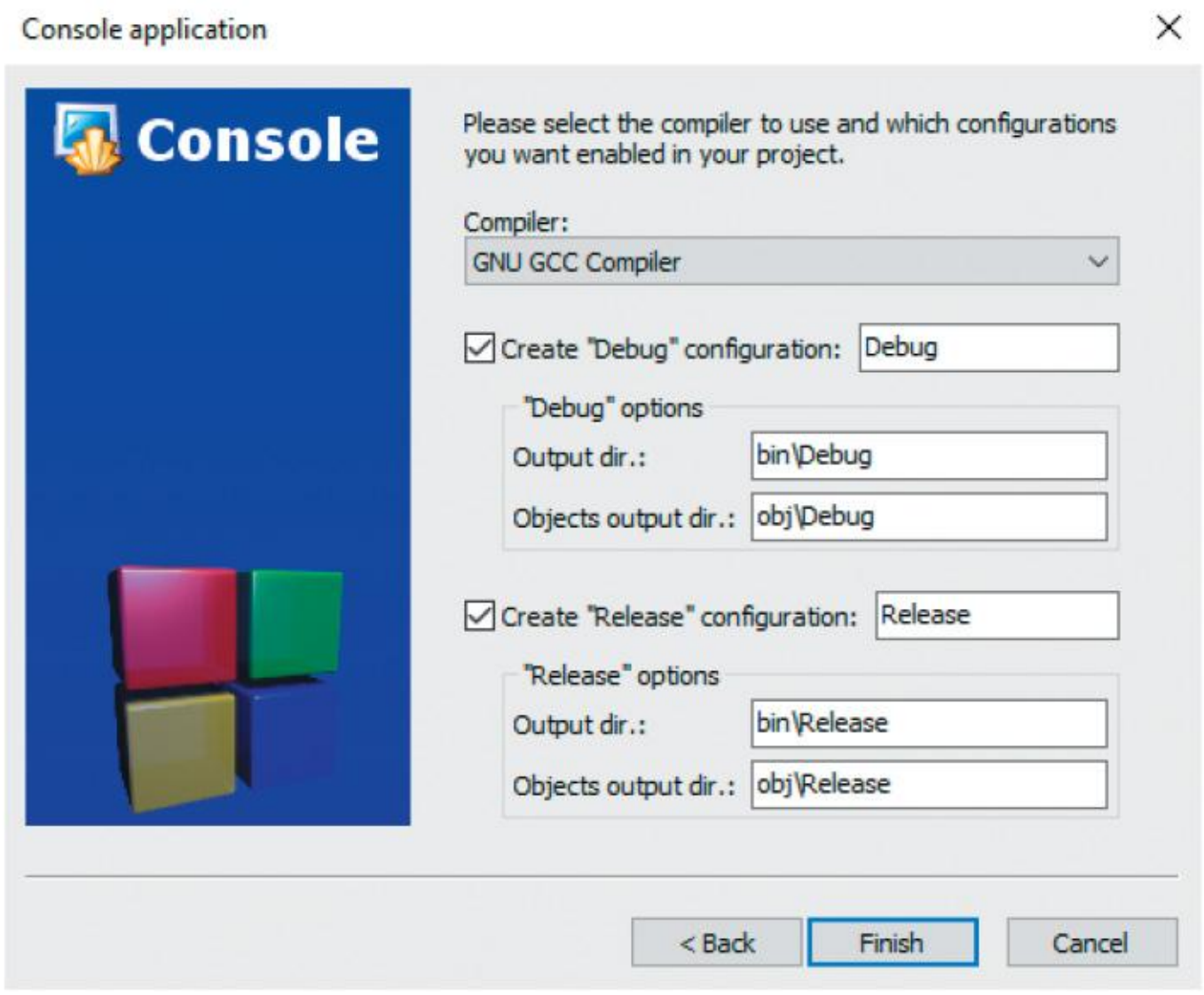
Rys. 27.4. Wybór języka programowania

W następnym oknie wstawia się nazwę projektu i określa folder docelowy. Pozostałe wartości uzupełnią się automatycznie.



Rys. 27.5. Wybór nazwy projektu i jego folderu docelowego

Na koniec otrzymuje się okno w postaci przedstawionej na rys. 27.6.



Rys. 27.6. Podsumowanie tworzenia nowego projektu

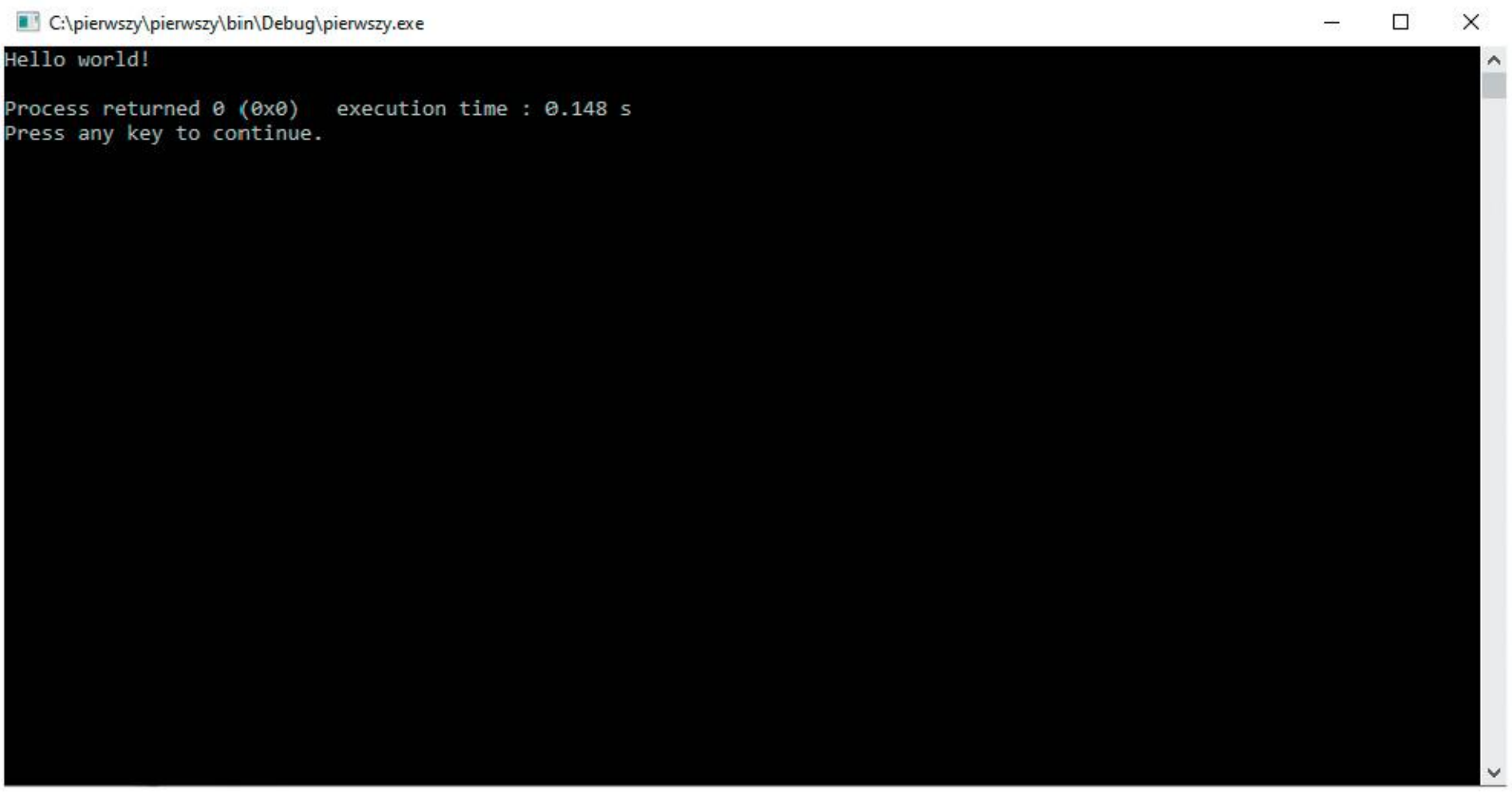
Naciśnięcie klawisza **`Finish`** poskutkuje utworzeniem nowego projektu z plikiem o nazwie **`main.cpp`**. Domyślną zawartość tego pliku przedstawiono na rys. 27.7.

main.cpp

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       cout << "Hello world!" << endl;
8       return 0;
9   }
10
```

Rys. 27.7. Początkowa zawartość głównego pliku projektu

Aby uruchomić ten program, należy nacisnąć klawisz [**F9**], którego zadaniem jest skompilowanie pliku ***.cpp** i przekształcenie go w tzw. plik wykonywalny. Otrzymamy okno w postaci pokazanej na rys. 27.8.

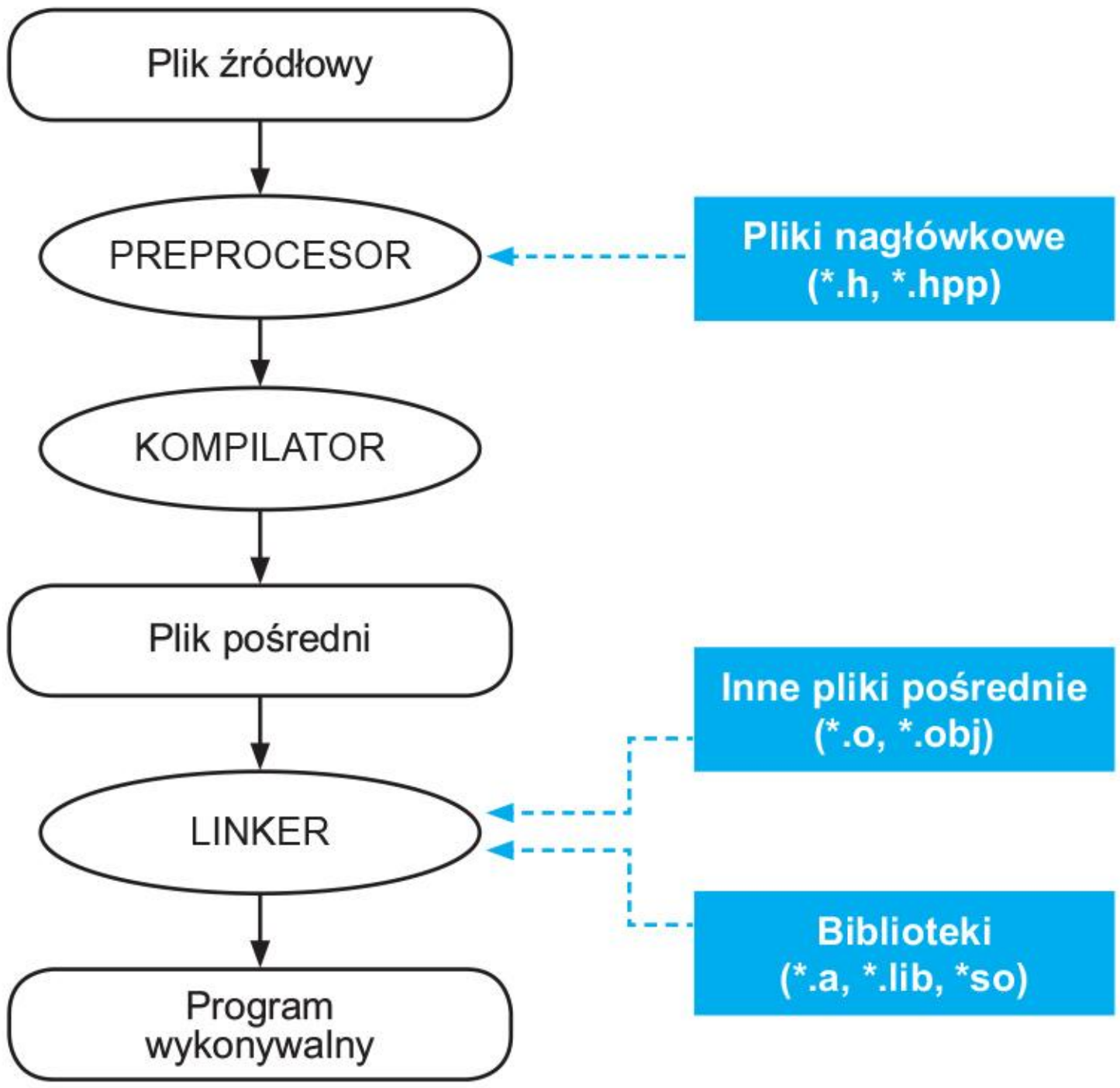


Rys. 27.8. Efekt działania programu po wciśnięciu klawisza [F9]

Kompilator to program tłumaczący kod źródłowy na plik wykonalny. Proces tłumaczenia nazywa się **kompilacją**. Jest to tylko jeden z etapów procesu tworzenia pliku wykonywalnego. W rzeczywistości są jeszcze potrzebne preprocesor i linker. Rolą linkera jest połączenie w jeden plik wykonywalny wielu plików otrzymanych w wyniku kompilacji poszczególnych plików źródłowych, których może być wiele i które mogą być kompilowane osobno i w różnym czasie.

Kompilacji nie trzeba powtarzać przed każdym uruchomieniem programu. Raz utworzony plik wykonywalny można uruchamiać dowolną liczbę razy.

Wadą języków w pełni kompilowanych, np. C++, jest uzależnienie kodu wykonywalnego od platformy. Oczywiście, sam tekst programu jest niezależny od platformy, jeśli trzyma się w nim ściśle standardu języka C++ i nie stosuje żadnych rozszerzeń specyficznych dla danego kompilatora.



Rys. 27.9. Schemat kompilacji

Na schemacie kompilacji (rys. 27.9) przedstawiono zasadę, że każdy kod źródłowy musi zostać przekształcony w plik wykonywalny, aby stał się zrozumiały dla komputera. Preprocesor jest technologią wykorzystywaną przez niektóre języki programowania, takie jak Perl, Python i C++. Jej zadaniem jest przekształcenie kodu źródłowego w **kod wyjściowy**. Jest to program komputerowy, który za pomocą tzw. dyrektyw preprocesora przeprowadza istotne (z punktu widzenia kompilatora) zmiany. Preprocesor, korzystając z dyrektyw (w C++ są to #include, #define, ...), m.in. integruje kod z zawartością załączonych bibliotek standardowych, własnych plików nagłówkowych .h, tworzy makra i definiuje stałe.

W następnej kolejności **kod** ulega kompilacji, podczas której poddaje się go analizie leksykalnej, syntaktycznej oraz semantycznej. Kompilator weryfikuje go pod kątem błędów w pisowni słów kluczowych, obecności niedozwolonych znaków i błędów składniowych języka, a następnie sprawdza poprawność typów, użytych instrukcji i nazw.

Podczas tzw. **linkowania** kod wynikowy jest łączony z innymi skompilowanymi modułami. Są nimi kody wynikowe dołączanych bibliotek oraz biblioteki uruchomieniowej, z której korzysta się w celu ograniczenia miejsca zajmowanego przez gotowy program. Efektem procesu konsolidacji jest plik wykonywalny gotowy do pracy w danym systemie.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Scharakteryzuj środowiska umożliwiające programowanie w C++.
2. Opisz proces przekształcania pliku źródłowego w plik wykonywalny.
3. Omów rolę linkera i preprocesora w procesie kompilacji.

28 Struktura prostego programu w C++

ZAGADNIENIA

- Struktura prostego programu w C++
- Dyrektywy w C++

Program napisany w języku C++ składa się zazwyczaj z części, którymi są:

- dyrektywy preprocesora,
- definicje funkcji MAIN,
- deklaracje i definicje globalnych danych i struktur danych,
- deklaracje i definicje funkcji.

```
1   #include<iostream>
2   using namespace std;          //obszar dyrcktyw
3
4   //obszar definicji i deklaracji zmiennych oraz funkcji
5
6   int main()              //nagłówek funkcji main
7   {
8       cout<<"Hallo world!"<<endl;
9       return 0;
10  }
```

Rys. 28.1. Struktura prostego programu

Na rys. 28.1 przedstawiono strukturę prostego programu w języku C++. Jak wiadomo z poprzedniego rozdziału, aby skompilować i uruchomić ten program, należy nacisnąć klawisz [**F9**]. Efektem działania będzie wyświetlenie na czarnej konsoli komunikatu „Hallo world".

Pierwsza linijka kodu zawiera potrzebne dyrektywy preprocesora. Są to polecenia, które kompilator wykonuje, zanim zacznie tworzyć kod programu. Każda dyrektywa rozpoczyna się znaczkiem **#**. Za koniec dyrektywy uważa się koniec wiersza, więc nie trzeba używać średnika kończącego polecenie. Ponadto każda dyrektywa musi występować w osobnej linii.

Najczęściej używaną dyrektywą jest **#include**. Służy ona do załączania do plików .cpp oraz .h definicji nagłówków funkcji czy też klas zawartych w oddzielnych plikach lub bibliotekach systemowych.

Podstawowe biblioteki w języku C++:

- cmath (math.h) – zawiera funkcje matematyczne, takie jak: funkcje trygonometryczne, logarytmy, zaokrąglanie liczb, pierwiastkowanie, potęgowanie;

- **cstdio** (stdio.h) – zawiera podstawowe operacje we/wy, czyli funkcje zarządzania plikami, obsługi klawiatury i ekranu;
- **iostream** (iostream.h) – biblioteka obsługująca operacje wejścia i wyjścia;
- **cstdlib** (stdlib.h) – zawiera funkcje zarządzania pamięcią, konwersja typów podstawowych, generowanie liczb losowych;
- **cstring** (string.h) – zawiera funkcje zarządzania łańcuchami znaków;
- **ctime** (time.h) – zawiera funkcje zarządzania czasem.

! UWAGA

W nawiasach umieszczono analogiczne biblioteki stosowane w języku C.

Składnia dyrektywy **#include** ma następującą postać:

#include <nazwa.h> lub **#include "plik.h"**

Załóżmy, że mamy dwa pliki źródłowe, w których chcemy użyć tej samej funkcji. Możemy zdefiniować tę funkcję dwa razy, w każdym z plików źródłowych. Wtedy kompilator jednak zaprotestuje, argumentując, że dana funkcja jest zdefiniowana dwa razy. Aby temu zapobiec, możemy zadeklarować funkcję w pliku nagłówkowym. Następnie ten plik dołączamy do obu plików źródłowych za pomocą **#include**.

Drugą najczęściej stosowaną dyrektywą jest dyrektywa **#define**, służąca do tworzenia stałych lub makr. Definiowanie stałych odbywa się w następujący sposób:

```
#define nazwa_stałej wartość
```

PRZYKŁAD 28.1

```
#define moja_stala 14.1516
```

Nie ma potrzeby definiowania stałych matematycznych, ponieważ wszystkie znajdują się w pliku nagłówkowym **cmath** (**math.h**).

Jeżeli preprocesor znajdzie w tekście programu nazwę jakiejś zdefiniowanej stałej, to zamieni ją na jej wartość i **nie sprawdzi** przy tym, czy wygenerowany kod jest poprawny.

Stała wcale jednak nie musi mieć wartości. Może być po prostu zdefiniowana i nie mówić o sobie nic więcej.

Dyrektywa **#define** służy także do tworzenia makr.

PRZYKŁAD 28.2

```
#include<iostream>
#define wyr(x, y) ((x*y)/2) //definicja makra
using namespace std;
int main()
```

```
{
float a=wyr(5, 4);
cout<<"Po zastosowaniu wyrazenia otrzymamy: "<<a;
return 0;
}
```

Polecenie **`using namespace std;`** nakazuje użycie standardowej przestrzeni nazw **`std`**. To pojęcie pozwala określić, które zmienne i funkcje można zastosować w danym miejscu. Można tworzyć własne przestrzenie nazw, my jednak będziemy korzystać ze standardowej.

Po wierszu **`using namespace std;`** można umieścić definicje funkcji własnych oraz deklaracje zmiennych globalnych wykorzystywanych w programie. Na końcu wstawiamy definicję najważniejszej funkcji, czyli funkcję **`main()`**, wewnątrz której tworzymy kod programu. Ciało każdej funkcji będziemy umieszczać w kalmarach {}. Przed klamrą zamykającą wpisujemy polecenie:

```
system("pause");
```

Zatrzymuje ono wykonanie programu do momentu naciśnięcia jakiegoś klawisza, co pozwala zobaczyć efekt działania programu.

Instrukcja **`return 0;`** powoduje, że funkcja **`main()`** zwraca wartość `0`, co oznacza, że program skompilował się pomyślnie.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień nazwy części składowych programu napisanego w języku C++.
2. Omów dyrektywę **`#define`** do tworzenia stałych i makr. Podaj po jednym przykładzie zastosowania.
3. Opisz składnię dyrektywy **`#include`**.
4. Wymień podstawowe biblioteki stosowane w C++.

29 Typy zmiennych w C++

ZAGADNIENIA

- Definicja zmiennej
- Zasady tworzenia nazw zmiennych
- Typy zmiennych
- Klasy zmiennych

Zmienna to pewne miejsce w pamięci komputera, któremu można przypisać różne wartości. Każda zmienna ma swój adres w pamięci oraz nazwę. Przed użyciem zmiennej w programie należy ją zadeklarować. Deklaracja nie przypisuje wartości zmiennej, lecz przydziela jej adres w pamięci (rezerwuje dla niej miejsce).

Przykładowa deklaracja może wyglądać następująco:

```
typ_zmiennej NazwaZmiennej;
```

Podczas tworzenia nazwy nowej zmiennej należy pamiętać, że:

- nazwa zmiennej nie może rozpoczynać się od liczby;
- w nazwie nie wolno umieszczać spacji, trzeba zastąpić ją znakiem „_”;
- nie wolno stosować znaków specjalnych;
- nie wolno używać polskich liter;
- kompilator rozróżnia małe i duże litery, czyli zmienna Suma oraz suma to kompletnie inne zmienne;
- w nazwach nie wolno używać słów kluczowych:
  alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t, char32_t, class, compl, const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, noexcept, not, not_eq, nullptr, operator, or, or_eq, private, protected, public, register, reinterpret_cast, return, short, signed, sizeof, static, static_assert, static_cast, struct, switch, template, this, thread_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar_t, while, xor, xor_eq.

Jak już wiadomo, podczas tworzenia zmiennej ustalamy jej nazwę oraz typ, który określa zakres przechowywanych wartości.

Jeśli nowa zmienna ma nazwę składającą się z kilku wyrazów, to zamiast łączenia ich znakiem „_” można zastosować zapis w postaci:

```
typ KolejneSlowoNazwyWielkaLitera;
```

Typy w języku C++ można podzielić na dwa sposoby:

- pierwszy podział dzieli typy na fundamentalne (podstawowe) i pochodne;
- drugi podział dzieli typy na wbudowane i zdefiniowane przez użytkownika.
- Podstawowymi typami są:
- typ znakowy – może reprezentować pojedynczy znak;
- typ numeryczny – może przechowywać liczby całkowite i zmiennoprzecinkowe;
- typ logiczny – może przechowywać dwie wartości: **TRUE** lub **FALSE**.

C++ pozwala także na użycie pewnych modyfikatorów przed typem zmiennej:

- **signed** – zmienna może przechowywać wartości dodatnie i ujemne (ma znak +/-);
- **unsigned** – zmienna może przechowywać tylko wartości dodatnie;
- **short** – zmienna jest typu krótkiego;
- **long** – zmienna jest typu długiego.

Zakresy podstawowych typów zmiennych języka C++ zamieszczono w tabeli 29.1.

Tabela 29.1. Typy fundamentalne i ich zakres

Grupa zmiennych	Nazwa zmiennej	Rozmiar	Zakres wartości
typ znakowy	char	1 bajt	od –128 do 127
typ całkowity (signed) • liczby ze znakiem	short short int signed int	2 bajty	od –32768 do 32767
	int signed int signed	4 bajty	od -2^{31} do $2^{31}-1$
	long signed long int signed long	4 bajty	od -2^{31} do $2^{31}-1$
	long long long long int signed long long signed long long int	4 bajty	od -2^{31} do $2^{31}-1$

Grupa zmiennych	Nazwa zmiennej	Rozmiar	Zakres wartości
typ całkowity (unsigned) • liczby bez znaku	unsigned short int unsigned short	2 bajty	od 0 do $2^{16}-1$
	unsigned int unsigned	4 bajty	od 0 do $2^{32}-1$
	unsigned long int unsigned long	4 bajty	od 0 do $2^{32}-1$
	unsigned long long int unsigned long long	8 bajtów	od 0 do $2^{64}-1$
typ zmiennoprzecinkowy	float	4 bajty	od $-3{,}4 \cdot 10^{38}$ do $3{,}4 \cdot 10^{38}$
	double	8 bajtów	od $-1{,}8 \cdot 10^{308}$ do $1{,}8 \cdot 10^{308}$
	long double	10 bajtów	od $-1{,}1 \cdot 10^{4932}$ do $1{,}1 \cdot 10^{4932}$
typ logiczny	bool	1 bajt	

Jeżeli będziemy korzystać z kilku zmiennych tego samego typu, wystarczy podać tylko raz typ, a następnie – nazwy kilku zmiennych, oddzielone przecinkami, np.:

```
int zmienna1, zmienna2, zmienna3;
```

Można także określać klasy zmiennej. Typ zmiennej definiuje sposób interpretacji przechowywanych w pamięci bitów, natomiast klasa zmiennej decyduje o sposobie przechowywania zmiennej w pamięci. W C++ występują następujące klasy zmiennych:

- **`const`** – stała, nie można zmieniać wartości zmiennej;
- **`static`** – zmienna w momencie uruchomienia programu otrzymuje stałe miejsce w pamięci;
- **`auto`** – przydział miejsca w pamięci dla zmiennej następuje dynamicznie (na stosie) w czasie wykonywania bloku, w którym zmienna została zadeklarowana; po zakończeniu wykonywania instrukcji z danego bloku pamięć po zmiennej zostaje zwolniona; to domyślna klasa zmiennych;
- **`register`** – zmienne lokalne, tzn. dostępne tylko w bloku, w którym zostały zadeklarowane;
- **`extern`** – jeżeli zmienna została tylko raz zadeklarowana w pojedynczym segmencie programu, zostanie w tym segmencie umieszczona w pamięci i potraktowana podobnie jak zmienna klasy *static*;
- **`volatile`** – oznacza zmienną, z której mogą korzystać także inne procesy; inny proces może zmieniać wartość tej zmiennej, przez co przed każdym jej użyciem musi ona zostać odczytana na nowo.

Wszystkie zmienne można podzielić na zmienne **globalne** i zmienne **lokalne**. Zmienne globalne są dostępne w **całym programie i przez cały czas jego działania**, natomiast zmienne lokalne są dostępne tylko w **pewnej części programu**, zazwyczaj tylko w **pewnym momencie działania programu**.

Zmienne globalne deklaruje się między blokiem dołączonych plików nagłówkowych a funkcją **`main`**. Wszystkie zmienne globalne są widoczne wewnątrz funkcji **`main`**. Oznacza to, że wewnątrz funkcji **`main`** (oraz wszystkich innych funkcji) można przeprowadzać wszystkie operacje, jakie są tylko możliwe dla zmiennej danego typu. Trzeba jednak pamiętać, że podczas tworzenia tego typu zmiennych istnieje niebezpieczeństwo przypadkowego nadpisania jej wartości, co może spowodować niewłaściwe działanie programu. To dlatego zaleca się korzystanie ze zmiennych lokalnych, które można deklarować wewnątrz funkcji lub danym bloku programu.

PRZYKŁAD 29.1

```
#include<iostream>
using namespace std;

int liczba1;     //zmienna globalna typu całkowitego
float a,b;       //zmienne globalne typu zmiennoprzecinkowego

int main()
{
    float wiek; //zmienna lokalna
    cout<<"Ile masz lat?"<<endl;
    cin>>wiek;
    return 0;
}
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień typy zmiennych w C++.
2. Jakie są zasady tworzenia nazw zmiennych?
3. Określ zakres zmiennych poszczególnych typów całkowitych i zmiennoprzecinkowych.
4. Jaka jest różnica między zmienną globalną a zmienną lokalną?

30 Instrukcje we / wy oraz operatory i ich działanie

ZAGADNIENIA

- Rodzaje instrukcji we / wy
- Rodzaje i zastosowanie operatorów

Do wyświetlania danych na ekran służy funkcja **cout** zdefiniowana w bibliotece **iostream** (out – jak wyjście, np. na ekran monitora). Jego konstrukcja jest następująca:

```
std::cout << "Treść komunikatu";
```

Jeżeli do programu dołączymy linijkę: **using namespace std;**, zapis możemy skrócić do postaci: **cout << "Treść komunikatu";**.

Powyższa konstrukcja pozwala wyświetlać napis, który jest umieszczony w cudzysłowie na ekranie monitora. Dodatkowym elementem jest operator wyjścia <<, który pokazuje kierunek przekierowania strumienia znaków.

Aby wyświetlić wartość zmiennych lub działań, zapisujemy je bez użycia cudzysłowu, np.:

```
cout << 2*b + 2*a;
```

Możemy także łączyć treść komunikatu z wartościami zmiennych lub wyrażeniami, oddzielając każde z nich operatorem **<<**, np.:

```
cout <<"Liczba "<<3*7<<" jest większa niż "<<2*7 << endl;
```

Na ekranie monitora otrzymamy wówczas następującą informację:

```
Liczba 21 jest większa niż 14
```

Manipulator strumienia **endl** wstawia znak nowej linii i przenosi do niej kursor.

Do pobierania danych z klawiatury służy funkcja **cin**, która jest również zdefiniowana w bibliotece **iostream**.

Tak jak w przypadku obiektu **cout**, mamy do dyspozycji dwie możliwości:

```
std::cin>>nazwa_zmiennej;
```

lub po dodaniu linijki

```
using namespace std;
```

mamy postać:

```
cin>>nazwa_zmiennej;
```

Wtedy stawiamy znaki przekierowania strumienia w odwrotną stronę: **>>**.

Jeżeli chcemy podać wartości kilku zmiennych, można to zrobić, jeżeli zmienne oddzieli się znakiem wejścia strumienia, np.:

```
cin>> zmienna1>> zmienna2>> zmienna3;
```

Wczytywać dane i wypisywać je na ekranie komputera można także za pomocą instrukcji **printf** oraz **scanf**. Działają one znacznie szybciej niż **cout** i **cin** oraz oferują więcej możliwości programiście.

Aby móc skorzystać z operacji we / wy **printf** oraz **scanf**, należy dodać bibliotekę **stdio**. Pierwsza z nich ma następującą postać:

```
printf("Treść komunikatu");
```

Jeżeli w środek tekstu ujętego w cudzysłów wpiszemy znaki **\n**, to efekt programu będzie następujący:

```
Treść
komunikatu
```

UWAGA

Znak **\n** (n – jak: *new line*, czyli: nowa linia) powoduje, że w trakcie wypisywania tekstu na ekranie następuje przejście do nowej linii i dalszy ciąg tekstu jest wypisywany poniżej. Oprócz niego można zastosować następujące znaki:

\t wstawienie znaku tabulacji,
\' wstawienie apostrofu,
\" wstawienie cudzysłowu,
\? wstawienie znaku zapytania.

W C++ komentarze można zapisać w dwojaki sposób:

- jako komentarz podwójnego ukośnika **//**
- jako ukośnik i gwiazdka **/***

Komentarze podwójnego ukośnika ignorują tekst do końca linii, natomiast komentarze ukośnika i gwiazdki ignorują tekst zawarty aż do znaku gwiazdki i ukośnika ***/**.

Jak już nadmieniono, w treści komunikatu można wyświetlić wartości zmiennych lub określonych działań. Aby wprowadzić formuły działające na wprowadzanych danych, należy zastosować operatory.

Operator to jeden lub kilka znaków (zazwyczaj niebędących literami), które mają specjalne znaczenie w języku programowania.

W C++ wyróżnia się następujące rodzaje operatorów:

- operatory arytmetyczne – wykonują działania na dwóch obiektach, czyli są to operatory dwuargumentowe:
  a) **+** operator dodawania,
  b) **–** operator odejmowania,

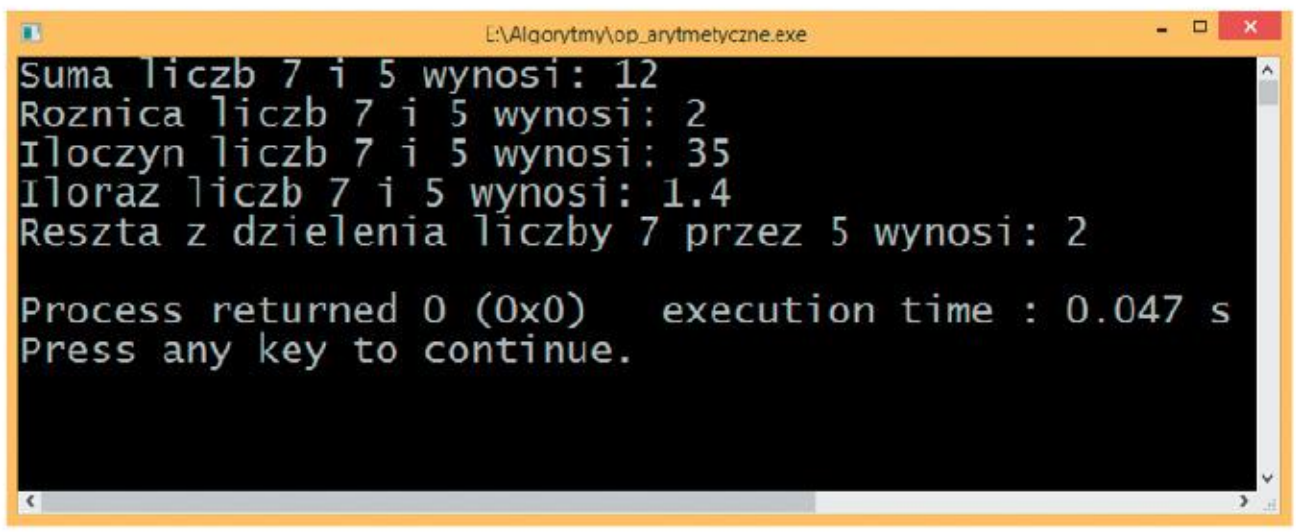
c) `*` operator mnożenia,
d) `/` operator dzielenia,
e) `%` operator modulo (wyznacz resztę z dzielenia);

PRZYKŁAD 30.1

```
#include<iostream>
using namespace std;
float suma,roznica,iloczyn,iloraz;
int modulo;
int main()
{
    suma=7+5;
    cout<<"Suma liczb 7 i 5 wynosi: "<<suma<<endl;
    roznica=7-5;
    cout<<"Roznica liczb 7 i 5 wynosi: "<<roznica<<endl;
    iloczyn=7*5;
    cout<<"Iloczyn liczb 7 i 5 wynosi: "<<iloczyn<<endl;
    iloraz=7.0/5;
    cout<<"Iloraz liczb 7 i 5 wynosi: "<<iloraz<<endl;
    modulo=7%5;
    cout<<"Reszta z dzielenia 7 przez 5 wynosi: "<<modulo;
    return 0;
}
```

Działanie operatorów arytmetycznych jest analogiczne jak operatorów stosowanych w matematyce, z wyjątkiem operatora modulo, który zwraca resztę z dzielenia dwóch liczb całkowitych. Operator `%` jest stosowany zawsze, gdy chcemy sprawdzić, czy podana wartość jest liczbą podzielną przez dowolną liczbę całkowitą, np. czy liczba jest parzysta.

Efektem działania powyższego kodu będzie wyświetlenie następujących informacji pokazanych na rys. 30.1.



Rys. 30.1. Przykład działania operatorów arytmetycznych

- **operatory porównania (relacji)** – są także operatorami dwuargumentowymi; stosuje się je podczas definiowania warunków dla danych zmiennych w programie:
  a) **>** większy od,
  b) **<** mniejszy od,
  c) **>=** większy od lub równy,
  d) **<=** mniejszy od lub równy,
  e) **==** równy,
  f) **!=** różny od;
- **operatory logiczne** – wykorzystywane do łączenia kilku warunków dla danej grupy zmiennych:
  a) **!** negacja,
  b) **&&** iloczyn logiczny (koniunkcja),
  c) **||** suma logiczna (alternatywa);
- **operatory przypisania**:
  a) **=** zwykłe przypisanie,
  b) **+=** dodaj i przypisz,
  c) **-=** odejmij i przypisz,
  d) ***=** pomnóż i przypisz,
  e) **/=** podziel i przypisz;

PRZYKŁAD 30.2

```
#include<iostream>
using namespace std;
int main()
{
int a=10;                    //przypisanie zmiennej a wartości 10
cout<<"Zmienna a ma wartosc: "<<a<<endl;
a+=2;                        //zwiekszenie a o 2
cout<<"Zmienna a po zwiekszeniu o 2 ma wartosc: "<<a<<endl;
a*=2;                        //dwukrotne zwiekszenie a
cout<<"Zmienna a po zwiekszeniu dwukrotnym ma wartosc:
"<<a<<endl;
a-=3;                        //zmniejszenie a o 3
cout<<"Zmienna a po zmniejszeniu o 3 ma wartosc: "<<a<<endl;
return 0;
}
```

Zapis **a+=2** jest równoważny zapisowi **a=a+2**. Analogicznie **a*=2** jest zgodne z zapisem **a=2*a** oraz **a-=3** działa identycznie jak zapis **a=a-3**. Efektem działania powyższego kodu jest rezultat pokazany na rys. 30.2.

```
Zmienna a ma wartosc: 10
Zmienna a po zwiekszeniu o 2 ma wartosc: 12
Zmienna a po zwiekszeniu dwukrotnym ma wartosc: 24
Zmienna a po zmniejszeniu o 3 ma wartosc: 21

Process returned 0 (0x0)   execution time : 0.359 s
Press any key to continue.
```

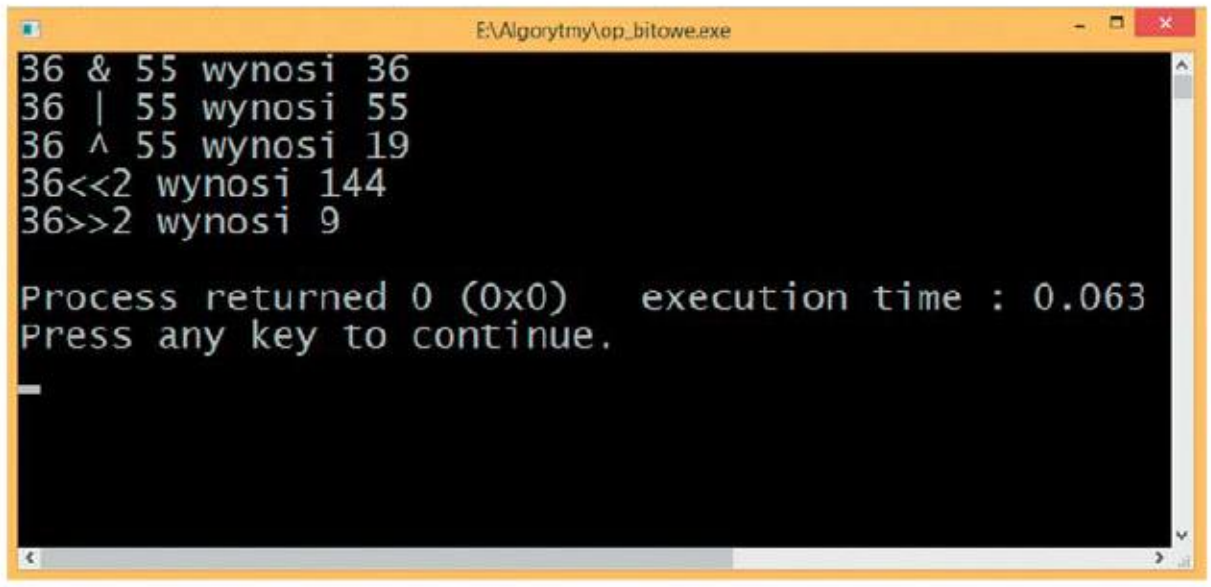
Rys. 30.2. Przykład zastosowania operatorów przypisania

- **operatory inkrementacji i dekrementacji** – operatory zwiększania wartości zmiennej o jeden i zmniejszania wartości o jeden; to operatory jednoargumentowe.
  Działanie typu `i=i+1` można zastąpić zapisem `i++`, natomiast działanie `i=i-1` można zastąpić zapisem `i--`.
  Jeżeli zmiennej `i` przypiszemy wartość `5`, to instrukcja `i++` zmieni wartość zmiennej na liczbę `6`.
- **operatory bitowe** – wykonują operacje na reprezentacji bitowej danych zmiennych:
  a) `<<` przesunięcie w lewo,
  b) `>>` przesunięcie w prawo,
  c) `&` bitowy iloczyn logiczny,
  d) `|` bitowa suma logiczna,
  e) `^` bitowa różnica symetryczna,
  f) `~` bitowa negacja.

PRZYKŁAD 30.3

```
#include<iostream>
using namespace std;
int main()
{
int a=36;     //a=100100
int b=55;     //b=110111
int iloczyn_bitowy=a&b;
cout<<"36 & 55 wynosi "<<iloczyn_bitowy<<endl;
int suma_bitowa=a|b;
cout<<"36 | 55 wynosi "<<suma_bitowa<<endl;
int roznica_symetryczna=a^b;
cout<<"36 ^ 55 wynosi "<<roznica_symetryczna<<endl;
int przesuniecie_lewo=36<<2;
cout<<"36<<2 wynosi "<<przesuniecie_lewo<<endl;
int przesuniecie_prawo=36>>2;
cout<<"36>>2 wynosi "<<przesuniecie_prawo<<endl;
return 0;
}
```

Efekt programu pokazano na rys. 30.3.



Rys. 30.3. Działanie operatorów bitowych

Działanie poszczególnych operatorów przedstawiono w tabeli 30.1.

Tabela 30.1. Zapis dziesiętny i binarny operatorów binarnych z przykładu 30.3

Zapis dziesiętny liczby	Zapis binarny liczby
36	00100100
55	00110111
36 & 55	00100100
36 | 55	00110111
36 ^ 55	00010011
36<<2	10010000
36>>2	00001001

Podczas stosowania operatorów należy pamiętać o kolejności ich wykonywania. Jeśli nie jesteśmy pewni priorytetów określonych działań, to zastosujmy nawiasy okrągłe, w których można zgrupować dany zakres obliczeń.

Priorytety operatorów przedstawiono w tabeli 30.2.

Tabela 30.2. Priorytety operatorów

Priorytet	Operator	Opis
1	: :	zakres
2	()	nawiasy
	++	inkrementacja przyrostkowa
	--	dekrementacja przyrostkowa

Priorytet	Operator	Opis

3	`++`
	`--`
	`*`
		inkrementacja przedrostkowa
		dekrementacja przedrostkowa
		wskaźnik

4	`(typ)`	rzutowanie, np. (int)x

5	`*`
	`/`
	`%`
		mnożenie
		dzielenie
		modulo

6	`+` `-`	dodawanie odejmowanie
7	`<<` `>>`	przesunięcie bitów w lewo przesunięcie bitów w prawo

9	`<`
	`>`
	`>=`
	`<=`
		porównywanie

10	`==` `!=`	porównywanie
11	`&`	bitowy iloczyn
12	`^`	bitowa różnica
13	`|`	bitowa suma
14	`&&`	iloczyn logiczny
15	`||`	suma logiczna
16	`x ? y:z`	operator warunkowy

17	`=`
	`*= %= /= -= +=`
	`>>= <<= &= ^=`
	`!=`
		przypisanie

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień operatory w C++ i je omów.
2. Podaj postać instrukcji **`cout`** oraz **`cin`**.
3. Do czego służy operator modulo?
4. Wymień operatory jednoargumentowe i je omów.
5. Omów sposoby tworzenia komentarzy w C++.
6. Który operator ma najniższy priorytet, a który – najwyższy?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz program, który wczyta trzy liczby: a, b, c. Na ich podstawie oblicz wartość wyrażenia $\frac{(a+2*b)}{4}+c^2$. Wynik wyświetl na ekranie monitora.
2. Napisz program, który poprosi użytkownika o podanie dwóch liczb całkowitych: x oraz y. Wykonaj dzielenie liczb x przez y. Wynik wyświetl na ekranie. Czy otrzymany rezultat jest wartością całkowitą, czy – zmiennoprzecinkową? Przedyskutuj ten problem z klasą.

31 Instrukcje sterujące

ZAGADNIENIA

- Instrukcja warunkowa
- Instrukcja `SWITCH`
- Pętla `WHILE`
- Pętla `DO...WHILE`
- Pętla `FOR`

Instrukcje służą do sterowania przebiegiem programu. Dzięki nim podejmuje się decyzje o wykonaniu danych instrukcji w programie. Decyzja zależy od sprawdzenia określonych warunków, zatem od tego, czy jest on prawdziwy (przyjmuje wartość różną od zera – PRAWDA), czy – fałszywy (przyjmuje wartość 0 – FAŁSZ).

W języku C++ instrukcje zawsze kończymy średnikiem (instrukcja może być zapisana w kliku wierszach). Najprostsza instrukcja składa się tylko ze średnika. Nazywa się ją wówczas **instrukcją pustą**. Jest ona użyteczna np. w niektórych pętlach.

Najczęściej używaną instrukcją jest tzw. **instrukcja przypisania**, która ma następującą postać:

```
zmienna = działanie;
```

PRZYKŁAD 31.1

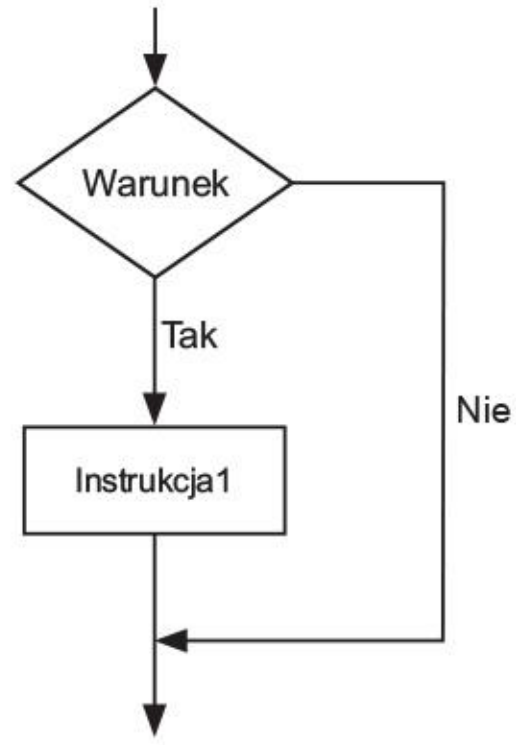
```
x=a+b;
```

Wszystko, co staje się wartością, w C++ jest uważane za **wyrażenie**. Każde wyrażenie zwraca daną wartość. Powyższy przykład oznacza, że do zmiennej **x** zostanie przypisana wartość sumy zmiennej **a** i **b**.

Instrukcja warunkowa

Instrukcja warunkowa może mieć dwie postaci: niepełną i pełną. **Warunek** to wyrażenie, które ma jakąś wartość. Jeżeli jest ona niezerowa, czyli ma wartość typu PRAWDA, to jest wykonywana **`Instrukcja1`**, w przeciwnym przypadku **`Instrukcja1`** nie jest wykonywana, następuje zakończenie programu lub przejście do kolejnego bloku instrukcji.

Niepełna instrukcja warunkowa



Rys. 31.1. Schemat blokowy niepełnej instrukcji warunkowej

Postać instrukcji:

```
if (warunek)
{
Instrukcja1;
}
```

PRZYKŁAD 31.2

Napisz program obliczający pole kwadratu.
Należy pamiętać, że długości boków każdej figury geometrycznej są liczbami dodatnimi. Trzeba więc sprawdzić, czy podana przez użytkownika liczba ma wartość większą od zera. Jeśli tak, to zostanie obliczone pole kwadratu, a w przeciwnym razie nastąpi zakończenie programu.

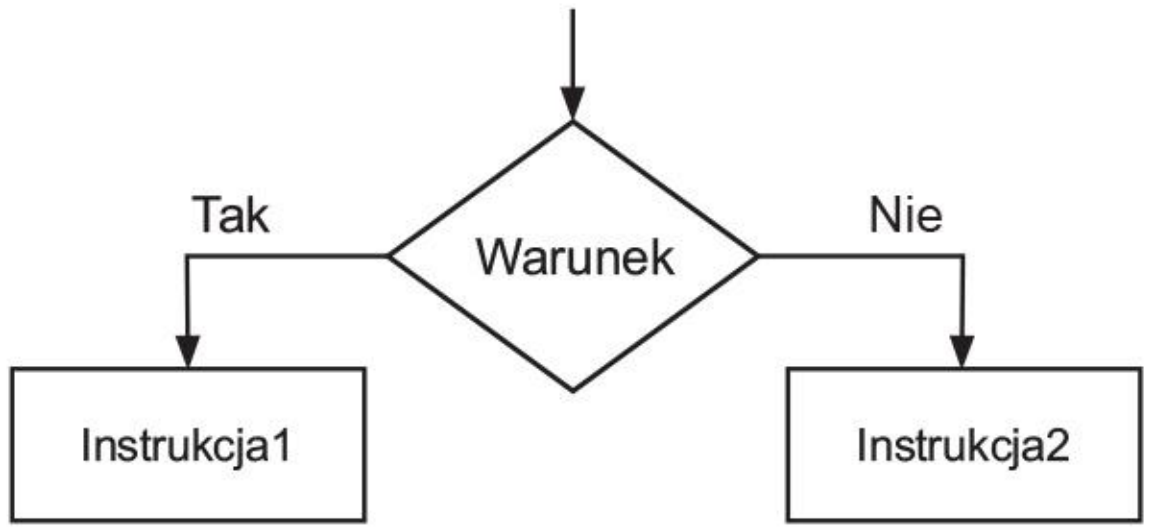
```
#include <iostream>
using namespace std;
int main()
{
float a,pole;
cout<<"Podaj dlugosc boku kwadratu: ";
cin>>a;
    if (a>0)
       {
         pole=a*a;
         cout<<"Pole kwadratu wynosi: "<<pole;
       }
return 0;
}
```

UWAGA

W standardowej konfiguracji programu Code::Blocks polskie znaki mogą nie być wyświetlane poprawnie.

Pełna instrukcja warunkowa

W pełnej instrukcji warunkowej, jeżeli wartość **`Warunku`** jest niezerowa, to jest wykonywana **`Instrukcja1`**. W przeciwnym przypadku jest wykonywana **`Instrukcja2`**.



Rys. 31.2. Schemat blokowy pełnej instrukcji warunkowej

Postać instrukcji:

```
if (warunek)
{
Instrukcja1;
}
else
{
Instrukcja2;
}
```

PRZYKŁAD 31.3

Napisz program obliczający pole kwadratu. Użycie pełnej instrukcji warunkowej w tym przykładzie sprawi, że poprawi się działanie poprzedniego programu, ponieważ jeżeli użytkownik poda ujemną wartość boku kwadratu, na ekranie zostanie wyświetlony komunikat o źle wprowadzonej wartości.

```
#include <iostream>
using namespace std;
int main()
{
float a,pole;
cout<<"Podaj dlugosc boku kwadratu: ";
cin>>a;
```

```
    if (a>0)
    {
       pole=a*a;
       cout<<"Pole kwadratu wynosi: "<<pole;
    }
    else
    {
       cout<<"Dlugosc boku kwadratu musi byc liczba dodatnia.";
    }
return 0;
}
```

Konstrukcja warunków w instrukcji `if` może być prosta – składać się z jednego warunku, lub złożona – z kilka warunków, połączonych operatorami logicznymi.

PRZYKŁAD 31.4

Napisz program, który sprawdzi, czy dana liczba naturalna jest liczbą parzystą i niepodzielną przez 3, i wyświetli odpowiedni komunikat.
To zadanie można rozwiązać, jeżeli zastosuje się dwa warunki proste w dwóch zagnieżdżonych instrukcjach warunkowych:

```
#include <iostream>
using namespace std;
int liczba;
int main()
{
cout<<"Podaj liczbe:";
cin>>liczba;
if (liczba%2==0)  //warunek prosty spr. podzielnosc przez 2
    {
       if (liczba%3!=0) //warunek prosty spr. niepodzielnosc
       przez 3
       {
       cout<<"Liczba "<<liczba<<" jest parzysta i niepodzielna
       przez 3";
       }
       else
       {
       cout<<"Liczba "<<liczba<<" nie jest jednoczesnie parzysta
       i niepodzielna przez 3";
       }
```

```
    }
else
    {
    cout<<"Liczba "<<liczba<<" nie jest jednoczesnie parzysta
    i niepodzielna przez 3";
    }
    return 0;
}
```

Można również zastosować jeden warunek złożony:

```
#include <iostream>
using namespace std;
int liczba;
int main()
{
    cout<<"Podaj liczbe:";
    cin>>liczba;
    if (liczba%2==0 && liczba%3!=0)  //warunek zlozony
    {
    cout<<"Liczba  "<<liczba<<"  jest  parzysta  i  niepodzielna
    przez 3";
    }
    else
    {
    cout<<"Liczba "<<liczba<<" nie jest jednoczesnie parzysta i
    niepodzielna przez 3";
          }
    return 0;
}
```

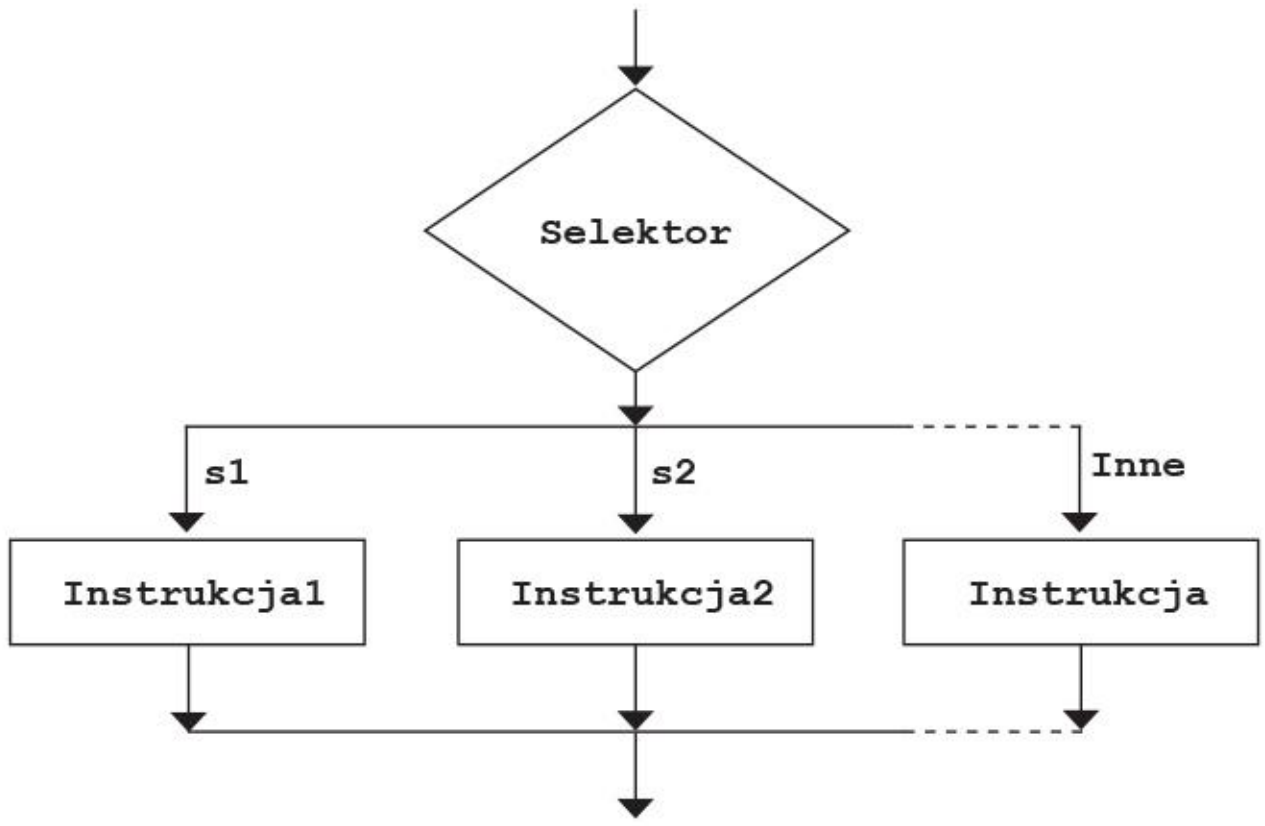
PRZYKŁAD 31.5

Rozważmy zadanie rozwiązania równania kwadratowego $ax^2 + bx = c = 0$. Program będzie miał trzy dane wejściowe: a, b, c. Na ich podstawie obliczy $\Delta = b^2 - 4ac$. Jeśli $\Delta > 0$, to równanie ma dwa pierwiastki. Jeśli $\Delta = 0$, to równanie ma jeden pierwiastek. Jeśli $\Delta > 0$, to równanie ma dwa pierwiastki. Jeśli $\Delta < 0$, to równanie nie ma rozwiązań.

```
#include <iostream>
#include<math.h>
using namespace std;
int main()
{
    float a,b,c,x1,x2,delta;
    cout<<"Rozwiazywanie rownania ax*x+bx+c=0"<<endl;
    cout<<"Podaj a: ";
    cin>>a;
    cout<<"Podaj b: ";
    cin>>b;
    cout<<"Podaj c: ";
    cin>>c;
    if (a==0)
    {
    cout<<"Rownanie jest liniowe i ma jedno rozwiazanie:
    "<<-c/b;
    }
else
{
      delta=b*b-4*a*c;
      if (delta<0)
      {
      cout<<"Rownanie nie ma rozwiazania";
      }
      else if (delta==0)
      {
      cout<<"Rownanie ma jedno rozwiazanie: "<<-b/a;
      }
      else
      {
         x1=(-b-sqrt(delta))/(2*a);
         x2=(-b+sqrt(delta))/(2*a);
         cout<<"Rownanie ma dwa rozwiazania"<<endl;
         cout<<"x1= "<<x1<<endl<<"x2= "<<x2;
      }
   }
   return 0;
}
```

Instrukcja SWITCH

Ta instrukcja służy do podejmowania wielowariantowych decyzji.



Rys. 31.3. Schemat blokowy instrukcji **SWITCH**

Selektor to zmienna, która może przyjmować różne wartości. Jeżeli wartość **Selektora** odpowiada wartości podanej w jednej z etykiet **case** (s1, s2, ...), to wtedy są wykonywane instrukcje (rozpoczynają się od tej etykiety). Wykonywanie kończy się po napotkaniu instrukcji **break**. Następuje wówczas wyjście z instrukcji **switch**.

Jeżeli wartość **Selektora** nie zgadza się z żadną z wartości podanych przy etykietach **case**, to są wykonywane instrukcje umieszczone po etykiecie **default** (może się ona znajdować w dowolnym miejscu instrukcji **switch**).

Instrukcja **break** powoduje natychmiastowe przerwanie wykonywania instrukcji. Jeżeli mamy do czynienia z kilkoma pętlami, zagnieżdżonymi jedna wewnątrz drugiej, to instrukcja **break** powoduje przerwanie tylko tej pętli, w której bezpośrednio tkwi.

Postać instrukcji:

```
switch (Selektor)
{
case s1:
     Instrukcja1;
     break;
case s2:
     Instrukcja2;
     break;
.........
default:
     Instrukcja;
  break;
  }
```

PRZYKŁAD 31.6

Napisz program, który:

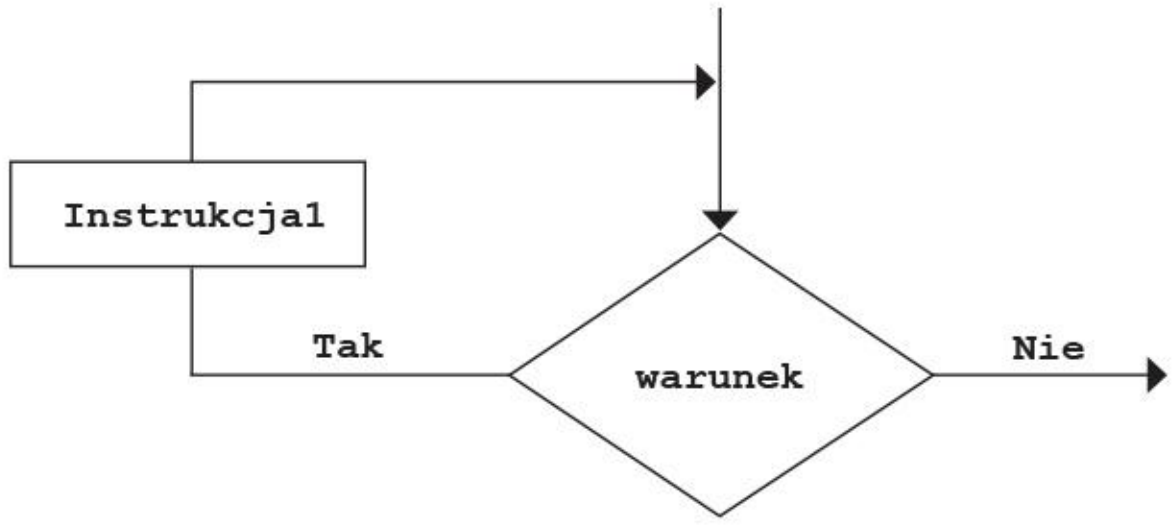
- wypisze na ekranie komunikat „Monday”, jeśli użytkownik wpisze liczbę 1;
- wypisze na ekranie komunikat „Tuesday”, jeśli użytkownik wpisze liczbę 2;
- wypisze na ekranie komunikat „Wednesday”, jeśli użytkownik wpisze liczbę 3;
- wypisze na ekranie komunikat „Thursday”, jeśli użytkownik wpisze liczbę 4;
- wypisze na ekranie komunikat „Friday”, jeśli użytkownik wpisze liczbę 5;
- wypisze na ekranie komunikat „Saturday”, jeśli użytkownik wpisze liczbę 6;
- wypisze na ekranie komunikat „Sunday”, jeśli użytkownik wpisze liczbę 7;
- wypisze na ekranie komunikat „Podałeś złą liczbę”, jeśli użytkownik wpisze każdą inną liczbę.

```
#include <iostream>
using namespace std;
int main()
{
    int dzien;
    cout<<"Podaj cyfre od 1 do 7"<<endl;
    cin>>dzien;
    switch(dzien)
    {
       case 1: cout<<"Monday";
               break;
       case 2: cout<<"Tuesday";
               break;
       case 3: cout<<"Wednesday";
               break;
       case 4: cout<<"Thursday";
               break;
       case 5: cout<<"Friday";
               break;
       case 6: cout<<"Saturday";
               break;
       case 7: cout<<"Sunday";
               break;
       default: cout<<"Podales zla liczbe";
               break;
    }
    return 0;
}
```

Pętla WHILE

Pętla to konstrukcja programowania, która umożliwia cykliczne wykonywanie ciągu instrukcji określoną liczbę razy, do momentu spełnienia określonych warunków lub w nieskończoność. W języku C++ można do tworzenia pętli zastosować instrukcje **WHILE**, **DO...WHILE** oraz **FOR**.

Pętlę **WHILE** stosuje się, gdy liczba powtórzeń zależy od warunku pętli. Pętli **FOR** używa się, gdy liczba powtórzeń danego bloku instrukcji jest znana. Natomiast pętlę **DO...WHILE** wykorzystuje się, gdy dany blok instrukcji ma zostać wykonany przynajmniej jeden raz.



Rys. 31.4. Schemat blokowy instrukcji **WHILE**

Tak jak w przypadku instrukcji warunkowej, najpierw oblicza się wartość **Warunku**. Jeśli wynik jest zerowy, to następuje wyjście z pętli. Jeśli wynik jest wartością niezerową, jest wykonywana **Instrukcja1**, a następnie – ponownie sprawdzana wartość **Warunku**. W instrukcji tej nie określamy liczby powtórzeń **Instrukcji1**. Wykonanie bloku instrukcji wewnątrz pętli odbywa się tylko wtedy, gdy wartość **Warunku** jest niezerowa.

Pętla **WHILE** jest uproszczoną wersją pętli **FOR**. Pętlę **FOR** używamy, gdy musimy określić liczbę powtórzeń danego bloku instrukcji. Natomiast pętli **WHILE** używamy, gdy liczba powtórzeń zależy od warunku pętli.

Postać instrukcji:

```
while (Warunek)
{
Instrukcja1;
}
```

PRZYKŁAD 31.7

Napisz program, który wypisze liczby od 1 do 25 w postaci kolumnowej.

```
#include <iostream>
using namespace std;
int main()
{
```

```
    int i=1;
    while (i<=25)
    {
       cout<<i<<endl;
       i++;
    }
    return 0;
}
```

Przed wejściem do pętli zmienna i ma wartość 1. Instrukcjami powtarzanymi są wyświetlenie wartości zmiennej i oraz inkrementacja tej zmiennej. Pętla działa do momentu, aż i osiągnie wartość 25.

PRZYKŁAD 31.8

Klasycznym zastosowaniem pętli **WHILE** jest ustalenie największego wspólnego dzielnika (NWD) dla dwóch liczb całkowitych.

```
#include <iostream>
using namespace std;
int main()
{
    int a,b;
    cout<<"Podaj a:";
    cin>>a;
    cout<<"Podaj b:";
    cin>>b;
    while(a!=b)
    {
       if (a<b) b-=a;
       else a-=b;
    }
    cout<<"Najwiekszy wspolny dzielnik wynosi "<<a<<endl;
    return 0;
}
```

Dopóki liczby a i b są różne, odejmujemy od większej liczby mniejszą. Gdy liczby będą sobie równe, zostanie wyświetlony największy wspólny dzielnik równy jednej z tych liczb.

Pętlę **WHILE** warto stosować w razie sprawdzania wprowadzanych danych, gdy np. chcemy, aby użytkownik podał wartość dodatnią. Z kolei dla wartości niedodatniej będzie ponownie wyświetlona prośba o podanie liczby.

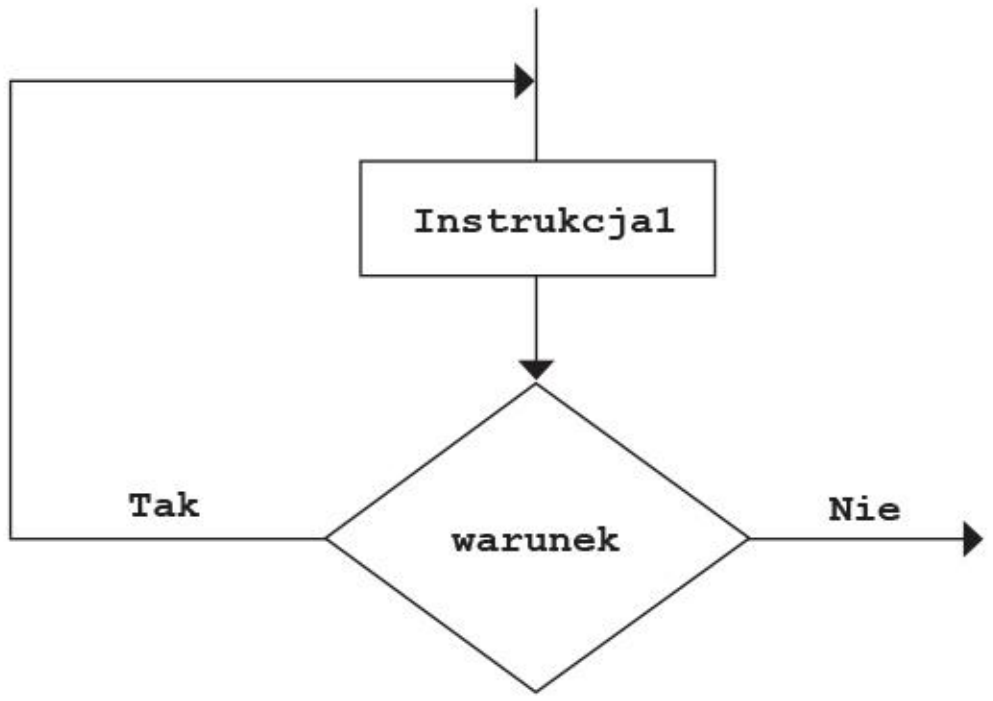
PRZYKŁAD 31.9

Wróćmy do programu obliczającego pole kwadratu. Za pomocą instrukcji **IF** mogliśmy tylko jednorazowo zareagować na błędnie wprowadzoną wartość. Pętla **WHILE** daje nam możliwość powtarzania instrukcji wczytywania danej do momentu podania wartości dodatniej.

```
#include <iostream>
using namespace std;
int main()
{
    float a,pole;
    cout<<"Podaj dlugosc boku kwadratu: ";
    cin>>a;
    while(a<=0)
    {
       cout<<"Podaj jeszcze raz dlugosc boku. Pamietaj a>0: ";
       cin>>a;
    }
    pole=a*a;
    cout<<"Pole kwadratu wynosi: "<<pole;
    return 0;
}
```

Pętla DO...WHILE

Słowa **DO...WHILE** oznaczają po angielsku „Rób... dopóki". Najpierw jest wykonywana **Instrukcja1**, a dopiero potem – sprawdzany warunek. Jeśli warunek został spełniony, to **Instrukcja1** jest wykonywana ponownie. Jeśli warunek nie jest spełniony, to **Instrukcja1** nie jest już więcej powtarzana.



Rys. 31.5. Schemat blokowy instrukcji **DO...WHILE**

Konstrukcja pętli **DO...WHILE** pozwala na wykonanie przynajmniej raz instrukcji wewnątrz pętli, pomimo niespełnionego warunku pętli. Natomiast instrukcja wewnątrz pętli **WHILE** nie zostanie wykonana ani razu, gdy warunek pętli jest zerowy.

Postać pętli:

```
do
{
Instrukcja1;
}
while (warunek);
```

PRZYKŁAD 31.10

Napisz program, który wypisze ciąg malejących liczb całkowitych od 25 do 1 w postaci kolumnowej.

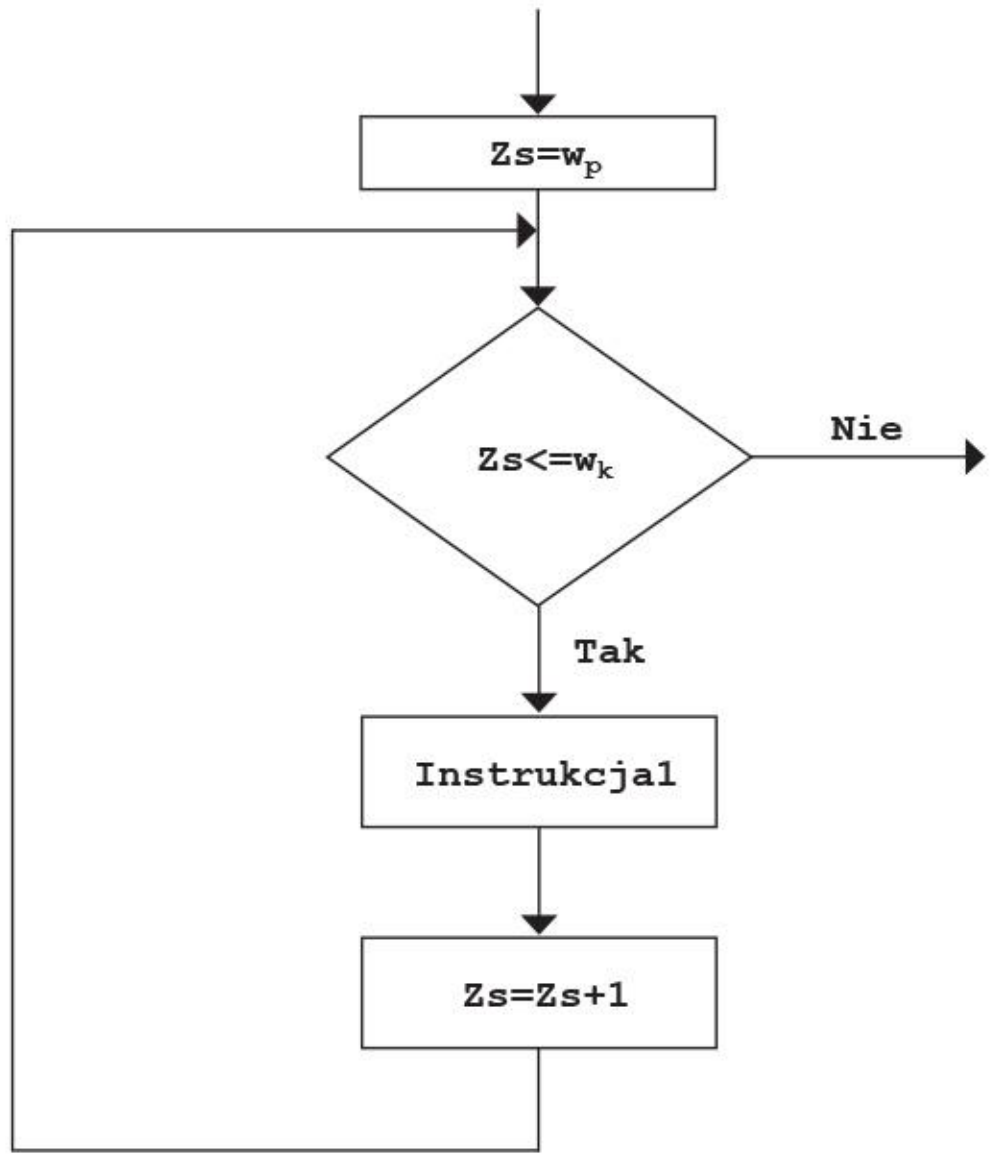
```
#include<iostream>
using namespace std;
int main()
{
    int i=25;
    do
    {
       cout<<i<<endl;
       i--;
    }while (i>=1);
    return 0;
}
```

To samo zadanie można rozwiązać za pomocą pętli **WHILE**.

```
#include<iostream>
using namespace std;
int main()
{
   int i=25;
   while (i>=1)
   {
      cout<<i<<endl;
      i--;
   }
   return 0;
}
```

Pętla FOR

Do wykonywania pętli o określonej liczbie powtórzeń stosuje się instrukcję **FOR**. Ma ona trzy argumenty. Pierwszy z nich **Zs=w_p** określa wartość początkową zmiennej sterującej pętli (**Zs** – zmienna sterująca). Warunek **Zs<w_k** określa wartość końcową zmiennej sterującej, natomiast instrukcja **Zs++** – sposób zmiany wartości zmiennej sterującej (w tym przypadku inkrementacja o 1).



Rys. 31.6. Schemat blokowy instrukcji **FOR**

Instrukcja **FOR** ma następującą postać:

```
for (Zs=wp; Zs<wk; Zs++)
{
Instrukcja1;
}
```

PRZYKŁAD 31.11

Napisz program, który pięć razy wypisze na ekranie komunikat: „Uczę się programować w języku C++”.

```
#include<iostream>
using namespace std;

int main()
```

```
{
    int i;
    for(i=1;i<=5;i++)
    {
       cout<<"Programowanie w C++"<<endl;
    }
    return 0;
}
```

PRZYKŁAD 31.12

Zadanie z przykładu 31.10 można rozwiązać także za pomocą pętli **FOR**.

```
#include<iostream>
using namespace std;

int main()
{
    int i;
    for(i=25;i>=1;i--)
    {
       cout<<i<<endl;
    }
    return 0;
}
```

W rozdziale o algorytmach przedstawiono algorytm iteracyjny obliczający silnię liczby podanej przez użytkownika. Pętla **FOR** jest stworzona do operacji iteracyjnych z określoną liczbą wykonywanych obliczeń. Po implementacji program powinien mieć następującą postać:

PRZYKŁAD 31.13

```
#include<iostream>
using namespace std;

int main()
{
    int i,n,silnia;
    cout<<"Podaj n:";
    cin>>n;
```

```
    silnia=1;
    for(i=1;i<=n;i++)
    {
       silnia=silnia*i;
    }
    cout<<"Silnia wynosi: "<<silnia;
    return 0;
}
```

Pętle zagnieżdżone

Zagnieżdżenie pętli polega na uruchomieniu jednej pętli wewnątrz drugiej. Podczas jednej iteracji pętli zewnętrznej zostanie wykonany cały przebieg pętli wewnętrznej.

PRZYKŁAD 31.14

Załóżmy, że chcemy narysować kwadrat o boku zbudowanym z dziesięciu znaków „x".

```
#include<iostream>
using namespace std;

int main()
{
int i,j;
    for(i=1;i<=10;i++)
       {
       for(j=1;j<=10;j++)
          {
          cout<<"x ";
          }
       cout<<endl;
       }
return 0;
}
```

Pętla wewnętrzna ma za zadanie wyświetlić dziesięć znaków „x", natomiast pętla wewnętrzna powoduje, że po każdym pełnym obiegu iteracji wewnętrznej jest przejście do nowej. Ostatecznie otrzymamy poniższy rysunek (rys. 31.7).

```
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x
x x x x x x x x x x

Process returned 0 (0x0)   execution time : 0.172 s
Press any key to continue.
```

Rys. 31.7. Zastosowanie zagnieżdżenia pętli **FOR**

PRZYKŁAD 31.15

Aby z kwadratu zrobić trójkąt, należy dopisać drugą pętlę wewnętrzną, która dostawi spacje w kolejnych wierszach rysowanego trójkąta.

```
#include<iostream>
using namespace std;

int main()
{
int i,j,k;
for(i=1;i<=10;i++)
{
//wstawianie spacji
for(k=1;k<=10-i;k++) cout<<" ";
//wstawianie #
for(j=1;j<=i;j++)    cout<<"# ";
cout<<endl;
      }
return 0;
}
```

Efektem programu będzie poniższy rysunek (rys. 31.8).

```
         #
        # #
       # # #
      # # # #
     # # # # #
    # # # # # #
   # # # # # # #
  # # # # # # # #
 # # # # # # # # #
# # # # # # # # # #

Process returned 0 (0x0)   execution time : 0.114 s
Press any key to continue.
```

Rys. 31.8. Przykład zagnieżdżenia kilku pętli

Pętle można zagnieżdżać w dowolny sposób i nie dotyczy to wyłącznie instrukcji **FOR**. Każda z pozostałych pętli, czyli **WHILE**, **DO...WHILE** mogą być w sobie zagnieżdżone.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Omów zastosowanie instrukcji warunkowej.
2. Do czego służy instrukcja **SWITCH**?
3. Opisz schemat blokowy instrukcji **SWITCH**.
4. Czym różni się instrukcja **IF** od instrukcji **SWITCH**?
5. Omów postać pętli **FOR**.
6. Opisz różnice między pętlami budowanymi za pomocą instrukcji: **WHILE**, **DO...WHILE** oraz **FOR**.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz program, który wykona mnożenie dwóch wcześniej podanych przez ciebie liczb, przypisze wynik pewnej zmiennej, a następnie sprawdzi, czy wynik mnożenia jest większy, równy, czy mniejszy od 100.
2. Napisz program, który wykona dodawanie dwóch wpisanych z klawiatury liczb, ale tylko wtedy, gdy nie będą sobie równe. W przeciwnym razie wykona dzielenie pierwszej przez drugą (pod warunkiem że liczby nie są zerami).
3. Napisz program, który pobierze wartości dla 3 zmiennych i wykona:
   - mnożenie liczby pierwszej oraz drugiej, gdy liczba pierwsza jest większa od trzeciej i liczba druga jest większa od pierwszej;
   - dzielenie drugiej przez trzecią, gdy liczba druga jest mniejsza od trzeciej albo mniejsza od pierwszej;
   - dodawanie wszystkich trzech liczb w przypadku, gdy liczba trzecia jest większa od pierwszej i liczba druga nie jest równa 5 lub liczba druga jest większa od trzeciej oraz liczba pierwsza nie jest równa 0.
4. Napisz program, który poprosi użytkownika o podanie czterech liczb rzeczywistych: x, y, z, t, oraz obliczy wartość wyrażenia $k = \frac{x}{y-2} + \sqrt{z+1} - 7t$. Wynik wyświetl w konsoli.

   Wskazówka: Zastosuj instrukcję warunkową w celu sprawdzenia poprawności wprowadzanych wartości: x, y, z, t.
5. Napisz program, który wczyta trzy liczby: a, b, c, oraz sprawdzi, czy można z nich zbudować trójkąt. Jeśli tak, to sprawdź, czy jest to trójkąt równoboczny, czy – prostokątny. Jeśli to trójkąt równoboczny, oblicz jego obwód i pole. Wyniki wyświetl na ekranie monitora. Jeśli jest to trójkąt prostokątny, oblicz jego obwód i wyświetl jego wartość. Jeśli liczby a, b, c nie tworzą trójkąta, wyświetl komunikat: „Z tych boków nie można zbudować trójkąta".

   Uwaga: Boki trójkąta są liczbami dodatnimi.

6. Napisz program, który poprosi użytkownika o podanie zmiennej znakowej k. Jeśli k będzie literą „p”, poproś o podanie długości boków prostokąta i sprawdź, czy wprowadzone liczbami dodatnimi. Jeśli tak, to oblicz pole i obwód prostokąta. Jeśli k będzie literą „k”, to poproś o podanie długości boku kwadratu. Sprawdź, czy wprowadzona wartość jest dodatnia. Jeśli tak, to oblicz długość przekątnej kwadratu. Dla każdej innej litery wprowadzonej przez użytkownika wyświetl komunikat: „Koniec programu”.
7. Napisz program, który wypisze w postaci kolumnowej liczby całkowite od 1 do 25 w porządku rosnącym i porządku malejącym.
8. Napisz program obliczający obwód trójkąta, którego długości boków wprowadza użytkownik. W przypadku podania liczb ujemnych lub równych zero użytkownik ma zostać poproszony o ponowne podanie długości boków.
9. Napisz program liczący pierwiastki równania kwadratowego, tak by pozwalał na kolejne obliczenia, aż do wpisania a=0.
10. Napisz program, który wczytuje liczby z klawiatury aż do momentu, gdy użytkownik poda liczbę z przedziału <10,15>.
11. Napisz program, który dla podanej przez użytkownika liczby całkowitej obliczy sumę cyfr tej liczby.
12. Napisz program, który obliczy sumę n liczb naturalnych kończących się cyfrą 5.
13. Napisz program, który obliczy sumę 10 dowolnych liczb podanych przez użytkownika.
14. Napisz program, który spośród 8 liczb podanych przez użytkownika wyszuka te, które są podzielne przez 3, i obliczy ich sumę.
15. Napisz program, który spośród liczb naturalnych mniejszych od 100 wyszuka i wypisze liczby podzielne przez 5 i jednocześnie niepodzielne przez 3.

32 Funkcje

ZAGADNIENIA

- Definicja funkcji
- Prototyp funkcji
- Wywoływanie funkcji
- Przekazywanie argumentu przez wartość
- Przekazywanie argumentu przez referencję
- Przeładowanie funkcji
- Argumenty domniemane

Budowa funkcji

Jedną z najciekawszych cech współczesnych języków programowania jest możliwość posługiwania się podprogramami.

Pewne instrukcje są wykonywane w danym programie wielokrotnie dla tych samych lub innych zmiennych. Z tego względu wygodnie jest zgrupować powtarzające się obliczenia pod jedną nazwę. Tworzymy w ten sposób funkcję, czyli podprogram wykonujący pewne zadanie na potrzeby programu głównego. Jest to fragment programu, któremu nadano nazwę i który możemy wykonać przez podanie jego nazwy oraz ewentualnych argumentów (o ile istnieją).

Każda funkcja musi cechować się trzema własnościami:

- mieć nazwę;
- zwracać wartość lub nie, jeśli tego nie chcemy;
- mieć dowolną liczbę argumentów.

Funkcja może pobierać pewne argumenty z programu głównego, wykorzystywane przez nią w celu wykonania określonych zadań.

Jeżeli funkcja zwraca jakąś wartość, należy określić typ zwracanej wartości. Sposób wywoływania funkcji zależy od jej typu. Zanim jednak ją wywołamy, musimy wiedzieć, czy pobiera argumenty z programu głównego i czy zwraca jakąś wartość. Każda funkcja, która zwraca wartość, musi być wywoływana przy jednoczesnym przypisaniu danej wartości pewnej zmiennej globalnej.

Funkcja, która nie zwraca wartości

Weźmy pod uwagę następującą funkcję:

```
void wypisz()
{
//ciało funkcji
}
```

Nazwą funkcji jest **wypisz**. Nie przyjmuje ona żadnych argumentów, ponieważ wewnątrz zaokrąglonych nawiasów jest pusto. Między klamrami umieszczamy kod, który ma zostać wykonany, gdy zostanie wywołana funkcja. Ten kod nazywa się **ciałem funkcji**.

Słowo kluczowe **void** przed nazwą funkcji informuje nas, że funkcja nie zwraca żadnych danych, które można byłoby przekazać innej funkcji bądź instrukcji. Funkcja tego typu może wykonywać pewne czynności, ale nie przekazuje informacji zwrotnej. Funkcja **wypisz** może np. wyświetlać komunikat: „Witaj w programie!". Oprócz wyświetlanej informacji żadne dane nie będą dalej przekazywane.

PRZYKŁAD 32.1

```
#include<iostream>
using namespace std;
    void wypisz()
    {
       cout<<"Witaj w programie!";
    }
    int main()
    {
       wypisz(); //wywolanie funkcji wypisz
       return 0;
    }
```

Definicję funkcji **wypisz** umieszczamy przed funkcją główną **main**. Wszystkie definiowane funkcje powinny być umieszczane przed funkcją główną lub w oddzielnym pliku, który zostanie dodany za pomocą dyrektywy **#include**.

Działanie każdego programu rozpoczyna się od uruchomienia funkcji **main**. To w niej wywołujemy zdefiniowane przez nas funkcje. Funkcja **wypisz** jest funkcją niezwracającą żadnej wartości oraz niemającą argumentów. Aby ją wywołać, w funkcji głównej należy wpisać jej nazwę, a za nią – nawiasy okrągłe. Efektem będzie wyświetlenie komunikatu „Witaj w programie!" na ekranie monitora.

Gdyby jednak funkcja **wypisz** miała za zadanie np. obliczyć resztę z dzielenia podanej liczby przez 2, to typ **void** nie będzie poprawny. Informacją zwrotną będzie liczba będąca resztą z dzielenia, czyli wartość typu **int**.

Funkcja mająca argumenty

Wartością zwrotną funkcji jest wyliczona wartość, która jest przekazywana do dalszych działań. Typ tej danej wypisujemy przed definicją naszej funkcji i jest ona zwracana za pomocą słowa kluczowego **return**.

```
typ_zwracanej_wartosci nazwa_funkcji(lista_argumentow)
{
return zwracana_wartosc;
}
```

Argumenty określają, jakiego typu zmienne należy przekazać do funkcji podczas jej wywoływania.

Są to dane, na których podstawie są wykonywane instrukcje wewnątrz funkcji.

PRZYKŁAD 32.2

```
void spr_parzysta(int a)
{
    if (a%2==0)
    {
        cout<<"Liczba "<<a<<" jest parzysta"<<endl;
    }
    else
    {
        cout<<"Liczba "<<a<<" jest nieparzysta"<<endl;
    }
}
```

Została zadeklarowana funkcja **sprawdz**, która ma jeden argument typu całkowitego oraz nie zwraca żadnej wartości, czyli zwraca typ **void**. Funkcja **sprawdz** weryfikuje, czy podana liczba jest parzysta, czy nie, i wyświetla odpowiedni komunikat.

Funkcja może mieć dowolną liczbę argumentów i ich typów, a także zwracać wartość. Przykładem jest funkcja o nazwie **wyrazenie** mająca trzy argumenty typu **float** i zwracająca wartość typu **float**.

PRZYKŁAD 32.3

```
float wyrazenie(float x, float y, float z)
{
    float w=x+2*y-sqrt(z);
    return w;
}
```

Zarówno argumenty funkcji, jak i zmienne definiowane w jej ciele mają zasięg lokalny, czyli są widoczne tylko przez tę funkcję.

Należy pamiętać, że funkcja, która zwraca wartość, musi zwracać ją zawsze. Jeżeli funkcja dojdzie do końca bloku funkcji i nie napotka słowa kluczowego **return**, zachowanie wyjścia z funkcji jest niezdefiniowane. Oznacza to, że aplikacja może zakończyć pracę przez wystąpienie błędu krytycznego aplikacji.

Wywoływanie funkcji i przekazywanie argumentów do funkcji

Jak już wcześniej wspomniano, aby wywołać w programie zdefiniowaną wcześniej funkcję, należy po prostu napisać jej nazwę w bloku instrukcji funkcji głównej. Trzeba pamiętać, że wywoływana funkcja musi być wcześniej zadeklarowana.

PRZYKŁAD 32.4

Rozpatrzmy program, który będzie dodawał dwie liczby rzeczywiste. To działanie będzie wykonywała funkcja **suma**.

```
#include<iostream>
using namespace std;
//definicja funkcji suma; zmienne a i b są zmiennymi lokalnymi
 float suma(float a, float b)
 {
    return a+b;
 }

 int main()
 {
    float x,y;   //zmienne globalne
    cout<<"Podaj x:";
    cin>>x;
    cout<<"Podaj y:";
    cin>>y;
    //wywolanie funkcji
    cout<<"Suma liczb "<<x<<" i "<<y<<" wynosi: "<<suma(x,y);
    return 0;
 }
```

Definicja funkcji jest umieszczona przed funkcją główną **main**. Ma ona dwa argumenty typu rzeczywistego i zwraca wartość typu rzeczywistego. Jeśli funkcja wykonuje tylko jedno działanie, to można je zapisać od razu w instrukcji **return**.

Funkcję **suma** wywołujemy przez instrukcję:

```
suma(x,y);
```

Zmienne **x** oraz **y** są zmiennymi globalnymi i są typu rzeczywistego, tak jak zmienne **a** oraz **b** będące zmiennymi lokalnymi, czyli rozpoznawalnymi tylko w obrębie funkcji **suma**. Zmienne **a**, **b** służą jedynie do zapisania wyrażenia, które funkcja ma wykonać, natomiast zmienne **x** i **y** są argumentami, na których funkcja pracuje.

Jest to przykład **przekazywania argumentów przez wartość**. W momencie przekazywania zmiennych jako argumentów funkcja tworzy ich kopie w pamięci. Oznacza to, że wszelkie zmiany na wartościach zmiennych nie będą widoczne w miejscu, w którym zostały przekazane do funkcji. Ponadto nazwy tych zmiennych mogą, ale nie muszą, być takie same jak nazwy argumentów funkcji.

Kopiowanie danych przekazywanych do funkcji bardzo często nie jest pożądanym efektem. Dane przekazywane do funkcji mogą przecież zajmować dużo miejsca w pamięci i wtedy niepotrzebnie będziemy tracili moc obliczeniową komputera na tworzenie ich kopii. Aby zapobiec kopiowaniu danych, wystarczy użyć symbolu **&** w odpowiednim miejscu argumentu, czyli zastosować **przekazywanie argumentów przez referencję** („przezwisko"). Symbol **&** umieszcza się za typem zmiennej i przed jej nazwą, np.:

```
int &zmienna;
```

Jeśli chcemy przekazywać argumenty przez referencję, stosujemy następujący zapis:

```
void funkcja (typ_arg1 &arg1 …, typ_argn &argn)
```

Dane przekazane przez referencję nie są kopiowane. Będziemy pracowali na oryginalnych danych, co w praktyce oznacza, że modyfikacja wartości zmiennej zmodyfikuje nam tę zmienną, która została przekazana do funkcji przez argument.

PRZYKŁAD 32.5

```
#include<iostream>
using namespace std;
void zamien(int &a, int &b)
{   int x;
    x=a;
    a=b;
    b=x;
}

    int main()
    {
       int x=6;
       int y=12;
       cout<<"Przed zamiana"<<endl;
       cout<<"x= "<<x<<" y= "<<y<<endl;
       zamien(x,y);
```

```
        cout<<"Po zamianie"<<endl;
        cout<<"x= "<<x<<" y= "<<y<<endl;
        return 0;
}
```

Jeśli definicję funkcji umieścimy po funkcji głównej **main**, to nie będzie ona widoczna dla kompilatora, chyba że przed funkcją **main** umieścimy prototyp naszej funkcji.

Prototyp (deklaracja) funkcji to model, pod którym funkcja będzie rozpoznawalna w programie. W przeciwieństwie do definicji prototyp jest zakończony średnikiem. Definicja funkcji musi natomiast mieć nagłówek, szkielet (zawiera wszystkie instrukcje wykonywane w obrębie danej funkcji) oraz instrukcję powrotu (część kończącą każdą funkcję). Nagłówek definicji funkcji ma budowę identyczną z prototypem, lecz nie jest zakończony średnikiem. Jeżeli funkcja ma oddzielną deklarację (prototyp) i definicję, nie jest konieczne nazywanie argumentów funkcji podczas deklaracji. Wystarczą jedynie typy tych argumentów. Często używa się tej właściwości, żeby nie zaciemniać deklaracji funkcji niepotrzebnymi nazwami argumentów.

PRZYKŁAD 32.6

```
#include <iostream>
using namespace std;
void parzysta(int a);       //prototyp (deklaracja) funkcji
int main()
{
    int liczba;
    cout<<"Podaj dowolną liczbe całkowitą: ";
    cin>>liczba;
    parzysta(liczba);
    return 0;
}
void parzysta(int a)        //definicja funkcji
{
    if (a%2==0)
       cout<<"Liczba jest parzysta";
    else
       cout<<"Liczba jest nieparzysta";
}
```

Program może zawierać dowolną liczbę funkcji, których wyniki będą przekazywane do innych funkcji. Rozważmy rozwiązywanie równania kwadratowego.

PRZYKŁAD 32.7

```
#include <iostream>
#include<math.h>
using namespace std;

//funkcja delta typu float, pracujaca na argumentach float
float delta(float a, float b, float c)
{
    return b*b-4*a*c;
}

//funkcja dwa_rozw typu void, pracujaca na argumentach float
void dwa_rozw(float a, float b, float d)
{
    float x1=(-b-sqrt(d))/(2*a;
    float x2=(-b+sqrt(d))/(2*a);
    cout<<"x1= "<<x1<<endl<<"x2= "<<x2;
}

//funkcja jedno_rozw typu float, pracujaca na argumentach float
float jedno_rozw(float a,float b)
{
    return -b/a;
}

int main()
{
    float a,b,c;
    cout<<"Rozwiazywanie rownania ax*x+bx+c=0"<<endl;
    cout<<"Podaj a: ";
    cin>>a;
    cout<<"Podaj b: ";
    cin>>b;
    cout<<"Podaj c: ";
    cin>>c;
if (a==0)
{
    cout<<"Rownanie jest liniowe i ma jedno rozwiazanie:
    "<<jedno_rozw(b,c);
    }
    else
```

```
{
float d=delta(a,b,c);          //wywolanie funkcji delta
if (d<0)
{
cout<<"Rownanie nie ma rozwiazania";
}
else
{
   if (d==0)
   {
   cout<<"Rownanie ma jedno rozwiazanie: "<<jedno_
   rozw(a,b);                  //wywolanie funkcji jedno_rozw
   }
   else
   {
   cout<<"Rownanie ma dwa rozwiazania"<<endl;
   dwa_rozw(a,b,d);//wywolanie funkcji dwa_rozw
   }
}
}
return 0;
}
```

Funkcja ma trzy funkcje: delta, jedno_rozw, dwa_rozw. Pierwszą wywoływaną funkcją jest delta, która oblicza wyróżnik trójmianu kwadratowego na podstawie liczb podanych przez użytkownika. Jeśli wynik funkcji jest liczbą dodatnią, zostaje wywołana funkcja dwa_rozw, której jednym z argumentów jest obliczona wartość przez funkcję delta. Jeśli wynik funkcji delta jest równy zero, zostaje powołana do działania funkcja jedno_rozw.

Przeładowanie funkcji

Przeładowanie nazwy funkcji polega na tym, że w danym programie występuje więcej niż jedna funkcja o takiej samej nazwie. To, która z nich zostanie zastosowana w kodzie, zależy od argumentów, z którymi zostanie ona wywołana.

Rozważmy funkcję do obliczania pola powierzchni wybranych figur. Inaczej będzie wyglądała funkcja licząca pole koła, a inaczej – trójkąta. Pisanie kilku funkcji o różnych nazwach dla poszczególnych figur jest nieco pracochłonne. Zamiast nazywać funkcje: **`pole_kola`**, **`pole_trapezu`**, **`pole_trojkata`** itd., wygodniej jest napisać **`pole`**. Język C++ dopuszcza taką możliwość. Jeżeli w jednym zakresie istnieje kilka funkcji o tych samych nazwach, to należy je jakoś odróżniać. Wystarczy zadbać o to, aby typy argumentów były różne w każdej z funkcji lub żeby była inna liczba argumentów. Wystarczy nawet zamienić kolejność i błędu już nie będzie.

PRZYKŁAD 32.8

```
float pole(float promien);
float pole(float dl, float szer);
```

Funkcje wyglądają podobnie. Mają jednakowe nazwy, a mimo to kompilator nie wyświetla żadnego błędu. Na tym właśnie polega przeładowanie nazwy funkcji. Takich funkcji może być więcej. Wystarczy zmienić typy argumentów, ich liczbę lub kolejność.

UWAGA

Podczas przeładowania nazwy funkcji ważna jest jedynie odmienność argumentów. Typ zwracany przez funkcję nie jest brany pod uwagę. W trakcie kompilacji takie przeładowanie będzie błędem.

PRZYKŁAD 32.9

```
#include <iostream>
#include<math.h>
using namespace std;

//funkcja obliczająca pole koła, ma jeden argument float
float pole(float promien)
{
return M_PI*promien*promien;
}

//funkcja obliczająca pole prostokata, ma dwa argumenty float
float pole(float a, float b)
{
return a*b;
}
int main()
{
float pr,x,y;
cout<<"Podaj dlugosc promienia: ";
cin>>pr;
cout<<"Pole kola o promieniu "<<pr<<" wynosi: "<<pole(pr)<<endl;
cout<<"Pole prostokata o bokach 3 i 4.5 wynosi: "<<pole(3,4.5);
return 0;
}
```

Zostały zdefiniowane dwie funkcje różniące się liczbą argumentów. Pierwsza z nich ma jeden argument typu zmiennoprzecinkowego, a druga pracuje na dwóch argumentach typu zmiennoprzecinkowego.

Argumenty domniemane

Jeśli podczas wywoływania funkcji podamy złą liczbę argumentów, to kompilator wyświetli błąd. W języku C++ można w funkcjach umieszczać argumenty domniemane.

Argument domniemany to argument, który może zostać podany w wywołaniu funkcji lub nie.

Te argumenty określa się w deklaracji funkcji, a nie – w jej definicji. Jeśli jednak funkcja jest napisana w programie powyżej jakiegokolwiek jej wywołania, to oddzielna deklaracja tej funkcji nie jest potrzebna. Sama definicja funkcji jest wtedy także jej pierwszą deklaracją występującą w tym programie.

Rozpatrzmy funkcję służącą do wyświetlenia informacji o temperaturze. Temperatura jest zazwyczaj podawana w stopniach Celsjusza. Czasem jednak chcemy mieć inną skalę. Rodzaj jednostki można określić w wywołaniu funkcji za pomocą określonej liczby całkowitej. Wartość 0 to stopnie Celsjusza, 1 – kelwiny, a 2 to stopnie Fahrenheita. Wygodnie byłoby mieć funkcję, która w domyśle podaje skalę w stopniach Celsjusza, a jedynie na specjalne życzenie – w stopniach Fahrenheita czy w kelwinach. Jest to możliwe za pomocą argumentów domniemanych. Jeżeli nie podamy niczego, zostanie wykorzystana skala w stopniach Celsjusza, w przeciwnym razie – w stopniach Fahrenheita.

PRZYKŁAD 32.10

```
#include <iostream>
using namespace std;

float temperatura(float stopnie, int skala = 0) ;
//skala jako argument domniemany
{
if (skala == 0) return stopnie;
if (skala == 1) return stopnie+273;
if (skala == 2) return stopnie*1.8+32;
}
int main()
{
float temp;
//wywołanie funkcji z argumentem skala=0
cout<<"Dzisiaj jest: "<<temperatura(26)<<" C"<<endl;
cout<<"Dzisiaj jest: "<<temperatura(26,1)<<" K"<<endl;
cout<<"Dzisiaj jest: "<<temperatura(26,2)<<" F"<<endl;
  return 0;
}
```

Taką funkcję można wywołać dwojako. Pierwszy sposób to normalne wywołanie z określeniem konkretnej wartości argumentu **skala**:

```
int t = temperatura(26,0);
```

Drugi sposób jest nieco krótszy, ponieważ pomijamy wartość drugiego argumentu:

```
int t = temperatura(26);
```

W obu powyższych zapisach zostanie po prostu wypisana temperatura 26 °C. Jeśli chcemy wyświetlić tę temperaturę w kelwinach, należy zastosować zapis:

```
int t = temperatura(26,1);
```

UWAGA

Wszelkie argumenty domniemane **trzeba** umieszczać na końcu. Nie może być sytuacji, że pierwszy argument będzie domniemany, a pozostałe – nie.

PRZYKŁAD 32.11

Rozważmy zapis:

```
void wypisz(int arg1=0, int arg2, int arg3=1);
```

Jeżeli podczas wywołania funkcji zostaną podane dwa argumenty, to może dojść do dwóch sytuacji:

a) funkcja zostanie wywołana z pierwszym argumentem domniemanym równym 0, argumenty drugi i trzeci to wartości podane w wywołaniu;
b) pierwszy i drugi argument określimy w wywołaniu, a trzeci, jako że jest domniemany, będzie równy 1.

Dlatego taka funkcja nie może zostać wywołana. Kompilator wyświetli komunikat o błędzie.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz program zawierający funkcję sprawdzającą, czy z trzech liczb podanych przez użytkownika można zbudować trójkąt prostokątny. Funkcja ma być typu **void** i mieć trzy argumenty typu **float**.
2. Napisz program zawierający cztery funkcje wykonujące cztery podstawowe działania arytmetyczne. Każda z funkcji ma zwracać wartość typu rzeczywistego i pobierać dwa argumenty typu **float**.
3. Napisz program, który pobierze z klawiatury trzy wartości, przypisze je pewnym zmiennym, które przekaże jako argumenty pewnej funkcji. Jeśli wartość pierwszej zmiennej pomnożonej przez wartość drugiej zmiennej jest większa od 100-krotności trzeciej zmiennej, to wynikiem funkcji będzie podwojona wartość pierwszej zmiennej, w przeciwnym razie wynikiem funkcji będzie wartość średnia arytmetyczna trzech zmiennych.

4. Napisz program zawierający trzy funkcje:
   - **numer_cwiartki** – funkcja mająca dwa parametry typu całkowitego i zwracająca wartość typu całkowitego;
   - **parzyste** – funkcja mająca dwa parametry typu całkowitego i niezwracająca żadnej wartości;
   - **wyrazenie** – funkcja mająca dwa parametry typu całkowitego i zwracająca wartość typu rzeczywistego (zmiennoprzecinkowego).

   Działanie programu

   Program prosi użytkownika o podanie dwóch współrzędnych punktu, czyli x oraz y. Następnie ma zostać wywołana funkcja **`numer_cwiartki`** z argumentami x i y, która zwraca następujące wartości:
   - 1 – gdy punkt należy do pierwszej ćwiartki układu współrzędnych (czyli obie współrzędne są dodatnie);
   - 2 – gdy punkt należy do drugiej ćwiartki układu współrzędnych (czyli pierwsza współrzędna jest ujemna, a druga dodatnia);
   - 3 – gdy punkt należy do trzeciej ćwiartki układu współrzędnych (czyli obie współrzędne są ujemne);
   - 4 – gdy punkt należy do czwartej ćwiartki układu współrzędnych (czyli pierwsza współrzędna jest dodatnia, a druga ujemna);
   - 0 – gdy punkt leży na jednej z osi układu (czyli jedna ze współrzędnych jest równa 0).

   Jeśli funkcja **`numer_cwiartki`** zwróci wartość 1, to ma zostać wywołana funkcja **`parzyste`**, która wypisuje liczby parzyste z zakresu od x do y. Jeśli x<y, wypisz liczby w postaci ciągu rosnącego, w przeciwnym razie – w postaci ciągu malejącego.
   Jeśli funkcja **`numer_cwiartki`** zwróci wartość 2 lub 4, to ma zostać wywołana funkcja **`wyrazenie`**, która oblicza wartość wyrażenia

$$\frac{5}{x-y+2}$$

   Wyświetl wynik zwracany przez funkcję **`wyrazenie`**.
   Jeśli funkcja **`numer_cwiartki`** zwróci wartość 0 lub 3, to wyświetl komunikat „Koniec programu”.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Podaj definicję funkcji.
2. Czym różni się prototyp od definicji funkcji?
3. Podaj sposoby przekazywania argumentów do funkcji.
4. Co oznacza, że funkcja jest typu **`void`**?
5. Co to są argumenty domniemane?
6. Na czym polega przeładowanie funkcji?

33 Tablice

ZAGADNIENIA

- Definicja tablicy
- Postać definicji tablicy jednowymiarowej i wielowymiarowej
- Nadawanie wartości elementom tablicy
- Przekazywanie tablicy do funkcji

Tablica to zmienna, która może przechowywać więcej niż jedną wartość. Wszystkie elementy tablicy muszą być tego samego typu. Czas dostępu do każdego z elementów tablicy nie zależy od położenia elementu w tej strukturze. Podczas definiowania tablicy należy określić:

- nazwę tablicy;
- typ wartości elementów, które będzie przechowywać tablica;
- liczbę elementów (wymiar tablicy).

Tablice jednowymiarowe

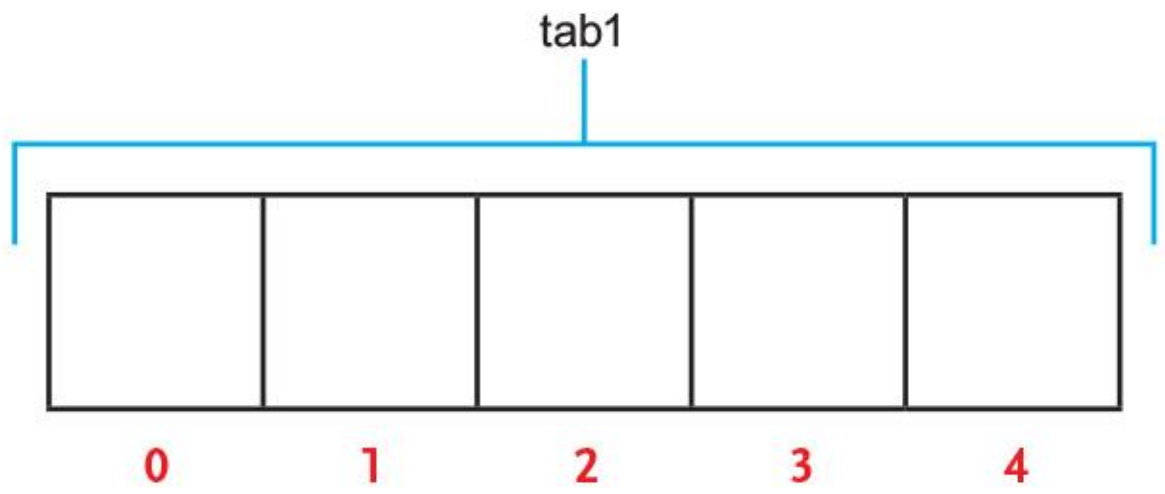
Definicja tablicy jednowymiarowej ma następującą postać:

```
typ nazwa_tablicy[ilość_elementów];
```

PRZYKŁAD 33.1

```
int tab1[5];
```

Jest to definicja pięcioelementowej tablicy o nazwie **tab1**, której elementy są typu całkowitego.



Rys. 33.1. Postać jednowymiarowej tablicy

UWAGA

Elementy w tablicy są numerowane od **zera**. W pięcioelementowej tablicy pierwszym elementem jest element zerowy, natomiast ostatnim – element czwarty.

Aby wpisać wartość do komórki tablicy, należy podać nazwę tej tablicy, a następnie – w nawiasach kwadratowych wpisać indeks komórki.

PRZYKŁAD 33.2

Aby wpisać kolejne liczby naturalne do tablicy **tab1**, możemy zastosować zapis:

```
tab1[0]=1;
tab1[1]=2;
tab1[2]=3;
tab1[3]=4;
tab1[4]=5;
```

Powyższy sposób przypisania wartości poszczególnym elementom tablicy jest poprawny, ale nie okazuje się praktyczny. Zwykle elementy tablicy czyta się lub wypisuje za pomocą pętli **FOR**.

PRZYKŁAD 33.3

```
#include<iostream>
using namespace std;
int tab1[5];
int i;
int main()
{
    for (i=0;i<5;i++)
    {
       tab1[i]=i+1;
       cout<<"tab1["<<i<<"]= "<<tab1[i]<<endl;
    }
    return 0;
}
```

Innym sposobem nadania wartości elementom tablicy jest **inicjalizacja**, czyli nadanie wartości początkowych w momencie definiowania tablicy.

PRZYKŁAD 33.4

Rozważmy przypisanie:

```
int tab1[5]={1,2,3,4,5,7};
```

Jeśli między klamrami zostanie umieszczona zbyt duża liczba elementów, to kompilator wyświetli błąd podczas kompilacji. Jeśli natomiast zastosujemy zapis:

```
int tab1[5]={1,2};
```

to elementowi zerowemu zostanie przypisana liczba 1, elementowi pierwszemu – liczba 2, natomiast trzy pozostałe zostaną uzupełnione zerami.

PRZYKŁAD 33.5

Przypisz wartości początkowe tablicy 5-elementowej: 1, 2, 5, 0, 0. Wyświetl wartości parzyste tej tablicy oraz wartości nieparzyste.

```
include<iostream>
using namespace std;
int i;
int main()
{
int tab[5]={1,2,7};
cout<<"Wypisanie wartosci parzystych:"<<endl;
    for (i=0;i<5;i++)
    {
       if(tab[i]%2==0)
       cout<<"tab["<<i<<"]= "<<tab[i]<<endl;
    }
cout<<"Wypisanie wartosci nieparzystych:"<<endl;
    for (i=0;i<5;i++)
    {
       if(tab[i]%2!=0)
       cout<<"tab["<<i<<"]= "<<tab[i]<<endl;
    }
return 0;
}
```

Zmienna **tab** została zainicjalizowana tylko trzema elementami różnymi od zera, ponieważ wiadomo, że pozostałe elementy tablicy zostaną uzupełnione zerami. Pierwsza pętla **FOR** wyszukuje i wyświetla tylko wartości parzyste. Aby sprawdzić parzystość, czyli podzielność przez 2, używamy operatora **%**. Druga pętla **FOR** wypisuje wartości nieparzyste, zatem warunek, który w niej sprawdzamy, jest przeciwny do tego w pętli pierwszej.

Żadna z pętli nie potrzebuje klamry otwierającej ani klamry zamykającej, ponieważ w ich wnętrzu znajduje się pojedyncza instrukcja.

PRZYKŁAD 33.6

Wylosuj liczby całkowite z zakresu od 1 do 100, wstaw je do 10-elementowej tablicy, a następnie wyświetl te wartości i oblicz ich sumę. Wynik wypisz na ekranie.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
{
int tab[10];
int i,suma;
      srand(time(NULL));
      for ( i=0;i<10;i++)
      {
      //losowanie wartości z zakresu od 1 o 100
      tab2[i]=rand() % 100 +1;
      cout<<"tab["<<i<<"]= "<<tab[i]<<endl;
      suma=suma+tab[i];
      }
cout<<"Suma wartosci w tablicy wynosi: "<<suma;
return 0;
}
```

Funkcja **rand**, losująca liczby całkowite, pochodzi ze standardowej biblioteki **cstdlib**. Ta funkcja losuje nam liczbę całkowitą w przedziale od 0 do RAND_MAX (predefiniowana stała symboliczna, $32767=2^{15}-1$).

Jeżeli uruchomimy kilka razy powyższy program bez użycia funkcji **srand**, to za każdym razem zostaną wylosowane te same liczby. Funkcja **srand** ustawia punkt startowy dla mechanizmu generowania kolejnych liczb całkowitych. Oznacza to, że zapis

```
srand(120);
```

ustawi punkt startowy 120.

Aby uzyskać każdorazowo inne wylosowane wartości, w raz skompilowanym programie należy powiązać funkcję **srand** z czasem rzeczywistym, ustawionym obecnie w komputerze. W tym celu należy dołączyć bibliotekę **ctime**, która umożliwia m.in. skorzystanie z funkcji **time**. Funkcja ta zwraca obecny czas na komputerze w postaci liczby.

Zapis **rand()%100 +1** zainicjuje losowanie liczb od 1 do 100.

UWAGA

Zapis **`srand(time(NULL))`** należy wywołać tylko raz, na samym początku programu.

Tabela 33.1. Zastosowanie funkcji rand()

Zastosowanie funkcji `rand()` do generowania liczb losowych	
liczba = p + rand()% (q-p+1);	generowanie liczby losowej całkowitej z zakresu [p;q]
liczba = rand() % (q+1);	generowanie liczby losowej całkowitej z zakresu [0;q]
liczba = p + (double) rand()/ RAND_MAX*(q-p);	generowanie liczby losowej rzeczywistej z zakresu [p;q]
liczba = (double) rand()/ RAND_MAX;	generowanie liczby losowej rzeczywistej z zakresu [0;1]

Przekazywanie tablicy do funkcji

Jak już wiadomo, przesyłanie parametru do funkcji odbywa się przez wartość lub przez referencję. Tablicy nie da się jednak przesłać przez wartość. W taki sposób można przekazać pojedynczy element, ale nie – całą zawartość tablicy.

Tablice przesyła się, podając funkcji tylko adres początku tablicy.

PRZYKŁAD 33.7

Weźmy pod uwagę funkcję:

```
void funkcja(int tab[]);
```

której argumentem jest tablica liczb całkowitych. Wywołujemy ją w następujący sposób:

```
int t[]={3,2,4,5};
funkcja(t);
```

W nazwie tablicy wywoływanej na potrzeby funkcji nie używamy nawiasów kwadratowych.

UWAGA

Nazwa tablicy jest równocześnie adresem jej zerowego elementu.

Wyrażenie **`tab+3`** jest adresem miejsca w pamięci, w którym istnieje element o indeksie 3. Element o indeksie 3 to element **`tab[3]`**, natomiast adres takiego elementu to **`&tab[3]`**.

Znak **&** (ampresand) jest jednoargumentowym operatorem uzyskującym adres danego obiektu.

Oznacza, że zapis **tab+3** jest równoważny zapisowi **&tab[3]**.

PRZYKŁAD 33.8

Zadanie z przykładu 33.6 można rozwiązać także z zastosowaniem własnej funkcji, której argumentem będzie nasza tablica. Funkcja **zsumuj** będzie wykonywała dodawanie poszczególnych elementów tablicy i wyświetlała wynik na ekranie monitora.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
void zsumuj (int tab[])
{
    int suma=0;
    for (int i=0;i<10;i++)
       suma=suma+tab[i];
    cout<<"Suma wynosi: "<<suma;
}
int main()
{
    int tab[10];
    int i,suma;
    srand(time(NULL));

    for ( i=0;i<10;i++)
    {
         tab[i]=rand() % 100 +1;
         cout<<"tab["<<i<<"]= "<<tab[i]<<endl;
    }
 zsumuj(tab);

    return 0;
}
```

Tablice wielowymiarowe

Tablice wielowymiarowe to tablice, których elementami są inne tablice. Są to po prostu tablice tablic.

Postać deklaracji tablicy dwuwymiarowej:

```
typ nazwa_tablicy[il_elementów][il_elementow];
```

PRZYKŁAD 33.9

```
float tab3[4][2];
```

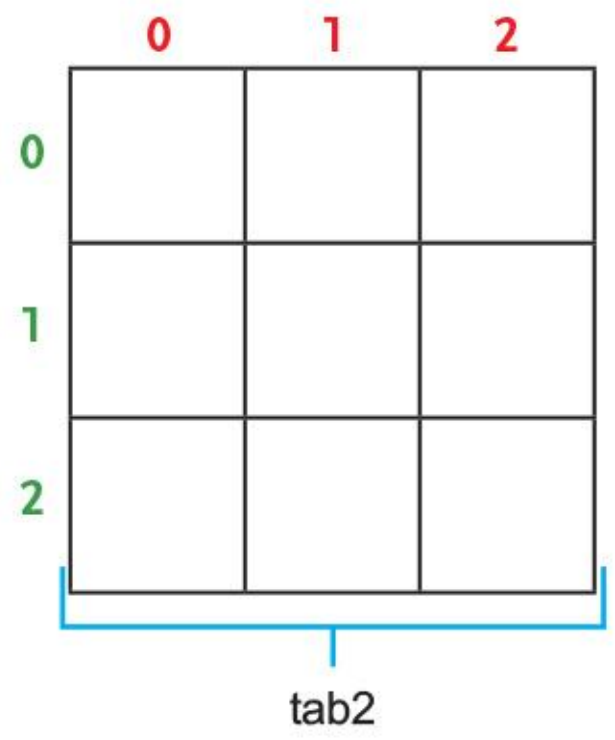
W ten sposób została zdefiniowana czteroelementowa tablica **tab3**. Każdy z elementów zawiera dwuelementową tablicę liczb typu rzeczywistego.

Aby odwołać się do każdej z komórek, należy w nawiasach kwadratowych podać numer wiersza i numer kolumny komórki, do której się odwołujemy. Trzeba pamiętać o tym, że numerujemy je od zera.

PRZYKŁAD 33.10

Załóżmy, że zadeklarowaliśmy zmienną **tab2**, która jest tablicą dwuwymiarową o trzech wierszach i trzech kolumnach, czyli:

```
float tab2[3][3];
```



Rys. 33.2. Postać dwuwymiarowej tablicy

Aby wpisać wartości do tej tablicy, można zastosować zapis:

```
tab2[0][0]=1;
tab2[0][1]=2;
tab2[0][2]=3;
tab2[1][0]=4;
tab2[1][1]=5;
tab2[1][2]=6;
tab2[2][0]=7;
tab2[2][1]=8;
tab2[2][2]=9;
```

Oczywiście, jest to sposób niewygodny i mało czytelny. Zwykle elementy tablicy dwuwymiarowej wczytuje się lub wypisuje za pomocą podwójnej pętli **FOR**.

PRZYKŁAD 33.11

```
#include <iostream>
using namespace std;

int main()
{
    float tab2[3][3];
    int i,j;
    for ( i=0;i<3;i++)
    {
       for ( j=0;j<3;j++)
       {
       cout<<"Podaj wartosc elementu ["<<i<<"]["<<j<<"]: ";
       cin>>tab2[i][j];
       }
    }
return 0;
}
```

Powyższy kod poprosi użytkownika o podanie wartości dla kolejnych elementów tablicy. Wprowadzane wartości będą uzupełniane wierszami, czyli w pierwszym wierszu są to elementy `[0][0]`, `[0][1]`, `[0][2]`,
w drugim wierszu `[1][0]`, `[1][1]`, `[1][2]`,
a w ostatnim elementy `[2][0]`, `[2][1]`, `[2][2]`.

Wprowadzone wartości mogą zostać wyświetlone za pomocą tych samych pętli **FOR**. Można również użyć oddzielnych instrukcji w celu wyświetlenia tablicy w postaci macierzowej.

PRZYKŁAD 33.12

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
{
int tab2[3][3];
int i,j;
srand(time(NULL));
    for ( i=0;i<3;i++)
    {
```

```
    for ( j=0;j<3;j++)
    {
    //losowanie wartości z zakresu od 1 o 10
    tab2[i][j]=rand() % 10 +1;
    }
}
for (i=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
    //wyświetlenie wartości w i-tym wierszu
    cout<<tab2[i][j]<<"\t";
    }
    //przejście do nowej linii po wypisaniu wartości w i-tym
    wiersz
    cout<<endl;
    }
return 0;
}
```

Powyższy program wygeneruje liczby od 1 do 10 i wstawi je do tablicy **tab2**, a następnie wypisze wpisane liczby w postaci macierzowej (rys. 33.3).

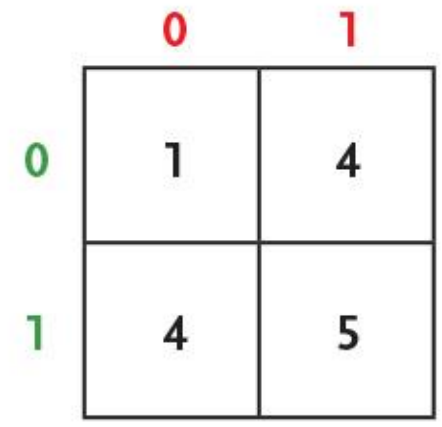
```
6       2       7
10      3       5
6       8       6

Process returned 0 (0x0)   execution time : 0.169 s
Press any key to continue.
```

Rys. 33.3. Wynik działania programu wypełniającego tablicę liczbami

Wartości początkowe tablicy możemy nadać także podczas jej deklaracji:

```
int tab[2][2]={{1,4},{4,5}};
```

Poszczególne wiersze tablicy są wyróżnione nawiasami kwadratowymi i klamrowymi. Powyższy zapis wypełni tablicę w następujący sposób:



Rys. 33.4. Wypełniona tablica dwuwymiarowa

Nawiasy klamrowe można pominąć.

```
int b[2][3]={3,6,2,4,1,0};
```

Ponadto podczas nadawania wartości tablicy można pominąć wartość w pierwszym nawiasie kwadratowym:

```
float tablica[][2] = {{1,1},{4,5},{0,-1}};
```

Deklaracja tablic wyższych wymiarów odbywa się analogicznie. Zapis lub odczyt wartości przechowywanych w tablicy wielowymiarowej następuje na tych samych zasadach, co dla tablicy jednowymiarowej. Należy odpowiednio podać „współrzędne" danego elementu w tablicy.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Podaj postać deklaracji tablicy jednowymiarowej oraz dwuwymiarowej.
2. W jaki sposób nadaje się wartości elementów tablicy?
3. Omów przekazywanie tablicy do funkcji.
4. Omów zastosowanie funkcji **rand**.
5. Dlaczego warto stosować instrukcję **FOR** podczas nadawania wartości elementów tablicy?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz program obliczający i wyświetlający średnią arytmetyczną wartości 8-elementowej tablicy. Następnie sprawdź, czy wynik jest liczbą większą, mniejszą, czy też równą 5, i wypisz odpowiedni komunikat.
2. Napisz program, który wylosuje liczby z zakresu od 1 do 500, wstawi je do 10-elementowej tablicy, a następnie wyświetli wartości, które są jednocześnie parzyste i podzielne przez 3.
3. Napisz program, w którym zadeklarujesz 50-elementową tablicę liczb całkowitych. Wygeneruj dla niej wartości z zakresu <1;50>. Wyświetl na ekranie kolejno wszystkie liczby parzyste spośród wylosowanych. Oblicz, ile ich jest. Wydrukuj tę wartość na ekranie.

4. Napisz program, w którym zadeklarujesz 80-elementową tablicę liczb całkowitych. Wygeneruj dla niej wartości z zakresu <1;100>. Spośród wylosowanych liczb wyświetl na ekranie kolejno wszystkie liczby, które należą do przedziału <10,50>. Oblicz ich średnią. Wydrukuj tę wartość na ekranie.
5. Napisz program, w którym zadeklarujesz 50-elementową tablicę liczb całkowitych. Wygeneruj dla niej wartości z zakresu <1;100>. Wydrukuj na ekranie te spośród liczb, których indeks jest liczbą będącą kwadratem liczby całkowitej (1, 4, 9, 16 itd.).
6. Napisz program wypełniający 40-elementową tablicę wartościami (n=40):
   a) od 0 do n-1
   b) od 7 do n+6
   c) ciągiem 4, 8, 12, 16, ...
   d) ciągiem 1, 2, 4, 8, 16, ...
   e) 2,3,4,....,n+1
7. Napisz program, który wygeneruje liczby z zakresu <50,100> dla 30-elementowej tablicy. Wyświetl kolejne elementy tablicy, oddzielone przecinkami. Wyzeruj liczby na parzystych miejscach w tablicy i wyświetl nową tablicę.
8. Napisz program, który wczytuje 10-elementową tablicę liczb całkowitych i wypisuje jej elementy na ekranie monitora. Następnie oblicz sumę kwadratów tych elementów tablicy, które podczas dzielenia przez 4 dają resztę 2 lub są niedodatnie. Wypisz tę sumę na ekranie.
9. Napisz program, który wczytuje 10-elementową tablicę wartości całkowitych podanych przez użytkownika, a następnie wypisuje informacje: czy te liczby tworzą ciąg arytmetyczny, czy liczby tworzą ciąg naprzemienny, czy nie jest to żaden z tych ciągów.
10. Napisz program wczytujący z klawiatury n liczb całkowitych z zakresu od 1 do 100. Program ma znaleźć drugi co do wielkości wyraz podanego ciągu liczb (czyli liczbę mniejszą od największej liczby występującej w ciągu, ale większą od wszystkich pozostałych).
11. Napisz program, który obliczy sumę elementów w wierszach tablicy dwuwymiarowej o trzech wierszach i trzech kolumnach.
12. Napisz program, który wczyta macierz kwadratową liczb całkowitych wymiaru 3 i wypisze jej elementy na ekranie monitora. Następnie program ma obliczyć iloczyn elementów tej macierzy, które są podzielne przez 3 lub 4, i wypisać obliczony iloczyn na ekranie.
13. Napisz program, który znajdzie minimalny oraz maksymalny element tablicy dwuwymiarowej o 4 wierszach i 5 kolumnach. Elementy tablicy wygeneruj z zakresu od <20,50> i wyświetl je na ekranie monitora w postaci macierzowej.
14. Napisz program, który zsumuje elementy parzyste w każdym wierszu tablicy dwuwymiarowej kwadratowej o wymiarze 4.
15. Napisz program, który zsumuje elementy podzielne przez 7 w każdej kolumnie tablicy dwuwymiarowej kwadratowej o wymiarze 3.
16. Napisz program, który znajdzie maksymalny element na głównej przekątnej kwadratowej tablicy dwuwymiarowej o wymiarze 4.

17. Napisz program, który wygeneruje wartości dla dwóch tablic kwadratowych o wymiarze 3. Wyświetl elementy tych tablic w postaci macierzowej. Parzyste wiersze pierwszej tablicy zastąp parzystymi elementami drugiej tablicy i wyświetl ponownie elementy nowych tablic.
18. Napisz program, który wygeneruje wartości dla tablicy kwadratowej o wymiarze 4, a następnie zamieni miejscami wiersze z kolumnami. Wypisz powstałą tablicę.
19. Napisz program, który wygeneruje wartości dla dwóch tablic kwadratowych o wymiarze 3 i wyświetli wartości obu tablic. Dodaj obie tablice. Wynikiem jest tablica kwadratowa o wymiarze 3. Wyświetl wartości nowej tablicy.
20. Napisz program, który wczyta wartości dla tablicy kwadratowej o wymiarze 3, podane przez użytkownika. Zlicz liczbę wystąpień określonych wartości i wyświetl wyniki na ekranie monitora.
21. Rozwiąż układ dwóch równań metodą wyznaczników z wykorzystaniem tablic dwuwymiarowych.

34 Tablice znakowe i stringi

ZAGADNIENIA

- Definicja stringu
- Inicjalizacja łańcucha znaków
- Podstawowe funkcje działające na tablicach znaków
- Przekazywanie tablicy do funkcji

Łańcuch znaków (ang. *string*) to ciąg złożony z zera lub większej liczby znaków zawartych między znakami cudzysłowu, np. „kot". W rzeczywistości łańcuch znaków jest tablicą, której elementami są pojedyncze znaki (tablica elementów typu char). Ostatnim elementem tablicy jest znak o kodzie 0 (stała liczbowa 0 lub stała znakowa '\0'), oznaczający koniec napisu.

UWAGA

Rozmiar fizycznej pamięci przeznaczonej na napis musi być o jeden większy niż liczba znaków zawartych między znakami cudzysłowie.

Deklaracja zmiennej mogącej przechowywać napis jest podobna do deklaracji zwykłej tablicy:

```
char nazwa_tablicy[ilość_elementow];
```

PRZYKŁAD 34.1

```
char tab2[10];
```

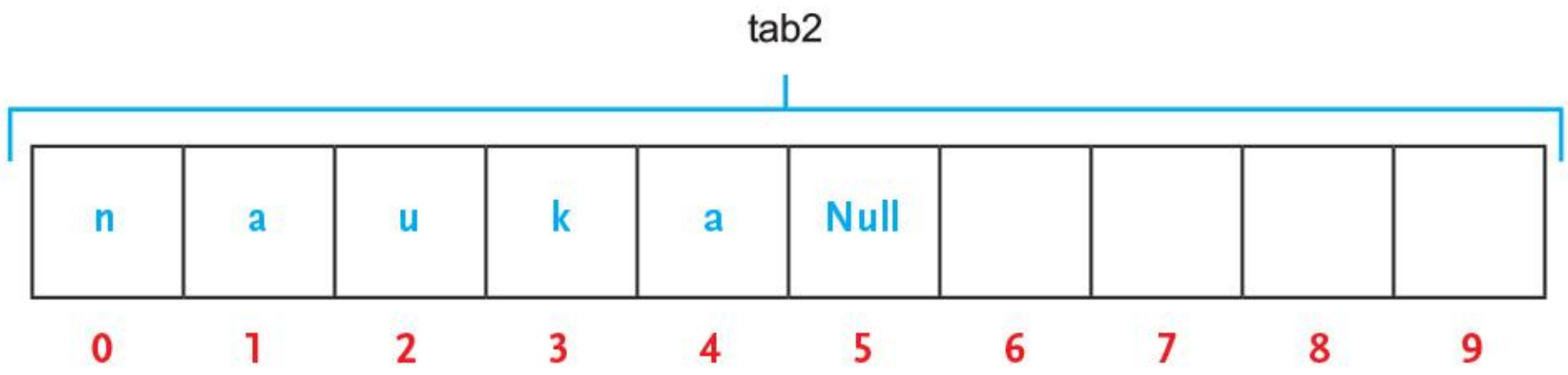
Zmienna **tab2** jest tablicą o maksymalnej długości do 9 znaków.

Inicjalizacja łańcucha znaków

Deklarowanemu łańcuchowi znaków możemy nadać wartość początkową:

```
char tab2[10]="nauka";
```

Napis umieszczamy w cudzysłowie. W rezultacie każda komórka tablicy **tab2** zawiera kolejną literę, komórka o indeksie 5 zawiera wartość NULL jako koniec tekstu, pozostałe elementy są uzupełnione zerami.



Rys. 34.1. Inicjalizacja łańcucha znaków

To samo można uzyskać przez podanie pojedynczej litery umieszczonej w apostrofach:

```
char tab2[10]={'n','a','u','k','a'};
```

Jeśli zastosujemy zapis:

```
char tab2[]="nauka";
```

to kompilator policzy, ile liter ma podany tekst (w tym przypadku 5). Ponieważ tekst został ujęty w cudzysłów, zostanie zarezerwowany jeszcze jeden element na znak NULL.

Powyższe sposoby nadawania wartości łańcuchowi znaków są poprawne tylko przy deklaracji. Błędny jest zatem poniższy zapis:

```
char tekst[10];
tekst = "Pies";
```

Prawidłowy zapis wymaga wykorzystania funkcji **strcpy()** z pliku nagłówkowego **string.h**:

```
char txt[10];
strcpy(txt,"Pies");
```

Podobnie jak podczas wpisywania tekstu, do wprowadzania tekstu posługujemy się tylko nazwą tablicy.

PRZYKŁAD 34.2

```
#include<iostream>
using namespace std;

int main()
{
char dane[50];
cout<<"Podaj imie i nazwisko: ";
cin>>dane;
cout<<"Twoje dane osobowe: "<<dane<<endl;
return 0;
}
```

Niestety, działanie funkcji **cin** spowoduje, że wprowadzane dane zostaną wczytane do pierwszej napotkanej spacji we wprowadzonym łańcuchu znaków. Aby pozbyć się tego problemu, należy zastosować metodę **getline()**, której postać jest następująca:

```
cin.getline(tab,bufor);
```

gdzie **tab** to tablica znaków, natomiast **bufor** to rozmiar tablicy.

Wystarczy więc zamienić linijkę **cin>>dane**; na linijkę **cin.getline(dane,50);**

Efekt poprawionego kodu jest następujący:

```
Podaj imie i nazwisko: Anna Kowalska
Twoje dane osobowe: Anna Kowalska

Process returned 0 (0x0)   execution time : 5.772 s
Press any key to continue.
```

Rys. 34.2. Działanie metody **getline()**

Podstawowe funkcje działające na tablicach znaków

Tabela 34.1. Funkcje działające na łańcuchach

Nazwa funkcji	Nagłówek funkcji	Działanie funkcji
strlen()	int strlen(const char *s);	• zwraca długość łańcucha znaków • nie bierze pod uwagę znaku **'\0'**
strcpy()	char *strcpy(char *s1, const char *s2);	• kopiuje łańcuch **s2** do łańcucha **s1**
strcat()	char *strcat(char *s1, const char *s2);	• dołącza do łańcucha **s1** łańcuch **s2**
strchr()	char *strchr(const char *s, int c);	• przeszukuje łańcuch **s** w celu znalezienia pierwszego wystąpienia znaku **c** • jeśli znak nie został znaleziony, to zwraca wskaźnik do znalezionego znaku lub **NULL**
strcmp()	int strcmp(const char *s1, const char *s2);	• porównuje łańcuchy **s1** i **s2** z rozróżnianiem wielkości liter • zwraca **0**, gdy **s1=s2**, wartość mniejszą od zera, gdy **s1<s2** i wartość większą od zera, gdy **s1>s2**
strcmpi()	int strcmpi(const char *s1, const char *s2);	• działa jak **strcmp()**, ale bez rozróżniania wielkości liter
strlwr()	char *strlwr(char *s);	• zamienia w łańcuchu **s** duże litery na małe
strrev()	char *strrev(char *s);	• odwraca kolejność znaków w łańcuchu **s**

Nazwa funkcji	Nagłówek funkcji	Działanie funkcji
strset()	char *strset(char *s, int c);	• wypełnia łańcuch **s** znakiem **c**
strstr()	char *strstr(const char *s1, const char *s2);	• przeszukuje łańcuch **s1** w celu odnalezienia podłańcucha **s2**, zwracając wskazanie do elementu łańcucha **s1**, od którego zaczynają się znaki takie same jak w łańcuchu **s2**, lub **NULL**, gdy **s2** nie występuje w **s1**
strupr()	char *strupr(char *s);	• zamienia w łańcuchu **s** litery małe na duże

PRZYKŁAD 34.3

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
char tekst[]="Ala ma kota";
cout<<"Twoj tekst to: "<<tekst<<endl;
int dl=strlen(tekst);
cout<<"Liczba znakow w tekscie: "<<dl<<endl;
cout<<"Twoj tekst pisany wielkimi literami:
"<<strupr(tekst)<<endl;
cout<<"Twoj tekst pisany malymi literami:"<<strlwr(tekst)<<endl;
cout<<"Dopiszemy dalsza czesc zdania i otrzymamy:"<<endl;
cout<<strcat(tekst,", a kot ma Ale")<<endl;
cout<<"Odwrocony tekst na postac: "<<endl;
cout<<strrev(tekst);
return 0;
}
```

Powyższy program ilustruje działanie funkcji operujących na tablicach znaków. Efekt jest następujący:

```
Twoj tekst to: Ala ma kota
Liczba znakow w tekscie: 11
Twoj tekst pisany wielkimi literami: ALA MA KOTA
Twoj tekst pisany malymi literami: ala ma kota
Dopiszemy dalsza czesc zdania i otrzymamy:
ala ma kota, a kot ma Ale
Odwrocony tekst na postac:
elA am tok a ,atok am ala
Process returned 0 (0x0)   execution time : 0.085 s
Press any key to continue.
```

Rys. 34.3. Wynik działania programu wykonującego operacje na łańcuchach tekstowych

PRZYKŁAD 34.4

Tablicę znaków traktujemy jak zwykłą tablicę, która zamiast liczb zawiera znaki. Można zatem przeszukiwać ją w celu znalezienia powtarzających się znaków czy też sortować elementy alfabetycznie.
Rozważmy program, który obliczy liczbę liter a występujących w tekście podanym przez użytkownika.

```
#include<iostream>
#include<string.h>
using namespace std;
int main()
{
char dane[100];
int dl,i,ilosc=0;
cout<<"Podaj dowolny tekst: ";
cin.getline(dane,100);
dl=strlen(dane);
for(i=0;i<dl;i++)
{
    if(dane[i]=='a' || dane[i]=='A')
    {
       ilosc=ilosc+1;
    }
}
cout<<"W tekscie znajduje sie "<<ilosc<<" liter a";
return 0;
}
```

Po zaczytaniu tekstu z klawiatury jest obliczana długość wprowadzonego łańcucha. Następnie za pomocą pętli **for** przeszukujemy, od pierwszego znaku do ostatniego, łańcuch w celu sprawdzenia, czy występuje w nim litera a (mała lub wielka).

W języku C++ możemy stosować zmienne **string**, które są obiektami klasy **string**.

Aby utworzyć zmienną typu string, należy zastosować polecenie:

```
string nazwa_zmiennej;
```

PRZYKŁAD 34.5

Rozważmy zadanie z przykładu 34.2. Zmieńmy zmienną tablicową na typ string.

```
#include<iostream>
#include<string>.h
using namespace std;
```

```
int main()
{
string dane;
cout<<"Podaj imie i nazwisko: ";
cin>>dane;
cout<<"Twoje dane osobowe: "<<dane<<endl;
return 0;
}
```

Niestety, tak jak w przypadku tablicy znaków, tekst podawany przez użytkownika jest wczytywany do pierwszej spacji. Tym razem musimy posłużyć się funkcją **getline**, która ma następującą postać:

```
getline(cin,zmienna);
```

Zastosowanie tej instrukcji spowoduje, że zostanie wczytany cały wprowadzany tekst.

Do odwołania się do poszczególnych metod zawartych w klasie **string** jest przeznaczony operator kropki. Do działania na zmiennych typu **string** służy wiele metod, np.:

- **size()**, **lenght()** – zwraca długość łańcucha;
- **append()** – dodaje do stringa kolejne znaki lub ciąg znaków;
- **assign()** – podmienia ciąg znaków na inny;
- **insert()** – wstawia ciąg znaków w określone miejsce;
- **replace()** – podmienia ciąg znaków (zaczyna od podanego miejsca w stringu);
- **swap()** – argumentami są dwa stringi, które zostają zamieniane miejscami, czyli jeden string przyjmuje wartość drugiego;
- **copy()** – kopiuje wybrany podciąg znaków ze stringa i wpisuje go do tablicy znaków;
- **compare()** – porównuje jeden string z drugim; metoda zwraca **0** , gdy dwa stringi są takie same co do wielkości liter, **-1**, gdy w porównywanym stringu na danej pozycji znajduje się litera, która w kodzie ASCII ma większą wartość, lub gdy ten ciąg jest dłuższy, oraz **1** w odwrotnej sytuacji (działanie identyczne jak w funkcji strcmp).

Dodawanie kilku stringów (konkatenacja) odbywa się za pomocą operatora „+”, natomiast porównanie – za pomocą operatora „==”.

Działanie niektórych z wymienionych metod przedstawiono w przykładzie 34.6.

PRZYKŁAD 34.6

```
#include <iostream>
#include <string.h>
using namespace std;
int main()
{
string tekst="Ala ma kota";
```

```
string tekst2=", ale go nie chce";
string napis1="Systemy ";
string napis2="operacyjne";
//konkatenacja stringow - sklejenie napisow
cout<<"Po dzialaniu operatora +: "<<napis1+napis2<<endl;
cout<<"Twoj tekst to: "<<tekst<<endl;
int dl=tekst.length();
cout<<"Liczba znakow w tekscie: "<<dl<<endl;
cout<<"Po dzialaniu metody append: ";
cout<<tekst.append(tekst2)<<endl;
//wstawienie nazwiska do tekstu
cout<<tekst.insert(3,"Kowalska")<<endl;
//zamiana nazwiska Kowalska na Nowak
cout<<tekst.replace(4,9,"Nowak")<<endl;
//tekst pisany wielkimi literami
for(int i=0;i<tekst.length();i++)
    tekst[i]=toupper(tekst[i]);
cout<<tekst<<endl;
return 0;
}
```

Efekt jest następujący:

```
Po dzialaniu operatora +: Systemy operacyjne
Twoj tekst to: Ala ma kota
Liczba znakow w tekscie: 11
Po dzialaniu metody append: Ala ma kota, ale go nie chce
Ala Kowalska  ma kota, ale go nie chce
Ala Nowak ma kota, ale go nie chce
ALA NOWAK MA KOTA, ALE GO NIE CHCE

Process returned 0 (0x0)   execution time : 0.079 s
Press any key to continue.
```

Rys. 34.4. Zastosowanie metod z klasy **string**

SPRAWDŹ SWOJĄ WIEDZĘ

1. Czym różni się tablica znaków od stringa?
2. Wymień sposoby inicjalizacji tablicy znaków i stringów.
3. Omów funkcje działające na łańcuchach.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz program, który wyświetli w postaci rozstrzelonej (poszczególne litery oddzielone spacjami) tekst wprowadzony przez użytkownika.
2. Napisz program, który wypisze z podanego 10-znakowego tekstu podciąg od drugiego znaku do 8.
3. Napisz program, który po podaniu przez użytkownika imienia odgadnie jego płeć.
4. Napisz program, który sprawdzi, czy podane słowo jest palindromem.
5. Napisz program, który wypisze wszystkie anagramy podanego pięcioliterowego słowa.

35 Wskaźniki

ZAGADNIENIA
- Definicja wskaźnika
- Zastosowanie wskaźników
- Wskaźniki jako argumenty funkcji

Wskaźniki mają szerokie zastosowanie w C++, co daje temu językowi ogromną elastyczność.

Wskaźnik (ang. *pointer*) to zmienna, która przechowuje adres w pamięci RAM do innej zmiennej. Może on wskazywać na jakąś zmienną, strukturę, tablicę, a nawet funkcję.

Podstawowe operatory niezbędne do operowania wskaźnikami

- `*` – **operator wyłuskania wartości** zmiennej, na którą wskazuje wskaźnik (wyciąga wartość ze wskaźnika);
- `&` – **operator pobrania adresu** danej zmiennej, tablicy, struktury (pobiera adres zmiennej).

Wskaźnik zajmuje zazwyczaj 4 bajty, bez względu na to, jaki typ danych wskazuje. Rozmiar wskaźnika może być jednak różny, zależnie od użytego kompilatora. Wskaźnik zwraca adres pierwszego bajta danych wybranej zmiennej.

Aby pobrać adres dowolnej zmiennej, wystarczy napisać: `&nazwa_zmiennej`.

Aby utworzyć zmienną wskaźnikową, po typie zmiennej dopisujemy `*` (gwiazdkę).

PRZYKŁAD 35.1

```
int wyr=25;
int *liczba;
liczba=&wyr;
```

Zmienna **wyr** jest zwykłą zmienną całkowitą. Zmienna **liczba** jest wskaźnikiem do obiektu typu **int**, a do niej zapisujemy adres zmiennej **wyr**. Treścią wskaźnika jest informacja o tym, gdzie wskazywany obiekt się znajduje, a nie – co się w nim znajduje.

Można definiować wskaźniki do obiektów różnych typów, np. **char**, **float**, **int**:

```
char *liczba2;
float *liczba3;
```

UWAGA

Należy pamiętać, że wskaźnik służący do pokazywania obiektów jednego typu nie nadaje się do pokazywania obiektów innego typu.

PRZYKŁAD 35.2

```
#include <iostream>
using namespace std;
int main()
{
int wyr=25;
int *liczba;
liczba=&wyr;
cout<<"*liczba = "<<*liczba<<endl;
cout<<"liczba = "<<liczba;
return 0;
}
```

W powyższym programie zdefiniowano wskaźnik **liczba** do zmiennej typu całkowitego oraz zmienną całkowitą **wyr**. Zapis **liczba=&wyr** oznacza zapisanie adresu zmiennej **wyr**. Kiedy wskaźnik już pokazuje określone miejsce, możemy odnieść się do tego obiektu, który pokazuje (odczytać jego zawartość). Do takiej operacji służy operator *****. Gwiazdka kieruje do wartości obiektu pokazywanego przez wskaźnik. Wskaźnik **liczba** pokazuje na zmienną **wyr**. Skoro zawartością **wyr** jest wartość 25, to oba komunikaty **cout** są równoważne.

Wskaźniki stosuje się, gdy zależy nam na:

- ulepszeniu pracy z tablicami, tj. zwiększeniu szybkości zapisu i odczytu wartości w tablicy;
- funkcjach, które mogą zmieniać wartość argumentów przesyłanych do nich;
- dostępie do specjalnych komórek pamięci;
- dynamicznym rezerwowaniu i zwalnianiu obszarów pamięci.

Przypisywanie wartości za pomocą wskaźnika

Podstawowymi operatorami wykorzystywanymi w działaniach na zmiennych wskaźnikowych są operator pobrania adresu **&** i operator odwołania do zmiennej wskazywanej *****. Jeśli **x** jest zmienną, to **&x** jest adresem tej zmiennej. Operator ***** umożliwia dostęp do obiektu wskazywanego przez wskaźnik.

PRZYKŁAD 35.3

```
#include <iostream>
using namespace std;
int main()
{
float x=36.6;
//wskaznik wsk_x zawiera adres zmiennej x
float *wsk_x=&x;
cout<<"*wsk ="<<*wsk_x<<endl;
//wyswietlenie adresu zmiennej x
cout<<"wsk ="<<wsk_x<<endl;
//wyswietlenie adresu wskaznika wsk_x
cout<<"&wsk ="<<&wsk_x<<endl;
//wyswietlenie adresu zmiennej x
cout<<"&x ="<<&x<<endl;
return 0;
}
```

Zmienna ***wsk_x** jest wskaźnikiem do zmiennej typu float. Po skorzystaniu z operatora wyłuskania **&** pobraliśmy wartość zmiennej **x**. Bez użycia tego operatora zostanie wyświetlona wartość wskaźnikowa, czyli adres zmiennej **x**. Za pomocą operatora **&** dla wskaźnika otrzymamy adres wskaźnika. Dzięki właściwemu przyłożeniu **&** do zmiennej **x** otrzymamy adres w pamięci.

Wynikiem programu jest następująca lista komunikatów.

```
*wsk =36.6
wsk =0x6dfeec
&wsk =0x6dfee8
&x =0x6dfeec
```

Rys. 35.1. Zastosowanie operatora wyłuskania

Zastosowanie wskaźników

Za pomocą wskaźników można **modyfikować wartości** innych zmiennych.

PRZYKŁAD 35.4

```
#include<iostream>
using namespace std;
int main()
{
int *a, *b, liczba;
```

```
liczba = 5;
//zmienna a wskazuje na adres zmiennej liczba
a = &liczba;
//zmienna b wskazuje na adres zmiennej liczba
b = &liczba;
//wypisanie wartości, ktora przechowuje zmienna liczba, czyli 5
cout<<*a<<" "<<*b<<endl;
//zmiana wartosci zmiennej liczba
*a = 7;
//zostanie wypisana trzy razy wartość 7
 cout<<*a<<" "<<*b<<" "<<liczba;
return 0;
}
```

Każda operacja, która korzysta z indeksów, może zostać wykonana za pomocą wskaźników. Posługiwanie się wskaźnikami zamiast indeksów na ogół sprawia, że przyśpiesza się działanie operacji. Należy pamiętać, że przed użyciem zarówno tablice, jak i funkcje wymagają deklaracji, co oznacza przydzielenie określonego miejsca pamięci.

Z rozdziału o tablicach wiadomo, że nazwa tablicy jest wskaźnikiem na jej pierwszy element. Oznacza to, że możemy odnieść się to tego elementu za pomocą nazwy tablicy i indeksu o wartości 0 lub przez ten wskaźnik.

Po elementach tablicy możemy poruszać się za pomocą indeksów – zarówno podawać kolejne numery komórek, począwszy od 0, jak i przeskakiwać kolejne komórki tablicy z wykorzystaniem wskaźnika.

Jeśli zadeklarujemy tablicę o nazwie **`TAB1`**, to zapis **`*TAB1`** oznacza wskazanie do początku tablicy TAB1.

PRZYKŁAD 35.5

```
#include<iostream>
using namespace std;
int main()
{
int tab[6] = {2, 4 ,6, 8, 10, 12};
//wypisanie pierwszego elementu tablicy, czyli tab[0]
cout<<*tab<<endl;
//wypisanie trzeciego elementu tablicy
cout<<*(tab+2)<<endl;
//wypisanie kolejno elementów tablicy poruszając się po niej
wskaźnikiem
for(int i=0;i<6;i++)
```

```
cout<<*(tab+i)<<" ";
cout<<endl;
return 0;
}
```

Efekt działania programu jest następujący:

```
2
6
2 4 6 8 10 12
Process returned 0 (0x0)   execution time : 0.203 s
Press any key to continue.
```

Rys. 35.2. Wynik działania programu, który ilustruje działania na tablicach za pomocą wskaźników

Wskaźniki jako argumenty funkcji

Należy pamiętać, że przekazywanie do funkcji zmiennej powoduje, że do tej funkcji jest przesyłana kopia wartości tej zmiennej. Po przekazaniu wskaźników jako argumenty mamy zapewnione, że każda zmiana wartości zmiennej wewnątrz funkcji będzie „widoczna" w miejscu, w którym została wywołana.

PRZYKŁAD 35.6

```
#include<iostream>
using namespace std;
void zamien(int *liczba1, int *liczba2)
{
    int pom = *liczba1;
    *liczba1 = *liczba2;
    *liczba2 = pom;
}
int main()
{
int a, b;
cin>>a>>b;
cout<<"Przed zamiana: "<<endl;
cout<<"a= "<<a<<" b= "<<b<<endl;
//przekazujemy adresy zmiennych
zamien(&a,&b);
//wartości zmiennych zostały zamienione
cout<<"Po zamianie: "<<endl;
```

```
cout<<"a= "<<a<<" b= "<<b;
return 0;
}
```

Użycie zwykłych zmiennych zamiast wskaźników nie spowodowałoby zamiany wartości, ponieważ funkcja pracowałaby na kopiach tych zmiennych. Efekt będzie zatem następujący.

```
34
78
Przed zamiana:
a= 34 b= 78
Po zamianie:
a= 78 b= 34
Process returned 0 (0x0)   execution time : 4.540 s
Press any key to continue.
```

Rys. 35.3. Wskaźniki jako argumenty funkcji

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest wskaźnik?
2. Jakie jest zastosowanie wskaźników?
3. W jaki sposób przekazuje się wskaźniki jako argumenty funkcji?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz program, w którym zadeklarujesz wskaźnik do zmiennej oraz przypiszesz mu odpowiedni adres. Wypisz na ekranie wartość zmiennej przez wskaźnik oraz wypisz adres zmiennej.
2. Napisz funkcję, która posortuje liczby zapisane w zmiennych *a*, *b* i *c* w rosnącej kolejności. Użyj wskaźników.
3. Napisz program, który zsumuje wartości wszystkich elementów 10-elementowej tablicy. Skonstruuj odpowiednią funkcję i zastosuj wskaźniki.

36 Operacje na plikach

ZAGADNIENIA

- Zapisywanie danych do pliku
- Otwieranie pliku do odczytu
- Odczytywanie danych z pliku
- Tryby otwarcia pliku

Narzędzia do pracy z plikami znajdują się w bibliotece **fstream** (termin pochodzi od angielskich słów file stream – 'strumień plikowy'), który dołączamy na początku programu za pomocą dyrektywy **#include <fstream>**. Aby móc odczytywać lub zapisywać dane do pliku, w pierwszej kolejności należy zdefiniować zmienną, za której pomocą będziemy mogli wykonywać te operacje. W tym celu wykorzystamy klasę **fstream**, która jest umieszczona w przestrzeni nazw **std**.

Operacje związane z przetwarzaniem pliku składają się z etapów, którymi są:

- zdefiniowanie strumienia, czyli stworzenie obiektu klasy **fstream**;
- otwarcie pliku;
- wykonanie operacji na pliku;
- zamknięcie pliku.

Zapisywanie danych do pliku tekstowego

Jak już wcześniej wspomniano, na początku programu należy dodać bibliotekę **fstream**, a następnie – utworzyć zmienną plikową. Zmienna umożliwi nam pracę na danym pliku. Jeśli chcemy zapisać dane do pliku, to najpierw należy go otworzyć, jeśli on istnieje, lub utworzyć, jeżeli na dysku nie ma pliku o takiej nazwie.

PRZYKŁAD 36.1

Utwórzmy program, który poprosi o podanie imienia i znaku zodiaku użytkownika. Te dane zostaną zapisane w pliku dane.txt, znajdującym się w folderze dane na dysku C:.

```
#include <iostream>
#include<fstream>
using namespace std;
string imie, zodiak;
int main()
{
```

```
cout<<"Podaj imie: ";
cin>>imie;
cout<<"Podaj znak zodiaku: ";
cin>>zodiak;
//zmienna plikowa
fstream plik;
//otwarcie pliku do zapisu
plik.open("c:/dane/dane.txt",ios::out);
//zapisanie imienia do pliku
plik<<imie<<endl;
//zapisanie znaku zodiaku do pliku
plik<<zodiak<<endl;
//zamkniecie pliku
plik.close();
return 0;
}
```

Po wprowadzeniu wartości dla dwóch zmiennych utworzono zmienną plikową o nazwie **plik**. Na rzecz tej zmiennej została wywołana metoda open, której pierwszym argumentem jest ścieżka dostępu do pliku. Ścieżka do pliku może być **względna** (określająca położenie pliku względem miejsca, w którym znajduje się program) lub **bezwzględna** (odnosząca się do katalogu głównego na dysku).

Drugi argument **ios::out** informuje kompilator, w jakim trybie ma zostać otwarty plik. Parametr **out** oznacza, że plik jest otwarty do zapisu.

Lista niektórych dostępnych trybów:

- **ios::app** – tryb umożliwiający dopisywanie danych do istniejących danych w pliku;
- **ios::in** – tryb umożliwiający odczytywanie danych z pliku;
- **ios::out** – tryb umożliwiający zapisywanie danych do pliku.

Po otwarciu pliku następuje zapisywanie do niego określonych wartości. Instrukcja jest bardzo prosta. Jeśli chcemy zapisać wartość zmiennej do pliku, wystarczy słowo **cin** zastąpić nazwą zmiennej plikowej. Po wykonaniu wszystkich operacji na pliku należy go zamknąć za pomocą metody **close**.

Po uruchomieniu programu i wprowadzeniu danych utworzy się plik o nazwie **dane.txt**. Będą w nim zawarte dane wprowadzone przez użytkownika.

Rys. 36.1. Utworzenie i zapisanie danych do pliku dane.txt

Niestety, po ponownym uruchomieniu programu i podaniu nowych danych okaże się, że poprzednie dane zostały nadpisane. Aby umożliwić dopisanie kolejnych danych do plików, wystarczy użyć dodatkowej instrukcji do metody open – modyfikatora app, który oznacza „dołącz" dane do pliku istniejącego:

```
plik.open("c:/dane/dane.txt",ios::out| ios::app);
```

Ta niewielka korekta programu spowoduje, że jeśli ponownie uruchomimy program i wprowadzimy inne dane, to zostaną one dopisane na koniec do pliku. Przykład pokazano na rys. 36.2.

Rys. 36.2. Dopisanie danych do pliku **dane.txt**

Odczytywanie zawartości pliku

Tym razem użyjemy metody **open** z modyfikatorem **in**, co oznacza, że plik otwieramy do odczytu danych.

Otwarcie pliku może zakończyć się zarówno powodzeniem, jak i fiaskiem. Zanim zaczniemy pracować na danych z pliku, warto sprawdzić, czy udało się otworzyć plik. Do tego celu należy użyć metody **good**, która należy do klasy **fstream**. Funkcja ta zwraca wartość TRUE, gdy plik istnieje, lub wartość FALSE, gdy plik nie istnieje.

UWAGA

Wiersze w pliku są numerowane od 1, a nie od zera – jak w tablicach.

PRZYKŁAD 36.2

```
#include <iostream>
#include<fstream>
using namespace std;
int main()
{
fstream plik;
plik.open("c:/dane.txt",ios::in);
    if(plik.good()==true)
       {
       cout<<"Udalo sie otworzyc plik";
       }
```

```
    else
       {
       cout<<"Nie udalo sie otworzyc pliku";
       }
plik.close();
return 0;
}
```

Na powyższym przykładzie widać, że jeśli plik istnieje, otrzymamy komunikat, iż udało się go otworzyć, i będziemy mogli przejść do etapu odczytania jego zawartości. Do odczytywania zawartości pliku stosujemy funkcję **getline**. Jej zadaniem jest wczytywanie kolejnych wierszy pliku oraz zwrócenie wartości logicznej, która informuje, czy można kontynuować odczyt. Jeśli funkcja zwróci wartość TRUE, to znaczy, że plik nie został jeszcze w całości odczytany i można dalej odczytywać dane z pliku. Jeśli natomiast funkcja zwróci FALSE, to znaczy, że został osiągnięty koniec pliku albo wystąpił błąd. Oczywiście odczyt za pomocą funkcji **getline** musi odbywać się w pętli.

PRZYKŁAD 36.3

```
#include <iostream>
#include<fstream>
using namespace std;
int main()
{
fstream plik;
plik.open("c:/dane/dane.txt",ios::in);
if( plik.good()==false)
{
cout<<"Nie udalo sie otworzyc pliku";
}
else
{
string wiersz;
while(getline(plik,wiersz))
{
cout<<wiersz<<endl;
}
}
plik.close();
return 0;
}
```

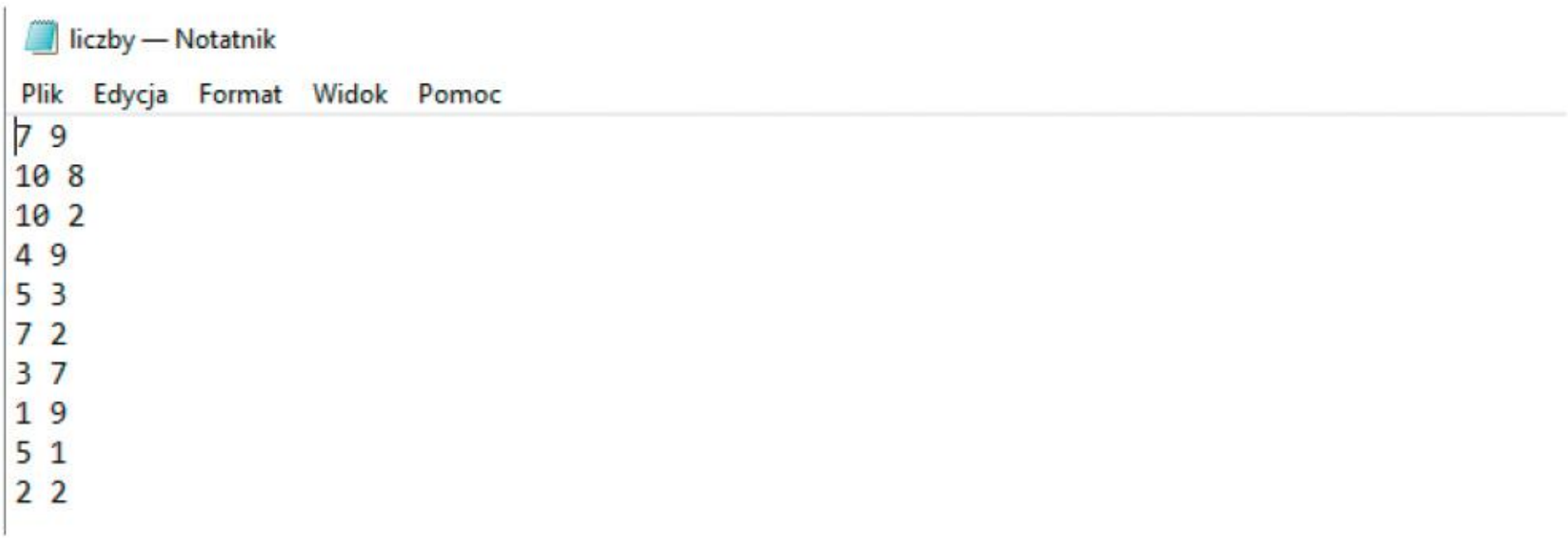
Powyższy kod otwiera plik o nazwie **dane.txt**. Jeśli plik istnieje, to będzie wyświetlona jego zawartość, w przeciwnym razie zostanie wyświetlony komunikat o błędzie odczytu. Zawartość pliku będzie wyświetlana dotąd, aż funkcja **getline** nie zwróci wartości FALSE.

PRZYKŁAD 36.4

Rozważmy program, który tworzy plik **liczby.txt**. Zostaną do niego zapisane wartości dwóch tablic **t1** oraz **t2**. Będą one zawierały wygenerowane liczby z zakresu <1,10>. Wartości w pliku będą zapisane w dwóch kolumnach odpowiadających liczbom z poszczególnych tablic.

```
#include <iostream>
#include<fstream>
#include<ctime>
#include<cstdlib>
using namespace std;
int t1[10],t2[10],i;
int main()
{
fstream plik;
srand(time(NULL));
//generowanie liczb dla t1 i t2, sumowanie wartości dla t3
      for(i=0;i<10;i++)
      {
      t1[i]=rand()%10+1;
      t2[i]=rand()%10+1;
      }
plik.open("c:/dane/liczby.txt",ios::out);
//zapisywanie danych do pliku
      for(i=0;i<10;i++)
      {
      plik<<t1[i]<<" "<<t2[i]<<endl;
      }
cout<<"Dane zostaly wczytane do pliku";
plik.close();
return 0;
}
```

Po uruchomieniu tego programu na dysku został utworzony plik o nazwie **liczby.txt**. Liczby są zapisane w dwóch kolumnach i przechowują po dziesięć wartości dla każdej z tablic (rys. 36.3).



Rys. 36.3. Wynik działania programu zapisującego wartości tablic do pliku

PRZYKŁAD 36.5

Kolejny kod będzie pobierał dane z tego pliku, wyświetlał je, wykonywał obliczenia na tych wartościach i wpisywał je do innego pliku.

```
#include <iostream>
#include<fstream>
using namespace std;
int t1[10],t2[10],t3[10],i,j;
int nr_linii=1;
string linia;
int main()
{
fstream plik,plik1;
plik.open("c:/dane/liczby.txt",ios::in);
    if(plik.good()==false)
    {
    cout<<"Nie moge otworzyc pliku";
    }
    else
    {
//wczytywanie odbywa sie az napotkamy na koniec pliku
      while(!plik.eof())
      {
      //wczytywanie danych z pliku do tablic
      for(i=0;i<10;i++)
      {
      plik>>t1[i]>>t2[i];
      }
      }
plik1.open("c:/dane/nowe_liczby.txt",ios::out);
```

```
//wyswietlenie wartosci tablic
      for(i=0;i<10;i++)
      {
      cout<<t1[i]<<"\t"<<t2[i]<<"\t"<<t1[i]+t2[i]<<endl;
      t3[i]=t1[i]+t2[i];
      plik1<<t1[i]<<" "<<t2[i]<<" "<<t3[i]<<endl;
      }
    }
plik.close();
plik1.close();
return 0;
}
```

Wynikiem będzie pobranie do tablic **t1** i **t2** wartości z dwóch kolumn pliku **liczby.txt**, a następnie – utworzenie trzeciej, która jest sumą dwóch pierwszych. Program wyświetli te wartości, a następnie zapisze je do nowego pliku o nazwie **nowe_liczby.txt**. W programie zastosowaliśmy metodę **eof()**, która sprawdza, czy został przeczytany cały plik.

Oprócz stosowania funkcji **getline**, aby odczytać dane z pliku, można zastosować jeszcze dwa sposoby:

- odczytanie pliku słowo po słowie za pomocą operatora **>>**;
- odczytanie znak po znaku za pomocą funkcji **get**.

W przypadku użycia operatora **>>** jest odczytywane wszystko, co znajduje się między miejscem w pliku, w którym aktualnie znajduje się kursor, a najbliższą spacją. Zależnie od typu użytej zmiennej odczytany tekst jest traktowany jako typ **double**, **int** lub **string**.

PRZYKŁAD 36.6

```
#include <iostream>
#include<fstream>
using namespace std;
string zdanie;
int main()
{
fstream plik;
//otwarcie pliku do zapisu i odczytu
plik.open("c:/dane/tekst.txt",ios::out| ios::in);
//odczyt z pliku
plik>>zdanie;
```

```
cout<<zdanie;
//zamkniecie pliku
plik.close();
    return 0;
}
```

Plik **tekst.txt** zawiera zdanie „programowanie jest fajne". Użycie operatora **>>** spowoduje, że na ekranie monitora zostanie wyświetlony tylko pierwszy wyraz. Jeśli chcemy wyświetlić całą zawartość pliku, musimy zastosować pętle **while**, która będzie sprawdzała, czy plik się skończył.

Ostatnia z metod odczytu danych z pliku – funkcja **get** odczytuje dane z pliku znak po znaku (również spacje, tabulacje itp.).

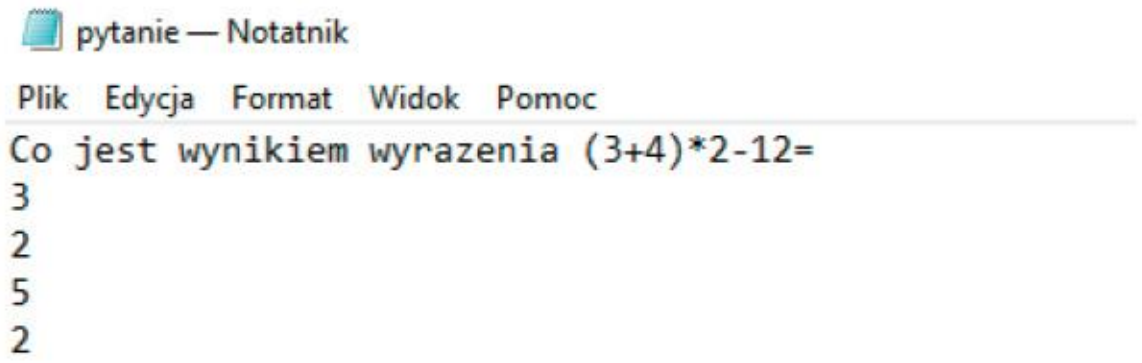
PRZYKŁAD 36.7

```
#include <iostream>
#include<fstream>
using namespace std;
char litera;
int main()
{
fstream plik;
//otwarcie pliku do zapisu i odczytu
plik.open("c:/dane/tekst.txt",ios::out| ios::in);
//odczytanie pliku znak po znaku
    while (!plik.eof())
    {
    plik.get(litera);
    cout<<litera;
    }
plik.close();
return 0;
}
```

Zadaniem programu jest odczytanie i wyświetlenie zawartości pliku **tekst.txt**. Gdybyśmy nie zastosowali w tym przypadku pętli, program wyświetliłby pierwszy znak tekstu, czyli literę p. Zastosowanie instrukcji **while** wyświetli pełny tekst z pliku „programowanie jest fajne".

PRZYKŁAD 36.8

Rozważmy zadanie, w którym w pliku tekstowym **pytanie.txt** umieścimy treść zadania matematycznego, poniżej wypiszemy trzy możliwe odpowiedzi, a ostatnia linijka tekstu będzie zawierała poprawne rozwiązanie.



Rys. 36.4. Zawartość pliku **pytanie.txt**

Program będzie odczytywał treść zadania oraz możliwe wyniki. Użytkownik zostanie poproszony o podanie odpowiedzi. Jeśli odpowiedź będzie poprawna, to zostanie wyświetlony odpowiedni komunikat.

```
#include <iostream>
#include <fstream>

using namespace std;
string pytanie;
string odpA, odpB, odpC;
string wynik;
string odpowiedz;
int main()
{
int nr_wiersza=1;
string wiersz;
fstream plik;
plik.open("c:/dane/pytanie.txt",ios::in);
while(getline(plik,wiersz))
    {
      if(nr_wiersza==1) pytanie = wiersz;
      if(nr_wiersza==2) odpA = wiersz;
      if(nr_wiersza==3) odpB = wiersz; ;
      if(nr_wiersza==4) odpC = wiersz;
      if(nr_wiersza==5) wynik = wiersz;
      nr_wiersza++;
    }
cout<<endl<<pytanie<<endl;
cout<<odpA<<endl;
```

```
cout<<odpB<<endl;
cout<<odpC<<endl;
cout<<"Podaj odpowiedz: ";
cin>>odpowiedz;
if (odpowiedz==wynik)
    {
       cout<<"Prawidlowa odpowiedz"<<endl;
    }
else cout<<"Zle! Poprawna odpowiedzia jest: "<<wynik;
plik.close();
return 0;
}
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Omów proces zapisywania danych do pliku.
2. Omów tryby otwarcia pliku.
3. W jaki sposób można odczytać dane z pliku?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Napisz program, który pobierze z klawiatury wartości 20-elementowej tablicy, a następnie policzy ich średnią arytmetyczną i wpisze wynik do pliku **`srednia.txt`**.
2. Do pliku o nazwie dane.txt wczytaj trzy liczby całkowite: **`a`**, **`b`**, **`c`**. Te liczby wykorzystaj jako współczynniki równania kwadratowego $ax^2+bx+c = 0$. Wyniki rozwiązanego równania wpisz do pliku **`rozwiązanie.txt`**.
3. Napisz program sprawdzający, czy trzy podane przez użytkownika punkty (na płaszczyźnie) są współliniowe. Współrzędne podaje użytkownik (dla każdego dwie współrzędne). Obie współrzędne należą do przedziały <–100,100>. Wartości podane przez użytkownika zapisz w pliku **`PUNKTY.TXT`** wraz z informacją, czy punkty są współliniowe, czy nie.

VIII. Skrypty po stronie klienta – JavaScript

- Podstawy JavaScript
- Okna dialogowe
- Zmienne i ich typy
- Operatory
- Instrukcje sterujące
- Funkcje
- Obiekty w JavaScript
- Zdarzenia
- Obsługa formularzy

37 Podstawy JavaScript

ZAGADNIENIA

- Charakterystyka języka JavaScript
- Różnice między JavaScript i Java
- Zastosowanie JavaScript
- Sposoby umieszczania skryptu w dokumencie HTML

Wiemy już, że HTML jest językiem odpowiedzialnym za treść wyświetlaną na stronie internetowej, natomiast CSS nadaje tej treści odpowiedni wygląd. Język JavaScript pozwala natomiast na dynamiczną zmianę strony internetowej po stronie przeglądarki.

JavaScript jest językiem skryptowym dla dokumentów internetowych. Skrypty napisane za pomocą JavaScript mogą być umieszczane bezpośrednio na stronach HTML. Są one wykonywane za pomocą przeglądarki internetowej. Język daje obszerne możliwości wzbogacania stron o interesujące elementy, np. zdefiniowanie reakcji na działania zainicjowane przez czytelnika. Niektóre efekty jeszcze niedawno były możliwe jedynie za pomocą CGI, a teraz jesteśmy w stanie zbudować wyrafinowane konstrukcyjnie strony za pomocą JavaScript.

JavaScript i Java nie są tymi samymi językami – to dwie odrębne techniki programowania w internecie. Java jest językiem programowania, natomiast JavaScript jest językiem skryptowym. Java to wywodzący się z C++ w pełni obiektowy język programowania. Za jego pomocą można tworzyć aplikacje niezależne od platformy sprzętowej. JavaScript jest interpretowanym językiem służącym do tworzenia niewielkich programów rozszerzających możliwości HTML-a w zakresie opisu stron.

Tabela 37.1. Podstawowe różnice między językami Java oraz JavaScript

JavaScript	Java
Język jest interpretowany na komputerze klienta	Język jest kompilowany do tzw. b-kodu, wykonywanego za pomocą wirtualnej maszyny Javy na komputerze klienta
Język jest oparty na predefiniowanych obiektach, nie pozwala na stosowanie mechanizmów programowania obiektowego	Język jest zorientowany obiektowo z obsługą wszystkich mechanizmów obiektowości

JavaScript	Java
Kod programu jest zagnieżdżony w kodzie HTML	Kod programu jest niezależny od kodu HTML i znajduje się w oddzielnych plikach
Zmienne i typy nie muszą być deklarowane przed użyciem	Zmienne i ich typ muszą być zadeklarowane przed ich użyciem
Odwołania do obiektów i funkcji są wykonywane podczas uruchamiania programu	Wszystkie odwołania do obiektów i funkcji są sprawdzane na etapie kompilacji

Skrypty tworzone w JavaScript zapewniają interaktywność strony przez reagowanie na zdarzenia, tworzenie elementów nawigacji oraz sprawdzenie poprawności formularzy.

Zastosowania języka JavaScript to:

- możliwość modyfikacji wyglądu bieżącego dokumentu,
- wyświetlanie okien dialogowych,
- sprawdzanie poprawności formularza,
- dostęp do daty i czasu.

Skrypty JavaScript mogą być umieszczane wewnątrz kodu HMTL, są to tzw. skrypty osadzone. Aby umieścić skrypt w HTML 4.01, należy użyć znacznika **<script>** w następujący sposób:

```
<script type="text/javascript">
kod skryptu
</script>
```

W przypadku HTML 5.0 parametr **type** nie jest już używany. Skrypt osadzony można umieszczać w nagłówku lub w ciele strony.

Drugą metodą umieszczenia skryptu w kodzie jest zapisanie go w oddzielnym pliku i użycie znacznika **<script>** z parametrem **src**:

```
<script src="lokalizacja skryptu">
kod skryptu
</script>
```

Plik ze skryptem ma rozszerzenie **js**.

W jednym dokumencie można naraz umieścić kilka skryptów, zarówno osadzonych, jak i zewnętrznych.

Ważne, aby wraz z wejściem HTML 5 do znacznika **<script>** dodano dwa atrybuty logiczne (nie mają wartości):

- async,
- defer.

Zastosowanie atrybutu **async** poskutkuje asynchronicznym wykonaniem skryptu, czyli jest ono niezależne od czasu załadowania całej treści kodu. Oznacza to, że w razie załadowania skryptu z zewnętrznego pliku będzie on wykonany podczas dalszego wczytywania źródła strony.

Atrybut **defer** spowoduje, że przeglądarka wykona zewnętrzny skrypt dopiero wtedy, gdy kod źródłowy strony zostanie wczytany w całości do przeglądarki.

Jeśli nie podamy jednego z powyższych atrybutów, to wykonanie skryptu zostanie wykonane w linii dołączenia, zanim przeglądarka wczyta dalszą część kodu programu.

Podczas pisania skryptów w JavaScript ważne jest wyłapanie błędów i literówek w pisanych poleceniach. Niestety, obsługa błędów w JavaScript jest zależna od rodzaju przeglądarki.

Aby wyszukać błędy w poszczególnych przeglądarkach, należy wybrać określone polecenia:

- w Mozilla Firefox – **Narzędzia / Konsola błędów**;
- w Google Chrome – **Narzędzia / Konsola JavaScript**;
- w Opera – **Narzędzia / Zaawansowane / Konsola błędów**.

Instrukcje umieszczane w skrypcie mogą być pisane w jednym lub w kilku wierszach. Średnik jest ogranicznikiem instrukcji, ale nie jest wymagany. Jeżeli w jednym wierszu zostanie zapisane kilka instrukcji, to trzeba je oddzielić średnikiem.

Słowa kluczowe są pisane małymi literami, a nazwy obiektów wbudowanych – od wielkiej litery. Nazwy obiektów DOM są pisane małymi literami, ale w nazwach metod danych obiektów mogą być małe i wielkie litery. Definiowanym zmiennym, funkcjom bądź obiektom można nadawać dowolne nazwy.

Komentarze mogą być umieszczane w dowolnym miejscu kodu. Jeśli komentarzem jest tekst w jednym wierszu, używamy znaku **//**:

```
//Komentarz do instrukcji
```

Jeżeli komentarz zawiera wiele wierszy, używamy **/*** do rozpoczęcia komentarza i ***/** do zakończenia komentarza.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest JavaScript?
2. Wymień sposoby umieszczania skryptu w dokumencie HTML.
3. Czym różni się JavaScript od Javy?
4. Opisz sposoby wprowadzania komentarzy w skrypcie.
5. Do czego służą atrybuty **async** i **defer**?
6. W jaki sposób w danej przeglądarce można zapoznać się z błędami w napisanym kodzie?

38 Wyświetlanie komunikatów na ekranie. Okna dialogowe

ZAGADNIENIA

- Działanie instrukcji `document.write`
- Formatowanie wyświetlanego tekstu
- Definicja okna dialogowego
- Rodzaje okien dialogowych

Aby wyświetlić dowolny tekst na ekranie przeglądarki, używamy instrukcji **`document.write`**:

```
document.write("Komunikat");
```

PRZYKŁAD 38.1

```
<html>
<head>
<meta charset="utf-8">
<title>Skrypt Nr 1</title>
</head>
<body>
<script>
document.write("To mój pierwszy skrypt");
</script>
</body>
</html>
```

Powyższy skrypt spowoduje, że na ekranie pojawi się napis: „To mój pierwszy skrypt”.

Argumenty funkcji **`write`** są traktowane przez przeglądarkę jak tekst w HTML-u. Oznacza to, że w wyświetlanych komunikatach można wstawiać zarówno dowolne znaczniki formatujące tekst, jak i znaczniki przejścia do nowego wiersza itd.

PRZYKŁAD 38.2

```
<html>
<head>
<meta charset="utf-8">
<title>Skrypt Nr 1</title>
```

```
</head>
<body>
<script>
document.write("<h1>To mój pierwszy <b>skrypt</b></h1>");
document.write("<br>Każdy kolejny <i>komunikat</i> może być
sformatowany znacznikami HTML");
</script>
</body>
</html>
```

Efekt jest następujący:

To mój pierwszy skrypt

Każdy kolejny *komunikat* może być sformatowany znacznikami HTML

Rys. 38.1. Formatowanie tekstu w JavaScript

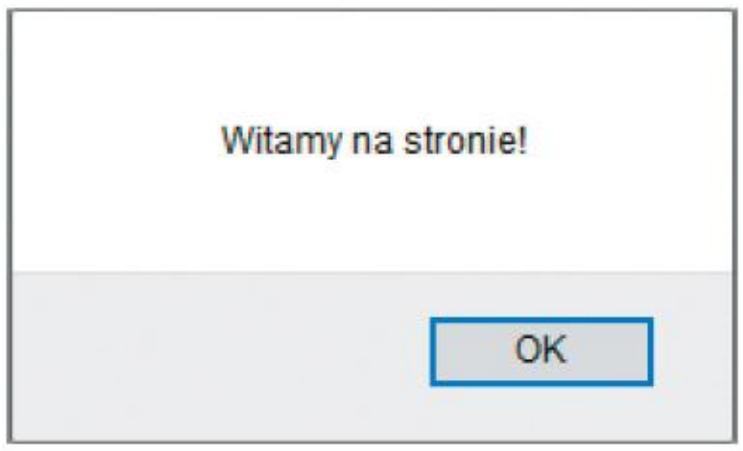
Okno dialogowe to narzędzie, które pozwala na interakcję z użytkownikiem. W języku JavaScript istnieją trzy rodzaje okien dialogowych:

- informacyjne,
- decyzyjne,
- tekstowe.

Okna informacyjne są przeznaczone do przekazania określonej informacji i nie mają wpływu na dalsze działanie skryptu. Do wyświetlenia tego rodzaju okna służy metoda **alert**:

```
alert("Witamy na stronie!");
```

Powyższa instrukcja spowoduje wyświetlenie okna dialogowego z tekstem „Witamy na stronie!" oraz przyciskiem OK (rys. 38.2).



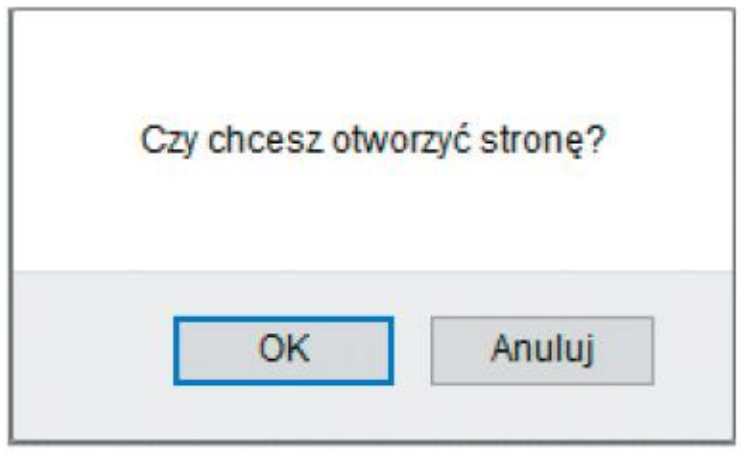
Rys. 38.2. Przykład okna informacyjnego

Okno decyzyjne służy do wyświetlenia treści komunikatu oraz dwóch przycisków – **OK** i **Anuluj**, które po wciśnięciu zwracają wartość logiczną **true** lub **false**.

Postać przykładowej instrukcji:

```
confirm("Czy chcesz otworzyć stronę?");
```

Spowoduje ona wyświetlenie następującego okna dialogowego:

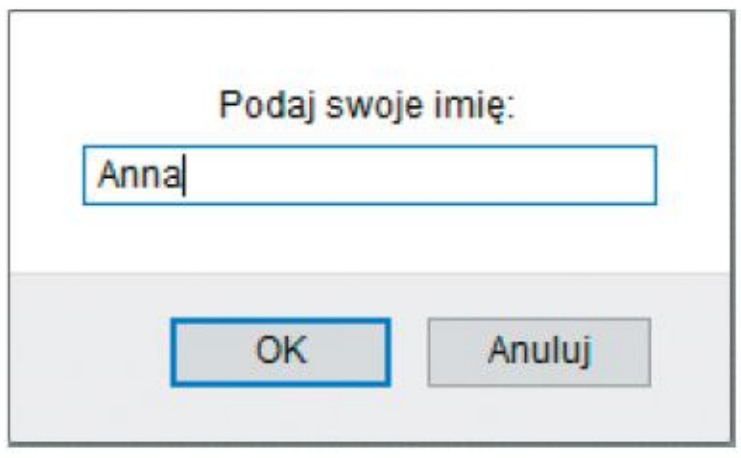


Rys. 38.3. Przykład okna decyzyjnego

Okna tekstowe są przeznaczone do wyświetlenia komunikatu oraz pola umożliwiającego wprowadzanie danych przez użytkownika. Do utworzenia takiego okna dialogowego służy metoda **prompt**:

```
prompt("Podaj swoje imię: ");
```

Na ekranie przeglądarki pojawi się następujące okno:



Rys. 38.4. Przykład okna tekstowego

SPRAWDŹ SWOJĄ WIEDZĘ

1. Podaj postać instrukcji **document.write**.
2. Opisz, jak można sformatować tekst wyświetlany przez skrypt.
3. Omów rodzaje okien dialogowych w JavaScript.
4. W jaki sposób można wyświetlić okno dialogowe na stronie?

39 Zmienne i ich typy

ZAGADNIENIA

- Definicja zmiennej
- Deklarowanie zmiennej
- Przypisywanie wartości do zmiennej
- Rodzaje typów danych w JavaScript
- Określanie typów danych
- Dane liczbowe wprowadzane w skrypcie

Zmienna to miejsce w pamięci, w którym są przechowywane dane i wyniki. Każda zmienna ma swoją nazwę. Na uwagę zasługuje to, że w JavaScript nie trzeba deklarować typu zmiennej przed jej użyciem. Typ jest przypisany do zmiennej po nadaniu jej wartości. Ponadto typ zmiennej może się zmienić podczas działania skryptu.

Nazwa zmiennej może zawierać tylko litery a–z, A–Z, cyfry oraz symbole $ i znak podkreślenia „_”. W trakcie deklarowania zmiennych należy pamiętać, że w nazwach zmiennych nie wolno:

- zaczynać nazwy od cyfry;
- stosować słów zarezerwowanych, tzw. keywords;
- stosować spacji.

UWAGA

Wielkość liter w nazwach ma znaczenie (ang. *case sensitive*), czyli suma oraz Suma są różnymi zmiennymi.

Zmienne deklaruje się za pomocą słowa kluczowego **var** poprzedzającego nazwę zmiennej. Aby zadeklarować zmienną, należy użyć zapisu:

```
var nazwa_zmiennej;
```

PRZYKŁAD 39.1

```
<html>
<head>
<meta charset="utf-8">
<title>Definiowanie zmiennych</title>
```

```
</head>
<body>
<script>
    var tekst="Twoja liczba wynosi";
    var liczba=25;
    document.write("Zadeklarowaliśmy dwie zmienne. jedna jest
    tekstem, a druga liczbą.<br>");
    document.write(tekst+" "+liczba);
</script>
</body>
</html>
```

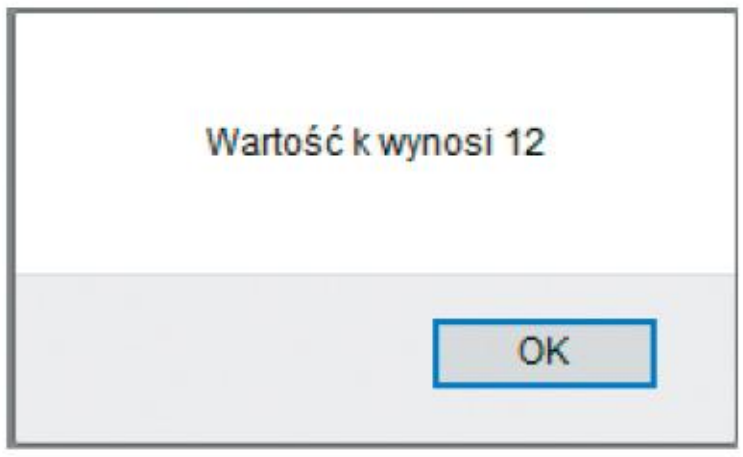
W skrypcie zadeklarowano dwie zmienne: **liczba** (przechowuje liczbę 25) i **tekst** (przechowuje ciąg znaków). Aby w wyświetlanym komunikacie połączyć tekst ze zmienną, należy użyć znaku „**+**”.

PRZYKŁAD 39.2

Operator „+” można również zastosować w instrukcji alert.

```
<html>
<head>
<meta charset="utf-8">
<title>definiowanie zmiennych</title>
</head>
<body>
<script>
    var x=5;     //definicja zmiennej x
    var y=7;     //definicja zmiennej y
    var k;    //deklaracja zmiennej k
    k=x+y;    //instrukcja przypisania
    alert("wartość k wynosi "+k);
</script>
</body>
</html>
```

W skrypcie zdefiniowano dwie zmienne: **x** oraz **y**. Definicja różni się od deklaracji tym, że podczas tworzenia zmiennej nadajemy jej określoną wartość. Zmienna **k** została zadeklarowana, a następnie za pomocą instrukcji przypisana – wczytała do niej wynik dodawania liczb **x** oraz **y**. Efektem skryptu będzie wyświetlenie okna informacyjnego następującej postaci:



Rys. 39.1. Połączenie tekstu z wartością zmiennej w oknie informacyjnym

Jak widać w poprzednich przykładach, operator „+" ma zastosowanie w łączeniu tekstu z wartością zmiennej.

Tym razem rozważmy skrypt, którego zadaniami będą pobranie z okna tekstowego wartości dla dwóch zmiennych, wykonanie na nich działania dodawania, a następnie wyświetlenie wyniku w oknie przeglądarki.

UWAGA

W dalszej części podręcznika będzie prezentowana tylko część skryptowa witryny.

PRZYKŁAD 39.3

```
<script>
    var x=prompt("Podaj x:");
    var y=prompt("Podaj y:");
    var suma=x+y;
    document.write("Suma liczb "+x+" oraz "+y+" wynosi "+suma);
</script>
```

Aby wczytać wartości dla zmiennych x oraz y, należy zastosować znaną już instrukcję prompt. Następnie zmiennej suma przypisujemy instrukcję dodawania liczb. Za pomocą instrukcji document.write wynik zostaje wyświetlony w oknie przeglądarki.

Niestety, po podaniu przykładowych wartości 4 oraz 5 otrzymujemy następujący wynik:

```
Suma liczb 4 oraz 5 wynosi 45
```

Okazuje się, że wartości wpisywane do okna tekstowego są traktowane jako ciągi znaków, czyli tekst. Należy zatem przekonwertować je na liczby, np. za pomocą funkcji **Number**.

```
<script>
    var x=Number(prompt("Podaj x:"));
    var y=Number(prompt("Podaj y:"));
    var suma=x+y;
    document.write("Suma liczb "+x+" oraz "+y+"wynosi "+suma);
</script>
```

Po wprowadzeniu poprawek otrzymamy poprawny wynik:

```
Suma liczb 4 oraz 5 wynosi 9
```

W języku JavaScript można stosować kilka typów danych: liczbowy, łańcuchowy, logiczny, obiektowy oraz typy specjalne.

Typ liczbowy obejmuje różnego rodzaju liczby. JavaScript nie uwzględnia podziału na liczby całkowite i zmiennoprzecinkowe. Umożliwia wprowadzanie liczb w postaci dziesiętnej, ósemkowej lub szesnastkowej. Liczby dziesiętne zapisujemy w postaci tradycyjnej, np. 8,123, -54 itd. Liczby w systemie szesnastkowym należy poprzedzić znakami **0x** lub **0X**, natomiast w zapisie ósemkowym przed liczbą wpisuje się zero.

Typ łańcuchowy to dowolny ciąg znaków, które należy umieścić w cudzysłowie lub między apostrofami. Może zawierać znaki specjalne.

Typ logiczny może przyjmować tylko wartość **TRUE** lub **FALSE** (prawda lub fałsz). Ten typ jest wykorzystywany podczas tworzeniu wyrażeń logicznych, porównywania wartości lub w celu sprawdzenia, czy dana operacja zakończyła się sukcesem.

Typ obiektowy służy do reprezentacji obiektów.

Typy specjalne to **NULL** oraz **UNDEFINED**. Pierwszy z nich oznacza wartość pustą, a drugi – wartość niezdefiniowaną.

W języku JavaScript istnieją również złożone typy danych, w których można przechowywać więcej niż jedną wartość. Są nimi:

- typ **OBJECT** – służy do reprezentacji obiektów;
- typ **ARRAY** – typ tablicowy.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Omów rodzaje typów danych.
2. Do czego służą typy specjalne?
3. Czy typ zmiennej może się zmienić podczas działania skryptu?
4. W jaki sposób określa się typ zmiennej?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz skrypt, który poprosi o podanie trzech liczb, obliczy ich średnią, a wynik wyświetli w oknie dialogowym.
2. Utwórz skrypt, który poprosi o podanie imienia i nazwiska użytkownika, a następnie w oknie przeglądarki wyświetli jego wizytówkę:
   Imię: Anna
   Nazwisko: Kowalska

40 Operatory

ZAGADNIENIA

- Rodzaje operatorów w JavaScript
- Konkatenacja
- Przypisanie do zmiennej
- Działania operatorów arytmetycznych, logicznych i bitowych
- Działania operatora inkrementacji i dekrementacji

Wyróżnia się następujące operatory:

- arytmetyczne,
- logiczne,
- bitowe,
- przypisania,
- porównania,
- łańcuchowy.

Operatory arytmetyczne

Są to standardowe operatory służące do wykonywania działań matematycznych na zmiennych:

- operator dodawania **+**
- operator odejmowania **–**
- operator mnożenia *****
- operator dzielenia **/**
- operator dzielenie modulo (wyznaczanie reszty z dzielenia) **%**.

PRZYKŁAD 40.1

```
<script>
var x=7;
var y=2;
document.write(x+" + "+y+" = "+(x+y)+"<br>");
document.write(x+" - "+y+" = "+(x-y)+"<br>");
document.write(x+" * "+y+" = "+(x*y)+"<br>");
document.write(x+" / "+y+" = "+(x/y)+"<br>");
document.write(x+" % "+y+" = "+(x%y)+"<br>");
</script>
```

Wynikiem tego skryptu są następujące wyniki:

```
7 + 2 = 9
7 - 2 = 5
7 * 2 = 14
7 / 2 = 3.5
7 % 2 = 1
```

Rys. 40.1. Zastosowanie operatorów arytmetycznych

Wyrażenia **x+y**, **x-y** itd. muszą być zapisane w nawiasach. Znak „**+**" z lewej strony wyrażenia służy do połączenia tekstu z wartością zmiennej.

Operatory porównania

Operatory porównania służą do porównywania dwóch argumentów. Wynikiem jest wartość **TRUE** lub **FALSE**.

Tabela 40.1. Operatory porównania w JavaScript

Symbol	Składnia	Działanie
!=	a!=b	Wynikiem jest wartość **TRUE**, jeżeli zmienne nie są równe
==	a==b	Wynikiem jest wartość **TRUE**, jeżeli zmienne są równe
===	a===b	Wynikiem jest wartość **TRUE**, jeżeli zmienne są równe i tego samego typu
>	a>b	Wynikiem jest wartość **TRUE**, jeżeli zmienna **a** jest większa od zmiennej **b**
>=	a>=b	Wynikiem jest wartość **TRUE**, jeżeli zmienna **a** jest większa lub równa zmiennej **b**
<	a<b	Wynikiem jest wartość **TRUE**, jeżeli zmienna **a** jest mniejsza od zmiennej **b**
<=	a<=b	Wynikiem jest wartość **TRUE**, jeżeli zmienna **a** jest mniejsza lub równa zmiennej **b**

PRZYKŁAD 40.2

```
<script>
var x=7;
var y=2;
document.write("Wynikiem porównania x &lt; y jest
"+(x<y)+"<br>");
```

```
var a=6;      //a jest liczbą
var b="6";    //b jest tekstem
document.write("Wynikiem porównania a == b jest
"+(a==b)+"<br>");
document.write("Wynikiem porównania a === b jest
"+(a===b)+"<br>");
</script>
```

Wynikiem działania skryptu jest wyświetlenie trzech komunikatów informujących użytkownika, jaki typ logiczny zwracają operatory **<**, **==** i **===**. Okazuje się, że dwa ostatnie zwracają przeciwne wartości. Operator == zwraca wartość **TRUE**, a operator **===** zwraca **FALSE**, ponieważ każda ze zmiennych jest innego typu.

```
Wynikiem porównania x < y jest false
Wynikiem porównania a == b jest true
Wynikiem porównania a === b jest false
```

Rys. 40.2. Zastosowanie operatorów relacji

Operatory logiczne

Argumentami operatorów logicznych muszą być wyrażenia mające wartość logiczną **TRUE** lub **FALSE**. Rodzaje operatorów logicznych opisano w tabeli 40.2.

Tabela 40.2. Operatory logiczne w JavaScript

Symbol	Nazwa	Składnia	Działanie
&&	koniunkcja	a&&b	Operator zwraca wartość **TRUE**, jeżeli zmienne a i b są prawdziwe
||	alternatywa	a||b	Operator zwraca wartość **TRUE**, jeżeli jedna ze zmiennych jest prawdziwa
!	negacja	!a	Operator neguje wyrażenie a

PRZYKŁAD 40.3

```
<script>
var x=7;
var y=2;
document.write((x<y && y==2)+"<br>");
document.write((x<y || y==2)+"<br>");
</script>
```

Skrypt wyświetla wartość logiczną zdania z użyciem operatora koniunkcji oraz alternatywy. Pierwsze zdanie ma wartość **FALSE**, ponieważ ma wartość logiczną 0, a drugie ma wartość 1. Aby koniunkcja była prawdziwa, oba zdania muszą być typu **TRUE**, czyli mieć wartość logiczną 1. Drugie zdanie ma wartość logiczną **TRUE**. Aby alternatywa była prawdziwa, wystarczy, żeby tylko jedno ze zdań składowych miało wartość 1.

Operatory bitowe

Służą one do wykonywania operacji na bitach w arytmetyce binarnej. Operatory funkcjonujące w JavaScript opisano w tabeli 40.3.

Tabela 40.3. Operatory bitowe w JavaScript

Symbol	Składnia	Nazwa	Działanie
&	a&b	operator bitowy **AND**	Wyświetla 1, jeżeli obie zmienne są jedynkami
^	x^y	operator bitowy **XOR**	Wyświetla 1, jeżeli jedna ze zmiennej (ale nie obie jednocześnie) jest jedynką
|	x|y	operator bitowy **OR**	Wyświetla 1, jeżeli jedna ze zmiennej jest jedynką
<<	x<<y	przesunięcie w lewo	Przesunięcie bitów w lewo o określoną liczbę miejsc
>>	x>>y	przesunięcie w prawo	Przesunięcie bitów w prawo o określoną liczbę miejsc

PRZYKŁAD 40.4

```
<script>
var x=8;
var y=15;
document.write("Liczba "+x+"= 1000");
document.write("<br>Liczba "+y+"= 1111");
document.write("<br>8 & 15 = " +(x&y)+", czyli 1000");
document.write("<br>8 ^ 15 = " +(x^y)+", czyli 0111");
document.write("<br>8 | 15 = " +(x|y)+", czyli 1111");
document.write("<br>8 << 2 = " +(x<<2)+", czyli 100000");
</script>
```

Argumenty są konwertowane do 32-bitowej liczby całkowitej i wyrażone binarnie. Każdy bit w pierwszym argumencie jest dopasowany do odpowiedniego bitu w drugim argumencie, czyli pierwszy bit do pierwszego bitu, drugi bit to drugiego bitu itd. Operator działa na każdej parze bitów i wynik jest bitowy. Efekt działania skryptu jest następujący:

```
Liczba 8 = 1000
Liczba 15 = 1111
8 & 15 = 8, czyli 1000
8 ^ 15 = 7, czyli 0111
8 | 15 = 15, czyli 1111
8 << 2 = 32, czyli 100000
```

Rys. 40.3. Efekt zastosowania operatorów bitowych

Operatory przypisania

Podstawowym **operatorem przypisania** jest znak równości. Instrukcja przypisania ma postać:

```
zmienna = wartość_lub_wyrażenie;
```

PRZYKŁAD 40.5

```
<script>
var x=Number(prompt("Podaj x:"));
var y= Number(prompt("Podaj y:"));
var srednia=(x+y)/2;
document.write("Średnia liczb wynosi: "+srednia);
</script>
```

Skrypt przypisuje zmiennym **x** oraz **y** wartość podaną przez użytkownika, trzecia zmienna jest równa średniej arytmetycznej dwóch poprzednich.

Oprócz często stosowanego operatora **=** istnieją jeszcze inne operatory przypisania (tabela 40.4).

Tabela 40.4. Operatory przypisania w JavaScript

Symbol	Składnia	Działanie
+=	a+=b	Wykonuje działanie a=a+b
-=	a-=b	Wykonuje działanie a=a-b
*=	a*=b	Wykonuje działanie a=a*b
/=	a/=b	Wykonuje działanie a=a/b
%=	a%=b	Wykonuje działanie a=a%b
<=	a<=b	Wykonuje działanie a=a<b
^=	a^=b	Wykonuje działanie a=a^b
|=	a|=b	Wykonuje działanie a=a|b

Symbol	Składnia	Działanie
<<=	a<<=b	Wykonuje działanie a=a<<b
>>=	a>>=b	Wykonuje działanie a=a>>b

PRZYKŁAD 40.6

```
<script>
var a=8;
var b=15;
var c=10;
document.write("<b>Przed działaniem operatorów przypisania:
</b>");
document.write("<br>a= "+a+" b= "+b+" c= "+c);
a+=b;
c*=b;
document.write("<br><b>Po działaniu operatorów przypisania:
</b>");
document.write("<br>a= "+a+" b= "+b+" c= "+c);
</script>
```

Skrypt wykona dwa działania: suma liczb 8 i 15 zostanie przypisana do zmiennej **a**, iloczyn liczb 10 i 15 zostanie przypisany do zmiennej **c**.

Przed działaniem operatorów przypisania:
a= 8 b= 15 c= 10
Po działaniu operatorów przypisania:
a= 23 b= 15 c= 150

Rys. 40.4. Efekt działania skryptu z przykładu 40.6

Często stosuje się operator **inkrementacji**, którego zadaniem jest zwiększenie wartości zmiennej o jeden. Zapisuje się go za pomocą dwóch plusów (**++**).

Drugim operatorem, równie często stosowanym, jest operator **dekrementacji**, który zmniejsza wartość zmiennej o jeden. Zapisuje się go za pomocą dwóch minusów (**--**).

Te operatory można przyłożyć zarówno z lewej, jak i z prawej strony zmiennej:

- **x++** – operator postinkrementacji (operator równoważny zapisowi **x=x+1**);
- **x--** – operator postdekrementacji (operator równoważny zapisowi **x=x-1**);
- **++x** – operator preinkrementacji (operator równoważny zapisowi **x=x+1**);
- **--x** – operator predekremenatcji (operator równoważny zapisowi **x=x-1**).

PRZYKŁAD 40.7

```
<script>
var a=8;
var b=15;
document.write((a++)+"<br>");
document.write(++b);
</script>
```

Efektem kodu jest wyświetlenie wartości zmiennej **a=8** oraz wartości zmiennej **b=16**. Zapis **a++** w instrukcji **document.write** oznacza, że najpierw nastąpi wyświetlenie zmiennej **a**, a później – jej zwiększenie. Natomiast zapis **++b** w instrukcji **document.write** oznacza, że najpierw nastąpi zwiększenie wartości, a potem – jej wyświetlenie.

Konkatenacja łańcuchów znaków

Operator **+** umożliwia łączenie łańcuchów znakowych, czyli **konkatenację**. Podobnie jak dodawanie do zmiennej liczbowej za pomocą operatora **+=**, tak samo można dołączyć jeden ciąg znaków z innym.

PRZYKŁAD 40.8

```
<script>
var imie="Anna";
var nazwisko="Kowalska";
imie+=nazwisko;
document.write("Nazywasz się :"+imie);
</script>
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień nazwy operatorów arytmetycznych i je opisz.
2. Jakie znasz operatory bitowe?
3. Co to są inkrementacja i dekrementacja?
4. Do czego służy operator `%`?

41 Instrukcje sterujące

ZAGADNIENIA

- Postać i zastosowanie instrukcji warunkowej
- Instrukcja przetwarzania warunkowego
- Zastosowani pętli
- Pętle dostępne w JavaScript
- Różnice między instrukcją warunkową a instrukcją switch

Zadaniem instrukcji sterujących jest sterowanie skryptem. Ich zadanie polega na sprawdzeniu określonych warunków poprawnego działania skryptu i odpowiedniej reakcji na ewentualnie błędnie wprowadzone dane przez użytkownika aplikacji.

Do instrukcji sterujących stosowanych w JavaScript należą:

- instrukcje warunkowe – **`if`**...**`else`** oraz **`switch`**;
- pętle – **`for, while, do`**...**`while.`**

Instrukcje warunkowe

Instrukcje warunkowe służą do wykonywania określonych instrukcji zależnie od spełnienia warunków. Podobnie jak w języku C++, istnieją instrukcje warunkowe – niepełna i pełna.

Instrukcja warunkowa niepełna wykonuje dane instrukcje, jeśli warunek przyjmuje wartość **`TRUE`**.

Postać instrukcji:

```
if (warunek)
{
instrukcje;
}
```

PRZYKŁAD 41.1

```
<script>
var liczba=Number(prompt("Podaj liczbę: "));
if (liczba %2==0)
    {
    alert("Liczba "+liczba+" jest parzysta");
    }
</script>
```

Skrypt sprawdza, czy zmienna liczba jest parzysta, zatem czy jest podzielna przez 2, a tym samym – czy reszta z dzielenia przez 2 wynosi 0. Jeśli liczba podana za pomocą okna dialogowego będzie parzysta, to otrzymamy komunikat o tym nas informujący. Jeśli podana wartość nie spełni warunku instrukcji, nie zostanie wyświetlony żaden komunikat.

Instrukcja warunkowa pełna pozwala na sprawdzenie warunku, a następnie – w zależności od jego spełnienia lub nie – na wykonywanie instrukcji.

Postać pełnej instrukcji warunkowej:

```
if (warunek)
{
instrukcje1;
}
else
{
instrukcje2;
}
```

PRZYKŁAD 41.2

```
<script>
var liczba=Number(prompt("Podaj liczbę: "));
if (liczba%2==0)
     {
     alert("Liczba "+liczba+" jest parzysta");
     }
else
     {
     alert("Liczba "+liczba+" jest nieprzysta");
     }
</script>
```

Poprzedni skrypt został rozszerzony o klauzulę **else**, po której wpisujemy instrukcje wykonujące się, jeśli warunek ma wartość **FALSE**. Tym razem, jeśli wpiszemy wartość niepodzielną przez 2, to otrzymamy informację, że podana liczba jest nieparzysta.

W języku JavaScript istnieje jeszcze instrukcja przetwarzania warunkowego, która jest skróconą wersją instrukcji **if...else.** Jeśli warunek ma wartość **TRUE**, to zostanie wykonana pierwsza instrukcja. Jeśli natomiast warunek ma wartość **FALSE**, to zostanie wykonana druga instrukcja.

Postać instrukcji:

```
(warunek)? {instrukcja1;}:{instrukcja2;}
```

PRZYKŁAD 41.3

Skrypt z przykładu 41.2 można napisać z wykorzystaniem instrukcji przetwarzania warunkowego:

```
<script>
var liczba=Number(prompt("Podaj liczbę: "));
var wynik=(liczba % 2==0)? "parzysta":"nieparzysta";
alert("Liczba "+liczba+" jest "+wynik);
</script>
```

Za pomocą okna dialogowego użytkownik zostanie poproszony o podanie liczby. Zmienna **wynik** zwraca wartość „**parzysta**” wtedy, gdy podana liczba jest podzielna przez 2.

Zmienna **wynik** zwraca wartość „**nieparzysta**” wtedy, gdy podana liczba nie jest podzielna przez 2.

Dostępna jest również opcja dokonania kilku kolejnych sprawdzeń z wykorzystaniem klauzuli **else if**:

```
if (wyrunek)
{
//kod wykonywany w przypadku, gdy warunek jest prawdziwy
}
else if (kolejny wyrunek)
{
//kod wykonywany w przypadku, gdy poprzedni warunek jest fałszywy,
a kolejny prawdziwy
}
else if (kolejne warunek)
{
//kod wykonywany w przypadku, gdy poprzedni warunek jest fałszywe,
a kolejny prawdziwy
}
else
{
//kod wykonywany w każdym innym przypadku
}
```

PRZYKŁAD 41.4

```
<script>
var liczba=Number(prompt("Podaj liczbę: "));
if (liczba>0)
```

```
    {
    alert("Liczba "+liczba+" jest dodatnia");
    }
else if(liczba<0)
    {
    alert("Liczba "+liczba+" jest ujemna");
    }
else
    {
    alert("Liczba jest równa 0");
    }
</script>
```

Skrypt sprawdza, czy podana liczba jest dodatnia, ujemna lub równa zero, i wyświetla odpowiedni komunikat.

Kolejnym przykładem zastosowania instrukcji warunkowej jest skrypt, który pobierze trzy wartości liczbowe i w oknie przeglądarki wyświetli największą z nich.

PRZYKŁAD 41.5

```
<script>
var a=Number(prompt("Podaj a: "));
var b=Number(prompt("Podaj b: "));
var c=Number(prompt("Podaj c: "));
document.write("<u>Podałeś liczby:</u> "+a+", "+b+",
"+c+"<br>");
if (a>=b && a>=c)
    {
    document.write("Największą z nich jest "+a);
    }
else if (b>=a && b>=c)
    {
    document.write("Największą z nich jest "+b);
    }
else
    {
    document.write("Największą z nich jest "+c);
    }
</script>
```

Instrukcja SWITCH

Instrukcja wyboru **switch** pozwala w wygodny sposób sprawdzić ciąg warunków i zależnie od wyników porównywania wykonać różne instrukcje.

Postać instrukcji:

```
switch(wyrażenie){
     case wartość1:
           instrukcje1;
           break;
     case wartość2:
           instrukcje2;
           break;
     case wartość3:
           instrukcje3;
           break;
.....
     default:
           instrukcje4;
}
```

Instrukcja **switch** sprawdza wartość wyrażenia i zależnie od wyniku są wykonywane różne instrukcje. Jeśli wynikiem jest **wartość1**, to są wykonywane **intrukcje1** i następuje przerwanie wykonywanie bloku **switch** (przerwanie jest wykonywane przez instrukcję **break**). Jeśli wynikiem jest **wartość2**, to są wykonywane **intrukcje2** i następuje przerwanie wykonywania bloku **switch** itd. Jeśli nie zachodzi żaden z wymienionych przypadków, są wykonywane **instrukcje4** i następuje zakończenie instrukcji **switch**. Blok **default** jest jednak opcjonalny i może zostać pominięty.

Łatwo zauważyć, że jest to odpowiednik złożonej instrukcji **if...else if**.

PRZYKŁAD 41.6

```
<script>
var a=prompt("Podaj liczbę od 1 do 7:");
switch (a)
{
case "1":  document.write("Dziś jest poniedziałek.");
           break;
case "2":  document.write("Dziś jest wtorek.");
           break;
case "3":  document.write("Dziś jest środa.");
           break;
case "4":  document.write("Dziś jest czwartek.");
           break;
```

```
case "5":  document.write("Dziś jest piątek.");
           break;
case "6":  document.write("Dziś jest sobota.");
           break;
case "7":  document.write("Dziś jest niedziela.");
           break;
default:   document.write("Wybrałeś złą liczbę.");
           break;
}
</script>
```

Argumentem instrukcji **switch** jest zmienna **a**, której wartość została wpisana za pomocą okna dialogowego. Wartość zmiennej **a** jest porównywana do wartości występujących po słowach **case**. Jeśli zgodność jest stwierdzona, zostaną wykonane instrukcje występujące w danym bloku **case**. Jeśli nie uda się dopasować wartości wyrażenia do żadnej z wartości występujących po słowach **case**, jest wykonywany blok **default**.

! UWAGA

Przypadkowe pominięcie instrukcji **break** może doprowadzić do nieoczekiwanych wyników i błędów w skrypcie.

Każdą instrukcję SWITCH można zamienić na zwykłą instrukcję warunkową, ale nie odwrotnie, ponieważ w instrukcji IF warunkiem może być wyrażenie określające dany zakres wartości, a nie – pojedynczą liczbę. Zatem skrypt z przykładu 41.6 może mieć taką postać, jak w przykładzie 41.7.

PRZYKŁAD 41.7

```
<script>
var a=prompt("Podaj liczbę od 1 do 7:");
if (a=="1")
    {
    document.write("Dziś jest poniedziałek.");
    }
else if(a=="2")
    {
    document.write("Dziś jest wtorek.");
    }
else if(a=="3")
    {
    document.write("Dziś jest środa.");
    }
```

```
else if(a=="4")
    {
    document.write("Dziś jest czwartek.");
    }
else if(a=="5")
    {
    document.write("Dziś jest piątek.");
    }
else if(a=="6")
    {
    document.write("Dziś jest sobota.");
    }
else if(a=="7")
    {
    document.write("Dziś jest niedziela.");
    }
else
    {
    document.write("Wybrałeś złą liczbę.");
    }
</script>
```

Pętla FOR

Zadaniem pętli **FOR** jest powtarzanie instrukcji określoną liczbę razy. Składnia tej instrukcji jest identyczna jak w języku C++:

```
for (wyr_początkowe; wyr_końcowe; wyr_modyfikujące)
{
instrukcje;
}
```

Wyrażenie początkowe określa wartość początkową zmiennej, która jest licznikiem liczby wykonywania pętli. Wyrażenie końcowe to warunek zakończenia działania pętli. Wyrażenie modyfikujące jest określeniem sposobu modyfikacji licznika.

PRZYKŁAD 41.8

```
<script>
for (var i=1;i<=5;i++)
{
document.write("Oto mój skrypt!<br>");
}
</script>
```

Efektem działania skryptu jest pięciokrotne wyświetlenie na ekranie przeglądarki tekstu „**Oto mój skrypt**”. Licznikiem pętli jest zmienna **i**, której wartość początkowa wynosi 1. Wartość końcowa licznika wynosi 5, a ostatni argument pętli zwiększa licznik o 1. Oznacza to, że dopóki wartość zmiennej **i** jest mniejsza bądź równa 5, zwiększa **i** o 1 oraz wykonuje instrukcję **document.write**.

Kolejnym przykładem zastosowaniem pętli może być skrypt wyświetlający na ekranie liczby od 1 do 20 i obliczający ich sumę.

PRZYKŁAD 41.9

```
<script>
var suma=0;
for (var i=1;i<=20;i++)
    {
    document.write(i+"<br>");
    suma+=i;  //równoważny zapis suma=suma+i;
    }
document.write("Suma liczb wynosi: "+suma);
</script>
```

PRZYKŁAD 41.10

Innym przykładem zastosowania **FOR** jest skrypt, który wczytuje dwie liczby **a** i **b** za pomocą funkcji **prompt**, a następnie wyświetla liczby nieparzyste z zakresu **<a,b>** oraz oblicza ich liczbę, a wynik wyświetla w oknie przeglądarki.

```
<script>
var i;
var ilosc=0;
var a=prompt("Podaj a:");
var b=prompt("Podaj b:");
if(a<b)
    {
    for (i=a;i<=b;i++)
       {
       if (i%2!=0)
          {
          document.write(i+"<br>");
          ilosc++;
          }
       }
    document.write("Ilość liczb w podanym zakresie: "+ilosc);
    }
```

```
if(a>b)
    {
    for (i=a;i>=b;i--)
       {
       if (i%2!=0)
          {
          document.write(i+"<br>");
          ilosc++;
          }
       }
    document.write("Ilość liczb w podanym zakresie: "+ilosc);
    }
if (a==b)
    {
    document.write("a=b więc nie ma przedziału");
    }
</script>
```

Przed uruchomieniem działań w pętli w pierwszej kolejności zachodzi sprawdzenie, która z liczb a, b jest większa. Jeśli a<b, liczby są wyświetlane rosnąco, jeśli a>b, wartości są wyświetlane malejąco. W przypadku, gdy a=b, zostaje wyświetlony komunikat, że taki przedział nie istnieje.

Pętla `WHILE`

Pętla ma następującą składnię:

```
while (wyr_warunkowe)
{
instrukcje;
}
```

Jeśli warunek w pętli jest spełniony, to zostaną wykonane instrukcje wewnątrz pętli. Jeśli warunek przyjmuje wartość **TRUE**, to pętla będzie wykonywana do czasu, aż warunek osiągnie wartość **FALSE**. Może zatem się zdarzyć, że instrukcje nie wykonają się ani razu, jeśli za pierwszym razem warunek przyjmie wartość **FALSE**.

PRZYKŁAD 41.11

```
<script>
var i=1;
while(i<=5)
{
document.write("Oto mój skrypt!<br>");
```

```
i++;
}
</script>
```

Podobnie jak w przykładzie 41.8, zadaniem skryptu jest pięciokrotne wyświetlenie tekstu. Tym razem zastosowano pętle **WHILE**. Przed wejściem do pętli wartość zmiennej **i** wynosi 1, warunek pętli zatem jest spełniony i zostaną wykonane instrukcje wewnętrzne.

Zarówno pętlę **FOR**, jak i pętlę **WHILE** można zastosować do wygenerowania liczb z danego zakresu i dodatkowo wstawić je w tabelę.

PRZYKŁAD 41.12

Przykład zawiera pełny kod, ponieważ wstawiana tabela jest sformatowana za pomocą styli CSS, które umieszczono w części nagłówkowej strony. Do wygenerowania liczb została użyta funkcja **Random** obiektu **Math**. Generuje ona losowo liczby z zakresu <0,1>. Jeśli chcemy zmienić zakres losowania, to należy zastosować taką samą zasadę, jak w języku C++. Zapis **Math.Random()*a+b** oznacza, że będziemy losowali liczby z zakresu **<a,b>**. Dodatkowo musimy zastosować funkcję **parseInt()**, która przekształca wylosowywane liczby w typ całkowity. Bez użycia tej funkcji otrzymalibyśmy liczby zmiennoprzecinkowe.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Tworzenie tabel za pomocą pętli</title>
<style>
table,tr,td
    {
    border:1px solid blue;
    }
table
    {
    width:400px;
    text-align: center;
    }
</style>
</head>
<body>
<script>
var i=1,a;
document.write("<table><tr>");
while(i<=10)
```

```
{
a=parseInt(Math.random()*100+1);
document.write("<td>"+a+"</td>");
i++;
}
document.write("</tr></table>");
</script>
</body>
</html>
```

Efektem skryptu z przykładu 41.12 będzie tabela z wylosowanymi liczbami (rys. 41.1).

100	64	64	82	33	54	50	79	31	74

Rys. 41.1. Efekt skryptu z przykładu 4.12

Pętla DO...WHILE

Pętla **do...while** pozwala na wykonanie instrukcji wewnętrznych co najmniej raz, zanim zostanie sprawdzony warunek logiczny.

Składnia pętli jest następująca:

```
do
{
instrukcje;
}
while(warunek)
```

PRZYKŁAD 41.13

```
<script>
var i=1;
do
    {
    document.write("Oto mój skrypt!<br>");
    i++;
    }
while (i<=5);
</script>
```

Zadaniem skryptu jest pięciokrotne wyświetlenie tekstu: „**Oto mój skrypt!**”, tak jak w przykładach 41.8 i 41.11. Przed wejściem do pętli wartość zmiennej **i** wynosi 1. Najpierw zostanie wyświetlony tekst i wartość licznika będzie zwiększona o 1, a dopiero potem zostanie sprawdzony warunek pętli.

UWAGA

Każdą pętlę **FOR** można przekształcić w **WHILE** bądź **DO...WHILE**, natomiast odwrotna zasada nie zachodzi. **FOR** jest pętlą stosowaną, gdy znamy liczbę wykonywanych iteracji, natomiast pozostałe dwie pętle mogą wykonywać się nieokreśloną liczbę razy.

Rozważmy zadanie z przykładu 41.12, tylko zastosujemy pętlę **DO...WHILE**. Wyświetlane liczby zostaną umieszczone w tabeli z jedną kolumną i dziesięcioma wierszami.

PRZYKŁAD 41.14

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Tworzenie tabel za pomocą pętli</title>
<style>
table,tr,td
    {
    border:1px solid blue;
    }
table
    {
    width:50px;
    text-align: center;
    }
</style>
</head>
<body>
<script>
var i=1,a;
document.write("<table>");
do
    {
    a=parseInt(Math.random()*100+1);
    document.write("<tr><td>"+a+"</td></tr>");
    i++;
    }while(i<=10);
document.write("</table>");
</script>
</body>
</html>
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jaka jest różnica między instrukcją warunkową a instrukcją przetwarzania warunkowego?
2. Podaj postać wszystkich instrukcji warunkowych.
3. Jaka jest różnica w zastosowaniu pętli **FOR** i pętli **WHILE**?
4. Jaka jest różnica w działaniu pętli **WHILE** i pętli **DO...WHILE**?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz skrypt, który wczyta za pomocą okna dialogowego wartość zmiennej x. Za pomocą **FOR** wyświetl liczby z zakresu od 1 do liczby x, a następnie oblicz, ile liczb parzystych mieści się w tym przedziale. Wykonaj to zadanie z zastosowaniem pętli **WHILE**.
2. Utwórz skrypt, który wygeneruje 20 liczb z przedziału <50,100> i wyświetli je w postaci tabeli złożonej z 20 kolumn i jednego wiersza. Zadanie wykonaj za pomocą pętli **DO...WHILE**.

42 Funkcje

ZAGADNIENIA

- Argumenty funkcji
- Definiowanie funkcji
- Wywoływanie funkcji

JavaScript to język, który zawiera ogromną liczbę wbudowanych funkcji (metod), ale umożliwia również tworzenie własnych.

Przykładem funkcji wbudowanych są znane nam już **`prompt`**, **`alert`**, **`confirm`** (omówione w rozdziale o oknach dialogowych). Ponadto istnieją inne funkcje wbudowane, np.:

- **`parseInt()`** – funkcja pobiera argument typu tekstowego i konwertuje go na liczbę;
- **`parseFloat()`** – funkcja działa podobnie jak **`parseInt()`**, ale działa także na ułamkach;
- **`Number()`** – funkcja przekonwertuje argument na obiekt numeryczny;
- **`isNaN()`** – funkcja sprawdza, czy wartość podana jako parametr nie jest liczbą.

Funkcja to grupa instrukcji służących do wykonania określonych, powtarzających się zadań.

Definicja funkcji musi zawierać:

- słowo kluczowe **`function`**;
- nazwę funkcji;
- listę argumentów w nawiasach okrągłych i rozdzielonych przecinkami lub puste nawiasy okrągłe;
- implementację.

Nazwa funkcji powinna zaczynać się od litery. Kolejnymi znakami nazwy mogą się stać litery, cyfry lub znak podkreślenia. W nazwach funkcji są rozróżniane małe i wielkie litery. W JavaScript przestrzega się ogólnej zasady tworzenia nazw funkcji: każde słowo w nazwie zaczyna się wielką literą, oprócz pierwszego pisanego od małej litery.

Lista argumentów musi być zawarta między nawiasami okrągłymi; poszczególne argumenty są rozdzielone przecinkami. Implementacja to ciąg instrukcji zawartych między nawiasami klamrowymi i wykonywanych przez funkcję. W ciele funkcji może się znajdować dowolna liczba instrukcji.

Ogólna definicja funkcji:

```
function nazwa(arg1, arg2, …, argN)
{
instrukcje;
}
```

PRZYKŁAD 42.1

```
function srednia(a,b)
{
document.write("Średnia liczb "+a+" oraz "+b+" wynosi
"+((a+b)/2));
}
```

Funkcja o nazwie **srednia** ma dwa argumenty i jej zadaniem jest obliczenie oraz wyświetlenie na ekranie przeglądarki średniej dwóch liczb.

Aby wywołać zdefiniowaną funkcję, wystarczy podać jej nazwę, a w nawiasach okrągłych – wartości, na których funkcja ma wykonać obliczenia.

Zapis:

```
srednia(7,3);
```

spowoduje, że zostanie uruchomiona funkcja **srednia**, której pierwszym argumentem jest wartość 7, a drugim – wartość 3. Efektem będzie wyświetlenie na ekranie przeglądarki następującego komunikatu:

```
"Średnia liczb 7 oraz 3 wynosi 5".
```

Funkcje nie służą wyłącznie do wyświetlania komunikatów. Na ogół funkcje są stosowane do wykonywania obliczeń albo do przetwarzania danych i zwracania wyników. Jeśli chcemy, aby funkcja, oprócz komunikatów, przekazywała obliczoną wartość w ostatniej linijce jej definicji, należy zastosować instrukcję **return**, a zaraz za nią umieścić zmienną, którą funkcja ma zwrócić.

PRZYKŁAD 42.2

```
<script>
function suma(a,b)
{
return a+b;
}
var a =Number(prompt("Podaj liczbę: "));
var b =Number(prompt("Podaj liczbę: "));
document.write("Suma liczb wynosi "+ suma(a,b));
</script>
```

Funkcja o nazwie **suma** ma dwa argumenty i zwraca wartość będącą sumą tych dwóch argumentów. Skrypt prosi o podanie dwóch liczb za pomocą okna dialogowego, a następnie te wartości są przekazywane do funkcji **suma**, której wynik jest wyświetlony w oknie przeglądarki.

Po dodaniu jeszcze trzech funkcji wykonujących podstawowe działania arytmetyczne (-, *, /) i zastosowaniu instrukcji **SWITCH** można utworzyć skrypt będący prostym kalkulatorem.

PRZYKŁAD 42.3

```
<script>
function suma(a,b)
{
return a+b;
}
function odejmij(a,b)
{
return a-b;
}
function iloczyn(a,b)
{
return a*b;
}
function iloraz(a,b)
{
return a/b;
}
var a =Number(prompt("Podaj liczbę a: "));
var b =Number(prompt("Podaj liczbę b: "));
var znak=prompt("wybierz działanie (+,-,* lub /):");
switch(znak){
    case '+':
    alert("Suma liczb "+a+" oraz "+b+" wynosi "+suma(a,b));
    break;
    case '-':
    alert("Różnica liczb "+a+" oraz "+b+" wynosi "+odejmij(a,b));
    break;
    case '*':
    alert("Iloczyn liczb "+a+" oraz "+b+" wynosi "+iloczyn(a,b));
    break;
    case '/':
    if (b!=0)
```

```
    alert("Iloraz liczb "+a+" oraz "+b+" wynosi "+iloczyn(a,b));
    else
       alert("Nie wykonano dzielenia, bo b=0");
    break;
    default:
    alert("Źle wybrałeś działanie!");
    break;
}
</script>
```

Podczas omawiania funkcji warto przybliżyć pojęcia iteracji i rekurencji, które omówiono w pierwszej części podręcznika w rozdziale o algorytmach.

Rozważmy skrypt zawierający funkcję obliczającą potęgę liczby 2. Skrypt będzie miał dwie funkcje: jedna będzie iteracyjna, a druga – rekurencyjna.

PRZYKŁAD 42.4

```
<script>
function potega1(x)
{
    var wynik=1;
if (x==0) return 1;
else
    {
for(var i=1;i<=x;i++)
    {
       wynik*=2;
    }
    }
    return wynik;
}
function potega2(x)
    {
       if (x==0) return 1;
       else
          var wynik=2*potega2(x-1);
       return wynik;
       }

var n =Number(prompt("Podaj potege liczby 2: "));
    if(n>0)
       {
```

```
document.write("Iteracyjnie 2 do potęgi "+n+" wynosi
"+(potega1(n))+"<br>");
document.write("Rekurencyjnie 2 do potęgi "+n+" wynosi
"+(potega2(n)));
      }
   else
      {
         alert("n nie może być liczbą ujemną");
      }
</script>
```

Funkcja **potega1** jest funkcją iteracyjną, natomiast **potega2** – rekurencyjną. Pierwsza z nich wykorzystuje pętlę **FOR**, która ma na celu wielokrotnie (n razy) powtórzyć wykonanie tej samej instrukcji (w tym przypadku mnożenie przez kolejną dwójkę). Natomiast druga funkcja wywołuje siebie określoną liczbę razy, aż do obliczenia wszystkich kolejnych iloczynów przez 2.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień znane ci funkcje wbudowane.
2. W jaki sposób definiuje się funkcje?
3. W jaki sposób wywołuje się funkcję z argumentami?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz skrypt pobierający wartość zmiennej **cena** i zawierający definicje funkcji **euro** oraz **funt**. Wywołaj te funkcje z argumentem cena (wartość w złotówkach), aby przeliczyć daną cenę na waluty euro i funt. Wyniki wyświetl w oknie przeglądarki.
2. Utwórz skrypt zawierający funkcję o nazwie **BMI**, która na podstawie masy i wzrostu podanych przez użytkownika obliczy wskaźnik BMI, a następnie wyświetli odpowiedni komunikat, czyli informację o tym, czy dana osoba ma odpowiednią wagę do wzrostu, czy – nie.
3. Utwórz skrypt zawierający funkcję o nazwie **czyParzysta**, której zadaniem będzie sprawdzenie, czy podana liczba jest podzielna przez 2. Jeśli liczba podana przez użytkownika będzie parzysta, funkcja ma zwrócić cyfrę 1, w przeciwnym razie ma zwrócić –1. Jeśli funkcja zwróci 1, główna część programu ma wyświetlić w oknie dialogowym słowo «PARZYSTA». Jeśli funkcja zwróci –1, główna część programu ma wyświetlić w oknie dialogowym słowo «NIEPARZYSTA».

43 Obiekty w JavaScript

ZAGADNIENIA

- Sposób tworzenia własnych obiektów
- Rodzaje obiektów wbudowanych
- Wywoływanie metod dla obiektów wbudowanych

Obiekty to konstrukcje programistyczne o właściwościach, którymi mogą być zmienne lub inne obiekty. Z obiektami są powiązane funkcje wykonujące operacje na właściwościach obiektu, które nazywa się **metodami**. Obiekt definiuje się przez utworzenie funkcji zwanej **konstruktorem**, a następnie – zastosowanie operatora **`new`**.

Aby dostać się do właściwości danego obiektu, należy zastosować zapis:

```
nazwa_obiektu.nazwa_właściwości;
```

Oprócz możliwości tworzenia własnych obiektów, JavaScript zawiera zbiór obiektów wbudowanych, mających własne właściwości i metody. Do właściwości i metod można odwołać się tak jak do zwykłych funkcji, lecz przed nazwą funkcji trzeba umieścić nazwę obiektu, do którego należą dana metoda bądź właściwość. Przykładem jest używana wcześniej metoda **`write`**, która należy do obiektu **`document`**. Dzięki interfejsowi DOM (Document Object Model) każdy element dokumentu jest reprezentowany przez odpowiedni obiekt w JavaScript. Każdy obiekt ma swoje właściwości i metody, które pozwalają wykonywać operacje na dokumencie.

Dla każdej strony są zdefiniowane obiekty:

- **`window`**,
- **`navigator`**,
- **`document`**,
- **`location`**,
- **`history`**.

Obiekt `WINDOW`

Znajduje się on na samym szczycie hierarchii obiektów w JavaScript. Jest to obiekt domyślny, do większości metod i właściwości można zatem się odwołać z pominięciem jego nazwy. Opisuje bieżące okno przeglądarki. Do metod obiektu **`WINDOW`** można się odwołać za pomocą następujących zapisów:

```
window.metoda();
metoda();
```

Tabela 43.1. Właściwości obiektu **WINDOW**

Właściwość	Opis
`frames`	tablica ramek zawartych w bieżącym oknie
`length`	liczba zdefiniowanych ramek
`self`	bieżące okno
`status`	tekst wyświetlany na pasku stanu okna przeglądarki
`defaultStatus`	tekst domyślnie pojawiający się w pasku stanu okna przeglądarki

Tabela 43.2. Metody obiektu **WINDOW**

Metoda	Opis
`alert()`	wyświetla okno dialogowe informacyjne
`close()`	zamyka okno z dokumentem
`confirm()`	wyświetla okno dialogowe decyzyjne
`open()`	otwiera okno przeglądarki
`prompt()`	wyświetla okno tekstowe umożliwiające wprowadzenie wartości przez użytkownika

Obiekt LOCATION

Ten obiekt zawiera informacje dotyczące bieżącego adresu URL dokumentu oraz metody pozwalające na operowanie tym adresem.

Tabela 43.3. Właściwości obiektu **LOCATION**

Właściwość	Opis
`href`	tekst (łańcuch) zawierający cały adres URL dokumentu
`protocol`	łańcuch zawierający nazwę protokołu
`host`	łańcuch zawierający nazwę serwera, nazwę domeny
`hostname`	łańcuch zawierający pełną nazwę serwera wraz z numerem portu
`port`	łańcuch zawierający numer portu
`pathname`	łańcuch zawierający adres ścieżki dostępu do pliku

Tabela 43.4. Metody obiektu LOCATION

Metoda	Opis
`assign()`	wczytuje dokument o adresie wskazanym przez argument URL
`reload()`	wymusza ponowne wczytanie bieżącej strony
`replace()`	zastępuje bieżący dokument wczytanym adresem wskazanym przez URL

Obiekt DOCUMENT

To obiekt zawierający metody i właściwości do modyfikacji dokumentu HTML, który jest aktualnie wczytany przez przeglądarkę.

Tabela 43.5. Właściwości obiektu **DOCUMENT**

Właściwość	Opis
`title`	łańcuch określający tytuł dokumentu
`location`	łańcuch zawierający pełny adres URL bieżącego dokumentu
`lastModified`	data ostatniej modyfikacji dokumentu
`referrer`	URL dokumentu, z którego został wywołany bieżący dokument
`bgColor`	kolor tła dokumentu
`fgColor`	kolor określający kolor tekstu w dokumencie
`linkColor`	kolor linku, który jeszcze nie został aktywowany
`vlinkColor`	kolor odwiedzonego linku
`alinkColor`	kolor aktywowanego linku
`forms[]`	tablica zawierająca wszystkie formularze w dokumencie
`links[]`	tablica zawierająca wszystkie linki znajdujące się w dokumencie

Tabela 43.6. Metody obiektu **DOCUMENT**

Metoda	Opis
`write()`	wypisuje podany komunikat w bieżącym dokumencie
`clear()`	czyści zawartość bieżącego okna
`close()`	zamyka bieżące okno
`getElementById(x)`	odnosi się do elementu o podanym identyfikatorze x
`getElementsByName(x)`	pobiera listy elementów mających atrybut **name** o wartości x
`getElementsByTagName(tag)`	pobiera listy elementów utworzonych za pomocą znacznika określonego przez argument **tag**

PRZYKŁAD 43.1

```
<script>
document.bgColor = "darkgrey";
document.fgColor="yellow";
document.write("Cały tekst na stronie będzie w żółtym kolorze");
</script>
```

Skrypt ustawił ciemnoszare tło, żółtą czcionkę i wyświetlił tekst: „Cały tekst na stronie będzie w żółtym kolorze".

Obiekt STRING

Służy do przechowywania łańcuchów znakowych. Ma tylko jedną właściwość **LENGTH**, która określa liczbę znaków w łańcuchu. Ten obiekt zawiera dużą liczbę metod (tabela 43.7).

Tabela 43.7. Metody obiektu **STRING**

Metoda	Opis
`big()`	zwiększenie rozmiaru czcionki
`blink()`	tekst migający
`bold()`	pogrubienie tekstu
`fixed()`	działanie znacznika <tt>
`italics()`	tekst pochylony
`small()`	zmniejszenie rozmiaru czcionki
`sub()`	tekst jako indeks dolny
`strike()`	przekreślenie tekstu
`sup()`	tekst jako indeks górny
`fontcolor()`	ustawienie koloru czcionki
`fontsize()`	ustawienie rozmiaru czcionki
`substring()`	zwraca podłańcuch wycięty z podanego łańcucha
`toUpperCase()`	konwertuje znaki w łańcuchu na wielkie litery

PRZYKŁAD 43.2

```
<script>
var tekst="Uczymy się programować w JavaScript";
document.write(tekst+"<br>");
document.write(tekst.italics()+"<br>");
document.write(tekst.fontcolor("red")+"<br>");
document.write(tekst.fontsize("14px")+"<br>");
document.write(tekst.small()+"<br>");
document.write(tekst.big()+"<br>");
document.write(tekst.bold()+"<br>");
document.write(tekst.toUpperCase()+"<br>");
```

```
document.write(tekst.substring(25,35)+"<br>");
document.write(tekst.italics().fontcolor("blue")+"<br>");
</script>
```

Skrypt wyświetli tekst w postaci zwykłej, a następnie ten sam tekst będzie wyświetlony w postaci pochylonej, zapisany kolorem czerwonym, dalej ten sam tekst zostanie napisany czcionką w rozmiarze 14 px. Kolejna linijka to tekst pisany czcionką zmniejszoną, pod spodem – czcionką powiększoną, a następnie – pogrubioną. Metoda **toUpperCase** spowoduje zamianę wszystkich znaków na wielkie litery, a metoda **substring** wyłuska w napisie słowo „JavaScript". Ostatni linijka to połączenie dwóch cech: pochylenia i czcionki. Efekt działania skryptu przedstawiono na rys. 43.1.

Uczymy się programować w JavaScript
Uczymy się programować w JavaScript
Uczymy się programować w JavaScript
Uczymy się programować w JavaScript
Uczymy się programować w JavaScript
Uczymy się programować w JavaScript
Uczymy się programować w JavaScript
UCZYMY SIĘ PROGRAMOWAĆ W JAVASCRIPT
JavaScript
Uczymy się programować w JavaScript

Rys. 43.1. Efekt działania skryptu z przykładu 43.2

Obiekt DATE

Służy do przechowywania wartości daty i czasu. Pozwala na uzyskanie aktualnej daty i czasu, na korzystanie z ich składowych oraz na niezależną zmianę każdej z nich.

Obiekty tego typu można tworzyć w następujący sposób:

```
nazwa=new Date();
nazwa=new Date("miesiąc dzień, rok godziny:minuty:sekundy");
nazwa=NewDate(rok, miesiąc, dzień);
nazwa=NewDate(rok, miesiąc, dzień, godziny, minuty, sekundy);
```

Tabela 43.8. Metody obiektu **DATE**

Metoda	Opis
`getDate()`	zwraca dzień miesiąca
`getDay()`	zwraca dzień tygodnia
`getHours()`	zwraca godzinę
`getMinutes()`	zwraca minuty
`getMonth()`	zwraca miesiąc
`getSeconds()`	zwraca sekundy

Metoda	Opis
`getYear()`	zwraca rok
`setDate()`	ustawia dzień miesiąca
`setHours()`	ustawia godzinę
`setMinutes()`	ustawia minuty
`setMonth()`	ustawia miesiąc
`setSeconds()`	ustawia sekundy
`setYear()`	ustawia rok
`toGMTString()`	zwraca datę w postaci GMT
`toLocaleString()`	zwraca datę w postaci lokalnej

PRZYKŁAD 43.3

```
<script>
napis="Uczymy się pisać skrypty.";
data=new Date();
document.write("Dzisaj jest: "+data+"<br>");
document.write("Czas w GMT: "+data.toGMTString()+"<br>");
document.write("Czas lokalny: "+data.toLocaleString()+"<br>");
document.write("Miesiąc: "+(data.getMonth()+1)+"<br>");
document.write("Dzień miesiąca: "+data.getDate()+"<br>");
document.write("Dzień tygodnia: "+data.getDay()+"<br>");
</script>
```

Efekt działania skryptu jest następujący.

Dzisaj jest: Mon Jan 06 2020 17:01:50 GMT+0100 (czas środkowoeuropejski standardowy)
Czas w GMT: Mon, 06 Jan 2020 16:01:50 GMT
Czas lokalny: 6.01.2020, 17:01:50
Miesiąc: 1
Dzień miesiąca: 6
Dzień tygodnia: 1

Rys. 43.2. Efekt działania skryptu z przykładu 43.3

Obiekt `MATH`

Stosuje się go do wykonywania różnego rodzaju obliczeń matematycznych. Udostępnia on wiele stałych matematycznych oraz metod.

Tabela 43.9. Właściwości obiektu **MATH**

Właściwość	Opis
`LN10`	stała wartość; logarytm naturalny z 10
`LN2`	stała wartość; logarytm naturalny z 2
`PI`	stała wartość; liczba π
`SQRT1_2`	stała wartość; pierwiastek kwadratowy z ½
`SQRT2`	stała wartość; pierwiastek kwadratowy z 2

Tabela 43.10. Metody obiektu **MATH**

Metoda	Opis
`abs(x)`	wartość bezwzględna liczby **x**
`ceil(x)`	najmniejsza liczba całkowita większa lub równa **x**
`exp(x)`	stała Eulera podniesiona do potęgi **x**
`floor(x)`	największa liczba całkowita mniejsza lub równa **x**
`log(x)`	logarytm naturalny **x**
`max(x,y)`	wyznaczanie większej z liczb **x** i **y**
`min(x,y)`	wyznaczanie mniejszej z liczb **x** i **y**
`pow(x,y)`	obliczenie **x** do potęgi **y**
`round(x)`	zaokrąglenie liczby **x** do najbliższej całkowitej
`sin(x)`, `cos(x)`, `tan()`	sinus, cosinus, tangens liczby **x** argumenty w radianach
`sqrt(x)`	pierwiastek kwadratowy z liczby **x**
`random(x)`	zwraca liczbą pseudolosową z przedziału [0, 1)

PRZYKŁAD 43.4

```
<script>
var a=36, b=2, c=-7;
document.write("Liczba a wynosi: "+a);
document.write("<br>Liczba b wynosi: "+b);
document.write("<br>Pierwiastek kwadratowy z liczby "+a+"
wynosi: "+Math.sqrt(a));
document.write("<br>"+a+" do potęgi "+b+" wynosi: "+Math.
pow(a,b));
```

```
document.write("<br>Spośród liczb "+a+" oraz "+b+" większa
jest liczba "+Math.max(a,b));
document.write("<br>Wartość bezwzględna liczby "+c+" wynosi:
"+Math.abs(c));
document.write("<br>Logarytm naturalny liczby "+b+" wynosi:
"+Math.log(b));
</script>
```

Na ekranie przeglądarki otrzymamy następujące informacje:

Liczba a wynosi: 36
Liczba b wynosi: 2
Liczba c wynosi: -7
Pierwiastek kwadratowy z liczby 36 wynosi: 6
36 do potęgi 2 wynosi: 1296
Spośród liczb 36 oraz 2 większa jest liczba 36
Wartość bezwzględna liczby -7 wynosi: 7
Logarytm naturalny liczby 2 wynosi: 0.693147180559453

Rys. 43.3. Przykład zastosowania obiektu **MATH**

Obiekt ARRAY

Tablice służą do przechowywania wartości wielu zmiennych. Obiekt **ARRAY** służy do modyfikacji tablicami zmiennych. Aby utworzyć nową tablicę, należy zastosować instrukcję:

```
var nazwa_tablicy=new Array();
```

lub

```
var nazwa_tablicy=[];
```

Jeżeli w nawiasach zostanie podana liczba 5, to utworzy się tablica zawierająca pięć pustych elementów.

PRZYKŁAD 43.5

```
var tab1=new Array(10);
```

Tablicę można również tworzyć, jeżeli wstawi się do niej konkretne wartości:

```
var tab2=new Array('Asia', 'Basia', 'Kasia');
```

Aby uzyskać dostęp do elementów tablicy, należy podać numer indeksu danego elementu. Elementy tablicy są numerowane od zera.

Instrukcja `document.write(tab2[1]);` wyświetli imię `Basia`.

PRZYKŁAD 43.6

```
<script>
t=new Array(5);
for (var i=0;i<5;i++)
    {
       t[i]=Math.round(Math.random()*100+1)
       document.write("t["+i+"]="+t[i]+"<br>");
    }
</script>
```

W skrypcie zadeklarowano pięcioelementową tablicę, dla której wygenerowano liczby z zakresu od 1 do 100. W celu wypisania wyświetlenia wygenerowanych liczb należy zastosować pętlę **FOR**.

PRZYKŁAD 43.7

```
<html>
<head>
<title>Sumowanie wartości w tablicy</title>
<style>
table, tr, td
    {
       border: 2px green solid;
    }
    td
    {
       width:50px;
    }
table
    {
       width:500px;
       height:50px;
       text-align: center;
       border-collapse: collapse;
    }
</style>
</head>
<body>
<script>
var tab1=new Array(5);
var s=0;
```

```
for(var i=0;i<10;i++){
    tab1[i]=Number(prompt("Podaj wartosci: "));
    s=s+tab1[i];
}
document.write("Wpisałeś następujące wartości do tablicy: ");
document.write("<table><tr>");
for(i=0;i<10;i++){
    document.write("<td>" +tab1[i]+"</td>");
}
 document.write("</tr></table>");
document.write("Suma wynosi: "+s);
</script>
</body>
```

Za pomocą pętli **FOR** użytkownik zostanie poproszony o wpisanie wartości dla dziesięcioelementowej tablicy, następnie te wartości zostaną wyświetlone w tabeli oraz wyświetli się wynik sumy tych wartości.

Wpisałeś następujące wartości do tablicy:

5	45	56	8	–4	223	78	1	2	3

Suma wynosi: 417

Rys. 43.4. Efekt skryptu z przykładu 43.7

W tabeli 43.11 opisano niektóre metody obiektu **ARRAY**.

Tabela 43.11. Metody obiektu **ARRAY**

Metoda	Opis
`concat()`	dołącza dwie tablice i zwraca nową
`join()`	łączy elementy tablicy
`pop()`	usuwa ostatni element z tablicy i zwraca ten element
`push(x)`	dodaje jeden lub więcej elementów tablicy i zwraca ten, który został dodany ostatni
`reverse()`	odwraca kolejność elementów tablicy
`sort()`	sortuje elementy tablicy

PRZYKŁAD 43.8

```
<script
var tab1=new Array(5,78,56);
var tab2=new Array(-15,5,123,45);

var tab3=tab1.concat(tab2);

var dl=tab3.length;  //rozmiar tablicy nr 3
document.write("<br><b><u>Po  połączeniu  tab1  z  tab2:</u></
b><br>");
for (i=0;i<dl;i++)
{
document.write(tab3[i]+"\t");     //znak \t wstawia tabulator
}
tab3.sort();
document.write("<br><b><u>Po posortowaniu:</u></b><br>");
for (i=0;i<dl;i++)
{
document.write(tab3[i]+"\t");
}
tab3.reverse();
document.write("<br><b><u>Po   odwróceniu   kolejności:</u></
b><br>");
for (i=0;i<dl;i++)
{
document.write(tab3[i]+"\t");
}
</script>
```

Efektem będzie wyświetlenie nowej tablicy uzyskanej po połączeniu tablic tab1 i tab2. Za pomocą metody **sort** wartości tablicy zostały posortowane, a na koniec za pomocą metody **reverse** ich kolejność została odwrócona.

Po połączeniu tab1 z tab2:
5 78 56 -15 5 123 45
Po posortowaniu:
-15 123 45 5 5 56 78
Po odwróceniu kolejności:
78 56 5 5 45 123 -15

Rys. 43.5. Efekt zastosowania metod obiektu **ARRAY**

W JavaScript można definiować tablice dwuwymiarowe, czyli tablice tablic.

PRZYKŁAD 43.9

```
<html>
<head>
<title>Tabele dwuwymiarowe w JS</title>
<style>
table,tr,td
    {
      border: 1px solid magenta;
    }
table
    {
      width:200px;
      height:100px;
      text-align: center;
    }
</style>
</head>
<body>
<script>
 var tab = new Array(2);
//deklaracja tablicy o 3 wierszach I 2 kolumnach
for (var i = 0; i < 3; i++) {
    tab[i] = [];
}
//wylosowanie wartości dla tablicy
for (i=0;i<2;i++)
{
for (j=0;j<3;j++)
{
tab[i][j]=Math.round(50*Math.random()+1);
}
}
//wyświetlenie wartości w tabeli
document.write("<table>");
for (i=0;i<2;i++)
{
document.write("<tr>");
for (j=0;j<3;j++)
{
```

```
document.write("<td>"+tab[i][j]+"</td>");
}
document.write("</tr>");
}
document.write("<table>");
</script>
</body>
```

Efektem kodu będzie wylosowanie tablicy z trzema wierszami i dwoma kolumnami z liczbami z zakresu <1,100> oraz wyświetlenie ich w tabeli.

9	2	26
28	3	37

Rys. 43.6. Efekt skryptu generującego liczby dla tablicy dwuwymiarowej

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest konstruktor?
2. Co to jest metoda?
3. Jakie znasz obiekty wbudowane?
4. Wymień podstawowe metody działające na obiektach **MATH**.
5. Do czego służy metoda **substring**? Podaj przykład działania.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz skrypt, który za pomocą okna dialogowego wczyta liczbę i obliczy jej pierwiastek kwadratowy, a wynik wyświetli w oknie przeglądarki.
2. Utwórz skrypt, który wygeneruje liczby z zakresu <1,50> dla dwudziestoelementowej tablicy, wyszuka w niej wartości podzielne przez 3, wyświetli je oraz obliczy ich średnią arytmetyczną.

44 Zdarzenia

ZAGADNIENIA

- Rodzaje dostępnych zdarzeń w JavaScript
- Przypisywanie funkcji do danego zdarzenia

Na stronie internetowej mogą zachodzić różnego rodzaju zdarzenia, np. po kliknięciu przycisku formularza zostanie uruchomiana funkcja wyświetlająca dane wcześniej wprowadzone do formularza.

Zdarzenia można podzielić na:

- zdarzenia myszy,
- zdarzenia klawiatury,
- zdarzenia dokumentu,
- zdarzenia formularza.

Tabela 44.1. Rodzaje zdarzeń

Nazwa zdarzenia	Rodzaj elementu	Opis działania
`onAbort`	obraz	anulowanie pobrania grafiki
`onBlur`	okno, formularz	usunięcie aktywności pola
`onChange`	text, textarea, select	zmiana wartości
`onClick`	dowolny znacznik	kliknięcie dowolnego znacznika
`onDbClick`	dowolny znacznik	podwójne kliknięcie dowolnego znacznika
`onDragDrop`	okno	przeciągnięcie obiektu poza okno
`onError`	obraz, okno	błąd podczas ładowania
`onFocus`	okno, formularz	uaktywnienie elementu
`onKeyDown`	dokument, obraz, link, textarea	wciśnięcie (przytrzymanie) klawisza na klawiaturze
`onKeyPress`	dokument, obraz, link, textarea	wciśnięcie (krótkotrwałe) klawisza na klawiaturze
`onKeyUp`	dokument, obraz, link, textarea	zwolnienie klawisza
`onLoad`	ciało dokumentu	ładowanie strony

Nazwa zdarzenia	Rodzaj elementu	Opis działania
`onMouseDown`	dokument, przycisk, link	wciśnięcie (przytrzymanie) klawisza myszy
`onMouseOut`	wszystko	opuszczenie danego obszaru przez kursor myszy
`onMouseOver`	wszystko	najechanie kursorem myszy na dany obszar
`onMouseUp`	wszystko	puszczenie klawisza myszy
`onMove`	okno	poruszanie oknem
`onReset`	formularz	wyczyszczenie formularza
`onResize`	okno	zmiana rozmiaru okna
`onSelect`	text, textarea	wybranie danego elementu
`onSubmit`	formularz	zatwierdzenie formularza
`onUnload`	ciało dokumentu	zamknięcie strony

PRZYKŁAD 44.1

Rozważmy aplikację, w której zostanie wywołane zdarzenie **`onLoad`** w momencie uruchomienia strony. Zdarzenie to wywoła funkcję **`podaj_imie()`**, która wyświetli okno dialogowe z pytaniem: „Jak masz na imię?”. Jeżeli imię nie zostanie podane w oknie przeglądarki, pojawi się komunikat: „Miło, że jesteś z nami”. W przypadku podania imienia komunikat będzie uzupełniony o podane imię.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Tworzenie skryptów w JS</title>
<script>
function podaj_imie()
{
var imie=prompt("Jak masz na imię? ");
if (imie==null || imie==" ")
{
document.write("Miło, że jesteś z nami!");
}
else
{
document.write("Miło, że jesteś z nami "+imie);
```

```
}
return imie;
}
</script>
</head>
<body onload="podaj_imie()">
</body>
</html>
```

PRZYKŁAD 44.2

Tym przykładem zobrazowano działanie zdarzenia **onMouseOver** oraz **onMouseOut**, które w tym przypadku wywołują funkcję **koloruj()**. Zadaniem tej funkcji jest ustawienie koloru tekstu na niebieski, gdy najedziemy kursorem myszy na tekst, i czerwony, gdy opuścimy kursor myszy z danego tekstu.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Zdarzenia w JS</title>
<script>
function koloruj(x)
{
    if (x==1)
       document.fgColor = "blue";
    if (x==2)
       document.fgColor = "red";
}
</script>
</head>
<body>
<h1 onmouseover="koloruj(1)"
onmouseout="koloruj(2)">PRZYKŁADOWY TEKST</h1>
</body>
</html>
```

PRZYKŁAD 44.3

Rozważmy aplikację, która po kliknięciu określonego obrazka wyświetli nazwę zwierzątka i krótki wierszyk.
W sekcji głównej umieścimy tabelę zawierającą poszczególne ilustracje.

```
<div id="tabela" >
<table id="dane" >
<tr>
<td><img src="papuga.png" id="papuga" onclick="papuga()" > </
td>
<td><img src="zyrafa.png" id="zyrafa" onclick="zyrafa()"></
td>
<td><img src="krokodyl.png"
id=″krokodyl"onclick="krokodyl()"></td>
<td><img src="zebra.png" id="zebra" onclick="zebra()" ></td>
<td><img src="wilk.png" id="wilk" onclick="wilk()"></td>
</tr>
</table>
</div>
```

W sekcji nagłówkowej umieszczamy skrypt, który zawiera funkcje uruchamiane dla poszczególnych obrazków:

```
<script>
function papuga()
{
document.getElementById("wynik").innerHTML="PAPUGA"
document.getElementById("tresc").innerHTML="Papużko,
papużko,<br> Powiedz mi coś na uszko.<br> Nic nie powiem, boś
ty plotkarz, <br>Powtórzysz każdemu, kogo spotkasz.";
}
function zyrafa()
{
document.getElementById("wynik").innerHTML="ŻYRAFA";
document.getElementById("tresc").innerHTML="Żyrafa
tym głównie żyje,<br>Że w górę wyciąga szyję. <br>A ja
zazdroszczę żyrafie, <br>Ja nie potrafię.";
}
function krokodyl()
{
document.getElementById("wynik").innerHTML="KROKODYL";
document.getElementById("tresc").innerHTML="Skąd ty jesteś,
krokodylu?<br>Ja? Znad Nilu.<br>Wypuść mnie na kilka chwil,
<br>To zawiozę cię nad Nil.";
}
function zebra()
{
```

```
document.getElementById("wynik").innerHTML="ZEBRA";
document.getElementById("tresc").innerHTML="Czy ta zebra jest
PRAWDZIWA?<br>Czy to tak NAPRAWDĘ bywa?<br>Czy też malarz
z bożej łaski<br>Pomalował osła W PASKI?";
}
function wilk()
{
document.getElementById("wynik").innerHTML="WILK";
document.getElementById("tresc").innerHTML="Powiem ci
w słowach kilku,<br>Co myślę o tym wilku:<br>Gdyby nie był na
obrazku,<br>Zaraz by cię zjadł, głuptasku.";
}
</script>
```

Efekt jest następujący:

Rys. 44.1. Efekt działania aplikacji z przykładu 44.2

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień nazwy grup zdarzeń.
2. Wymień nazwy zdarzeń myszy.
3. Wymień nazwy zdarzeń klawiatury.

45 Obsługa formularzy

ZAGADNIENIA

- Zdarzenia obsługujące formularze
- Zabezpieczenia formularzy
- Stosowanie zdarzeń związanych z formularzem
- Sprawdzanie poprawności formularza

Formularzy na stronach WWW używa się zazwyczaj do zebrania danych od osób odwiedzających stronę, a następnie wysłania ich do serwera WWW. Nic nie stoi jednak na przeszkodzie, aby z danych wpisanych do formularza mógł korzystać także klient. Język JavaScript pozwala na dostęp do formularzy znajdujących się na stronie WWW. Można m.in. odczytywać wprowadzane dane, zmieniać je lub wysłać formularz do serwera. Pomaga także w sprawdzaniu, czy wszystkie pola formularza zostały uzupełnione.

Do tworzenia formularzy w języku HTML służy znacznik <**FORM**>. Każdy element formularza jest traktowany jak osobny obiekt, który ma swój zestaw właściwości. Dodatkowo każdy formularz na stronie (ze swoimi elementami) jest osobną właściwością obiektu **document**. Można się do niego odwołać przez tablicę **forms** – trzeba podać odpowiedni numer porządkowy. Do pierwszego formularza zdefiniowanego na stronie odwołuje się przez wywołanie **document.forms[0]**, do drugiego – **document.forms[1]** itd. Jeżeli formularz ma swoją nazwę, to odwołanie do niego wygląda następująco:

```
document.nazwa_formularza
```

Każdy formularz to osobny obiekt, reprezentowany przez obiekt form języka JavaScript. Do jego elementów można się odwołać przez tablicę **elements**:

```
document.forms[0].elements[0]        //pierwszy element
                                     pierwszego
                                     //formularza na stronie
```

lub

```
document.nazwa_formularza.nazwa_elementu.
```

Każdy element formularza ma również swoje właściwości, do których można się odwołać, jeżeli poda się odpowiednią nazwę właściwości:

```
document.forms[0]. elements[0].name
```

lub

```
document.forms[0]. elements[0]. value.
```

W odwoływaniu się do elementu formularza wewnątrz jego definicji można użyć konstrukcji **this.właściwość**.

PRZYKŁAD 45.1

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<meta charset="utf-8">
<TITLE>Obsługa formularzy</TITLE>
<SCRIPT>
function powitanie(form1)
{
    if (form1.imie.value=="")
    {
    document.write("Nie podałeś swojego imienia.");
    }
    else
    {
    document.write("Witaj "+form1.imie.value);
    }
}
</SCRIPT>
</HEAD>
<BODY>
<form method="post" name="form1">
Podaj swoje imię: <input type="text" name="imie">
<br><input type="button" name="p1" value="Kliknij" onClick=
"powitanie(this.form)">
</form>
</BODY>
</HTML>
```

W sekcji **BODY** dokumentu HTML został stworzony formularz o nazwie **FORM1** z jednym polem tekstowym i przyciskiem. Polu tekstowemu nadano nazwę **imie**. W skrypcie zdefiniowano funkcję **powitanie**, która zostanie uruchomiona po kliknięciu przycisku formularza (zostanie uruchomione zdarzenie **onClick**). Do funkcji powitanie przekazujemy **THIS.FORM**, czyli obiekt reprezentujący bieżący formularz o nazwie **FORM1**.

Funkcja sprawdza, czy w polu imie zostało wprowadzone imię. Jeśli użytkownik wpisał imię, zostanie wyświetlony komunikat powitalny, w przeciwnym razie przeglądarka powiadomi użytkownika, że nie zostało podane imię.

PRZYKŁAD 45.2

Tym razem utworzymy formularz pobierający trzy dane: imię, nazwisko i rok urodzenia. Funkcja **pobierz** sprawdza, czy każde z pól formularza zostało uzupełnione. Jeśli użytkownik wprowadzi wszystkie informacje, to pojawi się komunikat, że formularz został w pełni uzupełniony. Jeśli natomiast któreś z pól podczas wprowadzania danych zostanie pominięte, będą wyświetlone dane, które pozostały w formularzu puste.

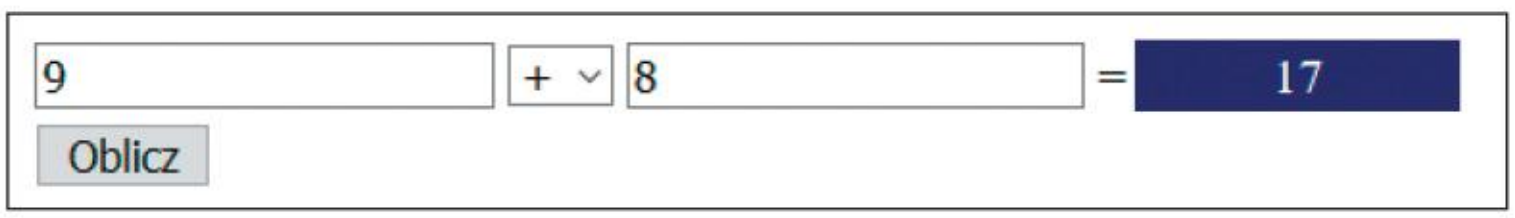
```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Obsługa formularzy</title>
<script>
function pobierz()
{
    f=document.forms[0]; //pierwszy element tablicy formularzy
    napis=" \n";
    braki=false;
    if (f.imie.value=="")
    { napis+="imię\n";      braki=true; }

    if (f.nazw.value=="")
    { napis+="nazwisko\n"; braki=true;}
    if (f.data.value=="")
    { napis+="data urodzenia"; braki=true;}
    if (braki==false) alert("Wprowadziłeś wszystkie dane");
    else alert("Brakuje następujacych danych: \n"+napis);
}
</script>
</head>
<body>
<form method="post" name="form2">
<table>
<tr><td>Podaj swoje imię: </td><td> <input type="text"
name="imie"></td></tr>
<tr><td>Podaj swoje nazwisko: </td><td> <input type="text"
name="nazw"></td></tr>
<tr><td>Podaj rok urodzenia: </td><td> <input type="data" name=
"data" size="4"></td></tr>
```

```
<tr><td colspan="2" align="right"><input type="button" name=
"p1" value="Kliknij" onClick="pobierz()"></td></tr>
</table>
</form>
</body>
</html>
```

Obsługa formularza może odbyć się także za pomocą metody **getElementById**.

PRZYKŁAD 45.3

Rozważmy formularz będący kalkulatorem wykonującym cztery podstawowe działania arytmetyczne.



Rys. 45.1. Widok kalkulatora wykonującego cztery podstawowe działania arytmetyczne

Formularz zawiera: dwa pola tekstowe, listę rozwijaną oraz przycisk, który po kliknięciu na niego uruchomi funkcję **oblicz()**. Ta funkcja pobiera z formularza dwie liczby z pól tekstowych oraz działanie arytmetyczne. Każdy z tych element ma unikatowe **id**, które jest pobierane za pomocą metody **getElementById**.

```
<html>
<head>
<title>Zadanie3</title>
<meta charset="utf-8">
<style>
#wynik{
background-color:#000066;
width:100px;
text-align:center;
color:white;
}
</style>
<script>
function oblicz()
```

```
{
var x=Number(document.getElementById("liczba1").value);
var y=Number(document.getElementById("liczba2").value);
var z=document.getElementById("wyb").value;

if (z=="d1") document.getElementById("wynik").innerHTML=x+y;
if (z=="d2") document.getElementById("wynik").innerHTML=x-y;
if (z=="d3") document.getElementById("wynik").innerHTML=x*y;
if (z=="d4") document.getElementById("wynik").innerHTML=x/y;
}
</script>
</head>
<body>
<form>
<table>
<tr><td><input type="text" id="liczba1"></td>
<td><select id="wyb">
<option value="d1″>+</option>
<option value="d2″>-</option>
<option value="d3″>*</option>
<option value="d4″>/</option>
</select>
</td>
<td>
<input type="text" id="liczba2"></td>
<td>=</td>
<td id ="wynik"></td>
</tr>
<tr><td colspan="5">
<input  type="button″  value="Oblicz"  onClick="oblicz()"></
td></tr>
</table>
</form>
</body>
</html>
```

PRZYKŁAD 45.4

Utwórzmy formularz do obsługi rezerwacji pokoju w hotelu. Formularz będzie umożliwiał wprowadzenie liczby osób dorosłych, dzieci oraz liczby noclegów. Pod blokiem zawierającym formularz umieszczamy blok, w którym wyświetlimy koszty wypoczynku.

Rys. 45.2. Widok bloku z formularzem umożliwiającym wprowadzanie danych hotelowych

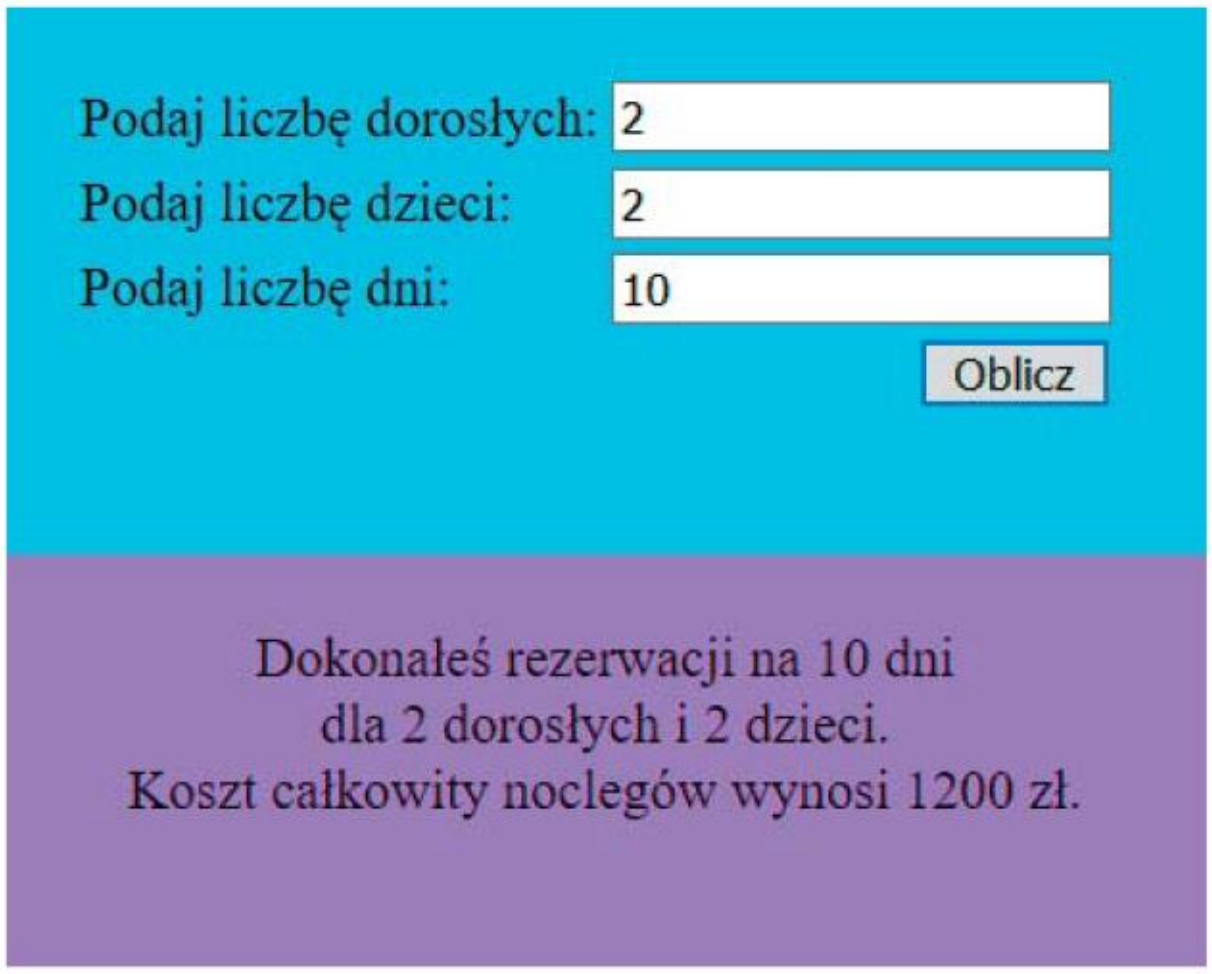
Skrypt ma za zadanie obliczyć koszt pobytu za każdą osobę dorosłą 35 zł/dobę, a za dziecko – 25 zł/dobę. Ponadto skrypt sprawdza, czy użytkownik wprowadził wartość do formularza i czy jest ona liczbą.

```
<html>
<head>
<title>Zadanie1</title>
<meta charset="utf-8">
<style>
#formularz{
margin:auto;
width:330px;
height:140px;
background-color:#66ccff;
padding-left:20px;
padding-top:20px;
}
#wynik{
margin:auto;
width:350px;
```

```
height:100px;
background-color:#cc66ff;
padding-top:20px;
text-align:center;
}
</style>
<script>
function przelicz()
{
var dor=document.getElementById("d1").value;
var dz=document.getElementById("d2").value;
var dni=document.getElementById("d3").value;
if (!isNaN(dor) && !isNaN(dz) && !isNaN(dni)&& dor!="" &&
dz!="" && dni!="")
{
var koszt=(dor*35+dz*25)*dni;
document.getElementById("wynik").innerHTML="Dokonałeś
rezerwacji na "+dni+" dni <br> dla "+dor+" dorosłych i "+dz+"
dzieci.<br>Koszt całkowity noclegów wynosi "+koszt+" zł.";
}
}
</script>
</head>
<html>
<body>
<div id="formularz">
<form method="post">
<table>
<tr>
<td>Podaj liczbę dorosłych:</td><td><input type="text"
id="d1" ></td>
</tr>
<tr>
<td>Podaj liczbę dzieci:</td><td><input type="text" id="d2"
></td></tr>
<tr>
<td>Podaj liczbę dni:</td><td><input type="text" id="d3" >
</td></tr>
<tr>
<td colspan="2" align="right">
```

```
<input type="button" id="przycisk" value="Oblicz"
onClick="przelicz()">
</td>
</tr>
</table>
</form>
</div>
<div id="wynik"></div>
</body>
</html>
```

Efektem aplikacji złożonej z powyższego formularza, skryptu oraz oczywiście sformatowanego według pewnego arkusza stylów jest następująca postać:



Rys. 45.3. Widok formularza z wprowadzonymi danymi dotyczącymi rezerwacji w hotelu

PRZYKŁAD 45.5

W przykładzie przedstawiono zastosowanie zdarzeń na witrynie internetowej. Jest to strona do obsługi hurtowni bombek świątecznych. Klient ma możliwość wyboru trzech zestawów w cenie: 12,50 zł, 17,50 zł i 15,50 zł. Każdy z zestawów jest przedstawiony w lewym panelu strony, w prawym panelu wstawiono formularz do wykonania zamówienia.

Rys. 45.4. Zrzut witryny internetowej hurtowni bombek SantaClaus

Witryna jest złożona z pliku html zawierającego treść strony, potrzebnego skryptu oraz pliku css, w którym określono wygląd strony.

Zawartość pliku index.html:

```
<html>
<head>
<title> Hurtownia bombek</title>
<meta charset="utf-8">
<link rel="stylesheet" href="style.css" type="text/css" />
<link href="https://fonts.googleapis.com/css?family=Sedgwick
+Ave" rel="stylesheet">
</head>
<body>
<section id="contener">
<section id="baner">
  Hurtownia bombek SantaClaus
  </section>
  <section id="lewy">
  <table>
        <tr><td colspan="3">ZESTAW Nr 1</td></tr>
      <tr class="bombki">

      <td><img src="b1.png" class="obrazek"></td>
        <td><img src="b2.png" class="obrazek"></td>
        <td><img src="b3.png" class="obrazek"></td>
      </tr>
```

```
   <tr><td colspan="3">ZESTAW Nr 2</td></tr>
<tr class="bombki">
<td><img src="b4.png" class="obrazek"></td>
   <td><img src="b5.png" class="obrazek"></td>
   <td><img src="b6.png" class="obrazek"></td>
</tr>
<tr><td colspan="3">ZESTAW Nr 3</td></tr>
<tr class="bombki">
<td><img src="b7.png" class="obrazek"></td>
   <td><img src="b8.png" class="obrazek"></td>
   <td><img src="b9.png" class="obrazek"></td>
</tr>
   </table>

</section>
<section id="prawy">
   <section id="gprawy">
      <img src="choinka1.png" style="float:right;">

         <p class="tyt">Dokonaj zamówienia:</p>
   <form>
<p>Numer zestawu: <select id="zestaw">
      <option value="z1">Zestaw Nr 1 (12,50 zł)</option>
      <option value="z2">Zestaw Nr 2 (17,50 zł)</option>
      <option value="z3">Zestaw Nr 3 (15,50 zł)</option>
      </select> </p>
<p>Ilość pudełek <input type="number" id="ilosc">
<p><input type="checkbox" id="tak">Jestem stałym klientem
   <p> <input type="button" onMouseover="oblicz();" value=
   "Przelicz zamówienie">

   </form>
        </section>
        <section id="dprawy">
<script>
   function oblicz()
   {
   var zestaw=document.getElementById("zestaw").value;
   var ilosc=Number(document.getElementById("ilosc").value);
   var zgoda=document.getElementById("tak").checked;
     if (zgoda==false)
```

```
        {
           if (zestaw=="z1")
           {var koszt=ilosc*12.5;
document.getElementById("wynik").innerHTML="Dokonałeś
zamówienia na zestaw Nr 1.<br>Ilość zamówionych pudełek:
"+ilosc+".<br>Koszt zamówienia wynosi: "+koszt+" zł";
           }
           if (zestaw=="z2")
              {koszt=ilosc*17.5;
document.getElementById("wynik").innerHTML="Dokonałeś
zamówienia na zestaw Nr 2.<br>Ilość zamówionych pudełek:
"+ilosc+".<br>Koszt zamówienia wynosi: "+koszt+" zł";
           }
           if (zestaw=="z3")
              {koszt=ilosc*15.5;
document.getElementById("wynik").innerHTML="Dokonałeś
zamówienia na zestaw nr 3.<br>Ilość zamówionych pudełek:
"+ilosc+"<br>Koszt zamówienia wynosi: "+koszt+" zł";
           }
        }
      else{
           if (zestaw=="z1")
           {koszt=(ilosc*12.5)*0.8;
document.getElementById("wynik").innerHTML="Dokonałeś
zamówienia na zestaw Nr 1.<br>Ilość zamówionych pudełek:
"+ilosc+".<br>Koszt zamówienia wynosi: "+koszt+" zł";
           }
           if (zestaw=="z2")
           {koszt=(ilosc*17.5)*0.8;
document.getElementById("wynik").innerHTML="Dokonałeś
zamówienia na zestaw Nr 2.<br>Ilość zamówionych pudełek:
"+ilosc+"<br>Koszt zamówienia wynosi: "+koszt+" zł";
              }
           if (zestaw=="z3")
           {koszt=(ilosc*15.5)*0.8;
document.getElementById("wynik").innerHTML="Dokonałeś
zamówienia na zestaw Nr 3.<br>Ilość zamówionych pudełek:
"+ilosc+"<br>Koszt zamówienia wynosi: "+koszt+" zł";
              }
           }
      }
```

```
</script>
        <section id="wynik"></section>
    </section>
</section>
    </section>
    </body>
</html>
```

Zawartość pliku style.css:

```
body
{
    background-image: url("tlo6.png");
    font-size:18px;
}
#contener
{
    width:90%;
    margin:0 auto;
    padding:0 auto;

}
#baner{
    height: 10%;
    text-align: center;
    color: #CC0000;
    font-size:40px;
    font-family: 'Sedgwick Ave', cursive;
    word-spacing: 5px;
    border-bottom: 5px #CC0000 double;
    margin-bottom: 2%;
    }
td{
    text-align: center;
    font-weight: bold;
}

table {
    width:80%;
    height:100%;
    margin:auto;
}
```

```
.bombki td{

    background-color: #CCCCCC;
    border-radius: 30%;
    border-spacing: 20px;

    text-align: center;
}
#lewy{
width:50%;
float:left;
height:70%;
}

#prawy{
width:50%;
float:left;
height:69%;
    line-height: 2em;
}
p{
    text-indent: 2em;
}
#bb{
width:100%;
text-align:center;
}
#gprawy{
    text-align: center;
    height:70%;

}
#dprawy{
    text-align: center;
    height:30%;
    color: #330066;
    font-weight: bold;

}
input[type="button"]
{
```

```
background-color: #CC0000;
height:45px;
font-weight:bold;
color:white;
    border-radius: 5px;
}
input[type="number"]
{
    width:50px;
}
select{
    width:160px;
}
#aa
{
text-align:center;
margin:auto;
}
.tyt{
    font-size:20px;
    color:#CC0000;
    letter-spacing: 2px;
    font-family: 'Sedgwick Ave', cursive;
}
.obrazek{
    height: 80%;
    width: 80%;

}
#tab
{
    width:70%;
    height:60%;
}
#stopka{
    height: 10%;
background-color: #50344f;
    text-align: center;
    border-radius: 8px;
    color:white;
    clear: both;
```

```
    }
#wynik{
    text-align: center;
}
i{
    font-size:14px;
    color:#a35fc0;
}
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień zdarzenia do obsługi formularza.
2. Wyjaśnij, w jaki sposób sprawdzić, czy użytkownik wprowadził dane do formularza.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Wykonaj stronę internetową przedstawioną na poniższym rysunku.

Stronę podziel na: baner, panel lewy, panel prawy i stopkę. W panelu lewym umieść formularz umożliwiający wpisanie liczby dzieci, osób dorosłych i noclegów. Poniżej wstaw przycisk, który spowoduje wyświetlenie kosztu wczasów. W panelu prawym w tabeli wstaw zdjęcia związane z wypoczynkiem nad morzem, które sam przygotujesz. W stopce umieść baner reklamowy firmy turystycznej. Wykonaj zadanie według własnego pomysłu w dowolnym edytorze graficznym. Kolorystyka strony powinna być zbliżona do przedstawionego wzoru.

IX. Skrypty po stronie serwera – PHP

- Struktura języka PHP
- Typy danych
- Operatory
- Instrukcje sterujące
- Funkcje własne i wbudowane
- Obsługa formularzy
- Obsługa bazy danych
- Pliki cookie i sesje

46 Struktura języka PHP

ZAGADNIENIA

- Zastosowanie języka PHP
- Umieszczanie skryptu PHP w dokumencie HTML
- Sposoby umieszczania komentarzy
- Definicja zmiennej

PHP (ang. *Hypertext Preprocessor*) jest skryptowym językiem programowania przeznaczonym do tworzenia i generowania dynamicznych stron WWW, coraz częściej nazywanych aplikacjami WWW. Integruje się on z określonym serwerem WWW (popularny serwer Apache) jako moduł serwera lub skrypt CGI. Kod skryptu, na którego podstawie ma być wygenerowana strona WWW, jest najczęściej osadzony wewnątrz tej strony.

Kod PHP umieszcza się w plikach o domyślnym rozszerzeniu *.php, a następnie – na serwerze z interpreterem PHP. Jeśli zwykłemu plikowi HTML nada się rozszerzenie .php, to zostanie on prawidłowo wyświetlony, mimo że nie jest to skrypt PHP. Dzieje się tak dlatego, że podczas przetwarzania strony parser PHP pracuje w dwóch trybach: HTML, w którym cała treść jest wyświetlana, bez przetwarzania, i PHP, w którym treść jest traktowana jako skrypt do przetworzenia.

Istnieje kilka sposobów umieszczania skryptów PHP w dokumencie HTML. Są one reprezentowane przez:

- znaczniki kanoniczne,
- znaczniki typu SGML,
- znaczniki typu ASP,
- znaczniki HTML.

Znaczniki kanoniczne to standardowe znaczniki PHP. Są one rozpoznawalne niezależnie od tego, jakie opcje są włączone w pliku konfiguracyjnym. Polecenie umieszczające kod PHP ma wtedy następującą postać:

```
<?php
instrukcje skryptu
?>
```

Znaczniki typu SGML to znaczniki w tzw. postaci skróconej. Aby móc korzystać z tego sposobu, należy włączyć rozpoznawanie tych znaczników, czyli uruchomić

opcję **short_open_tag**. Konstrukcja umieszczania kodu PHP ma wtedy następującą postać:

```
<?
instrukcje skryptu
?>
```

Aby móc zastosować znaczniki typu ASP, należy w pliku konfiguracyjnym włączyć opcję **asp_tags**. Ta konstrukcja ma następującą postać:

```
<%
instrukcje skryptu
%>
```

Ostatni ze sposobów jest dobrze znany użytkownikom HTML. Znacznikiem otwierającym skrypt jest **<script>** z parametrem **language**, a jego wartością – **„php"**. Znacznikiem zamykającym jest **</script>**. Polecenie pozwalające umieszczać kod PHP ma wówczas następującą postać:

```
<script language="php">
instrukcje skryptu
</script>
```

Ta postać jest rozpoznawalna tak, jak w przykładzie postaci kanonicznej. Nie wymaga ustawiania żadnych dodatkowych opcji w pliku konfiguracyjnym. Jest ona już jednak rzadko stosowana.

Język PHP umożliwia przetwarzanie danych z formularzy, dynamiczne generowanie zawartości stron internetowych, wysyłanie i odbieranie cookies, a także wykonywanie operacji na bazie danych. Pozwala również na dynamiczne tworzenie obrazów, dokumentów PDF oraz animacji Flash.

Wyróżnia się trzy główne obszary, w których stosuje się PHP:

- skrypty po stronie serwera,
- skrypty wywoływane z wiersza poleceń,
- aplikacje po stronie klienta.

Skrypt wykonywany po stronie serwera jest niewidoczny dla użytkownika. Pozwala na dynamiczne generowanie strony bądź ograniczanie dostępu do danych na podstronie naszej aplikacji.

Skrypty PHP są plikami tekstowymi, można je zatem tworzyć w dowolnym edytorze tekstu. Podczas prezentacji przykładów będziemy korzystali z klasycznego sposobu dołączania kodu PHP, a zatem skrypty zostaną umieszczone między znacznikami **<?php** i **?>**. Pierwszy z nich otwiera sekcję kodu PHP, drugi ją zamyka. Liczba sekcji PHP nie jest ograniczona, mogą one być umieszczane w dowolnym miejscu strony.

Kod zapisany w języku PHP może zostać wstawiony bezpośrednio do kodu HTML.

PRZYKŁAD 46.1

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Pierwszy skrypt</title>
</head>
<body>
    <?php
       echo "To jest nasz pierwszy skrypt";
    ?>
</body>
</html>
```

W przykładzie przedstawiono skrypt umieszczony w dokumencie HTML, którego zadaniem jest wyświetlenie w oknie przeglądarki komunikatu: „To jest nasz pierwszy skrypt". Aby zobaczyć efekt jego działania, należy go uruchomić w przeglądarce internetowej. Plik **`index.php`**, zawierający kod źródłowy z przykładu 46.1, zapisano w podkatalogu R8 katalogu głównym serwera WWW (**`C:\xampp\htdocs\R8`**). W przeglądarce należy wpisać adres http:// http://localhost/r8/index.php lub http://127.0.0.1/r8/index.php. Efekt działania skryptu przedstawiono na rys. 46.1.

Rys. 46.1. Efekt działania skryptu z przykładu 46.1

Aby obejrzeć kod strony z pozycji przeglądarki (np. w Mozilla Firefox), wystarczy w jej oknie nacisnąć prawy klawisz myszy i wybrać **`Pokaż źródło strony`**. Spowoduje to wyświetlenie kodu HTML strony.

Rys. 46.2. Podgląd źródła strony z przykładu 46.1 w przeglądarce Mozilla Firefox

Jak widać, w podglądzie nie ma znaczników PHP, pozostały wyłącznie znaczniki HTML. W pozycji skryptu wstawiono treść, która może być poprawnie zinterpretowana przez przeglądarkę. Kod PHP wykonano na serwerze, natomiast przeglądarka wyświetliła go jak zwykły kod HTML.

Komentarze

W skrypcie PHP można wstawiać **komentarze**. Czasem zachodzi potrzeba umieszczania opisu danego fragmentu kodu dla kogoś innego bądź nawet dla siebie (zwłaszcza jeśli pracuje się nad dużym projektem). Komentarze przydają się także do tymczasowego „wyłączania" niektórych linii kodu.

Istnieją trzy sposoby umieszczania komentarza w PHP:

- komentarz jednowierszowy znany z języka C++,
- komentarz jednowierszowy uniksowy,
- komentarz wielowierszowy.

PRZYKŁAD 46.2

```
<?php
echo "Przykłady komentarzy";

//Komentarz jednowierszowy

/*Komentarz składający się
z wielu
wierszy*/

# Komentarz jednowierszowy Uniksowy
?>
```

Komentarz blokowy zaczyna się od znaków `/*` i kończy znakami `*/`. Komentarz jednowierszowy natomiast zaczyna się od znaków `//` lub od znaku `#` i kończy się w bieżącej linii skryptu.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Przedstaw charakterystykę języka PHP.
2. W jaki sposób tworzy się komentarze w języku PHP?
3. Do czego służy polecenie ECHO?
4. Na czym polega działanie skryptu po stronie serwera?

47 Typy zmiennych i rodzaje operatorów

ZAGADNIENIA

- Definicja zmiennej
- Typy skalarne w języku PHP
- Typy złożone w języku PHP
- Typy specjalne w języku PHP
- Operatory działające na zmiennych

Zmienne

Zmienna to identyfikator znakowy, któremu przypisano jakąś wartość. Zmienna służy do przechowywania danych i wyników wykonywanych operacji. Wartością zmiennej mogą być liczba, łańcuch znaków, tablica lub obiekt.

Zmienna jest inicjalizowana (czyli jest rezerwowany dla niej pewien obszar w pamięci) podczas jej pierwszego użycia. Nie trzeba wcześniej deklarować zmiennej ani określać jej typu (tak jak w przypadku C++).

Nazwy zmiennych muszą zaczynać się od litery (dużej lub małej) lub znaku podkreślenia „ _”. W nazwach rozróżnia się wielkość liter i można stosować polskie znaki. Przed nazwą zmiennej należy umieścić znak **$**. Do nadawania wartości zmiennym używa się operatora przypisania „**=**” (tak jak w C++ i JavaScript).

Sposób przypisywania wartości zmiennym i wykorzystanie ich w poleceniu echo przedstawiono w przykładzie 47.1.

PRZYKŁAD 47.1

```
<?php
$x=18;                          //x jest zmienną numeryczną
$tekst="Anna Kowalska";         //tekst jest łańcuchem znaków, czyli
tekstem
$y=$x-5;                        //zmiennej y jest przypisana wartość
                                zmiennej x pomniejszoną o 5
echo "Witamy na stronie użytkownika:<b> $tekst </b>";
echo "<br>Wykonajmy dodawanie liczb x oraz y: <br>";
echo "$x + $y = ";
echo $x + $y;
```

```
echo "<br>Za pomocą jednej instrukcji echo możemy otrzymać to
samo:<br>";
echo "$x + $y = ".($x + $y);?>
```

W skrypcie zdefiniowano trzy zmienne. Zmienne **$x** oraz **$y** są zmiennymi liczbowymi, natomiast **$tekst** jest tekstem. Każdej z nich przypisano wartości, które wyświetliły się za pomocą instrukcji **echo**.

Rys. 47.1. Efekt działania instrukcji **echo**

Instrukcja echo umożliwia wyświetlanie komunikatu w oknie przeglądarki. Komunikat może być połączeniem tekstu z dowolną zmienną. Umieszcza się go w cudzysłowie. Aby w tej instrukcji wykonać działanie arytmetyczne na określonych zmiennych, nie należy go umieszczać w cudzysłowie. Dlatego też instrukcja **echo "$x + $y = "** spowoduje wyświetlenie tekstu 18 + 13 = , natomiast **echo $x + $y** obliczy sumę liczb 18 i 13. Ostatnia linijka skryptu przedstawia sposób łączenia tekstu z wartością wyświetlanej zmiennej. W JavieScript tego typu łączenie odbywało się za pomocą znaku plus „+”, natomiast w języku PHP stosuje się znak kropki.

UWAGA

Wartość przypisywana dla danej zmiennej jednocześnie określa typ tej zmiennej.

W trakcie działania skryptu rodzaj przechowywanych wartości przez zmienną może się zmienić. Może więc się zdarzyć, że trzeba będzie sprawdzić, jaki typ ma zmienna w danej chwili. W tym celu można zastosować jedną z funkcji przedstawionych w tabeli 47.1.

Tabela 47.1. Funkcje kontrolujące typy zmiennych

Nazwa funkcji	Opis działania
`is_array()`	Zwraca wartość **TRUE**, jeżeli zmienna zawiera wartość typu **array**, w przeciwnym razie zwraca wartość **FALSE**
`is_bool()`	Zwraca wartość **TRUE**, jeżeli zmienna zawiera wartość typu **boolean**, w przeciwnym razie zwraca wartość **FALSE**

Nazwa funkcji	Opis działania
`is_float()`	Zwraca wartość **TRUE**, jeżeli zmienna zawiera wartość typu **float**, w przeciwnym razie zwraca wartość **FALSE**
`is_integer()`	Zwraca wartość **TRUE**, jeżeli zmienna zawiera wartość typu **integer**, w przeciwnym razie zwraca wartość **FALSE**
`is_null()`	Zwraca wartość **TRUE**, jeżeli zmienna zawiera wartość typu **null**, w przeciwnym razie zwraca wartość **FALSE**
`is_string()`	Zwraca wartość **TRUE**, jeżeli zmienna zawiera wartość typu **string** (łańcuch znaków), w przeciwnym razie zwraca wartość **FALSE**
`gettype()`	Zwraca ciąg znaków określających typ zmiennej lub ciąg **unknown**, gdy typ jest nieokreślony

Jak już wcześniej wspomniano, typ zmiennej określa się w momencie przypisywania dla niej wartości.

Podstawowymi typami zmiennych w języku PHP są:

- typy podstawowe,
- typy złożone,
- typy specjalne.

Typy podstawowe

Dzielą się one na:

- typ **BOOLEAN**,
- typ **INTEGER**,
- typ **FLOAT**,
- typ **STRING**.

Typ **BOOLEAN** jest typem logicznym, który przyjmuje tylko dwie wartości: **TRUE** lub **FALSE**. Jest on wykorzystywany podczas konstruowania wyrażeń logicznych bądź sprawdzania warunków.

Typ **INTEGER** to typ liczb całkowitych, który reprezentuje dodatnie oraz ujemne liczby całkowite. Mogą one być zapisane w czterech różnych formatach:

- dziesiętnym (domyślny);
- ósemkowym (**0** przed liczbą);
- szesnastkowym (**0x** przed liczbą) – w zapisie liczby można stosować zarówno małe, jak i wielkie litery;
- binarnych (**0b** przed liczbą) – format od wersji PHP 5.4.

Maksymalny zakres typu całkowitego zależy od platformy sprzętowo-systemowej, na której jest uruchomiony PHP. W razie przekroczenia zakresu wartość jest konwertowana na typ **FLOAT**.

PRZYKŁAD 47.2

2356	dodatnia liczba całkowita w formacie dziesiętnym
–526	ujemna liczba całkowita w formacie dziesiętnym
012	dodatnia liczba całkowita w formacie ósemkowym
–024	ujemna liczba całkowita w formacie ósemkowym
0xDA	dodatnia liczba całkowita w formacie szesnastkowym
–0x0d	ujemna liczba całkowita w formacie szesnastkowym
0b0111	dodatnia liczba całkowita w formacie binarnym
–0b1011	ujemna liczba całkowita w formacie binarnym

Typ **FLOAT** reprezentuje dodatnie i ujemne liczby rzeczywiste (zmiennoprzecinkowe, zmiennopozycyjne). Ich zakres, tak jak dla liczb całkowitych, zależy od platformy sprzętowo-systemowej, na której jest uruchamiany PHP. Z reguły są to liczby z przedziału od $-1{,}8*10^{308}$ do $1{,}8*10^{308}$. Do zapisu tego typu liczb stosuje się kropki dziesiętne lub notację wykładniczą.

PRZYKŁAD 47.3

1.1	dodatnia liczba rzeczywista
–12.1156	ujemna liczba rzeczywista
5.1E3	dodatnia liczba rzeczywista w postaci wykładniczej
–2.3E-3	ujemna liczba rzeczywista w postaci wykładniczej

Typ **STRING** to typ łańcuchowy, który służy do zapamiętywania sekwencji znaków. Maksymalna długość łańcucha wynosi $2^{31}–1$ bajtów. Do zapamiętywania pojedynczego znaku wykorzystuje się 1 bajt.

Łańcuch znaków można tworzyć na cztery sposoby:

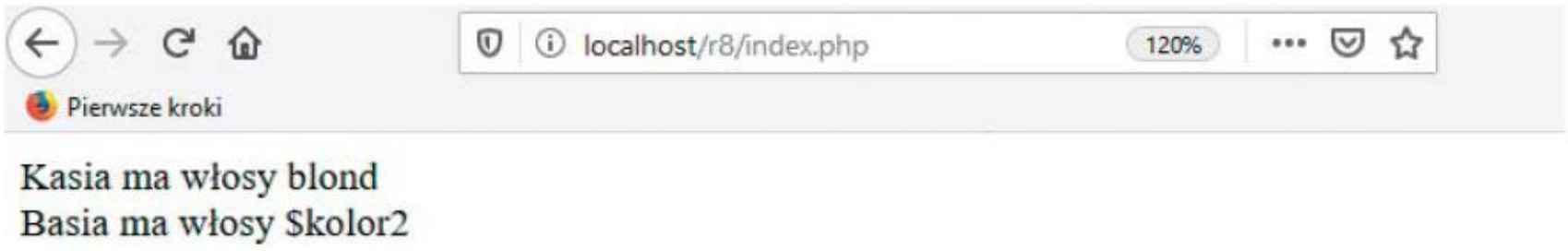
- z wykorzystaniem apostrofów,
- z wykorzystaniem cudzysłowów,
- z użyciem składni **heredoc**,
- z użyciem składni **nowdoc**.

Stosowanie zarówno cudzysłowów, jak i apostrofów jest poprawne. Efekt wykonania będzie identyczny, ale różnica polega na interpretacji danej zawartości i szybkości przetwarzania kodu.

PRZYKŁAD 47.4

```
<?php
$kolor1="blond";
$kolor2='rude';
echo "Kasia ma włosy $kolor1<br>";
echo 'Basia ma włosy $kolor2';
?>
```

Efektem skryptu będą komunikaty w postaci przedstawionej na rys. 47.2.



Rys. 47.2. Efekt działania apostrofów i cudzysłowów

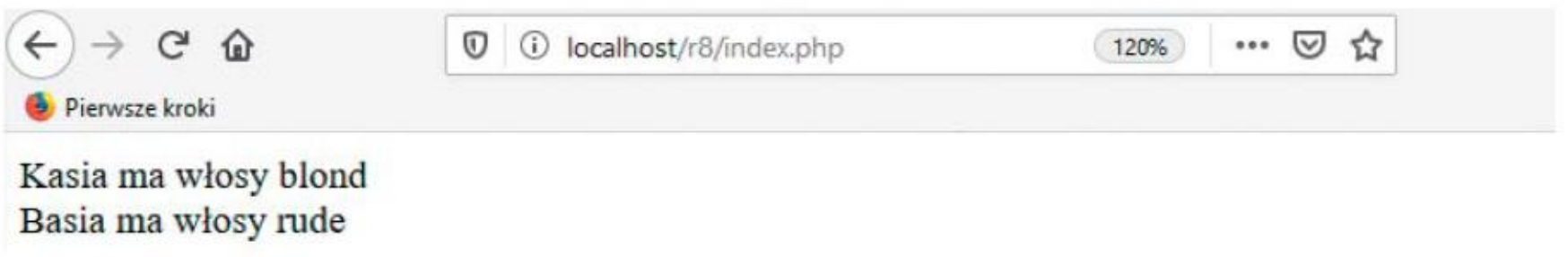
Jak widać, wyświetlanie zmiennych **`$kolor1`** i **`$kolor2`** daje inny wynik. Zmienne tekstowe umieszczone między cudzysłowami są interpretowane przez PHP. Ciąg tekstowy umieszczony w apostrofach jest traktowany jak zwykły tekst i nie jest w żaden sposób przetwarzany. W wyniku użycia cudzysłowów ciąg jest dodatkowo interpretowany, czyli nazwa zmiennej zostanie zamieniona na jej wartość.

Pozostaje zatem zadać pytanie, czy dzięki stosowaniu apostrofów można wyświetlić wartość zmiennej. Wystarczy, tak jak w przykładzie 47.5, zastosować konkatenację, czyli operator kropki.

PRZYKŁAD 47.5

```
<?php
$kolor1="blond";
$kolor2='rude';
echo "Kasia ma włosy $kolor1<br>";
echo 'Basia ma włosy '.$kolor2;
?>
```

Dzięki zastosowaniu operatora kropki otrzymuje się następujący efekt:



Rys. 47.3. Zastosowanie konkatenacji w instrukcji **`echo`**

Jak widać, lepiej używać apostrofów, ponieważ interpreter nie musi niepotrzebnie dodatkowo przetwarzać łańcuchów znaków.

W przypadku wykorzystania składni **`heredoc`** łańcuch znakowy rozpoczyna się od sekwencji **`<<<`**, po której występuje identyfikator. Nazwy identyfikatora tworzy się według takich samych reguł jak nazwy zmiennych. Tego samego identyfikatora należy użyć na końcu łańcucha znakowego. Linia kończąca nie może zawierać nic więcej niż identyfikator i średnik.

Przykład zastosowania składni **heredoc** przedstawiono w przykładzie 47.6.

PRZYKŁAD 47.6

```
<?php
$pytanie="<br>Czy znasz odpowiedź?";
$zagadka= <<<EOD
Co stanie się o połowę większe, gdy postawimy to do góry
nogami? $pytanie
EOD;
echo $zagadka;
?>
```

Efekt kodu z przykładu 47.6 przedstawiono na rys. 47.4.

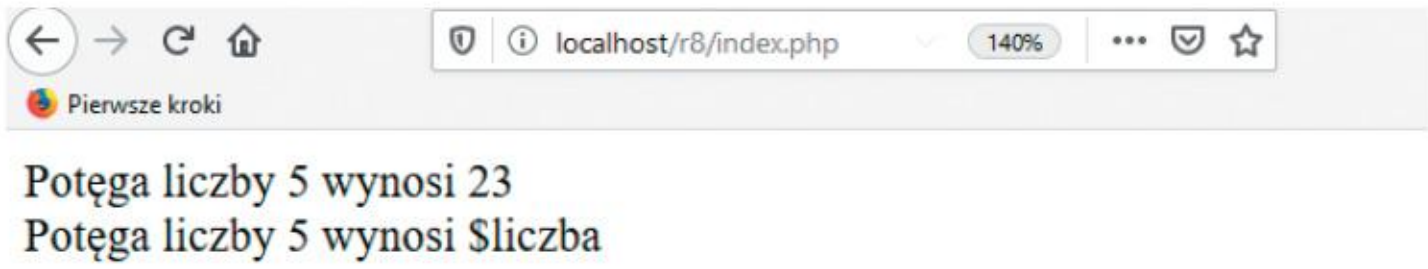
Rys. 47.4. Efekt działania skryptu z przykładu 47.6

Składnia **nowdoc** jest podobna do składni **heredoc**. Różnica polega na tym, że ciąg nie jest interpretowany, czyli nazwy zmiennych nie są zmieniane na ich wartości, oraz identyfikator rozpoczynający sekwencję jest umieszczony w apostrofach. Tę składnię wprowadzono do PHP w wersji 5.3.0.

PRZYKŁAD 47.7

```
<?php
$liczba=5*5-16/8;
$komunikat = <<<EOD
Wartość zmiennej liczba wynosi $liczba
EOD;
echo $komunikat;
$nowy = <<<'EOD'
<br>Wartość zmiennej liczba wynosi $liczba
EOD;
echo $nowy;
?>
```

Efekt kodu z przykładu 47.7 przedstawiono na rys. 47.5.



Rys. 47.5. Wynik działania skryptu z przykładu 47.7

Typy złożone

Wyróżnia się dwa rodzaje **typów złożonych**:

- typ tablicowy **ARRAY**,
- typ obiektowy **OBJECT**.

Tablice to zmienne pozwalające na przechowywanie zbioru danych określonego typu. W PHP tablice mogą być indeksowane klasycznie (numerycznie) lub asocjacyjnie. Tablice są jednowymiarowe (postać wektora) lub wielowymiarowe (postać macierzowa).

Aby utworzyć tablicę indeksowaną numerycznie, należy zastosować słowo kluczowe **array** w następującej postaci:

```
$nazwa_tablicy = array(wart1, wart2, ….., wartN);
```

PRZYKŁAD 47.8

Definicja i inicjalizacja tablicy jednowymiarowej pięcioelementowej przez automatyczne tworzenie indeksu:

```
$tablica = array(2,15, 34,-5,45);
```

lub

```
$tablica = [2,15, 34,-5,45];
```

Aby wyświetlić wartość pierwszego elementu tablicy, należy zastosować instrukcję:

```
echo $tablica[0];
```

UWAGA

Podobnie jak w języku C++, elementy tablicy są numerowane od zera.
Składnia tworzenia tablicy indeksowanej przez ręczne przypisanie indeksu ma postać:

```
<?php
$tablica[0]=7;
$tablica[1]=-3;
$tablica[2]=0;
?>
```

Numer indeksu może zostać pominięty, nawiasy muszą zostać puste. PHP nada numery indeksów automatycznie. Niezależnie od sposobu utworzenia tablicy, można do niej dodawać kolejne elementy. Jeśli podczas przypisywania kolejnych wartości nie wpiszemy numerów indeksów, zostaną one uzupełnione automatycznie.

PRZYKŁAD 47.9

```
<?php
$tablica[]="Anna";
$tablica[]="Joanna";
$tablica[]="Katarzyna";
?>
```

Tablice asocjacyjne (tablice skojarzeniowe) to tablice, w których indeksami mogą być dowolne ciągi znaków, tzw. klucze. W tablicy tego typu występują pary klucz – wartość, w których każdy klucz jednoznacznie identyfikuje daną wartość.

Definicja tablicy asocjacyjnej ma postać:

```
$tablica = array(klucz1 => wartość1,
klucz2 => wartość2,...
kluczN => wartość);
```

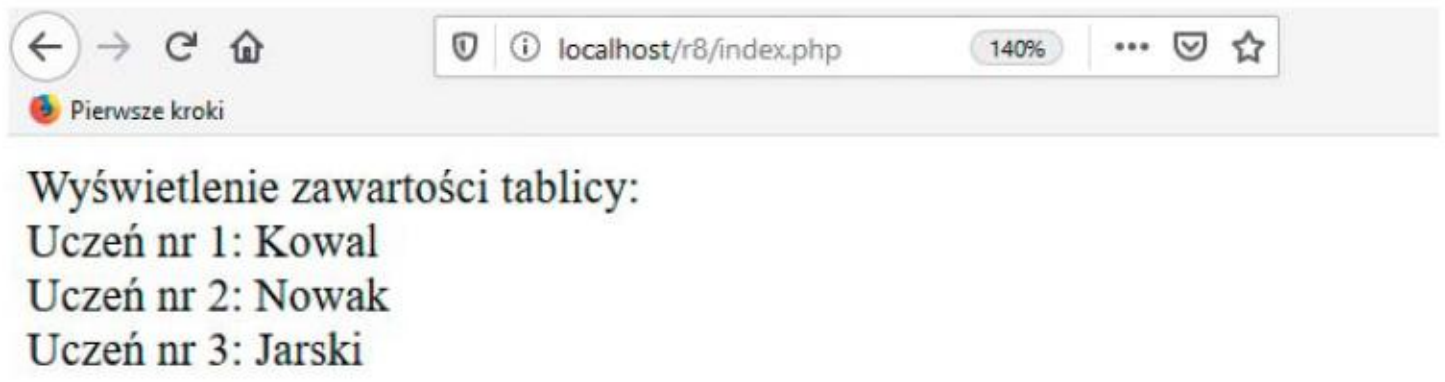
lub

```
$tablica[klucz1]=wartosc1;
$tablica[klucz2]=wartosc2;
...
$tablica[kluczN]=wartoscN;
```

PRZYKŁAD 47.10

```
<?php
$uczniowie=array("u1"=>"Kowal","u2"=>"Nowak","u3"=>"Jarski");
echo 'Wyświetlenie zawartości tablicy:<br>';
echo "Uczeń nr 1: ".$uczniowie['u1'].'<br>';
echo "Uczeń nr 2: ".$uczniowie['u2'].'<br>';
echo "Uczeń nr 3: ".$uczniowie['u3'].'<br>';
?>
```

W skrypcie zdefiniowano tablicę **$uczniowie** zawierającą trzy klucze o nazwach **u1**, **u2**, **u3**. Kluczowi **u1** przypisano ciąg znaków **Kowal**, kluczowi **u2** – ciąg znaków **Nowak**, a kluczowi **u3** – ciąg znaków **Jarski**. Wszystkie wartości tablicy wyświetlono w oknie przeglądarki.



Rys. 47.6. Efekt skryptu wyświetlającego zawartość tablicy asocjacyjnej

Typ **OBJECT** jest typem obiektowym służącym do przechowywania obiektów.

Typy specjalne

Typami specjalnymi są:

- **RESOURCE**,
- **NULL**.

Typ **RESOURCE** jest typem specjalnym, wskazującym na to, że zmienna przechowuje odwołanie do zasobu zewnętrznego za pomocą specjalnych funkcji.

Typ **NULL** informuje, że dana zmienna nie przechowuje żadnej wartości, czyli jest pusta. Aby zdefiniować zmienną typu **NULL**, należy zastosować zapis:

```
$nazwa_zmiennej = NULL;
```

UWAGA

Wielkość liter w pisowni NULL nie ma znaczenia.

Operatory działające na zmiennych

Operatory służą do wykonywania działań na zmiennych. Wyróżnia się operatory:

- arytmetyczne,
- porównania,
- bitowe,
- logiczne,
- przypisania,
- inkrementacji i dekrementacji.

Operatory arytmetyczne służą do wykonywania operacji na zmiennych numerycznych.

Tabela 47.2. Operatory arytmetyczne

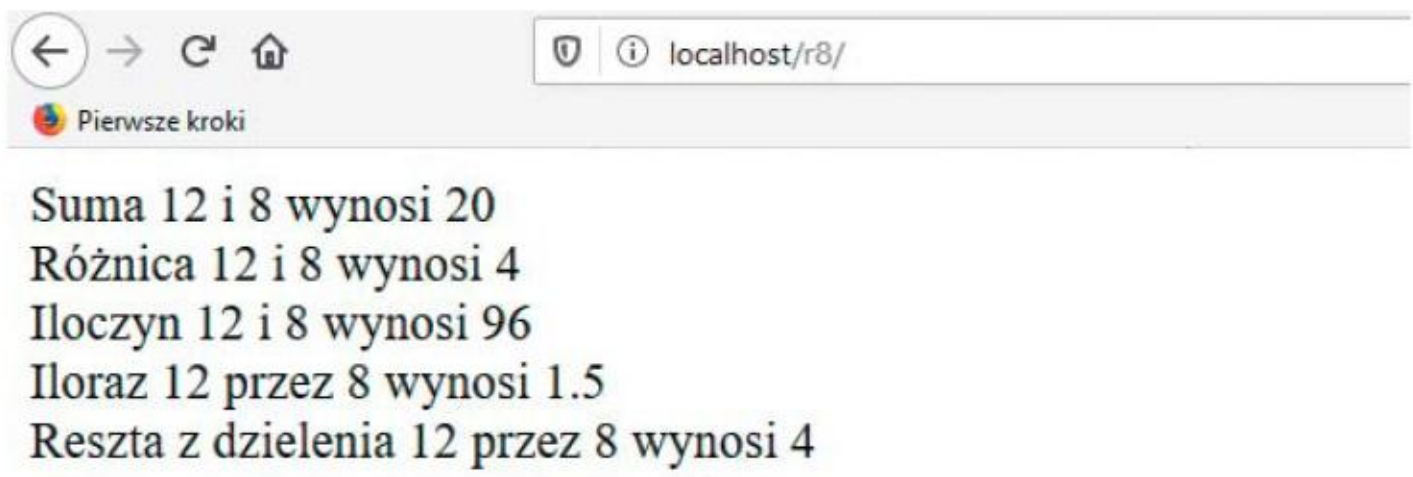
Operator	Przykład	Wynik
+	$a + $b	suma $a i $b
-	$a - $b	różnica $a i $b
*	$a * $b	iloczyn $a i $b
/	$a / $b	iloraz $a i $b
%	$a % $b	reszta z dzielenie $a przez $b

PRZYKŁAD 47.11

```
<?php
$x=12;
$y=8;
$suma=$x+$y;
```

```
$roznica=$x-$y;
$iloczyn=$x*$y;
$iloraz=$x/$y;
$reszta=$x%$y;
echo "Suma $x i $y wynosi $suma<br>";
echo "Różnica $x i $y wynosi $roznica<br>";
echo "Iloczyn $x i $y wynosi $iloczyn<br>";
echo "Iloraz $x przez $y wynosi $iloraz<br>";
echo "Reszta z dzielenia $x przez $y wynosi $reszta<br>";
?>
```

Efektem skryptu będzie wyświetlenie następujących komunikatów:



Rys. 47.7. Efekt działania skryptu z przykładu 47.11

Operatory porównania porównują wartości dwóch zmiennych. Ich wynikiem jest wartość logiczna **TRUE** lub **FALSE**. Stosuje się je najczęściej w celu sprawdzenia poprawności danych wprowadzanych przez użytkownika lub wyeliminowania błędów podczas wykonywanych obliczeń.

Tabela 47.3. Operatory porównania

Operator	Przykład	Wynik
= =	$a==$b	TRUE, jeżeli $a jest równe $b
= = =	$a===$b	TRUE, jeżeli $a jest równe $b i są one tych samych typów
!=	$a!= b	TRUE, jeżeli $a jest różne od $b
<	$a<$b	TRUE, jeżeli $a jest mniejsze od $b
>	$a>$b	TRUE, jeżeli $a jest większe od $b
<=	$a<=$b	TRUE, jeżeli $a jest mniejsze lub równe $b
>=	$a>=$b	TRUE, jeżeli $a jest większe lub równe $b

Operatory bitowe służą do wykonywania działań na poszczególnych bitach liczb.

Tabela 47.4. Operatory bitowe

Operator	Przykład	Wynik
`&`	`$a & $b`	iloczyn bitowy **AND**
`|`	`$a | $b`	suma bitowa **OR**
`^`	`$a ^ $b`	różnica symetryczna **XOR**
`~`	`~$a`	negacja **NOT**
`<<`	`$a << $b`	przesunięcie w lewo
`>>`	`$a >> $b`	przesunięcie w prawo

Iloczyn bitowy to inaczej koniunkcja bitowa (AND). Są włączone tylko te bity (mające wartość 1), które były włączone w obu argumentach. Wynik działania iloczynu bitowego przedstawiono w tabeli 47.5.

Tabela 47.5. Tabela wyników działania iloczynu bitowego

$a	$b	$a & $b
0	0	0
0	1	0
1	0	0
1	1	1

Suma bitowa to inaczej alternatywa bitowa (OR). Są włączone te bity, które były włączone przynajmniej w jednym argumencie. Wynik działania sumy bitowej przedstawiono w tabeli 47.6.

Tabela 47.6. Tabela wyników działania sumy bitowej

$a	$b	$a | $b
0	0	0
0	1	1
1	0	1
1	1	1

Negacja bitowa (NOT) to działanie, w którym bit jest ustawiany na 1, gdy bit argumentu jest ustawiony na 0. Wynik działania negacji bitowej przedstawiono w tabeli 47.7.

Tabela 47.7. Tabela wyników działania negacji bitowej

$a	~$a
0	1
1	0

Bitowa różnica symetryczna to inaczej różnica symetryczna (XOR). Są włączone tylko te bity, które miały różną wartość dla obu argumentów. Wynik działania XOR na pojedynczych bitach przedstawiono w tabeli 47.8.

Tabela 47.8. Tabela wyników działania bitowej różnicy symetrycznej

$a	$b	$a ^ $b
0	0	0
0	1	1
1	0	1
1	1	0

Działanie operatorów bitowych przedstawiono w przykładzie 47.12.

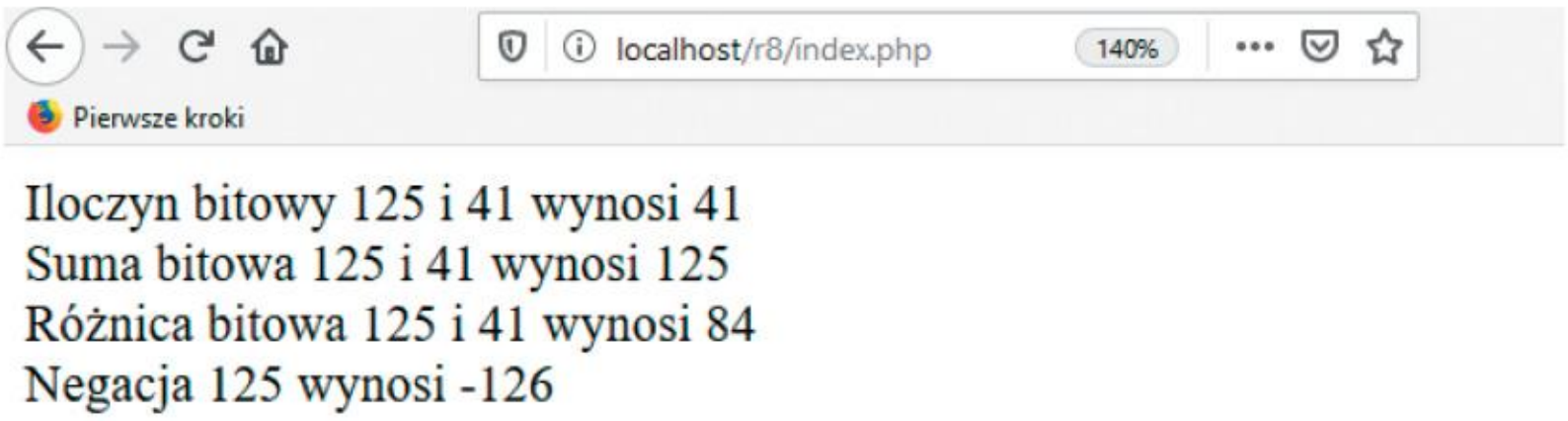
PRZYKŁAD 47.12

```
<?php
$x=125;
$y=41;
$iloczyn_bitowy=$x & $y;
$suma_bitowa=$x | $y;
$roznica_sym=$x^$y;
$negacja=~$x;
echo "Iloczyn bitowy $x i $y wynosi $iloczyn_bitowy<br>";
echo "Suma bitowa $x i $y wynosi $suma_bitowa<br>";
echo "Różnica bitowa $x i $y wynosi $roznica_sym<br>";
echo "Negacja $x wynosi $negacja<br>";
?>
```

Aby zrozumieć operacje podane w skrypcie z przykładu 47.12, należy liczby 125 i 41 oraz poszczególne działania na tych liczbach zapisać w postaci binarnej.

```
125       = 0111 1101(2)
41        = 0010 1001(2)
125 & 41  = 0111 1101(2) & 0010 1001(2) =  0010 0100(2) = 41
125 | 41  = 0111 1101(2) | 0010 1001(2) =  0111 1101(2) = 175
125 ^ 41  = 0111 1101(2) ^ 0010 1001(2) =  0101 0100(2) = 84
~125      = ~0111 1101(2)               =  -126
```

Efekt działania kodu z przykładu 47.12 przedstawiono na rys. 47.8.



Rys. 47.8. Wynik działania skryptu z przykładu 47.12

Przesunięcie bitowe w lewo (**w prawo**) to działanie polegające na przesunięciu wszystkich bitów argumentu znajdującego się z lewej strony operatora w lewo (w prawo) o liczbę miejsc wskazaną przez argument znajdujący się z prawej strony operatora.

PRZYKŁAD 47.13

Operacja

```
$liczba=68 << 1;
```

spowoduje przypisanie zmiennej $liczba wartość 136. Liczba 68 w zapisie binarnym jest równa 0100 0100. Po przesunięciu wszystkich bitów o jedno miejsce otrzymamy liczbę 10001000.

Operatory logiczne służą przede wszystkim do sprawdzania warunków w instrukcjach sterujących. Tak jak w przypadku innych języków programowania, do dyspozycji pozostają iloczyn logiczny, suma logiczna oraz negacja.

Tabela 47.9. Operatory logiczne

Operator	Przykład	Wynik
`&&`	`$a && $b`	iloczyn logiczny **$a** i **$b** zwraca **TRUE**, jeżeli **$a** i **$b** mają wartość **TRUE**
`||`	`$a || $b`	suma logiczna **$a** i **$b** zwraca **TRUE**, jeżeli **$a** i **$b** mają wartość **TRUE**
`!`	`!$a`	negacja **$a** **TRUE**, jeżeli **$a** nie jest **TRUE**.

Operatory przypisania dają możliwość zmiany wartości zmiennej. Najczęściej stosuje się operator „=”, za pomocą którego nadaje się wartość danej zmiennej. Pozostałe z nich to operatory złożone, ponieważ wykonują dwa działania naraz. Są to skrócone zapisy operacji zwiększania lub zmniejszania wartości zmiennej o określoną wartość wyrażenia.

Tabela 47.10. Operatory przypisania

Operator	Przykład	Wynik
=	$a = $b	przypisuje wartość $b do $a
+=	$a += $b	przypisuje wartość ($a+$b) do $a jest to identyczne z $a=$a+$b
-=	$a -= $b	przypisuje wartość ($a-$b) do $a jest to identyczne z $a=$a-$b
*=	$a *= $b	przypisuje wartość ($a*$b) do $a jest to identyczne z $a=$a*$b
/=	$a /= $b	przypisuje wartość ($a/$b) do $a jest to identyczne z $a=$a/$b
%=	$a %= $b	przypisuje wartość ($a%$b) do $a jest to identyczne z $a=$a%$b
|=	$a |= $b	przypisuje wartość ($a\|$b) do $a jest to identyczne z $a=$a|$b
&=	$a &= $b	przypisuje wartość ($a&$b) do $a jest to identyczne z $a=$a&$b
^=	$a ^= $b	przypisuje wartość ($a^$b) do $a jest to identyczne z $a=$a^$b
<<=	$a <<= $b	przypisuje wartość ($a<<$b) do $a jest to identyczne z $a=$a<<$b
>>=	$a >>= $b	przypisuje wartość ($a>>$b) do $a jest to identyczne z $a=$a>>$b

Operatory inkrementacji i **dekrementacji** są operatorami jednoargumentowymi. Ich zadaniem jest zwiększanie lub zmniejszanie wartości zmiennej o 1.

Tabela 47.11. Operatory inkrementacji i dekrementacji

Operator	Nazwa	Wynik
$a++	postinkrementacja	zwraca $a, a następnie zwiększa $a o jeden
++$a	preinkrementacja	zwiększa $a o jeden i zwraca $a
$a--	postdekrementacja	zwraca $a, a następnie zmniejsza $a o jeden
--$a	predekrementacja	zmniejsza $a o jeden i zwraca $a

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień typy skalarne w języku PHP.
2. W jaki sposób zapisać liczbę w postaci szesnastkowej?
3. Czym różni się stosowanie cudzysłowów od apostrofów?
4. W jaki sposób definiuje się tablice asocjacyjne?
5. Wymień typy specjalne w języku PHP.
6. Wymień znane ci operatory przypisania.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz skrypt, który w oknie przeglądarki wyświetli resztę z dzielenia liczby 23 przez 7.
2. Utwórz skrypt, który dla `$x=` oraz `$y=6` obliczy wartość wyrażenia `(2*$x-0.5*$y)/4`. Wynik wyświetl w oknie przeglądarki.
3. Utwórz skrypt, który dla `$x=57` oraz `$y=208` obliczy i wyświetli w oknie przeglądarki iloczyn, sumę i różnicę bitową tych liczb.

48 Instrukcje sterujące

ZAGADNIENIA

- Instrukcja warunkowa
- Instrukcja `SWITCH`
- Operator warunkowy
- Pętla `FOR`
- Pętla `WHILE`
- Pętla `DO...WHILE`
- Pętla `FOREACH`

Instrukcja warunkowa

Instrukcje warunkowe są podstawą każdego języka programowania. Mają one za zadanie wykonać określone instrukcje zależnie od spełnienia pewnego warunku. Instrukcja warunkowa występuje w dwóch postaciach: niepełnej i pełnej.

Najprostszą instrukcją warunkową jest **niepełna instrukcja warunkowa**:

```
if (wyrażenie)
     {
     //Instrukcje, które mają się wykonać po spełnieniu warunku
     }
```

Jeśli wyrażenie podane w nawiasie ma wartość logiczną TRUE, to są wykonywane instrukcje ograniczone klamrami. W przeciwnym razie następuje zakończenie programu albo przejście do kolejnych sekwencji programu.

PRZYKŁAD 48.1

```
<?php
$x=126;
if ($x%2==0)
{
  echo 'Liczba jest parzysta';
}
?>
```

Jeżeli zmienna **$x** jest liczbą parzystą (czyli reszta z dzielenia przez 2 wynosi 0), to wyświetli się tekst „Liczba jest parzysta". Instrukcję można rozbudować przez dodanie słowa kluczowego **else**, po którym zostaną określone instrukcje, które mają być wykonane, jeśli liczba nie jest parzysta.

PRZYKŁAD 48.2

```
<?php
$x=125;
    if ($x%2==0)
       {
          echo 'Liczba jest parzysta';
       }
    else
       {
          echo 'Liczba jest nieparzysta';
       }
?>
```

Klauzula **else** określa, co ma się stać, gdy warunek nie zostanie spełniony. Jeśli trzeba sprawdzić kilka wersji, należy skorzystać z klauzuli **elseif**.

PRZYKŁAD 48.3

Weźmy pod uwagę program, który sprawdza, czy podana liczba jest podzielna przez 35. Wiadomo, że każda liczba podzielna przez 35 jest podzielna przez 5 i przez 7. Do sprawdzenia tych warunków można użyć kilku oddzielnych instrukcji niepełnych warunkowych. Jeśli jednak podamy liczbę podzielną przez 35, to będą spełnione trzy warunki i zostaną wykonane trzy instrukcje. Jeśli zastosujemy klauzulę **elseif**, to zostanie wykonana tylko jedna instrukcja.

```
<?php
$x=21;
//sprawdzenie, czy x jest podzielne przez 35
if ($x%35==0)
    {
    echo "Liczba $x jest podzielna przez 35";
    }
//sprawdzenie, czy x jest podzielne przez 5
elseif ($x%5==0)
    {
    echo "Liczba $x jest podzielna przez 5, ale nie jest
    podzielna przez 7";
    }
```

```
//sprawdzenie, czy x jest podzielne przez 7
elseif ($x%7==0)
    {
    echo "Liczba $x jest podzielna przez 7, ale nie jest
    podzielna przez 5";
    }
else
    {
    echo "Liczba $x nie jest podzielna przez 5 i nie jest
    podzielna przez 7";
    }
?>
```

Dzięki zastosowaniu **elseif** można w jednym podejściu rozważyć kilka możliwości. Gdy jedna z nich zostanie spełniona, pozostałe nie będą już sprawdzane. Podobnie jak ze zwykłym **if**, można pominąć klamrowe nawiasy w przypadku tylko jednej instrukcji.

Oczywiście, tak jak w JavaScript, warunki można łączyć za pomocą operatorów logicznych.

PRZYKŁAD 48.4

```
<?php
$x=49;
if ($x%2==0 && $x%3==0)
    {
    echo "Liczba $x jest liczbą parzystą i podzielną przez 3";
    }
elseif ($x%2!=0 && $x%3==0)
    {
    echo "Liczba $x jest liczbą nieparzystą i podzielną przez 3";
    }
elseif ($x%2==0 && $x%3!=0)
    {
    echo "Liczba $x jest liczbą parzystą i niepodzielną przez 3";
    }
else
    {
    echo "Liczba $x jest liczbą nieparzystą i niepodzielną
    przez 3";
    }
?>
```

W podanym przykładzie sprawdza się, czy podana liczba jest podzielna przez 2 i jednocześnie podzielna przez 3. W tym celu zastosowano operator iloczynu logicznego **&&**.

Instrukcja SWITCH

Instrukcja **SWITCH** jest instrukcją wyboru. Wyrażenie występujące w tej konstrukcji jest porównywane z listą szukanych wartości aż do znalezienia wartości pasującej.

Postać instrukcji jest identyczna jak w języku C++ czy JavaScript:

```
switch(wyrażenie)
{
case wart1:
     instrukcje1;
     break;
case wart2:
     instrukcje2;
     break;
…
cane wartN:
     instrukcjeN;
     break;
default:
     instrukcje;
}
```

Instrukcja **SWITCH** sprawdza wartość wyrażenia. Jeżeli wynikiem jest **wart1**, to są wykonywane **instrukcje1** i następuje wyjście z bloku **SWITCH** (polecenie **BREAK**). Jeżeli wynikiem jest **wart2**, to są wykonywane **instrukcje2** itd. Jeśli wartość wyrażenia nie jest równa żadnej z podanych wartości, to są wykonywane instrukcje po słowie default.

PRZYKŁAD 48.5

```
<?php
$dzialanie='*';
$x=32;
$y=5;
switch($dzialanie)
{
    case '+':
       echo "Suma $x oraz $y wynosi ".($x+$y);
       break;
    case '-':
       echo "Różnica $x oraz $y wynosi ".($x-$y);
       break;
```

```
case '*':
      echo "Iloczyn $x oraz $y wynosi ".($x*$y);
      break;
case '/':
      if ($y!=0) echo "Iloraz $x przez $y wynosi ".($x/$y);
      else echo "Nie mogę wykonać dzielenia po y=0!";
      break;
case '%':
      echo "Reszta z dzielenia $x przez $y wynosi ".($x%$y);
      break;
default:
      echo "Nie wybrałeś żadnego działania";
      break;
}
?>
```

Efektem skryptu będzie wykonanie jednego z działań arytmetycznych: dodawania, odejmowania, mnożenia, dzielenia lub dzielenia z resztą. Jeśli wybrano inny znak niż +, -, *, / czy %, w oknie przeglądarki wyświetli się komunikat, że działania nie wybrano.

Każdą instrukcję **SWITCH** można zastąpić instrukcją warunkową z klauzulą **elseif**. Nie zawsze jest to opłacalna zamiana – ponieważ kod programu może stać się mniej czytelny. Gdyby skrypt z przykładu 48.5 napisać z użyciem zwykłej złożonej instrukcji warunkowej, miałby on następującą postać przedstawioną w przykładzie 48.6.

PRZYKŁAD 48.6

```
<?php
$dzialanie='*';
$x=32;
$y=0;
    if ($dzialanie=='+')
       echo "Suma $x oraz $y wynosi ".($x+$y);
    elseif ($dzialanie=='-')
       echo "Różnica $x oraz $y wynosi ".($x-$y);
    elseif ($dzialanie=='*')
       echo "Iloczyn $x oraz $y wynosi ".($x*$y);
    elseif ($dzialanie=='/')
       {
       if ($y!=0) echo "Iloraz $x przez $y wynosi ".($x/$y);
       else echo "Nie mogę wykonać dzielenia po y=0!";
       }
```

```
    elseif ($dzialanie=='%')
      echo "Reszta z dzielenia $x przez $y wynosi ".($x%$y);
    else
      echo "Nie wybrałeś żadnego działania";
?>
```

Operator warunkowy

Działanie operatora jest podobne do instrukcji warunkowej **IF**. Zwraca wartość jednego z dwóch wyrażeń rozdzielonych dwukropkiem. Operator warunkowy ma postać:

```
warunek ? wartosc1:wartosc2;
```

Jeżeli warunek jest spełniony, to wartością wyrażenia jest **wartosc1**. W przeciwnym przypadku wartością jest **wartosc2**.

PRZYKŁAD 48.7

```
<?php
$liczba1=56;
$warunek=($liczba1>=10  &&  $liczba1<=99)?  "Liczba  $liczba1
jest liczbą dwucyfrową" : "Liczba $liczba1 nie jest liczbą
dwucyfrową";
echo $warunek;
?>
```

Skrypt sprawdza, czy wartość wpisana pod zmienną **$liczba1** należy do przedziału <10,99>, a zatem czy jest liczbą dwucyfrową. Jeśli warunek jest typu **TRUE**, zmienna **$warunek** staje się komunikatem informującym użytkownika, że liczba jest dwucyfrowa. W przeciwnym razie **$warunek** informuje, że liczba nie jest dwucyfrowa.

Pętla FOR

Czasem zachodzi potrzeba wykonania jakiejś czynności określoną liczbę razy. Z pomocą przychodzi jedna z najczęściej używanych składni w większości języków programowania, czyli pętla **FOR**, którą stosuje się, gdy liczba wykonywanych powtórzeń jest znana.

Postać instrukcji

```
for (wyr_poczatkowe; wyr_koncowe;wyr_modyfikujące)
{
instrukcje;
}
```

Wyr_poczatkowe określa wartość początkową zmiennej, która jest licznikiem w pętli, **wyr_koncowe** określa warunek określający wartość końcową licznika, **wyr_modyfikujace** określa sposób modyfikacji licznika.

PRZYKŁAD 48.8

```
<?php
for ($i=1;$i<=50; $i++)
    {
       if($i%2==0)     echo $i."\t";
    }
?>
```

Zadaniem skryptu z przykładu 48.8 jest wyświetlenie liczb parzystych z przedziału [1,50]. Instrukcjami powtarzającymi się w pętli jest instrukcja warunkowa sprawdzająca podzielność kolejnych liczb naturalnych przez 2 oraz instrukcja echo wyświetlająca kolejne liczby parzyste. Pętla wykona się pięćdziesiąt razy, ponieważ wartość początkowa licznika **$i** jest równa 1, końcowa wynosi 50, a zmienna jest inkrementowana o 1 (**$i++**).

Pętlę **FOR** wykorzystuje się w działaniach na tablicach. Przykładem może być wylosowanie dla tablicy dwudziestoelementowej wartości z przedziału [1,20], wyświetlenie ich oraz obliczenie ich sumy. Aby wylosować liczby z danego przedziału, należy zastosować funkcję **rand()**.

PRZYKŁAD 48.9

```
<?php
$suma=0;
for ($i=1;$i<=20; $i++)
    {
       $tab[$i]=rand(1,20);
          echo $tab[$i]."\t";
    $suma+=$tab[$i];
    }
echo "<br>Suma wartości tablicy wynosi $suma";
?>
```

Efektem skryptu będzie wyświetlenie wartości tablicy i sumy tych liczb.

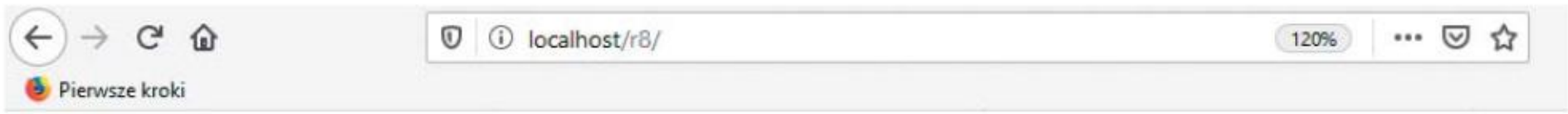
Rys. 48.1. Efekt działania skryptu z przykładu 48.9

Pętla **FOR** może być dowolnie zagnieżdżana, czyli jedną pętlę można wywoływać wewnątrz drugiej. Polega to na tym, że w jednej iteracji pętli zewnętrznej zostanie wykonany cały przebieg pętli wewnętrznej. Przykładem może być skrypt wyświetlający tabliczkę mnożenia.

PRZYKŁAD 48.10

```
<?php
echo '<table style="width:300px; height:300px;">';
for ($i=1;$i<=10;$i++)
{
echo "<tr>";
for($j=1;$j<=10;$j++)
{
echo "<td>".($i*$j)."</td>";
}
echo "</tr>";
}
echo "</table>";
?>
```

Efektem skryptu będzie wyświetlenie tabliczki mnożenia w postaci macierzowej.



1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Rys. 48.2. Efekt działania skryptu z przykładu 48.10

Pętla WHILE

Pętlę WHILE stosuje się, gdy liczba wykonywanych powtórzeń nie jest znana. Zakończenie działania pętli jest uzależnione od spełnienia określonego warunku.

Postać instrukcji **WHILE**:

```
while (warunek)
{
instrukcje;
}
```

Instrukcje wewnątrz pętli są wykonywane, dopóki wyrażenie warunkowe jest prawdziwe.

PRZYKŁAD 48.11

Skrypt z przykładu 48.8, wykorzystujący pętlę FOR, zastąpimy pętlą **WHILE**.

```
<?php
$i=1;                    //wartość początkowa dla pętli
while($i<=50)            //warunek zakończenia pętli
{
    if($i%2==0)
    echo $i."\t";
    $i++;                //zwiększenie i o jeden
}
?>
```

W przykładzie pokazano, że każdą instrukcję FOR można zamienić na instrukcję **WHILE**.

UWAGA

Zamiana odwrotna nie zawsze zachodzi.

Innym przykładem zastosowania pętli WHILE jest wyświetlenie podanej liczby w odwrotnej kolejności.

PRZYKŁAD 48.12

```
<?php
$liczba=132;
echo "Liczba $liczba napisana w odwrotnej kolejności to ";
while ($liczbaa>0)
    {
    //reszta z dzielenia przez 10 wyświetli ostatnią cyfrę
    echo $liczba%10;
    //dzielimy liczbę przez 10 i zaokrąglamy ją do liczby
    całkowitej
```

```
    $liczba=floor($liczba/10);
    }
?>
```

Liczbą wyświetloną w odwrotnej kolejności będzie 132. W tym celu należy pobrać kolejne cyfry, zaczynając od końca. W pętli zostaną wykonywane dwa działania: pierwsze wyznacza resztę z dzielenia liczby przez 10 (132%10=2), natomiast drugie dzieli liczbę przez 10 i zaokrągla ją do liczby całkowitej (132/10=13.2, po zaokrągleniu 13). Działania powtarza się do momentu, aż będzie liczbą dodatnią.

Kolejnym przykładem zastosowania **WHILE** jest wyznaczenie największego wspólnego dzielnika dla dwóch liczb. Algorytm wyszukiwania jest każdemu znany z lekcji matematyki.

PRZYKŁAD 48.13

```
<?php
$x = 345;
$y = 27;
while ($x != $y)
    {
       if ($x < $y) {
       //zamiana liczb miejscami, aby różnica była dodatnia
       $pomoc = $x;
       $x = $y;
       $y = $pomoc;
    }
$x = $x - $y;
}
echo "Największy wspólny dzielnik wynosi: $x";
?>
```

Pętla DO...WHILE

Stosuje się ją, gdy liczba wykonywanych powtórzeń jest nieznana. Jej działanie jest podobne do działania pętli **WHILE** – ponieważ zakończenie działania pętli zależy od spełnienia określonego warunku.

Postać instrukcji **WHILE**:

```
do
{
instrukcje;
}
while (warunek);
```

W pętli **DO...WHILE** instrukcje wewnątrz pętli są wykonywane przynajmniej raz, nawet jeśli wyrażenie warunkowe jest fałszywe – ponieważ najpierw wykonuje się instrukcje, a dopiero potem sprawdza warunek.

W skrypcie z przykładu 48.12 można zamiast pętli **WHILE** użyć pętli **DO...WHILE**.

PRZYKŁAD 48.14

```
<?php
$i=1;                    //wartość początkowa dla pętli
do
{
    if($i%2==0)
    echo $i."\t";
    $i++;   //zwiększenie i o jeden
} while($i<=50);      //warunek zakończenia pętli
?>
```

Pętla FOREACH

Umożliwia ona dostęp do elementów tablicy lub właściwości obiektu.

Postać pętli **FOREACH**:

```
foreach ($tablica as $wartosc)
{
instrukcje;
}
lub
foreach ($tablica as $klucz => $wartosc)
{
instrukcje;
}
```

Pierwszą składnię stosuje się w działaniach na zwykłych tablicach.

PRZYKŁAD 48.15

```
<?php
$s=0;
$tab=[7,45,23,78,90,12,34,7,-4,3];
foreach ($tab as $w)
{
echo $w."<br>";
$s+=$w;
}
echo "Suma wartości w tablicy wynosi $s";
?>
```

Skrypt wyświetla wartości wpisane do tablicy, oblicza i wyświetla ich sumę.

Drugiej składni używa się na potrzeby działania na tablicach asocjacyjnych, a zatem indeksy tablicy są w postaci klucza.

PRZYKŁAD 48.16

```
<?php
$tab = array( 'Anna' => 'Kowalska', 'Joanna' => 'Nowak',
'Jerzy' => 'Grzegorzewski', 'Piotr'=>'Nowak' );
foreach ($tab as $klucz => $wartosc)
echo $klucz." ". $wartosc."<br>";
?>
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Podaj postać instrukcji warunkowej.
2. Do czego służy instrukcja **SWITCH**?
3. Omów różnice w działaniu pętli **WHILE** oraz pętli **DO...WHILE**.
4. Omów postać pętli **FOREACH** dla tablic indeksowanych i asocjacyjnych.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz skrypt, który sprawdzi, czy zdefiniowana zmienna **$liczba** jest podzielna przez 9. W oknie przeglądarki powinien się wyświetlić odpowiedni komunikat.
2. Utwórz skrypt, który sprawdzi, czy wartości zmiennych **$x**, **$y**, **$z** tworzą trójkąt prostokątny. Jeśli tak, to oblicz jego obwód i pole. W przeciwnym razie wyświetl komunikat: „To nie jest trójkąt pitagorejski”.
3. Utwórz skrypt, który wylosuje dziesięć liczb z zakresu <1,20>. Oblicz ich średnią arytmetyczną. Wynik i wylosowane liczby wyświetl w oknie przeglądarki. Do rozwiązania zadania zastosuj pętlę **FOR**.
4. Utwórz skrypt, który za pomocą pętli **WHILE** spośród liczb od 1 do 100 wyszuka liczby nieparzyste i jednocześnie podzielne przez 3. Wyświetl te liczby w oknie przeglądarki.

49 Funkcje i ich zastosowanie

ZAGADNIENIA

- Definicja funkcji własnej
- Funkcja zwracająca wartość
- Zasięg zmiennych w skrypcie
- Przekazywanie argumentów do funkcji
- Argumenty domyślne funkcji
- Funkcje tablic
- Funkcje daty i czasu
- Funkcje formatowania na ciągach znaków

Definiowanie funkcji

Funkcja to fragment kodu, który może być wielokrotnie wykorzystywany w różnych miejscach programu. Funkcja jest wywoływana przez nazwę i listę argumentów.

W języku PHP można definiować własne funkcje. Definicję funkcji rozpoczyna słowo kluczowe **`function`**, następnie należy podać nazwę funkcji i w nawiasach okrągłych wpisać listę argumentów rozdzielonych przecinkami. W nawiasach klamrowych umieszcza się treść funkcji.

Definicja funkcji ma postać:

```
function nazwa(arg1, arg2, …, argN)
{
instrukcje;
}
```

Nazwy funkcji muszą być tworzone zgodnie z zasadami tworzenia nazw zmiennych.

UWAGA

Nazwa funkcji nie może zaczynać się od $.

PRZYKŁAD 49.1

Przykład definicji funkcji **wyrazenie**, która ma trzy argumenty x, y, z, oblicza wartość wyrażenia $2*x-,y+\frac{z}{2}$, a następnie wypisuje wynik w oknie przeglądarki.

```
<?php
function wyrazenie($x, $y, $z)
{
echo "Wartość wyrażenia dla liczb $x, $y, $z wynosi
".(2*$x-0.5*$y+$z/2);
}
?>
```

Aby wywołać funkcję, należy napisać nazwę funkcji, a w nawiasach okrągłych podać odpowiednią liczbę argumentów, np. `wyrazenie (1,2,4);`

Funkcja **wyrazenie** ma w swoim ciele instrukcję wyświetlania obliczonej wartości.

Funkcja zwracająca wartość

Aby funkcja zwracała wartość, należy na końcu jej definicji umieścić słowo kluczowe **return**, a po nim – określić zmienną lub wartość. Klauzula **return** zatrzymuje działanie funkcji i zwraca wskazaną wartość do bloku kodu, z którego funkcję wywołano.

Definicja funkcji zwracającej wartość ma postać:

```
function nazwa_funkcji(arg1, arg2, …, argN)
{
instrukcje;
return wartość;
}
```

PRZYKŁAD 49.2

Funkcję z przykładu 49.1 można przekształcić w funkcję zwracającą wartości przez dodanie w ostatniej linijce słowa kluczowego **return**, a po nim – nazwę przekazywanej zmiennej.

```
<?php
function wyrazenie($x, $y, $z)
{
$wynik=2*$x-0.5*$y+$z/2;
echo "Wartość wyrażenia dla liczb $x, $y, $z wynosi ".$wynik;
return $wynik;
}
?>
```

Zasięg zmiennych w skrypcie

Gdy korzysta się z funkcji, należy pamiętać o zasięgu zmiennych w skrypcie. Zasięg zmiennej to miejsce w skrypcie, w którym jest ona widoczna.

Zmienne w PHP mogą być:

- lokalne,
- globalne,
- superglobalne.

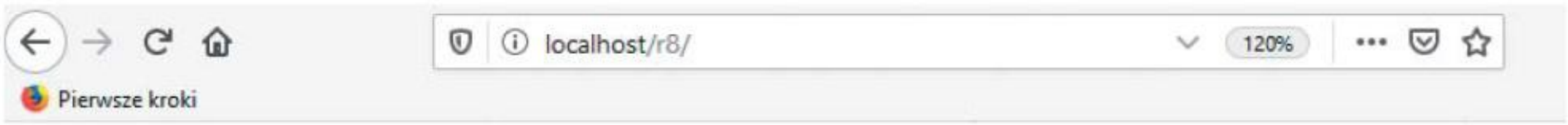
Zmienne lokalne mają zasięg lokalny i są definiowane wewnątrz funkcji. Ich zasięg ogranicza się do funkcji (poza nią są niewidoczne).

Zmienne globalne są dostępne w całym skrypcie oprócz wnętrza funkcji.

PRZYKŁAD 49.3

```
<?php
    $x=7;
function pole()
    {
    $pole=$x*$x;
    echo "Pole kwadratu o boku $x wynosi ".$pole;
    }
pole();
?>
```

Uruchomienie skryptu z przykładu 49.3 poskutkuje wyświetleniem komunikatu:



Notice: Undefined variable: x in **C:\xampp\htdocs\R8\index.php** on line **13**

Notice: Undefined variable: x in **C:\xampp\htdocs\R8\index.php** on line **13**

Notice: Undefined variable: x in **C:\xampp\htdocs\R8\index.php** on line **14**
Pole kwadratu o boku wynosi 0

Rys. 49.1. Efekt użycia zmiennej globalnej wewnątrz funkcji `pole()`

Zmienna `$x` została zdefiniowana na początku skryptu i jest zmienną globalną. Funkcja `pole()` nie ma do niej dostępu, zatem instrukcja `echo` nie zadziała poprawnie. Dodatkowo pojawił się komunikat o niezdefiniowanej zmiennej `x`.

Dostęp do zmiennych globalnych wewnątrz funkcji może odbywać się na dwa sposoby:

- za pomocą słowa kluczowego `global`,
- za pomocą zmiennej superglobalnej `$GLOBALS`.

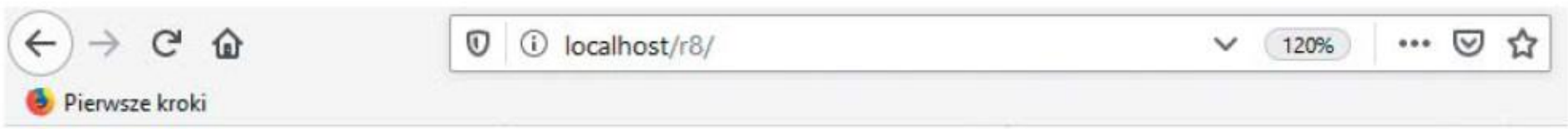
W pierwszym przypadku deklaracja ma postać:

```
global nazwa_zmiennej;
```

PRZYKŁAD 49.4

```
<?php
    $x=7;
function pole()
    {
    global $x;    //x jest zmienną globalną
    $pole=$x*$x;
    echo "Pole kwadratu o boku $x wynosi ".$pole;
    }
pole();
?>
```

Efektem działania skryptu będzie prawidłowe wyświetlenie wartości zmiennej **$x** oraz uruchomienie funkcji **pole()**. W oknie przeglądarki pojawi się następujący komunikat:



Rys. 49.2. Efekt zastosowania słowa kluczowego **global**

Drugi sposób dostępu do zmiennych globalnych z wnętrza funkcji polega na wykorzystaniu zmiennej superglobalnej **$GLOBALS**. Zmienne tego rodzaju są dostępne zawsze w każdej części skryptu.

$GLOBALS jest tablicą, za pomocą której można uzyskać dostęp do każdej zmiennej globalnej istniejącej w danym skrypcie. W tym celu należy użyć następującej konstrukcji:

```
$GLOBALS[,nazwa_zmiennej']
```

PRZYKŁAD 49.5

```
<?php
    $x=7;
function pole()
{
    $x=$GLOBALS['x'];
    $pole=$x*$x;
    echo "Pole kwadratu o boku $x wynosi ".$pole;
}
```

```
pole();
?>
```

Efekt działania skryptu będzie identyczny jak w przykładzie 49.4.

Przekazywanie argumentów do funkcji

Argumenty do funkcji mogą być przekazywane na dwa sposoby:

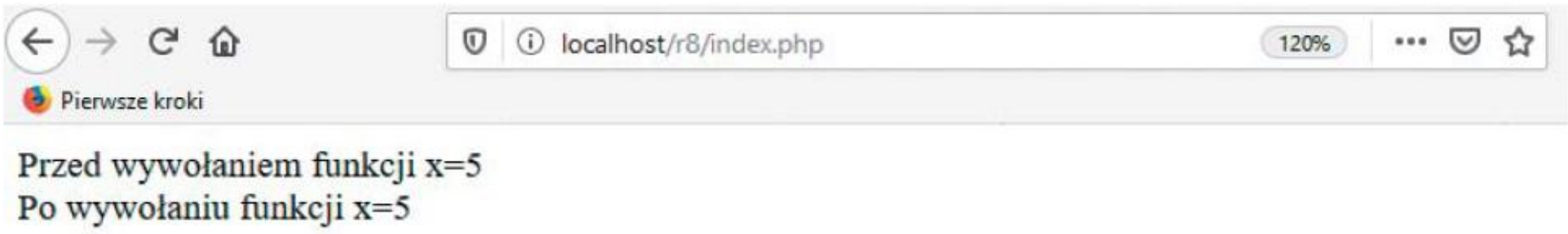
- przez wartość,
- przez referencję.

Domyślne przekazywanie argumentów do funkcji odbywa się przez wartość. Polega to na tym, że do funkcji przekazuje się kopie argumentów źródłowych i wszystkie działania funkcji wykonuje się na kopiach.

PRZYKŁAD 49.6

```
<?php
function zwieksz_liczbe($x)
    {
       $x+=6;          //zwiększenie liczby o 6
    }
$x=5;
echo "Przed wywołaniem funkcji x=".$x;
zwieksz_liczbe($x);
echo "<br>Po wywołaniu funkcji x=".$x;
?>
```

Funkcja **zwieksz_liczbe** ma jeden argument, a jej zadaniem jest zwiększenie podanej liczby o 6. Przed wywołaniem funkcji wartość zmiennej **$x** wynosi 5. Po wyświetleniu wartości tej zmiennej zostanie wywołana funkcja zwieksz_liczbe. Ostatni linijka skryptu spowoduje ponowne wyświetlenie wartości zmiennej **$x**. Wartość zmiennej **$x** się nie zmieniła, ponieważ wszystkie działania funkcji odbywały się na kopii tej zmiennej.



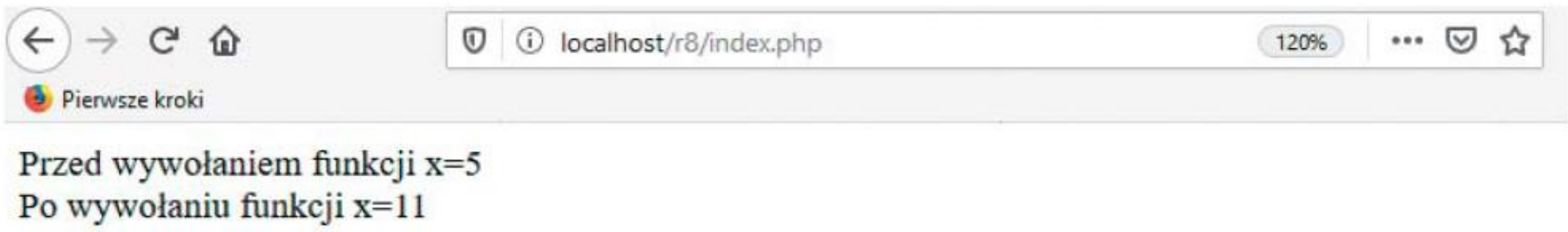
Rys. 49.3. Wynik działania funkcji z argumentem przekazywanym przez wartość

Drugim sposobem przekazywania argumentów do funkcji jest przekazywanie przez referencję (tzw. przezwisko). Funkcja działa wtedy na oryginalnych argumentach. W razie wykorzystania tego sposobu przekazywania argumentów przed argumentem należy umieścić znak **&** (ampersand). Ten sposób przedstawiono w przykładzie 49.7.

PRZYKŁAD 49.7

```
<?php
function zwieksz_liczbe(&$x)
    {
       $x+=6;          //zwiększenie liczby o 6
    }
$x=5;
echo "Przed wywołaniem funkcji x=".$x;
zwieksz_liczbe($x);
echo "<br>Po wywołaniu funkcji x=".$x;
?>
```

Efektem działania skryptu będzie inna wartość zmiennej **$x** przed wywołaniem funkcji **zwieksz_liczbe** i bezpośrednio po jej wywołaniu.



Rys. 49.4. Wynik działania funkcji z argumentem przekazywanym przez referencję

Argumenty domyślne funkcji

Tak jak w języku C++, również w PHP istnieje możliwość tworzenia funkcji z argumentami domniemanymi. Pozwala to na wywołanie funkcji bez podawania jej argumentów. Podczas wywołania funkcji będą wstawione wartości podane podczas definiowania funkcji.

Postać definicji funkcji z argumentami domniemanymi:

```
function nazwa($arg1=wart1, $arg2=wart2, ..., $argN=wartN)
{
instrukcje;
}
```

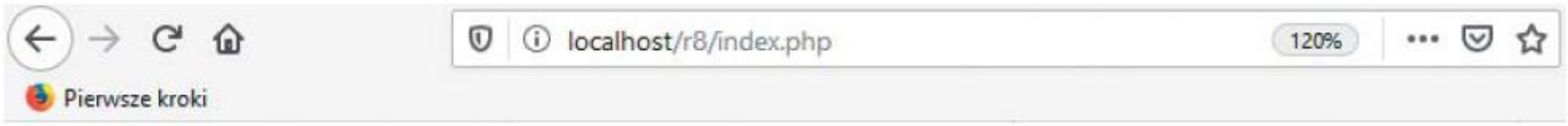
Jeśli podczas wywołania funkcji zostaną podane inne argumenty, to argumenty domniemane zostaną zaniechane.

PRZYKŁAD 49.8

```
<?php
function pole_trojkata($x=6, $y=4)
    {
       return ($x*$y)/2;
    }
$pole1=pole_trojkata();
```

```
echo "Funkcja z argumentami domyślnymi daje wartość: ".$pole1;
$pole2=pole_trojkata(7,8);
echo "<br>Funkcja bez argumentów domyślnych daje wartość:
".$pole2;
?>
```

Zmienna **$pole1** przechowuje wynik działania funkcji **pole_trojkata** wywołanej z argumentami domniemanymi, natomiast zmienna **$pole2** to wynik działania funkcji **pole_trojkata** z wartościami, które nie są wartościami domyślnymi.



Rys. 49.5. Wynik działania funkcji z argumentem domniemanymi

Oczywiście, dany skrypt może zawierać wiele zdefiniowanych funkcji. Należy jednak pamiętać, że jeśli definiuje się wiele funkcji, to warto je definiować w oddzielnym pliku. Pisanie całego kodu w jednym pliku może sprawić, że zmniejszy się jego czytelność i przejrzystość. Poza tym tworzenie funkcji w jednym konkretnym pliku oznacza, że niemożliwe staje się wielokrotne użycie napisanych w nim funkcji w innych plikach. Dobrym zwyczajem jest utworzenie oddzielnego pliku, w którym umieszcza się definicje wszystkich potrzebnych funkcji, a następnie za pomocą instrukcji **include()** lub **require()** dołącza ten plik w żądane miejsce innego pliku.

PRZYKŁAD 49.9

Utwórzmy plik o nazwie **funkcje.php**, w którym umieścimy funkcje obliczające pola poszczególnych figur geometrycznych. Treść pliku ma postać:

```
<?php
function trojkat($a,$h)
    {
       echo "Pole trójkąta o podstawie $a i wysokości $h wynosi:
       ".(($a*$h)/2);
    }
function prostokat($a,$b)
    {
       echo "Pole prostokąta o bokach $a i $b wynosi: ".($a*$b);
    }
function trapez($a,$b,$h)
    {
```

```
        $p=(($a+$b)*$h)/2;
        echo "Pole trapezu o podstawach $a oraz $b i wysokości
        $h wynosi: ".$p;
    }
function kolo($r)
    {
        $p=3.14*$r*$r;
        echo "Pole koła o promieniu $r wynosi: ".$p;
    }
?>
```

Aby skorzystać ze zdefiniowanych funkcji w innym pliku, należy dołączyć go za pomocą instrukcji **include()** lub **require()**. Od tej pory można wywoływać funkcje w innym pliku.

Wszystkie zdefiniowane funkcje zastosujemy w pliku **index.php**, w którym umieścimy tabelę z nazwami figur, ich rysunkami i przykładowymi polami.

```
<?php
//dołączenie pliku z definicjami funkcji
include("funkcje.php");
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Pętle</title>
 <style>
table,td,th
    {
        margin: auto;
        border: 2px blue double;
        border-collapse: collapse;
    }
th, .figura
    {
        background: #003366;
        height: 50px;
        color: white;
    }

</style>
```

```
</head>
<body>
<table>
<tr>
<th>Nazwa figury</th>
<th>Rysunek</th>
<th>Przykładowe pole</th>
</tr>
<tr>
<td class="figura">TRÓJKĄT</td>
<td><img src="trojkat.png"></td>
<td>
<?php trojkat(4,7);?>
</td>
</tr>
<tr>
<td class="figura">PROSTOKĄT</td>
<td><img src="prost.png"></td>
<td>
<?php prostokat(6,9);?>
</td>
</tr>
<tr>
<td class="figura">TRAPEZ</td>
<td><img src="trapez.png"></td>
<td>
<?php trapez(6,9,4);?>
</td>
</tr>
<tr>
<td class="figura">KOŁO</td>
<td><img src="circle.png"></td>
<td>
<?php kolo(2);?>
</td>
</tr>
</table>
</body>
</html>
```

Efektem będzie aplikacja w następującej postaci:

Nazwa figury	Rysunek	Przykładowe pole
TRÓJKĄT		Pole trójkąta po podstawie 4 i wysokości 7 wynosi: 14
PROSTOKĄT		Pole prostokąta o bokach 6 i 9 wynosi: 54
TRAPEZ		Pole trapezu o podstawach 6 oraz 9 i wysokości 4 wynosi: 30
KOŁO		Pole koła o promieniu 2 wynosi: 12.56

Rys. 49.6. Efekt działania skryptu z przykładu 49.9

W języku PHP istniej wiele funkcji wbudowanych, które są przydatne podczas tworzenia aplikacji internetowych. Są one podzielone na grupy tematyczne, takie jak:

- funkcje tablic,
- funkcje daty i czasu,
- funkcje formatowania na ciągach znaków.

Funkcje tablic

Są to funkcje wykonujące operacje na tablicach. Do zliczania elementów tablicy służą:

- `COUNT($zmienna_tablicowa)`,
- `SIZEOF()`.

PRZYKŁAD 49.10

```
<?php
$tab=array("Alicja","Jolanta","Katarzyna","Małgorzata",
"Agnieszka");
$dl=count($tab);      //lub sizeof($tab)
echo "W tablicy znajduje się $dl elementów i są nimi:<br>";
for($i=0;$i<$dl;$i++)
    {
    echo "tab[$i]=$tab[$i]<br>";
    }
?>
```

Skrypt z przykładu 49.10 oblicza liczbę elementów tablicy, a następnie za pomocą pętli **FOR** wypisuje zainicjowane elementy tablicy w oknie przeglądarki. Wynik jest następujący:

Rys. 49.7. Zastosowanie funkcji **COUNT**

Sortowanie tablicy polega na uporządkowaniu elementów tablicy w porządku rosnącym lub malejącym. Funkcje służące do sortowania tablicy:

- **SORT()** – sortowanie tablicy indeksowanej w kolejności od najmniejszej do największej;
- **RSORT()** – sortowanie tablicy indeksowanej w kolejności od największej do najmniejszej;
- **ASORT()** – sortowanie tablicy asocjacyjnej według zawartości, w kolejności od najmniejszej do największej;
- **ARSORT()** – sortowanie tablicy asocjacyjnej według zawartości, w kolejności od największej do najmniejszej;
- **KSORT()** – sortowanie tablicy asocjacyjnej według klucza, w kolejności od najmniejszej do największej;
- **KRSORT()** – sortowanie tablicy asocjacyjnej według klucza, w kolejności od najmniejszej do największej.

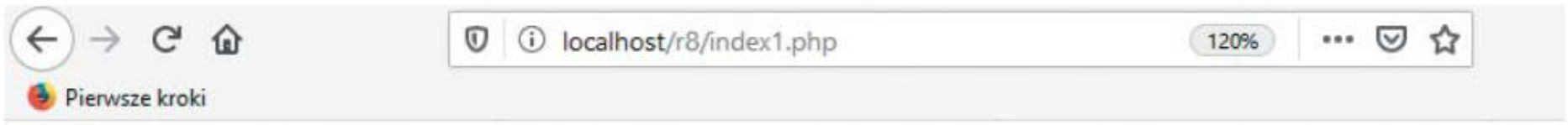
PRZYKŁAD 49.11

Weźmy pod uwagę tablicę indeksowaną. Aby posortować jej elementy rosnąco, należy zastosować funkcję **SORT()**. Wypisanie elementów przed sortowaniem i po nim wykonuje pętla **FOREACH**, która jest prostsza w konstrukcji niż zwykła pętla **FOR**.

```
<?php
$tab=array("Alicja","Paulina","Katarzyna",
"Małgorzata","Agnieszka","Jolanta","Celina");
$dl=count($tab);
echo "<b>Tablica przed posortowaniem:</b><br>";
foreach ($tab as $x)
    {
    echo "$x ";
    }
sort($tab);
```

```
echo "<br><b>Tablica po posortowaniu:</b><br>";
foreach ($tab as $x)
{
echo "$x ";
}
?>
```

Efekt działania skryptu jest następujący:



Tablica przed posortowaniem:
Alicja Paulina Katarzyna Małgorzata Agnieszka Jolanta Celina
Tablica po posortowaniu:
Agnieszka Alicja Celina Jolanta Katarzyna Małgorzata Paulina

Rys. 49.8. Wynik działania funkcji **SORT()**

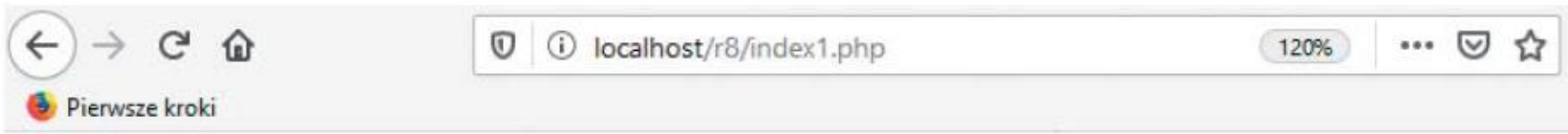
PRZYKŁAD 49.12

Tym razem zdefiniujemy tablicę asocjacyjną, w której kluczem będą imiona, a wartością – nazwiska. Posortujemy tę tablicę według wartości (nazwisk) rosnąco, a potem – według klucza (imion) rosnąco.

```
<?php
$tab=array("Alicja"=>"Kowalska","Paulina"=>"Nowak",
"Katarzyna"=>"Malinowska",
"Małgorzata"=>"Nowak","Agnieszka"=>"Czekalska","Jolanta"=>
"Kowalska","Celina"=>"Brzęczyszczykiewicz");
echo "<b>Tablica przed sortowaniem:</b><br>";
foreach ($tab as $klucz=>$x)
    {
    echo "$x $klucz<br> ";
    }
asort($tab);
echo "<b>Tablica po sortowaniu według zawartości:</b><br>";
foreach ($tab as $klucz=>$x)
    {
    echo "$x $klucz<br> ";
    }
ksort($tab);
echo "<b>Tablica po sortowaniu według klucza:</b><br>";
foreach ($tab as $klucz=>$x)
```

```
{
echo "$x $klucz<br> ";
}
```

Efekt działania skryptu jest następujący:



Tablica przed sortowaniem:
Kowalska Alicja
Nowak Paulina
Malinowska Katarzyna
Nowak Małgorzata
Czekalska Agnieszka
Kowalska Jolanta
Brzęczyszczykiewicz Celina
Tablica po sortowaniu według zawartości:
Brzęczyszczykiewicz Celina
Czekalska Agnieszka
Kowalska Alicja
Kowalska Jolanta
Malinowska Katarzyna
Nowak Paulina
Nowak Małgorzata
Tablica po sortowaniu według klucza:
Czekalska Agnieszka
Kowalska Alicja
Brzęczyszczykiewicz Celina
Kowalska Jolanta
Malinowska Katarzyna
Nowak Małgorzata
Nowak Paulina

Rys. 49.9. Wynik sortowania tablicy asocjacyjnej

Funkcje daty i czasu

Do funkcji wykonujących operacje na dacie i czasu należą:

- **`TIME()`**,
- **`GETDATE()`**,
- **`DATE()`**,
- **`MKTIME()`**.

Funkcja **`TIME()`** zwraca aktualny czas, podawany jako liczba sekund, które upłynęły od 1 stycznia 1970 00:00:00. Jest to tzw. **`znacznik_czasu`** (*timestamp*).

Funkcja **`GETDATE()`** zwraca tablicę asocjacyjną zawierającą informacje o podanej dacie.

Postać funkcji:

```
getdate([znacznik_czasu])
```

Argument **znacznik_czasu** jest opcjonalny. Oznacza to, że w razie niepodania tego argumentu działanie funkcji będzie odnosić się do daty bieżącej. Wynikiem funkcji jest tablica asocjacyjna, która zawiera dane dotyczące daty i czasu.

Elementy, które zawiera zwrócona tablica, przedstawiono w tabeli 49.1.

Tabela 49.1. Klucze zwróconej tablicy asocjacyjnej

Klucz	Opis	Zakres zwracanych wartości
seconds	liczba sekund	od 0 do 59
minutes	liczba minut	od 0 do 59
hours	liczba godzin	od 0 do 23
mday	dzień miesiąca	od 1 do 31
wday	dzień tygodnia w postaci cyfry	od 0 (niedziela) do 6 (sobota)
mon	miesiąc w postaci liczby	od 1 do 12
year	rok w postaci liczby czterocyfrowej	np. 2005
yday	numer danego dnia roku	0 do 365
weekday	nazwa dnia tygodnia w języku angielskim	np. sunday
month	nazwa miesiąca w języku angielskim	np. december
0	aktualny znacznik czasu	zależnie od systemu, zazwyczaj od –2147483648 do 2147483647

PRZYKŁAD 49.13

```
<?php
$data=getdate();
$dzien=$data["mday"]; //pobranie dnia miesiąca
$miesiac=$data["mon"]; //pobranie miesiąca
$rok=$data["year"];   //pobranie roku
if ($dzien<10) $dzien="0".$dzien;
if ($miesiac<10) $miesiac="0".$miesiac;
echo "Dzisiaj jest $dzien-$miesiac-$rok";
?>
```

Do zmiennej **$data** przypisano tablicę asocjacyjną, w której są przechowywane atrybuty daty i czasu. Zmienne **$dzien**, **$miesiac** oraz **$rok** przechowują kolejno dzień, miesiąc oraz bieżący rok. Jeśli numer dnia bądź miesiąca jest wartością mniejszą od 10, to do daty dopisujemy 0. Skrypt w oknie przeglądarki wyświetli datę w postaci 00-00-0000.

Funkcja **date()** pozwala z kolei na sformatowanie daty i czasu.

Postać funkcji jest następująca:

```
date (format[,znacznik_czasu])
```

Parametr **FORMAT** określa rodzaj informacji na temat daty i czasu.

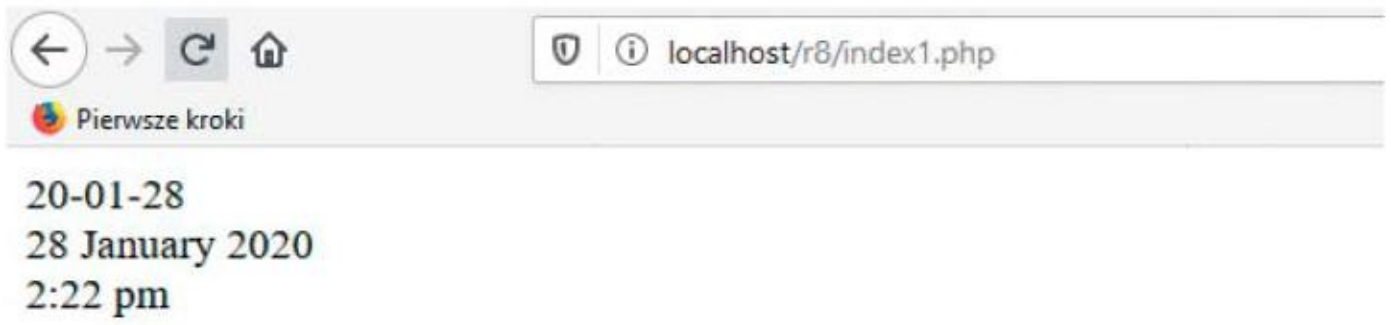
Tabela 49.2. Przykładowe znaczniki formatujące funkcję date()

Znacznik	Znaczenie	Zwracane wartości
	Dzień	
`d`	dzień miesiąca	01 do 31
`D`	dzień miesiąca w języku angielskim (trzy litery)	np. *Mon*
`l`	nazwa dnia tygodnia w języku angielskim	np. *Sunday*
`w`	liczbowa forma dnia tygodnia	od 0 (niedziela) do *6* (soboty)
`z`	dzień roku (od 0)	0 aż do 365
	Miesiąc	
F	nazwa miesiąca w języku angielskim	np. *January*
m	liczbowa forma miesiąca	od 01 do 12
M	nazwa miesiąca w języku angielskim (trzy litery)	np. *Jan*
t	liczba dni w danym miesiącu	28 do 31
	Rok	
L	informacja o tym, czy rok jest przestępny	1, jeśli rok jest przestępny, 0 w przeciwnym razie
Y	pełna liczbowa forma roku, cztery cyfry	np. *2003*
y	dwie cyfry reprezentujące rok	np. *03*
	Czas	
a	przed południem lub po południu	*am* lub *pm*
A	przed południem lub po południu	*AM* lub *PM*
g	godzina, w formacie 12-godzinnym	od 1 do 12
G	godzina, w formacie 24-godzinnym	od 0 do 23
h	godzina, w formacie 12-godzinnym	od 01 do 12
H	godzina, w formacie 24-godzinnym	od 00 do 23
i	minuty z zerami wiodącymi	od 00 do 59
s	sekundy z zerami wiodącymi	od 00 do 59

PRZYKŁAD 49.14

```
<?php
 echo date("y-m-d")."<br>";
 echo date("d F Y")."<br>";
 echo date("g:i a");
?>
```

Efekt działania skryptu jest następujący:



Rys. 49.10. Efekt zastosowania funkcji **date()**

Funkcja **mktime()** zwraca znacznik czasu podanej daty. Jej argumentami są kolejno:

- godzina,
- minuta,
- sekunda,
- miesiąc,
- dzień miesiąca,
- rok.

PRZYKŁAD 49.15

```
<?php
$data=mktime(10,15,5,11,30,1998);
echo date("d-m-Y G:i",$data);
?>
```

Efektem jest wyświetlenie następującego komunikatu w oknie przeglądarki:

Rys. 49.11. Efekt zastosowania funkcji **mktime()**

Funkcje formatowania ciągów znaków

W języku PHP istnieje wiele funkcji działających na ciągach znaków.

Funkcja **`nl2br()`** dla wybranego bloku tekstu przed każdym znakiem końca linii automatycznie wstawi znak **
** i zwróci przetworzony tekst.

Funkcja **`wordwrap()`** służy do formatowania tekstu w postaci kolumny o podanej szerokości.

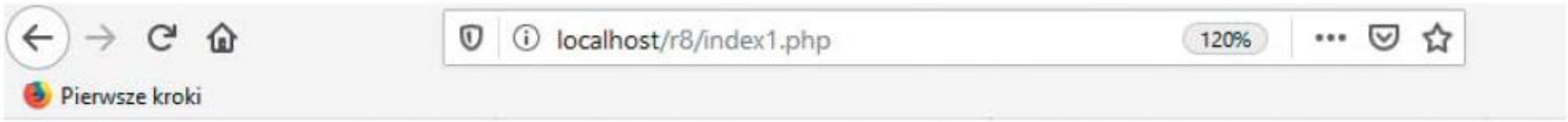
Funkcje służące do zmiany wielkości liter:

- **`strtoupper()`** – zamiana wszystkich liter na wielkie litery w podanym tekście;
- **`strtolower()`** – zamiana wszystkich liter na małe litery w podanym tekście;
- **`ucfirst()`** – pierwsza litera w podanym tekście zostaje zamieniona na wielką literę;
- **`ucwords()`** – wszystkie pierwsze litery wyrazów zostaną zamienione na duże.

PRZYKŁAD 49.16

```
<?php
$tekst="Alicja w krainie czarów i Alicja po drugiej stronie
lustra.";
echo "<u>Oryginalny tekst:</u><br>".$tekst."<br>";
echo "<u>Tekst po zastosowaniu strtoupper: </u><br>".
strtoupper($tekst)."<br>";
echo "<u>Tekst po zastosowaniu strtolower: </u><br>".
strtolower($tekst)."<br>";
echo "<u>Tekst po zastosowaniu ucfirst: </u><br>".
ucfirst($tekst)."<br>";
 echo "<u>Tekst po zastosowaniu ucwords: </u><br>".
ucwords($tekst)."<br>";
?>
```

W oknie przeglądarki zostanie wyświetlony następujący komunikat:



Oryginalny tekst:
Alicja w krainie czarów i Alicja po drugiej stronie lustra.
Tekst po zastosowaniu strtoupper:
ALICJA W KRAINIE CZARóW I ALICJA PO DRUGIEJ STRONIE LUSTRA.
Tekst po zastosowaniu strtolower:
alicja w krainie czarów i alicja po drugiej stronie lustra.
Tekst po zastosowaniu ucfirst:
Alicja w krainie czarów i Alicja po drugiej stronie lustra.
Tekst po zastosowaniu ucwords:
Alicja W Krainie Czarów I Alicja Po Drugiej Stronie Lustra.

Rys. 49.12. Wynik zastosowania funkcji służących do zmiany wielkości liter

Funkcje służące do usuwania ciągu znaków:

- `trim()` – usuwa białe znaki z początku i końca podanego ciągu znaków;
- `ltrim()` – usuwa białe znaki z początku podanego ciągu znaków;
- `rtrim()` – usuwa białe znaki z końca podanego ciągu znaków.

Wszystkie funkcje mają postać:

```
nazwa_funkcji („ciąg znaków")
```

Funkcje te usuwają: znak spacji (kod `32`), znak tabulacji (kod `9`), znak nowej linii (kod `10`), znak powrotu karetki (kod `13`), znak tabulacji pionowej (kod `11`) oraz znak o kodzie `0`.

Aby usunąć inne znaki, należy użyć następującego zapisu:

```
nazwa_funkcji („ciąg znaków", „znaki do usunięcia")
```

Aby wyznaczyć długość łańcucha znaków, należy użyć funkcji `strlen()`.

PRZYKŁAD 49.17

```
<?php
$tekst="Alicja w krainie czarów i Alicja po drugiej stronie
lustra";
echo "Długość tekstu wynosi: ".strlen($tekst);
?>
```

Efektem skryptu jest wyświetlenie komunikatu: „Długość tekstu wynosi 60".

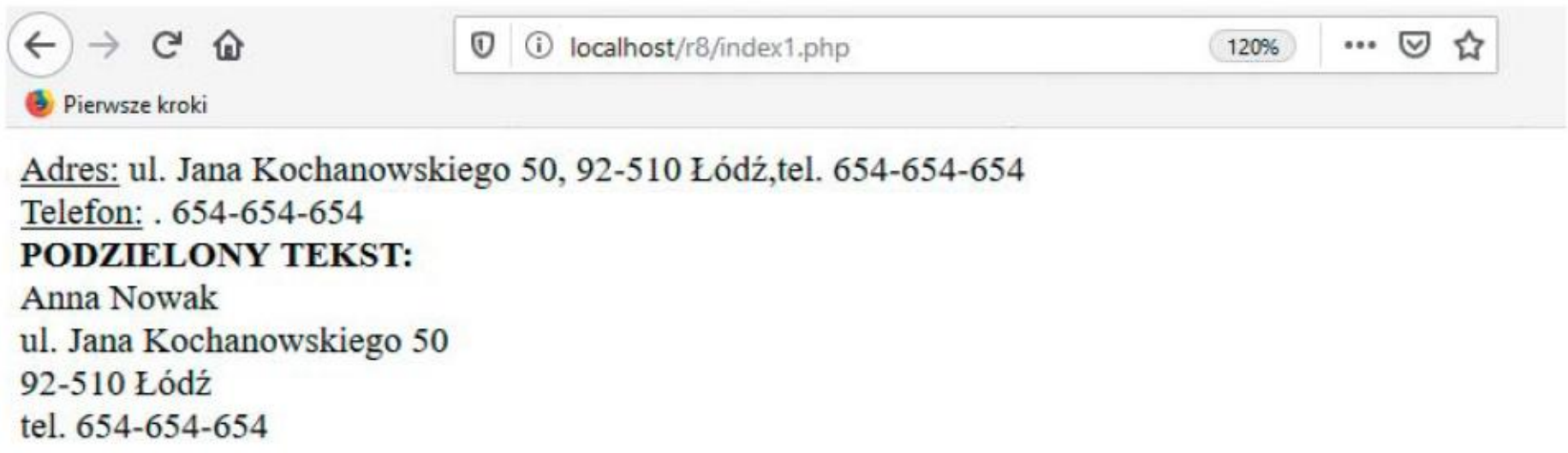
Funkcje służące do wyszukiwania podciągów w podanym łańcuchu znaków:

- `strstr()` – sprawdza, czy w podanym ciągu jest inny podciąg; jeśli podany podciąg nie został znaleziony, to funkcja zwraca wartość `FALSE`; w przeciwnym razie funkcja zwraca fragment ciągu źródłowego – od znalezionego podciągu do jego końca; funkcja rozróżnia wielkość liter;
- `strpos()` – sprawdza, czy szukany podciąg znajduje się w ciągu źródłowym; jeżeli tak, to zwraca jego położenie; ma trzy argumenty: pierwszy to ciąg źródłowy, drugi to poszukiwany podciąg, trzeci to pozycja w ciągu źródłowym, od którego rozpocznie się poszukiwanie;
- `substr()` – zwraca część ciągu źródłowego; ma trzy argumenty: pierwszy to ciąg źródłowy, drugi to indeks określający miejsce początku zwracanego podciągu, trzeci, opcjonalny, określa długość pobranego ciągu;
- `strtok()` – pozwala na podzielenie ciągu źródłowego na poszczególne podciągi; funkcja ma dwa argumenty: pierwszy to ciąg źródłowy, drugi to ciąg znaków rozdzielających podciągi.

PRZYKŁAD 49.18

```
<?php
$tekst="Anna Nowak, ul. Jana Kochanowskiego 50, 92-510 Łódź,
tel. 654-654-654";
echo "<u>Adres:</u> ".strstr($tekst, "ul.")."<br>";
echo "<u>Telefon:</u> ".substr($tekst,58)."<br>";
echo "<b>PODZIELONY TEKST:</b><br>";
$z=",";
$podciag=strtok($tekst,$z);
    while(is_string($podciag))
       {
          if($podciag)
             {
             echo $podciag. "<br>";
             }
          $podciag=strtok($z);
       }
?>
```

Efekt działania skryptu jest następujący:



Rys. 49.13. Efekt zastosowania funkcji znajdujących określony podciąg

Porównywanie ciągów

Porównywanie ciągów znaków można wykonać za pomocą operatorów porównania lub funkcji porównujących:

- `strcmp()`,
- `strcasecmp()`.

Funkcja strcmp() ma postać:

```
strcmp("tekst1","tekst2")
```

Ta funkcja zwraca wartość mniejszą od 0, jeżeli **tekst1** jest mniejszy od **tekst2**, a wartość większą od 0 – w przeciwnym razie. Funkcja zwraca wartość 0, gdy oba ciągi są sobie równe.

Funkcja **strcasecmp()** działa tak, jak funkcja **strcmp()**, ale nie uwzględnia wielkości liter.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Podaj postać definicji funkcji.
2. W jaki sposób przekazuje się argumenty do funkcji?
3. Co to są argumenty domyślne funkcji?
4. Co to są zmienne globalne i lokalne?
5. Do czego służą funkcje **COUT()** i **SIZEOF()**?
6. Wymień i omów funkcje służące do sortowania tablic.
7. Jaka jest różnica między funkcjami **rtrim()** a **ltrim()**?

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz skrypt, w którym zdefiniujesz funkcję o nazwie **spr_pierwsza()**. Funkcja ma sprawdzać, czy podana liczba jest liczbą pierwszą, i wyświetlać w oknie przeglądarki odpowiedni komunikat.
2. Utwórz skrypt, w którym zdefiniujesz funkcję **pole()**, która obliczy pole prostokąta. Funkcja ma mieć dwa argumenty i zwracać wartość obliczonego pola. Wywołaj funkcję z argumentami 4 i 6.
3. Utwórz skrypt, w którym zdefiniujesz dwie funkcje: **pitagoras()** oraz **pole()**. Pierwsza z nich ma sprawdzać, czy trzy podane wartości tworzą boki trójkąta prostokątnego. Jeśli tak, to ma zostać wywołana funkcja obliczająca pole tego trójkąta. W przeciwnym razie ma być wyświetlony komunikat „To nie są boki trójkąta prostokątnego”.
4. Utwórz skrypt, w którym zdefiniujesz 10-elementową tablicę i wpiszesz do niej wartości: 230, 5, –89, 3, 12, 123, 45, –67, 456, 12. Posortuj elementy tablicy rosnąco, a wynik sortowania wyświetl w oknie przeglądarki.
5. Utwórz skrypt, w którym zmiennej **$tekst** przypiszesz treść dowolnego przysłowia. Zastosuj odpowiednią funkcję i sprawdź, czy w podanym przysłowiu znajduje się słowo **,kto'**, oraz wyświetl odpowiedni komunikat.

50 Obsługa formularzy

ZAGADNIENIA

- Pobieranie danych z formularza
- Walidacja formularza

Pobieranie danych z formularza

Podstawowym sposobem przekazywania danych z przeglądarki internetowej do serwera są formularze HTML. Użytkownik za pomocą formularza wprowadza informacje, które są przesyłane do serwera za pomocą skryptów PHP.

Weźmy pod uwagę formularz (**`index.html`**). Składa się on z dwóch pól tekstowych do wprowadzania tekstu oraz jednego przycisku **`submit`** (rys. 50.1).

Rys. 50.1. Przykładowy formularz

Kod źródłowy tego formularza ma następującą postać (plik o nazwie **`index.php`**):

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Kasa biletowa</title>
</head>
<body>
<h1>Kasa biletowa online</h1>
<form action="oblicz.php" method="post">
```

```
    <p><b>Liczba biletów normalnych (27 zł/szt.): </b>
    <br><input type="text" name="adult"></p>
    <p><b>Liczba biletów ulgowych (18 zł/szt.): </b>
    <br><input type="text" name="child"></p>
    <p><input type="submit" value="Zamów">  </p>
</form>
</body>
</html>
```

Po kliknięciu przycisku **Zamów** dane z formularza powinny zostać przesłane na serwer za pomocą metody **POST**. Atrybut **ACTION** wskazuje, gdzie zostaną wysłane dane z formularza, który zostanie zatwierdzony. W tym przypadku zostanie uruchomiony skrypt **oblicz.php** i do niego zostaną przekazane dane z formularza. Atrybut **METHOD** określa metodę przekazywania informacji do wybranego skryptu. Atrybut może przyjmować jedną z dwóch wartości: **POST** lub **GET** (wartość domyślna).

Metodę **GET** stosuje się, gdy parametrów jest niewiele, ponieważ w tej metodzie parametry są przekazywane za pomocą adresu URL. Długość adresu jest ograniczona, a przekazywane parametry stają się widoczne w pasku adresu przeglądarki. Adres skryptu PHP od przesyłanych parametrów oddziela się znakiem zapytania.

Metoda **POST** do przekazywania danych wykorzystuje nagłówek strony. Umożliwia ona przekazywanie większej liczby parametrów; nie są one widoczne w oknie przeglądarki.

Przesyłane wartości pobrane z formularza są zapisywane do tablicy asocjacyjnej. Dane przesyłane metodą **GET** są zapisywane w tablicy **$_GET**, a w przypadku metody **POST** – do tablicy **$_POST**. Indeksy tych tablic to nazwy pól formularza, czyli wartości atrybutu name wstawione w formularzu HTML. Odwołanie do poszczególnych pól formularza ma następującą postać:

```
$nazwa_zmiennej=$_GET['nazwa_pola'];
```

lub

```
$nazwa_zmiennej=$_POST['nazwa_pola'];
```

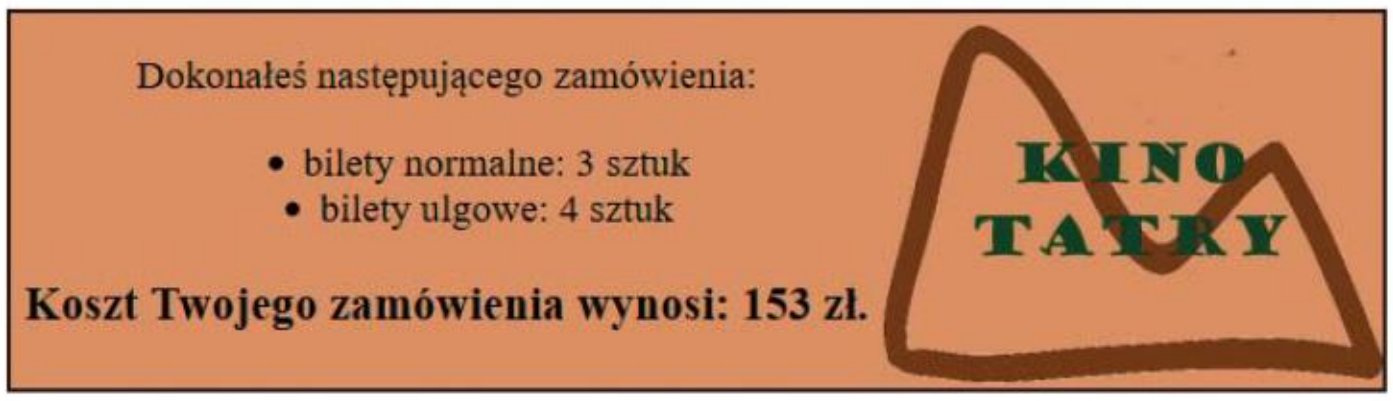
PRZYKŁAD 50.1

W pliku **index.html** dane wysłano metodą **POST** i przesłano do pliku **oblicz.php**. Należy zatem utworzyć ten plik, pobrać dane z formularza, wykonać na nich obliczenia i wyświetlić informacje o złożonym zamówieniu.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Kasa biletowa</title>
<style>
#kontener
```

```
{
   width:550px;
   height:150px;
   border:2px solid #330000;
   text-align: center;
   background-color: #CC9966;
}
#blok
{

   width:350px;

   float: left;
}
img{
   width:200px;
   height: 150px;
}
#obraz
{
   float:left;
}
</style>
</head>
<body>
   <div id="kontener">
<div id="blok">
<?php
echo "<p>Dokonałeś następującego zamówienia: </p>";
$norm=$_POST['adult']; //pobranie liczby biletów normalnych
$ulg=$_POST['child'];  //pobranie liczby biletów ulgowych
echo "<ul><li>bilety normalne: $norm sztuki</li>";
echo "<li>bilety ulgowe: $ulg sztuki</li></ul>";
$koszt=$norm*27+$ulg*18;
echo "<h3>Koszt Twojego zamówienia wynosi: $koszt zł.</h3>";
?>
</div>
<div id="obraz"><img src="logo.png"></div>
</div>
</body>
</html>
```

Po wpisaniu danych do formularza i zaakceptowaniu ich przyciskiem **`Zamów`** przeglądarka wyświetli przesłane dane (rys. 50.2).



Rys. 50.2. Dane przesłane za pomocą formularza

PRZYKŁAD 50.2

Innym przykładem zastosowania formularza, z którego można pobrać dane, jest formularz zgłoszeniowy, który – oprócz pól tekstowych – ma jeszcze listę rozwijaną z możliwością wybrania poziomu wykształcenia oraz pole do wprowadzania adresu e-mail (rys. 50.3).

Rys. 50.3. Przykład formularza zgłoszeniowego

Poniżej formularza wprowadzimy blok wyświetlający dane wpisane przez użytkownika. Formularz oraz skrypt pobierający dane umieścimy w pliku **`index.php`** o następującej zawartości:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Formularz zgłoszeniowy</title>
<style>
table
```

```
    {
      width:100%;;
      height:300px;
      background-image: url("tlo.jpg");
      margin: auto;
    }
input[type="submit"]
    {
      background: #777777;
      color:white;
      font-family: Verdana;
      height: 40px;
    }
#formularz
    {
      width:400px;
      height:300px;
      margin: auto;
    }
#wynik
    {
      width:400px;
      height:200px;
      margin: auto;
    border:2px groove #666666;
      text-align: center;
    }
    h3{
      color:blue;
    }
    h2{
      text-align: center;
    }
</style>
</head>
<body>
<div id="formularz">
<form action="" method="post">
<table>
<tr><td colspan="2"><h2>Formularz zgłoszeniowy</h2></td></tr>
<tr>
```

```
<td>Imię:</td>
<td><input type="text" name="imie"></td>
</tr>
<tr>
<td>Nazwisko:</td>
<td><input type="text" name="nazwisko"></td>
</tr>
<tr>
<td>Wykształcenie:</td>
<td><select name="wiedza">
    <option>wyższe</option>
    <option>średnie </option>
    <option>podstawowe</option>
    </select></td>
    </tr>
<tr>
<td>Adres e-mail:</td>
<td><input type="email" name="mail"></td>
</tr>
<tr>
<td colspan="2" style="text-align: right;"><input type="sub-
mit" value="Zapisz uczestnika"></td>
</tr>
</table>
</form>
    </div>
<div id="wynik">
<?php
$imie=$_POST['imie'];
$nazwisko=$_POST['nazwisko'];
$wiedza=$_POST['wiedza'];
$mail=$_POST['mail'];
echo "<p>Zapisaliśmy użytkownika o następujących danych:</p>";
echo "<h3>$imię $nazwisko</h3>";
echo "<h4>Wykształcenie: $wiedza</h4>";
echo "<h4>E-mail: $mail</h4>";
?>
</div>
</body>
</html>
```

Po wprowadzeniu danych do formularza otrzymamy komunikat o następującej treści:

Zapisaliśmy użytkownika o następujących danych:

Anna Nowak

Wykształcenie: wyższe

E-mail: a.nowak@gmail.com

Rys. 50.4. Efekt skryptu z przykładu 50.2

Walidacja formularza

Podczas przetwarzania formularza należy sprawdzić, czy uzupełniono jego wszystkie pola. W tym celu należy zastosować funkcję **isSet()**, która pozwala na sprawdzenie, czy określone pole formularza wypełniono, a zatem czy do skryptu przekazano wartość danego pola. Funkcja zwraca wartość **TRUE**, jeśli wartość przekazano, w przeciwnym razie zostanie zwrócona wartość **FALSE**.

PRZYKŁAD 50.3

```
if (isset($_POST[imie]))

    {
       echo "Masz na imię ".$_POST['imie'];
    }
```

Jeśli w formularzu zostanie uzupełnione pole tekstowe, to w oknie przeglądarki zostanie wyświetlony komunikat: „Masz na imię”, a dalej imię, które wpisano do formularza.

Funkcja **empty()** natomiast sprawdza, czy istniejąca zmienna, użyta jako argument, jest pusta. Jeśli zmienna jest pusta, to funkcja zwraca wartość **TRUE**, w przeciwnym przypadku zwraca wartość **FALSE**.

PRZYKŁAD 50.4

```
<?php
    if  (!empty($_POST['imie'])  &&  !empty($_POST['nazwisko'])
    && !empty($_POST['mail']) )
    {
       echo "Wpisałeś wszystkie dane";
    }
```

```
    else
    {
       echo "Nie wprowadzono wszystkich danych";
    }
?>
```

Skrypt sprawdza, czy do wszystkich pól formularza wprowadzono wartości. Jeśli tak, to zostaną wyświetlone podane informacje, w przeciwnym razie zostanie wyświetlony komunikat: „Nie wprowadziłeś wszystkich danych".

W przykładowym formularzu z przykładu 50.1 należy wpisać liczbę biletów normalnych i ulgowych. Istnieje możliwość, że użytkownik zamiast liczby wpisze tekst. Wtedy, jeśli będziemy wykonywać obliczenia na wprowadzonych danych, skrypt wykona błędne obliczenia lub wyświetli błąd. Należy zatem zastosować funkcję, która sprawdzi, czy podana wartość jest liczbą.

Do sprawdzenia, czy w podanym polu formularza jest określony typ danych, można zastosować jedną z funkcji:

- **`is_array()`** – zwraca **`TRUE`**, jeżeli zmienna jest typu **`ARRAY`**, lub **`FALSE`** w przeciwnym razie;
- **`is_bool()`** – zwraca **`TRUE`**, jeżeli zmienna jest typu **`BOOLEAN`**, lub **`FALSE`** w przeciwnym razie;
- **`is_double()`** – zwraca **`TRUE`**, jeżeli zmienna jest zmiennoprzecinkowa, lub **`FALSE`** w przeciwnym razie;
- **`is_float()`** – zwraca **`TRUE`**, jeżeli zmienna jest zmiennoprzecinkowa, lub **`FALSE`** w przeciwnym razie;
- **`is_int()`** – zwraca **`TRUE`**, jeżeli zmienna jest typu **`INTEGER`**, lub **`FALSE`** w przeciwnym razie;
- **`is_null()`** – zwraca **`TRUE`**, jeżeli zmienna jest typu **`NULL`**, lub **`FALSE`** w przeciwnym razie;
- **`is_numeric()`** – zwraca **`TRUE`**, jeżeli na początku zmiennej można znaleźć liczbę, lub **`FALSE`** w przeciwnym razie.

PRZYKŁAD 50.5

Skrypt w przykładzie 50.1 należy zmodyfikować: wstawić instrukcję najpierw sprawdzającą, czy dane do formularza zostały wpisane, a następnie – czy są one liczbami. Jeśli oba warunki są spełnione, zostanie obliczona wartość zamówienia.

```
<?php
//sprawdzenie, czy pola zostały uzupełnione
if (isset($_POST['adult']) && isset($_POST['child']))
{
//sprawdzenie, czy podane wartości są liczbami
```

```
if (is_numeric($_POST['adult']) &&
is_numeric($_POST['child']))
    {
echo "<p>Dokonałeś następującego zamówienia: </p>";
$norm=$_POST['adult']; //pobranie liczby biletów normalnych
$ulg=$_POST['child'];  //pobranie liczby biletów ulgowych
echo "<ul><li>bilety normalne: $norm sztuk</li>";
echo "<li>bilety ulgowe: $ulg sztuk</li></ul>";
$koszt=$norm*27+$ulg*18;
echo "<h3>Koszt Twojego zamówienia wynosi: $koszt zł.</h3>";
    }
}
?>
```

SPRAWDŹ SWOJĄ WIEDZĘ

1. Jaka jest różnica między metodą **GET** a metodą **POST**?
2. Przedstaw proces walidacji formularza za pomocą skryptu PHP.
3. Opisz funkcje sprawdzające, czy w podanym polu jest określony typ danych.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz aplikację internetową, która będzie zawierała formularz umożliwiający zakup zeszytów w linię (1,5 zł/szt.), zeszytów w kratkę (1,3 zł/szt.) oraz zeszytów gładkich (1 zł/szt.). Aplikacja ma zawierać skrypt działający po stronie serwera. W oknie przeglądarki wyświetl wartość dokonanego zakupu.
2. Utwórz aplikację internetową zawierającą formularz pobierający kwotę produktu w zł oraz pole typu radio, które wybierze funt brytyjski lub euro. Po naciśnięciu przycisku **Przelicz** skrypt wyświetli wartość produktu w wybranej walucie.

51 Obsługa bazy danych w PHP

ZAGADNIENIA

- Połączenie z bazą danych
- Wysyłanie zapytań do bazy i ich wyświetlanie
- Biblioteka PDO

Połączenie z bazą danych

Bazy danych są zbiorem danych dla aplikacji internetowych. Można z takich baz pobrać informacje, które są prezentowane użytkownikowi strony. Do obsługi baz danych wykorzystuje się systemy zarządzania bazami danych. Najpopularniejszym systemem współpracującym z PHP jest MySQL.

Aby współpraca języka PHP z systemem MySQL była możliwa, musi istnieć baza danych, z którą można się połączyć.

Proces komunikacji z MySQL podzielono na kilka etapów. Są nimi:

- nawiązanie połączenia z MySQL i wybór bazy danych;
- utworzenie zapytania i jego wykonanie;
- otrzymanie rezultatów i wyświetlenie ich na stronie;
- rozłączenie z MySQL.

Do połączenia z bazą danych służy funkcja **`mysqli_connect()`**. Ma ona cztery argumenty: nazwę hosta, nazwę użytkownika, hasło użytkownika i nazwę bazy danych, której będziemy używali. To polecenie jednocześnie wykonuje połączenie z serwerem bazy danych oraz wybiera bazę, z której będą pobierane dane.

PRZYKŁAD 51.1

```
$polaczenie=mysqli_connect('localhost', 'root','','komis');
```

Nazwa hosta oznacza nazwę lub adres IP serwera, na którym umieszczono bazę danych.

! UWAGA

Jeżeli użyjemy operatora @ przed funkcją, to w razie problemu z połączeniem nie zostanie wyświetlone ostrzeżenie wygenerowane przez PHP.

Do zakończenia połączenia z bazą danych służy funkcja mysqli_close() w postaci:

```
mysqli_close([identyfikator_połączenia])
```

Funkcja zwróci wartość **TRUE**, gdy operacja zamknięcia połączenia zakończy się powodzeniem.

PRZYKŁAD 51.2

```
<?php
$conn=@mysqli_connect('localhost','root','','firma');
if (!$conn)
    {
    exit("Błąd połączenia z serwerem");
    }
else
    {
    echo "Połączyłeś się z bazą FIRMA";
    }
mysqli_close($conn);
?>
```

Po wywołaniu funkcji **mysqli_connect()** sprawdza się, czy nawiązano połączenie. Jeśli tak, to wyświetli się komunikat: „Połączyłeś się z bazą FIRMA", w przeciwnym razie pojawi się komunikat: „Błąd połączenia z serwerem". Na końcu nastąpi zakończenie połączenia z bazą **FIRMA**.

Wysyłanie zapytań do bazy i ich wyświetlanie

Po nawiązaniu połączenia z wybraną bazą danych można wysyłać do niej zapytania. W tym celu stosuje się funkcję **mysqli_query()** w następującej postaci:

```
mysqli_query(identyfikator_połączenia, 'zapytanie')
```

Zwracane wartości zależą od rodzaju zapytania. Jeśli zapytanie pobierało dane, to funkcja zwraca identyfikator do tych danych. Jeśli zapytanie nie pobiera danych, to funkcja zwraca wartość **TRUE**, natomiast w przeciwnym razie zwraca wartość **FALSE**.

Dla zapytań pobierających dane jest zwracany identyfikator zasobów. Do odczytania tego typu danych można zastosować funkcję **mysqli_fetch_array()**.

PRZYKŁAD 51.3

```
<?php
$conn=mysqli_connect('localhost','root','','firma');
if (!$conn)
```

```
    {
    exit("Błąd połączenia z serwerem");
    }
else
    {
$pyt=mysqli_query($conn,'SELECT  nazwisko,imie,  pensja  FROM
pracownicy WHERE nazwisko like "A%" ORDER by nazwisko');
echo "<ol>";
    while ($result=mysqli_fetch_array($pyt))
    {
    echo   "<li>".$result['nazwisko']."   ".$result['imie']."
    ".$result['pensja']." zł</li>";
    }
echo "</ol>";
    }
mysqli_close($conn);
?>
```

Polecenie **mysqli_query()** wysyła zapytanie do bazy **FIRMA** – wypisz imię, nazwisko i pensję pracowników, których nazwisko rozpoczyna się literą A, a dane posortuj rosnąco według nazwisk. Zmienna **$result** jest tablicą asocjacyjną, która przechowuje dane pobrane z zapytania. Kolejnymi elementami tej tablicy są **$result['imie']**, **$result['nazwisko']** oraz **$result['pensja']**.

Aby odczytać całą zawartość tabeli, należy wywołać funkcję w pętli. Gdy wszystkie dane zostaną odczytane, funkcja zwróci wartość **FALSE**.

Efektem będzie wyświetlenie wykazu w następującej postaci:

Rys. 51.1. Efekt skryptu z przykładu 51.3

Kolejną przydatną funkcją pracującą na bazie danych jest funkcja **mysqli_num_rows()**. Zwraca ona liczbę wierszy znajdujących się w wyniku zapytania. Tej wartości można użyć do zbudowania pętli FOR odczytującej kolejne wiersze zwrócone przez zapytanie.

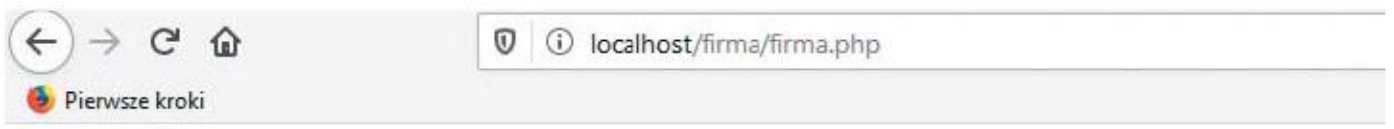
PRZYKŁAD 51.4

Skrypt pobierze z bazy imię, nazwisko oraz datę urodzenia kobiet pracujących na stanowisku informatyka. Zamiast pętli **WHILE** zastosujemy pętlę **FOR**. Z tego powodu najpierw trzeba użyć instrukcji **mysqli_num_rows()**, która zliczy liczbę rekordów wyświetlonych dzięki temu zapytaniu. Wynik zapytania zostanie wstawiony do tabeli.

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Baza danych o pracownikach</title>
<style>
table,td,th
    {
      border:2px solid grey;

    }
th
    {
      background-color: #dddddd;
    }
th, td
    {
      width:100px;
    }
</style>
</head>
<body>
<?php
$conn=mysqli_connect('localhost','root','','firma');
if (!$conn)
{
    exit("Błąd połączenia z serwerem");
}
    else
    {
      $pyt=mysqli_query($conn,'SELECT imie, nazwisko,
data_urodzenia from pracownicy where zawod="informatyk"
and plec="k"');
      $il_wierszy=mysqli_num_rows($pyt);
```

```
    echo "<table><tr><th>Imię</th><th>Nazwisko</th><th>Data
    urodzenia</th>";
    for($i=0;$i<$il_wierszy;$i++)
    {
       $result=mysqli_fetch_array($pyt);

       echo "<tr><td>".$result['nazwisko']."</td><td>"
       .$result['imie']."</td><td>
       ".$result['data_urodzenia']."</td></tr>";
    }
    echo "</table>";
 }
    mysqli_close($conn);
?>
</body>
</html>
```

Efektem skryptu jest tabela następującej postaci:



Imię	Nazwisko	Data urodzenia
Kowalska	Ewa	1981-12-01
Kiel	Barbara	1970-05-09
Paczka	Magdalena	1973-03-16
Kilarski	Barbara	1968-07-15

Rys. 51.2. Efekt skryptu z przykładu 51.4

Dla zapytań modyfikujących funkcja **mysqli_query()** zwraca wartość **TRUE** lub **FALSE**.

Funkcja **mysqli_affected_rows()** wyświetla liczbę zmodyfikowanych rekordów w tabeli. Funkcja ma postać:

```
mysqli_affected_rows([identyfikator])
```

UWAGA

Jeśli w wywołaniu funkcji zostanie pominięty identyfikator połączenia z serwerem, to działanie funkcji będzie dotyczyło ostatnio otwartego połączenia.

PRZYKŁAD 51.5

Do tabeli **placówki** w bazie FIRMA zostaną dodane dwa rekordy:

```
<?php
$conn=mysqli_connect('localhost','root','','firma');
if (!$conn)
{
    exit("Błąd połączenia z serwerem");
}
    else
    {
mysqli_query($conn,"INSERT into placowki(id_placowki,nazwa,
miasto,adres) VALUES (7,'Iks','Warszawa','Tuwima7'),
(8,'DamaX','Kielce','Mickiewicza 8')");
$ile_dodano=mysqli_affected_rows($conn);
echo "Liczba dodanych rekordów: ".$ile_dodano;
}
mysqli_close($conn);
?>
```

PRZYKŁAD 51.6

Utworzymy aplikację internetową, która wyświetli informacje o zaopatrzeniu pewnej hurtowni owoców.

Rys. 51.3. Wygląd witryny internetowej z przykładu 51.6

Strona wyświetla dane firm korzystających z hurtowni i zamówienia złożone przez firmę Nowak&Kowalski.
Kod źródłowy pliku zawierającego żądany skrypt ma następującą postać:

```
<!doctype html>
<html>
<head>
<title> Hurtownia owoców</title>
<meta charset="utf-8">
<link rel="stylesheet" href="style.css" type="text/css" />
</head>
<body>
<section id="kontener">
<section id="baner">
Hurtownia owoców
</section>
<section id="prawy">
<?php
$conn=mysqli_connect('localhost','root','','skupow');
$z1=mysqli_query($conn,'select * from firma');
echo "<p>FIRMY KORZYSTAJĄCE Z HURTOWNI</p>";
while ($row=mysqli_fetch_array($z1))
{
echo      "<b>".$row['Nazwa'].'</b>,      '.$row['Miasto'].',
'.$row['Adres'].'<br>';
}
$z2=mysqli_query($conn,'SELECT Nazwa, ilosc, Data zamowienia
from owoce, zamowienie where IdFirmy=1 and owoce.IdOwocu=za-
mowienie.IdOwocu order by Nazwa');
echo "<p>ZAMÓWIENIA FIRMY Nowak&Kowalski</p>";
echo '<table><tr><th>Nazwa owocu</th><th>Ilość w kg</th><th>Da-
ta zamówienia</th></tr>';
while ($row=mysqli_fetch_array($z2))
{
echo   "<tr><td>".$row['Nazwa'].'</td><td>'.$row['ilosc'].'</
td><td> '.$row['Datazamowienia'].'</td></tr>';
}
echo "</table>";
mysqli_close($conn);
?>
</section>
<section id="galeria">
```

```
<table >
<tr>
<td> <img src="czeresnia.png" class="obrazki"></td>
<td> <img src="gruszka.png" class="obrazki"></td>
<td> <img src="truskawka.png" class="obrazki"></td>
<td> <img src="sliwka.png" class="obrazki"></td>
<td> <img src="jagoda.png" class="obrazki"></td>
</tr>
</table>
</section>
</section>
</body>
</html>
```

Wygląd strony opisuje plik **style.css**:

```
#kontener
{
height:90%;
}
#baner{
height: 10%;
font-size:50px;
background-color:#CCFF33;
text-align: center;
letter-spacing: 15px;
}
#galeria{
background-color:#CCFF66;
height: 20%;
}
#prawy{
background-color:#FCFFBC;
height:70%;
text-align:center;
}

table{
width:100%;
margin:0 auto;
text-align: center;
}
```

```
p{
padding-top:20px;
margin-top:0px;
text-align:center;
font-weight: bold;
color:#990000;
}
.obrazki{
width:100px;
height:100px;
}
```

Biblioteka PDO

PDO (ang. *PHP Data Objects*) udostępnia interfejs do komunikacji z bazami danych w technice obiektowej. Zaletą PDO jest to, że w skryptach można korzystać z tych samych metod, niezależnie od bazy.

Aby połączyć się z bazą danych, należy za pomocą konstruktora stworzyć nowy obiekt klasy PDO:

```
PDO(źródło_danych, nazwa_użytkownika, hasło, opcje)
```

Źródło_danych to ciąg znaków opisujący rodzaj bazy danych i sposób połączenia oraz nazwę sterownika, adres serwera, nazwę bazy, numer portu. **Opcje** to dodatkowe informacje związane z połączeniem. Argumentem wymaganym jest **źródło_danych**, pozostałe argumenty mogą zostać pominięte.

Instrukcja uproszczona dla MySQL ma następującą postać:

```
mysql:host=nazwa_serwera;port=numer_portu;dbname=nazwa_bazy
```

Nazwa_serwera to nazwa lub adres serwera bazy danych (np. localhost).

Numer_portu to port, przez który nastąpi połączenie z bazą danych; domyślnie jest to wartość standardowa.

Nazwa_bazy to nazwa bazy, z którą się łączymy.

PRZYKŁAD 51.7

Połączenie z bazą FIRMA znajdująca się na serwerze MySQL pracującym na lokalnym komputerze.

```
<?php
$conn="mysql:host=localhost;dbname=firma";
try
    {
    $pol=new PDO($conn,"root","");
```

```
    echo "Udało się połączyć z serwerem";
    }
catch (PDOException $x)
    {
    echo 'Błąd połączenia '.$e->getMessage();
    exit;
    }
?>
```

Połączenie, które nawiązano, istnieje do momentu usunięcia obiektu z pamięci. Obiekt zostanie usunięty automatycznie po zakończeniu działania skryptu.

Aby wykonać zapytanie do bazy, należy zastosować metodę **query()**. Postać polecenia jest następująca:

```
$obiekt_PDO->query("Zapytanie");
```

Ta metoda zwraca obiekt klasy **PDOStatement**, zawierający wynik wykonania zapytania.

PRZYKŁAD 51.8

```
<?php
$conn="mysql:host=localhost;dbname=firma";
try
    {
    $pol=new PDO($conn,"root","");
    $query=$pol->query('SELECT imie, nazwisko, pensja FROM
    pracownicy WHERE zawod="grafik" ORDER BY pensja DESC');
      foreach ($query as $wiersz)
      {
      echo $wiersz['imie']." ".$wiersz['nazwisko']."
      ".$wiersz['pensja']."<br>";
      }
    }
catch (PDOException $x)
    {
    echo 'Błąd połączenia '.$e->getMessage();
    exit;
    }
?>
```

W przykładzie metoda **query()** zwraca wynik zapytania, a następnie za pomocą pętli **FOREACH** zostają wyświetlone kolejne wiersze tablicy asocjacyjnej **$rekord**.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień etapy komunikacji PHP z MySQL.
2. Do czego służy **`mysqli_connect()`**?
3. Opisz działanie funkcji **`mysqli_num_rows()`** oraz funkcji **`mysqli_affected_row()`**.
4. Podaj sposób tworzenia zapytań do bazy danych za pomocą PDO.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz aplikację internetową obsługującą bazę danych o pracownikach. Interfejs aplikacji wykonaj według wzoru przedstawionego na rys. 51.4.
   Dodatkowe wytyczne dotyczące wizualizacji witryny zawarto w opisach kodu HTML i arkusza CSS. Aby przygotować poprawnie działającą aplikację, trzeba utworzyć bazę danych **`FIRMA`** według listingu zamieszczonego w załączniku nr 2.

Rys. 51.4. Projekt witryny internetowej

Cechy CSS witryny internetowej:

Plik *firma.css* zawiera formatowanie dla:

- sekcji głównej: kolor tła #8BB381;
- banera: kolor tła #336600, biały kolor czcionki, rozmiar czcionki 120 %, wyrównanie tekstu do środka, wysokość banera 10 %;
- panelu lewego: kolor tła #AFC7C7, szerokość 40 %, wysokość 70 %, wyrównanie tekstu do środka;
- panelu prawego: kolor tła #CFECEC, szerokość 60 %, wysokość 70 %, rozmiar czcionki 15 px;
- stopki: kolor tła #669966, biały kolor czcionki, wyrównanie tekstu do środka, rozmiar czcionki 120 %, krój czcionki Tahoma, wysokość stopki 10 %, marginesy wewnętrzne 10 px;
- tabela: szerokość: 300 px, obramowanie 2 px kolor zielony, wyrównanie tekstu do środka;

- paragraf: wyrównanie tekstu do środka, krój czcionki Tahoma, kolor czcionki ##003300, rozmiar czcionki 20 px, tekst pogrubiony;
- nagłówek tabeli: wypełnienie kolorem #006633, biały kolor czcionki.

Cechy witryny:

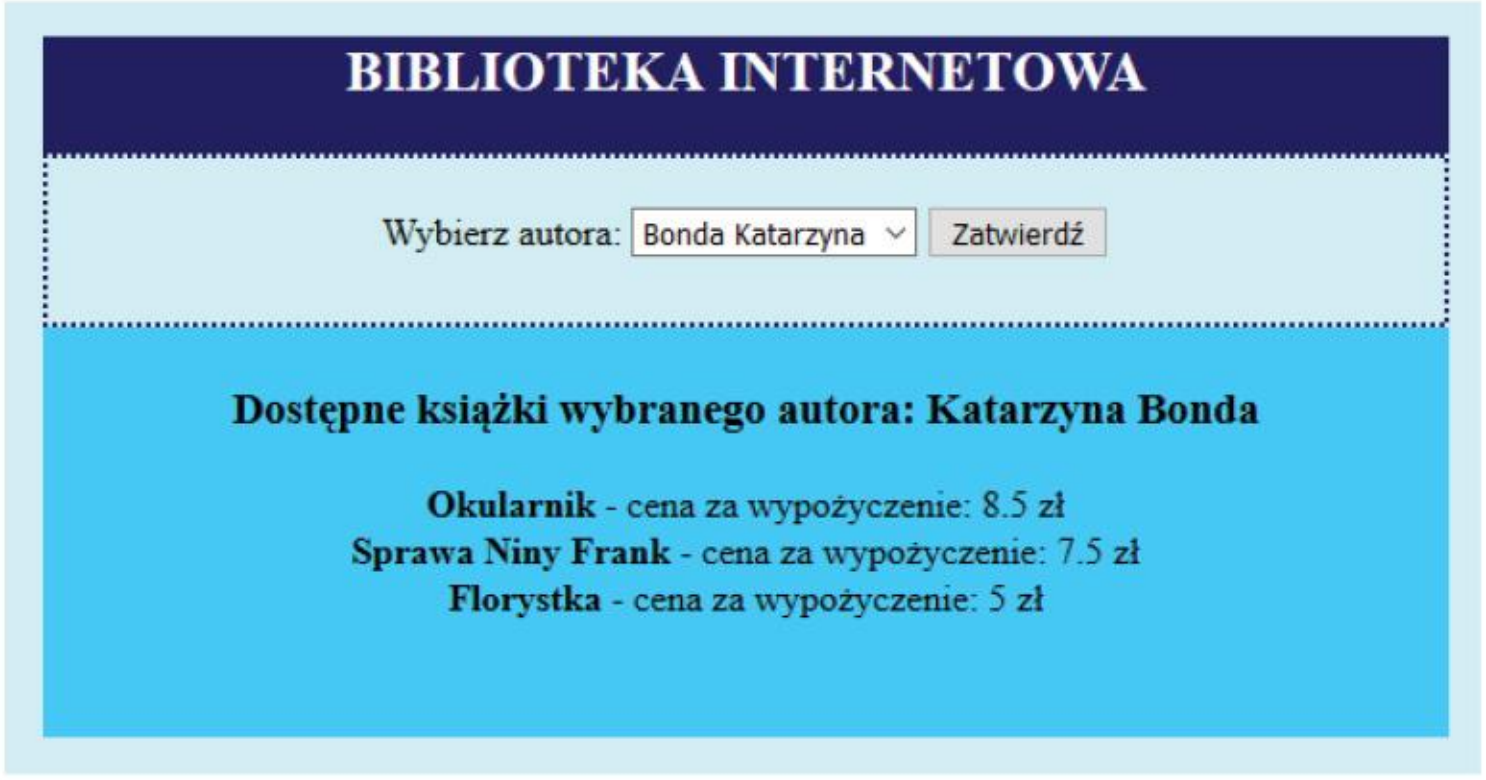
- nazwa pliku: *firma.php*;
- zastosowany standard kodowania polskich znaków;
- tytuł strony widoczny na karcie przeglądarki: „Baza danych o pracownikach firm";
- arkusz stylów w pliku o nazwie firma.css prawidłowo połączony z kodem strony;
- podział strony na bloki: baner, panel lewy, panel prawy, stopka – zrealizowany za pomocą znaczników sekcji zgodnie z rysunkiem;
- zawartość banera: nagłówek pierwszego stopnia o treści: „BAZA DANYCH O PRACOWNIKACH";
- zawartość panelu lewego: nagłówek drugiego stopnia o treści: „Informatycy poniżej roku 1975" oraz tabela wypełniona za pomocą skryptu nr 1;
- zawartość panelu prawego: nagłówek drugiego stopnia o treści: „Sekretarki firmy „Omega"", lista numerowana wypełniona za pomocą skryptu nr 2. Pod listą linia pozioma, pod którą umieść informacje wypełnione za pomocą skryptu nr 3.
- zawartość stopki: wpisz treść: „Autor:" i podaj swój PESEL. Z lewej i prawej strony stopki wstaw plik *logoc.png*.

  Skrypt nr 1 (w języku PHP) – wyświetla imiona, nazwiska i pensje informatyków (mężczyzn), których urodzili się przed rokiem 1975. Informacje wyświetl malejąco według pensji (zapytania do bazy *firma.sql*).

  Skrypt nr 2 (w języku PHP) – wyświetla imiona i nazwiska sekretarek z firmy „Omega", których pensja wynosi 2000 (zapytania do bazy *firma.sql*).

  Skrypt nr 3 (w języku PHP) – wyświetla najwyższą, najniższą i średnią pensję z całej bazy *firma.sql*.

2. Utwórz aplikację internetową obsługującą bazę danych dla biblioteki. Interfejs aplikacji wykonaj według wzoru przedstawionego na rys. 51.5.



Rys. 51.5. Postać aplikacji internetowej

Dodatkowe wytyczne dotyczące wizualizacji witryny zawarto w opisach kodu HTML i arkusza CSS. Aby przygotować poprawnie działającą aplikację, utwórz bazę danych **BIBLIOTEKA** według listingu zamieszczonego w załączniku nr 3.

Cechy CSS witryny internetowej:

- Plik *book1.css* zawiera formatowanie dla:
- sekcji głównej: kolor tła #CCFFFF;
- sekcji artykułu: wysokość 400 px, szerokość 600 px, marginesy wewnętrzne i zewnętrzne ustawione na auto;
- sekcji nagłówka: kolor tła #000066, wysokość 50 px, wyrównanie tekstu do środka, kolor czcionki biały;
- sekcji o identyfikatorze **gorna**: kolor tła #CCFFFF, wysokość 50 px, wyrównanie tekstu do środka, margines wewnętrzny górny 20 px, obramowanie 2 px, kropkowane kolor #000066;
- sekcji o identyfikatorze **dolna**: kolor tła #33CCFF, wyrównanie tekstu do środka, wysokość 170 px, margines wewnętrzny górny 5 px.

Cechy witryny:

- nazwa pliku: *book1.php*;
- zastosowany standard kodowania polskich znaków;
- tytuł strony widoczny na karcie przeglądarki: „Biblioteka internetowa”;
- arkusz stylów w pliku o nazwie *book1.css* prawidłowo połączony z kodem strony;
- podział strony na: sekcję artykułu, w której znajdują się sekcja nagłówka oraz dwie sekcje o identyfikatorze **gorna** oraz **dolna**, zrealizowany za pomocą znaczników sekcji zgodnie z rys. 51.5;
- zawartość banera: nagłówek pierwszego stopnia o treści: „BIBLIOTEKA INTENETOWA”;
- zawartość sekcji górnej: formularz zawierający listę z nazwiskami autorów istniejących w bibliotece: Bonda Katarzyna, Coben Harlan, Link Charlotte, Nesbo Jo oraz przycisk „Zatwierdź”;
- zawartość sekcji dolnej: nagłówek trzeciego stopnia o treści „Dostępne książki wybranego autora:” oraz imię i nazwisko autora wybranego z listy formularza. Poniżej lista książek wybranego autora razem z ceną za wypożyczenie. Dane mają być pobierane z bazy *biblioteka.sql* za pomocą skryptu działającego po stronie serwera.

52 Pliki cookies i sesje

ZAGADNIENIA

- Pliki cookies w PHP
- Sesje w języku PHP

Pliki cookies

Pliki cookies (z ang. *cookie* – 'ciastko') to tekstowe porcje danych, które są wymieniane między przeglądarką a serwerem. Te pliki są przechowywane przez przeglądarkę w systemie użytkownika i przesyłane w postaci nagłówka http o nazwie **`Set-Cookie`**.

Pliki cookies umożliwiają zliczanie odwiedzin danego użytkownika na stronie oraz pozwalają zapamiętywać dane użytkownika. Są one przechowywane na dysku do czasu ich wygaśnięcia. Jeśli nie zostanie określony czas wygaśnięcia, to plik straci ważność po zamknięciu przeglądarki.

Istnieje możliwość ustawienie opcji przeglądarki w taki sposób, że przed pobraniem pliku będzie się wyświetlał odpowiedni komunikat lub wszystkie pliki cookies zostaną odrzucone.

Aby utworzyć pliki cookies w skrypcie PHP, należy zastosować funkcję **`setcookies()`** o następującej postaci:

```
setcookie($nazwa, $wart, $czas, $sciezka, $domena, $bezp);
```

Ta funkcja może mieć sześć argumentów, którymi są:

- **`$nazwa`** – nazwa ciasteczka, argument obowiązkowy;
- **`$wart`** – wartość przechowywana w pliku cookie;
- **`$czas`** – znacznik czasu, który określa datę wygaśnięcia pliku cookie;
- **`$sciezka`** – określa ścieżkę dostępu na serwerze, do której zostanie ograniczona dostępność cookie;
- **`$domena`** – określa adres, dla którego cookie jest dostępne;
- **`$bezp`** – wartość parametru ustawiona na **`TRUE`** oznacza, że cookies będzie przesłane tylko przez bezpieczne połączenie **`https`**; wartość **`FALSE`** (wartość domyślna) oznacza, że cookie będzie wysyłany przez zwykłe połączenie **`http`**.

Ponieważ pliki cookies są częścią nagłówka http, funkcja **`setcookie()`** musi być wywołana, zanim jakiekolwiek dane zostaną wysłane do przeglądarki. Należy ją zatem umieścić przed znacznikiem **`<html>`**.

PRZYKŁAD 52.1

Wysłanie plików cookies do przeglądarki odbywa się za pomocą następującego kodu:

```
<?php
setcookie('moje');
?>
<!DOCTYPE HTML>
<html>
<head>
    <meta charset="utf-8" />
    <title>Ciasteczka</title>
</head>
<body>
</body>
</html>
```

Pliki cookies są wysyłane przez przeglądarkę do serwera i umieszczane w tablicy **$_COOKIES**. Indeksami tej tablicy są nazwy plików, a dostęp do poszczególnych plików odbywa się za pomocą polecenia:

```
$_COOKIE['nazwa'];
```

PRZYKŁAD 52.2

```
<?php
if (!isset($_COOKIE['moje']))
{
    setcookie('moje','ciacho1');
      $info="Ciasteczko nie jest ustawione";
}
else
{
    $info="Ciasteczko jest ustawione, a jego wartością jest
    ".$_COOKIE['moje'];
}
?>
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Ciasteczka</title>
```

```
</head>
<body>
    <?php
    echo $info;
    ?>
</body>
</html>
```

Skrypt z przykładu 52.2 wywołuje funkcję **isset()**, której zadaniem jest sprawdzenie, czy do serwera przesłano ciasteczko o nazwie ‚moje'. Jeśli funkcja zwróci wartość **FALSE**, to funkcja **setcookie()** ustawi cookie o nazwie **'moje'** i wartości **'ciacho1'**, a zmienna **$info** przechowa informację, że ciasteczko nie zostało ustawione. W przeciwnym razie **$info** przechowuje wartość odczytaną z cookie oraz komunikat, że ciasteczko jest ustawione.

Jak już nadmieniono, pliki cookies zostają domyślnie usunięte po zamknięciu okna przeglądarki. Aby usunąć te pliki samodzielnie z komputera użytkownika, należy:

- ustawić czas ważności, który upłynął;
- ustawić wartość pliku cookie pusty ciąg znaków lub wartość **FALSE**.

PRZYKŁAD 52.3

- `setcookie('moje','ciasteczko1',time()-100);`
- `setcookie("moje","");`
- `setcookie("moje", FALSE);`

Sesje

Mechanizm sesji pozwala na śledzenie działań użytkownika na stronie. Sesja to czas, w którym użytkownik jest aktywny na stronie internetowej. Każda sesja użytkownika ma własny identyfikator. Jest to liczba generowana losowo przez PHP i przechowywana w ciasteczkach lub w adresie URL. Każdej sesji można przypisać zmienne, do których będzie dostęp przez cały czas trwania danej sesji. Te zmienne są przechowywane na serwerze w tablicy **$_SESSION**.

Aby rozpocząć sesję, należy użyć bezargumentowej funkcji **session_start()**, natomiast do zakończenia sesji – bezargumentowej funkcji **session_destroy()**.

PRZYKŁAD 52.4

```
<?php
session_start();
echo "Identyfikator sesji: ".session_id();
session_destroy();
?>
```

Do pobrania identyfikatora sesji zastosowano funkcję **session_id()**.

Wszystkie zmienne związane z aktywnością użytkownika na stronie są przechowywane na serwerze w superglobalnej tablicy **$_SESSION**. Aby zapisać zmienną sesyjną, należy zastosować polecenie:

```
$_SESSION['nazwa_zmiennej']=wartość;
```

Aby sprawdzić, czy określona zmienna istnieje, należy zastosować funkcję **isset()** w następującej postaci:

```
isset($_SESSION['nazwa_zmiennej']);
```

Aby wyrejestrować zmienną przed zakończeniem sesji, należy zastosować funkcję **unset()** w postaci:

```
unset($_SESSION['nazwa_zmiennej']);
```

Zmienne utworzone w sesji są przechowywane w pliku tymczasowym, a dodanie kolejnej zmiennej do tablicy polega na dopisaniu jej nazwy i wartości do tego pliku.

PRZYKŁAD 52.5

```
<?php
session_start();
if (!isset($_SESSION['liczba']))
{
    $_SESSION['liczba']=0;
}
else
{
    $_SESSION['liczba']=$_SESSION['liczba']+1;
    echo 'Odwiedziłeś nas: '.$_SESSION['liczba'];
}
?>
```

Zadaniem skryptu jest otwarcie sesji. Za pomocą funkcji **isset()** sprawdza się, czy istnieje zmienna sesyjna o nazwie **$liczba**. Jeżeli zmienna nie jest zarejestrowana, przypisuje się jej wartość 0. W przeciwnym razie wartość zmiennej jest inkrementowana o 1. W oknie przeglądarki wyświetla się komunikat, który informuje o liczbie odwiedzin przez danego użytkownika w danej przeglądarce.

Mechanizm działania sesji można zastosować podczas autoryzacji użytkownika na stronie.

PRZYKŁAD 52.6

Utworzymy następujące pliki:

- **logowanie.php** – plik zawierający formularz logowania oraz procedury autoryzacji użytkownika (sprawdzenie, czy istnieją dany login i hasło w bazie użytkowników);
- **wyloguj.php** – plik zawierający instrukcję umożliwiającą powrót do strony logowania i zamknięcie sesji;
- **nowy.php** – plik umożliwiający rejestrowanie nowego użytkownika.

Kod źródłowy pliku **logowanie.php**:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Sesje</title>
<?php
    session_start();
      if (isset($_POST['login']) && isset($_POST['haslo']))
      {
        $login=$_POST['login'];
        $haslo=$_POST['haslo'];
      $conn=mysqli_connect('localhost','root','','baza1');
        $z1=mysqli_query($conn,'select  login,  haslo  from
        uzytkownik');
        while($z2=mysqli_fetch_array($z1))
    {
    if ($login==$z2['login'] && $haslo==$z2['haslo'])
        {
        $_SESSION['zalogowany']=$z2['login'];
        header('location: wyloguj.php');
          exit();
        }
          else
          {
            $_SESSION['zalogowany']=0;
            header('location:nowy.php');
    }
    }
    }
    ?>
</head>
```

```
<body>
    <b>ZALOGUJ SIĘ NA STRONĘ</b>
    <form action="" method="POST">
    <br>Login: <input type="text" name="login">
    <br>Hasło: <input type="password" name="haslo">
    <br><input type="submit" value="Zaloguj">
    </form>
</body>
</html>
```

Kod źródłowy pliku wyloguj.php:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Sesje</title>
    </head>
<?php
session_start();
if (isset($_SESSION['zalogowany']))
{
    echo "Jesteś zalogowany jako: ".$_SESSION['zalogowany'];
}
session_destroy();
?>
<body>
    <a href="rejestracja.php">Powrót do strony logowania</a>
</body>
</html>
```

Kod źródłowy pliku nowy.php:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Sesje</title>
</head>
<body>
    <b>ZAREJESTRUJ SIĘ NA STRONIE</b>
    <form action="" method="POST">
    <br>Login: <input type="text" name="login">
```

```
    <br>Hasło: <input type="password" name="haslo">
    <br><input type="submit" value="ZAREJESTRUJ SIĘ">
    </form>
    <?php
session_start();
if (isset($_POST['login']) && isset($_POST['haslo']))
    {
    $login=$_POST['login'];
    $haslo=$_POST['haslo'];
    $_SESSION['rejestracja']=1;
    $conn=mysqli_connect('localhost','root','','baza1');
    $z1=mysqli_query($conn,"insert into uzytkownik
    values(NULL,'$login','$haslo')");
    header('location: logowanie.php');
    exit();
    }
?>
</body>
</html>
```

W pierwszym z plików **logowanie.php** zawarto formularz umożliwiający użytkownikowi zalogowanie się na stronę. Po otwarciu sesji sprawdza się, czy uzupełniono pola **login** i **hasło**. Następnie dochodzi do połączenia z bazą, w której znajdują się loginy i hasła zarejestrowanych użytkowników. Jeśli podany login i hasło zgadzają się z danymi istniejącymi w bazie, to zmienna sesyjna o nazwie „zalogowany" przyjmuje wartości loginu użytkownika i następuje przekierowanie do strony **wyloguj.php**. W przeciwnym razie zmienna sesyjna przyjmuje wartość 0 i następuje przejście do strony **nowy.php**.

W pliku **wyloguj.php** skrypt sprawdza, czy istnieje zmienna sesyjna. Jeśli tak, to w oknie przeglądarki zostaje wyświetlony komunikat o zalogowaniu użytkownika i wstawiony link umożliwiający powrót do strony głównej.

Plik **nowy.php** umożliwia rejestrację nowego użytkownika. W oknie przeglądarki ponownie zostaje wyświetlony formularz rejestracyjny. Po wprowadzeniu nowych danych dochodzi do połączenia z bazą danych i wstawienia do niej danych z formularza. Po tym następuje przejście do strony, na której można się zalogować za pomocą nowego loginu i hasła.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Do czego służą pliki cookie?
2. Co to jest sesja?
3. Podaj sposób definiowania zmiennej sesyjnej.
4. Opisz składnię funkcji `setcookie()`.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz aplikację internetową, która będzie zawierała formularz logowania. Po zalogowaniu na konto dyrektora (login:Dyr2020, hasło: qwert34!) powinna wyświetlić się strona z listą uczniów (imię i nazwisko) oraz ich ocenami z poszczególnych przedmiotów. Dane uczniów i ich oceny pobierz z utworzonej wcześniej bazy o nazwie SZKOŁA.

53 Obsługa plików w PHP

ZAGADNIENIA

- Zapisywanie danych do pliku
- Odczytywanie zawartości plików

Zapisywanie danych do pliku

Pliki w PHP są obsługiwane podobnie jak w języku C++. Zapisywanie danych do pliku tekstowego odbywa się w następujących etapach:

1. otwarcie pliku istniejącego lub utworzenie nowego;
2. wprowadzanie danych do pliku;
3. zamknięcie pliku.

Plik otwiera się za pomocą funkcji **`fopen()`** o następującej postaci:

```
fopen(nazwa_pliku, tryb_otwierania);
```

Podczas wpisywania nazwy pliku należy pamiętać o podaniu ścieżki dostępu do tego pliku.

Funkcja **`fopen()`** obsługuje następujące tryby otwarcia pliku:

„r” Plik otwarty tylko do odczytu, wskaźnik jest umieszczony na początku pliku. Jeśli plik nie istnieje, funkcja zwraca wartość **`FALSE`**.

„r+” Plik otwarty tylko do odczytu i zapisu, wskaźnik jest umieszczony na początku pliku. Jeśli plik nie istnieje, funkcja zwraca wartość **`FALSE`**.

„w” Plik otwarty tylko do zapisu, wskaźnik jest umieszczony na początku pliku, a zawartość pliku zostaje wyzerowana. Jeśli plik nie istnieje, funkcja próbuje go otworzyć.

„w+” Plik otwarty tylko do odczytu i zapisu, wskaźnik jest umieszczony na początku pliku, a zawartość pliku zostaje wyzerowana. Jeśli plik nie istnieje, funkcja próbuje go otworzyć.

„a” Plik otwarty tylko do zapisu, wskaźnik jest umieszczony na końcu pliku. Jeśli plik nie istnieje, funkcja próbuje go otworzyć.

„a+” Plik otwarty tylko do zapisu i odczytu, wskaźnik jest umieszczony na końcu pliku. Jeśli plik nie istnieje, funkcja próbuje go otworzyć.

PRZYKŁAD 53.1

Weźmy pod uwagę skrypt tworzący plik o nazwie **wizytowka.txt**. Zapiszemy tam imię i nazwisko użytkownika.

```
<?php
$plik=fopen("wizytowka.txt","a");
$dane="Anna Kowalska";
fwrite($plik,$dane);
fclose($plik);
?>
```

Po otwarciu pliku do zapisu została zdefiniowana zmienna **$dane**, a jej zawartość umieszczona w pliku za pomocą funkcji **fwrite()**. Pierwszym argumentem tej funkcji jest zmienna plikowa, a drugim – zmienna będąca treścią przekazywanej informacji do pliku. Ta funkcja ma również argument opcjonalny, który umożliwia określenie maksymalnej liczby wprowadzanych znaków.

UWAGA

Funkcją równoważna jest **fputs()**.

Po zakończeniu wszystkich działań na pliku należy go zamknąć za pomocą funkcji **fclose()**.

PRZYKŁAD 53.2

Możemy oczywiście wpisywać do pliku liczby wygenerowane do tablicy indeksowanej.

```
<?php
$plik=fopen("liczby.txt","a");
    for($i=0;$i<10;$i++)
    {
       $tab[$i]=rand(10,20);
       fwrite($plik,$tab[$i]."\n");
    }
    fclose($plik);
?>
```

Odczytywanie zawartości plików

Odczytywanie z pliku również odbywa się w etapach:

1. otwarcie pliku,
2. odczytanie danych z pliku,
3. zamknięcie pliku.

Aby odczytać zawartość z pliku, można skorzystać z jednej z wymienionych funkcji:

- **`fread()`,**
- **`fgets()`,**
- **`fgetc()`.**

Funkcja **`fread()`** ma za zadanie odczytanie z pliku określoną liczbę znaków. Odczytywanie danych zakończy się, gdy wartość zostanie określona drugim argumentem funkcji. Pierwszym argumentem funkcji jest zmienna plikowa.

PRZYKŁAD 53.3

```
<?php
$plik=fopen("wizytowka.txt","r");
$tresc=fread($plik,4);
echo $tresc;
fclose($plik);
?>
```

Skrypt otwiera do odczytu plik **`wizytowka.txt`** z przykładu 53.1. Następnie za pomocą funkcji **`fread()`** odczytuje treść pliku o maksymalnej długości czterech znaków. Efektem będzie wyświetlenie imienia „Anna".

Innym sposobem na czytanie zawartości pliku jest jednoargumentowa funkcja **`fgets()`**. Jej zadaniem jest czytanie jednej linii pliku aż do napotkania znacznika przejścia do kolejnej linii. Aby odczytać całą zawartość pliku, należy zastosować pętlę. Podczas stosowania tej funkcji jedynym problemem może być to, czy dany plik został odczytany w pełni lub czy nie przekroczono liczby odczytanych znaków. W tym celu należy zastosować funkcję **`feof()`**, która zwraca wartość **`TRUE`**, jeżeli osiągnięto koniec pliku.

PRZYKŁAD 53.4

W pliku **`liczby.txt`** znajduje się dziesięć liczb całkowitych. Wygenerowano je za pomocą skryptu z przykładu 53.2.

Rys. 53.1. Zawartość pliku tekstowego **liczby.txt**

Te liczby odczytamy i wyświetlimy w oknie przeglądarki – w postaci kolumnowej, czyli w takiej, w jakiej wpisano je do pliku.

```
<?php
$plik=fopen("liczby.txt","r");
    $i=0;
    while (!feof($plik))
    {
       $liczba[$i]=fgets($plik);
       echo $liczba[$i]."<br>";
       $i++;
    }
    fclose($plik);
?>
```

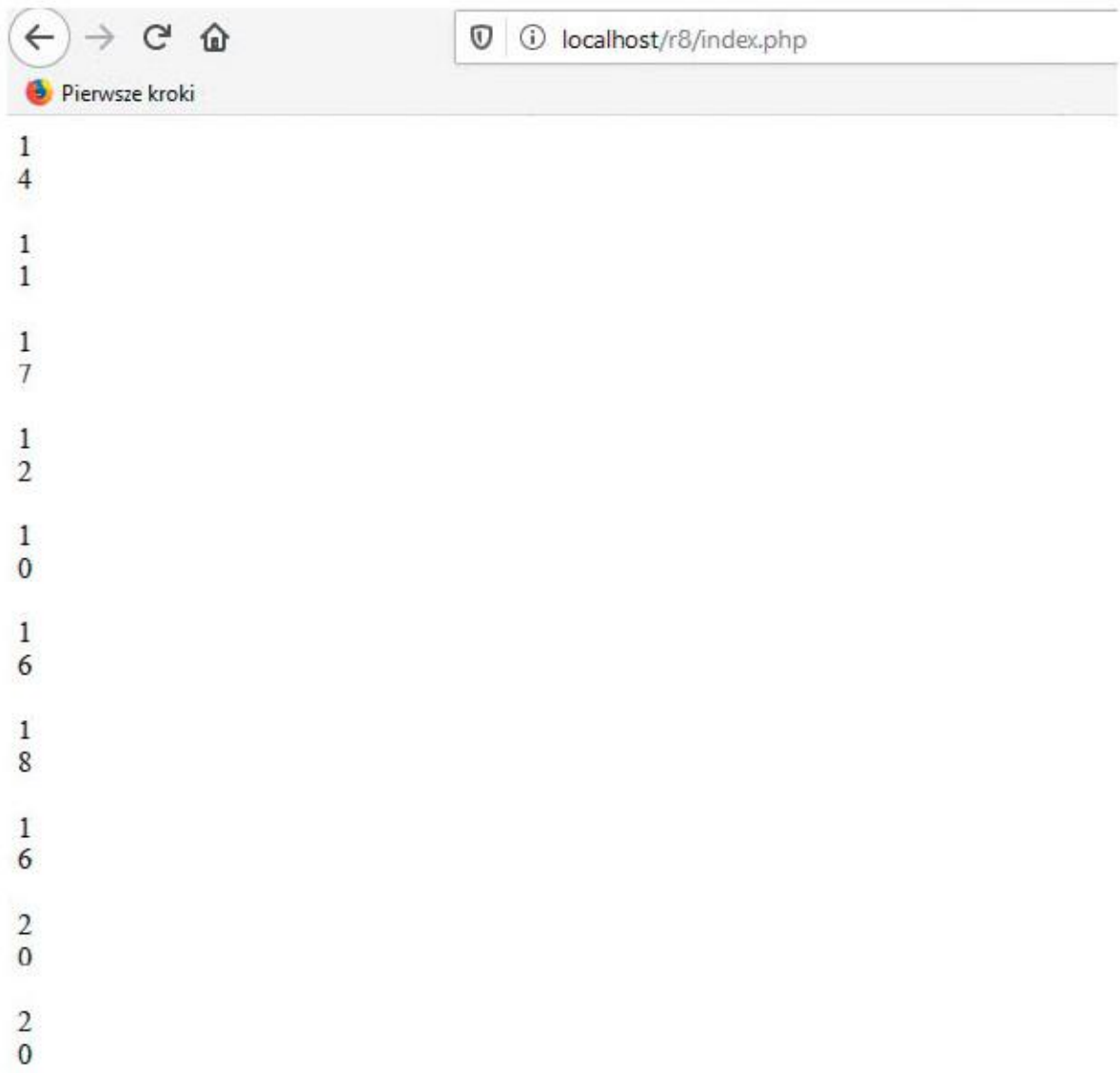
Ostatnim sposobem odczytywania danych z pliku jest funkcja **fgetc()**. Różni się od funkcji **fgets()** tym, że dane z pliku są odczytywane po jednym znaku.

PRZYKŁAD 53.5

W skrypcie z przykładu 53.4 funkcję **fgest()** zamienimy na funkcję **fgetc()**.

```
<?php
$plik=fopen("liczby.txt","r");
    $i=0;
    while (!feof($plik))
    {
       $liczba[$i]=fgetc($plik);
       echo $liczba[$i]."<br>";
       $i++;
    }
?>
```

Wprowadzenie funkcji **`fgetc()`** poskutkuje wyświetleniem każdej cyfry w oddzielnym wierszu.



Rys. 53.2. Efekt działania funkcji **`fgetc()`**

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień etapy zapisu danych do pliku.
2. Opisz etapy odczytywania danych z pliku.
3. Jaka jest różnica między funkcją **`fgets()`** a funkcją **`fegtc()`**?
4. Jaka funkcja służy do otwarcia pliku tekstowego?
5. Wymień i omów tryby otwarcia pliku.

SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Utwórz skrypt, który wylosuje 15 liczb z przedziału <1,50> i zapisze je do pliku o nazwie **`dane.txt`**. Następnie pobierz te dane, wyświetl w oknie przeglądarki tylko parzyste, oblicz ich sumę, a wynik zapisz w nowo utworzonym pliku o nazwie **`wynik.txt`**.
2. Utwórz skrypt otwierający plik o nazwie **`osoby.txt`**, w którym są zapisane imiona i nazwiska pięciu osób. Pobierz te dane do tablicy asocjacyjnej, posortuj według nazwisk, a następnie uporządkowane dane zapisz do pliku o nazwie **`posortowane.txt`**.

X. Projektowanie i dokumentowanie oprogramowania

- Etapy projektowania
- Dokumentowanie programów
- Przykład realizacji aplikacji

54 Projektowanie programów

ZAGADNIENIA

- Definicja tworzenia i projektu oprogramowania
- Etapy tworzenia oprogramowania
- Metody projektowania

Tworzenie oprogramowania to proces określony w czasie, złożony z wielu czynności i związanych z nimi wyników, które mają prowadzić do powstania nowego produktu w postaci programu komputerowego, aplikacji internetowej lub aplikacji mobilnej.

Proces projektowania może obejmować:

- tworzenie nowego produktu programistycznego na indywidualne zamówienie klienta,
- modernizację bądź rozbudowę istniejących systemów informatycznych lub aplikacji,
- modyfikację istniejących modułów w celu wykorzystania ich do nowych zastosowań.

W każdym procesie tworzenia oprogramowania można wyróżnić etapy, takie jak:

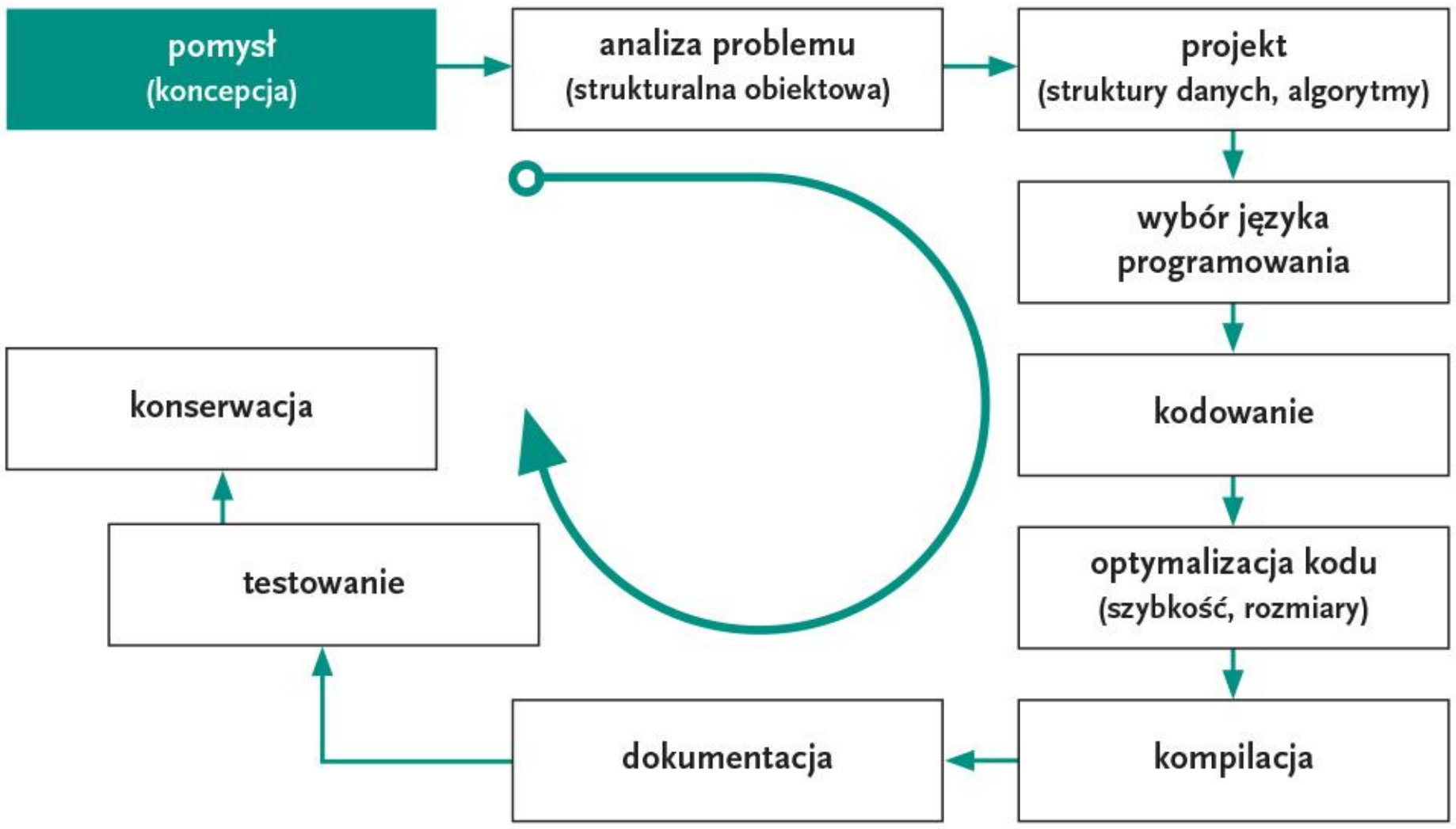
1. Planowanie – zdefiniowanie problemu, który trzeba rozwiązać, oraz dokładna jego analiza i wybór metody jego rozwiązania.
2. Implementacja (kodowanie) – zapisanie algorytmu w postaci kodu źródłowego.
3. Kompilacja – zamiana kodu źródłowego na binarny.
4. Konsolidacja – łączenie plików obiektowych i biblioteki w program wykonywalny.
5. Testowanie – wykrywanie błędów.
6. Wdrażanie i optymalizacja – ulepszanie programu.
7. Użytkowanie i ewaluacja.

Planowanie

Pierwszym etapem tworzenia programu komputerowego jest określenie problemu, który należy rozwiązać. Formułujemy problem, który rozwiązujemy za pomocą algorytmiki. Program komputerowy powinien być funkcjonalny, a zadanie stawiane programiście powinno być możliwe do wyrażenia za pomocą algorytmu i przede wszystkim rozwiązywalne. W wyniku analiz przeprowadzonych na tym etapie powinniśmy otrzymać dokładną specyfikację wymagań systemowych.

Implementacja

Kodowanie jest bardzo ważnym etapem, ponieważ wymaga dokładnego jego udokumentowania. Zapis powinien być czytelny. Ważne jest właściwe zorganizowanie kodu źródłowego.



Rys. 54.1. Etapy tworzenia programu

Należy stosować odpowiednie komentarze, tak aby osoba przejmująca kodowanie potrafiła szybko zorientować się w nazwach zmiennych oraz funkcji. Ponadto wcięcia ułatwiają znajdowanie instrukcji, co przyśpiesza wyszukiwanie błędów oraz modyfikację programu.

Kompilacja

Podczas pisania programu konieczne jest przetestowanie fragmentów kodu pod kątem poprawności syntaktycznej, zawartości błędów i zgodności działania z zamierzeniem programisty.

Konsolidacja

Konsolidacja, czyli linkowanie (łączenie), polega na łączeniu poszczególnych modułów programu w celu wygenerowania programu. Istnieją dwa rodzaje konsolidacji:

- **statyczna** – najczęściej przeprowadzana podczas tworzenia programu; jest to proces dołączania modułów bibliotecznych do wykonywanego programu oraz łączenia wszystkich modułów składowych samego programu;
- **dynamiczna** – zachodzi podczas ładowania programu do pamięci, w trakcie współpracy systemu operacyjnego, co umożliwia lepsze wykorzystanie pamięci oraz ułatwia aktualizowanie bibliotek.

Testowanie

Testowanie programu odgrywa bardzo ważną rolę zarówno w procesie tworzenia aplikacji, jak i podczas jej użytkowania. Ma na celu wykrycie błędów, tzn. takich jego elementów, które mogą prowadzić do nieprawidłowego działania programu.

Wdrażanie i optymalizacja

Wdrożenie obejmuje instalację i konfigurację aplikacji oraz dostarczenie wsparcia technicznego. W niektórych sytuacjach może także dotyczyć szkolenia użytkowników.

Optymalizacja programu to czynności, których celem jest ulepszenie i poprawa wydajności programu komputerowego. Optymalizację działania programu uzyskujemy np. przez zmniejszenie ilości przydzielanej pamięci bądź przez szybkość działania programu czy usunięcie zbędnych linii kodu.

Program komputerowy rozwiązujący określony problem ma do dyspozycji dwa podstawowe zasoby: czas i pamięć. Czasowa złożoność obliczeniowa to ilość czasu niezbędnego do rozwiązania problemu zależnie od liczby danych wejściowych. Złożoność czasowa jest funkcją liczby danych wejściowych. Złożoność czasowa wyraża się w jednostkach czasu albo w liczbie operacji dominujących, które należy wykonać dla n danych, aby rozwiązać problem. Podawanie złożoności czasowej w jednostkach czasu jest niewygodne, ponieważ wynik zależy od szybkości komputera, na którym wykonano obliczenia. Częściej złożoność czasowa jest wyrażana w liczbie operacji dominujących, ponieważ każdy komputer musi te operacje wykonać.

Złożoność pamięciowa określa liczbę komórek pamięci, która będzie zajęta przez dane i wyniki pośrednie tworzone w trakcie pracy algorytmu.

Użytkowanie i ewaluacja

Proces użytkowania obejmuje efektywne korzystanie z aplikacji przez użytkownika. Ewaluacja polega na badaniu działania aplikacji w miejscu pracy i na nanoszeniu poprawek w celu doskonalenia i rozwijania funkcjonalności programu komputerowego.

Projekt oprogramowania to opis struktury oprogramowania, które będzie poddane implementacji, opis danych przetwarzanych przez program oraz opis użytych algorytmów.

Często do tworzenia projektów budowy poszczególnych modułów oprogramowania wykorzystuje się metody strukturalne polegające na tworzeniu graficznych modeli o określonej notacji oraz opisów zależności. Do najczęściej stosowanych zalicza się modele:

- przepływu danych,
- strukturalne,
- obiektowe,
- ER.

SPRAWDŹ SWOJĄ WIEDZĘ

1. Wymień etapy występujące w procesie projektowania aplikacji.
2. Co to jest projekt oprogramowania?
3. Na czym polega etap konsolidacji w procesie projektowania oprogramowania?

55 Dokumentowanie tworzonych aplikacji

ZAGADNIENIA

- Zasady dokumentowania tworzonych aplikacji
- Definicja repozytorium
- Rola komentarzy wewnątrz kodu
- Rodzaje programów do tworzenia dokumentacji
- Przykład realizacji prostej aplikacji

Wszystkie działania twórcy programu mające na celu opisanie rozwiązań programistycznych oraz wskazań dla przyszłego użytkownika sprowadzają się do tworzenia:

- dokumentacji technicznej programu,
- dokumentacji obsługi programu dla użytkownika.

Dokumentacja techniczna programu powinna zawierać, oprócz komentarzy wewnątrz kodu źródłowego, również opis projektu, którego efektem końcowym jest program. Taka dokumentacja zazwyczaj jest znormalizowana (np. ISO 9000) i dotyczy wszystkich etapów tworzenia aplikacji.

Na etapie realizacji projektu mającego na celu wytworzenie określonego programu **dokumentację techniczną** mogą tworzyć:

- opisy wymagań zleceniodawcy lub zapytania przetargowe,
- specyfikacje techniczne (np. zastosowany język programowania), architektura aplikacji,
- modele, schematy (np. diagramy UML), procedury bądź algorytmy,
- raporty, umowy, e-maile bądź inne formy komunikacji między zleceniodawcą a autorami aplikacji,
- listingi kodu źródłowego wraz z komentarzami,
- repozytorium programu,
- opis procesu testowania (np. raporty z testów).

W procesie dokumentowania prac nad oprogramowaniem bardzo przydatne są repozytoria. **Repozytorium** to zasób, w którym tworzy się całą historię zmian kodu. Dzięki temu każdy programista lub menedżer projektu mogą śledzić postępy w tworzeniu kodu oraz pracować nad własnymi poprawkami. Wyróżnia się trzy podstawowe rodzaje repozytoriów: lokalne, scentralizowane i rozproszone.

Do najistotniejszych zalet repozytorium zalicza się:

- śledzenie pełnej historii zmian pojawiających się w plikach;
- podgląd wcześniejszych zmian i możliwość sprawdzenia, kiedy i kto je utworzył;
- możliwość cofania zmian;
- możliwość sprawnego przeprowadzania testów i kopii oprogramowania.

Właściwie sporządzona dokumentacja techniczna jest podstawowym źródłem informacji dla programistów również w razie naprawy lub modyfikacji gotowego już programu.

W skład **dokumentacji obsługi programu dla użytkownika** mogą wchodzić:

- ogólny funkcjonalny opis systemu,
- wymagania sprzętowe,
- instrukcja obsługi programu,
- przewodnik instalacji i opis spotykanych problemów,
- informacje dotyczące pomocy technicznej,
- opis dokumentujący daną wersję programu.

Dokumentacja obsługi często jest dostępna w formie elektronicznej. Powinna zawierać szczegółowy spis treści oraz indeks pojęć.

Istotną rolę w dokumentowaniu aplikacji odgrywają **komentarze**. Zależnie od technologii wykorzystywanych w tworzeniu kodu wyróżnia się wiele metod i standardów komentowania kodu.

Dla języków systemowych wykorzystywanych w systemach Windows i Linux stosuje się określone polecenia lub symbole komentujące wykonywane skrypty. Przykładem takiego polecenia w konsoli cmd jest **REM** lub symbol `::`. W skryptach systemu Linux wykorzystuje się symbol **#**.

PRZYKŁAD 55.1

```
@echo off
rem to jest skrypt formatujący dysk zewnętrzny
:: o nazwie f
format f:
```

W rodzinie języków wywodzących składnię od C (C, C++, C#, PHP, Java) stosuje się style QtDoc lub JavaDoc.

PRZYKŁAD 55.2

```
QtDoc:
//! Tekst komentujący kod
lub
/*!
Tekst komentujący kod
```

```
*/
JavaDoc:
/// Tekst komentujący kod
lub
/**
Tekst komentujący kod
**/
```

W języku C# można spotkać się z komentarzami w postaci znaczników XML. Komentarze dokumentacji XML używają potrójnych ukośników **///** i treści komentarza w formacie XML. Jednym z podstawowych znaczników XML jest **<summary>**. Dodaje on krótkie informacje dotyczące typu lub składowej.

PRZYKŁAD 55.3

```
/// <summary>
/// Ta klasa robi ciekawe rzeczy
/// </summary>
public class SomeClass
{
}
```

W języku Python do komentowania kodu wykorzystuje się symbol **#**.

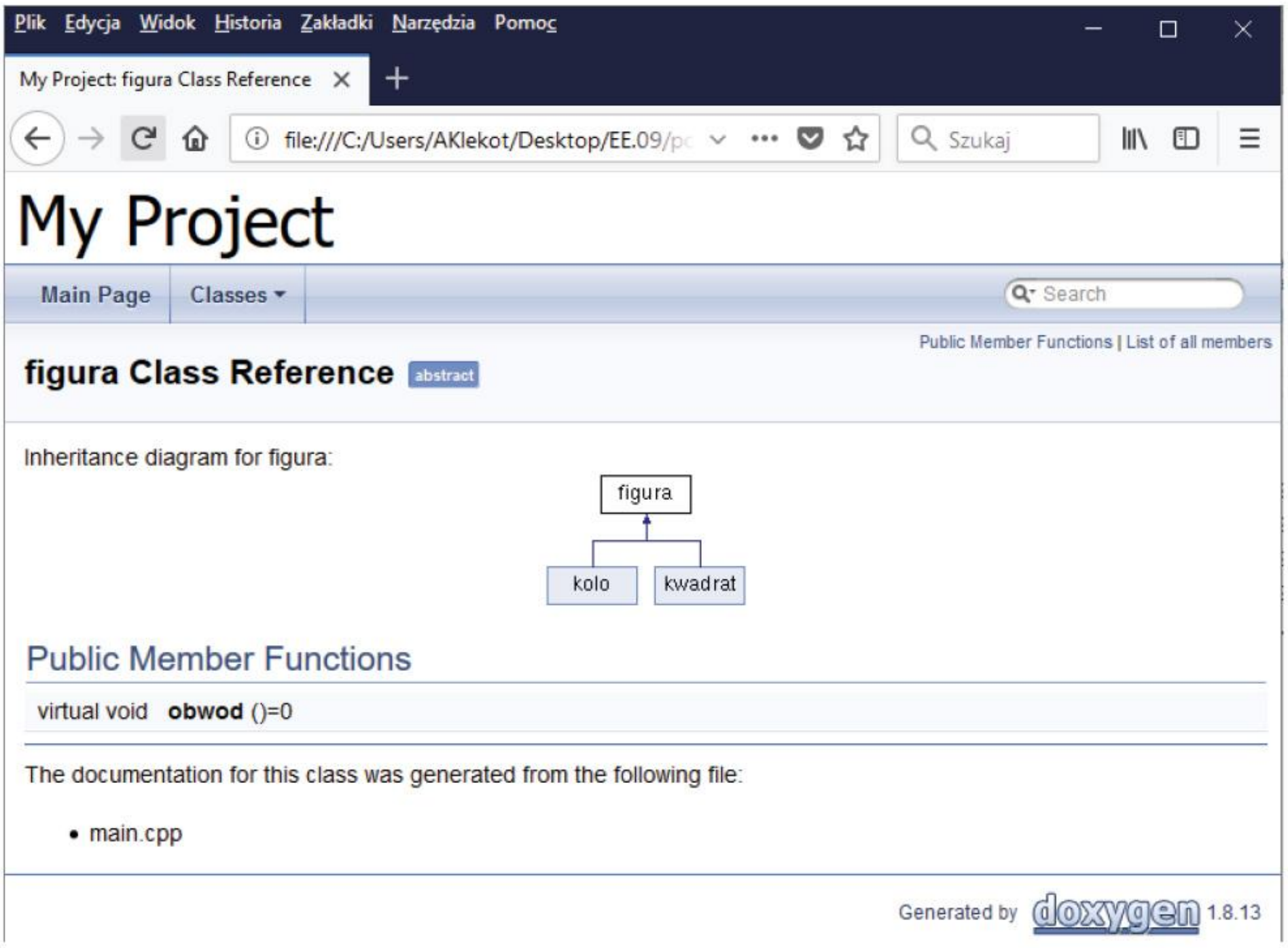
PRZYKŁAD 55.4

```
print(1) # oto komentarz
# print(2)
print(3) # inny komentarz
```

W każdym języku programowania komentarze są ignorowane przez interpreter lub kompilator.

Dużym ułatwieniem w tworzeniu dokumentacji technicznej programów jest wykorzystanie aplikacji specjalnie do tego przeznaczonych, np. Doxygen.

Doxygen to aplikacja dokumentowania oprogramowania pisanego w takich językach, jak C++, C, Java, Objective-C, Python, IDL, Fortran, VHDL, PHP, C# oraz D. Aplikacja może generować dokumentację w HTML, RTF, PDF oraz źródła dla edytora LaTeX. Dokumentacja może być tworzona bezpośrednio ze źródeł, jak też z plików stowarzyszonych z nimi. Zastosowanie oprogramowania do dokumentacji typu Doxygen ma sens w bardzo złożonych kodach źródłowych. W prostych aplikacjach główną metodą pozostaje komentowanie określonych części kodu.



Rys. 55.1. Przykład dokumentacji tworzonej w aplikacji Doxygen)

Przykład realizacji prostej aplikacji

Tworzenie aplikacji powinno się składać z następujących etapów:

1. **Sformułowanie zadania**

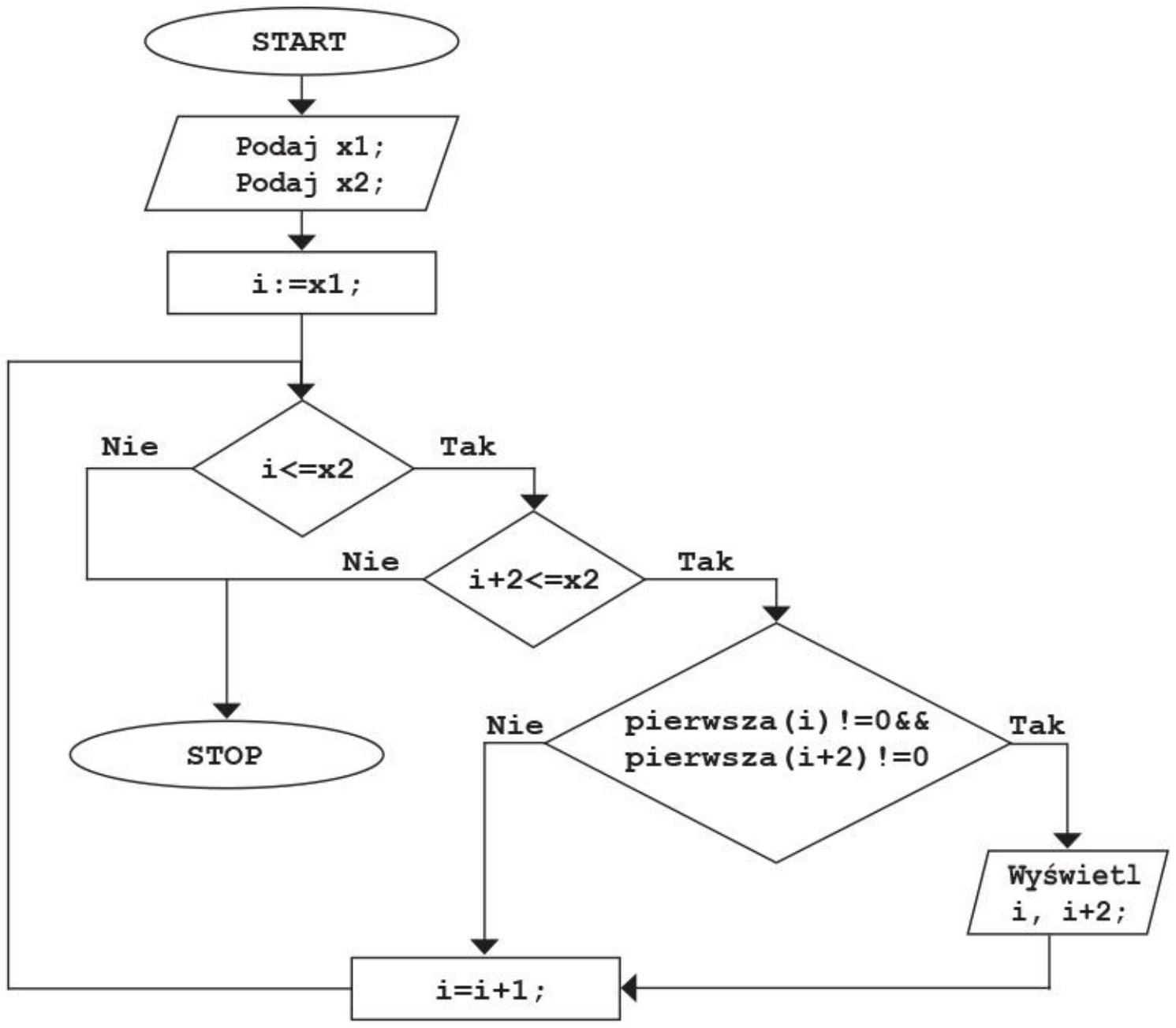
   Napisz program szukający liczb bliźniaczych w zadanym przedziale <x1, x2>. Liczby bliźniacze to dwie liczby pierwsze różniące się o 2 (np. 3 i 5 bądź 29 i 31).
   Utwórz funkcje, która sprawdzi, czy liczba z podanego zakresu jest liczbą pierwszą. Liczba pierwsza to liczba, która ma dokładnie dwa dzielniki.

2. **Specyfikacja algorytmu**

   Dane wejściowe: przedział <x1,x2>, gdzie x1,x2 to dowolne liczby całkowite.
   Dane wyjściowe: liczby bliźniacze, czyli para liczb pierwszych różniących się o 2.

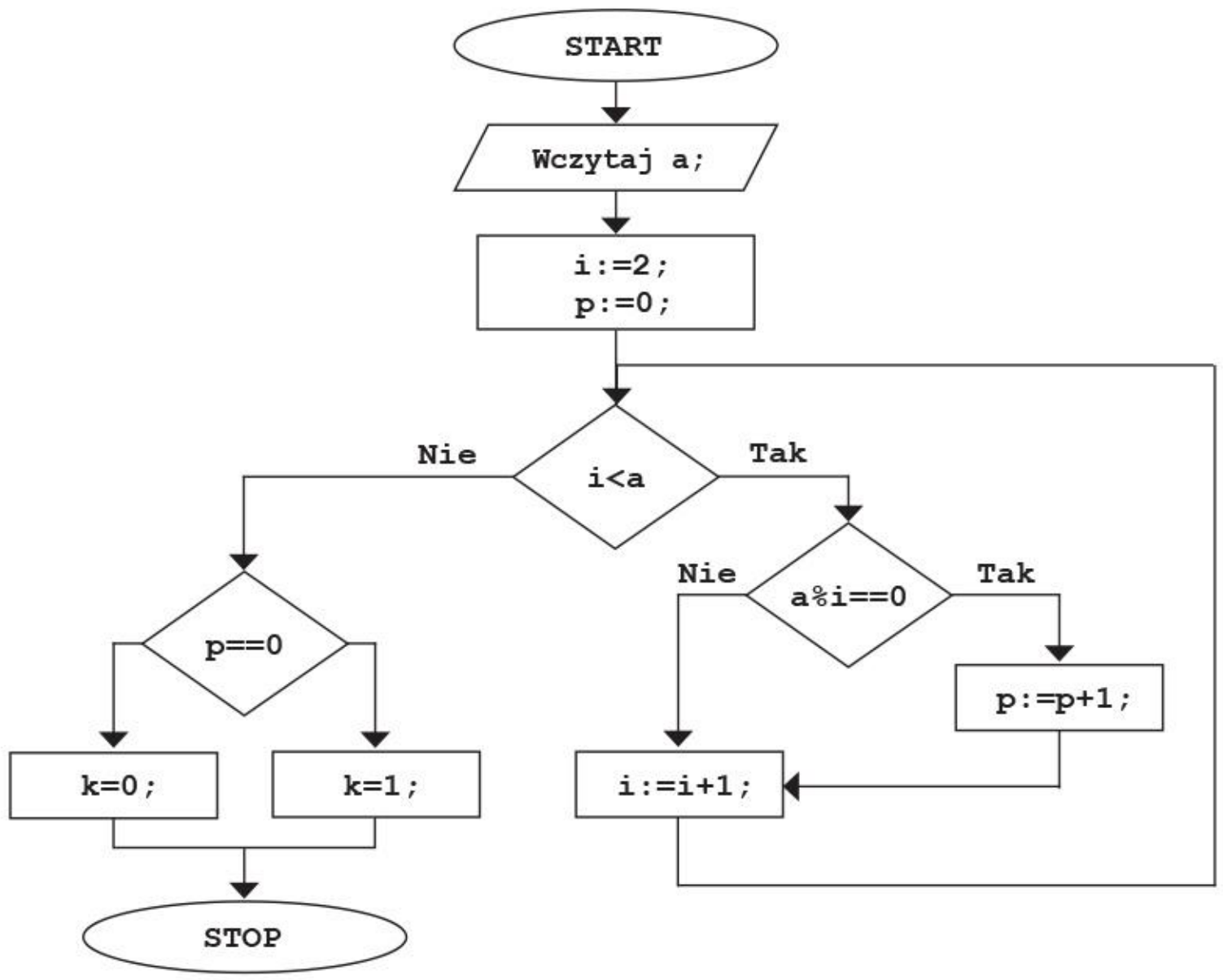
3. **Schemat blokowy algorytmu**

   Schemat blokowy głównego bloku programu.



Rys. 55.2. Schemat blokowy algorytmu głównego bloku programu

Schemat blokowy funkcji **`pierwsza(int a)`**, sprawdzającej, czy dana liczba jest liczbą pierwszą.



Rys. 55.3. Schemat blokowy algorytmu głównego bloku programu

4. Implementacja kodu na podstawie napisanych schematów blokowych

```
#include <iostream>
using namespace std;
int pierwsza(int a)
{
    int i,k,p=0;
    if(a<2) k=0;
    else
    {
    for(i=2; i<a; i++)          //sprawdzenie czy dana liczba a jest podzielna przez kolejne liczby naturalne, bez 1 i jej samej
    {
        if(a%i==0) p++;
    }
    if(p==0) k=1;
    else  k=0;
    }
    return k;
}
int main(void)
{
    int x1,x2;
    cout<<"Podaj x1= "; cin>>x1;
    cout<<"Podaj x2= "; cin>>x2;
    for(int i=x1;i<=x2;i++)
    {
        if(i+2<=x2)
        {
            if(pierwsza(i)!=0 && pierwsza(i+2)!=0) //wywolanie funkcji pierwsza
            {
            cout<<i<<", "<<i+2<<endl;
            }
        }
}
 return 0;
}
```

Rys. 55.4. Schemat blokowy algorytmu głównego bloku programu

SPRAWDŹ SWOJĄ WIEDZĘ

1. Co to jest repozytorium?
2. Wyszukaj w sieci Internet opisy programów do tworzenia dokumentacji oprogramowania. Sporządź notatkę w zeszycie przedmiotowym na ich temat.
3. Omów, w jaki sposób można komentować kod programu w różnych językach programowania.

ZAŁĄCZNIK 1

```
-- phpMyAdmin SQL Dump
-- version 4.8.5
-- https://www.phpmyadmin.net/
--
-- Host: 127.0.0.1
-- Czas generowania: 04 Lut 2020, 10:16
-- Wersja serwera: 10.1.38-MariaDB
-- Wersja PHP: 7.3.3

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Baza danych: `books`
--

-------------------------------------------------------

--
-- Struktura tabeli dla tabeli `klient`
--

CREATE TABLE `klient` (
  `Idklienta` int(11) NOT NULL,
  `imie` text COLLATE utf8_polish_ci NOT NULL,
  `nazwisko` text COLLATE utf8_polish_ci NOT NULL,
  `miejscowosc` text COLLATE utf8_polish_ci NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;

--
-- Zrzut danych tabeli `klient`
--
INSERT INTO `klient` (`Idklienta`, `imie`, `nazwisko`, `miejscowosc`)
VALUES
(1, ,Piotr', ,Lewandowski', ,Kielce'),
(2, ,Jan', ,Nowak', ,Katowice'),
(3, ,Maciej', ,Nowak', ,Bydgoszcz'),
(4, ,Agnieszka', ,Jankowska', ,Lublin'),
(5, ,Tomasz', ,Mazur', ,Zgierz'),
```

```
(6, ,Marek', ,Kowalski', ,Lublin'),
(7, ,Artur', ,Rutkowski', ,Kielce'),
(8, ,Mateusz', ,Skorupa', ,Warszawa'),
(9, ,Jerzy', ,Rutkowski', ,Rybnik'),
(10, ,Anna', ,Karenina', ,Katowice'),
(11, ,Maria', ,Lewandowska', ,Bydgoszcz'),
(12, ,Henryk', ,Kowalski', ,Bydgoszcz'),
(13, ,Agnieszka', ,Nowak', ,Warszawa'),
(14, ,Piotr', ,Krupski', ,Zgierz');

-- --------------------------------------------------------

--
-- Struktura tabeli dla tabeli `ksiazka`
--

CREATE TABLE `ksiazka` (
  `Idksiazki` int(11) NOT NULL,
  `imieautora` text COLLATE utf8_polish_ci NOT NULL,
  `nazwiskoautora` text COLLATE utf8_polish_ci NOT NULL,
  `tytul` text COLLATE utf8_polish_ci NOT NULL,
  `cena` float NOT NULL,
  `rokwydania` int(11) NOT NULL,
  `wydawnictwo` text COLLATE utf8_polish_ci NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;

--
-- Zrzut danych tabeli `ksiazka`
--

INSERT INTO `ksiazka` (`Idksiazki`, `imieautora`, `nazwiskoautora`,
`tytul`, `cena`, `rokwydania`, `wydawnictwo`) VALUES
(1, ,Jan', ,Michalak', ,Zaawansowane programowanie w PHP', 47.29, 2000,
,Albatros'),
(2, ,Andrzej', ,Krawczyk', ,Windows 8 PL. Zaawansowana administracja
systemem', 49.99, 2001, ,Czarny Kruk'),
(3, ,Piotr', ,Jakubowski', ,HTML5. Tworzenie witryn', 53.65, 2000,
,Albatros'),
(4, ,Tomasz', ,Kowalski', ,Urządzenia techniki komputerowej', 34.15,
2015, ,Komp2002'),
(5, ,Jan', ,Pasternak', ,PHP. Tworzenie nowoczesnych stron WWW', 29.99,
2015, ,EdukacjaPl'),
(6, ,Robin', ,Nixon', ,PHP, MySQL i JavaScript', 99, 2002, ,Czarny
Kruk'),
(7, ,Jolanta', ,Pokorska', ,Tworzenie stron internetowych', 23.95, 2000,
,EdukacjaPl'),
(8, ,Krzysztof', ,Pytel', ,Pracownia aplikacji internetowych', 35.4,
2000, ,EdukacjaPl');

-- --------------------------------------------------------
```

```
--
-- Struktura tabeli dla tabeli `zamowienie`
--

CREATE TABLE `zamowienie` (
  `Idzamowienia` int(11) NOT NULL,
  `Idklienta` int(11) NOT NULL,
  `Idksiazki` int(11) NOT NULL,
  `data` date NOT NULL,
  `status` text COLLATE utf8_polish_ci NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;

--
-- Zrzut danych tabeli `zamowienie`
--

INSERT INTO `zamowienie` (`Idzamowienia`, `Idklienta`, `Idksiazki`,
`data`, `status`) VALUES
(1, 2, 4, ,2014-10-08', ,oczekiwanie'),
(2, 7, 1, ,2014-09-05', ,wyslano'),
(3, 9, 1, ,2014-10-11', ,wyslano'),
(4, 2, 2, ,2014-10-15', ,oczekiwanie'),
(5, 2, 5, ,2014-08-12', ,oczekiwanie'),
(6, 3, 2, ,2014-10-20', ,wyslano'),
(7, 4, 3, ,2014-08-14', ,wyslano'),
(8, 8, 1, ,2014-08-19', ,wyslano'),
(9, 3, 5, ,2014-11-19', ,wyslano'),
(10, 9, 2, ,2014-12-28', ,oczekiwanie');

--
-- Indeksy dla zrzutów tabel
--

--
-- Indeksy dla tabeli `klient`
--
ALTER TABLE `klient`
  ADD PRIMARY KEY (`Idklienta`),
  ADD KEY `id` (`Idklienta`);

--
-- Indeksy dla tabeli `ksiazka`
--
ALTER TABLE `ksiazka`
  ADD PRIMARY KEY (`Idksiazki`);

--
-- Indeksy dla tabeli `zamowienie`
--
```

```
ALTER TABLE `zamowienie`
  ADD PRIMARY KEY (`Idzamowienia`);

--
-- AUTO_INCREMENT for dumped tables
--


--
-- AUTO_INCREMENT dla tabeli `klient`
--
ALTER TABLE `klient`
  MODIFY `Idklienta` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=15;

--
-- AUTO_INCREMENT dla tabeli `ksiazka`
--
ALTER TABLE `ksiazka`
  MODIFY `Idksiazki` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;

--
-- AUTO_INCREMENT dla tabeli `zamowienie`
--
ALTER TABLE `zamowienie`
  MODIFY `Idzamowienia` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=11;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

ZAŁĄCZNIK 2

```
-- phpMyAdmin SQL Dump
-- version 4.5.1
-- http://www.phpmyadmin.net
--
-- Host: 127.0.0.1
-- Czas generowania: 29 Gru 2017, 16:17
-- Wersja serwera: 10.1.13-MariaDB
-- Wersja PHP: 5.6.23
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
--
-- Baza danych: `firma`
--
-- --------------------------------------------------------
--
-- Struktura tabeli dla tabeli `pensje`
--
CREATE TABLE `pensje` (
`id` int(2) NOT NULL,
`id_os` int(2) NOT NULL,
`pensja` int(6) NOT NULL,
`od` date NOT NULL,
`do` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
--
-- Zrzut danych tabeli `pensje`
--
INSERT INTO `pensje` (`id`, `id_os`, `pensja`, `od`, `do`) VALUES
(1, 1, 1100, '1999-03-01', '1999-06-01'),
(2, 2, 1300, '1999-03-01', NULL),
(3, 3, 1500, '1999-03-01', '2000-05-31'),
(4, 3, 1800, '2000-06-01', NULL),
(5, 4, 1400, '1999-07-01', '2001-04-30'),
(6, 4, 1700, '2001-05-01', NULL),
(7, 5, 1800, '1999-10-01', '2001-09-30'),
(8, 5, 2000, '2001-10-01', NULL),
(9, 6, 1100, '1999-03-01', '2000-03-01'),
(10, 7, 2000, '1999-05-01', NULL),
(11, 8, 2100, '1999-08-01', NULL),
(13, 10, 2500, '2000-04-01', '2001-05-31'),
(14, 10, 3000, '2001-06-01', NULL);
-- --------------------------------------------------------
--
-- Struktura tabeli dla tabeli `placowki`
```

```
--
CREATE TABLE `placowki` (
`id_placowki` int(1) NOT NULL,
`nazwa` varchar(15) COLLATE utf8_polish_ci NOT NULL,
`miasto` varchar(15) COLLATE utf8_polish_ci NOT NULL,
`adres` varchar(20) COLLATE utf8_polish_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;
--
-- Zrzut danych tabeli `placowki`
--
INSERT INTO `placowki` (`id_placowki`, `nazwa`, `miasto`, `adres`)
VALUES
(1, 'H&M', 'Łódź', 'Sukiennicza 14'),
(2, 'Omega', 'Łódź', 'Piotrkowska 79'),
(3, 'Komputroniks', 'Łódź', 'Rojna 10'),
(5, 'Milka', 'Łódź', 'Sukiennicza 14'),
(6, 'C&A', 'Zgierz', 'Łódzka 15'),
(15, 'Lala', 'Sopot', 'Nowa 21'),
(16, 'Krokodyl', 'Szczecin', 'Wierzbowa 7');
-- --------------------------------------------------------
--
-- Struktura tabeli dla tabeli `pracownicy`
--
CREATE TABLE `pracownicy` (
`id_pracownika` int(4) NOT NULL,
`nazwisko` varchar(20) COLLATE utf8_polish_ci NOT NULL,
`imie` varchar(20) COLLATE utf8_polish_ci NOT NULL,
`pesel` char(11) COLLATE utf8_polish_ci DEFAULT NULL,
`plec` char(1) COLLATE utf8_polish_ci NOT NULL,
`data_urodzenia` date NOT NULL,
`data_zatrudnienia` date NOT NULL,
`zawod` varchar(15) COLLATE utf8_polish_ci NOT NULL,
`pensja` decimal(10,0) NOT NULL,
`dodatek` decimal(10,0) DEFAULT NULL,
`idplacowki` int(1) NOT NULL,
`idszefa` int(4) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;
--
-- Zrzut danych tabeli `pracownicy`
--
INSERT INTO `pracownicy` (`id_pracownika`, `nazwisko`, `imie`, `pesel`,
`plec`, `data_urodzenia`, `data_zatrudnienia`, `zawod`, `pensja`,
`dodatek`, `idplacowki`, `idszefa`) VALUES
(1, 'Smith', 'Jan', '', 'm', '1971-01-22', '2003-05-15', 'informatyk',
'3000', '0', 4, 0),
(2, 'Kowal', 'Małgorzata', '', 'k', '1969-02-12', '2000-02-01',
'sekretarka', '1700', '0', 2, 2),
(3, 'Anusz', 'Anna', '', 'k', '1974-08-22', '2003-05-15', 'prawnik',
'3000', '0', 1, 90),
```

```
(4, 'Chiński', 'Marek', '', 'm', '1968-11-14', '2000-02-01', 'księgowy',
'2200', '0', 2, 1),
(5, 'Mięta', 'Roman', '', 'm', '1963-03-17', '2000-05-07', 'grafik',
'2000', '0', 3, 2),
(6, 'Gates', 'Karol', '', 'm', '1967-05-18', '2002-01-14', 'akwizytor',
'1200', '0', 4, 2),
(7, 'Janasik', 'Anatol', '', 'm', '1968-01-21', '2001-10-15',
'menadżer', '3000', '0', 2, 2),
(8, 'Milion', 'Ewa', '', 'k', '1963-02-14', '2000-05-07', 'kasjer',
'2800', '0', 6, 2),
(9, 'Bosman', 'Bogdan', '', 'm', '1980-01-01', '2006-04-01', 'portier',
'1300', '0', 5, 10),
(10, 'Kildare', 'Dariusz', '', 'm', '1968-02-17', '2001-10-15',
'elektronik', '1400', '0', 5, 2),
(11, 'Holmes', 'Jakub', '', 'm', '1968-06-30', '2001-10-15',
'sprzedawca', '1700', '0', 2, 2),
(12, 'Kmicic', 'Jan', '', 'm', '1976-10-08', '2005-02-01', 'informatyk',
'3000', '0', 1, 1),
(13, 'Potocki', 'Grzegorz', '', 'm', '1963-12-22', '2000-05-07',
'akwizytor', '1200', '0', 3, 10),
(14, 'Gigant', 'Grzegorz', '', 'm', '1976-11-16', '2005-02-01',
'elektronik', '1850', '0', 1, 10),
(15, 'Inny', 'Stefan', '', 'm', '1966-07-06', '2002-01-14', 'grafik',
'2200', '0', 1, 4),
(16, 'Rzecki', 'Bogdan', '', 'm', '1968-01-22', '2001-10-15',
'informatyk', '3000', '0', 1, 1),
(17, 'Boss', 'Bohdan', '', 'm', '1966-03-26', '2002-01-14', 'księgowy',
'1900', '0', 1, 10),
(18, 'Gates', 'Anna', '', 'k', '1970-03-07', '2004-09-03', 'menadżer',
'3000', '0', 2, 2),
(19, 'Potocki', 'Stanisław', '', 'm', '1965-01-09', '2007-01-08',
'portier', '1300', '0', 2, 10),
(20, 'Mięta', 'Karol', '', 'm', '1965-04-05', '2006-01-19', 'kasjer',
'2500', '0', 2, 0),
(21, 'Bosman', 'Janina', '', 'k', '1967-04-05', '2002-01-14',
'sekretarka', '2000', '0', 2, 0),
(22, 'Smith', 'Jan', '', 'm', '1965-04-25', '2006-01-19', 'sprzedawca',
'1400', '0', 2, 2),
(23, 'Różny', 'Maria', '', 'k', '1965-05-22', '2005-08-20',
'sekretarka', '2000', '0', 3, 4),
(24, 'Potocki', 'Anna', '', 'k', '1965-01-10', '2007-01-08', 'kasjer',
'2500', '0', 4, 0),
(25, 'Potocki', 'Karol', '', 'm', '1965-01-13', '2007-01-08',
'informatyk', '2500', '0', 4, 1),
(26, 'Milion', 'Roman', '', 'm', '1966-02-10', '2002-01-14', 'portier',
'1200', '0', 4, 2),
(27, 'Mięta', 'Karol', '', 'm', '1965-05-16', '2005-08-20', 'menadżer',
'3000', '0', 5, 0),
(28, 'Mięta', 'Anatol', '', 'm', '1970-05-07', '2004-09-03', 'księgowy',
'1900', '0', 5, 0),
```

```
(29, 'Kowalska', 'Ewa', '', 'k', '1981-12-01', '2006-04-01',
'informatyk', '3000', '0', 5, 1),
(30, 'Kmicic', 'Bogdan', '', 'm', '1964-06-13', '2000-05-07', 'grafik',
'2000', '0', 5, 1),
(31, 'Kopeć', 'Anna', '', 'k', '1970-10-15', '2003-05-15', 'elektronik',
'1400', '0', 5, 90),
(32, 'Janasik', 'Iza', '', 'k', '1969-02-24', '2000-02-04',
'elektronik', '1850', '0', 5, 10),
(33, 'Inny', 'Jan', '', 'm', '1968-02-28', '2001-10-15', 'akwizytor',
'1200', '0', 6, 10),
(34, 'Holmes', 'Adam', '', 'm', '1970-12-25', '2003-05-15', 'akwizytor',
'1200', '0', 4, 2),
(35, 'Gigant', 'Grzegorz', '', 'm', '1968-02-25', '2001-10-15',
'księgowy', '1400', '0', 3, 4),
(36, 'Gates', 'Stefan', '', 'm', '1965-11-18', '2005-08-20', 'menadżer',
'3000', '0', 2, 0),
(37, 'Gates', 'Stanisław', '', 'm', '1968-05-01', '2001-10-15',
'portier', '1200', '0', 1, 4),
(38, 'Chiński', 'Bohdan', '', 'm', '1969-04-05', '2000-02-01', 'kasjer',
'2800', '0', 6, 2),
(39, 'Boss', 'Anna', '', 'k', '1965-04-25', '2006-01-19', 'sekretarka',
'2000', '0', 2, 0),
(40, 'Bosman', 'Stanisław', '', 'm', '1969-09-17', '2004-09-03',
'sprzedawca', '1700', '0', 2, 10),
(41, 'Bosman', 'Karolina', '', 'k', '1970-09-29', '2004-09-03',
'sekretarka', '2000', '0', 2, 1),
(42, 'Aanusz', 'Jan', '', 'm', '1965-02-02', '2007-01-08', 'kasjer',
'2500', '0', 2, 0),
(43, 'Kiel', 'Barbara', '', 'k', '1970-05-09', '2004-09-03',
'informatyk', '3000', '0', 4, 1),
(44, 'Gagatek', 'Zenon', '', 'm', '1967-03-12', '2002-01-14', 'portier',
'1300', '0', 4, 0),
(45, 'Borowik', 'Michał', '', 'm', '1970-10-12', '2003-05-15', 'prezes',
'3000', '0', 4, 2),
(46, 'Soski', 'Robert', '', 'm', '1968-07-26', '2000-02-04', 'księgowy',
'1900', '0', 4, 0),
(47, 'Misiak', 'Jacek', '', 'm', '1967-08-30', '2002-01-14',
'informatyk', '2500', '0', 4, 0),
(48, 'Zysk', 'Maryla', '', 'k', '1965-11-04', '2005-08-20', 'grafik',
'2200', '0', 4, 0),
(49, 'Robak', 'Jacek', '', 'm', '1967-02-07', '2002-01-14',
'elektronik', '1850', '0', 4, 10),
(50, 'Soplica', 'Jacek', '', 'm', '1963-01-01', '2005-02-01',
'elektronik', '1400', '0', 4, 10),
(51, 'Sawyer', 'Tomek', '', 'm', '1964-09-13', '2000-05-07', 'ekonom',
'1200', '0', 4, 2),
(52, 'Boruta', 'Michał', '', 'm', '1964-12-21', '2007-01-08',
'magazynier', '1600', '0', 4, 10),
(53, 'Jasiak', 'Monika', '', 'k', '1968-08-03', '2000-02-04',
'sprzedawca', '1600', '0', 4, 10),
```

```
(54, 'Zysk', 'Anna', '', 'k', '1966-07-03', '2002-01-14', 'sekretarka',
'1400', '0', 4, 1),
(55, 'Soplica', 'Jacek', '', 'm', '1965-05-19', '2005-08-20', 'kasjer',
'2500', '0', 6, 0),
(56, 'Sawyer', 'Jacek', '', 'm', '1961-03-22', '2005-02-01',
'informatyk', '3000', '0', 6, 1),
(57, 'Robak', 'Marta', '', 'k', '1965-07-16', '2005-08-20', 'portier',
'1300', '0', 6, 10),
(58, 'Kowalski', 'Michał', '', 'm', '1963-09-11', '2000-05-07',
'menadżer', '3000', '0', 6, 0),
(59, 'Jasiak', 'Morus', '', 'm', '1968-07-15', '2000-02-01', 'księgowy',
'1900', '0', 6, 1),
(60, 'Grześkowiak', 'Szymon', '', 'm', '1969-05-14', '2004-09-03',
'ochroniarz', '1700', '0', 6, 2),
(61, 'Boruta', 'Tomek', '', 'm', '1965-12-25', '2005-08-20',
'ochroniarz', '1400', '0', 6, 10),
(62, 'Szumowski', 'Andrzej', '', 'm', '1955-11-26', '2001-02-01',
'akwizytor', '1200', '0', 6, 10),
(63, 'Karpowicz', 'Dorota', '', 'k', '1959-03-27', '2001-02-01',
'rzecznik', '3000', '0', 6, 90),
(64, 'Kilarski', 'Ewa', '', 'k', '1965-12-28', '2005-08-20', 'redaktor',
'2000', '0', 6, 0),
(65, 'Waz', 'Fryderyk', '', 'm', '1963-02-26', '2000-05-07', 'księgowy',
'2200', '0', 6, 1),
(66, 'Kostrzewski', 'Grzegorz', '', 'm', '1960-10-30', '2004-02-01',
'elektryk', '1200', '0', 4, 10),
(67, 'Gigant', 'Grzegorz', '', 'm', '1970-10-05', '2004-09-03',
'konserwator', '1600', '0', 4, 10),
(68, 'Inny', 'Stefan', '', 'm', '1963-07-20', '2000-05-07', 'operator',
'1000', '0', 4, 2),
(69, 'Różny', 'Stanisław', '', 'm', '1987-10-06', '2006-04-01',
'ochroniarz', '1500', '0', 3, 10),
(70, 'Boss', 'Bohdan', '', 'm', '1975-11-04', '2005-02-01',
'informatyk', '2500', '0', 3, 1),
(71, 'Gates', 'Anna', '', 'k', '1967-02-07', '2002-01-14', 'grafik',
'2000', '0', 3, 1),
(72, 'Potocki', 'Stanisław', '', 'm', '1963-02-01', '2000-05-07',
'kasjer', '2800', '0', 3, 2),
(73, 'Mięta', 'Karol', '', 'm', '1964-11-09', '2007-01-08', 'księgowy',
'1900', '0', 3, 0),
(74, 'Bosman', 'Jan', '', 'm', '1964-04-30', '2000-05-07',
'konserwator', '1000', '0', 3, 10),
(75, 'Inny', 'Anna', '', 'k', '1968-02-28', '2001-10-15', 'kierowca',
'1800', '0', 1, 2),
(76, 'Holmes', 'Barbara', '', 'k', '1970-12-25', '2003-05-15',
'sprzedawca', '1500', '0', 1, 1),
(77, 'Gigant', 'Bożena', '', 'k', '1968-02-25', '2001-10-15',
'sekretarka', '2000', '0', 1, 4),
(78, 'Gates', 'Grzegorz', '', 'm', '1979-09-29', '2006-04-01',
'techniczny', '1000', '0', 1, 10),
```

```
(79, 'Gatek', 'Jan', '', 'm', '1967-09-18', '2002-01-14', 'kasjer',
'2500', '0', 1, 2),
(80, 'Chrust', 'Jan', '', 'm', '1969-05-14', '2004-09-03', 'kasjer',
'2500', '0', 1, 2),
(81, 'Bastek', 'Janusz', '', 'm', '1962-05-01', '2005-02-01',
'informatyk', '3000', '0', 1, 1),
(82, 'Boran', 'Karol', '', 'm', '1962-11-18', '2005-02-01',
'porządkowy', '1000', '0', 1, 10),
(83, 'Baran', 'Michał', '', 'm', '1960-04-25', '2001-02-02',
'porządkowy', '1000', '0', 1, 90),
(84, 'Anusz', 'Stanisław', '', 'm', '1965-02-02', '2007-01-08',
'menadżer', '3000', '0', 1, 0),
(85, 'Kiel', 'Stanisław', '', 'm', '1971-05-10', '2003-05-15',
'menadżer', '3000', '0', 2, 1),
(86, 'Gagatek', 'Stefan', '', 'm', '1967-03-11', '2002-01-14',
'księgowy', '1900', '0', 3, 1),
(87, 'Borowik', 'Zenon', '', 'm', '1970-10-12', '2003-05-15',
'techniczny', '1300', '0', 2, 2),
(88, 'Kowalski', 'Marzena', '', 'k', '1963-02-01', '2005-02-01',
'kierowca', '1800', '0', 2, 10),
(89, 'Kaczan', 'Ewa', '', 'k', '1964-11-09', '2007-01-08', 'elektryk',
'1200', '0', 2, 10),
(90, 'Wiss', 'Jan', '', 'm', '1964-04-30', '2000-05-07', 'malarz',
'1200', '0', 3, 10),
(91, 'Wilk', 'Sabrina', '', 'k', '1957-12-13', '2001-02-01', 'pakowacz',
'1000', '0', 4, 10),
(92, 'Paczka', 'Magdalena', '', 'k', '1973-03-16', '2003-05-15',
'informatyk', '3000', '0', 3, 10),
(93, 'Gałązka', 'Katarzyna', '', 'k', '1961-03-02', '2004-02-01',
'prezes', '4000', '0', 3, 0),
(94, 'Karpowicz', 'Mateusz', '', 'm', '1955-05-05', '2001-02-01',
'konserwator', '1600', '0', 3, 10),
(95, 'Mater', 'Krzysztof', '', 'm', '1960-09-07', '2001-02-02',
'pakowacz', '1000', '0', 3, 10),
(96, 'Kulak', 'Józef', '', 'm', '1954-07-21', '2001-02-02',
'porządkowy', '1200', '0', 3, 90),
(97, 'Busz', 'Stanisław', '', 'm', '1965-05-17', '2005-08-20',
'pakowacz', '1000', '0', 3, 2),
(98, 'Karpowicz', 'Anna', '', 'k', '1973-09-11', '2003-05-15',
'elektryk', '1300', '0', 3, 1),
(99, 'Kilarski', 'Barbara', '', 'k', '1968-07-15', '2000-02-01',
'informatyk', '2500', '0', 3, 1),
(100, 'Waz', 'Bohdan', '', 'm', '1959-05-14', '2001-02-01', 'kierowca',
'1800', '0', 4, 10),
(101, 'Kurka', 'Grzegorz', '', 'm', '1965-11-21', '2005-08-20',
'asystent', '1200', '0', 4, 2),
(102, 'Jarek', 'Jan', '', 'm', '1955-01-27', '2001-02-01', 'techniczny',
'1000', '0', 4, 10),
(103, 'Wanik', 'Kamil', '', 'm', '1960-03-27', '2001-02-01',
'informatyk', '3000', '0', 5, 1),
```

```
(104, 'Lisowski', 'Julian', '', 'm', '1964-12-28', '2007-01-08',
'prezes', '3000', '0', 6, 0),
(105, 'Łapa', 'Henryk', '', 'm', '1963-02-26', '2000-05-07', 'księgowy',
'1900', '0', 3, 1),
(106, 'Miturski', 'Mateusz', '', 'm', '1960-10-30', '2004-02-01',
'informatyk', '2500', '0', 3, 2),
(107, 'Lysiak', 'Helena', '', 'k', '1970-10-05', '2004-09-03',
'sekretarka', '1500', '0', 3, 1),
(108, 'Wnuk', 'Jan', '', 'm', '1963-07-20', '2000-05-07', 'menadżer',
'3000', '0', 2, 0),
(109, 'Mazurowski', 'Janusz', '', 'm', '1977-10-06', '2005-02-01',
'menadżer', '3000', '0', 2, 0),
(110, 'Bodek', 'Karol', '', 'm', '1965-11-05', '2005-08-20',
'informatyk', '3000', '0', 2, 1),
(111, 'Magierowicz', 'Stanisław', '', 'm', '1963-12-01', '2000-05-07',
'pakowacz', '1000', '0', 2, 10),
(112, 'Myślicki', 'Stanisław', '', 'm', '1962-11-08', '2005-02-01',
'porządkowy', '1000', '0', 2, 10),
(113, 'Rengifo', 'Stefan', '', 'm', '1984-04-30', '2006-04-01',
'grafik', '1700', '0', 1, 1),
(114, 'Golanczyk', 'Karol', '', 'm', '1959-03-27', '2001-02-01',
'techniczny', '1300', '0', 1, 10),
(115, 'Ludziejewski', 'Kamil', '', 'm', '1965-12-28', '2005-08-20',
'księgowy', '2200', '0', 2, 0),
(116, 'Iwaszko', 'Katarzyna', '', 'k', '1963-02-26', '2000-05-07',
'konserwator', '1600', '0', 4, 10),
(117, 'Muraszkowski', 'Michał', '', 'm', '1960-10-30', '2004-02-01',
'kierowca', '1800', '0', 6, 10),
(118, 'Grzegorczyk', 'Kazimierz', '', 'm', '1970-10-05', '2004-09-03',
'sprzedawca', '1500', '0', 4, 10),
(119, 'Abak', 'Karolina', '', 'k', '1957-10-06', '2001-02-01', 'grafik',
'2200', '0', 3, 4),
(120, 'Bosman', 'Stefan', '', 'm', '1970-01-01', '2004-09-03',
'portier', '1300', '0', 3, 10),
(121, 'Boss', 'Stanisław', '', 'm', '1966-03-26', '2002-01-14',
'księgowy', '1400', '0', 5, 1),
(122, 'Kildare', 'Jan', '', 'm', '1968-02-27', '2001-10-15',
'elektronik', '1400', '0', 5, 4),
(123, 'Gigant', 'Jakub', '', 'm', '1966-11-16', '2002-01-14',
'elektronik', '1400', '0', 5, 2),
(124, 'Potocki', 'Grzegorz', '', 'm', '1963-12-22', '2000-05-07',
'akwizytor', '1200', '0', 5, 10),
(125, 'Holmes', 'Grzegorz', '', 'm', '1968-06-30', '2001-10-15',
'sprzedawca', '1500', '0', 5, 2),
(126, 'Kmicic', 'Dariusz', '', 'm', '1986-10-08', '2006-04-01',
'informatyk', '2500', '0', 5, 1),
(127, 'Rogal', 'Bohdan', '', 'm', '1968-01-22', '2001-10-15',
'informatyk', '3000', '0', 6, 1),
(128, 'Inny', 'Bogdan', '', 'm', '1966-07-06', '2002-01-14', 'grafik',
'2000', '0', 6, 1),
```

```
(129, 'Adaszynska', 'Barbara', '', 'k', '1973-01-03', '2003-05-15',
'sekretarka', '2000', '0', 6, 2),
(130, 'Cebula', 'Sebastian', '', 'm', '1975-11-02', '2005-02-01',
'informatyk', '3000', '0', 6, 1);
--
-- Indeksy dla zrzutów tabel
--
--
-- Indexes for table `pensje`
--
ALTER TABLE `pensje`
ADD PRIMARY KEY (`id`),
ADD KEY `id_os` (`id_os`);
--
-- Indexes for table `placowki`
--
ALTER TABLE `placowki`
ADD PRIMARY KEY (`id_placowki`);
--
-- Indexes for table `pracownicy`
--
ALTER TABLE `pracownicy`
ADD PRIMARY KEY (`id_pracownika`),
ADD UNIQUE KEY `id_pracownika` (`id_pracownika`),
ADD KEY `id_pracownika_2` (`id_pracownika`),
ADD KEY `idplacowki` (`idplacowki`);
--
-- AUTO_INCREMENT for dumped tables
--
--
-- AUTO_INCREMENT dla tabeli `placowki`
--
ALTER TABLE `placowki`
MODIFY `id_placowki` int(1) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=17;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;


COMMIT;
```

ZAŁĄCZNIK 3

```
-- phpMyAdmin SQL Dump
-- version 4.5.1
-- http://www.phpmyadmin.net
--
-- Host: 127.0.0.1
-- Czas generowania: 29 Gru 2017, 16:16
-- Wersja serwera: 10.1.13-MariaDB
-- Wersja PHP: 5.6.23
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
--
-- Baza danych: `biblioteka`
--
-- --------------------------------------------------------
--
-- Struktura tabeli dla tabeli `autorzy`
--
CREATE TABLE `autorzy` (
`IDAutor` int(10) UNSIGNED NOT NULL,
`Imie` text,
`Nazwisko` text
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
--
-- Zrzut danych tabeli `autorzy`
--
INSERT INTO `autorzy` (`IDAutor`, `Imie`, `Nazwisko`) VALUES
(3, 'Jo', 'Nesbo'),
(4, 'Charlotte', 'Link'),
(5, 'Katarzyna', 'Bonda'),
(8, 'Harlan', 'Coben');
-- --------------------------------------------------------
--
-- Struktura tabeli dla tabeli `egzemplarze`
--
CREATE TABLE `egzemplarze` (
`IDEgzemplarz` int(10) UNSIGNED NOT NULL,
`KsiazkaID` int(10) UNSIGNED NOT NULL,
`DoWypozyczenia` tinyint(1) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
--
-- Zrzut danych tabeli `egzemplarze`
--
INSERT INTO `egzemplarze` (`IDEgzemplarz`, `KsiazkaID`,
`DoWypozyczenia`) VALUES
```

```
(1, 1, 1),
(2, 1, 0),
(3, 1, 1),
(4, 2, 1),
(5, 2, 1),
(6, 3, 1),
(7, 4, 1),
(8, 4, 1),
(9, 4, 0),
(10, 5, 1),
(11, 5, 0),
(12, 2, 1),
(13, 6, 1),
(14, 7, 0);
-- --------------------------------------------------------
--
-- Struktura tabeli dla tabeli `ksiazki`
--
CREATE TABLE `ksiazki` (
`IDKsiazki` int(10) UNSIGNED NOT NULL,
`AutorID` int(10) UNSIGNED NOT NULL,
`Tytul` mediumtext COLLATE utf8_polish_ci,
`Wydawnictwo` mediumtext COLLATE utf8_polish_ci,
`RokWydania` int(10) UNSIGNED DEFAULT NULL,
`Cena_za_wyp` float NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_polish_ci;
--
-- Zrzut danych tabeli `ksiazki`
--
INSERT INTO `ksiazki` (`IDKsiazki`, `AutorID`, `Tytul`, `Wydawnictwo`,
`RokWydania`, `Cena_za_wyp`) VALUES
(6, 3, 'Policja', 'Agra', 2010, 7),
(7, 3, 'Pancerne serce', 'Agra', 2012, 7),
(8, 5, 'Lampiony', 'Ossolineum', 2016, 8),
(9, 3, 'Pragnienie', 'Dolnoślaskie', 2017, 10),
(10, 3, 'Trzeci klucz', 'Dolnośląskie', 2014, 6),
(11, 5, 'Okularnik', 'Muza S.A.', 2016, 8.5),
(12, 5, 'Sprawa Niny Frank', 'Muza S.A.', 2015, 7.5),
(13, 5, 'Florystka', 'Muza S.A.', 2014, 5),
(14, 4, 'Drugie dziecko', 'Sonia Draga', 2012, 5),
(15, 4, 'Obserwator', 'Sonia Draga', 2014, 5),
(16, 4, 'Drugie dziecko', 'Sonia Draga', 2015, 6.5),
(17, 4, 'Wielbiciel', 'Sonia Draga', 2016, 7),
(18, 8, 'Jedyna szansa', 'Albatros', 2016, 7),
(19, 8, 'Odnaleziony', 'Albatros', 2015, 6),
(20, 8, 'Schronienie', 'Albatros', 2016, 8),
(21, 8, 'Niewinny', 'Albatros', 2014, 5.5);
--
-- Indeksy dla zrzutów tabel
--
```

```
--
-- Indexes for table `autorzy`
--
ALTER TABLE `autorzy`
ADD PRIMARY KEY (`IDAutor`);
--
-- Indexes for table `egzemplarze`
--
ALTER TABLE `egzemplarze`
ADD PRIMARY KEY (`IDEgzemplarz`);
--
-- Indexes for table `ksiazki`
--
ALTER TABLE `ksiazki`
ADD PRIMARY KEY (`IDKsiazki`);
--
-- AUTO_INCREMENT for dumped tables
--
--
-- AUTO_INCREMENT dla tabeli `autorzy`
--
ALTER TABLE `autorzy`
MODIFY `IDAutor` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=9;
--
-- AUTO_INCREMENT dla tabeli `egzemplarze`
--
ALTER TABLE `egzemplarze`
MODIFY `IDEgzemplarz` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=15;
--
-- AUTO_INCREMENT dla tabeli `ksiazki`
--
ALTER TABLE `ksiazki`
MODIFY `IDKsiazki` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
AUTO_INCREMENT=22;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

WYKAZ PODSTAWOWYCH POJĘĆ W JĘZYKACH: POLSKIM, ANGIELSKIM I NIEMIECKIM

JĘZYK POLSKI	JĘZYK ANGIELSKI	JĘZYK NIEMIECKI
atrybut encji	entity attribute	die Entitätseigenschaft
autonumerowanie	AutoNumber	die automatische Nummerierung
baza danych	database	die Datenbank
bezpieczeństwo danych	data safety	die Datensicherheit
diagram ER	ER diagram	das ER-Diagramm
druga postać normalna	the second normal form	die zweite Normalform
encja	entity	die Entität
formularz	form	das Formular
funkcja agregująca	aggregate function	die Aggregatfunktion
identyfikator	ID card / CAC (Common Access Card)	der Bezeichner, der Identifikator, der Identifizierer
indeksowanie	indexing	die Indexierung
integralność danych	data integrity	die Datenintegrität
język SQL	SQL language	die Strukturierte Abfragesprache
kaskadowe aktualizowanie danych	cascade data update	die kaskadierende Aktualisierung, die Aktualisierungsweitergabe
kaskadowe usuwanie danych	cascade data deletion	die Löschweitergabe
klauzula	clause	die Klausel
klucz główny	main key	der Primärschlüssel
klucz kandydujący	candidate key	der Schlüsselkandidat / der Kandidatenschlüssel
klucz obcy	foreign key	der Fremdschlüssel
klucz podstawowy	primary key	der Primärschlüssel
klucz sztuczny	artificial key	der Surrogatschlüssel / der künstliche Schlüssel
klucz złożony	composite key	der zusammengesetzter Schlüssel
kolumna tabeli	table column	die Spalte
krotka	tuple	das Tupel
kwerenda	query	die Abfrage
kwerenda aktualizująca	updating query	die Aktualisierungsabfrage
kwerenda krzyżowa	cross-query	die Kreuztabellenabfrage
kwerenda wybierająca	select query	die Auswahlabfrage

JĘZYK POLSKI	JĘZYK ANGIELSKI	JĘZYK NIEMIECKI
literał	literal	das Literal, das Buchstabensymbol
makro	macro	der Makro
maska wprowadzania	input mask	die Eingabemaske
model ER	ER model	das ER-Modell
model fizyczny bazy danych	the physical model of the database	das physische Datenbankmodell
model hierarchiczny	hierarchical model	das hierarchische Datenbankmodell
model jednorodny	homogeneous model	die Flatfile-Datenbank
model logiczny bazy danych	logical model of the database	das logische Datenmodell
model relacyjny	relational model	das relationale Datenbankmodell
model sieciowy	network model	das Netzwerkdatenbankmodell
moduł	module	das Modul
normalizacja	normalization	die Normalisierung
operator	operator	der Operator
operator specjalny	special operator	der Sonderoperator
pakiet aplikacji serwerowych	server application package	das Web-App-Paket
pierwsza postać normalna	first normal form	die erste Normalform
podzapytanie	subquery	die Unterabfrage
pole obliczeniowe	calculated field	das berechnete Feld
pole tabeli	table field	das Tabellenfeld
połączenie krzyżowe	cross connection	die Kreuzverknüpfung
połączenie wewnętrzne	nternal connection	die innere Verknüpfung / der Inner-Join
połączenie zewnętrzne	external connection	die äußere Verknüpfung / der Outer-Join
reguła poprawności	correctness rule	die Gültigkeitsregel
relacja	relation	die Relation, die Verknüpfung
relacyjna baza danych	relational database	die relationale Datenbank
selekcja rekordów	selection of records	die Datensatzselektion
serwer bazy danych	database server	der Datenbankserver
spójność danych	consistency of data	die Datenkonsistenz
strategia projektowania	design strategy	die Entwurfstrategie
strukturalny język zapytań	the structured language of queries	die Strukturierte Abfragesprache
system bazy danych	database system	das Datenbanksystem
system zarządzania bazą danych	database management system	das Datenbankverwaltungssystem

JĘZYK POLSKI	JĘZYK ANGIELSKI	JĘZYK NIEMIECKI
tabela	table	die Tabelle
trwałość danych	durability of data	die Datenbeständigkeit
trzecia postać normalna	third normal form	die dritte Normalform
typ danych	data type	der Datentyp
wartość domyślna	the default value	der Defaultwert, der Standardwert
wiersz tabeli	table row	die Tabellenzeile
więzy integralności	integrity bonds	die referentielle Integrität
właściwości pola	field properties	die Feldeigenschaften
właściwości tabeli	table properties	die Tabelleneigenschaften
wyrażenie	expression	der Ausdruck
związek rekurencyjny	recursive relationship	die Rekursionsbeziehung

Wydawnictwa Szkolne i Pedagogiczne oświadczają, że podjęły starania mające na celu dotarcie do właścicieli i dysponentów praw autorskich zamieszczonych utworów. Wydawnictwa Szkolne i Pedagogiczne, przytaczając w celach dydaktycznych utwory lub fragmenty, postępują zgodnie z art. 27^1 ustawy o prawie autorskim. Jednocześnie Wydawnictwa Szkolne i Pedagogiczne oświadczają, że są jedynym podmiotem właściwym do kontaktu autorów tych utworów lub innych podmiotów uprawnionych w wypadkach, w których twórcy przysługuje prawo do wynagrodzenia.