



DO NOWEJ
PODSTAWY PROGRAMOWEJ

Część 2

Projektowanie i administrowanie
bazami danych

Kwalifikacja INF.03

Tworzenie i administrowanie
stronami i aplikacjami internetowymi
oraz bazami danych



Podręcznik do nauki zawodu
technik informatyk i technik programista

Jolanta Pokorska

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Joanna Zaręba

Projekt okładki: Jan Paluch

Fotografia na okładce została wykorzystana za zgodą Shutterstock.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

http://helion.pl/user/opinie?inf032_ebook

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-7089-0

Copyright © Helion 2019

- [Poleć książkę na Facebook.com](#)
- [Kup w wersji papierowej](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

| | |
|--|-----|
| Wstęp | 5 |
| Rozdział 1. Zasady projektowania relacyjnych baz danych | 7 |
| 1.1. Wprowadzenie do baz danych | 7 |
| 1.2. Modele baz danych | 8 |
| 1.3. Relacyjny model danych | 11 |
| 1.4. Projektowanie bazy danych | 18 |
| 1.5. Cechy relacyjnej bazy danych | 46 |
| 1.6. Pytania i zadania | 49 |
| Rozdział 2. Systemy zarządzania bazą danych | 51 |
| 2.1. Wprowadzenie | 51 |
| 2.2. Serwery bazodanowe | 54 |
| 2.3. Pytania i zadania | 68 |
| Rozdział 3. SQL — strukturalny język zapytań | 71 |
| 3.1. Wprowadzenie | 71 |
| 3.2. Standardy języka SQL | 71 |
| 3.3. Składnia języka SQL | 72 |
| 3.4. Język definiowania danych (DDL) | 77 |
| 3.5. Język manipulowania danymi (DML) | 92 |
| 3.6. Instrukcja SELECT | 96 |
| 3.7. Łączenie tabel | 108 |
| 3.8. Więzy integralności (MS SQL) | 114 |
| 3.9. Łączenie wyników zapytań | 119 |
| 3.10. Podzapytania | 121 |
| 3.11. Widoki (perspektywy) | 128 |
| 3.12. Indeksy | 130 |
| 3.13. Transakcje | 131 |
| 3.14. Współbieżność | 136 |
| 3.15. T-SQL | 143 |
| 3.16. Pytania i zadania | 156 |

| | | |
|---------------------|--|-----|
| Rozdział 4. | Administrowanie serwerami baz danych | 165 |
| 4.1. | Wprowadzenie | 165 |
| 4.2. | MS SQL Server | 166 |
| 4.3. | Prawa dostępu do serwera | 172 |
| 4.4. | Kopie bezpieczeństwa | 178 |
| 4.5. | Import i eksport danych | 182 |
| 4.6. | MySQL | 185 |
| 4.7. | Prawa dostępu do serwera | 188 |
| 4.8. | Kopie bezpieczeństwa | 193 |
| 4.9. | Import i eksport danych | 196 |
| 4.10. | Optymalizacja wydajności SZBD | 198 |
| 4.11. | Pytania i zadania | 203 |
| Bibliografia | | 205 |
| Skorowidz | | 206 |

Wstęp

Branża związana z technologią informacyjną nie przestaje się rozwijać. Z roku na rok liczba danych przetwarzanych w sieci zwiększa się prawie o 50%. Dysponowanie wiedzą i praktycznymi umiejętnościami z zakresu projektowania i tworzenia baz danych, przetwarzania dużych ilości danych oraz administrowania bazami danych otwiera drogę do najbardziej poszukiwanego obecnie zawodu badacza danych (*data science*).

Zdobycie zawodu technik informatyk lub technik programista to nabycie poszukiwanych na rynku pracy kompetencji cyfrowych i możliwość szybkiego znalezienia bardzo atrakcyjnej pracy.

Podręcznik *Kwalifikacja INF.03. Tworzenie i administrowanie stronami i aplikacjami internetowymi oraz bazami danych. Część 2. Projektowanie i administrowanie bazami danych. Podręcznik do nauki zawodu technik informatyk i technik programista* zawiera treści przeznaczone dla osób kształcących się w zawodach technik informatyk i technik programista. Podręcznik został opracowany zgodnie z nową podstawą programową, obowiązującą od 1 września 2019 r. i obejmuje materiał kwalifikacji *INF.03. Tworzenie i administrowanie stronami i aplikacjami internetowymi oraz bazami danych* w zakresie projektowania i administrowania bazami danych.

W publikacji zostały omówione zagadnienia teoretyczne z zakresu projektowania, tworzenia i administrowania bazami danych. Zamieszczono też liczne ćwiczenia, które ułatwią nabycie umiejętności praktycznych, a proponowane rozwiązania pomogą w ugruntowaniu zdobytej wiedzy.

Podręcznik składa się z czterech rozdziałów. Ich budowa pozwala na realizację treści programowych w sposób wybrany przez nauczyciela oraz umożliwia samodzielne rozwijanie umiejętności przez uczniów.

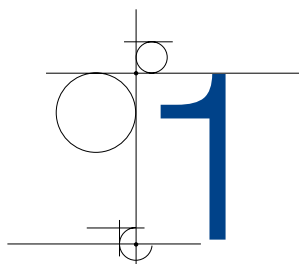
Rozdział 1., „Zasady projektowania relacyjnych baz danych”, zawiera omówienie podstawowych zagadnień związanych z relacyjnymi bazami danych. Dotyczy efektów łączących się z umiejętnością posługiwania się podstawowymi pojęciami z zakresu relacyjnych baz danych oraz stosowania relacyjnych baz danych. Efektami tymi są: identyfikowanie podstawowych pojęć dotyczących relacyjnych baz danych, identyfikowanie elementów bazy danych, stosowanie zasad projektowania baz danych, tworzenie graficznych schematów baz danych, projektowanie tabel baz danych, identyfikowanie i stosowanie zasad normalizacji tabel, definiowanie związków między tabelami bazy danych, identyfikowanie właściwości relacyjnych baz danych.

Rozdział 2., „Systemy zarządzania bazą danych”, zawiera omówienie zagadnień związanych z architekturą systemów baz danych oraz z instalowaniem i konfigurowaniem systemów bazodanowych i systemów zarządzania bazą danych. Dotyczy efektów związanych z instalowaniem systemów baz danych, konfigurowaniem serwerów baz danych,

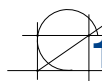
wykorzystaniem narzędzi graficznych do tworzenia baz danych oraz do zarządzania bazami. Efektami tymi są: identyfikowanie różnych systemów zarządzania bazami danych, identyfikowanie funkcji systemów zarządzania bazami danych, identyfikowanie serwerów baz danych, instalowanie systemów baz danych, zarządzanie bazami danych, stosowanie narzędzi graficznych do tworzenia baz danych.

Rozdział 3., „SQL — strukturalny język zapytań”, zawiera omówienie zagadnień dotyczących tworzenia bazy danych, tworzenia obiektów bazy danych oraz zarządzania bazą danych w języku SQL. Dotyczy efektów związanych z korzystaniem z funkcji strukturalnego języka zapytań oraz z posługiwaniem się strukturalnym językiem zapytań do tworzenia tabel i zapytań oraz modyfikowania i rozbudowy struktury bazy danych. Efektami tymi są: identyfikowanie i stosowanie składni strukturalnego języka zapytań, stosowanie funkcji strukturalnego języka zapytań, stosowanie instrukcji strukturalnego języka zapytań dotyczących tworzenia struktury bazy danych, modyfikowania obiektów oraz struktury bazy danych, rozbudowywanie struktury bazy danych, tworzenie zapytań i podzapytań oraz tworzenia połączeń między tabelami, identyfikowanie i stosowanie transakcji przy użyciu strukturalnego języka zapytań, stosowanie instrukcji języka T-SQL.

Rozdział 4., „Administrowanie serwerami baz danych”, zawiera omówienie zagadnień dotyczących administrowania serwerami bazodanowymi. Odnosi się do efektów związanych z określaniem uprawnień poszczególnych użytkowników, określaniem zabezpieczeń dla użytkowników, dobieraniem sposobów ustawienia zabezpieczeń dostępu do danych, zarządzaniem kopiami zapasowymi baz danych i ich odzyskiwaniem, kontrolowaniem spójności bazy danych, dokonywaniem naprawy bazy danych. Efektami tymi są: zarządzanie bazą danych, kontrolowanie spójności bazy danych, identyfikowanie i dobieranie sposobów ustawienia zabezpieczeń dostępu do danych, zarządzanie bezpieczeństwem bazy danych, identyfikowanie uprawnień użytkowników bazy danych, zarządzanie kopiami zapasowymi baz danych, zarządzanie odzyskiwaniem danych.



Zasady projektowania relacyjnych baz danych



1.1. Wprowadzenie do baz danych

Tradycyjne dane przechowywane są w postaci dokumentów papierowych. Dokumenty takie zawierają opisy przechowywanych obiektów. Obiektami mogą być na przykład samochody, książki lub osoby. Opisywane są także związki zachodzące między obiektami — kto jest właścicielem samochodu, kto wypożyczył książkę, gdzie pracuje dana osoba. Dokumenty opisujące obiekty i związki zachodzące między nimi są gromadzone w postaci kartotek, katalogów lub archiwów i są przechowywane w kopertach, teczkach albo segregatorach. Dostęp do tak opracowanych dokumentów jest trudny i zajmuje dużo czasu. Aby przyspieszyć wyszukiwanie danych, sporządza się różne spisy, wyciągi i katalogi.

Dobrym rozwiązaniem tych problemów jest przeniesienie takich dokumentów do komputera. Po zapisaniu danych w pamięci komputera można obsługiwać tak utworzoną bazę danych, korzystając z dostępnych narzędzi.

Zalety korzystania z komputerowych baz danych to:

- szybkie wyszukiwanie informacji,
- łatwe wykonywanie obliczeń,
- możliwość przechowywania dużej ilości danych na małej powierzchni,
- szybkie porządkowanie danych.

WSKAZÓWKA

Baza danych to uporządkowany zbiór danych z określonej dziedziny tematycznej zorganizowany w sposób ułatwiający do nich dostęp.

WSKAZÓWKA

System zarządzania bazą danych to program zarządzający danymi w bazie i umożliwiający ich przetwarzanie.

WSKAZÓWKA

System bazy danych to baza danych i system zarządzania bazą danych.

U podstaw konstruowania bazy danych leży założenie, że użytkownik, dla którego ta baza jest przeznaczona, nie musi być specjalistą z dziedziny baz danych, może w ogóle ich nie znać. Mimo to powinien bez problemów radzić sobie z obsługą zaprojektowanej bazy danych. Aby to było możliwe, podstawą tworzonej bazy musi być solidny projekt określający potrzeby użytkownika dotyczące gromadzenia, przechowywania i przetwarzania danych oraz definiujący czynności składające się na obsługę bazy danych.

1.2. Modele baz danych

Aby przechowywać dane na komputerze, konieczne jest określenie formy ich przechowywania. Na potrzeby baz danych zostały zdefiniowane klasyczne techniki organizowania informacji, zwane modelami baz danych.

Model danych to abstrakcyjny opis sposobu przedstawiania i wykorzystania danych.

Na model danych składają się:

- *struktura* — opis sposobu przedstawiania obiektów (encji) modelowanego wycinka świata oraz ich związków,
- *ograniczenia* — reguły kontrolujące spójność i poprawność danych,
- *operacje* — zbiór działań, które umożliwiają dostęp do struktur.

Głównymi modelami baz danych są:

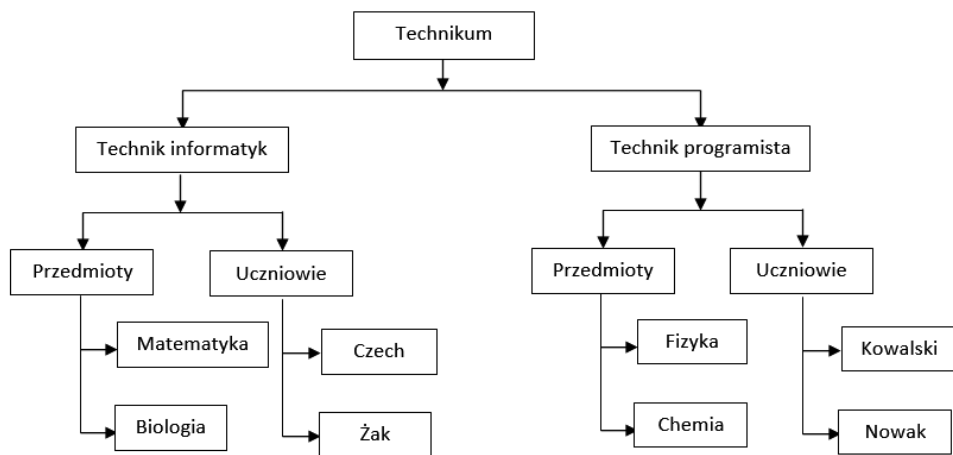
- model hierarchiczny,
- model sieciowy,
- model obiektowy,
- model relacyjny.

1.2.1. Model hierarchiczny

W modelu hierarchicznym przechowywane dane są zorganizowane w postaci odwróconego drzewa (rysunek 1.1). Każdy obiekt (z wyjątkiem obiektu podstawowego) jest połączony z jednym obiektem nadrzędnym.

Tak zbudowana baza danych umożliwia proste wyszukiwanie danych. Rozpoczyna się ono od obiektu podstawowego i poprzez rozgałęzienia oraz kolejne obiekty dochodzi się do obiektu szukanego. Informacja jest zawarta w kolejnych dokumentach oraz w strukturze drzewa (podobnej do drzewa folderów na dysku komputera).

Hierarchiczna baza danych zakłada podstawowe warunki integralności danych.

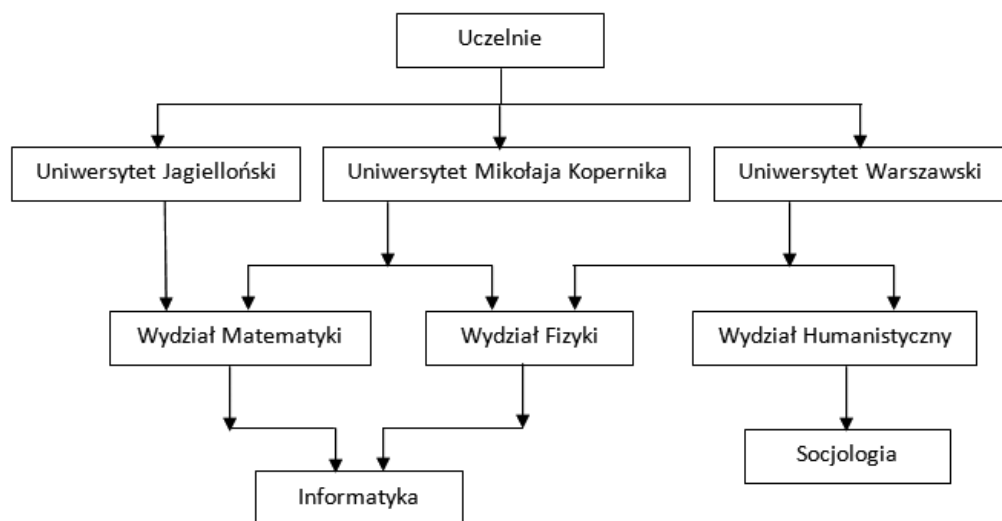


Rysunek 1.1. Model hierarchiczny

Każdy dokument (z wyjątkiem głównego — na górze) jest połączony z dokumentem nadrzędnym. Dokument podrzędny może zostać utworzony dopiero, gdy zostanie powiązany z dokumentem nadrzędnym. Kiedy zostanie usunięty dokument nadrzędny, automatycznie zostaną usunięte wszystkie dokumenty podrzędne.

1.2.2. Model sieciowy

W modelu sieciowym połączenia między dokumentami tworzą sieć (rysunek 1.2). Jest to zmodyfikowana wersja modelu hierarchicznego, w której dozwolone są połączenia na tym samym poziomie drzewa danych. Informacja jest zawarta w dokumentach oraz w przebiegu połączeń sieci.

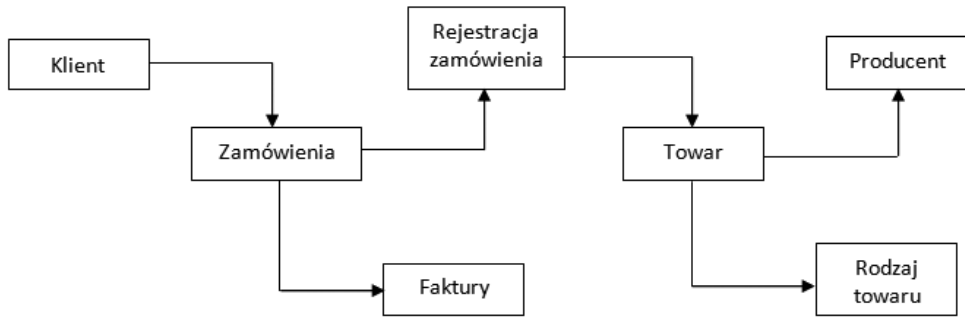
**Rysunek 1.2.** Model sieciowy

1.2.3. Model obiektowy

Model obiektowy łączy cechy programów komputerowych tworzonych w językach programowania obiektowego z cechami aplikacji bazodanowych. Dane są udostępniane w postaci obiektów. Ich stan i zachowanie są opisywane za pomocą narzędzi dostępnych w programowaniu obiektowym (własności, metody, klasy obiektów). Do przechowywania danych stosowane są obiekty. Obiekty obsługiwane za pomocą tych samych metod i własności są instancjami tej samej klasy.

1.2.4. Model relacyjny

Model relacyjny baz danych został oparty na matematycznym modelu organizacji danych i pojęciu relacji. W tym modelu dane są przedstawiane w postaci relacji reprezentowanych przez tabele. Relacje (tabele) składają się z rekordów (wierszy) o takiej samej strukturze. Tworzą one między sobą powiązania zwane relacjami (rysunek 1.3). Relacje (tabele) grupuje się w schematy bazy danych.



Rysunek 1.3. Model relacyjny

Ze względu na funkcjonalność model relacyjny jest najczęściej wykorzystywany przy projektowaniu baz danych.



1.3. Relacyjny model danych

1.3.1. Model relacyjny według E.F. Codda

Twórcą teorii relacyjnych baz danych jest Edgar Frank Codd. W 1970 roku opublikował on pracę pt. „Relacyjny model danych dla dużych banków danych”, która wprowadzała główne założenia dotyczące modelu relacyjnego baz danych. Później zostały uszczegółowione terminy *algebra relacji* i *rachunek relacyjny*.

Według podanych założeń określenie *relacyjna baza danych* oznacza bazę zbudowaną z relacji. Podstawowy obiekt takiej bazy danych, tabela, jest reprezentacją relacji w ujęciu pojęć matematycznych. Ale terminy *relacja* i *tabela* nie są jednoznaczne, ponieważ jedna relacja może być odwzorowana za pomocą wielu różnych tabel. Nie używa się również pojęć związanych z tabelą: *kolumna* i *wiersz*, lecz *atribut* i *krotka*.

W modelu relacyjnym przyjmuje się, że:

- kolejność wierszy i kolumn w tabelach jest nieistotna,
- wiersze zawierające takie same dane są identyczne.

Każda tabela składa się z pewnej liczby wierszy i kolumn. Na przecięciu wiersza z kolumną znajduje się pole. Pole zawiera najmniejszą niepodzielną wartość, czyli taką część informacji, która nie może być dalej dzielona ze względu na spójność logiczną.

Definicja relacji według E.F. Codda

Relacja r to dowolny podzbiór iloczynu kartezjańskiego jednego lub więcej zbiorów:

$$r \subset D_1 \times D_2 \times \dots \times D_k$$

$$D_1 \times D_2 \times \dots \times D_k = \{(a_1, a_2, \dots, a_k) : a_i \in D_i, i \in \{1, 2, \dots, k\}\}$$

Schematem R relacji nazywamy zbiór atrybutów $\{A_1, \dots, A_n\}$.

Relacją r o schemacie $R = \{A_1, \dots, A_n\}$ nazywamy skończony zbiór $r = \{t_1, \dots, t_m\}$ odwzorowań $t_i: R \rightarrow D$, gdzie D jest równie sumie dziedzin atrybutów A_1, \dots, A_n takich, że $t_i(A_j) \in D_j$ dla $i = 1, \dots, m, j = 1, \dots, n$.

Każde takie odwzorowanie nazywamy krotką.

Przykład 1.1

Schematem relacji nazywamy zbiór atrybutów $R = \{A_1, A_2, \dots, A_n\}$, gdzie A_1, A_2, \dots, A_n są atrybutami reprezentowanymi w tabeli poprzez nazwy kolumn.

Przykładowym schematem relacji *towar* będzie zbiór:

`towar{Id_towaru, nazwa, cena, ilość}.`

Każdemu atrybutowi (A_1, A_2, \dots, A_n) przyporządkowana jest dziedzina (zakres dopuszczalnych wartości atrybutu) reprezentowana przez typ danych.

W tym przypadku dla poszczególnych kolumn zdefiniowano następujące typy danych:

`Id_towaru int,`
`nazwa varchar(64),`
`cena numeric(7,2),`
`ilość numeric(6).`

Dziedziną relacji o schemacie $R = \{A_1, A_2, \dots, A_n\}$ nazywamy sumę dziedzin wszystkich jej atrybutów:

$$D(R) = D(A_1) \cup D(A_2) \cup \dots \cup D(A_n).$$

Przykładową dziedziną relacji *towar* będzie dziedzina:

$$D(\text{towar}) = \text{int} \cup \text{varchar}(64) \cup \text{numeric}(7,2) \cup \text{numeric}(6).$$

Relacją o schemacie $R = \{A_1, A_2, \dots, A_n\}$ nazywamy skończony zbiór $r = \{t_1, t_2, \dots, t_m\}$ odwzorowań $t_i: R \rightarrow D(R)$ takich, że dla każdego j z zakresu $1 \leq j \leq n$ zachodzi zależność:

$$t_i(A_j) \in D(A_j).$$

Tak zdefiniowane pojedyncze odwzorowanie nosi nazwę krotki i odpowiada mu pojedynczy wiersz tabeli. Wartością krotki jest suma wartości poszczególnych atrybutów.

Przykładową relacją *towar* będzie:

`komputer, 2300, 6`
`drukarka, 350, 13`
`monitor, 400, 7`
`klawiatura, 90, 16`

Klucz

Kluczem schematu R relacji nazywamy taki zbiór atrybutów K tego schematu, że na podstawie wartości atrybutów z tego zbioru można jednoznacznie zidentyfikować każdą krotkę.

Właściwości klucza:

- Wartość klucza pozwala jednoznacznie identyfikować krotki.
- Dany schemat może posiadać kilka kluczy.
- Każdy nadzbiór klucza jest kluczem.
- Klucz, którego żaden podzbiór właściwy nie jest kluczem, nazywamy kluczem właściwym lub kandydującym.

Wśród kluczy wybiera się jeden, który staje się kluczem głównym (podstawowym).

Integralność danych

Integralność danych (spójność danych) oznacza poprawność struktury bazy danych (czyli zgodność ze schematem bazy danych) oraz poprawność przechowywanych w niej danych. W relacyjnym modelu danych występują następujące rodzaje więzów integralności:

- *Integralność encji* — każda relacja posiada klucz główny i żaden element klucza głównego nie może posiadać wartości pustej (*NULL*).
- *Integralność referencyjna* — każda wartość klucza obcego jest równa wartości klucza głównego określonej krotki lub wynosi *NULL*.
- *Więzy ogólne* — dodatkowe warunki dotyczące poprawności danych określone przez użytkowników lub administratorów baz danych.

Algebra relacji

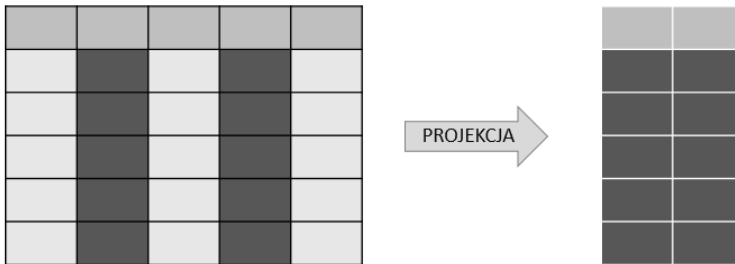
Algebra relacji zawiera zbiór operacji, które pozwalają na tworzenie potrzebnych relacji z relacji dostępnych w bazie danych.

Selekcja to wybór tych krotek relacji, które spełniają określone warunki (rysunek 1.4).



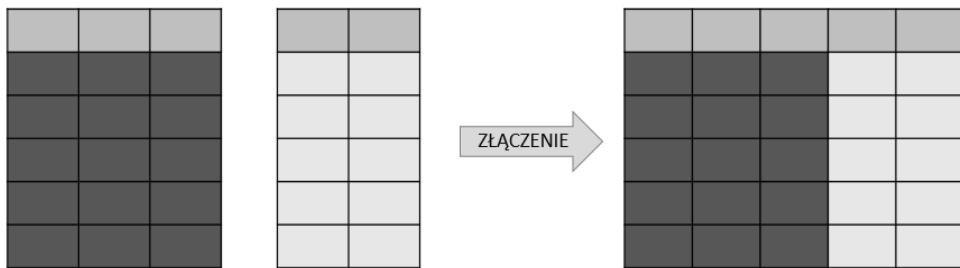
Rysunek 1.4. Wybór krotek z relacji (selekcja)

Projekcja to wybór z relacji określonych atrybutów (rysunek 1.5). Projekcja nazywana jest czasami selekcją pionową lub rzutem.



Rysunek 1.5. Wybór atrybutów z relacji (projekcja)

Złączenie to utworzenie z danych zawartych w dwóch relacjach jednej relacji (rysunek 1.6). Najczęściej łączone są relacje, które mają taki sam atrybut.



Rysunek 1.6. Złączenie dwóch relacji

Iloczynem kartezjańskim dwóch relacji — relacji r o schemacie $R = \{A_1, \dots, A_k\}$ oraz relacji s o schemacie $S = \{B_1, \dots, B_m\}$ takich, że $R \cap S = \emptyset$ — nazywamy relację $q = r \times s$ o schemacie $Q = R \cup S$ taką, że

$$t \in q \Leftrightarrow (\exists u \in r)(t \upharpoonright R = u) \wedge (\exists v \in s)(t \upharpoonright S = v)$$

$$t \in q \Leftrightarrow t \upharpoonright R \in r \wedge t \upharpoonright S \in s.$$

Iloczyn kartezjański dwóch relacji jest zbiorem wszystkich możliwych połączeń krotek obu relacji.

Wartość NULL

Jednym z kluczowych problemów relacyjnego modelu danych było podejście do brakującej informacji (na przykład nieznanego numeru telefonu lub numeru domu). Ostatecznie E.F. Codd wprowadził do modelu relacyjnego dodatkową wartość: **NULL**. Wskutek tego nastąpiło rozszerzenie logiki dwuwartościowej operatorów porównania (*Tak*, *Nie*) do logiki trójwartościowej (na każde pytanie można odpowiedzieć: *Tak*, *Nie*, *Nieznane*).

Wartość **NULL** określa wartość atrybutu, która w danej chwili nie jest znana lub nie może zostać ustalona.

1.3.2. Relacyjny model baz danych

Istnieje wiele różnych podejść do relacyjnego modelu baz danych. Dwa główne to podejście formalne (opis relacyjnego modelu baz danych za pomocą reguł matematycznych) oraz podejście intuicyjne (podejście do relacyjnego modelu baz danych w sposób czysto użytkowy).

Cechy relacyjnego modelu baz danych:

- Podstawową formą przechowywania danych jest tabela (relacja).
- Użycie kombinacji „wartość klucza podstawowego, nazwa tabeli i nazwa kolumny” zapewnia dostęp do dowolnych danych.
- Manipulacje na danych są realizowane za pomocą selekcji, projekcji i złączenia.
- Musi być obsługiwana wartość **NULL** (wartość **NULL** przedstawia brakujące lub bezużyteczne informacje, na przykład nieznaną numer telefonu).
- Integralność danych powinna być naturalną cechą projektu bazy danych.

Korzyści płynące z używania modelu relacyjnego:

- efektywność przechowywania danych,
- pewność integralności danych,
- możliwość rozbudowy bazy danych,
- możliwość łatwej zmiany w strukturze bazy danych,
- zwiększenie szybkości dostępu do danych.

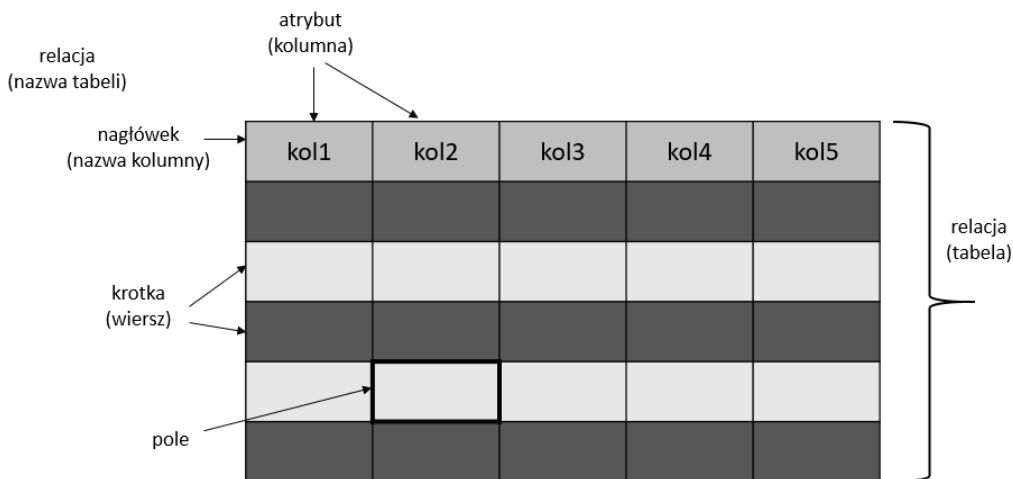
Opisując relacyjny model danych, musimy określić:

- struktury danych — czyli w jaki sposób i według jakich zasad zostanie zorganizowane przechowywanie danych oraz według jakich zasad będziemy je projektować,
- język manipulowania danymi — czyli w jaki sposób będą pobierane, zapisywane, modyfikowane i usuwane dane znajdujące się w bazie danych,
- integralność danych — czyli w jaki sposób zostanie zapewniona poprawność przechowywanych danych.

Tabele

W modelu relacyjnym baz danych wszystkie dane są przechowywane w dwuwymiarowych tabelach (relacjach). W skład bazy danych może wchodzić wiele relacji.

Tabela to zbiór powiązanych ze sobą danych. Jest to układ poziomych wierszy, nazywanych rekordami lub krotkami, i pionowych kolumn, nazywanych polami rekordu lub atrybutami. Tabela jest identyfikowana poprzez nazwę (rysunek 1.7).



Rysunek 1.7. Struktura tabeli

Przecięcie kolumny i wiersza tworzy pole. Nazwa kolumny jest jednocześnie nazwą pola. Pojęcia *pole* używa się również do określenia kolumny.

W modelu relacyjnym:

- każda tabela musi mieć jednoznaczną nazwę,
- każda kolumna w tabeli musi mieć jednoznaczną nazwę w obrębie tej tabeli,
- wszystkie wartości w kolumnie są tego samego typu,
- w tabeli nie mogą istnieć dwa identyczne wiersze,
- tabela nie może istnieć bez wierszy,
- przechowywane w tabeli dane oparte są na typach prostych (dane elementarne),
- kolejność wierszy i kolejność kolumn w tabeli nie ma żadnego znaczenia.

Klucz podstawowy

Tabela musi mieć zestaw pól, które będą jednoznacznie identyfikować dany rekord. W najlepszym przypadku w tabeli może istnieć jedno pole spełniające to założenie. To klucz podstawowy, nazywany również kluczem głównym lub pierwotnym.

WSKAZÓWKA

Klucz podstawowy to minimalna kombinacja pól identyfikująca każdy rekord w tabeli w sposób jednoznaczny.

Klucz podstawowy pozwala w sposób efektywny przeszukiwać i odczytywać dane w bazie oraz łączyć dane zapisane w różnych tabelach.

Klucz podstawowy nie może zawierać powtarzających się danych oraz nie może być pusty. Oznacza to, że w tabeli musi się znaleźć jedno lub kilka pól, które pozwolą odróżniać dane zapisane w jednym rekordzie od danych zapisanych w innym rekordzie.

Jeżeli tabela zawiera dane osobowe, takim kluczem może być kolumna z nazwiskiem. Każdą osobę zapisaną w tabeli rozpoznamy po nazwisku i odczytamy jej dane z właściwego rekordu. Co jednak zrobić, jeżeli w tabeli zapisaliśmy informacje o kilku osobach, które mają takie samo nazwisko?

W takim wypadku kluczem podstawowym może być kombinacja pól, czyli do pola *Nazwisko* możemy dodać pole *Imię*. Teraz każdą osobę w tabeli znajdziemy, podając jej nazwisko i imię. A co zrobić, jeśli w tabeli są umieszczone informacje o dwóch osobach, które mają identyczne nazwisko i imię? Można do klucza dodać kolejne pole, na przykład *Data urodzenia*.

W celu uniknięcia dodawania kolejnych pól do klucza bardzo często funkcję klucza podstawowego pełni klucz sztuczny. Najprostszym sposobem utworzenia klucza sztucznego jest dodanie do tabeli dodatkowego pola i umieszczenie w nim kolejnych numerów.

WSKAZÓWKA

Klucz sztuczny to pole zawierające unikatowy numer identyfikacyjny nadany w sposób sztuczny każdemu obiektowi umieszczonemu w tabeli.

Jeżeli w tabeli zostało utworzone takie pole, staje się ono automatycznie kluczem podstawowym, ponieważ spełnia wszystkie wymagania dotyczące tego klucza.

Klucz obcy to jedno pole lub więcej pól tabeli (kolumn), które odwołują się do pola lub pól klucza podstawowego w innej tabeli. Klucz obcy pokazuje, w jaki sposób tabele są powiązane. Jest niezbędny do zdefiniowania połączenia między tabelami.

Integralność danych

W modelu relacyjnym baz danych integralność danych zachowana jest poprzez:

- klucz podstawowy,
- klucz obcy,
- zawężenie dziedziny,
- unikatowość wartości,
- możliwość nadawania wartości pustych.

Połączenia

Projektując bazę danych, dzielimy dane na wiele tabel tematycznych, tak aby każda informacja została zapisana tylko raz. Aby zestawiać razem dane zapisane w różnych tabelach, tworzy się między nimi połączenia. W wyniku powstaje nowa tabela (relacja) zawierająca połączone wiersze tabel, dla których wartości klucza obcego i klucza

podstawowego są takie same. Definiowanie logicznego połączenia między tabelami bazy danych nazywane jest również relacją.

Typy połączeń (związków)

W rzeczywistości (w życiu codziennym) występują trzy typy połączeń (związków).

Związek „jeden do jednego” (1:1)

W związku „jeden do jednego” każdemu rekordowi z pierwszej tabeli może odpowiadać tylko jeden rekord z drugiej tabeli i każdemu rekordowi z drugiej tabeli może odpowiadać tylko jeden rekord z pierwszej tabeli. Jest to nietypowy rodzaj związku, ponieważ najczęściej informacje powiązane w ten sposób są przechowywane w jednej tabeli. Opisywanego związku używa się w celu odizolowania części tabeli ze względu na bezpieczeństwo danych lub w celu podzielenia danych na podzbiory.

Związek „jeden do wielu” (1:n lub 1:∞)

W związku „jeden do wielu” każdemu rekordowi z pierwszej tabeli może odpowiadać wiele rekordów z drugiej tabeli, a każdemu rekordowi z drugiej tabeli może odpowiadać najwyżej jeden rekord z pierwszej tabeli. Jest to typ związku najczęściej występujący w relacyjnych bazach danych.

Związek „wiele do wielu” (m:n)

W związku „wiele do wielu” każdemu rekordowi z pierwszej tabeli może odpowiadać wiele rekordów z drugiej tabeli i każdemu rekordowi z drugiej tabeli może odpowiadać wiele rekordów z pierwszej tabeli.

Związek „wiele do wielu” zawsze jest realizowany jako dwa związki „jeden do wielu”.



1.4. Projektowanie bazy danych

W bazie danych przechowujemy tylko niektóre informacje o świecie rzeczywistym. Wybór właściwych wycinków rzeczywistości i dotyczących ich danych jest bardzo istotny — od niego zależy prawidłowe działanie bazy. Aby ten wybór był właściwy, należy wskazać informacje, które powinny być przechowywane w bazie danych, oraz określić ich strukturę.

1.4.1. Zasady projektowania bazy danych

Cały proces projektowania bazy danych możemy podzielić na kilka etapów:

- planowanie bazy danych,
- tworzenie modelu konceptualnego (diagramy ERD),
- transformacja modelu konceptualnego na model relacyjny,
- proces normalizacji bazy danych,
- wybór struktur i określenie zasad dostępu do bazy danych.

1.4.2. Podstawowe pojęcia

Z punktu widzenia relacyjnej bazy danych świat rzeczywisty widzimy i analizujemy jako zestaw **encji** i związków zachodzących między nimi.

Encja

WSKAZÓWKA

Encja jest każdy przedmiot, zjawisko, stan lub pojęcie, czyli każdy obiekt, który potrafimy odróżnić od innych obiektów (na przykład: osoba, samochód, książka, stan pogody).

Encje podobne do siebie (opisywane za pomocą podobnych parametrów) grupujemy w zbiory encji. Projektując bazę danych, należy precyzyjnie zdefiniować encje i określić parametry, przy użyciu których będą opisywane.

Atrybut

Encje mają określone cechy wynikające z ich natury. Cechy te nazywamy atrybutami. Zestaw atrybutów, które określamy dla encji, zależy od potrzeb bazy danych.

Dziedzina

Atrybuty encji mogą przyjmować różne wartości. Projektując bazę danych, możemy określić, jakie wartości może przyjmować dany atrybut. Zbiór wartości atrybutu nazywamy dziedziną (domeną).

1.4.3. Planowanie bazy danych

Prostym przykładem bazy danych jest lista klientów sklepu internetowego zajmującego się sprzedażą książek. Zawiera ona zbiór informacji o wycinku otaczającej nas rzeczywistości, czyli o klientach dostępnego w internecie sklepu oraz o oferowanych książkach. Encją jest klient. Zbiór klientów tworzy zbiór encji.

Klient jest encją i jako encja jest określany przez kilka atrybutów (nazwisko, imię, adres zamieszkania, PESEL). Dla każdego atrybutu można ustalić zbiór wartości, na przykład atrybut *Nazwisko* to zbiór wszystkich kombinacji liter alfabetu łacińskiego. W rzeczywistości zbiór nazwisk jest podzbiorem tego zbioru, ponieważ niektóre kombinacje liter nigdy nie wystąpią w nazwisku.

Podobnie *Książka* jest encją i jest określana przez atrybuty: tytuł, autor, cena, rok wydania, wydawnictwo, miejsce wydania, język.

Ocena, które z atrybutów encji są istotne, wynika z przeznaczenia bazy danych. Należy zwrócić uwagę, czy w bazie nie zostały umieszczone atrybuty, które tylko pozornie

należą do danej encji, a naprawdę są atrybutami innej encji. Załóżmy, że w bazie danych trzeba określić narodowość autora książki. Jeżeli atrybut *Narodowość* zostanie dodany do encji *Książka*, nastąpi pomieszczenie atrybutów, ponieważ narodowość jest atrybutem autora, a nie książki.

Można wymienić następujące niekorzystne skutki wspomnianego pomieszczenia informacji:

- Narodowość autora jest powtórzona tyle razy, ile książek tego autora znajdzie się w bazie.
- Jeżeli w bazie nie została umieszczona jeszcze żadna książka danego autora, nie ma o nim informacji w tej bazie.
- Znalezienie informacji o dowolnym autorze staje się wyjątkowo żmudne.
- Występują problemy z poprawianiem błędów popełnionych w czasie wprowadzania danych.

Ponieważ trudno uniknąć błędów podczas wprowadzania danych, należy odpowiednio zabezpieczyć się przed nimi na etapie projektowania bazy danych. Jeśli na przykład pojawiają się różne narodowości dla tego samego autora, jak rozstrzygnąć, która jest właściwa? Jak usunąć ten błąd?

Właściwym postępowaniem jest zaprojektowanie oddzielnej encji dla autora. Każdy autor w tym zbiorze encji występuje raz, a jego atrybutami, oprócz nazwiska i imienia, mogą być: narodowość, okres tworzenia, rodzaj twórczości. Oto zalety tego rozwiązania:

- Wyszukiwanie autora i jego danych jest proste.
- Każdy autor występuje na liście tylko raz i jeżeli nawet jego książek na razie nie ma w bazie, to autor pojawia się na liście.
- Ewentualne błędy występują tylko jednokrotnie i łatwo je poprawić.

Należy pamiętać, że prawidłowa klasyfikacja danych jest podstawą dobrego projektu bazy danych. Najgorsze efekty otrzymujemy, próbując zaprojektować bazę danych, która będzie gromadziła wszystkie możliwe dane i przetwarzała je na wszystkie możliwe sposoby.

Każda encja powinna mieć przynajmniej jeden atrybut lub kombinację kilku atrybutów, które identyfikują ją jednoznacznie. Ten atrybut to **klucz podstawowy** encji.

Projektując bazę danych na przykład dla klientów sklepu, widzimy, że tym atrybutem będzie nazwisko klienta. Aby uniknąć niejasności, gdy w bazie pojawi się kilku klientów o tym samym nazwisku, a czasami również imieniu, możemy przyjąć, że kluczem podstawowym będzie kombinacja atrybutów *Nazwisko*, *Imię* oraz *PESEL*. Jeśli klucz podstawowy składa się z kilku atrybutów, dobrym rozwiązaniem jest zastąpienie go **kluczem sztucznym**. Najczęściej zawiera on unikatowe liczby przypisane kolejnym encjom.

Projektowanie modelu bazy danych powinno składać się z następujących działań:

- określenie występujących zbiorów encji,
- określenie atrybutów przypisanych do poszczególnych encji,
- określenie dziedziny poszczególnych atrybutów,
- ustalenie kluczy podstawowych,
- określenie typów występujących związków,
- zweryfikowanie utworzonego modelu.

1.4.4. Tworzenie modelu konceptualnego (diagramy ERD)

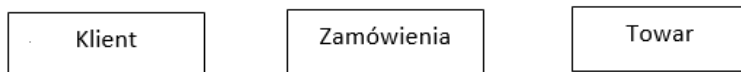
Konceptualne projektowanie bazy danych to konstruowanie schematu danych niezależnego od wybranego modelu danych, docelowego systemu zarządzania bazą danych, programów użytkowych czy języka programowania.

Do tworzenia modelu graficznego schematu bazy danych wykorzystywane są diagramy związków encji, z których najpopularniejsze są diagramy ERD (ang. *Entity Relationship Diagram*). Pozwalają one na modelowanie struktur danych oraz związków zachodzących między tymi strukturami. Nadają się szczególnie do modelowania relacyjnych baz danych, ponieważ umożliwiają prawie bezpośrednie przekształcenie diagramu w schemat relacyjny. Pozwalają na analizę struktury bazy danych, mogą też stanowić część dokumentacji tworzonego systemu baz danych.

Na diagramy ERD składają się trzy rodzaje elementów:

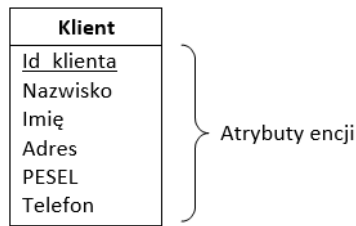
- zbiory encji,
- atrybuty encji,
- związki zachodzące między encjami.

Encja to reprezentacja obiektu przechowywanego w bazie danych. Graficzną reprezentacją encji jest najczęściej prostokąt (rysunek 1.8).

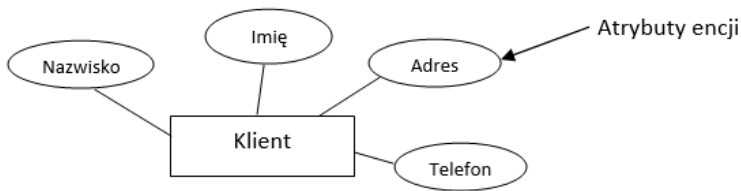


Rysunek 1.8. Graficzna reprezentacja encji

Atrybut opisuje encję. Może on być liczbą, tekstem lub wartością logiczną. W relacyjnym modelu baz danych atrybut jest reprezentowany przez kolumnę tabeli. Graficzna reprezentacja atrybutów dla encji *Klient* została pokazana na rysunkach 1.9 i 1.10.



Rysunek 1.9. Atrybuty encji Klient

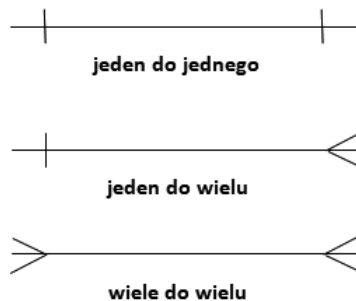


Rysunek 1.10. Atrybuty encji Klient

Związek to powiązanie między dwoma zbiorami encji. Każdy związek ma dwa końce, do których są przypisane następujące atrybuty:

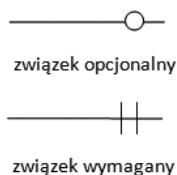
- nazwa,
- stopień związku,
- uczestnictwo lub opcjonalność związku.

Stopień związku określa, jakiego typu związek zachodzi między encjami. „Jeden do jednego” oznacza, że encji odpowiada dokładnie jedna encja. „Jeden do wielu” oznacza, że encji odpowiada jedna lub kilka encji. „Wiele do wielu” oznacza, że encji lub kilku encjom odpowiada jedna lub kilka encji. Opis reprezentacji graficznej stopnia związku został pokazany na rysunku 1.11.



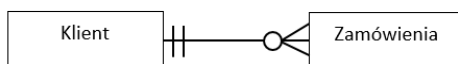
Rysunek 1.11. Graficzna reprezentacja związków zachodzących między encjami

Opcjonalność związku określa, czy związek jest opcjonalny, czy wymagany. Opis reprezentacji graficznej opcjonalności związku został pokazany na rysunku 1.12.



Rysunek 1.12. Graficzna reprezentacja opcjonalności związku

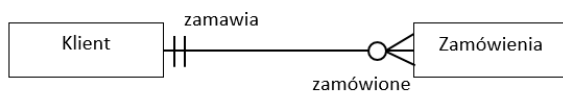
Przykład prostego diagramu związków encji został pokazany na rysunku 1.13. *Zamówienie* musi mieć przypisanego *Klienta*, natomiast *Klient* może złożyć *Zamówienie*. *Klient* może złożyć wiele *Zamówień*, ale złożone *Zamówienie* dotyczy tylko jednego *Klienta*.



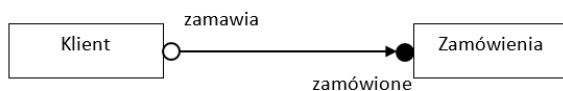
Rysunek 1.13. Diagram związków encji

Diagramy ERD spotyka się w wielu różnych notacjach, na przykład: Martina, Bachmana, Chena, IDEF1X (rysunek 1.14).

Notacja Martina



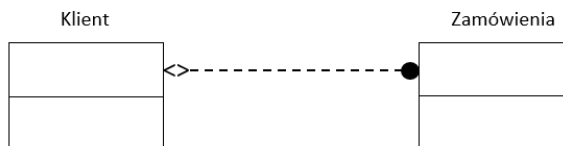
Notacja Bachmana



Notacja Chena



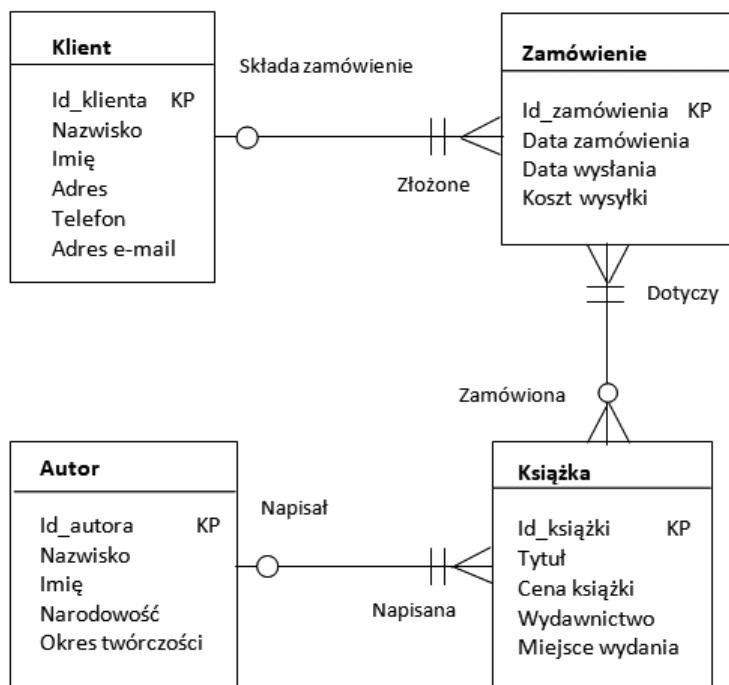
Notacja IDEF1X



Rysunek 1.14. Diagram związków encji zapisany w różnych notacjach

Istnieje wiele narzędzi wspomagających rysowanie diagramów ERD, ale jedynie w przypadku narzędzi klasy CASE (ang. *Computer Aided Software Engineering*) można mówić o określonej notacji. Narzędzia CASE są wykorzystywane do projektowania oprogramowania. Pozwalają tworzyć model graficzny oraz poprzez generowanie gotowych skryptów wspomagają wytwarzanie oprogramowania.

Prosty przykład diagramu ERD w notacji Martina dla księgarni został przedstawiony na rysunku 1.15.



Rysunek 1.15. Diagram ERD w notacji Martina dla księgarni

W pokazanym na rysunku schemacie encje zostały przedstawione za pomocą prostokątów zawierających listę atrybutów. Klucze podstawowe zostały oznaczone symbolem KP. Stopień związku i uczestnictwo zostały oznaczone liniami łączącymi z odpowiednimi symbolami opisującymi stopień oraz opcjonalność związku. Należy zwrócić uwagę na to, że w encjach nie umieszcza się kluczy obcych. Zostaną one dodane na etapie przekształcania encji w tabele.

Tak przygotowany diagram ERD pozwala na późniejszą weryfikację i optymalizację bazy danych, a także stanowi podstawową dokumentację projektowanej bazy danych. Można go również wykorzystać w jednym z narzędzi CASE do wygenerowania fizycznej struktury bazy danych.

1.4.5. Projektowanie bazy danych za pomocą narzędzi CASE

Narzędzia CASE (ang. *Computer Aided Software Engineering*) są wykorzystywane podczas projektowania różnego rodzaju oprogramowania, najczęściej wspomagają proces jego wytwarzania.

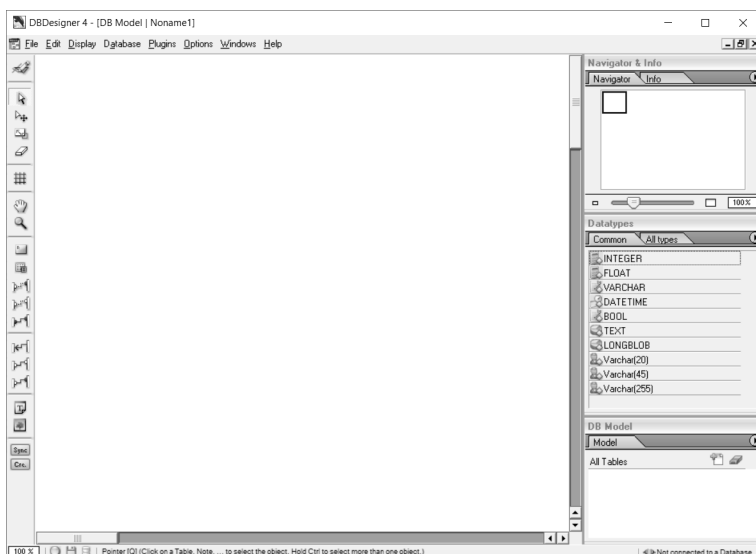
Narzędzia te pozwalają tworzyć modele graficzne odpowiadające konstrukcjom programistycznym. Wykorzystywane są tutaj edytory notacji graficznych, które dają możliwości tworzenia diagramów i powiązań między poszczególnymi elementami. Bardziej zaawansowane edytory umożliwiają przetwarzanie informacji i udostępnianie danych do aplikacji zewnętrznych, na przykład kodów w językach Visual Basic, SQL, ODBC.

Narzędzia CASE mogą być stosowane do generowania kodu na podstawie zaprojektowanego modelu danych, można również za ich pomocą, na podstawie analizy kodu źródłowego, odtworzyć projekt i specyfikację bazy danych.

Przykładem narzędzia typu CASE jest program DBDesigner4. Jest to narzędzie do wizualnego projektowania, modelowania i tworzenia baz danych. Program został stworzony z myślą o bazie MySQL, ale obsługuje również bazy danych Oracle, SQLite, MS SQL. Jest rozpowszechniany jako open source i jest dostępny na stronie <http://fabforce.net/index.php>.

Zainstalowany program można wykorzystać do przygotowania logicznego projektu bazy danych.

Okno programu DBDesigner4 składa się z pięciu obszarów (rysunek 1.16). Pusty obszar na środku ekranu to obszar roboczy. Z lewej strony znajduje się pasek narzędzi. Z prawej w górnej części znajduje się okno nawigacji i informacji, na środku okno typów danych, a w dolnej części okno bieżącego modelu bazy danych.



Rysunek 1.16. Okno programu DBDesigner4

Pracę w programie rozpoczynamy od utworzenia nowego projektu (menu *File/New*). Kolejnym etapem jest utworzenie pierwszej tabeli. Należy wybrać na pasku narzędzi ikonę tworzenia tabeli (*New Table*) i kliknąć obszar roboczy. W efekcie zostanie utworzona tabela. Po dwukrotnym kliknięciu tabeli można otworzyć okno jej edytowania, w którym należy wprowadzić nazwę tabeli i nazwy pól oraz wybrać ich typy (rysunek 1.17).

Table Editor

Table Name: Klient Table Prefix: Default (no prefix) Table Type: MYISAM (Standard) Weak entity: ☐ is n:m Table

| Column Name | DataType | NN | AI | Flags | Default Value | Comments |
|-------------|-------------|-------------------------------------|-------------------------------------|--|---------------|----------|
| id_klienta | INTEGER | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL | | |
| Nazwisko | VARCHAR(45) | | | <input type="checkbox"/> BINARY | | |
| Imię | VARCHAR(20) | | | <input type="checkbox"/> BINARY | | |
| Adres | VARCHAR(45) | | | <input type="checkbox"/> BINARY | | |
| Telefon | VARCHAR(9) | | | <input type="checkbox"/> BINARY | | |
| E-mail | VARCHAR(45) | | | <input type="checkbox"/> BINARY | | |
| | | | | | | |

Indices

- Table Options
- Advanced
- Standard Inserts
- Comments

PRIMARY

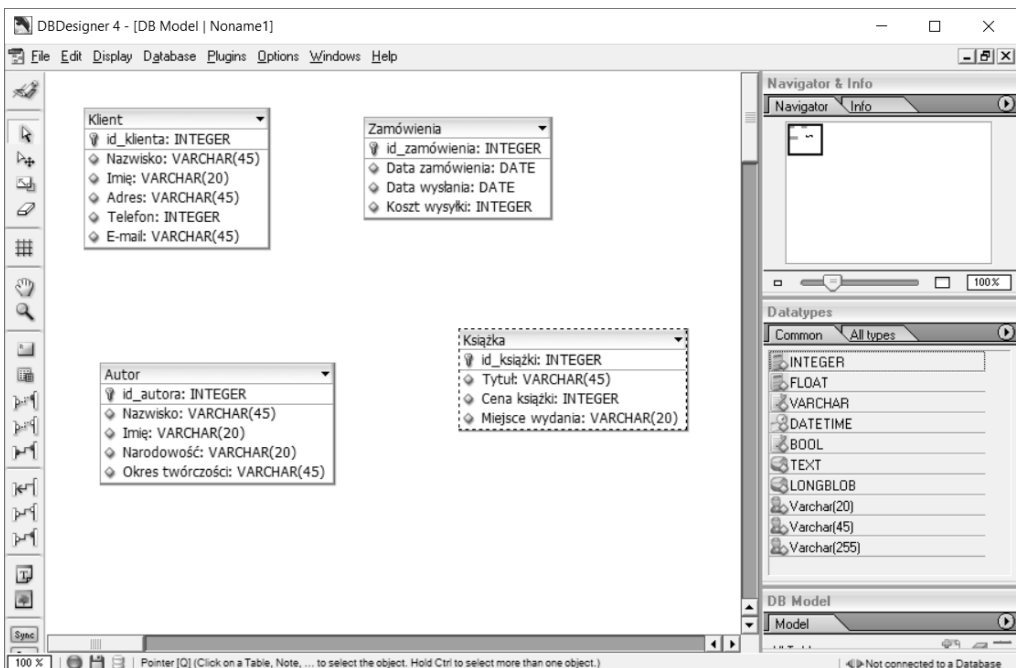
Indexname: PRIMARY Columns (use Drag'n'Drop to add Columns): id_klienta

Index Type: PRIMARY

Rysunek 1.17. Okno edytowania tabeli

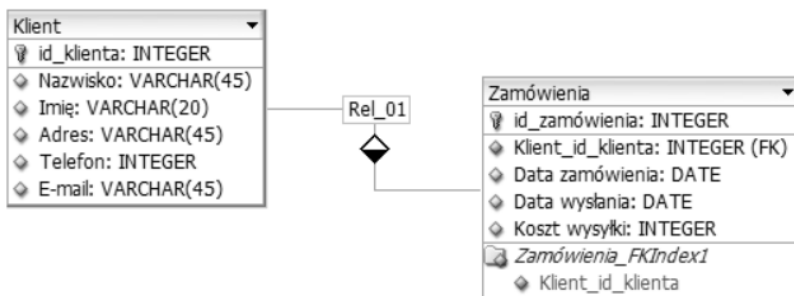
W oknie edytowania tabeli w kolumnie *Column Name* należy wprowadzić nazwę tworzonego pola, w kolumnie *DataType* określić typ danych, w kolumnie *NN* określić, czy dozwolona jest wartość *NULL (NOT NULL)*, w kolumnie *AI* zaznaczyć automatyczne zwiększanie wartości o 1 (*AUTO INCREMENT*), w kolumnie *Flags* zdefiniować dodatkowe opcje zależne od typu danych, w kolumnie *Default Value* ustawić wartość domyślną pola, a w kolumnie *Comments* wstawić komentarz.

Dla projektowanego modelu graficznego bazy danych w podobny sposób należy utworzyć pozostałe tabele (rysunek 1.18).



Rysunek 1.18. Tabele zaprojektowane w programie DBDesigner4

Po utworzeniu wszystkich tabel należy zdefiniować połączenia między nimi. Program DBDesigner4 obsługuje wszystkie rodzaje połączeń występujących w bazie danych. Ikony odpowiednich połączeń są dostępne na pasku narzędzi. Aby dodać połączenie typu „jeden do wielu”, należy wybrać ikonę *New 1:n*. Po wybraniu ikony rodzaju połączenia klikamy najpierw tabelę ze strony „jeden” (*Klient*), a następnie tabelę ze strony „wiele” (*Zamówienia*). W wyniku zdefiniowania połączenia w tabeli ze strony „wiele” (*Zamówienia*) pojawiło się nowe pole (*Klient_id_klienta*), opisujące związek między tabelami, które stanie się kluczem obcym (rysunek 1.19).



Rysunek 1.19. Definiowanie połączenia jeden do wielu

Aby edytować utworzone połączenie, należy dwukrotnie kliknąć narysowaną linię. Zostanie otwarte okno edytowania połączenia, w którym można zmienić nazwę relacji oraz nazwę pola klucza obcego (rysunek 1.20).

Rysunek 1.20. Okno edytowania połączenia

W podobny sposób należy zdefiniować połączenie między tabelami *Autor* i *Książka*.

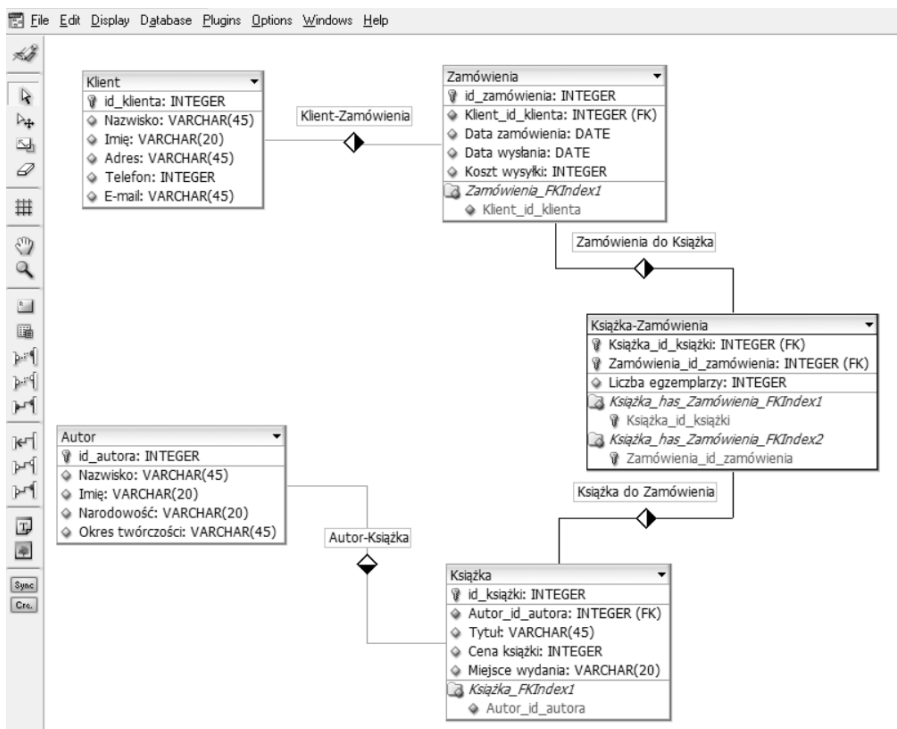
Ponieważ w jednym zamówieniu może znaleźć się kilka książek, a jedna książka może zostać wskazana w wielu zamówieniach, między tabelami *Zamówienia* i *Książki* występuje połączenie „wiele do wielu”. Aby dodać taki typ połączenia, należy wybrać ikonę *New n:m*. Po wybraniu tej ikony klikamy na przykład tabelę *Książka*, a następnie tabelę *Zamówienia* (lub w odwrotnej kolejności). W wyniku zdefiniowania połączenia w projekcie zostanie umieszczona dodatkowa tabela opisująca zdefiniowane połączenie. Będzie ona zawierała klucze podstawowe łączonych tabel (rysunek 1.21).



Rysunek 1.21. Definiowanie połączenia „wiele do wielu”

Do utworzonej tabeli można dodać pola będące atrybutami połączenia, np. gdy chcemy przechowywać informację, w ilu egzemplarzach została zamówiona każda książka.

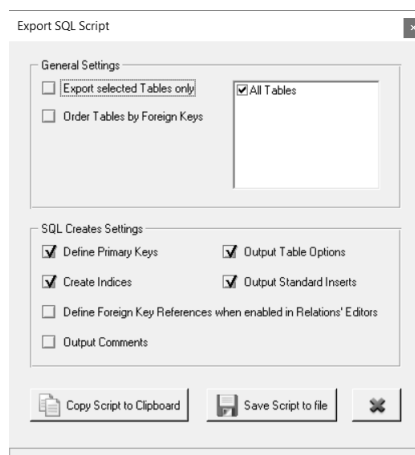
Po utworzeniu połączeń i dodaniu potrzebnych atrybutów uzyskamy efekt podobny do pokazanego na rysunku 1.22.



Rysunek 1.22. Schemat bazy danych uzyskany w programie DBDesigner4

Utworzony projekt należy zapisać w pliku, wybierając z menu *File/Save*. Można również zaimportowaną bazę danych wyeksportować do pliku *.sql*. W tym celu należy wybrać z menu *File/Export/SQL Create Script* i w otwartym oknie zaznaczyć opcje, tak jak pokazano na rysunku 1.23, a następnie kliknąć przycisk *Save Script to file*.

Po wykonaniu tych czynności zostanie wygenerowany skrypt, którego zawartość można zobaczyć, otwierając plik na przykład w edytorze tekstowym Notepad++ (rysunek 1.24).



Rysunek 1.23. Opcje eksportowania projektu bazy do kodu SQL

```

1  CREATE TABLE Autor (
2      id_autora INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
3      Nazwisko VARCHAR(45) NULL,
4      Imię VARCHAR(20) NULL,
5      Narodowość VARCHAR(20) NULL,
6      Okres twórczości VARCHAR(45) NULL,
7      PRIMARY KEY(id_autora)
8  );
9
10 CREATE TABLE Klient (
11     id_klienta INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
12     Nazwisko VARCHAR(45) NULL,
13     Imię VARCHAR(20) NULL,
14     Adres VARCHAR(45) NULL,
15     Telefon INTEGER UNSIGNED NULL,
16     E-mail VARCHAR(45) NULL,
17     PRIMARY KEY(id_klienta)
18 );
19
20 CREATE TABLE Książka (
21     id_książki INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
22     Autor_id_autora INTEGER UNSIGNED NOT NULL,
23     Tytuł VARCHAR(45) NULL,
24     Cena książki INTEGER UNSIGNED NULL,
25     Miejsce wydania VARCHAR(20) NULL,
26     PRIMARY KEY(id_książki),
27     INDEX Książka_FKIndex1(Autor_id_autora)
28 );
29
30 CREATE TABLE Książka-Zamówienia (
31     Książka_id_książki INTEGER UNSIGNED NOT NULL,
32     Zamówienia_id_zamówienia INTEGER UNSIGNED NOT NULL,
33     Liczba egzemplarzy INTEGER UNSIGNED NULL,
34     PRIMARY KEY(Książka_id_książki, Zamówienia_id_zamówienia),
35     INDEX Książka_has_Zamówienia_FKIndex1(Książka_id_książki),
36     INDEX Książka_has_Zamówienia_FKIndex2(Zamówienia_id_zamówienia)
37 );
38
39 CREATE TABLE Zamówienia (
40     id_zamówienia INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
41     Klient_id_klienta INTEGER UNSIGNED NOT NULL,
42     Data zamówienia DATE NULL,

```

Rysunek 1.24. Kod SQL wygenerowany automatycznie za pomocą programu DBDesigner4

Tak przygotowany skrypt może zostać uruchomiony na jednym z serwerów bazodanowych w celu wygenerowania gotowej bazy danych.

1.4.6. Transformacja modelu konceptualnego do modelu relacyjnego

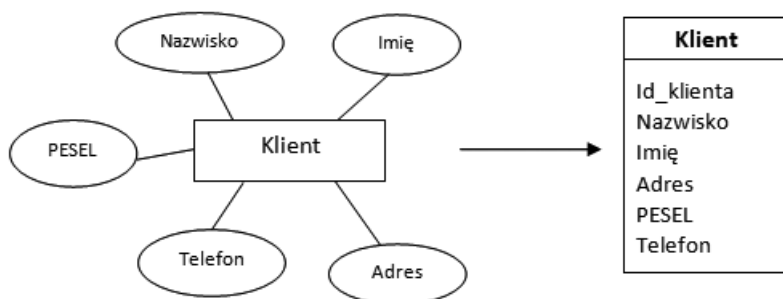
Jeżeli w projektowanej bazie danych występuje kilka zbiorów encji (na przykład encje *Książka* i *Autor*), między tymi encjami zachodzą związki. Każda książka jest dziełem określonego autora, każdy autor ma przypisany zbiór książek. Zarówno zbiory encji, jak i związki zachodzące między encjami zapisujemy w tabelach projektowanej bazy danych. Przy projektowaniu tabel stosujemy reguły, które określają, w jaki sposób należy przekształcać model bazy danych utworzony za pomocą diagramów ERD i narzędzi CASE w schemat bazy danych.

Reguły transformacji diagramów ERD do tabel

Transformacja encji do tabeli

WSKAZÓWKA

Do opisu każdego zbioru podobnych encji stosuje się oddzielną tabelę (rysunek 1.25). Jednej encji odpowiada jeden wiersz. Atrybutowi odpowiada kolumna. Dla każdego atrybutu określa się typ informacji.

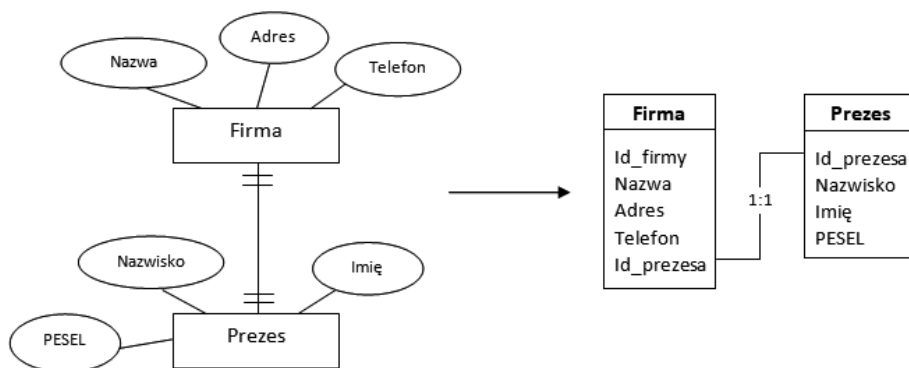


Rysunek 1.25. Transformacja encji do tabeli

Transformacja związku „jeden do jednego”

WSKAZÓWKA

Zapis związku „jeden do jednego” może zostać umieszczony w dodatkowej kolumnie w tabeli należącej do związku. Kolumna ta może znaleźć się w dowolnej tabeli. Dołączona kolumna zawiera klucz tabeli, z którą zachodzi związek (rysunek 1.26).

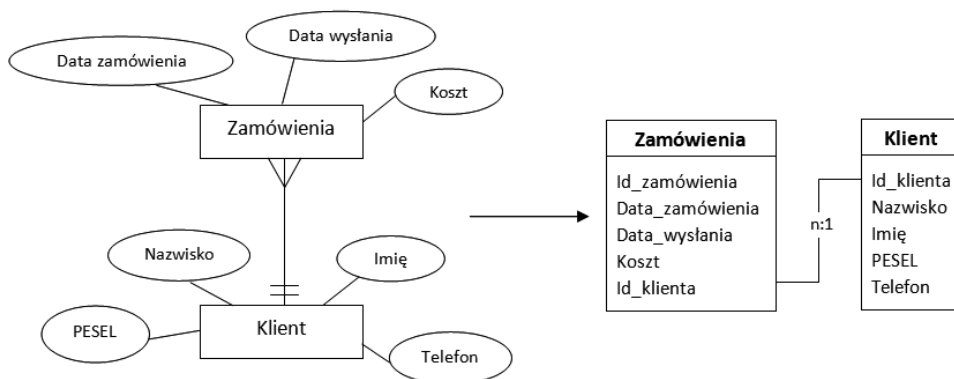


Rysunek 1.26. Transformacja związku „jeden do jednego” do tabeli

Transformacja związku „wiele do jednego”

WSKAZÓWKA

Zapis związku „wiele do jednego” może być umieszczony w dodatkowej kolumnie w tabeli należącej do związku. Kolumna ta musi znaleźć się w tabeli ze strony „wiele”. Dołączona kolumna zawiera klucz tabeli, z którą zachodzi związek (rysunek 1.27).

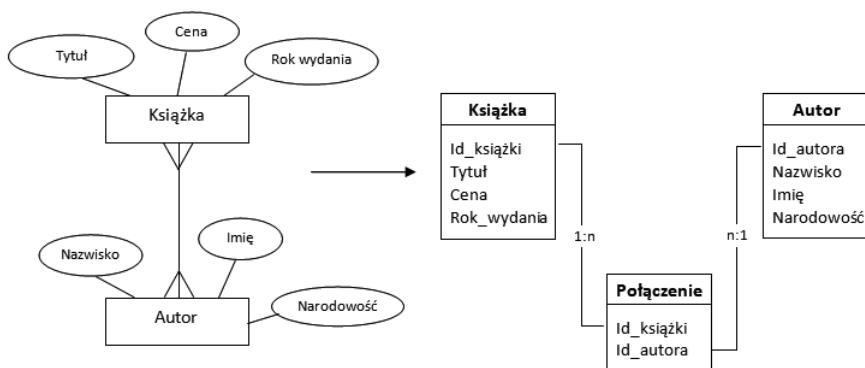


Rysunek 1.27. Transformacja związku „wiele do jednego” do tabeli

Transformacja związku „wiele do wielu”

WSKAZÓWKA

Związek „wiele do wielu” opisuje się w oddzielnej tabeli, której kolumny tworzone są z kluczy encji należących do związku (rysunek 1.28).

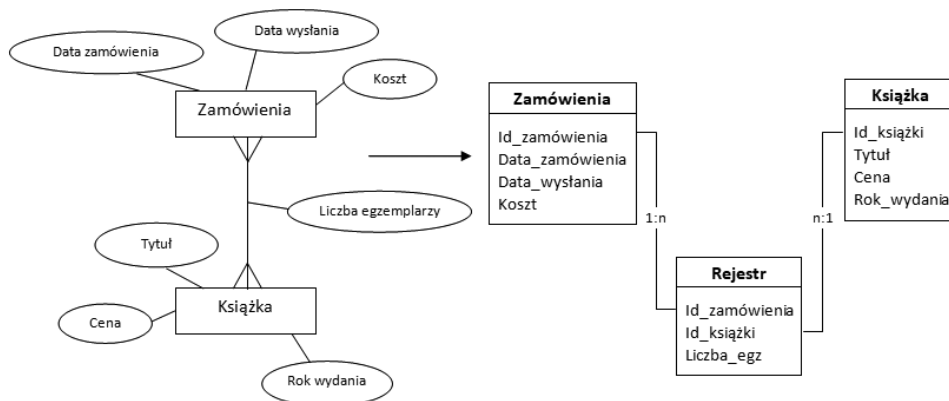


Rysunek 1.28. Transformacja związku „wiele do wielu” do tabeli

Transformacja związku „wiele do wielu” z atrybutami

WSKAZÓWKA

Jeżeli związek „wiele do wielu” posiada atrybuty, należy je umieścić w tabeli opisującej związek (rysunek 1.29).



Rysunek 1.29. Transformacja związku „wiele do wielu” z atrybutami

Jeśli klucze w tabeli opisującej związek składają się z wielu atrybutów lub są długie, należy zastąpić je kluczami sztucznymi.

1.4.7. Normalizacja tabel

Normalizację stosuje się, aby sprawdzić, czy zaprojektowane tabele mają prawidłową strukturę. Proces normalizacji rozpoczynamy, gdy zostanie utworzony wstępny projekt tabel. Pozwala ona określić, czy informacje przewidziane w projekcie bazy zostały przydzielone do właściwych tabel. Natomiast nie da odpowiedzi na pytanie, czy projekt bazy danych jest prawidłowy.

Korzyści płynące z normalizacji tabel są następujące:

- zlikwidowanie problemu powtarzania danych,
- optymalizacja objętości bazy danych,
- optymalizacja efektywności obsługi bazy danych,
- minimalizacja zagrożenia błędami przy wprowadzaniu danych.

Normalizacja bazy danych wymaga rozbicia dużych tabel na mniejsze. Zmniejsza to wydajność bazy, dlatego w niektórych przypadkach nie normalizuje się tabel — szczególnie w systemach niekorzystających z modelu relacyjnego.

Stosowane są cztery reguły normalizacji, ale w większości projektów baz danych wystarczy sprawdzić trzy pierwsze. Zostaną one omówione poniżej.

Dla każdej z nich stosowane są określenia: pierwsza postać normalna (**I PN**), druga postać normalna (**II PN**) i trzecia postać normalna (**III PN**).

Pierwsza postać normalna

DEFINICJA

Tabela jest w pierwszej postaci normalnej (**I PN**), gdy każdy wiersz w tabeli przechowuje informacje o pojedynczym obiekcie, a każde pole tabeli zawiera informację elementarną (atomową).

Oznacza to, że w komórce tabeli nie może wystąpić lista wartości, na przykład w polu *Narodowość autora* nie można umieścić dwóch narodowości — polskiej i angielskiej.

Załóżmy, że w projektowanej bazie danych dla księgarni została zaprojektowana tabela *Realizacja zamówień* z polami: *Nazwisko klienta*, *Imię*, *Adres*, *Telefon*, *PESEL*, *Tytuł książki*, *Liczba egzemplarzy*, *Cena*. W polu *Tytuł książki* będą umieszczane tytuły książek zakupionych przez klienta. Gdy klient kupi dwie książki, w polu *Tytuł książki* należałoby wpisać dwa tytuły. Powstałaby lista wartości (rysunek 1.30). Tak zaprojektowana tabela nie jest w **I PN**. Nie będzie możliwe prawidłowe przetwarzanie danych zapisanych w tabeli. Rozwiązaniem jest zapisanie informacji o zakupionych książkach

w dwóch wierszach. W pierwszym wierszu w polu *Tytuł książki* należy wpisać tytuł pierwszej książki, w drugim — tytuł drugiej książki, natomiast nazwisko klienta zostanie powtórzone w liczbie wierszy równej liczbie zakupionych książek (rysunek 1.31). Teraz tabela jest w **I PN**.

| Realizacja zamówień | | | | | | | |
|---------------------|----------|----------|--------------|-------------|------------------|-------|----------|
| Nazwisko klij | Imię | Adres | Telefon | PESEL | Tytuł książki | Liczł | Cena |
| Nowak | Marek | Toruń | (56) 6589234 | 79120307431 | Dziady | 2 | 20,00 zł |
| Kowalski | Adam | Warszawa | (22) 3451234 | 80122401871 | Balladyna, Tango | 1 | 15,00 zł |
| Górecki | Grzegorz | Poznań | (45) 2367897 | 82061203983 | Pan Tadeusz | 1 | 21,00 zł |
| Zan | Marcin | Gdańsk | (33) 8373635 | 82020201875 | | | |

Rysunek 1.30. Tabela nie jest w **I PN**, ponieważ w polu Tytuł książki pojawiła się lista wartości

| Realizacja zamówień | | | | | | | |
|---------------------|----------|----------|--------------|-------------|---------------|-------|----------|
| Nazwisko klie | Imię | Adres | Telefon | PESEL | Tytuł książki | Liczł | Cena |
| Nowak | Marek | Toruń | (56) 6589234 | 79120307431 | Dziady | 2 | 20,00 zł |
| Kowalski | Adam | Warszawa | (22) 3451234 | 80122401871 | Balladyna | 1 | 15,00 zł |
| Kowalski | Adam | Warszawa | (22) 3451234 | 80122401871 | Tango | 2 | 18,00 zł |
| Górecki | Grzegorz | Poznań | (45) 2367897 | 82061203983 | Pan Tadeusz | 1 | 21,00 zł |

Rysunek 1.31. Tabela jest w **I PN**, ponieważ w polu Tytuł książki występują pojedyncze wartości

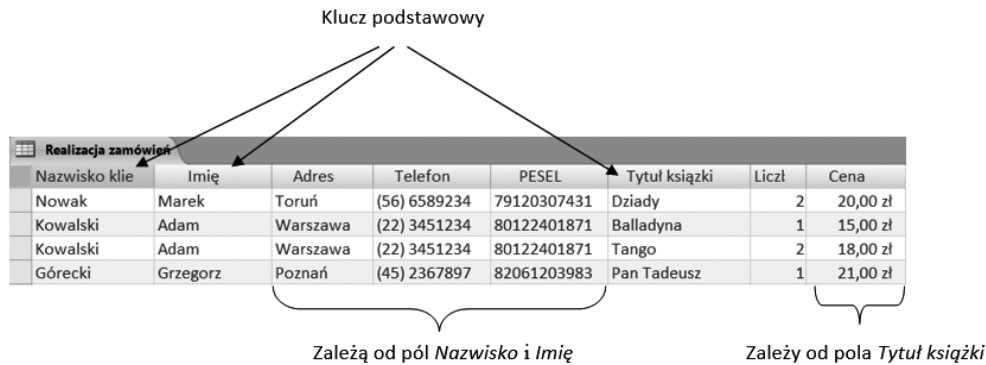
Druga postać normalna

DEFINICJA

Tabela jest w drugiej postaci normalnej (**II PN**), jeżeli jest w pierwszej postaci normalnej (**I PN**) oraz każde z pól niewchodzących w skład klucza podstawowego zależy od całego klucza, a nie od jego części.

Ta reguła i następna służą do sprawdzenia, czy w tabeli i bazie danych nie doszło do redundancji, czyli niepotrzebnego powtarzania danych. Ponieważ **II PN** odnosi się do klucza podstawowego, należy określić ten klucz dla tabeli. Jeżeli klucz podstawowy składa się z jednego pola, tabela jest w **II PN**, ponieważ wszystkie pola, poza polem klucza podstawowego, muszą odnosić się do pola klucza podstawowego.

W zaprojektowanej tabeli *Realizacja zamówień* kluczem podstawowym będzie kombinacja pól *Nazwisko klienta*, *Imię* oraz *Tytuł książki*. Pola niewchodzące w skład klucza podstawowego (*Adres*, *Telefon*, *PESEL*) zależą od pól *Nazwisko* i *Imię*, natomiast nie zależą od pola *Tytuł książki*. Pole *Cena* zależy jedynie od pola *Tytuł książki* (rysunek 1.32). Tylko pole *Liczba egzemplarzy* zależy zarówno od pól *Nazwisko* i *Imię*, jak i od pola *Tytuł książki*. Tabela nie jest w **II PN**.



Rysunek 1.32. Tabela nie jest w II PN, ponieważ tylko pole *Liczba egzemplarzy* zależy od całego klucza

Normalizacja polega na podzieleniu tabeli na takie tabele, które spełnią warunek II PN. Tabelę *Realizacja zamówień* należy podzielić na trzy tabele: *Klient*, z polami *Nazwisko*, *Imię*, *Adres*, *Telefon*, *PESEL*; *Zamówienia*, z polami *Nazwisko klienta*, *Tytuł książki*, *Liczba egzemplarzy*; oraz *Książki*, z polami *Tytuł książki* i *Cena* (rysunek 1.33). Kluczem podstawowym w tabeli *Klient* są pola *Nazwisko* i *Imię*, w tabeli *Zamówienia* pola *Nazwisko klienta* i *Tytuł książki*, a w tabeli *Książki* pole *Tytuł książki*. Zostało zlikwidowane powtarzanie danych w tabeli *Realizacja zamówień* i wszystkie tabele są w II PN.

| Klient | | | | | Zamówienia | | | Książki | |
|---------------|------------|----------|--------------|-------------|-----------------|---------------|------------|---------------|----------|
| Nazwisko klie | Imię | Adres | Telefon | PESEL | Nazwisko klient | Tytuł książki | Liczba egz | Tytuł książki | Cena |
| Nowak | Marek | Toruń | (56) 6589234 | 79120307431 | Nowak | Dziady | 2 | Dziady | 20,00 zł |
| Kowalski | Adam | Warszawa | (22) 3451234 | 80122401871 | Kowalski | Balladyna | 1 | Balladyna | 15,00 zł |
| Górecki | Grzegorz | Poznań | (45) 2367897 | 82061203983 | Kowalski | Tango | 2 | Tango | 18,00 zł |
| Zan | Marcin | Gdańsk | (33) 8373635 | 82020201875 | Górecki | Pan Tadeusz | 1 | Pan Tadeusz | 21,00 zł |
| Bagińska | Anna | Warszawa | (22) 7336252 | 91110402837 | | | | | |
| Pol | Aleksander | Szczecin | (23) 6517830 | 70073003228 | | | | | |

Rysunek 1.33. Podział tabeli *Realizacja zamówień* na tabele spełniające warunek II PN

Aby wyświetlić zestawienie dotyczące klienta i kupionych przez niego książek, należy zdefiniować połączenie między tabelami.

Trzecia postać normalna

DEFINICJA

Tabela jest w trzeciej postaci normalnej (III PN), jeżeli jest w pierwszej i w drugiej postaci normalnej oraz każde z pól niewchodzących w skład klucza podstawowego niesie informację bezpośrednio o kluczu i nie odnosi się do żadnego innego pola.

Załóżmy, że w projektowanej bazie danych dla księgarni została zaprojektowana tabela *Faktura* z polami: *Nazwisko klienta*, *Imię*, *Adres*, *PESEL*, *Numer faktury*, *Sposób płatności* i *Data wystawienia faktury*. Załóżmy również, że w pola tabeli będą wpisywane tylko

wartości elementarne, czyli tabela jest w I PN. Klucz podstawowy to pole *Numer faktury*. Wszystkie pola niewchodzące w skład klucza zależą od całego klucza, czyli tabela jest w II PN. Sprawdźmy, czy tabela jest w III PN. Pola *Sposób płatności* i *Data wystawienia faktury* odnoszą się do faktury, czyli zawierają informacje o kluczu. Natomiast pola *Adres* i *PESEL* zawierają informacje na temat klienta, a nie faktury (rysunek 1.34), czyli nie niosą informacji bezpośrednio o kluczu. Tabela nie jest w III PN.

Klucz podstawowy

| Nazwisko klie | Imię | Adres | PESEL | Numer faktury | Sposób płatn | Data wystaw |
|---------------|------------|----------|-------------|---------------|--------------|-------------|
| Nowak | Marek | Toruń | 79120307431 | | 1 gotówka | 2013-04-06 |
| Kowalski | Adam | Warszawa | 80122401871 | | 2 przelew | 2012-07-29 |
| Kowalski | Adam | Warszawa | 80122401871 | | 4 gotówka | 2012-01-17 |
| Bagińska | Anna | Warszawa | 91110402837 | | 5 gotówka | 2012-08-29 |
| Pol | Aleksander | Szczecin | 70073003228 | | 6 gotówka | 2012-03-02 |
| Pol | Aleksander | Szczecin | 70073003228 | | 3 przelew | 2012-12-10 |

Odnoszą się do pól *Nazwisko i Imię* Odnoszą się do pola *Numer faktury*

Rysunek 1.34. Tabela nie jest w III PN, ponieważ pola *Adres* i *PESEL* nie niosą informacji o kluczu

Normalizacja, podobnie jak w przypadku II PN, polega na podzieleniu tabeli na takie tabele, które spełniają warunek III PN.

Tabelę *Faktura* należy podzielić na dwie tabele: *Faktura* (z polami *Numer faktury*, *Sposób płatności* i *Data wystawienia faktury*) oraz *Klient* (z polami *Nazwisko klienta*, *Imię*, *Adres*, *PESEL*) (rysunek 1.35). Zostało zlikwidowane powtarzanie danych o kliencie w tabeli *Faktura*. Dane o kliencie będą zapisane tylko raz, w tabeli *Klient*.

| Nazwisko klie | Imię | Adres | PESEL |
|---------------|------------|----------|-------------|
| Nowak | Marek | Toruń | 79120307431 |
| Kowalski | Adam | Warszawa | 80122401871 |
| Górecki | Grzegorz | Poznań | 82061203983 |
| Zan | Marcin | Gdańsk | 82020201875 |
| Bagińska | Anna | Warszawa | 91110402837 |
| Pol | Aleksander | Szczecin | 70073003228 |

| Numer faktury | Sposób płatn | Data wystawie | Nazwisko klienta |
|---------------|--------------|---------------|------------------|
| 1 gotówka | | 2013-04-06 | Nowak |
| 2 przelew | | 2012-07-29 | Kowalski |
| 3 przelew | | 2012-12-10 | Pol |
| 4 gotówka | | 2012-01-17 | Kowalski |
| 5 gotówka | | 2012-08-29 | Bagińska |
| 6 gotówka | | 2012-03-02 | Pol |

Rysunek 1.35. Podział tabeli *Faktura* na tabele spełniające warunek III PN

Przykład 1.2

Przestrzegając reguł tworzenia tabel, po sprawdzeniu za pomocą normalizacji, czy tabele mają prawidłową strukturę, baza danych dla księgarni mogłaby składać się z następujących tabel:

Klient → *Id_klienta*, *Nazwisko*, *Imię*, *Kod pocztowy*, *Miejscowość*, *Ulica*, *Nr domu*, *PESEL*, *Telefon*, *Adres e-mail*.

Zamówienia → *Id_zamówienia*, *Id_klienta*, *Data złożenia zamówienia*, *Data wysłania*, *Koszt wysyłki*.

Rejestr → *Id_zamówienia*, *Id_książki*, *Liczba egzemplarzy*.

Książka → *Id_książki*, *Tytuł*, *Id_autora*, *Cena książki*, *Rok wydania*, *Wydawnictwo*, *Rodzaj literatury*, *Miejsce wydania*, *Język książki*, *Opis*, *Zdjęcie okładki*.

Autor → *Id_autora*, *Nazwisko*, *Imię*, *Narodowość*, *Okres tworzenia*, *Rodzaj twórczości*, *Język tworzenia*, *Osiągnięcia*.

1.4.8. Prawidłowy projekt bazy danych

Prawidłowy projekt bazy danych jest bardzo istotny dla efektywnej pracy, dlatego warto poświęcić trochę czasu, aby opanować zasady projektowania bazy.

Dobry projekt nie powinien zawierać powtarzających się danych. Aby osiągnąć ten cel, musimy podzielić dane na wiele tabel. Następnie powinniśmy zdefiniować połączenia między tabelami, aby można było tworzyć zestawienia danych pochodzących z różnych tabel. Na przykład w bazie danych *Księgarnia* podzieliliśmy dane na oddzielne zbiory (*Klient*, *Zamówienia*, *Rejestr*, *Książka*, *Autor*, *Faktura*), następnie zdefiniowaliśmy połączenia między tabelami, aby utworzyć zestawienie dotyczące realizacji zamówienia (*Nazwisko*, *Imię*, *Tytuł książki*, *Nazwisko autora*, *Cena książki*, *Liczba egzemplarzy*).

Proces projektowania bazy danych składa się z następujących kroków:

- **Określenie celu, jakiemu ma służyć baza danych.** Baza danych może na przykład służyć do gromadzenia informacji na temat sprzedaży książek, do wystawiania faktur dotyczących sprzedaży, do modyfikowania na bieżąco tych danych, do przetwarzania zgromadzonych danych.
- **Określenie zakresu potrzebnych informacji.** Należy określić, jakie informacje będą przechowywane w bazie, na przykład: nazwisko i imię klienta oraz jego dane osobowe, tytuły książek, informacje o autorach, informacje na temat realizacji zamówień.
- **Podzielenie informacji na tabele.** Zebrane informacje należy podzielić według tematów i dla każdego przewidzieć oddzielną tabelę, na przykład *Klient*, *Książki*.
- **Podzielenie elementów informacji na kolumny.** Trzeba zdecydować, jakie informacje mają być przechowywane w poszczególnych tabelach. Każdy element informacji zostanie przypisany do kolumny, na przykład tabela *Klient* będzie zawierała kolumny *Nazwisko klienta* i *Adres*.
- **Wybranie kluczy podstawowych.** Należy wybrać klucz podstawowy dla każdej tabeli, na przykład w tabeli *Klient* może to być identyfikator przypisany do każdego klienta lub PESEL.
- **Zastosowanie reguł normalizacji.** Za pomocą reguł normalizacji można sprawdzić, czy tabele mają prawidłową strukturę.

- **Poprawienie projektu.** Po sprawdzeniu, jeżeli to konieczne, trzeba skorygować projekt bazy.
- **Utworzenie relacji pomiędzy tabelami.** Należy przejrzeć projekt i zdecydować, jakie relacje powinny znaleźć się w bazie.

Po zaprojektowaniu bazy danych zgodnie z podanymi regułami można przystąpić do jej tworzenia, korzystając z aplikacji przeznaczonych do obsługi relacyjnych baz danych.

Przykład 1.3

Podczas analizowania przeznaczenia bazy danych tworzymy jej strukturę. Jeżeli w bazie danych dla księgarni zmienimy jej przeznaczenie, może okazać się, że tabela *Autor* nie jest potrzebna. Natomiast konieczne jest sporządzanie dla każdej sprzedaży faktury. Wtedy niezbędna będzie tabela do przechowywania informacji, które powinny znaleźć się na fakturze.

Baza danych mogłaby składać się z następujących tabel:

Klient → *Id_klienta*, *Nazwisko*, *Imię*, *Kod pocztowy*, *Miejscowość*, *Ulica*, *Nr domu*, *PESEL*, *Telefon*, *Adres e-mail*.

Zamówienia → *Id_zamówienia*, *Id_klienta*, *Data złożenia zamówienia*, *Data wysłania*, *Koszt wysyłki*, *Numer faktury*.

Rejestr → *Id_zamówienia*, *Id_książki*, *Liczba egzemplarzy*.

Książka → *Id_książki*, *Tytuł*, *Nazwisko i imię autora*, *Cena książki*, *Wydawnictwo*, *Rodzaj literatury*, *Miejsce wydania*, *Język książki*, *Opis*.

Faktura → *Numer faktury*, *Sposób płatności*, *Data wystawienia faktury*.

Ćwiczenie 1.1

Firma Globtroter wynajmuje klientom autokary do przewozu osób. Posiada flotę dobrze wyposażonych autokarów oraz zatrudnia grupę kierowców z najwyższymi umiejętnościami. Utwórz projekt graficzny bazy danych, która będzie służyła do rejestracji usług świadczonych przez tę firmę. W bazie danych powinna znaleźć się informacja o posiadanych autokarach, pracujących w firmie kierowcach oraz klientach korzystających z usług firmy. Każda usługa zamówienia może dotyczyć jednego lub kilku autokarów i powinna zostać zarejestrowana w bazie z informacją, dla kogo realizowane jest zamówienie, które autokary zostaną wysłane na trasę i którzy kierowcy będą je obsługiwali.

Rozwiązanie

Określenie potrzebnych informacji:

Przechowywane informacje o autokarach: model, liczba miejsc, rok produkcji, pojemność silnika, spalanie, stawka za 1 km.

Przechowywane informacje o kierowcach: nazwisko, imię, PESEL, adres, telefon, kategoria prawa jazdy, data zatrudnienia, doświadczenie.

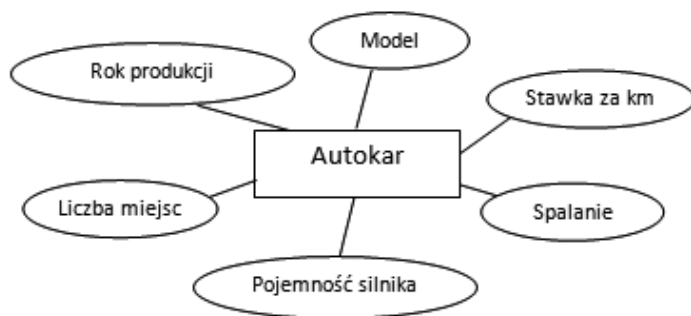
Przechowywane informacje o klientach: nazwisko, imię, PESEL, adres, telefon.

Przechowywane informacje o realizowanych usługach: cel, liczba km, data realizacji, opłata za wynajęcie.

Przechowywane informacje o fakturach: nr faktury, data wystawienia, sposób płatności.

Tworzenie diagramu ERD:

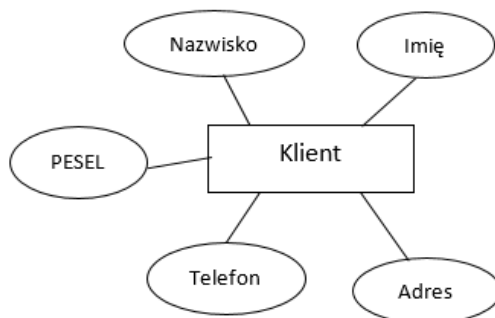
Tworzone zbiory encji zostały pokazane na rysunkach 1.36–1.40.



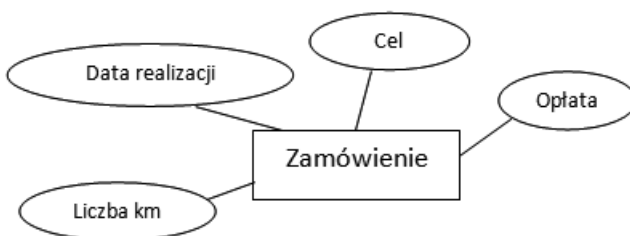
Rysunek 1.36. Zbiór encji Autokar



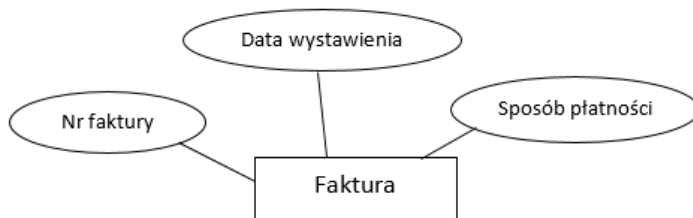
Rysunek 1.37. Zbiór encji Kierowca



Rysunek 1.38. Zbiór encji Klient



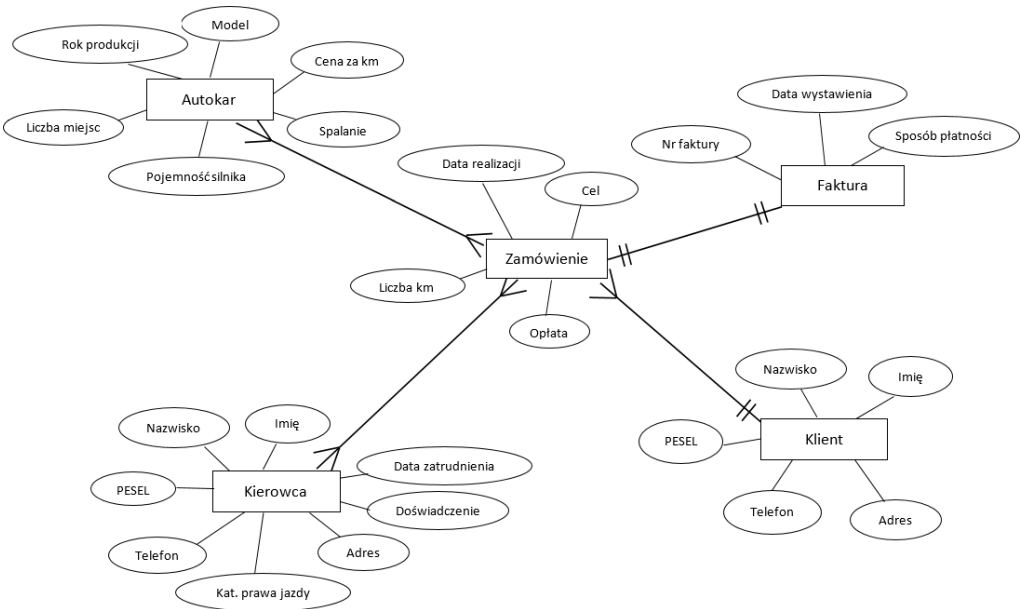
Rysunek 1.39. Zbiór encji Zamówienie



Rysunek 1.40. Zbiór encji Faktura

Definiowanie związków między zbiorami encji:

Tworzone związki zostały pokazane na rysunku 1.41.



Rysunek 1.41. Związki między zbiorami encji bazy danych dla firmy Globtroter

Ćwiczenie 1.2

Na podstawie utworzonego w ćwiczeniu 1.1 projektu graficznego bazy danych dla firmy Globtroter zaprojektuj w modelu relacyjnym tabele do przechowywania danych oraz tabele opisujące związki. Umieść w nich wszystkie atrybuty zdefiniowanych encji oraz atrybuty opisujące związki zachodzące między encjami.

Rozwiązanie

Przykładowy zestaw tabel dla bazy danych firmy Globtroter:

Klient → *Id_klienta*, *Nazwisko*, *Imię*, *Kod pocztowy*, *Miejscowość*, *Ulica*, *Nr domu*, *PESEL*, *Telefon*.

Zamówienia → *Id_zamówienia*, *Id_klienta*, *Data realizacji*, *Cel*, *Oplata*, *Liczba km*, *Nr faktury*.

Kierowca → *Id_kierowcy*, *Nazwisko*, *Imię*, *Kod pocztowy*, *Miejscowość*, *Ulica*, *Nr domu*, *PESEL*, *Telefon*, *Data zatrudnienia*, *Doświadczenie*, *Kat. prawa jazdy*.

Autokar → *Id_autokaru*, *Model*, *Rok produkcji*, *Liczba miejsc*, *Pojemność silnika*, *Spalanie*, *Stawka za km*.

Faktura → *Nr faktury*, *Sposób płatności*, *Data wystawienia*.

Zamówienie-Kierowca → *Id_zamówienia*, *Id_kierowcy*.

Zamówienie-Autokar → *Id_zamówienia*, *Id_autokaru*.

1.4.9. Typy danych

Projektując bazę danych, zwracamy uwagę na to, aby wartości przechowywane w polach były wartościami elementarnymi (atomowymi). W praktyce wymóg ten jest realizowany przez określenie typu pola.

Teoria relacyjnych baz danych mówi, że wszystkie wartości danych oparte są na prostych typach danych. Oznacza to, że dla każdego pola powinien zostać wybrany typ danych najmniejszy z możliwych, który spełni związane z nim wymagania projektowe. Typ danych określi, jaki rodzaj informacji może zostać zapisany w wybranym polu. Poza tym definiowanie typów danych znacząco wpływa na optymalizację działania bazy danych.

Jeżeli w tabeli występuje pole *Imię*, dla którego wybierzemy typ `char(100)` — typ tekstowy o długości 100 znaków — to na każde imię zostanie zarezerwowane 100 bajtów. Przyjmując, że najdłuższe imię nie zawiera więcej niż 20 znaków, dodatkowo rezerwujemy 80 bajtów, które nigdy nie zostaną wykorzystane. Gdy w tabeli wystąpi 200 tysięcy rekordów, to zajmą one kilkanaście GB nigdy niewykorzystanej pamięci.

To samo dotyczy pola *Data_urodzenia*. Jeżeli zostanie wybrany typ `date`, to zajmie on 3 bajty, jeżeli zostanie zastosowany typ `datetime`, to dla każdego rekordu zostanie zarezerwowane 8 bajtów na datę urodzenia.

Typy danych stosowane w językach komunikowania się z bazą danych można podzielić na kilka grup. Wybrane typy zostały pokazane w tabeli 1.1.

Tabela 1.1. Typy danych stosowane w MS SQL i MySQL

| Grupa | Typy w MS SQL | Typy w MySQL |
|--------------|---|---|
| Tekstowe | char, varchar, nchar, ntext, nvarchar | char, varchar |
| Liczbowe | int, smallint, bigint, tinyint, float, real, decimal, numeric | int, smallint, bigint, tinyint, float, real, decimal, numeric |
| Daty i czasu | date, datetime | date, datetime, timestamp, time |
| Binarne | binary, varbinary | binary, varbinary |
| Waluty | money, smallmoney | |
| Specjalne | text, image, bit | text, bit, enum, set |

Typy tekstowe

Typy tekstowe to pola, które mogą zawierać litery, liczby i symbole. Po zadeklarowaniu długości można przechowywać teksty o określonej liczbie znaków.

Typ char(długość) — można przechowywać do 255 znaków. Jest to pole o stałej liczbie znaków, np. `char(10)`. Jeżeli tekst jest krótszy, zostanie uzupełniony do zadeklarowanej

długości spacjami. Jeśli tekst jest dłuższy niż zadeklarowana liczba znaków, zostanie obcięty.

Typ varchar(długość) — można przechowywać do 255 znaków. Jest to pole o zmiennej liczbie znaków, np. `varchar(10)`. Jeżeli tekst jest krótszy niż zadeklarowana długość, nie jest uzupełniany spacjami, co powoduje zmniejszenie zasobów pamięci zajmowanych przez bazę danych. Przy deklarowaniu typów pól zaleca się stosowanie typów o zmiennej liczbie znaków.

Typ nvarchar(długość) — można przechowywać do 255 znaków. Ten typ działa w standardzie Unicode, który zapisuje pojedynczy znak na 2 bajtach. Jest to związane z kodowaniem znaków narodowych. Powoduje to zwiększenie objętości bazy danych, ale jeżeli będzie ona funkcjonowała w środowisku wielojęzycznym, należy zastosować typ, który będzie poprawnie interpretował znaki narodowe, np. `nvarchar(10)`.

Typ ntext(długość) — pozwala na wprowadzanie tekstu o wielkości do 2 GB. Typ stosowany, jeśli w polu będzie przechowywany tekst o dużej liczbie znaków.

Typy liczbowe

Typy liczb całkowitych

Jeżeli w polu będą wprowadzane tylko liczby całkowite, powinien zawsze być wybierany typ liczb całkowitych. Typy całkowite różnią się tylko liczbą bajtów przeznaczonych na zapisanie liczby, a więc również zakresem liczb, które mogą zostać zapisane.

Typ tinyint — 1 bajt, zapis liczb z zakresu od -128 do 127 lub bez znaku z zakresu od 0 do 255.

Typ smallint — 2 bajty, zapis liczb z zakresu od -32 768 do 32 767 lub bez znaku z zakresu od 0 do 65 535.

Typ int — 4 bajty, zapis liczb z zakresu od -2 147 483 648 do 2 147 483 647 lub bez znaku z zakresu od 0 do 4 294 967 295.

Typ bigint — 8 bajtów, zapis liczb z zakresu od -2^{63} do $2^{63}-1$.

Typy liczb rzeczywistych

Jeżeli w polu będą przechowywane liczby, dla których należy dokładnie określić liczbę miejsc po przecinku, powinien zostać zastosowany typ `decimal` lub `numeric`.

Typ numeric(precyzja, skala) — dla tego typu należy podać dwa parametry: precyzję i skalę. Pierwszy parametr określa całkowitą liczbę cyfr, drugi mówi, ile cyfr znajduje się po przecinku, np. `numeric(4, 2)`. W podanym przykładzie możliwe będzie przechowywanie liczb nie większych od 9 999 z dokładnością do dwóch miejsc po przecinku. Jeśli nie podamy tych parametrów, zostaną one ustawione automatycznie na wartości 10 jako precyzja i 0 jako skala.

Typ decimal(precyzja, skala) — podobnie jak dla typu `numeric` należy podać dwa parametry: precyzję i skalę.

Jeżeli dokładność przechowywanych w bazie liczb jest mniej istotna, natomiast najważniejsza jest ich rozpiętość (np. dane statystyczne), należy stosować typ `float` lub `real`.

Typ `float(n)` — 4 bajty lub 8 bajtów, liczba zmiennoprzecinkowa. Parametr *n* określa precyzję w bitach. Jeśli *n* jest w zakresie od 1 do 24, to mamy do czynienia z pojedynczą precyzją (dokładność do około 7 cyfr po przecinku), jeżeli wynosi od 25 do 53, to mamy do czynienia z podwójną precyzją (dokładność do około 15 cyfr po przecinku). Używanie typu `float` może powodować problemy przy wykonywaniu obliczeń.

Typ `float(m,d)` — liczba zmiennoprzecinkowa o pojedynczej precyzji. Typ ten jest niestandardowym rozszerzeniem MySQL.

Typ `real` — 4 bajty, liczba zmiennoprzecinkowa. Definiowana podobnie jak typ `float`.

Jeżeli nie jest to konieczne, należy unikać używania typów przybliżonych. Jeżeli jest to możliwe, należy stosować typy `decimal` lub `numeric`.

Typy daty i czasu

Typ `datetime` — 8 bajtów, zapis daty i czasu z dokładnością do sekundy w postaci RRRR-MM-DD HH:MM:SS.

Typ `date` — 3 bajty, zapis daty w postaci RRRR-MM-DD.

Typ `time` — 3 bajty, zapis czasu w postaci HH:MM:SS.

Typ daty powinien być dostosowany do rodzaju przechowywanej informacji. Na ogół nie ma potrzeby przechowywania zarówno daty, jak i czasu. Jeśli chcemy przechowywać datę urodzin, w zupełności wystarczy pole typu `date`. Jeśli wymagane jest przechowywanie tylko roku urodzin, wystarczy zastosować pole typu `int`. Dzięki temu zostanie zaoszczędzona znaczna ilość miejsca w pamięci i uzyskamy większą efektywność działań na danych.

Typy binarne

Typy binarne służą do przechowywania danych binarnych. Dane reprezentowane są za pomocą liczb heksadecymalnych.

Typ `binary` — do przechowywania danych binarnych o stałej długości, długość maksymalna 8000 bajtów.

Typ `varbinary` — do przechowywania danych binarnych o zmiennej długości, długość maksymalna do 8000 bajtów.

Typy walutowe

Typy walutowe służą do przechowywania wartości walutowych. Mogą również służyć do przechowywania wartości innych niż wartości walutowe.

Typ `money` — 8 bajtów, do przechowywania wartości walutowych z zakresu od -2^{31} do $2^{31}-1$ z dokładnością do jednej dziesięciotysięcznej (cztery cyfry po przecinku).

Typ smallmoney — 4 bajty, do przechowywania wartości walutowych z zakresu od -2^{63} do $2^{63}-1$ z dokładnością do jednej dziesięciotysięcznej (cztery cyfry po przecinku).

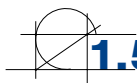
Typy specjalne

Typ text — do przechowywania łańcuchów tekstowych o zmiennej długości. Może przechowywać do 2 GB danych.

Typ image — do przechowywania danych binarnych o zmiennej długości. Może przechowywać do 2 GB danych.

Mimo że typy *text* i *image* pozwalają na przechowywanie dużej ilości informacji, powinniśmy zastanowić się, czy lepszym rozwiązaniem nie byłoby umieszczenie pliku z tą informacją poza bazą, a w bazie danych przechowywanie jedynie ścieżki do pliku.

Typ bit — 1 bit, do przechowywania wartości logicznych w postaci *0* lub *1*, *włączony/wyłączony*, *tak/nie*.



1.5. Cechy relacyjnej bazy danych

Baza danych powinna charakteryzować się następującymi cechami:

- trwałość danych,
- integralność danych,
- bezpieczeństwo danych,
- współdzielenie danych,
- abstrakcja danych,
- niezależność danych,
- integracja danych.

1.5.1. Trwałość danych

Trwałość danych zapisanych w bazie jest podstawową cechą baz danych. Trwałość danych oznacza, że dane zostały zapisane w bazie w sposób nieulotny. Wszystkie współczesne systemy baz danych muszą spełniać ten wymóg. Trwałość danych jest niezależna od działania aplikacji oraz od platformy sprzętowej i programowej. Dane gromadzone w bazie danych są przechowywane w pamięci zewnętrznej (dyskowej, optycznej, taśmowej). Powinny one być przechowywane w pamięci dopóty, dopóki wymagają tego użytkownicy systemu baz danych.

1.5.2. Integralność, czyli poprawność danych

Integralność (spójność) danych oznacza, że dane muszą:

- wiernie odzwierciedlać dane rzeczywiste (dane są prawdziwe oraz są aktualizowane, gdy ulega zmianie rzeczywistość),

- spełniać ograniczenia nałożone przez użytkowników (istnieje system kontroli danych wejściowych),
- wykazywać brak anomalii wynikających ze współbieżnego dostępu do danych (istnieją mechanizmy, które zabezpieczają dane przed błędami pojawiającymi się podczas współbieżnego dostępu do nich).

Integralność bazy danych to także:

- odporność na błędy i awarie wynikające z zawodności sprzętu i oprogramowania (istnieją mechanizmy, które zabezpieczają dane przed skutkami awarii sprzętu i oprogramowania),
- odporność na błędy użytkowników (istnieją mechanizmy, które zabezpieczają dane przed następstwami błędów logicznych).

Na poziomie danych wyróżniamy dwa rodzaje integralności danych:

- *semantyczna* (spójność logiczna) — oznacza zgodność danych z rzeczywistością, czyli poprawność odwzorowania rzeczywistości (baza danych odpowiada zaprojektowanemu schematowi i zadeklarowanym ograniczeniom),
- *bazowa* (spójność fizyczna) — oznacza poprawność procesów zachodzących w bazie (operacje wykonywane w bazie danych kończą się sukcesem).

Na poziomie struktur bazy danych można wyróżnić następujące rodzaje integralności:

- *referencyjna* — odnosi się do powiązań między tabelami i oznacza, że każdej wartości klucza obcego odpowiada dokładnie jedna wartość klucza podstawowego;
- *encji* — odnosi się do schematu bazy danych i oznacza, że każda encja musi posiadać klucz podstawowy, czyli pole lub zbiór pól o wartościach unikatowych i niebędących wartością NULL. Z tego wynika, że w danej tabeli nie mogą istnieć dwa identyczne wiersze;
- *atrybutu* — oznacza, że wartość każdego atrybutu należy do jego dziedziny.

1.5.3. Bezpieczeństwo danych

Bezpieczeństwo danych oznacza, że dostęp do bazy danych mają tylko jej użytkownicy identyfikowani unikatową nazwą (loginem) i hasłem. Ponadto każdy użytkownik posiada określone uprawnienia w bazie danych.

Podstawowym sposobem zapewnienia bezpieczeństwa danych jest stosowanie procedur uwierzytelniania oraz ograniczania uprawnień dostępu do danych. Sposobem pośrednim może być szyfrowanie danych oraz ograniczenie fizycznego dostępu do systemu komputerowego.

Bezpieczeństwo danych oznacza również, że baza danych jest zabezpieczona przed awarią sprzętu lub oprogramowania.

1.5.4. Współdzielenie danych

Współdzielenie danych oznacza, że istnieje możliwość równoczesnej pracy wielu użytkowników z tą samą bazą danych. Użytkownicy mogą też jednocześnie pracować z tym samym zbiorem danych. Skutkiem takiej pracy mogą być konflikty w dostępie do danych, gdy jeden użytkownik modyfikuje zbiór danych, a drugi próbuje ten sam zbiór odczytać lub zmodyfikować w inny sposób. W bazie danych muszą istnieć mechanizmy zapewniające poprawne rozwiązanie takich konfliktów (np. stosowanie mechanizmów transakcji i współbieżności).

1.5.5. Abstrakcja danych

Abstrakcja to uogólnienie (uproszczenie) rozpatrywanego problemu, które polega na wyodrębnieniu wspólnych jego cech.

Baza danych zawiera pewien model rzeczywistości. Ale przechowywane są w niej tylko niektóre dane o obiektach. Powinny one opisywać wyłącznie istotne cechy obiektów świata rzeczywistego. Baza danych to abstrakcyjny model pewnego wycinka rzeczywistości, a jej struktura powinna poprawnie odzwierciedlać obiekty świata rzeczywistego i powiązania pomiędzy tymi obiektami.

Wyodrębniamy trzy poziomy abstrakcji:

- *poziom wewnętrzny* — określa sposób pamiętania danych;
- *poziom pojęciowy (konceptualny)* — wyższy poziom abstrakcji danych. Definiuje dane oraz związki zachodzące między nimi (przedstawia on logiczną strukturę bazy danych);
- *poziom zewnętrzny* — najwyższy poziom abstrakcji danych. Opisuje sposób, w jaki dane są widziane przez użytkownika. Dla każdego użytkownika obraz danych definiuje się za pomocą schematu zewnętrznego (podschematu).

Dla tej samej bazy danych istnieje jeden model pojęciowy i wiele schematów zewnętrznych.

1.5.6. Niezależność danych

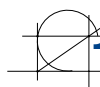
Niezależność danych polega na oddzieleniu danych od aplikacji, które używają tych danych. Niezależność danych może być rozpatrywana w dwóch aspektach:

- *Logiczna niezależność danych* — niezależność danych widzianych przez użytkownika (przez jego aplikacje) od logicznej struktury bazy danych (schematu pojęciowego) oraz zmian tej struktury w czasie.
- *Fizyczna niezależność danych* — niezależność logicznej struktury bazy danych (schematu pojęciowego) od sposobu przechowywania danych w pamięci zewnętrznej (nośnikach zewnętrznych).

We współczesnych bazach danych niezależność danych jest osiągana tylko częściowo.

1.5.7. Integracja danych

Integracja danych polega na zapewnieniu współpracy danych przechowywanych w różnych miejscach i obsługiwanych przez różne systemy. Często bazy danych fizycznie są umieszczane na różnych komputerach połączonych ze sobą w taki sposób, że użytkownik nie wie, iż dane, z którymi pracuje, pochodzą z różnych baz i komputerów. Zmiany zawartości bazy na jednym komputerze powinny być uwzględniane również na innych komputerach. Tak funkcjonujące bazy nazywamy *rozproszonymi bazami danych*. Bazy danych powinny być zaprojektowane tak, aby dane były zawsze dostępne, niezależnie od tego, gdzie i w jakiej postaci zostały zapisane i jak są przechowywane. Serwery bazodanowe powinny zapewniać integrację rozproszonych danych znajdujących się na wielu komputerach. Mechanizmy integracyjne dostępne na nich powinny działać w czasie rzeczywistym i współpracować z różnorodnymi bazami danych.



1.6. Pytania i zadania

1.6.1. Pytania

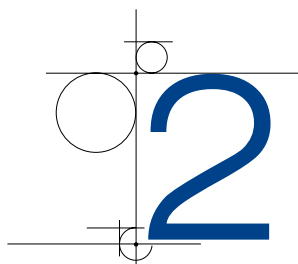
1. Podaj zalety korzystania z komputerowych baz danych.
2. Podaj definicję bazy danych.
3. Omów poznane modele baz danych.
4. Omów występujące w modelu relacyjnym rodzaje więzów integralności.
5. Podaj podstawowe cechy relacyjnego modelu baz danych.
6. Podaj definicję klucza podstawowego.
7. Jakiego typu relacje mogą wystąpić w bazie danych?
8. Co oznacza występujące w bazach danych pojęcie *encja*?
9. Do czego służą diagramy *ERD*?
10. Jakie zastosowanie w projektowaniu bazy danych mają narzędzia *CASE*?
11. W jaki sposób w tabelach opisywany jest związek „wiele do wielu”?
12. Na czym polega normalizacja tabel?
13. Wymień cechy, którymi powinna charakteryzować się baza danych.
14. Na czym polega integralność referencyjna bazy danych?

1.6.2. Zadania

Zadanie 1.

Zaprojektuj zgodnie z poznanymi zasadami bazę danych dotyczącą Twojej szkoły. Nazwij ją na przykład *Moja szkoła*. Umieść w niej informacje na temat uczniów, klas, przedmiotów i nauczycieli, sal lekcyjnych i pracowników. Określ funkcje tworzonej bazy danych oraz zdefiniuj zbiory przechowywanych informacji. Wykorzystując diagramy ERD, opracuj model graficzny schematu bazy danych. Zaprojektuj tabele i sprawdź za pomocą reguł normalizacji, czy mają one prawidłową strukturę.





Systemy zarządzania bazą danych

2.1. Wprowadzenie

Bazy danych są obsługiwane przez złożone systemy zawierające zbiory gotowych narzędzi zapewniających dostęp do danych. Umożliwiają one manipulowanie danymi zgromadzonymi w systemach komputerowych i aktualizowanie tych danych. Są to systemy zarządzania bazą danych, w skrócie nazywane SZBD (ang. *Database Management Systems*).

Do najważniejszych zadań SZBD można zaliczyć:

- operowanie na dużych i bardzo dużych zbiorach danych,
- zarządzanie złożonymi strukturami.

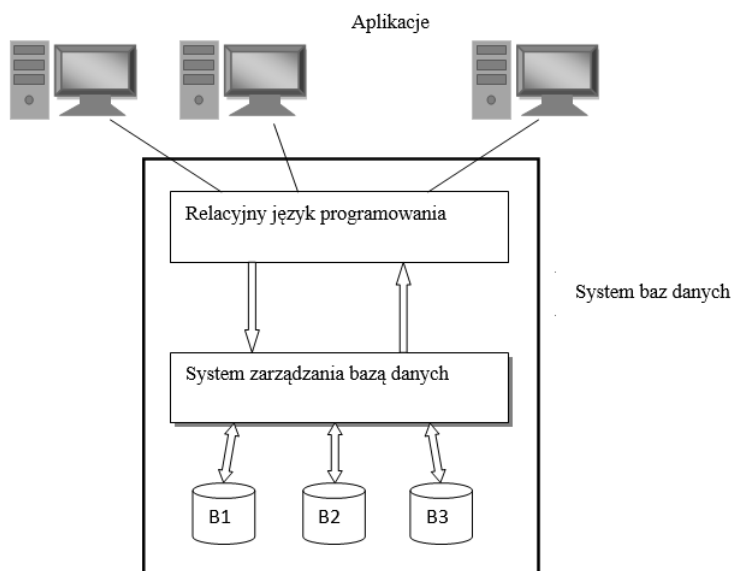
2.1.1. Architektura systemu baz danych

W skład systemu baz danych wchodzi, oprócz baz danych, także system zarządzania bazą danych oraz język komunikowania się z bazą (rysunek 2.1).

Interakcja programu użytkowego (aplikacji) z relacyjną bazą danych odbywa się za pomocą języka SQL. Jest to jedyny sposób komunikowania się aplikacji z bazą danych.

Język SQL jest narzędziem dostępu do bazy danych stosowanym głównie przez projektantów aplikacji, projektantów baz danych i administratorów baz danych. Użytkownicy komunikują się z bazą danych za pomocą aplikacji, a następnie aplikacje komunikują się z bazą danych za pomocą poleceń SQL.





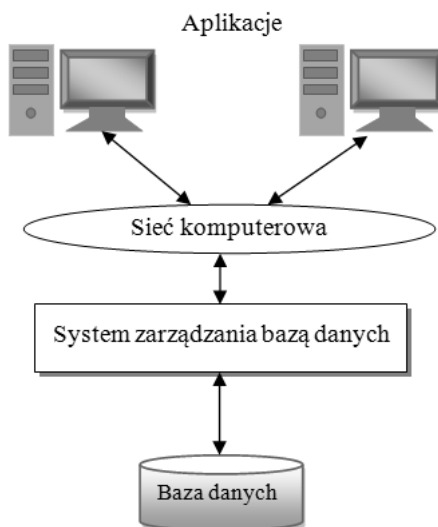
Rysunek 2.1. Architektura systemu baz danych

Dwa popularne sposoby łączenia z bazą danych to:

- architektura klient-serwer,
- architektura 3-warstwowa.

Architektura klient-serwer

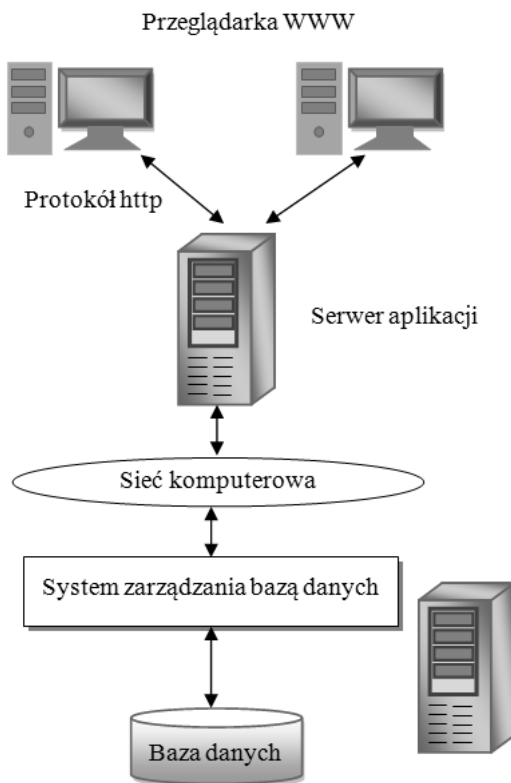
W architekturze klient-serwer aplikacje zainstalowane na stacjach użytkowników komunikują się z bazą danych, wykorzystując sieciowe oprogramowanie przeznaczone do komunikacji z systemem zarządzania bazą danych (rysunek 2.2).



Rysunek 2.2. Architektura komunikacyjna klient-serwer

Architektura 3-warstwowa

W architekturze 3-warstwowej pomiędzy użytkownikami a serwerem bazy danych znajduje się tzw. serwer aplikacji, który udostępnia umieszczone na nim aplikacje. Jest to architektura typowa dla aplikacji WWW. Aplikacje są udostępniane przez serwer aplikacji w postaci stron internetowych. Użytkownik komunikuje się z bazą danych przez przeglądarkę stron WWW. W odpowiedzi na polecenia użytkownika serwer aplikacji wysyła odpowiednie żądania do systemu zarządzania bazą danych, który wykonuje polecenia i przesyła ich wyniki do serwera aplikacji. Serwer aplikacji przesyła te wyniki do aplikacji użytkownika (rysunek 2.3).



Rysunek 2.3. Architektura komunikacyjna 3-warstwowa

2.1.2. System zarządzania bazą danych

System zarządzania bazą danych (ang. *Database Management System* — DBMS) decyduje o sposobie pobierania i przechowywania danych w relacyjnych bazach danych. Może on udostępniać bazę lokalnie na jednym komputerze (*bazy jednostanowiskowe*) lub może być serwerem bazy danych udostępniającym zasoby komputerom połączonym w sieć (*bazy typu klient-serwer*).

System zarządzania bazą danych musi posiadać mechanizmy, które pozwalają administrować zbiorami danych umieszczonymi w bazie, zapewniają bezpieczeństwo i integralność danych, umożliwiają dostęp do danych za pomocą języka SQL, zapewniają wielodostępność danych (na przykład przez transakcje) oraz pozwalają na autoryzację dostępu do danych.

Często takie systemy posiadają narzędzia do przechowywania i przetwarzania danych multimedialnych, narzędzia do konwersji danych w celu współpracy z innymi systemami baz danych, graficzne środowiska do tworzenia aplikacji oraz narzędzia do udostępniania bazy danych w internecie.

W systemach zarządzania bazą danych można wyodrębnić dwa elementy:

- **serwer** — przechowuje dane, umożliwia ich pobieranie i aktualizowanie oraz zapewnia ich integralność i bezpieczeństwo;
- **oprogramowanie pośredniczące** (ang. *middleware*) — realizuje komunikację między użytkownikiem a serwerem. Wyposażone jest w mechanizmy pozwalające wykorzystywać pobierane dane, na przykład w narzędzia do tworzenia i obsługi formularzy i raportów.

Systemy zarządzania bazą danych zwykle działają w trybie *klient-serwer*, czyli baza umieszczona na serwerze jest udostępniana klientom poprzez oprogramowanie pośredniczące (systemy bazodanowe). Przykładami takich systemów są: MySQL, MS SQL Server, Oracle, PostgreSQL, DB2.

Istnieją również systemy, które nie uwzględniają podziału typu klient-serwer. Przykładem takiego systemu jest poznany wcześniej Microsoft Access.

System zarządzania bazą danych (serwer bazodanowy), który należy do architektury klient-serwer, składa się z dwóch współpracujących ze sobą części. Jedna część, działająca na serwerze, odpowiedzialna jest za wydajność, bezpieczeństwo, kopie zapasowe itp. Druga, działająca po stronie klienta, zapewnia interfejs użytkownika, dzięki któremu możliwe jest przeprowadzanie operacji na bazie danych. Serwer bazodanowy komunikuje się z bazą danych najczęściej za pomocą języka SQL.

2.2. Serwery bazodanowe

Do najpopularniejszych serwerów bazodanowych należą:

- SQL Server firmy Microsoft,
- Oracle Database firmy Oracle,
- MySQL — produkt powstały jako *open source*,
- PostgreSQL — darmowy serwer baz danych, utworzony na Uniwersytecie Kalifornijskim,
- DB2 firmy IBM.

Do pracy z bazami danych można wybrać dowolny serwer bazodanowy. Niezależnie od tego, czy jest to produkt darmowy, czy komercyjny, zestaw podstawowych elementów języka SQL w większości z nich jest podobny. Z kolei każdy serwer zawiera niestandardowe rozszerzenia języka SQL.

2.2.1. Serwer MySQL

MySQL to bardzo wydajny i stabilny serwer o małych wymaganiach sprzętowych. Charakterystyczne cechy serwera MySQL to:

- możliwość działania na niemal wszystkich dostępnych platformach;
- udostępnianie różnych silników bazodanowych (na przykład bardzo szybkie tabele MyISAM, tabele InnoDB — standard w nowych wersjach MySQL lub tabele HEAP);
- podstawowa implementacja złączeń;
- wykorzystywanie systemu przesyłania skompresowanych danych pomiędzy klientem i serwerem;
- udostępnianie serwera w postaci osobnego programu lub biblioteki;
- obsługa zapytań rozproszonych;
- udostępnianie mechanizmów replikacji.

Zalety serwera MySQL:

- jest bardzo szybki, nadaje się do obsługi często odwiedzanych stron WWW;
- ma niewielkie wymagania sprzętowe;
- jest darmowy i nielimitowany.

Wada serwera MySQL:

- transakcje wymagają korzystania z silnika bazodanowego InnoDB, ponieważ MyISAM nie obsługuje transakcji.

Instalowanie serwera w systemie Ubuntu

Większość dystrybucji systemu Linux (na przykład Ubuntu) zawiera pakiet MySQL. Może on być standardowo dostarczony w trakcie instalacji systemu lub zainstalowany przez użytkownika.

Aby zainstalować serwer MySQL, należy w terminalu wpisać następujące polecenie:

```
sudo apt-get install mysql-server
```

Podczas instalacji pojawi się pytanie o ustawienie hasła do konta root serwera MySQL. Jeżeli takie pytanie nie pojawi się, to po zakończeniu instalacji należy ręcznie stworzyć hasło dla administratora, wpisując:

```
mysqladmin -u root -p password hasło
```

Do nawiązania połączenia z serwerem można użyć polecenia:

```
mysql -u root -h localhost -p(hasło)
```

lub

```
mysql -u root -h (nazwa_hosta) -p(hasło)
```

Standardowym narzędziem do pracy z serwerem MySQL jest klient *mysql*. Po zakończeniu instalowania najprawdopodobniej będzie trzeba jeszcze tę usługę uruchomić (zależy to od wybranych opcji instalacyjnych). W celu uruchomienia usługi *mysql* należy otworzyć okno konfiguracji usług, na liście znaleźć usługę *mysqld* i włączyć ją, klikając przycisk *Włącz*. W trakcie konfigurowania serwera MySQL trzeba ustawić hasło dla administratora bazy oraz założyć na serwerze bazę danych i konto użytkownika do bieżącej pracy.

Aby zainstalować serwer MySQL w systemie Ubuntu 18.04, należy w wierszu poleceń wpisać:

```
sudo apt-get install mysql-server
```

System baz danych zostanie zainstalowany na komputerze. W celu zwiększenia bezpieczeństwa baz danych należy skonfigurować instalację serwera, wpisując polecenie:

```
sudo mysql_secure_installation
```

Zostanie uruchomiona aplikacja, która pozwoli na odpowiednie skonfigurowanie zabezpieczeń MySQL.

W celu sprawdzenia, czy serwer MySQL jest uruchomiony, należy w terminalu wpisać następujące polecenie:

```
sudo service mysql status
```

W wyniku wykonania polecenia zostanie wyświetlona informacja zbliżona do podanej niżej:

```
mysql start/running, process 1218
```

Jeżeli serwer MySQL nie działa prawidłowo, należy wpisać polecenie:

```
sudo service mysql restart
```

W celu zalogowania się jako administrator należy wpisać polecenie:

```
sudo mysql -u root -p
```

Dodanie nowego użytkownika nastąpi po wpisaniu polecenia:

```
CREATE USER nazwa_uzytkownika IDENTIFIED BY 'hasło_uzytkownika';
```

Utworzenie nowej bazy danych nastąpi po wpisaniu polecenia:

```
CREATE DATABASE nazwa_bazy;
```


Po wykonaniu tych czynności można przystąpić do pracy z bazą.

Do pracy z bazą można wykorzystywać utworzone konto użytkownika i utworzoną bazę, logując się do serwera za pomocą polecenia:

```
mysql -u nazwa_uzytkownika -phaslo nazwa_bazy
```

lub — bez wybierania bazy danych — za pomocą polecenia:

```
mysql -u nazwa_uzytkownika -p
```

Po uzyskaniu połączenia należy wybrać bazę danych, wpisując polecenie:

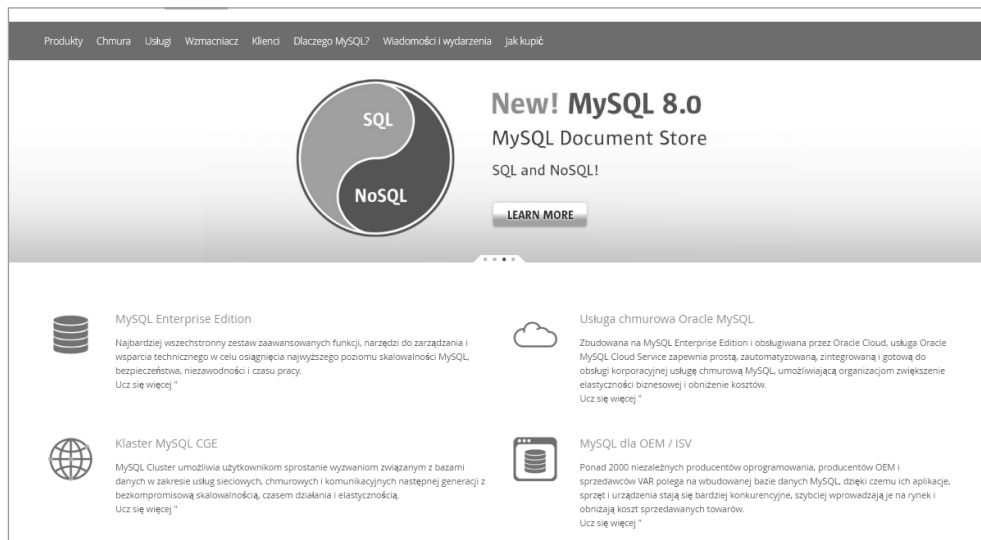
```
use nazwa_bazy;
```

Do rozłączenia się z serwerem służy polecenie:

```
quit
```

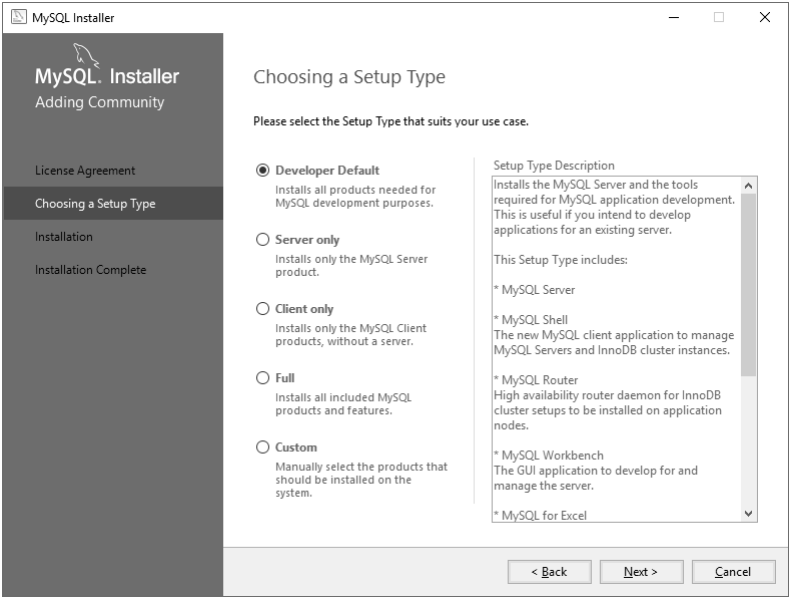
Instalowanie serwera MySQL w systemie Windows

Pakiet MySQL jest dostępny dla wszystkich wersji systemu Windows i proces jego instalowania dla każdej wersji jest w zasadzie taki sam. Aby zainstalować MySQL, trzeba ze strony <https://dev.mysql.com/downloads/mysql/> pobrać pakiet instalacyjny programu (rysunek 2.4).



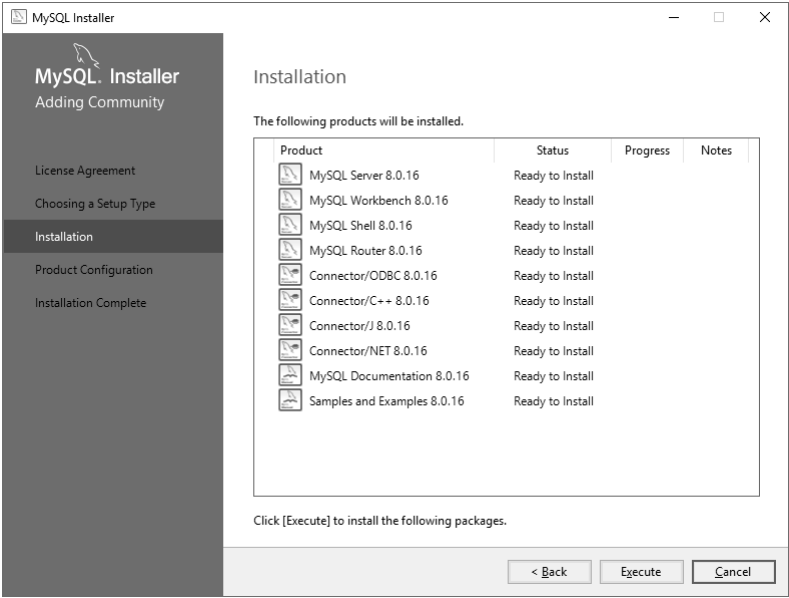
Rysunek 2.4. Pobieranie pakietu MySQL

Po jego pobraniu należy uruchomić plik instalacyjny *MySQL Installer* i po zaakceptowaniu umowy licencyjnej zaznaczyć domyślny sposób instalacji w systemie Windows (rysunek 2.5).



Rysunek 2.5. Wybór sposobu instalacji MySQL

W kolejnym oknie zostanie wyświetlona lista produktów MySQL, które zostaną zainstalowane (rysunek 2.6).



Rysunek 2.6. Lista produktów MySQL do zainstalowania

Należy wybrać przycisk *Execute* i rozpocząć instalowanie pakietu. Instalator poprowadzi nas przez cały proces instalacji. Po zakończeniu instalowania zostanie uruchomiony kreator konfiguracji, który pozwoli określić sposób uruchamiania usługi MySQL. W ostatnim oknie kreatora należy ustawić serwer MySQL jako usługę uruchamianą automatycznie.

Do serwera można zalogować się jako administrator, wpisując z konsoli systemowej polecenie:

```
mysql -u root -p
```

Po wpisaniu polecenia

```
CREATE DATABASE nazwa_bazy;
```

zostanie utworzona nowa baza danych.

Po wpisaniu polecenia

```
GRANT ALL ON nazwa_bazy.* TO nazwa_uzytkownika IDENTIFIED BY  
'haslo_uzytkownika';
```

zostanie utworzone konto użytkownika o podanej nazwie i podanym hasle. Użytkownik będzie miał dostęp do bazy, której nazwa została podana w klauzuli `GRANT ALL ON`. Będzie mógł na niej wykonywać wszystkie operacje.

Rozłączenie z serwerem nastąpi po wpisaniu polecenia:

```
quit
```

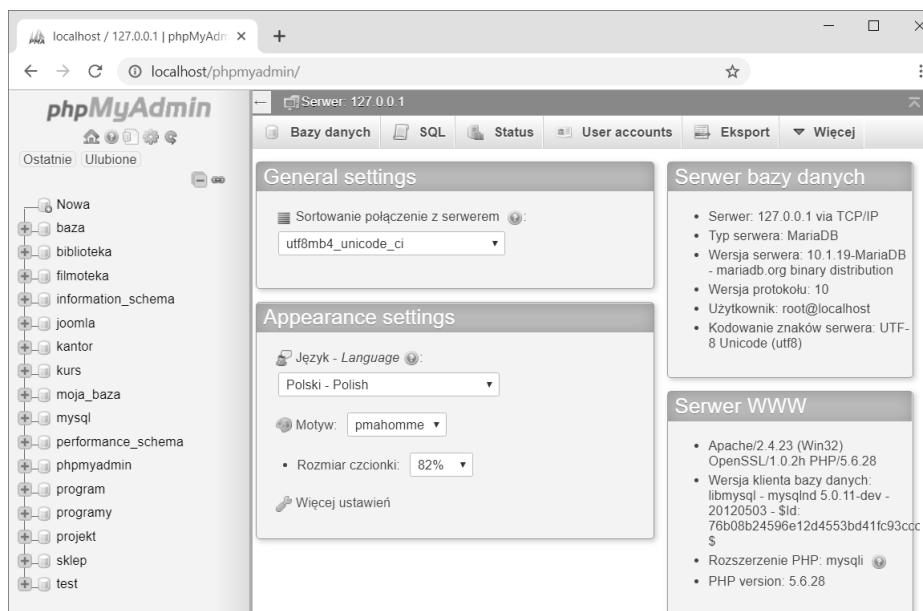
Pakiet XAMPP

Jeżeli w pracy z bazami danych planujemy wykorzystać serwer bazodanowy MySQL, najwygodniejszym rozwiązaniem może okazać się instalacja pakietu XAMPP. Pakiet ten zawiera między innymi serwer stron internetowych Apache, serwer baz danych MySQL, interpretery języków PHP i Perl oraz narzędzie phpMyAdmin. W systemach Linux taki zestaw aplikacji znany jest pod nazwą LAMP, a w systemie Windows występuje w formie pakietu pod nazwą WAMP. Istnieje również jego uniwersalna, wieloplatformowa wersja, nazywana XAMPP.

Narzędzie phpMyAdmin

Za pomocą narzędzia phpMyAdmin można z poziomu interfejsu graficznego zarządzać bazą danych MySQL. Umożliwia między innymi tworzenie i usuwanie baz danych, tworzenie tabel i relacji oraz modyfikowanie struktury tabel i ich zawartości. Wszystkie operacje na bazie danych są wykonywane z poziomu przeglądarki internetowej.

Aby rozpocząć pracę z phpMyAdmin, należy uruchomić XAMPP Control Panel i w otwartym oknie kliknąć przycisk **START** przy module Apache i MySQL. Następnie trzeba otworzyć przeglądarkę internetową i wpisać adres: <http://localhost/phpmyadmin/>. Zostanie uruchomiony phpMyAdmin (rysunek 2.7).



Rysunek 2.7. Okno narzędzia phpMyAdmin

Aby rozpocząć pracę z nową bazą danych, należy wybrać zakładkę **Bazy danych** i w polu **Utwórz bazę danych** wpisać nazwę tworzonej bazy, a następnie kliknąć przycisk **Utwórz**. Na serwerze zostanie utworzona pusta baza danych.

Aby utworzyć tabelę w bazie danych, należy w oknie nawigacyjnym zaznaczyć nazwę bazy danych, wybrać zakładkę **Struktura** i w polu **Utwórz tabelę** wpisać jej nazwę, określić liczbę kolumn i kliknąć przycisk **Wykonaj**. W otwartym oknie można zaprojektować strukturę tabeli, podając nazwy kolejnych kolumn oraz określając typy i właściwości pól (rysunek 2.8).

Nazwa tabeli: Add column(s) **Wykonaj**

| Nazwa | Typ | Długość/Wartości | Ustawienia domyślne | Metoda porównywania napisów | Atrybuty | Null | Indeks |
|---------------------------|-----|----------------------|---------------------|-----------------------------|----------------------|--------------------------|--------|
| <input type="text"/> | INT | <input type="text"/> | Brak | <input type="text"/> | <input type="text"/> | <input type="checkbox"/> | --- |
| Pick from Central Columns | | | | | | | |
| <input type="text"/> | INT | <input type="text"/> | Brak | <input type="text"/> | <input type="text"/> | <input type="checkbox"/> | --- |
| Pick from Central Columns | | | | | | | |

Rysunek 2.8. Tworzenie nowej tabeli

Po utworzeniu wszystkich pól tabeli należy zapisać jej strukturę — służy do tego przycisk *Wykonaj*. Po jego kliknięciu zostanie wyświetlone okno ze strukturą utworzonej tabeli (rysunek 2.9). W podobny sposób można tworzyć struktury kolejnych tabel.

| # | Nazwa | Typ | Metoda porównywania napisów | Atrybuty | Null | Ustawienia domyślne | Dodatkowo | Działanie |
|---|----------|-------------|-----------------------------|----------|------|---------------------|----------------|------------|
| 1 | Id_osoby | int(11) | | | Nie | Brak | AUTO_INCREMENT | Zmień Usuń |
| 2 | nazwisko | varchar(30) | | | Nie | Brak | | Zmień Usuń |
| 3 | imie | varchar(30) | | | Nie | Brak | | Zmień Usuń |

Rysunek 2.9. Struktura zaprojektowanej tabeli

Aby wprowadzić do tabeli dane, należy wybrać zakładkę *Wstaw* i w otwartym oknie wprowadzić dane (rysunek 2.10), a następnie kliknąć przycisk *Wykonaj*.

| Kolumna | Typ | Funkcja | Null | Wartość |
|----------|-------------|----------------------|------|------------|
| Id_osoby | int(11) | <input type="text"/> | | 1 |
| nazwisko | varchar(30) | <input type="text"/> | | Kowalski |
| imie | varchar(30) | <input type="text"/> | | Jan |
| adres | varchar(35) | <input type="text"/> | | Gdańsk |
| data_ur | date | <input type="text"/> | | 2001-05-01 |

Wykonaj

Rysunek 2.10. Wprowadzanie danych do tabeli

2.2.2. MS SQL Server

MS SQL Server należy do grupy zaawansowanych serwerów bazodanowych. Jego zalety to:

- restrykcyjne mechanizmy zapewniające bezpieczeństwo systemu,
- wbudowane mechanizmy replikacji i synchronizacji danych,
- partycjonowanie danych (a co za tym idzie — zwiększenie wydajności),
- raportowanie danych, dzięki czemu można przeprowadzać dokładne analizy danych,
- łatwość instalowania,
- dostępna wersja darmowa systemu.

Wady serwera MS SQL:

- w wersji darmowej limit rozmiaru bazy do 10 GB oraz brak niektórych narzędzi dostępnych w wersji płatnej.

Cechy systemu zwiększające bezpieczeństwo danych:

- autoryzacja występująca zarówno przy instalowaniu, jak i na poziomie dostępu do baz danych,
- dwa tryby uwierzytelniania (Windows i SQL Server),
- zarządzanie dzięki zastosowaniu ról,
- szyfrowanie danych,
- zastosowanie certyfikatów,
- szybkie przywracanie systemu.

MS SQL Server 2017 jest dostępny dla systemów Windows i Linux. Jest pakietem komercyjnym, ale występuje również w wersji Express, z której można korzystać nieodpłatnie. Oprogramowanie można pobrać ze strony <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>. Wersji Express możemy używać zarówno do celów naukowych, jak i komercyjnych. MS SQL Server w wersji Express umożliwia tworzenie małych, opartych na danych aplikacji sieci WWW i aplikacji mobilnych, których rozmiar nie przekracza 10 GB.

Po pobraniu pakietu można przystąpić do jego instalacji. Uruchomienie pliku instalacyjnego rozpocznie proces instalowania, który umożliwia również konfigurację serwera. Po zakończeniu instalowania usługa jest uruchamiana automatycznie po każdym uruchomieniu komputera.

Po zakończeniu procesu instalacji można zalogować się do serwera i utworzyć bazę danych. W celu zalogowania się należy w wierszu poleceń wpisać:

```
sqlcmd -Snazwa_komputera\nazwa_uslugi
```

Po zalogowaniu można utworzyć nową bazę danych, wpisując polecenie:

```
CREATE DATABASE nazwa_bazy
```

Przy kolejnych połączeniach z serwerem podczas logowania można wskazać bazę danych, dodając w poleceniu opcję -d:

```
sqlcmd -Snazwa_komputera\nazwa_uslugi -d nazwa_bazy
```

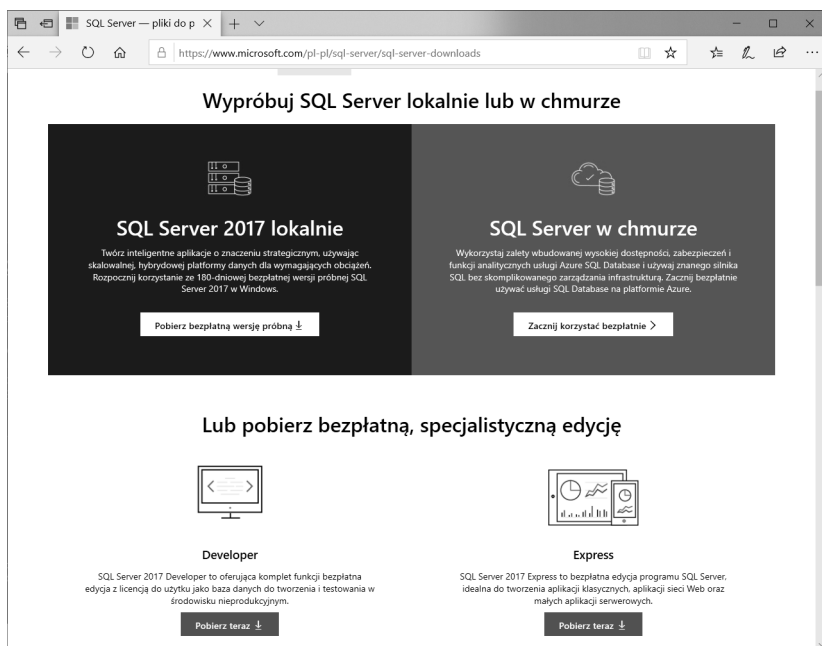
Instalowanie serwera SQL Server 2017

SQL Server 2017 Express to bezpłatna wersja programu SQL Server do tworzenia aplikacji klasycznych, aplikacji sieci WWW oraz małych aplikacji serwerowych. Zawiera narzędzia graficznego interfejsu (SQL Management Studio) i pozwala na bezpłatne rozpowszechnianie własnych aplikacji bazodanowych.

Zalety tego serwera to:

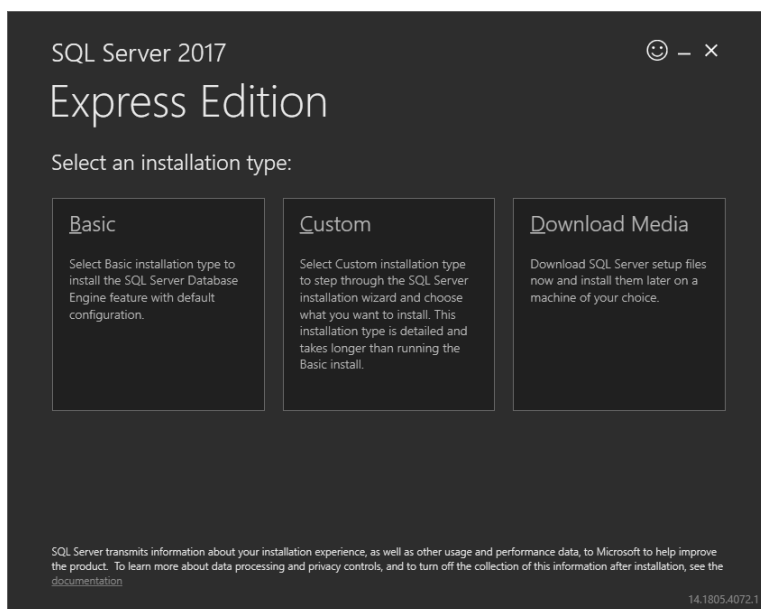
- graficzne narzędzia do administrowania serwerem,
- możliwość łatwego instalowania w systemach Windows i Linux,
- duża zgodność ze standardem SQL3.

Jeżeli zamierzamy zainstalować SQL Server 2017 Express i przygotować środowisko do pracy, należy pobrać i zainstalować MS SQL Server oraz SQL Server Management Studio. Plik instalacyjny MS SQL Server można pobrać na przykład ze strony <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>. Należy wybrać edycję SQL Server Express (rysunek 2.11), a następnie uruchomić plik instalacyjny *SQLServer2017-SSEI-Expr*.



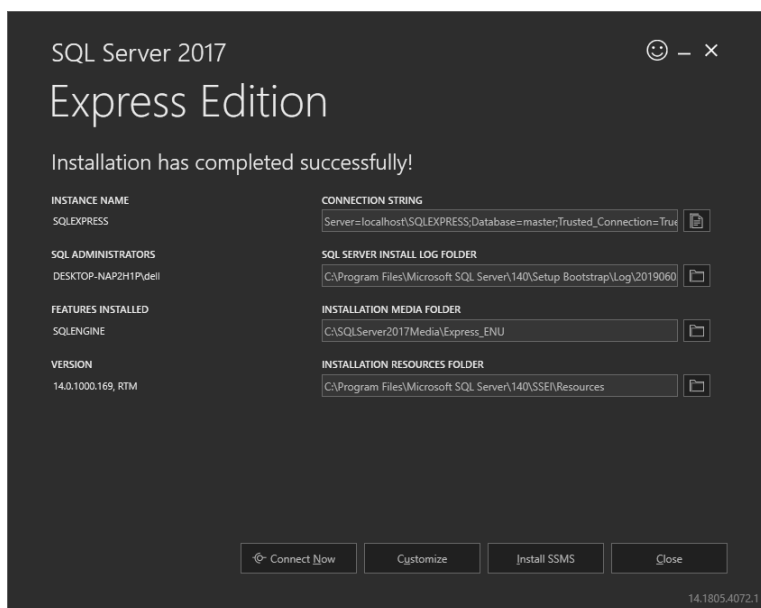
Rysunek 2.11. Strona pobierania SQL Server Express

Zostanie uruchomiony kreator instalacji SQL Server. W pierwszym oknie instalatora należy wybrać typ instalacji (rysunek 2.12) i zaznaczyć opcję *Basic*. W kolejnym oknie należy określić język instalacji. Ponieważ nie ma polskiej wersji serwera, wybieramy wersję w języku angielskim i akceptujemy warunki licencji.



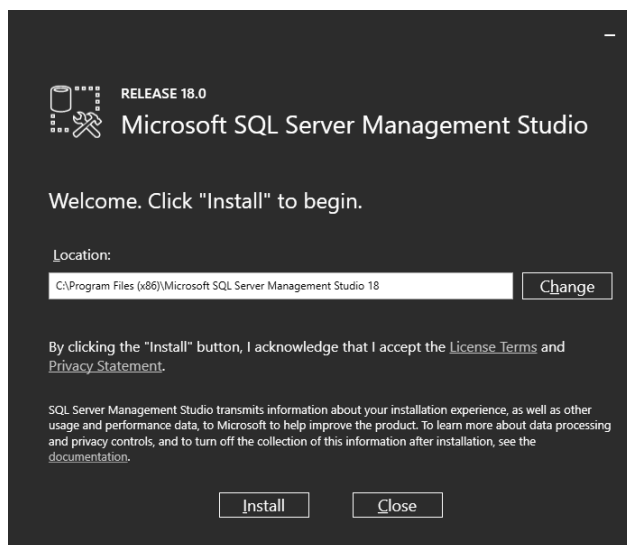
Rysunek 2.12. Wybór typu instalacji SQL Server

Po kliknięciu przycisku *Install* rozpocznie się instalacja programu. Jeżeli przebiegała bez zakłóceń, po jej zakończeniu pojawi się okno podsumowujące przebieg instalacji (rysunek 2.13).



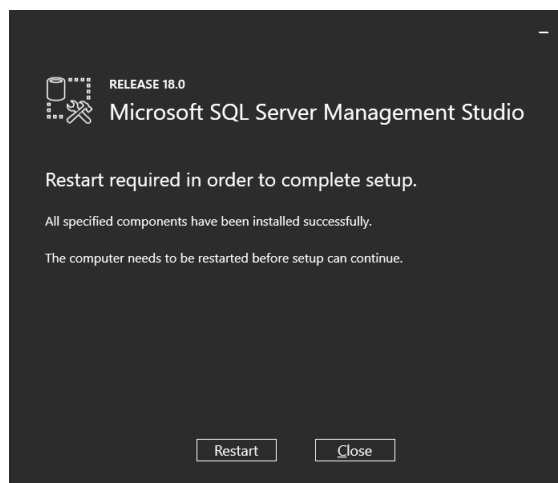
Rysunek 2.13. Finał instalacji MS SQL Server

Aby przejść do pobierania i instalacji SQL Server Management Studio, w otwartym oknie należy wybrać przycisk *Install SSMS*. Po jego wybraniu zostaniemy przekierowani na stronę firmy Microsoft, z której można pobrać *SQL Server Management Studio*. Po pobraniu i zapisaniu należy uruchomić plik instalacyjny *SSMS-Setup-ENU.exe*. W otwartym oknie instalatora należy wybrać opcję *Install* (rysunek 2.14).



Rysunek 2.14. Instalacja SQL Server Management Studio

Po pomyślnym zakończeniu instalowania programu pojawi się okno z informacją o kompletności instalacji. Po wybraniu przycisku *Restart* instalator zostanie zamknięty, a komputer ponownie uruchomiony (rysunek 2.15).

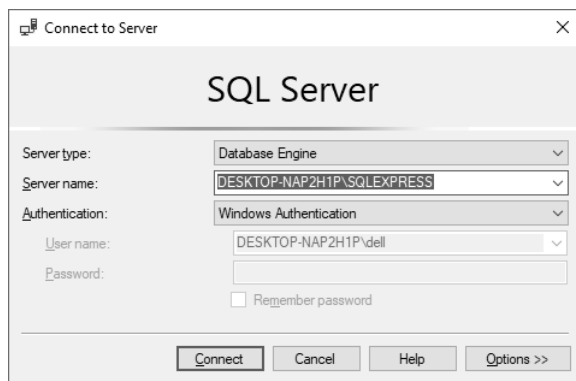


Rysunek 2.15. Pomyślne zainstalowanie SQL Server Management Studio

SQL Server Management Studio

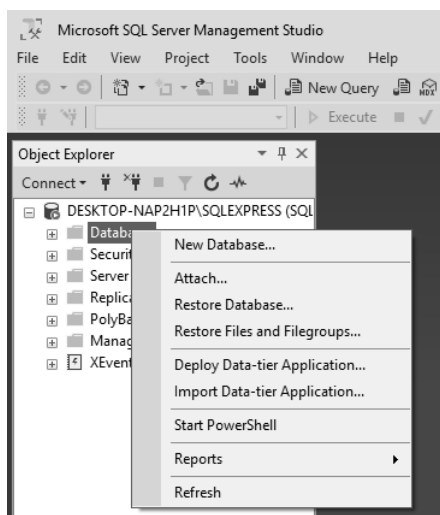
SQL Server Management Studio to bezpłatne narzędzie, za pomocą którego można na poziomie interfejsu graficznego zarządzać bazami danych SQL Server Express. Narzędzie to może również posłużyć do zarządzania instancjami silnika SQL Server Database Engine. Jest ono uzupełnieniem darmowych serwerów baz danych.

Po uruchomieniu aplikacji pojawi się okno logowania do serwera. W oknie tym zostanie wyświetlona nazwa serwera (*Server name*) (rysunek 2.16). Aby zalogować się do wybranego serwera, należy kliknąć przycisk *Connect*.



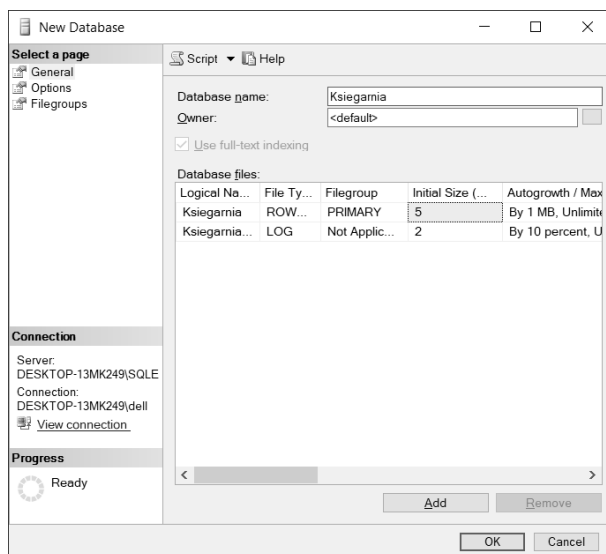
Rysunek 2.16. Logowanie do serwera baz danych

Po zalogowaniu uzyskujemy dostęp do serwera za pośrednictwem SSMS (*SQL Server Management Studio*). Aby rozpocząć pracę z nową bazą danych, należy kliknąć prawym przyciskiem myszy folder *Databases* i wybrać opcję *New Database...* (rysunek 2.17), a następnie w otwartym oknie wprowadzić nazwę tworzonej bazy danych.



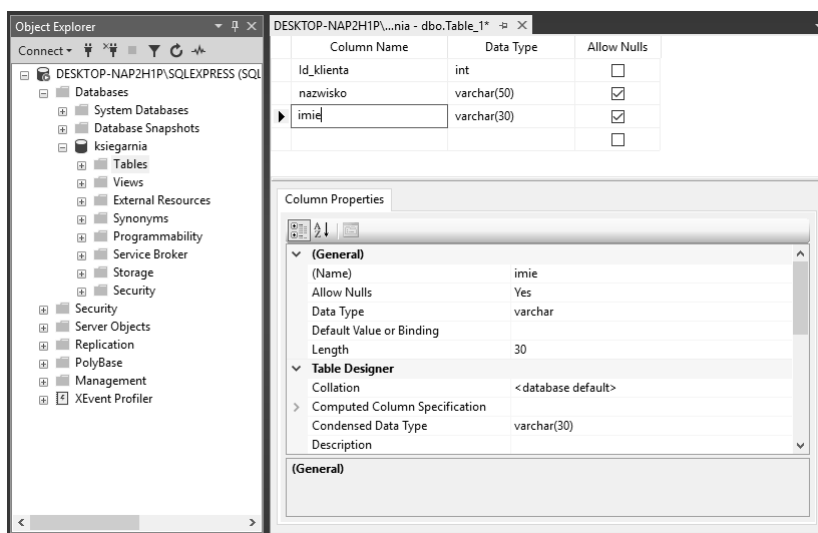
Rysunek 2.17. Opcja tworzenia nowej bazy danych

W wyniku tych działań uzyskamy na serwerze bazodanowym pustą bazę danych (rysunek 2.18).



Rysunek 2.18. Nowa baza danych

Aby utworzyć w bazie danych tabelę, należy w oknie *Object Explorer* rozwinąć gałąź z bieżącą bazą danych, wybrać opcję *Tables* i po jej kliknięciu prawym przyciskiem myszy wybrać z menu *New/Table....* W otwartym oknie można zaprojektować strukturę tabeli, podając nazwy kolejnych kolumn i typy pól (rysunek 2.19). W dolnej części okna można ustawić właściwości tworzonego pola.

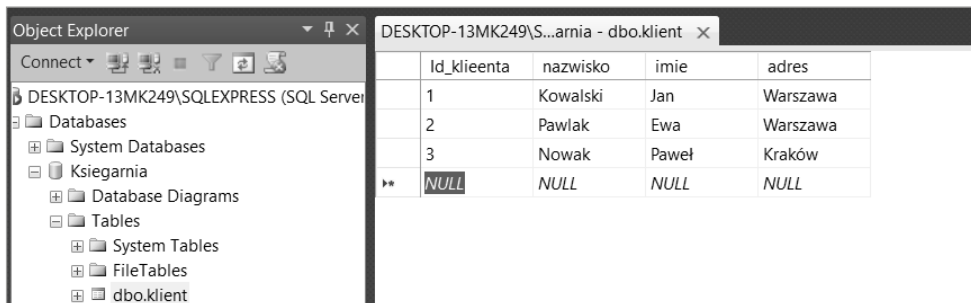


Rysunek 2.19. Tworzenie nowej tabeli

Po utworzeniu wszystkich pól tabeli można zapisać jej strukturę. W tym celu należy wybrać z menu *File/Save Table_1*, podać nazwę tabeli i zatwierdzić jej utworzenie. W podobny sposób można tworzyć struktury kolejnych tabel.

Aby wprowadzić dane do tabeli, trzeba w oknie *Object Explorer* rozwinąć gałąź *Tables*, kliknąć prawym przyciskiem myszy wybraną tabelę i z listy wybrać opcję *Edit Top 200 Rows*. Po wybraniu tej opcji zostanie otwarte okno służące do wprowadzania danych do tabeli (rysunek 2.20).

Jeżeli utworzone obiekty bazy danych nie są widoczne w oknie *Object Explorer*, należy kliknąć ikonę *Refresh* w celu odświeżenia jego zawartości.



The screenshot shows the 'Object Explorer' window on the left, expanded to 'Tables' > 'dbo.klient'. The main window displays a table with the following data:

| Id_klienta | nazwisko | imie | adres |
|------------|----------|-------|----------|
| 1 | Kowalski | Jan | Warszawa |
| 2 | Pawlak | Ewa | Warszawa |
| 3 | Nowak | Paweł | Kraków |
| NULL | NULL | NULL | NULL |

Rysunek 2.20. Wprowadzanie danych do tabeli

2.3. Pytania i zadania

2.3.1. Pytania

1. Podaj definicję systemu zarządzania bazą danych.
2. Z jakich elementów składa się system baz danych?
3. Czym charakteryzuje się architektura klient-serwer?
4. Czym charakteryzuje się architektura 3-warstwowa?
5. Wymień popularne serwery baz danych.
6. Podaj podstawowe cechy serwera MySQL.
7. Podaj podstawowe cechy serwera MS SQL Server.

2.3.2. Zadania

Zadanie 1.

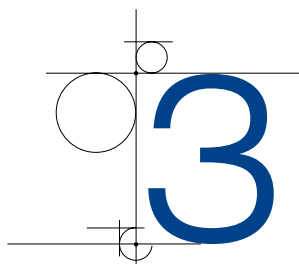
Zbuduj model komunikowania się użytkowników sieci dostępnej w Twojej szkole z bazą *Moja_szkola*. Określ miejsce gromadzenia danych oraz serwer, na którym będzie znajdował się system zarządzania bazą danych. Zastanów się, jak zapewnić bezpieczeństwo i współdzielenie danych.

Zadanie 2.

Dla bazy danych *Moja_szkoła* utwórz w MySQL tabele i schemat połączeń. Zdefiniuj klucze podstawowe i typy pól. Wprowadź przykładowe dane.

Zadanie 3.

Dla bazy danych *Moja_szkoła* utwórz w programie SQL Server Management Studio tabele i schemat połączeń. Zdefiniuj klucze podstawowe i typy pól. Wprowadź przykładowe dane.



SQL — strukturalny język zapytań

3.1. Wprowadzenie

SQL (ang. *Structured Query Language*) to uniwersalny język zapytań stosowany w systemach relacyjnych baz danych do komunikowania się z bazą. Jest on również podstawowym językiem programowania baz danych, pozwalającym na tworzenie i modyfikowanie obiektów bazy danych (na przykład tabel).

Język SQL jest językiem **deklaratywnym**. W językach deklaratywnych definiuje się warunki, jakie musi spełniać końcowy wynik, natomiast nie definiuje się sposobu, w jaki ten wynik zostanie osiągnięty. W instrukcjach języka SQL nie znajdziemy informacji, w jaki sposób serwer bazodanowy powinien uzyskać wymagany wynik. Sposób wykonania instrukcji zależy od serwera baz danych i to zadaniem serwera jest znalezienie najlepszego sposobu osiągnięcia spodziewanego wyniku.

3.2. Standardy języka SQL

Każdy z dostępnych na rynku systemów bazodanowych zawiera specyficzne, niedostępne w innych systemach elementy. W różnych systemach mogą być stosowane różne wersje języka SQL. W celu ujednolicenia wersji języka SQL Amerykański Narodowy Instytut Standardów (ang. *American National Standards Institute* — ANSI) i Międzynarodowa Organizacja Normalizacyjna (ang. *International Organization for Standardization* — ISO) opracowują i publikują standardy języka SQL. W 1999 roku został przyjęty standard ANSI SQL99 (SQL3) i obecnie większość producentów systemów bazodanowych tworzy systemy zgodne z tym standardem.

Standard SQL99 nie definiuje wielu rozszerzeń języka SQL, dlatego w 2003 roku został opublikowany czwarty standard języka SQL, który jest rozszerzeniem SQL3 i definiuje standardy związane z obsługą języka XML. Obecnie obowiązująca wersja to ANSI SQL:2016.

Mimo zdefiniowanych standardów systemy zarządzania bazą danych nadal dodają własne rozszerzenia, ponieważ użytkownicy oczekują od tych systemów funkcji nieujętych w standardzie języka. Można jednak przyjąć, że wszystkie typowe operacje są wykonywane tak samo, niezależnie od używanego systemu zarządzania bazą danych.

Dialekty języka SQL

Wprowadzenie standardu języka SQL określiło wymagania, jakie musi spełniać system zarządzania bazą danych, który obsługuje język SQL, ale wielu dostawców oferuje dodatkowe funkcje ujęte w tzw. dialekty języka SQL. Najbardziej znane to:

- T-SQL (ang. *Transact-SQL*) — stosowany na serwerach Microsoft SQL Server i Sybase Adaptive Server Enterprise,
- PL/SQL (ang. *Procedural Language/SQL*) — stosowany na serwerach firmy Oracle,
- PL/pgSQL (ang. *Procedural Language/PostgreSQL Structured Query Language*) — podobna do PL/SQL wersja języka zaimplementowana na serwerze PostgreSQL,
- SQL PL (ang. *SQL Procedural Language*) — wersja używana na serwerach firmy IBM.

Terminatory SQL

W języku SQL mamy do czynienia z dwoma rodzajami terminatorów. Są to terminatory poleceń i terminatory wsadowe. Terminatory poleceń kończą każde polecenie napisane w języku SQL. Terminatory wsadowe kończą ciąg poleceń języka SQL. Powodują one przesłanie ciągu poleceń do serwera.

Terminatory poleceń najczęściej związane są z dialektami. W niektórych dialektach języka SQL, na przykład Oracle i InterBase, wymagane jest kończenie każdego polecenia średnikiem.

Terminatory wsadowe najczęściej związane są ze stosowanymi narzędziami. Na przykład edytor pakietu Sybase wymaga zakończenia ciągu poleceń słowem kluczowym GO, a w edytorze WINSQL średnik na końcu ciągu poleceń jest opcjonalny.

W systemach baz danych, które pozwalają na wykonanie jednocześnie wielu instrukcji SQL, średnik służy do oddzielenia kolejnych instrukcji.

3.3. Składnia języka SQL

Standard języka SQL92 wprowadził klasyfikację poleceń ze względu na ich przeznaczenie. Są to:

- Instrukcje DDL (ang. *Data Definition Language*) — tworzą język definiowania danych i służą do tworzenia, modyfikowania i usuwania obiektów bazy danych.
- Instrukcje DML (ang. *Data Manipulation Language*) — tworzą język manipulowania danymi i służą do odczytywania i modyfikowania danych.
- Instrukcje DCL (ang. *Data Control Language*) — tworzą język kontroli dostępu do danych i umożliwiają nadawanie i odbieranie uprawnień użytkownikom.

Tak opracowana klasyfikacja poleceń realizuje trzy podstawowe typy zadań: definiowanie danych, manipulowanie danymi i kontrolowanie danych. W zależności od zastosowań języka SQL można spotkać bardziej szczegółowy podział jego poleceń:

- Instrukcje **DDL** (ang. *Data Definition Language*) — tworzą język definiowania danych i służą do tworzenia i modyfikowania struktury bazy danych.
- Instrukcje **DML** (ang. *Data Manipulation Language*) — tworzą język manipulowania danymi i służą do modyfikowania danych.
- Instrukcje **DCL** (ang. *Data Control Language*) — tworzą język kontroli uprawnień użytkowników.
- Instrukcje **TCL** (ang. *Transaction Control Language*) — tworzą język obsługi transakcji.
- Instrukcja **DQL** (ang. *Data Querying Language*) — tworzy język definiowania zapytań, który zawiera tylko jedno polecenie *SELECT* i służy do tworzenia kwerend (zapytań).

3.3.1. Instrukcje języka SQL

Instrukcje języka SQL składają się ze słów kluczowych, klauzul, wyrażeń i warunków. Słowa kluczowe są niezmiennymi elementami poleceń. Wyrażenie jest poleceniem, które mówi, co należy zrobić. Klauzula określa ograniczenia dla danego polecenia i jest definiowana w formie warunków (na przykład klauzula *WHERE*).

Przykład 3.1

```
SELECT nazwisko, imię
FROM Klient
WHERE miejscowość='Warszawa';
```

Wyrażenie *SELECT* mówi, że trzeba wybrać z bazy dane, które zostały wymienione za poleceniem. W następnej linii zostało określone miejsce, z którego mają pochodzić dane. W ostatniej linii został zdefiniowany warunek, który musi zostać spełniony przy wybieraniu danych. Polecenie kończy się średnikiem.

Wyrażenia języka SQL mogą zawierać: identyfikatory, literały i operatory.

Identyfikator jednoznacznie definiuje obiekt bazy danych. Każdy obiekt bazy danych (baza, tabela, kolumna) musi mieć niepowtarzalną nazwę. Identyfikatory muszą być zgodne ze zdefiniowanymi w standardzie regułami:

- nie mogą być dłuższe niż 128 znaków,
- mogą zawierać litery, cyfry oraz znaki: @, \$, #,
- nie mogą zawierać spacji ani innych znaków specjalnych,
- muszą zaczynać się literą,
- nie mogą być słowami kluczowymi języka SQL.

Dodatkowo zaleca się, aby nazwy były krótkie, ale jednoznacznie opisywały obiekt. Wielkość liter powinna być zgodna z przyjętymi regułami.

Literał jest stałą wartością. Wszystkie wartości liczbowe, ciągi znaków i daty, jeżeli nie są identyfikatorami, są traktowane jako stałe, czyli literały.

Typy liczbowe mają domyślną postać liczby, na przykład: 150, -375, 5.39, 3E4.

Ciągi znaków muszą być umieszczone w apostrofach, na przykład: 'Gdańsk', 'KOMPUTER'.

Typy daty muszą być umieszczone w apostrofach, na przykład: '20-09-2012', '2010-02-13'.

Operatory odgrywają rolę łączników. Ze względu na zastosowanie zostały podzielone na:

- **arytmetyczne** — suma +, różnica -, iloczyn *, iloraz /, modulo %,
- **znakowe** — konkatenacja (złączenie ciągu znaków) +, symbol zastępujący dowolny ciąg znaków %, symbol zastępujący jeden znak _,
- **logiczne** — koniunkcja AND, alternatywa OR, negacja NOT,
- **porównania** — =, <, >, <=, >=, <>,
- **operatory specjalne** (zależą od systemu bazodanowego) — IS NULL lub IS NOT NULL — porównuje dowolną wartość z wartością NULL; LIKE "wzorzec" — porównuje łańcuch tekstowy z podanym wzorcem; wzorzec może być definiowany z użyciem symboli wieloznacznych * i ?; BETWEEN *wartość1* AND *wartość2* — sprawdza, czy wartość wyrażenia mieści się w podanym przedziale wartości; IN (*ciąg1*, *ciąg2*...) — określa, czy dana wartość należy do zestawu listy wartości.

Słowa kluczowe to wyrazy zastrzeżone, interpretowane przez język SQL w ściśle określony sposób. Do słów kluczowych należą:

- instrukcje języka SQL,
- klauzule języka SQL,
- nazwy typów danych,
- nazwy funkcji systemowych,
- terminy zarezerwowane do późniejszego użycia w systemie.

3.3.2. Typy danych

Typy danych określają rodzaj informacji przechowywanej w kolumnach tabeli. Określają również, jakiego rodzaju dane mogą być przekazywane jako parametry do procedur i funkcji lub jakie dane mogą być przechowywane w zmiennych. Typy danych mogą różnić się nieznacznie od standardowych typów w różnych systemach baz danych, nawet jeżeli mają takie same nazwy. Możemy je podzielić według kategorii na:

- typy liczbowe,
- typy daty i czasu,
- typy znakowe,
- typy walutowe,
- typy binarne,
- typy specjalne.

Zestawienie niektórych typów danych w różnych systemach baz danych zostało pokazane w tabeli 3.1.

Tabela 3.1. Niektóre typy danych w SQL

| Kategoria | Standard SQL | MS SQL Server | MySQL |
|-------------------|----------------------------|----------------|----------------------------|
| typy liczbowe | integer | int | integer (n) |
| | smallint | smallint | smallint |
| | — | bigint | — |
| | — | tinyint | — |
| | float | float | float (n) |
| | — | real | real |
| | — | decimal (p, s) | decimal |
| | — | numeric (p, s) | number (p, s) |
| typy daty i czasu | — | datetime | — |
| | — | — | date |
| | — | smalldatetime | — |
| typy znakowe | character | char (n) | char (n) |
| | character varying | varchar (n) | varchar |
| | national character | nchar (n) | national character |
| | — | — | nvarchar2 (n) |
| | — | ntext | — |
| | national character varying | nvarchar (n) | national character varying |
| typy walutowe | — | money | — |
| | — | smallmoney | — |
| typy binarne | binary | binary | — |
| | binary varying | varbinary | — |
| typy specjalne | — | text | — |
| | — | image | — |
| | bit | bit | — |

Ze względu na niezgodność typów danych próba przeniesienia bazy danych z jednego systemu do innego może zakończyć się niepowodzeniem. Rozwiązaniem może być posługiwanie się podczas tworzenia struktury bazy danych najpopularniejszymi typami danych występującymi w wielu systemach.

Wartość NULL

NULL jest wartością specjalną. Oznacza wartość pustą (w komórce tabeli nie została umieszczona żadna wartość). Wartość NULL reprezentuje w bazie danych nieznaną lub nieistotną wartość. Jest ona różna od wartości 0 i od pustego ciągu znaków. Ze względu na to, że systemy bazodanowe muszą przetwarzać wartość NULL jako brakującą informację, obowiązuje w nich logika trójwartościowa. Oznacza to, że w wyniku porównania wartości NULL z inną wartością otrzymamy wartość nieznaną (ang. *unknown*). Wynikiem dowolnej operacji wykonanej na wartości NULL jest zawsze wartość NULL.

3.3.3. Hierarchia obiektów bazy danych

Obiekty dostępne na serwerze bazodanowym tworzą hierarchię: serwer zawiera wiele baz danych, baza danych może zawierać wiele schematów, w każdym schemacie może występować wiele tabel, a każda tabela może składać się z wielu kolumn. Każdy z tych obiektów powinien mieć nadaną niepowtarzalną nazwę (ang. *alias*).

Przy odwoływaniu się w instrukcjach do obiektu jego nazwa powinna zawierać następujące elementy:

nazwa_serwera.nazwa_bazy_danych.nazwa_schematu.nazwa_obiektu

W praktyce niektóre z tych elementów mogą zostać pominięte.

- Nazwa serwera wskazuje serwer bazodanowy, na którym znajduje się obiekt. Jeżeli ten element zostanie pominięty, to instrukcja zostanie wykonana przez serwer, z którym jesteśmy połączeni.
- Nazwa bazy danych wskazuje bazę danych, w której znajduje się obiekt. Jeżeli pominiemy tę nazwę, serwer wykona instrukcję w bazie danych, z którą jesteśmy połączeni.
- Nazwa schematu wskazuje schemat, w którym znajduje się obiekt. Schemat jest zbiorem powiązanych ze sobą obiektów, tworzonym w bazie danych przez użytkownika w celu usprawnienia administrowania bazami danych. Jeżeli nazwa schematu zostanie pominięta, serwer przyjmie, że obiekt znajduje się w domyślnym schemacie użytkownika wykonującego instrukcję. Jeżeli nie zadeklarujemy schematu, domyślnym schematem użytkownika staje się schemat *dbo* (dla MS SQL).
- Różni użytkownicy bazy danych mogą mieć przypisane różne schematy domyślne, dlatego podając nazwę schematu, unikniemy ewentualnych błędów. Poza tym posługiwanie się nazwami schematów skraca czas wykonania instrukcji.

3.4. Język definiowania danych (DDL)

Język DDL używany jest do tworzenia, modyfikowania i usuwania bazy danych oraz jej obiektów. Instrukcje wchodzące w skład tego języka to:

- `CREATE nazwa_obiektu` — tworzy nowy obiekt,
- `ALTER nazwa_obiektu` — zmienia strukturę istniejącego obiektu,
- `DROP nazwa_obiektu` — usuwa istniejący obiekt.

Ponieważ instrukcje te wykorzystują indywidualne cechy stosowanego przez użytkownika systemu bazodanowego, ich składnia może być różna.

Pracę z bazą danych rozpoczniemy od jej utworzenia. Na potrzeby ćwiczeń w tej części podręcznika zostanie wykorzystana wcześniej zaprojektowana baza danych *księgarnia* po niewielkich modyfikacjach (rozdział 1., przykład 1.2). Struktura bazy danych *księgarnia* została pokazana na rysunku 3.1.



Rysunek 3.1. Baza danych księgarnia

Tworzenie bazy danych jest realizowane za pomocą instrukcji:

```
CREATE DATABASE nazwa_bazy
```

Przykład 3.2

```
CREATE DATABASE Dane_osobowe;
```

Polecenie usuwające bazę danych również nie jest skomplikowane:

```
DROP DATABASE nazwa_bazy
```

Przykład 3.3

```
DROP DATABASE Dane_osobowe;
```

Usunąć bazę danych może tylko użytkownik, który ma odpowiednie uprawnienia, na przykład administrator serwera bazodanowego lub właściciel bazy danych. W trakcie usuwania bazy nikt nie może być z nią połączony.

Jeżeli baza danych będzie tworzona w SQL Server Management Studio za pomocą skryptów, należy kliknąć przycisk *New Query*. Spowoduje to otwarcie nowego okna, w którym trzeba wpisać odpowiednie polecenie (rysunek 3.2).

Ćwiczenie 3.1

Utwórz bazę danych *ksiegarnia*, korzystając ze skryptów języka SQL.

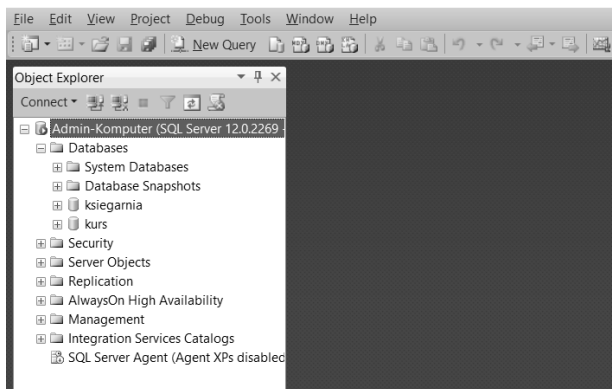
Rozwiązanie

W SQL Server klikamy przycisk *New Query* i w otwartym oknie wprowadzamy poniższy skrypt:

```
CREATE DATABASE ksiegarnia;
```

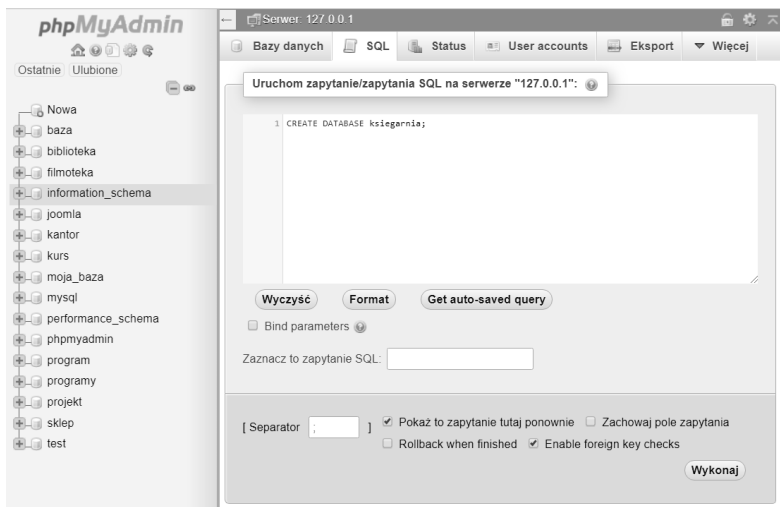
Po wprowadzeniu kodu klikamy przycisk *Execute* (lub wybieramy z klawiatury *F5*). W wyniku wykonania skryptu zostanie utworzona baza danych *ksiegarnia*.

Jeżeli baza danych będzie tworzona w MySQL za pomocą phpMyAdmin, wybieramy zakładkę *SQL*, w otwartym oknie wpisujemy polecenie tworzenia bazy danych i klikamy przycisk *Wykonaj* (rysunek 3.3).



Rysunek 3.2.

Okno tworzenia zapytań dla bazy danych w MS SQL Server



Rysunek 3.3. Okno tworzenia zapytań dla bazy danych w MySQL

3.4.1. Tworzenie i usuwanie tabel

Po utworzeniu bazy danych następnym etapem pracy jest tworzenie tabel i połączeń, tak aby została zbudowana cała struktura dla bazy danych.

Do tworzenia tabel służy polecenie `CREATE TABLE` w postaci:

```
CREATE TABLE nazwa_tabeli
(
    nazwa_kolumny_1 typ_kolumny_1 [atrybuty],
    nazwa_kolumny_2 typ_kolumny_2 [atrybuty],
    . . . .
    nazwa_kolumny_n typ_kolumny_n [atrybuty],
)
```

Przykład 3.4

```
CREATE TABLE Osoba
(
    id_osoby int,
    nazwisko varchar (60),
    imie varchar (40),
    miasto varchar (50),
    ulica varchar (50),
    nr_domu varchar (7)
);
```

Podczas tworzenia tabel należy pamiętać, że:

- każda tabela musi mieć unikatową nazwę i unikatowego właściciela,
- każda kolumna musi mieć unikatową nazwę,
- nazwy tabel i kolumn muszą być zgodne z zasadami dotyczącymi standardów SQL,
- każda kolumna w tabeli musi mieć zdefiniowany typ,
- jeżeli kolumna jest typu znakowego, trzeba podać jej maksymalną długość,
- utworzone tabele są puste.

Ćwiczenie 3.2

Dla bazy danych *ksiegarnia* utwórz tabelę *Klient* z kolumnami takimi jak widoczne na rysunku 3.1. Ustaw odpowiednie typy dla kolumn tworzonej tabeli.

Rozwiązanie

Klikamy przycisk *New Query* (SQL Server Management Studio) lub wybieramy zakładkę *SQL* (phpMyAdmin) i w otwartym oknie wprowadzamy następujący skrypt:

```
USE ksiegarnia;

CREATE TABLE Klient
(
    id_klienta int NOT NULL,
    nazwisko varchar (60) NOT NULL,
    imie varchar (40) NOT NULL,
    kod_pocztowy varchar (6),
    miejscowosc varchar (50),
    ulica varchar (50),
    nr_domu varchar (7),
    PESEL varchar (11),
    telefon varchar (12),
    adres_e_mail varchar (70)
);
```

Po wprowadzeniu skryptu klikamy przycisk *Execute* (MS SQL Server) lub *Wykonaj* (MySQL). W wyniku wykonania skryptu do bazy danych zostanie dodana tabela *Klient*.

Na serwerze może istnieć wiele baz danych. W celu określenia, dla której bazy danych będą wykonywane kolejne polecenia, została użyta instrukcja USE.

Ćwiczenie 3.3

Utwórz tabelę *Książki* z kolumnami: *id_ksiazki*, *tytuł*, *autor*, *cena*, *wydawnictwo*, *temat*, *miejsce_wydania*, *język_ksiazki*, *opis*.

Rozwiązanie

Polecenie tworzące tabelę (dla MS SQL Server) będzie miało postać:

```
CREATE TABLE Ksiazki
(
    id_ksiazki int,
    tytul varchar (100) NOT NULL,
    autor varchar (80) NOT NULL,
    cena money,
    wydawnictwo varchar (20),
    temat varchar (30),
    miejsce_wydania varchar (28),
    jezyk_ksiazki varchar (15),
    opis varchar (100)
);
```


Ponieważ w systemie MySQL nie występuje typ `money`, deklaracja kolumny *cena* w tabeli *Książki* będzie miała postać:

```
CREATE TABLE Książki
(
    id_książki int,
    tytuł varchar (100) NOT NULL,
    autor varchar (80) NOT NULL,
    cena decimal,
    wydawnictwo varchar (20),
    temat varchar (30),
    miejsce_wydania varchar (28),
    język_książki varchar (15),
    opis varchar (100)
);
```

Typ `decimal` w MySQL jest stosowany do przechowywania kwot pieniędzy i wszędzie tam, gdzie potrzebna jest duża precyzja.

Do usuwania tabel służy instrukcja `DROP TABLE` w następującej postaci:

```
DROP TABLE nazwa_tabeli
```

Przykład 3.5

```
DROP TABLE Osoba;
```

W wyniku wykonania instrukcji tabela *Osoba* zostanie usunięta z bazy danych.

3.4.2. Schematy (MS SQL Server)

Baza danych może zawierać wiele tabel. Zarządzanie bazą, która zawiera kilkadziesiąt, a czasami kilkaset tabel, może być bardzo trudne. W celu usprawnienia administrowania takimi bazami danych możemy tworzyć **schematy** i przydzielać do nich obiekty bazy danych.

Obiekty bazy danych powinny być przydzielane do schematów według powiązań, jakie zachodzą między nimi. Jeżeli obiekty bazy zostaną przydzielone do schematów, to administrator bazy danych będzie mógł zdefiniować uprawnienia użytkownika nie na poziomie poszczególnych tabel, ale na poziomie całych schematów.

Schemat jest tworzony za pomocą instrukcji `CREATE SCHEMA`, która ma postać:

```
CREATE SCHEMA nazwa_schematu
```

Przykład 3.6

```
CREATE SCHEMA Zasoby;
```

Podczas tworzenia schematu można tworzyć tabele i widoki, definiować prawa dostępu. Obiekty tworzone instrukcją `CREATE SCHEMA` są umieszczane wewnątrz definiowanego schematu.

Przykład 3.7

```
CREATE SCHEMA Magazyn
CREATE TABLE Ksiazki
(
    id_ksiazki int,
    tytul nvarchar (100) NOT NULL,
    cena money,
    wydawnictwo nvarchar (20));
CREATE TABLE Autor
(
    id_autora int,
    nazwisko nvarchar (100) NOT NULL,
    imie nvarchar (30) NOT NULL);
```

Przypisanie obiektu do schematu może nastąpić na dwa sposoby: jawnie i niejawnie. Aby jawnie przypisać tabelę do schematu, należy poprzedzić jej nazwę nazwą schematu:

```
CREATE TABLE nazwa_schematu.nazwa_tabeli
```

Każdy obiekt tworzony w bazie danych należy do jakiegoś schematu. Jeżeli podczas tworzenia tabeli nie przypiszemy jej jawnie do schematu, zostanie ona domyślnie umieszczona w schemacie, w którym obecnie pracujemy. Najprawdopodobniej będzie to schemat *dbo*. Jednak gdyby zalogowany użytkownik znajdował się domyślnie w schemacie *Zasoby*, to tabela zostałaby dodana do schematu *Zasoby*. Niejawne przypisywanie do schematów nie jest zalecane.

Przykład 3.8

```
CREATE TABLE Zasoby.Ksiazki
(
    id_ksiazki int,
    tytul nvarchar (100) NOT NULL,
    autor nvarchar (80) NOT NULL,
    cena money,
    wydawnictwo nvarchar (20),
    temat nvarchar (30),
    miejsce_wydania nvarchar (28),
```

```

    jezyk_ksiazki nvarchar (15),

    opis nvarchar (100)

);

```

Schemat może mieć tylko jednego właściciela, ale jeden użytkownik może mieć wiele schematów. Każdy użytkownik ma zdefiniowany domyślny schemat, który może zostać zmieniony poleceniem podczas tworzenia użytkownika:

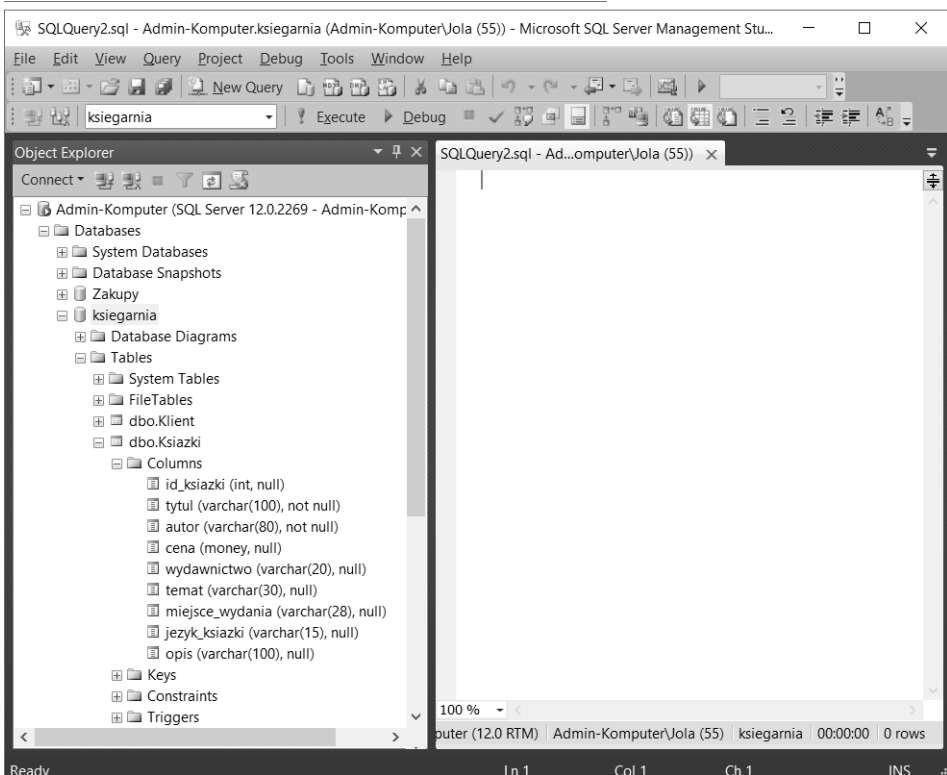
```
CREATE USER nazwa_uzytkownika WITH DEFAULT_SCHEMA = nazwa_schematu
```

lub gdy użytkownik już istnieje:

```
ALTER USER nazwa_uzytkownika WITH DEFAULT_SCHEMA = nazwa_schematu
```

Jeżeli domyślny schemat nie został zdefiniowany, użytkownikowi zostanie przypisany schemat *dbo*.

Rysunek 3.4 pokazuje strukturę bazy danych i tabel w oknie *SQL Server Management Studio* w domyślnym schemacie *dbo*.



Rysunek 3.4. Struktura bazy danych ksiegarnia

3.4.3. Zmiana struktury tabeli

Zmiana struktury tabeli może polegać na dodaniu kolumny, usunięciu kolumny, dodaniu atrybutu lub usunięciu atrybutu. Do modyfikowania struktury tabeli służy polecenie:

```
ALTER TABLE nazwa_tabeli zmiana
```

Dodanie kolumny

Przykład 3.9

Dodanie kolumny do tabeli *Ksiazki*:

```
ALTER TABLE Ksiazki  
ADD liczba_stron nvarchar (5);
```

Ćwiczenie 3.4

Dodaj do tabeli *Ksiazki* kolumnę *rok_wydania* typu *int*.

Rozwiązanie

```
ALTER TABLE Ksiazki  
ADD rok_wydania int;
```

Usunięcie kolumny

Przykład 3.10

Usunięcie kolumny z tabeli *Ksiazki*:

```
ALTER TABLE Ksiazki  
DROP COLUMN liczba_stron;
```

Modyfikowanie kolumny

Przykład 3.11

Zmiana definicji istniejącej kolumny tabeli *Ksiazki* w SQL Server:

```
ALTER TABLE Ksiazki  
ALTER COLUMN rok_wydania nvarchar (4) NOT NULL;
```

Przykład 3.12

Zmiana definicji istniejącej kolumny tabeli *Ksiazki* w MySQL:

```
ALTER TABLE Ksiazki  
MODIFY COLUMN rok_wydania nvarchar (4) NOT NULL;
```

lub w wersji skróconej:

```
ALTER TABLE Ksiazki
MODIFY rok_wydania nvarchar (4) NOT NULL;
```

Ćwiczenie 3.5

Usuń z tabeli *Ksiazki* kolumnę *autor* i dodaj kolumnę *id_autora* typu int.

Rozwiązanie

```
USE ksiegarnia;

ALTER TABLE Ksiazki
DROP COLUMN autor;

ALTER TABLE Ksiazki
ADD id_autora int NOT NULL;
```

3.4.4. Atrybuty kolumn

Każda kolumna tabeli może mieć zdefiniowane za pomocą atrybutów ograniczenia, które określają, jakie dane mogą zostać w niej zapisane. Ograniczenia dotyczące kolumn mogą być definiowane w trakcie tworzenia tabeli lub w trakcie jej modyfikowania.

PRIMARY KEY

Klucz podstawowy (*Primary Key*) to kolumna lub kombinacja kolumn, które w sposób jednoznaczny definiują wiersz w tabeli.

Do określenia, która kolumna tabeli będzie kluczem podstawowym, stosuje się atrybut `PRIMARY KEY`. Kolumna z tym atrybutem jest unikatowa i automatycznie indeksowana i nie może mieć wartości `NULL`.

Przykład 3.13

Definiowanie klucza podstawowego podczas tworzenia tabeli *Zamowienia*:

```
CREATE TABLE Zamowienia
(
    id_zamowienia int PRIMARY KEY,
    id_klienta int NOT NULL,
    data_zlozenia_zamowienia datetime,
    data_wyslania datetime,
    koszt_wysylki money,
    id_faktury int
);
```

Ponieważ w systemie MySQL nie występuje typ `money`, w deklaracji kolumny *koszt_wysylki* należy użyć typu `decimal` w postaci:

```
koszt_wysylki decimal
```

Aby dodać atrybut klucza podstawowego w istniejącej tabeli, można użyć następującej składni:

```
ALTER TABLE Klient
ADD PRIMARY KEY (id_klienta);
```

Jeżeli w SQL Server do definiowania klucza podstawowego użyte zostało polecenie `ALTER TABLE`, kolumna klucza podstawowego musi mieć ustawiony atrybut `NOT NULL`.

Aby odebrać kolumnie atrybut `PRIMARY KEY`, należy w SQL Server użyć polecenia:

```
ALTER TABLE Klient
DROP CONSTRAINT nazwa_klucza;
```

a w MySQL użyć polecenia:

```
ALTER TABLE Klient
DROP PRIMARY KEY;
```

NOT NULL

Atrybut `NOT NULL` oznacza, że w kolumnie nie mogą wystąpić wartości puste. Aby zabronić wstawiania do kolumny wartości `NULL`, podczas tworzenia tabeli należy po nazwie kolumny wpisać `NOT NULL`.

Tworzona kolumna domyślnie może zawierać wartość `NULL`, ale można również jawnie zezwolić na wprowadzanie do kolumny wartości `NULL`, wpisując po jej nazwie słowo `NULL`.

Przykład 3.14

Blokowanie wartości `NULL` podczas tworzenia tabeli *Ksiazki*:

```
CREATE TABLE Ksiazki
(
    id_ksiazki int NOT NULL PRIMARY KEY,
    tytul nvarchar (100) NOT NULL,
    ....
);
```

Aby ustawić w istniejącej kolumnie atrybut `NOT NULL`, należy w SQL Server użyć polecenia:

```
ALTER TABLE Klient
```

```
ALTER COLUMN PESEL varchar (11) NOT NULL;
```

Do ustawienia atrybutu NOT NULL w MySQL można użyć polecenia:

```
ALTER TABLE Klient
MODIFY PESEL varchar (11) NOT NULL;
```

Ćwiczenie 3.6

W tabeli *Książki* zmodyfikuj kolumnę *id_książki*, dodając do niej atrybut klucza podstawowego. Podobnie w tabeli *Klient* ustaw dla kolumny *id_klienta* atrybut klucza podstawowego.

Rozwiązanie

```
ALTER TABLE Klient
ALTER COLUMN id_klienta int NOT NULL;
GO
ALTER TABLE Klient
ADD PRIMARY KEY (id_klienta);

ALTER TABLE Książki
ALTER COLUMN id_książki int NOT NULL;
GO
ALTER TABLE Książki
ADD PRIMARY KEY (id_książki);
```

Dla serwera MySQL:

```
ALTER TABLE Klient
ADD PRIMARY KEY (id_klienta);
ALTER TABLE Książki
ADD PRIMARY KEY (id_książki);
```

IDENTITY/AUTO_INCREMENT

Atrybut `IDENTITY` stosowany w SQL Server oznacza automatyczne ustawienie unikatowej wartości w kolumnie, dla której został zdefiniowany. Wartość jest wstawiana, gdy w tabeli zostanie dodany nowy rekord. Najczęściej kolumną, dla której ustawiany jest ten atrybut, jest kolumna klucza podstawowego, wtedy `IDENTITY` powoduje automatyczny przyrost wartości dla kolejnego rekordu. Na przykład `IDENTITY (1,1)` oznacza wzrost wartości kolumny o 1, począwszy od wartości 1. Niemożliwe jest nadawanie atrybutu `IDENTITY` istniejącej kolumnie.

Podobne znaczenie w MySQL ma atrybut `AUTO_INCREMENT`.

Przykład 3.15

Definiowanie w SQL Server automatycznej inkrementacji wartości dla kolumny *id_autora* podczas tworzenia tabeli *Autor*:

```
CREATE TABLE Autor
(
    id_autora INT IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    nazwisko varchar (50) NOT NULL,
    imie varchar (30) NOT NULL,
    narodowosc varchar (30),
    okres_tworzenia varchar (35),
    jezyk varchar (30),
    rodzaj_tworczosci varchar (35),
    osiagniecia varchar (100)
);
```

Jeżeli automatyczne numerowanie powinno rozpocząć się od innej wartości niż 1 i powinno wzrastać o określoną wartość różną od 1, należy użyć np. składni `IDENTITY(10, 3)`, gdzie wartością początkową jest 10, a krok wynosi 3.

Przykład 3.16

Definiowanie automatycznej inkrementacji wartości dla kolumny *id_autora* w MySQL podczas tworzenia tabeli *Autor*:

```
CREATE TABLE Autor
(
    id_autora int AUTO_INCREMENT NOT NULL PRIMARY KEY,
    nazwisko varchar (50) NOT NULL,
    imie varchar (30) NOT NULL,
    narodowosc varchar (30),
    okres_tworzenia varchar (35),
    jezyk varchar (30),
    rodzaj_tworczosci varchar (35),
    osiagniecia varchar (100)
);
```

Domyślnie wartość początkowa dla atrybutu `AUTO_INCREMENT` wynosi 1 i dla kolejnych rekordów wzrasta o 1. Jeżeli autonumerowanie powinno rozpocząć się od innej wartości, należy użyć składni:


```

CREATE TABLE Autor
(
    id_autora int AUTO_INCREMENT NOT NULL PRIMARY KEY,
    nazwisko varchar (50) NOT NULL,
    imie varchar (30) NOT NULL,
    narodowosc varchar (30),
    okres_tworzenia varchar (35),
    jezyk varchar (30),
    rodzaj_tworczosci varchar (35),
    osiagniecia varchar (100)
)
AUTO_INCREMENT=12;

```

Ćwiczenie 3.7

Zmodyfikuj tabele *Klient*, *Książki* i *Zamowienia* w taki sposób, aby wartość pola klucza podstawowego w tych tabelach (*id_klienta*, *id_ksiazki* i *id_zamowienia*) dla nowych rekordów automatycznie wzrastała o 1, počzawszy od wartości 1.

DEFAULT

Atrybut `DEFAULT` jest stosowany do wprowadzania do kolumny wartości domyślnej. Domyślna wartość będzie wstawiana do kolumny we wszystkich nowych rekordach tabeli.

Przykład 3.17

Definiowanie wartości domyślnej podczas tworzenia tabeli:

```

CREATE TABLE Ksiazki
(
    ....
    rok_wydania nvarchar (4) NOT NULL DEFAULT '2019'
    ....
);

```

Aby dodać atrybut `DEFAULT` w istniejącej tabeli, można użyć w MySQL następującej składni:

```

ALTER TABLE Klient
ALTER miejscowosc SET DEFAULT 'Kraków';

```

W SQL Server należy użyć składni:

```
ALTER TABLE Klient
ADD CONSTRAINT df_miejsce
DEFAULT 'Kraków' FOR miejscowosc;
```

gdzie *df_miejsce* to nazwa ograniczenia, które zostało zdefiniowane dla kolumny *miejscowosc*.

UNIQUE

Atrybut `UNIQUE` jest stosowany, jeśli wartości w kolumnie nie mogą się powtarzać. Ograniczenie powtarzalności w kolumnie nie blokuje możliwości wpisania do niej wartości `NULL`. Atrybut `UNIQUE`, podobnie jak atrybut `PRIMARY KEY`, nie pozwala na umieszczanie w kolumnie wartości powtarzających się. Jednak w tabeli może być wiele atrybutów `UNIQUE`, ale tylko jeden atrybut `PRIMARY KEY`.

Przykład 3.18

Definiowanie wartości unikatowych w SQL Server podczas tworzenia tabeli:

```
CREATE TABLE Ksiazki
(
    ....
    tytul nvarchar (100) NOT NULL UNIQUE,
    ....
);
```

Przykład 3.19

Definiowanie wartości unikatowych w MySQL podczas tworzenia tabeli:

```
CREATE TABLE Ksiazki
(
    ....
    tytul nvarchar (100) NOT NULL,
    ....
    UNIQUE (tytul)
);
```

Przykład 3.20

Atrybut `UNIQUE` można ustawić w istniejącej tabeli, wpisując polecenie:

```
ALTER TABLE Ksiazki
ADD UNIQUE (tytul);
```

Ćwiczenie 3.8

Zmodyfikuj w tabeli *Klient* kolumny *PESEL* i *telefon* w taki sposób, aby można było wpisywać w nich tylko wartości niepowtarzające się.

Rozwiązanie

```
ALTER TABLE Klient
ADD UNIQUE (PESEL);

ALTER TABLE Klient
ADD UNIQUE (telefon);
```

Warunek logiczny CHECK

Atrybut *CHECK* pozwala na zdefiniowanie warunków ograniczających zakres danych wprowadzanych do kolumny. Dla każdej kolumny można definiować wiele warunków. Można również tworzyć za pomocą operatorów *NOT*, *AND* i *OR* złożone warunki ograniczające.

Przykład 3.21

Definiowanie ograniczenia *CHECK* w SQL Server podczas tworzenia tabeli:

```
CREATE TABLE Ksiazki
(
    . . . .
    rok_wydania int CHECK (rok_wydania BETWEEN 2015 AND 2020),
    . . . .
);
```

Definiowanie ograniczenia *CHECK* w MySQL jest możliwe od wersji 8.0.16.

Przykład 3.22

Ograniczenie *CHECK* definiowane podczas tworzenia tabeli:

```
CREATE TABLE Ksiazki
(
    . . . .
    cena decimal,
    CHECK (cena > 20),
    . . . .
);
```

Przykład 3.23

Atrybut CHECK, podobnie jak atrybut UNIQUE, można ustawić w istniejącej tabeli po poleceniu:

```
ALTER TABLE Ksiazki
ADD CHECK (rok_wydania BETWEEN 2015 AND 2020);
```

lub

```
ALTER TABLE Ksiazki
ADD CHECK (rok_wydania >= 202019);
```

Przykład 3.24

Istniejący atrybut CHECK można usunąć, stosując w SQL Server składnię:

```
ALTER TABLE Ksiazki
DROP CONSTRAINT nazwa_ograniczenia;
```

lub w MySQL

```
ALTER TABLE Ksiazki
DROP CHECK nazwa_ograniczenia;
```

Zadanie 3.1

W bazie danych *ksiegarnia* utwórz tabele *Rejestracja_zamowienia* i *Faktura* zgodnie ze schematem zamieszczonym na rysunku 4.1. Określ dopuszczalne wartości w kolumnach tabel i — jeżeli jest to wymagane — zdefiniuj klucze podstawowe.

3.5. Język manipulowania danymi (DML)

Jednym z zadań realizowanych przez język SQL jest pobieranie i modyfikowanie danych. Instrukcje do tego służące tworzą tak zwany język manipulowania danymi (ang. *Data Manipulation Language* — DML). Są to:

- SELECT — wybiera dane z bazy danych,
- INSERT — umieszcza nowe wiersze w tabeli,
- UPDATE — zmienia zawartość istniejącego wiersza,
- DELETE — usuwa wiersze z tabeli.

3.5.1. Instrukcja INSERT

Instrukcja INSERT służy do wstawiania nowych wierszy do tabeli i ma postać:

```
INSERT INTO nazwa_tabeli (kolumna1, kolumna2, ... )
VALUES (wartość1, wartość2, ...)
```

W wyniku wykonania instrukcji do tabeli zostanie dodany nowy wiersz. W polu *kolumna1* zostanie zapisana wartość *wartość1*, w polu *kolumna2* zostanie zapisana wartość *wartość2* itd.

Jeżeli jawnie nie podamy, do jakich kolumn powinny zostać wstawione wartości, to dane podane w klauzuli `VALUES` zostaną wstawione do kolejnych kolumn tabeli.

Przykład 3.25

```
INSERT INTO Ksiazki (tytul, id_autora, cena, wydawnictwo, temat)
VALUES ('Projektowanie stron internetowych', 1, 35, 'Helion', 'Internet');
```

Po wykonaniu tego polecenia do tabeli *Ksiazki* zostanie dodany nowy wiersz, a w polach *tytul*, *id_autora*, *cena*, *wydawnictwo*, *temat* zostaną wstawione podane ciągi znaków.

Jeżeli odczytamy wstawiony wiersz, okaże się, że oprócz wymienionych pól została wstawiona również wartość dla pola *id_ksiazki*, ponieważ podczas definiowania tabeli pole to zostało wybrane jako pole klucza podstawowego z automatycznym wstawianiem kolejnych wartości, poczynając od liczby 1 (`id_ksiazki int IDENTITY (1, 1) NOT NULL PRIMARY KEY`). Jeżeli dla jakiejś kolumny została zdefiniowana wartość domyślna (klauzula `DEFAULT`), to w analogiczny sposób zostanie ona automatycznie wstawiona do niej, chyba że podczas wstawiania wiersza podamy jej wartość.

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

Ze względu na to, że pojedyncze wstawianie wierszy jest uciążliwe, niektóre serwery pozwalają na wpisanie w klauzuli `VALUES` wartości, które zostaną umieszczone w kilku wierszach.

Przykład 3.26

```
INSERT INTO Ksiazki (tytul, id_autora, cena, wydawnictwo, temat)
VALUES ('Aplikacje internetowe', 2, 57, 'Helion', 'Internet'),
('Programowanie w PHP', 2, 72, 'Helion', 'Internet'),
('SQL Server 2008', 3, 45, 'PWN', 'Bazy danych');
```

W wyniku wykonania instrukcji do tabeli *Ksiazki* zostaną dodane trzy nowe wiersze z podanymi wartościami.

Przykład 3.27

```
INSERT INTO Klient (nazwisko, imie, PESEL)
VALUES ('Nowak', 'Andrzej', '78021203121'),
('Kowalski', 'Jan', '81092902821'),
('Górski', 'Antoni', '89120217239');
```

W wyniku wykonania instrukcji do tabeli *Klient* zostaną dodane trzy nowe wiersze z danymi klientów.

Ćwiczenie 3.9

Do tabeli *Klient* dodaj dane następujących klientów:

Adamska Anna, ul. Górna 7, 87-100 Toruń, tel. 123 456 789;

Bolecki Miłosz, ul. Nowa 24, 45-404 Opole, tel. 234 567 891;

Wilk Dawid, ul. Szeroka 1, 87-100 Toruń, tel. 345 678 912;

Żak Jan, ul. Mała 45, 45-404 Opole, tel. 456 789 123.

Rozwiązanie

```
INSERT INTO Klient (nazwisko, imie, ulica, nr_domu, kod_pocztowy,
miejscowosc, telefon)
VALUES
('Adamska ', 'Anna', 'Górna' , 7, '87-100', 'Toruń', '123456789'),
('Bolecki ', 'Miłosz', 'Nowa' , 24, '45-404', 'Opole', '234567891'),
('Wilk ', 'Dawid', 'Szeroka' , 1, '87-100', 'Toruń', '345678912'),
('Żak ', 'Jan', 'Mała' , 45, '45-404', 'Opole', '456789123');
```

3.5.2. Instrukcja UPDATE

Do aktualizowania danych służy instrukcja UPDATE w postaci:

```
UPDATE nazwa_tabeli
SET kolumna1=wartość, kolumna2=wartość, ...)
```

W klauzuli UPDATE podajemy nazwę tabeli, a w klauzuli SET nazwę modyfikowanej kolumny oraz przypisaną jej nową wartość. Warto pamiętać, że wykonanie takiej instrukcji spowoduje zmianę we wszystkich wierszach podanej kolumny. Dlatego jeżeli modyfikacja będzie dotyczyła tylko wybranych wierszy, do instrukcji należy dołączyć klauzulę WHERE.

Przykład 3.28

```
UPDATE Ksiazki SET jezyk_ksiazki ='polski'
WHERE rok_wydania>=2018;
```

W podanym przykładzie w tabeli *Ksiazki* kolumnie *jezyk_ksiazki* zostanie przypisana wartość *polski*, ale tylko dla książek wydanych w roku 2018 lub później.

Przy użyciu instrukcji UPDATE możliwe jest modyfikowanie wielu kolumn jednocześnie. Jest to zalecane rozwiązanie, ponieważ modyfikacja wielu kolumn za pomocą jednej

instrukcji jest dużo bardziej wydajna niż modyfikowanie pojedynczych kolumn z wykorzystaniem kilku operacji.

Przykład 3.29

```
UPDATE Klient SET kod_pocztowy = '87-100', miejscowosc = 'Toruń',
ulica='Szeroka', nr_domu=34
WHERE nazwisko='Nowak' AND imie='Andrzej';
```

W podanym przykładzie w tabeli *Klient* zostaną zmodyfikowane kolumny *kod_pocztowy*, *miejscowosc*, *ulica* i *nr_domu* dla klienta *Nowak Andrzej*.

Ćwiczenie 3.10

Zmodyfikuj dane następujących klientów:

Jan Kowalski mieszka w Warszawie na ulicy Mickiewicza 1, kod pocztowy 05-120.

Antoni Górski posiada telefon o numerze 987654321 i mieszka w Toruniu.

Rozwiązanie

```
UPDATE Klient SET kod_pocztowy = '05-120', miejscowosc = 'Warszawa',
ulica='Mickiewicza', nr_domu=1
WHERE nazwisko='Kowalski' AND imie='Jan';

UPDATE Klient SET kod_pocztowy = '87-100', miejscowosc = 'Toruń',
telefon='987654321'
WHERE nazwisko='Górski' AND imie='Antoni';
```

3.5.3. Instrukcja DELETE

Instrukcja DELETE usuwa wiersze z wybranej tabeli.

Aby usunąć wszystkie wiersze z tabeli, wystarczy w klauzuli FROM podać nazwę tabeli.

Przykład 3.30

Usunięcie wszystkich wierszy z tabeli *Klient*:

```
DELETE FROM Klient;
```

Po dodaniu klauzuli WHERE ze zdefiniowanym warunkiem z tabeli zostaną usunięte te wiersze, dla których warunek będzie prawdziwy.

Przykład 3.31

Usunięcie z tabeli *Klient* klientów mieszkających w Opolu:

```
DELETE FROM Klient
WHERE miejscowosc = 'Opole';
```

Jeżeli usunięcie danych mogłoby spowodować utratę spójności danych, instrukcja nie zostanie wykonana.

Ćwiczenie 3.11

Z tabeli *Klient* usuń dane klientów, którzy mieszkają w Toruniu i nie podali nazwy ulicy, przy której mieszkają.

Rozwiązanie

```
DELETE FROM Klient
WHERE miejscowosc = 'Toruń' AND ulica IS NULL;
```



3.6. Instrukcja SELECT

Instrukcja *SELECT* służy do wybierania danych z tabel bazy danych i określa, jakie dane zostaną zwrócone w wyniku jej wykonania. Najprostsza postać instrukcji wygląda następująco:

```
SELECT {nazwa_kolumny | wyrażenie} [[AS] nazwa_kolumny] | nazwa_kolumny =
wyrażenie } [, ... n]
FROM Nazwa_tabeli;
```

Przykład 3.32

```
SELECT nazwisko, imie
FROM Klient;
```

W podanym przykładzie instrukcja *SELECT* zwróci zawartość pól *nazwisko* i *imie* z tabeli *Klient*.

Wybieranie niektórych kolumn z tabeli nazywane jest **projekcją** lub **selekcją pionową** tabeli.

W poleceniu *SELECT* zamiast wypisywać listę wszystkich pól tabeli można użyć symbolu ***.

Przykład 3.33

Chcemy otrzymać wszystkie kolumny tabeli *Klient*.

```
SELECT *
FROM Klient;
```

Może się zdarzyć, że w wyniku zastosowania symbolu *** otrzymamy błędne wyniki. Sytuacja taka ma miejsce, gdy ktoś inny zmienia kolejność kolumn lub dodaje kolumnę do tabeli albo ją usuwa. Jeżeli chcemy odczytać zawartość tylko niektórych kolumn, nie powinniśmy definiować klauzuli odczytującej całą tabelę.

3.6.1. Konstrukcja zapytania

W instrukcji `SELECT` można umieszczać dodatkowe konstrukcje. Są to:

- podzapytania,
- zapytania połączone,
- dodatkowe klauzule, takie jak `DISTINCT`, `TOP`.

Podstawowe klauzule dla instrukcji `SELECT` to:

- `SELECT` — określenie kształtu wyniku, selekcja pionowa (wybór kolumn),
- `FROM` — określenie źródła (źródeł) danych i relacji zachodzących między nimi,
- `WHERE` — filtrowanie rekordów, definiuje warunek logiczny dla selekcji poziomej (wybór wierszy),
- `GROUP BY` — określenie grup rekordów (agregowanie pól podsumowywanych),
- `HAVING` — dla funkcji agregujących filtrowanie grup,
- `ORDER BY` — określenie sposobu sortowania wyniku.

Wykonywanie przez system baz danych wymienionych bloków instrukcji `SELECT` odbywa się w ściśle określonej, podanej niżej kolejności:

(krok 1.) `FROM` — określenie źródła (źródeł) danych i relacji zachodzących między nimi,

(krok 2.) `WHERE` — filtrowanie rekordów zgodnie ze zdefiniowanym warunkiem,

(krok 3.) `GROUP BY` — grupowanie rekordów,

(krok 4.) `HAVING` — filtrowanie utworzonych grup,

(krok 5.) `SELECT` — wybór kolumn, których zawartość zostanie zwrócona,

(krok 6.) `ORDER BY` — sortowanie wyniku.

Wynikiem przetwarzania każdego z kroków jest wirtualna tabela (VT). Jest to tak zwana konstrukcja pośrednia. Jest ona obiektem wejściowym dla kolejnego etapu. Konstrukcje pośrednie tworzy system baz danych i nie jest możliwy dostęp do nich.

3.6.2. Klauzula `DISTINCT`

Klauzula `DISTINCT` eliminuje z wyświetlania wyniku zapytania powtarzające się wiersze.

Przykład 3.34

Chcemy otrzymać informację o tym, z jakich miejscowości pochodzą klienci naszej księgarni.

```
SELECT DISTINCT kod_pocztowy, miejscowosc
FROM Klient;
```

3.6.3. Wyrażenia w instrukcji SELECT

W instrukcji `SELECT`, oprócz nazw kolumn, mogą występować wyrażenia. Tworzone są one z nazw kolumn, funkcji systemowych, stałych i operatorów i muszą zwracać pojedyncze wartości.

Przykład 3.35

Chcemy obliczyć marżę narzuconą na książki. Marża wynosi 7% ceny książki.

```
SELECT tytuł, cena, cena*0.07 AS Marża
FROM Ksiazki;
```

Wynik zostanie obliczony dla wszystkich wierszy tabeli *Ksiazki*.

Przykład 3.36

Chcemy uzyskać ostateczną cenę książki, uwzględniającą marżę.

```
SELECT tytuł, cena+cena*0.07 AS Cena_sprzedaży
FROM Ksiazki;
```

Ćwiczenie 3.12

Połącz dane klienta z kilku kolumn tabeli *Klient* w następujący sposób: łączymy pola *imie* i *nazwisko* pod nazwą *Klient*, pola *kod_pocztowy* i *miescowosc* pod nazwą *Miasto*, pola *ulica* i *nr_domu* pod nazwą *Adres*.

Rozwiązanie

Serwer SQL Server nie obsługuje operatora konkatenacji, lecz udostępnia operator `+`.

```
SELECT nazwisko+' '+imie AS Klient, kod_pocztowy+' '+miescowosc AS Miasto,
ulica+' '+nr_domu AS Adres
FROM Klient;
```

Serwer MySQL nie obsługuje ani operatora konkatenacji, ani operatora `+`. Do łączenia ciągów tekstowych służy funkcja `CONCAT(ciąg_tekstowy, [ciąg_tekstowy, ...])`.

```
SELECT CONCAT(nazwisko, ' ', imie) AS Klient, CONCAT(kod_pocztowy, ' ',
miescowosc) AS Miasto, CONCAT(ulica, ' ', nr_domu) AS Adres
FROM Klient;
```

Ćwiczenie 3.13

Ile dni dla każdego zamówienia upłynęło między datą złożenia zamówienia a datą wysłania książek? Podaj datę złożenia zamówienia, datę wysłania oraz liczbę dni, które upłynęły między tymi datami.

Rozwiązanie

```
SELECT data_zlozenia_zamowienia, data_wyslania,
       DateDiff(d, data_wyslania, data_zlozenia_zamowienia)
FROM Zamowienia;
```

W rozwiązaniu została wykorzystana funkcja `DateDiff`, która zwraca informację, ile czasu upłynęło między dwiema datami, i jest zapisywana w postaci:

`DATEDIFF`(interwał czasowy, data początkowa, data końcowa)

Interwałem czasowym może być np. dzień (dd, d), tydzień (wk, ww), miesiąc (mm, m) lub rok (yy, yyyy). Interwał czasowy nie występuje w funkcji `DateDiff()`, stosowanej na serwerze MySQL.

Często stosowane funkcje związane z datą to funkcje systemowe: `GetDate()` zwracająca datę i `SysDateTime()` zwracająca czas oraz `Year(data)` zwracająca rok, `Month(data)` zwracająca miesiąc i `Day(data)` zwracająca dzień.

3.6.4. Klauzula ORDER BY — sortowanie

W celu posortowania danych należy dodać klauzulę `ORDER BY`. W klauzuli umieszcza się nazwy lub numery kolumn, według których nastąpi sortowanie.

Przykład 3.37

```
SELECT tytuł, cena
FROM Ksiazki
ORDER BY tytuł;
```

Książki zostaną posortowane według kolumny *tytuł*.

Inny przykład:

```
SELECT tytuł, cena
FROM Ksiazki
ORDER BY 2;
```

Książki zostaną posortowane według kolumny *cena*.

Domyślnie dane są sortowane rosnąco. Do tego rodzaju sortowania można użyć również słowa kluczowego `ASC`. Aby posortować dane malejąco, należy po nazwie lub jej numerze użyć słowa kluczowego `DESC`.

Przykład 3.38

```
SELECT tytuł, cena
FROM Ksiazki
ORDER BY cena DESC;
```

Książki zostaną ustawione w kolejności od najdroższej do najtańszej.

Dane mogą być sortowane według wielu kolumn. Klauzule definiujące sposób sortowania są od siebie niezależne. Można również odwoływać się do kolumn niewymienionych w klauzuli `SELECT`.

Przykład 3.39

```
SELECT tytuł, rok_wydania, cena
FROM Książki
ORDER BY rok_wydania ASC, cena DESC;
```

Sortowanie zwykle wydłuża czas wykonywania zapytania, dlatego jeżeli wynik zapytania nie musi być posortowany, należy unikać używania klauzuli `ORDER BY`.

Ćwiczenie 3.14

Wyświetl informacje o klientach (*nazwisko*, *imie*, *mięscowosc*, *PESEL*). Dane posortuj malejąco według miejscowości, a klientów z tej samej miejscowości posortuj według numeru PESEL od osoby najstarszej do najmłodszej.

Rozwiązanie

```
SELECT nazwisko, imie, miejscowosc, PESEL
FROM Klient
ORDER BY miejscowosc DESC, PESEL;
```

3.6.5. Klauzula `WHERE` — wybieranie wierszy

Projektując zapytanie, najczęściej chcemy ograniczyć wynik do interesujących nas danych. Do zwracania tylko wybranych wierszy służy poznana już klauzula `WHERE`.

Wybieranie niektórych wierszy z tabeli nazywane jest **selekcją rekordów**.

Klauzula `WHERE` musi wystąpić bezpośrednio po klauzuli `FROM`.

Przykład 3.40

```
SELECT tytuł, cena
FROM Książki
WHERE rok_wydania=2019;
```

lub:

```
SELECT tytuł, cena
FROM Książki
WHERE cena BETWEEN 50 AND 100;
```

Przykład 3.41

Chcemy odczytać nazwiska klientów zaczynające się na literę **A**.

```
SELECT nazwisko, imie
FROM Klient
WHERE nazwisko LIKE 'A%';
```

Przykład 3.42

Chcemy odczytać nazwiska klientów, którzy nie podali numeru telefonu.

```
SELECT nazwisko, imie
FROM Klient
WHERE telefon IS NULL;
```

Przykład 3.43

Złożony warunek logiczny:

```
SELECT nazwisko, imie
FROM Klient
WHERE miejscowosc = 'Warszawa' AND adres_e_mail IS NULL;
```

W wyniku zostaną odczytane nazwiska i imiona klientów, którzy mieszkają w Warszawie i nie podali swojego adresu e-mail.

Klauzula `WHERE` może być używana na podobnych zasadach w instrukcjach `UPDATE` i `DELETE`. Jej składnia jest we wszystkich przypadkach taka sama.

Zadanie 3.2

1. Wyświetl tytuły książek wydawnictwa Helion.
2. Wyświetl tytuły książek, których cena jest większa od 50 zł i mniejsza od 120 zł.
3. Wyświetl nazwiska i imiona autorów, którzy napisali książki w języku angielskim lub niemieckim.
4. Wyświetl informacje o autorach, którzy mają jakiekolwiek osiągnięcia. Wyświetl informację o tych osiągnięciach.
5. Wyświetl informację o książkach (tytuł, temat, wydawnictwo, miejsce wydania, język), które zostały napisane w języku polskim, a nie podano dla nich tematu i wydawnictwa.
6. Wyświetl informację o książkach (tytuł, cena, wydawnictwo, temat), które są powieścią lub dramatem i ich cena przekracza 50 zł.
7. Wyświetl informację o książkach (tytuł, temat, język), w których tytule w dowolnym miejscu występuje słowo „pan”.
8. Z jakich wydawnictw pochodzą książki zgromadzone w bazie danych?

Rozwiązanie

Ad 1.

```
SELECT tytuł
FROM Ksiazki
WHERE wydawnictwo='Helion';
```

Ad 2.

```
SELECT tytuł
FROM Ksiazki
WHERE cena BETWEEN 50 AND 120;
```

Ad 3.

```
SELECT nazwisko, imie
FROM Autor
WHERE język='angielski' OR język='niemiecki';
```

Ad 4.

```
SELECT nazwisko, imie, osiagniecia
FROM Autor
WHERE osiagniecia IS NOT NULL;
```

Ad 5.

```
SELECT tytuł, temat, wydawnictwo, miejsce_wydania, język_ksiazki
FROM Ksiazki
WHERE język_ksiazki='polski' AND temat IS NULL AND wydawnictwo IS NULL;
```

Ad 6.

```
SELECT tytuł, cena, wydawnictwo, temat
FROM Ksiazki
WHERE (temat='powieść' OR temat='dramat') AND cena>50;
```

Ad 7.

```
SELECT tytuł, temat, język_ksiazki
FROM Ksiazki
WHERE tytuł LIKE '%pan%';
```

Ad 8.

```
SELECT DISTINCT wydawnictwo
FROM ksiazki;
```

3.6.6. Klauzula TOP

Klauzula `TOP` służy do wybrania określonej liczby wierszy. Liczba wierszy może być podana jawnie lub procentowo. Klauzula `TOP` musi wystąpić bezpośrednio po instrukcji `SELECT` przed nazwami kolumn. Nie wszystkie systemy bazodanowe obsługują tę klauzulę.

Przykład 3.44

```
SELECT TOP 1 tytuł, wydawnictwo, rok_wydania, cena
FROM Ksiazki
ORDER BY cena DESC;
```

Zostanie zwrócony wiersz opisujący najdroższą książkę w bazie.

Zwykle razem z klauzulą `TOP` występuje klauzula `ORDER BY`. Bez niej klauzula `TOP` jest właściwie bezużyteczna, ponieważ to klauzula `ORDER BY` określa kolejność wierszy wyświetlanych w wyniku zapytania.

Otrzymany wynik jest poprawny wyłącznie wtedy, gdy w bazie danych jest tylko jedna książka z tą ceną. A jeżeli książek z tą samą ceną jest więcej?

Wówczas można użyć rozszerzonej składni klauzuli `TOP` w postaci: `TOP n WITH TIES`. W wyniku zostaną zwrócone wszystkie wiersze z tą samą najwyższą ceną książki. Użycie rozszerzonej składni klauzuli `TOP` pokazuje przykład podany niżej.

Przykład 3.45

```
SELECT TOP 1 WITH TIES tytuł, cena
FROM Ksiazki
ORDER BY cena DESC;
```

Zostaną zwrócone wszystkie wiersze opisujące książki o tej samej najwyższej cenie.

Na serwerze MySQL zamiast klauzuli `TOP` stosowana jest klauzula `LIMIT`.

Przykład 3.46

```
SELECT tytuł, wydawnictwo, rok_wydania, cena
FROM Ksiazki
ORDER BY cena DESC
LIMIT 1;
```

3.6.7. Grupowanie danych

Funkcje agregujące

Funkcje agregujące działają na wartościach wybranego pola w grupie wierszy. Wynikiem może być suma, średnia, liczba wierszy, wartość maksymalna lub minimalna. Funkcje te zwracają pojedyncze wartości i są wywoływane w instrukcji `SELECT`. Funkcje agregujące są jednymi z najważniejszych narzędzi relacyjnych baz danych.

Podstawowe funkcje agregujące to:

- `COUNT(nazwa_kolumny)` — zwraca liczbę wierszy w grupie,
- `SUM(nazwa_kolumny)` — zwraca sumę wartości w grupie dla wskazanej kolumny,
- `AVG(nazwa_kolumny)` — zwraca średnią wartości w grupie dla wskazanej kolumny,
- `MAX(nazwa_kolumny)` — zwraca największą wartość w grupie dla wskazanej kolumny,
- `MIN(nazwa_kolumny)` — zwraca najmniejszą wartość w grupie dla wskazanej kolumny.

Oprócz powyższych funkcji agregujących niektóre serwery bazodanowe akceptują inne, mniej popularne funkcje.

Funkcja `COUNT` może zostać wywołana z symbolem `*` zamiast nazwy kolumny.

Przykład 3.47

```
SELECT COUNT(*) AS 'Liczba klientów'
FROM Klient;
```

W wyniku wykonania polecenia zostaną policzone wszystkie wiersze tabeli *Klient*, czyli otrzymamy odpowiedź na pytanie: „Ilu klientów jest zarejestrowanych w bazie danych *ksiegarnia*?”. W wyniku takiego wywołania funkcji `COUNT()` zostaną policzone także puste wiersze. Jest to jedyny przypadek, gdy funkcja agregująca uwzględnia wartość `NULL`.

Natomiast wywołanie tej funkcji z jawnym podaniem nazwy kolumny spowoduje pominięcie wierszy, dla których wybrana kolumna ma wartość `NULL`.

Przykład 3.48

```
SELECT COUNT(telefon) AS 'Liczba klientów'
FROM Klient;
```

Jeżeli dla niektórych klientów kolumna *telefon* nie została wypełniona, zostaną oni pominięci w obliczeniach.

Przykład 3.49

```
SELECT AVG(cena) AS 'Średnia cena książek'
FROM Książki;
```


Wynikiem będzie średnia cena książek znajdujących się w księgarni internetowej.

Jeżeli funkcja agregująca w swoich obliczeniach uwzględnia tylko wartości niepowtarzające się, to argumentem funkcji staje się słowo kluczowe `DISTINCT`.

Przykład 3.50

```
SELECT COUNT(DISTINCT miejscowosc) AS 'Liczba miejscowości'
FROM Klient;
```

W wyniku otrzymamy liczbę miejscowości, z których pochodzą klienci księgarni internetowej. Słowo kluczowe `DISTINCT` zostało umieszczone w nawiasie `()` jako argument funkcji `COUNT()`, ponieważ dotyczy kolumny *miejscowosc*, a nie tej funkcji.

Jeżeli w poleceniu dodamy klauzulę ograniczającą `WHERE`, to obliczenia będą dotyczyły tylko wierszy, które spełniają warunek zdefiniowany w klauzuli.

Przykład 3.51

```
SELECT COUNT(tytul) AS 'Liczba tytułów'
FROM Ksiazki
WHERE rok_wydania>2017;
```

Wynikiem będzie liczba książek wydanych po roku 2017.

Funkcje agregujące mogą być częścią wyrażeń.

Przykład 3.52

```
SELECT MAX(cena) - MIN(cena)
FROM Ksiazki;
```

W wyniku otrzymamy różnicę między maksymalną i minimalną ceną książki.

Zadanie 3.3

1. Ile tytułów książek znajduje się w bazie danych?
2. Ile razy w księgarni internetowej zostały wystawione faktury?
3. Ile zamówień zostało złożonych w sierpniu 2019 roku?
4. Z ilu wydawnictw pochodzą książki dostępne w bazie danych?

Rozwiązanie

Ad 1.

```
SELECT COUNT(tytul) AS 'Liczba tytułów'
FROM Ksiazki;
```

Ad 2.

```
SELECT COUNT(id_faktury) AS 'Liczba faktur'
FROM Zamowienia;
```

Ad 3.

```
SELECT COUNT(id_zamowienia) AS 'Liczba zamówień'
FROM zamowienia
WHERE data_zlozenia_zamowienia>='2019-08-01'
AND data_zlozenia_zamowienia<='2019-08-31';
```

Ad 4.

```
SELECT COUNT(DISTINCT wydawnictwo) AS 'Liczba wydawnictw'
FROM Ksiazki;
```

Klauzula GROUP BY

Funkcje agregujące mają zastosowanie głównie dla danych, które zostały pogrupowane. Do grupowania wierszy służy klauzula GROUP BY.

Przykład 3.53

Mamy policzyć książki o tym samym temacie i znaleźć trzy tematy z największą liczbą książek w bazie danych.

```
SELECT TOP 3 COUNT(tytul), temat
FROM Ksiazki
GROUP BY temat
ORDER BY 1 DESC;
```

Użycie klauzuli GROUP BY spowoduje pogrupowanie wierszy z tabeli *Ksiazki* według wartości w kolumnie *temat*. Funkcja COUNT() zliczy wiersze w każdej grupie, klauzula ORDER BY uporządkuje otrzymane wyniki od największej (DESC) do najmniejszej wartości w kolumnie *1*, a klauzula TOP 3 ograniczy wynik zapytania do trzech pierwszych wierszy.

Przykład 3.54

Rozwiązanie tego samego problemu w MySQL:

```
SELECT COUNT(tytul), temat
FROM Ksiazki
GROUP BY temat
ORDER BY 1 DESC
LIMIT 3;
```

Klauzula HAVING

Klauzula HAVING jest ściśle powiązana z klauzulą GROUP BY. Określa, które wiersze zostaną zwrócone przez klauzulę GROUP BY.

Przykład 3.55

Chcemy otrzymać informację o tematach książek, dla których w bazie danych jest co najmniej pięć tytułów.

```
SELECT temat, COUNT(tytul)
FROM Ksiazki
GROUP BY temat
HAVING COUNT(tytul)>=5
ORDER BY 2 DESC;
```

Istotne jest zrozumienie różnicy między klauzulami `WHERE` i `HAVING`. Klauzula `WHERE` filtruje wiersze przed grupowaniem i obliczeniami, tym samym określa, dla których wierszy funkcje agregujące będą wykonywały obliczenia. Klauzula `HAVING` wybiera wiersze już pogrupowane, po wykonaniu obliczeń przez funkcje agregujące. Klauzula `WHERE` nie może zawierać funkcji agregujących. Klauzula `HAVING` zawsze zawiera funkcje agregujące. Można użyć klauzuli `HAVING` bez funkcji agregujących, ale wykona ona wtedy te same działania co klauzula `WHERE` z tym samym warunkiem i będzie od niej mniej efektywna.

Podsumowując, klauzula `WHERE` pozwala filtrować wiersze, natomiast klauzula `HAVING` pozwala filtrować grupy zwracane przez zapytanie.

Przykład 3.56

Chcemy otrzymać informację o tematach, dla których w bazie danych jest co najmniej pięć tytułów książek wydanych po roku 2017.

```
SELECT temat, COUNT(tytul)
FROM Ksiazki
WHERE rok_wydania>2017
GROUP BY temat
HAVING COUNT(tytul)>=5
ORDER BY 2 DESC;
```

W podanym przykładzie w pierwszym kroku zostanie określona tabela *Ksiazki* jako źródło danych (klauzula `FROM`). W drugim kroku zostaną wybrane książki, które w polu *rok_wydania* mają zapisaną wartość 2017 (klauzula `WHERE`). W trzecim kroku otrzymane książki zostaną pogrupowane według pola *temat* (klauzula `GROUP BY`) i za pomocą funkcji agregującej `COUNT()` zostanie policzone, ile książek znajduje się w każdej grupie (w każdym temacie). W czwartym kroku zostaną wybrane te grupy (tematy), dla których liczba książek wynosi co najmniej pięć egzemplarzy (klauzula `HAVING`). Dopiero w piątym kroku zostaną wybrane kolumny do zwrócenia instrukcją `SELECT`, czyli temat i liczba książek o podanej tematyce. W ostatnim kroku te dane zostaną uporządkowane malejąco według kolumny z liczbą książek (klauzula `ORDER BY`). Tak przygotowane dane zostaną zwrócone jako wynik wykonania instrukcji `SELECT`.

Zadanie 3.4

1. Znajdź trzy miejscowości, z których pochodzi największa liczba klientów zarejestrowanych w bazie. Podaj nazwę miejscowości i liczbę klientów.
2. Wyświetl informację o średniej cenie książek z każdego wydawnictwa, dla których został podany temat książki. Pomiń wydawnictwa, dla których średnia cena książek jest niższa od 30 zł. Podaj nazwę wydawnictwa oraz średnią cenę książek z tego wydawnictwa.
3. Wyświetl informację o liczbie tytułów książek wydanych po roku 2018 z każdego z wydawnictw dostępnych w bazie danych, jeżeli tych tytułów jest więcej niż dwa. Pomiń tytuły książek, dla których nie podano wydawnictwa. Wynik posortuj alfabetycznie według wydawnictw.



3.7. Łączenie tabel

Omawiane do tej pory zapytania dotyczyły pojedynczych tabel, ale wiemy, że istotą relacyjnych baz danych jest odwoływanie się w zapytaniach do wielu powiązanych ze sobą tabel. Połączenia realizowane są przez porównanie wartości kolumny lub kilku kolumn z jednej tabeli z podobnymi kolumnami z drugiej tabeli.

Połączenia są najważniejszym mechanizmem relacyjnych baz danych. Dzięki nim możemy wybierać pasujące do siebie dane z wielu tabel, tworzyć raporty i podsumowania danych pochodzących z wielu tabel. Poprawne tworzenie połączeń jest podstawą projektowania i tworzenia profesjonalnych systemów baz danych.

Połączenia są realizowane między kluczem podstawowym jednej tabeli i kluczem obcym drugiej. Klucz obcy jest kolumną lub kombinacją kolumn, które są kluczem podstawowym w innej tabeli.

Wartości klucza podstawowego są niepowtarzalne, natomiast w kolumnie klucza obcego każda z wartości może powtórzyć się wielokrotnie. Wtedy tworzone powiązanie jest związkiem typu „jeden do wielu”.

3.7.1. Połączenie wewnętrzne i zewnętrzne

W języku SQL połączenie między tabelami jest definiowane w sekcji `FROM` zapytania. Słowo kluczowe `INNER JOIN` definiuje połączenie między tabelami. Klauzula realizująca połączenie ma postać:

```
tabela1 INNER JOIN tabela2
ON tabela1.kolumna1= tabela2.kolumna2
```

Przykład 3.57

```
SELECT Ksiazki.tytul, Autor.nazwisko, Autor.imie
FROM Ksiazki INNER JOIN Autor
ON Ksiazki.id_autora=Autor.id_autora
```

Aby prawidłowo utworzyć zapytanie, należy zdefiniować połączenia między tabelami bazy danych. Definicji połączeń można używać wielokrotnie, chociaż wpłynie to na szybkość działania zapytania.

Przedstawione w przykładzie połączenie jest **połączeniem wewnętrznym** (INNER JOIN). Oznacza to, że wynikiem zapytania są tylko wiersze zawierające w polu klucza podstawowego i w polu klucza obcego pasujące do siebie dane. Połączenie wewnętrzne jest domyślnym typem połączenia.

Innym rodzajem połączenia jest **połączenie zewnętrzne** (OUTER JOIN). Przy takim połączeniu wynikiem zapytania są wszystkie wiersze z jednej tabeli i pasujące do nich wiersze z drugiej tabeli.

Złączenia *zewnętrzne* dzielimy na:

- LEFT OUTER JOIN — zapytanie zwraca wszystkie wiersze z pierwszej tabeli i pasujące wiersze z drugiej tabeli,
- RIGHT OUTER JOIN — zapytanie zwraca wszystkie wiersze z drugiej tabeli i pasujące wiersze z pierwszej tabeli,
- FULL OUTER JOIN — zapytanie zwraca wszystkie pasujące i niepasujące wiersze z obu tabel (tylko SQL Server).

Gdybyśmy chcieli uzyskać listę wszystkich zamówień zrealizowanych w księgarni internetowej wraz z numerami wystawionych faktur, to przy zastosowaniu domyślnego połączenia między tabelami *Zamowienia* i *Faktura* (INNER JOIN) nie uzyskalibyśmy informacji o zamówieniach, dla których nie wystawiono faktury.

Jeżeli zastosujemy połączenie zewnętrzne, otrzymamy wszystkie wiersze z tabeli *Zamowienia* oraz numery faktur dla zamówień, dla których faktury zostały wystawione.

Przykład 3.58

```
SELECT Zamowienia.data_zlozenia_zamowienia, Zamowienia.koszt_wyslki, Fak-
tura.nr_faktury, Faktura.sposob_platnosci
FROM Zamowienia LEFT OUTER JOIN Faktura
ON Zamowienia.id_faktury=Faktura.nr_faktury;
```

Tak jak lewostronne połączenie zewnętrzne (LEFT OUTER JOIN) zwraca wszystkie wiersze z lewej tabeli, tak prawostronne połączenie zewnętrzne (RIGHT OUTER JOIN) zwraca wszystkie wiersze z prawej tabeli. Zmieniając kolejność tabel w klauzuli FROM, można zastąpić połączenie lewostronne połączeniem prawostronnym.

Przykład 3.59

```
SELECT Zamowienia.data_zlozenia_zamowienia, Zamowienia.koszt_wyslki,
Faktura.nr_faktury, Faktura.sposob_platnosci
FROM Faktura RIGHT OUTER JOIN Zamowienia
ON Faktura.nr_faktury=Zamowienia.id_faktury;
```

Połączenie zewnętrzne obustronne (FULL OUTER JOIN) zwraca wszystkie wiersze obu połączonych tabel, również te, które nie spełniają warunku połączenia.

Przykład 3.60

Jeżeli przyjmimy, że w tabeli *Zamowienia* znajdują się zamówienia, dla których nie zostały wystawione faktury, a w tabeli *Faktura* znajdują się informacje o fakturach, które nie są powiązane z żadnym zamówieniem, to zapytanie, które wyświetli informację o wszystkich zamówieniach i fakturach, będzie miało postać:

```
SELECT Zamowienia.data_zlozenia_zamowienia, Zamowienia.koszt_wysylki, Fak-
tura.nr_faktury, Faktura.sposob_platnosci
FROM Zamowienia FULL OUTER JOIN Faktura
ON Zamowienia.id_faktury=Faktura.nr_faktury;
```

3.7.2. Połączenie krzyżowe

Wszystkie możliwe połączenia wierszy dwóch tabel nazywamy *iloczynem kartezjańskim* lub *połączeniem krzyżowym* (CROSS JOIN). W tego typu połączeniu nie określa się warunku połączenia. Połączeniem krzyżowym można połączyć dowolne dwie tabele. Wynik tak zaprojektowanego zapytania dla tabel o pięciu i dziesięciu wierszach to tabela o 50 wierszach. Przy większej liczbie wierszy w tabelach wynik może być bardzo duży.

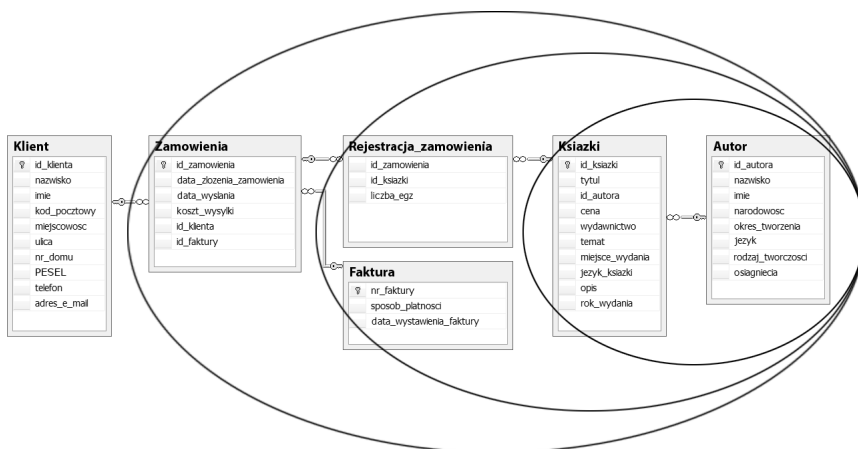
Tego typu połączenia są rzadko stosowane w relacyjnych bazach danych.

Przykład 3.61

```
SELECT Klient.nazwisko, Klient.imie, Zamowienia.data_zlozenia_zamowienia,
Zamowienia.koszt_wysylki
FROM Klient CROSS JOIN Zamowienia;
```

3.7.3. Połączenia wielokrotne

Projektując zapytanie, możemy definiować w nim dowolną liczbę połączeń między tabelami. Maksymalna liczba dopuszczalnych połączeń zależy od serwera bazodanowego. Przy ich definiowaniu należy pamiętać, że dołączenie w zapytaniu każdej następnej tabeli powoduje zmniejszenie wydajności zapytania. Mechanizm obsługiwanego przez serwer zapytań zawierających zdefiniowane połączenie między tabelami działa tak, że zawsze łączone są dwie tabele. Po połączeniu dwóch pierwszych tabel powstaje struktura pośrednia, która jest łączona z kolejną tabelą i tworzona jest kolejna struktura pośrednia. Tworzenie struktur pośrednich trwa aż do połączenia wszystkich tabel (rysunek 3.5).



Rysunek 3.5. Mechanizm obsługiwan przez serwer połączeń wielokrotnych

Przykład 3.62

Mamy prześledzić historię zamówień złożonych w księgarni w czerwcu 2019 roku. Wynikiem zapytania ma być nazwisko i imię klienta, data złożenia zamówienia, liczba zamówionych egzemplarzy książki oraz tytuł i nazwisko autora książki.

```

SELECT Klient.nazwisko, Klient.imie, Zamowienia.data_zlozenia_zamowienia,
Rejestracja_zamowienia.liczba_egz, Ksiazki.tytul, Autor.nazwisko, Autor.imie
FROM Klient INNER JOIN Zamowienia
ON Klient.id_klienta=Zamowienia.id_klienta
INNER JOIN Rejestracja_zamowienia
ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia
INNER JOIN Ksiazki
ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki
INNER JOIN Autor
ON Ksiazki.id_autora=Autor.id_autora
WHERE Zamowienia.data_zlozenia_zamowienia BETWEEN '2019-06-1' AND
'2019-06-30';

```

3.7.4. Złączenie tabeli z nią samą

Złączenie tabeli z nią samą jest definiowane w podobny sposób jak połączenia różnych tabel. Przy łączeniu tej samej tabeli jej nazwy w klauzuli FROM byłyby takie same. Aby odróżnić je od siebie, należy nadać im nowe nazwy (*aliasy*).

Przykład 3.63

```

SELECT K1.nazwisko, K1.imie
FROM Klient AS K1 CROSS JOIN Klient AS K2;

```

Ponieważ w obu wirtualnych tabelach istnieją kolumny *nazwisko* i *imie*, nazwy te są niejednoznaczne i muszą być poprzedzone nazwą tabeli.

Taki sposób połączenia tabeli z nią samą może zostać wykorzystany do wykrycia klientów, którzy zarejestrowali się w bazie danych kilkakrotnie.

Przykład 3.64

```
SELECT K1.nazwisko, K1.imie
FROM Klient AS K1 CROSS JOIN Klient AS K2
WHERE K1.nazwisko=K2.nazwisko AND K1.imie=K2.imie AND K1.PESEL=K2.PESEL;
```

Ćwiczenie 3.15

1. Ile wynosi całkowity koszt każdego zamówienia? Podaj nazwisko i imię klienta, który złożył zamówienie, datę jego złożenia oraz cenę zamówienia.
2. Podaj listę wszystkich książek wydanych w bieżącym roku. Na liście umieść tytuł książki oraz nazwisko i imię autora połączone w jednym polu.

Rozwiązanie

Ad 1.

```
SELECT Klient.nazwisko, Klient.imie, Zamowienia.data_zlozenia_zamowienia,
zamowienia.koszt_wysylki+Ksiazki.cena*Rejestracja_zamowienia.liczba_egz
FROM
Klient INNER JOIN (Zamowienia INNER JOIN (Rejestracja_zamowienia
INNER JOIN Ksiazki
ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki)
ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia)
ON Klient.id_klienta=Zamowienia.id_klienta;
```

Ad 2.

```
SELECT Ksiazki.tytul, Autor.nazwisko+' '+ Autor.imie AS Autor,
YEAR (GETDATE()) AS 'Rok wydania'
FROM Ksiazki INNER JOIN Autor
ON Ksiazki.id_autora=Autor.id_autora
WHERE rok_wydania= YEAR (GETDATE());
```


Zadanie 3.5

1. Ile razem kosztują wszystkie książki każdego autora? Podaj nazwisko autora oraz kwotę, za którą można kupić wszystkie jego książki.
2. Podaj nazwiska i imiona oraz miejsca zamieszkania klientów, którzy zarejestrowali się w księgarni, ale nie kupili żadnej książki.
3. Podaj tytuły oraz nazwiska i imiona autorów książek, które ani razu nie zostały sprzedane.
4. Podaj tytuł oraz nazwisko i imię autora książki, która była najczęściej sprzedawana.
5. Podaj informacje o wszystkich miejscowościach, w których mieszkają klienci księgarni i których kod pocztowy zaczyna się od cyfry 0.

Rozwiązanie

Ad 1.

```
SELECT autor.nazwisko, sum(cena)
FROM Ksiazki INNER JOIN Autor
ON Ksiazki.id_autora=Autor.id_autora
GROUP BY (nazwisko);
```

Ad 2.

```
SELECT Klient.nazwisko, Klient.imie, Klient.miejscowosc, Zamowienia.
data_zlozenia_zamowienia
FROM Klient LEFT OUTER JOIN Zamowienia
ON Klient.id_klienta=Zamowienia.id_klienta
WHERE Zamowienia.data_zlozenia_zamowienia IS NULL;
```

Ad 3.

```
SELECT Ksiazki.tytul, Autor.nazwisko, Autor.imie,
Zamowienia.data_zlozenia_zamowienia
FROM Zamowienia INNER JOIN Rejestracja_zamowienia
ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia
RIGHT OUTER JOIN Ksiazki
ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki
INNER JOIN Autor
ON Ksiazki.id_autora=Autor.id_autora
WHERE Zamowienia.data_zlozenia_zamowienia IS NULL;
```

Ad 4.

```
SELECT TOP 1 WITH TIES ksiazki.tytul, autor.nazwisko, autor.imie,
count (tytul)
```

```
FROM Rejestracja_zamowienia INNER JOIN Ksiazki
ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki
INNER JOIN Autor
ON Ksiazki.id_autora=Autor.id_autora
GROUP BY tytul, autor.nazwisko, autor.imie
ORDER BY 4 DESC;
```



3.8. Więzy integralności (MS SQL)

Dane przechowywane w bazie danych powinny spełniać wymogi integralności wynikające z założeń przyjętych podczas projektowania bazy.

3.8.1. Definiowanie klucza obcego

Jeżeli w bazie danych *ksiegarnia* do tabeli *Ksiazki* zostanie wpisana nowa książka z numerem autora 27, a autora o numerze 27 nie ma w bazie, to znaczy, że powstał błąd, który łatwo popełnić przy wprowadzaniu danych. Aby tego uniknąć, należy odpowiednio zdefiniować więzy integralności — wtedy serwer bazodanowy będzie automatycznie sprawdzał poprawność dokonywanych wpisów.

Sprawdzanie spójności w bazie danych odbywa się po jawnym zdefiniowaniu klucza obcego. Dla tabeli *Ksiazki* możemy zdefiniować klauzulę, która poinformuje bazę, że *id_autora* w tej tabeli to klucz obcy pochodzący z kolumny *id_autora* w tabeli *Autor*. Klauzulę dotyczącą ograniczeń klucza obcego trzeba umieścić za definicją kolumn w poleceniu CREATE TABLE lub ALTER TABLE.

Ogólna postać polecenia wygląda następująco:

```
[CONSTRAINT nazwa] FOREIGN KEY (kolumna1, kolumna 2, ...)
REFERENCE nazwa_tabeli (kolumna1, kolumna 2, ...)
```

gdzie:

- CONSTRAINT *nazwa* jest nazwą ograniczenia, może zostać pominięta — wtedy ograniczeniu zostanie nadana nazwa systemowa;
- FOREIGN KEY (*kolumna1*, *kolumna 2*, ...) określa kolumny zawierające klucz obcy;
- REFERENCE *nazwa_tabeli* (*kolumna1*, *kolumna 2*, ...) określa, z której tabeli pochodzi klucz obcy i które kolumny są w niej kluczem podstawowym.

Przykład 3.65

Podczas tworzenia tabeli *Ksiazki* należy zdefiniować klauzulę ograniczeń klucza obcego dla kolumny *id_autora*:

```
CREATE TABLE Ksiazki
(
```

```

    id_ksiazki int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    tytul varchar (100) NOT NULL,
    id_autora int REFERENCES Autor (id_autora),
    cena money,
    rok_wydania varchar (4),
    wydawnictwo varchar (20),
    temat varchar (30),
    miejsce_wydania varchar (28),
    jezyk_ksiazki varchar (15),
    opis varchar (100)
);

```

lub w MySQL:

```

CREATE TABLE Ksiazki
(
    id_ksiazki int AUTO_INCREMENT NOT NULL PRIMARY KEY,
    tytul varchar (100) NOT NULL,
    id_autora int REFERENCES Autor (id_autora),
    cena decimal,
    rok_wydania varchar (4),
    wydawnictwo varchar (20),
    temat varchar (30),
    miejsce_wydania varchar (28),
    jezyk_ksiazki varchar (15),
    opis varchar (100)
);

```

Przykład 3.66

W istniejącej tabeli (*Ksiazki*) można definiować klauzulę ograniczeń klucza obcego dla nowej kolumny (*id_autora*) podczas dodawania tej kolumny do tabeli:

```

ALTER TABLE Ksiazki
ADD id_autora int REFERENCES Autor (id_autora);

```

Przykład 3.67

W istniejącej tabeli (*Ksiazki*) dla istniejącego pola klucza obcego (*id_autora*) należy zdefiniować klauzulę ograniczeń klucza obcego przez zmodyfikowanie kolumny:

```
ALTER TABLE Ksiazki
ADD CONSTRAINT Ksiazki_FK FOREIGN KEY (id_autora)
REFERENCES Autor (id_autora);
```

Klauzula `REFERENCES` we wszystkich przypadkach zdefiniuje ograniczenie dla klucza obcego. Nie będzie możliwe wpisanie w kolumnie `id_autora` tabeli *Ksiazki* wartości, która nie istnieje w kolumnie `id_autora` tabeli *Autor*.

W klauzuli można również zdefiniować nazwę ograniczenia klucza obcego.

Przykład 3.68

```
ALTER TABLE Ksiazki
ADD id_autora int CONSTRAINT Ksiazki_FK
REFERENCES Autor (id_autora);
```

Klauzula `CONSTRAINT Ksiazki_FK` nadaje ograniczeniu klucza obcego nazwę *Ksiazki_FK*.

Przy definiowaniu ograniczenia nałożonego na klucz obcy w istniejącej tabeli powoduje ono skutki wynikające z zasad kaskadowego usuwania i aktualizowania danych. I tak po zdefiniowaniu ograniczenia nałożonego na klucz obcy w tabeli *Ksiazki* zmiana lub usunięcie klucza podstawowego z tabeli *Autor* powoduje automatyczne zmiany w tabeli *Ksiazki*.

Jeżeli w połączonych tabelach występuje konflikt danych (np. w tabeli *Ksiazki* występuje `id_autora`, którego nie ma w tabeli *Autor*), to w wyniku wykonania klauzuli `REFERENCES` otrzymamy komunikat o błędzie (rysunek 3.6).

The **ALTER TABLE** statement conflicted with the FOREIGN KEY constraint "Ksiazki_FK". The conflict occurred in database "ksiegarnia", table "dbo.Autor", column 'id_autora'.

Rysunek 3.6. Wynik kontrolowania więzów integralności w połączonych tabelach

W celu rozwiązania problemu można wybrać jedno z kilku zaproponowanych rozwiązań:

PIERWSZY SPOSÓB

Nie naprawiamy błędów w istniejących danych, dlatego tworzymy klucz obcy, który będzie działał tylko dla nowo wprowadzanych danych:

```
ALTER TABLE Ksiazki WITH NOCHECK
ADD CONSTRAINT Ksiazki_FK FOREIGN KEY (id_autora)
REFERENCES Autor (id_autora);
```

W tej wersji zapytania została dodana klauzula `WITH NOCHECK`, która informuje bazę danych, że nie należy weryfikować wcześniej wprowadzonych danych.

DRUGI SPOSÓB

Usuujemy wiersz, który zawiera książkę ze źle podanym *id_autora*, lub ustawiamy wartość *id_autora* na NULL, czyli na wartość nieokreśloną. W ten sposób zachowamy informacje o książce i zweryfikujemy poprawność pozostałych danych.

TRZECI SPOSÓB

Usuujemy zdefiniowany w sposobie pierwszym klucz obcy:

```
ALTER TABLE Ksiazki DROP CONSTRAINT Ksiazki_FK;
```

Następnie ustawiamy wartość NULL dla danych, które nie spełniają zależności klucza obcego:

```
UPDATE Ksiazki SET id_autora = NULL
WHERE id_autora NOT IN
(SELECT id_autora FROM Autor);
```

Teraz możemy założyć klucz obcy według standardowych ustawień:

```
ALTER TABLE Ksiazki
ADD CONSTRAINT Ksiazki_FK FOREIGN KEY (id_autora) REFERENCES Autor
(id_autora);
```

Relacja, która została utworzona między tabelami *Ksiazki* i *Autor*, jest typu jeden do wielu. W kolumnie *id_autora* tabeli *Autor* zawsze występują wartości unikatowe (wymóg klucza podstawowego). W tabeli *Ksiazki* możemy mieć wiele książek przypisanych do jednego autora (kolumna *id_autora*).

Jeżeli spróbujemy teraz usunąć autora z tabeli *Autor* poleceniem:

```
DELETE FROM Autor WHERE id_autora = 2;
```

otrzymamy komunikat, że jest to niemożliwe.

Zadanie 3.6

Zdefiniuj klauzulę dotyczącą ograniczeń klucza obcego w tabeli *Rejestracja_zamowienia* i w tabeli *Zamowienia*.

Zadanie 3.7

Usuń z tabeli *Klient* dane wybranego klienta, który składał zamówienia w bazie. Zastanów się, czy jest to możliwe. Co należy zrobić, gdybyś mimo wszystko chciał usunąć dane klienta z bazy?

3.8.2. Kaskadowe usuwanie i aktualizowanie danych

Efekty modyfikacji klucza podstawowego w powiązanych tabelach są odzwierciedlane przez zdefiniowanie kaskadowego usuwania lub aktualizowania danych.

- Aktualizowanie klucza podstawowego wymaga aktualizacji wartości w powiązonym z nim kluczu obcym. W relacyjnej bazie danych klucze podstawowe nie powinny być w ogóle modyfikowane, więc kaskadowe aktualizowanie definiujemy tylko w wyjątkowych przypadkach.
- Usunięcie wiersza w tabeli nadrzędnej lub wartości klucza podstawowego wymaga usunięcia lub zaktualizowania wartości w powiązonym z nim kluczu obcym. Kaskadowe usuwanie może doprowadzić do usunięcia wielu wierszy z różnych tabel, a w konsekwencji do usunięcia istotnych danych, zatem należy je definiować wyłącznie dla tabel pomocniczych opisujących związki „wiele do wielu”. Aktualizowanie wartości klucza obcego jest bezpieczniejsze. Jeżeli kolumna klucza obcego zezwala na wpisywanie wartości NULL, należy wartości usuniętego klucza podstawowego zastąpić tą wartością. Jeżeli jest to niemożliwe, trzeba zastąpić wartości NULL specjalnie zdefiniowaną wartością domyślną.

Kaskadowe usuwanie i aktualizowanie danych definiuje się w klauzulach `ON UPDATE` i `ON DELETE` z wartościami:

- `NO ACTION` — dane w powiązanych tabelach nie będą automatycznie modyfikowane. Jest to domyślna wartość.
- `CASCADE` — modyfikacja ma zostać automatycznie powtórzona we wszystkich powiązanych tabelach.
- `SET NULL` — zmodyfikowane wartości klucza podstawowego mają zostać zastąpione wartością NULL w powiązanych kolumnach klucza obcego.
- `SET DEFAULT` — zmodyfikowane wartości klucza podstawowego mają zostać zastąpione w powiązanych kolumnach klucza obcego wartością domyślną.

Przykład 3.69

```
ALTER TABLE Ksiazki
DROP CONSTRAINT Ksiazki_FK;

ALTER TABLE Ksiazki
ADD CONSTRAINT Ksiazki_FK FOREIGN KEY (id_autora)
REFERENCES Autor (id_autora)
ON UPDATE CASCADE
ON DELETE SET NULL;
```

Polecenie `DROP CONSTRAINT` usuwa wcześniej zdefiniowane ograniczenie. Następnie zostaje zdefiniowane nowe ograniczenie klucza obcego, w którym polecenie `ON UPDATE CASCADE` określa, że aktualizacja wykonana w tabeli nadrzędnej ma zostać powtórzona w kolumnie klucza obcego tabeli podrzędnej, a polecenie `ON DELETE SET NULL` oznacza, że gdy będzie usuwany wiersz w tabeli nadrzędnej, w kolumnie klucza obcego tabeli podrzędnej zostanie wstawiona wartość NULL.



3.9. Łączenie wyników zapytań

W języku SQL istnieją mechanizmy, które pozwalają łączyć wyniki kilku zapytań. Do połączenia zapytań można użyć jednej z trzech instrukcji: UNION, INTERSECT, EXCEPT. Serwer MySQL nie obsługuje instrukcji INTERSECT i EXCEPT standardu SQL. Aby otrzymać wynik zapytania, należy użyć innych konstrukcji języka SQL (najczęściej są to podzapytania lub łączenie tabel).

3.9.1. Instrukcja UNION

Instrukcja UNION łączy wyniki zapytań i ma postać:

```
zapytanie1 UNION zapytanie2
```

Warunkiem wykonania instrukcji jest, aby łączone zapytania miały taką samą liczbę kolumn oraz aby typy kolumn były takie same.

Przykład 3.70

```
SELECT nazwisko, imie
FROM Klient
WHERE miejscowosc='Warszawa'

UNION

SELECT nazwisko, imie
FROM Klient
WHERE miejscowosc='Gdańsk';
```

Pierwsze zapytanie zwróci listę klientów z Warszawy, drugie — z Gdańska. Wynikiem połączenia zapytań będzie lista klientów z Warszawy i z Gdańska.

Instrukcja UNION domyślnie powoduje usunięcie powtarzających się wierszy. W celu zachowania w wyniku wszystkich wierszy należy użyć instrukcji UNION ALL. Takie użycie instrukcji spowoduje dużo szybsze zwrócenie wyniku, ponieważ nie wymaga wyszukiwania i usuwania powtarzających się wierszy.

Instrukcja UNION (lub UNION ALL) może zastępować w zapytaniu operator OR.

Przykład 3.71

```
SELECT nazwisko, imie
FROM Klient
WHERE miejscowosc = 'Warszawa' OR miejscowosc = 'Gdańsk';
```

3.9.2. Instrukcja INTERSECT

Instrukcja `INTERSECT` zwraca część wspólną wyników dwóch zapytań i ma postać:

```
zapytanie1 INTERSECT zapytanie2
```

Podobnie jak poprzednio, obydwa zapytania powinny zwracać taką samą liczbę kolumn o takich samych typach.

Przykład 3.72

```
SELECT nazwisko, imie
FROM Klient
WHERE telefon IS NULL
INTERSECT
SELECT nazwisko, imie
FROM Klient
WHERE adres_e_mail IS NULL;
```

Instrukcja `INTERSECT` zwróci dane klientów, którzy nie podali numeru telefonu ani adresu e-mail.

3.9.3. Instrukcja EXCEPT

Instrukcja `EXCEPT` ma postać:

```
zapytanie1 EXCEPT zapytanie2
```

Zwraca te wiersze, które wystąpiły w wyniku pierwszego zapytania, ale nie było ich w wyniku drugiego zapytania. Zmiana kolejności zapytań w tej instrukcji spowoduje zmianę wyniku.

Przykład 3.73

```
SELECT nazwisko, imie
FROM Klient
WHERE miejscowosc='Warszawa'
EXCEPT
SELECT nazwisko, imie
FROM Klient
WHERE telefon IS NOT NULL;
```

Instrukcja `EXCEPT` zwróci dane klientów, którzy mieszkają w Warszawie i nie podali numeru telefonu komórkowego.

Ćwiczenie 3.16

Wyświetl tytuły książek, które zostały zakupione zarówno przez klienta o nazwisku *Kowalski*, jak i przez klienta o nazwisku *Wiśniewski*.

Rozwiązanie

```
SELECT tytuł
FROM Klient INNER JOIN Zamowienia
ON Klient.id_klienta=Zamowienia.id_klienta
INNER JOIN Rejestracja_zamowienia
ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia
INNER JOIN Ksiazki
ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki
WHERE nazwisko ='Kowalski'

INTERSECT

SELECT tytuł
FROM Klient INNER JOIN Zamowienia
ON Klient.id_klienta=Zamowienia.id_klienta
INNER JOIN Rejestracja_zamowienia
ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia
INNER JOIN Ksiazki
ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki
WHERE nazwisko ='Wiśniewski'
```

Zadanie 3.8

Wyświetl nazwiska klientów, którzy kupili książki 20 lipca 2019 roku, przy czym nie były to książki wydane w języku angielskim.



3.10. Podzapytania

Zapytanie definiowane przy użyciu instrukcji `SELECT` może zostać umieszczone wewnątrz innej instrukcji `SELECT`. Tak zagnieżdżone zapytanie nazywamy **podzapytaniem**. Możemy używać również nazw **zapytanie zagnieżdżone** lub **zapytanie wewnętrzne**. Ponieważ serwery bazodanowe najpierw wykonują zapytania wewnętrzne, wyniki tych właśnie zapytań mogą być używane jako warunki logiczne kolejnych zapytań. Zagnieżdżanie zapytań można stosować w instrukcjach `SELECT`, `INSERT`, `UPDATE` i `DELETE` w klauzuli `WHERE` lub `FROM`. Każde podzapytanie klauzuli `WHERE` lub `FROM` jest umieszczane w nawiasach okrągłych i może zawierać dowolną liczbę podzapytań.

Najczęściej wynikiem zapytania typu `SELECT` jest tabela składająca się z kolumn i wierszy, więc można na niej wykonywać kolejne zapytania typu `SELECT`.

Ogólna postać zapytania zawierającego podzapytanie w klauzuli `WHERE` może wyglądać tak:

```
SELECT kolumna1, kolumna2, kolumna3, ...
FROM Tabela1
WHERE kolumna1 =
  (SELECT kolumnaA1
   FROM TabelaA1
   WHERE warunek_podzapytania AS nazwa)
```

3.10.1. Podzapytania klauzuli `WHERE`

Zapytania wewnętrzne używane w klauzuli `WHERE` mogą zwracać pojedynczą wartość lub listę wartości.

Przykład 3.74

Na podstawie numeru faktury chcemy odczytać nazwisko i imię klienta, dla którego faktura została wystawiona.

```
SELECT Klient.nazwisko, Klient.imie
FROM Klient
WHERE Klient.id_klienta =
  (SELECT Zamowienia.id_klienta
   FROM Zamowienia
   WHERE Zamowienia.Id_faktury=2);
```

Wynik zapytania wewnętrznego jest pojedynczą wartością. Zwraca *id_klienta*, dla którego została wystawiona faktura o numerze 2. Zapytanie zewnętrzne zwraca imię i nazwisko klienta o wskazanym *id_klienta*.

Jeżeli takie same nazwy kolumn powtarzają się w kilku tabelach, w podzapytaniach powinny być one poprzedzone nazwami tabel — pozwoli to uniknąć błędów serwera.

Jeżeli zapytanie wewnętrzne zwraca tylko jedną wartość, to w zapytaniu zewnętrznym w klauzuli `WHERE` można użyć operatorów: `=`, `<`, `>`, `<=`, `>=`, `<>`.

Gdyby zwróciło listę wartości, użycie tych operatorów byłoby nielogiczne. Serwer bazodanowy nie potrafiłby jednoznacznie określić ich znaczenia. Jeżeli zaistnieje taka sytuacja, to do porównania należy użyć operatora `IN`. Podzapytanie zwróci prawdę, jeżeli chociaż jedna wartość z listy spełnia zdefiniowany warunek.

Przykład 3.75

Na podstawie daty wysłania chcemy odczytać tytuły i ceny książek wysłanych do klientów 1 czerwca 2019 roku.

```
SELECT Ksiazki.tytul, Ksiazki.cena
FROM Ksiazki
WHERE Ksiazki.id_ksiazki IN
(SELECT Rejestracja_zamowienia.id_ksiazki
FROM Rejestracja_zamowienia INNER JOIN Zamowienia
ON Rejestracja_zamowienia.id_zamowienia=Zamowienia.id_zamowienia
WHERE data_wyslania='2019-06-01');
```

Wynik zapytania wewnętrznego jest listą wartości. Zwraca *id_ksiazki* książek zakupionych 1 czerwca 2019 roku. W związku z tym dla porównania wartości w klauzuli WHERE został zastosowany operator IN. Zapytanie zewnętrzne zwróci tytuły i ceny książek zakupionych tego dnia.

Zapytanie wewnętrzne może wywoływać funkcje grupujące.

Przykład 3.76

Chcemy odczytać tytuły książek, których cena dwukrotnie przekracza średnią cenę książek zapisanych w bazie danych.

```
SELECT Ksiazki.tytul
FROM Ksiazki
WHERE Ksiazki.cena>
(SELECT AVG(Ksiazki.cena)*2
FROM Ksiazki);
```

Ćwiczenie 3.17

Wyświetl tytuły książek, które zostały zakupione zarówno przez klienta o nazwisku *Nowak*, jak i przez klienta o nazwisku *Polak*.

Rozwiązanie

Podobne ćwiczenie zostało rozwiązane z zastosowaniem instrukcji INTERSECT. Niestety serwer MySQL jej nie obsługuje. Proponowane rozwiązanie wykorzystuje mechanizm zagnieżdżania zapytań i może zostać zastosowane zarówno na serwerze MySQL, jak i na serwerze SQL Server.

```
SELECT tytul
FROM Klient INNER JOIN Zamowienia
ON Klient.id_klienta=Zamowienia.id_klienta
```

```
INNER JOIN Rejestracja_zamowienia
ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia
INNER JOIN Ksiazki
ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki
WHERE nazwisko ='Nowak' And tytul IN
(
SELECT tytul
FROM Klient INNER JOIN Zamowienia
ON Klient.id_klienta=Zamowienia.id_klienta
INNER JOIN Rejestracja_zamowienia
ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia
INNER JOIN Ksiazki
ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki
WHERE nazwisko ='Polak'
)
```

3.10.2. Operatory podzapytań

Podzapytania zwracające w zapytaniu wewnętrznym listę wartości wymagają zastosowania w klauzuli `WHERE` specjalnych operatorów. Są to operatory: `IN`, `EXISTS`, `ANY`, `SOME`, `ALL`.

`IN` sprawdza, czy chociaż jedna wartość z listy spełnia zdefiniowany warunek. Lista wartości musi składać się z jednej kolumny.

`EXISTS` służy do sprawdzenia, czy w wyniku wykonania zapytania wewnętrznego zostały zwrócone jakiekolwiek wartości.

`ANY` i `SOME` są synonimami i działają dokładnie tak samo: sprawdzają wartość wybranego wiersza wyniku podzapytania.

`ALL` sprawdza wartość wszystkich wierszy wyniku podzapytania.

Operator EXISTS

Operator `EXISTS` zwraca tylko prawdę lub fałsz. Jeżeli w wyniku wykonania zapytania wewnętrznego zostały zwrócone jakiekolwiek wartości, operator `EXISTS` zwraca prawdę (*TRUE*), a jeżeli nie zostały zwrócone żadne wartości, zwraca fałsz (*FALSE*). Operator ten sprawdza tylko, czy wynik podzapytania jest pusty czy nie, i nie ma znaczenia, jakiego typu elementy zostały zwrócone.

Przykład 3.77

```
SELECT Klient.nazwisko, Klient.imie
FROM Klient
WHERE EXISTS
(SELECT *
FROM Zamowienia
WHERE Klient.id_klienta =Zamowienia.id_klienta);
```

W wyniku wykonania zapytania zostanie zwrócona lista klientów, którzy złożyli zamówienia na książki.

Przykład 3.78

```
SELECT nazwisko, imie
FROM Klient AS K1
WHERE EXISTS
(SELECT *
FROM Klient AS K2
WHERE K1.nazwisko=K2.nazwisko AND K1.imie=K2.imie AND K1.PESEL=K2.PESEL);
```

W podanym przykładzie operator EXISTS został użyty do wyszukania powtarzających się danych klientów. Dane te mogły przez pomyłkę zostać kilkakrotnie wpisane do bazy.

W celu stwierdzenia braku danych operator EXISTS może zostać zastosowany razem z operatorem NOT. W przykładzie poniżej dodanie operatora NOT spowoduje wyświetlenie listy klientów, którzy nie złożyli żadnego zamówienia na książki.

Przykład 3.79

```
SELECT Klient.nazwisko, Klient.imie
FROM Klient
WHERE NOT EXISTS
(SELECT *
FROM Zamowienia
WHERE Klient.id_klienta =Zamowienia.id_klienta);
```

Operator ANY lub SOME

Operator ANY zwraca prawdę, jeśli którakolwiek ze zwróconych wartości wyniku zapytania wewnętrznego spełnia poprzedzający je warunek.

Przykład 3.80

```
SELECT Klient.nazwisko, Klient.imie
FROM Klient INNER JOIN Zamowienia
ON Klient.id_klienta=Zamowienia.id_klienta
WHERE Zamowienia.data_zlozenia_zamowienia= ANY
(SELECT Faktura.Data_wystawienia_faktury
FROM Faktura);
```

Podzapytanie zwróci nazwiska klientów, dla których faktury zostały wystawione w dniu złożenia zamówienia.

Operator ALL

Operator ALL zwraca prawdę, jeżeli wszystkie zwrócone wartości wyniku zapytania wewnętrznego spełniają poprzedzający je warunek.

Przykład 3.81

```
SELECT Ksiazki.tytul
FROM Ksiazki
WHERE Ksiazki.cena< ALL
(SELECT Zamowienia.koszt_wysylki
FROM Zamowienia INNER JOIN Rejestracja_zamowienia
ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia);
```

Podzapytanie zwróci tytuły książek, które są tańsze od najniższych kosztów wysyłki. Najprawdopodobniej niewielu klientom opłaca się zamawiać książki, których koszt wysyłki przekracza cenę książki.

3.10.3. Podzapytania klauzuli FROM

Jeżeli wynikiem zapytania typu SELECT jest tabela zawierająca określone w zapytaniu kolumny, możliwe jest wykonanie na niej kolejnego zapytania typu SELECT. W ten sposób tworzone są podzapytania w klauzuli FROM. Tabeli zwróconej przez podzapytanie użyte w klauzuli FROM należy przypisać nazwę.

Tabele pochodne

Jeżeli zapytanie wewnętrzne zwraca wartości w postaci danych tabelarycznych, to tworzone są tabele pochodne. Są one dostępne tylko w zapytaniu, w którym zostały utworzone.

Przykład 3.82

```
SELECT Lista.tytul, Lista.cena
FROM
(SELECT tytul, cena, miejsce_wydania, rok_wydania
FROM Ksiazki
WHERE rok_wydania<2019 AND cena>70) AS Lista
WHERE Lista.miejsce_wydania='Warszawa';
```

W podanym przykładzie w wyniku wykonania zapytania wewnętrznego otrzymaliśmy wybrane wiersze i kolumny z tabeli *Ksiazki*. Tak powstałej tabeli pochodnej została nadana nazwa (*alias*) *Lista*. W zapytaniu zewnętrznym zawartość tej tabeli jest odczytywana w podobny sposób jak zawartość zwykłej tabeli.

Tabele pochodne są stosowane w celu uproszczenia zapytań i poprawienia ich czytelności.

3.10.4. Podzapytania w instrukcjach modyfikujących dane

Podzapytania mogą być definiowane w instrukcjach modyfikujących dane, takich jak: INSERT, UPDATE, DELETE.

Przykład 3.83

W bazie danych zostanie utworzona nowa tabela *Archiwum*, która będzie zawierała imiona i nazwiska klientów oraz informacje o złożonych zamówieniach.

```
CREATE TABLE Archiwum
(
    nazwisko varchar (60),
    imie varchar (40),
    data_zlozenia_zamowienia date,
    Liczba_zamowien int
);
```

Powstałą tabelę wypełnimy danymi z tabel *Klient* i *Zamowienia*. Możemy to zrealizować za pomocą instrukcji INSERT z odpowiednio zdefiniowanym podzapytaniem.

Przykład 3.84

```
INSERT INTO Archiwum (nazwisko, imie, data_zlozenia_zamowienia)
(SELECT nazwisko, imie, data_zlozenia_zamowienia
FROM Klient, Zamowienia
WHERE Klient.id_klienta=Zamowienia.id_klienta);
```

Przykład 3.85

Skoro w utworzonej tabeli nazwiska klientów powtarzają się tyle razy, ile razy zamawiali oni książki, można tak zmodyfikować polecenie, aby w tabeli zostały umieszczone nazwiska i imiona klientów z liczbą zamówień złożonych przez każdego z nich. W podzapytaniu trzeba użyć grupowania i funkcji agregujących.

```
INSERT INTO Archiwum (nazwisko, imie, Liczba_zamowien)
(SELECT nazwisko, imie, COUNT(data_zlozenia_zamowienia) AS 'Liczba zamówień'
FROM Klient INNER JOIN Zamowienia
ON Klient.id_klienta=Zamowienia.id_klienta
GROUP BY Klient.nazwisko, Klient.imie);
```

3.10.5. Podzapytania skorelowane

Używane do tej pory podzapytania proste były wykonywane tylko jeden raz. W podzapytaniach skorelowanych następuje odwołanie w podzapytaniu do wyniku zapytania zewnętrznego. W związku z tym podzapytanie wykonywane jest osobno dla każdego wiersza zwróconego przez zapytanie zewnętrzne.

Przykład 3.86

Chcemy sprawdzić, które książki mają cenę wyższą niż średnia cena książek wydanych w tym samym roku.

```
SELECT tytuł, cena, rok_wydania
FROM Ksiazki
WHERE cena >
  (SELECT AVG(cena)
   FROM Ksiazki AS Ksiegi
   WHERE Ksiazki.rok_wydania=Ksiegi.rok_wydania)
ORDER BY cena;
```

Dla każdej książki rozpatrywanej w zapytaniu zewnętrznym zostanie wyliczona w zapytaniu wewnętrznym średnia cena książek wydanych w tym samym roku co analizowana książka. W zapytaniu wewnętrznym tabela *Ksiazki* otrzymała nazwę *Ksiegi*, aby warunek w klauzuli WHERE zapytania wewnętrznego miał sens.

3.11. Widoki (perspektywy)

Widoki (perspektywy) w języku SQL to wirtualne tabele tworzone na podstawie zapytań. Składają się z kolumn i wierszy pobranych z prawdziwych tabel. Pokazywane w widoku dane są zawsze aktualne, ponieważ widoki są tworzone w momencie wykonania zapytania.

Widoki nie przechowują zapisanych w tabelach danych. W bazie danych jest zapamiętywana tylko definicja widoku i związane z nią metadane. Natomiast widoki są tworzone za każdym razem, gdy do widoku zostanie skierowane zapytanie. Umożliwiają wykonywanie operacji podobnych do tych wykonywanych na tabelach. Zapewniają bezpieczeństwo danych przez ograniczenie dostępu do danych zapisanych w tabelach. Na podstawie istniejących widoków można tworzyć następne.

3.11.1. Tworzenie i usuwanie widoku

W celu zapisania zapytania w postaci widoku należy poprzedzić je poleceniem `CREATE VIEW`.

Przykład 3.87

```
CREATE VIEW Zbior_ksiazek AS
SELECT tytul, nazwisko, imie, cena
FROM Autor INNER JOIN Ksiazki
ON Autor.id_autora= Ksiazki.id_autora;
```

Do usuwania zdefiniowanego widoku używamy polecenia `DROP VIEW`.

Przykład 3.88

```
DROP VIEW Zbior_ksiazek;
```

W wyniku wykonania kodu z bazy danych zostanie usunięty widok *Zbior_ksiazek*.

3.11.2. Modyfikowanie widoku

Zdefiniowany wcześniej widok można zmodyfikować. Modyfikując widok, należy jeszcze raz zapisać tworzącą go instrukcję `SELECT`. Różnica między tworzeniem widoku od nowa a modyfikowaniem istniejącego polega na tym, że widok modyfikowany zachowuje nadane mu wcześniej uprawnienia. Widok modyfikujemy instrukcją `ALTER VIEW`.

Przykład 3.89

```
ALTER VIEW Zbior_ksiazek AS
SELECT tytul, nazwisko, imie, cena
FROM Ksiazki INNER JOIN Autor
ON Ksiazki.id_autora = Autor.id_autora
WHERE cena > 50;
```

UWAGA

W widokach nie można stosować klauzuli `ORDER BY`. Jedyny wyjątek to użycie klauzuli `ORDER BY` do określenia wierszy, które są zwracane przez klauzulę `TOP`.

Widoki są szczególnie przydatne, gdy w zapytaniach często pobieramy dane z wielu połączonych ze sobą tabel. Zamiast każdorazowego definiowania tych połączeń, można do pracy z bazą wykorzystać zdefiniowane widoki. Dane są odczytywane poprzez widoki tak samo jak bezpośrednio z tabel.

Przykład 3.90

```
CREATE VIEW Sprzedaz AS

SELECT Klient.nazwisko AS K_Nazw, Klient.imie AS K_Im, Ksiazki.tytul,
Autor.nazwisko AS A_Nazw, Autor.imie AS A_Im, Ksiazki.cena

FROM Klient INNER JOIN Zamowienia

ON Klient.id_klienta=Zamowienia.id_klienta

INNER JOIN Rejestracja_zamowienia

ON Zamowienia.id_zamowienia=Rejestracja_zamowienia.id_zamowienia

INNER JOIN Ksiazki

ON Rejestracja_zamowienia.id_ksiazki=Ksiazki.id_ksiazki

INNER JOIN Autor

ON Ksiazki.id_autora=Autor.id_autora

WHERE Zamowienia.data_zlozenia_zamowienia = '2019-06-01';
```

Zaleca się, aby użytkownicy baz danych ze względu na bezpieczeństwo danych nie mieli bezpośredniego dostępu do tabel. Dlatego dobrym rozwiązaniem jest definiowanie widoków i udostępnianie danych poprzez utworzone widoki.

3.12. Indeksy

Indeksy są definiowane w celu zwiększenia prędkości wykonywania operacji pobierania danych z tabeli. Są to uporządkowane struktury zawierające dane z wybranych kolumn tabeli. Zaletą stosowania indeksów jest ograniczenie ilości danych odczytywanych z bazy, przyspieszenie wyszukiwania informacji oraz sortowanie danych. Ich wadą jest to, że zajmują na dysku dodatkowe miejsce, muszą być na bieżąco aktualizowane, a każde wstawienie, usunięcie lub zaktualizowanie danych w tabeli wiąże się z aktualizacją wszystkich zdefiniowanych dla niej indeksów.

Indeksy można budować na etapie tworzenia tabel lub definiować je w już istniejącej tabeli. Instrukcja tworzenia indeksu ma postać:

```
CREATE INDEX nazwa_indeksu ON nazwa_tabeli (nazwa_kolumny)
```

Przykład 3.91

Tworzenie indeksu w istniejącej tabeli:

```
CREATE INDEX Indeks_tytul ON Ksiazki (tytul);
```

Tworzenie unikatowego indeksu ma postać:

```
CREATE UNIQUE INDEX nazwa_indeksu
ON nazwa_tabeli (nazwa_kolumny)
```

W tak zaindeksowanej kolumnie powtarzające się wartości są niedozwolone.

UWAGA

Instrukcja tworzenia indeksów w innych serwerach bazodanowych może różnić się od podanej.

Aktualizacja tabeli z indeksami zajmuje więcej czasu niż aktualizacja tabeli bez indeksów. Dlatego należy tworzyć indeksy tylko dla kolumn, które będą często przeszukiwane. Przed utworzeniem indeksów warto przeanalizować zapytania i podjąć decyzję, gdzie i jakie indeksy utworzyć. Na pewno indeksy powinny być utworzone dla pól kluczy podstawowych i kluczy obcych.

Do usuwania indeksu służy instrukcja `DROP INDEX`:

```
DROP INDEX nazwa_indeksu ON nazwa_tabeli
```

3.13. Transakcje

W bazach danych transakcja to zbiór wykonywanych operacji, które stanowią całość. Muszą zostać wykonane wszystkie operacje wchodzące w skład transakcji lub nie zostanie wykonana żadna z nich. Przykładem transakcji jest wykonywanie przelewu z jednego konta na drugie. Operacja przelewu musi zostać wykonana w całości, a jeżeli nie jest to możliwe, należy powrócić do stanu sprzed rozpoczęcia wykonywania operacji przelewu.

Transakcja składa się z trzech etapów:

- rozpoczęcia,
- wykonania,
- zakończenia.

3.13.1. Właściwości transakcji

Transakcja ma przypisane jej własności:

- *atomic* — transakcja jest niepodzielna, czyli jest wykonywana w całości lub w całości jest odwoływana.
- *consistent* — transakcja nie zmienia spójności (integralności) bazy danych, czyli wykonanie transakcji nie doprowadzi do utraty spójności danych. Jeżeli baza danych była spójna przed wykonaniem transakcji, to jest spójna również po jej zakończeniu.

- *isolated* — transakcja musi być izolowana, czyli nie może istnieć konflikt z innymi transakcjami wykonywanymi w tym samym czasie na tym samym zbiorze danych.
- *durable* — transakcja jest trwała, czyli działania wykonane w transakcji są trwałe niezależnie od tego, co się będzie działo po jej zakończeniu.

W trakcie wykonywania transakcji stosowany jest **mechanizm blokowania**, który gwarantuje spójność bazy danych. Oznacza to, że podczas wykonywania transakcji dane, na których jest ona wykonywana, nie ulegną zmianie lub usunięciu.

Pierwsze litery nazw własności transakcji tworzą akronim **ACID**, określający reguły, które muszą być spełnione przez serwery bazodanowe, aby transakcje mogły być wykonywane.

W SQL Server transakcja może być wykonywana na trzy sposoby:

- *Explicit* — jawnie. Rozpoczęcie transakcji jest realizowane za pomocą polecenia `BEGIN TRANSACTION`.
- *Autocommit* — automatycznie. Operacje wykonywane na serwerze są standardowo traktowane jako transakcje, w związku z czym nie ma potrzeby ich rozpoczynania poleceniem `BEGIN TRANSACTION`. Po poprawnym wykonaniu każde z poleceń jest automatycznie zatwierdzane.
- *Implicit* — niejawnie. Transakcje są wywoływane przez programy użytkowe działające na bazie danych.

Transakcje Explicit

Transakcje *Explicit* są wykonywane, jeżeli zadeklarujemy chęć wykonania zapytania lub bloku zapytań w ramach transakcji za pomocą polecenia `BEGIN TRANSACTION`.

Zatwierdzenie zmian i zakończenie transakcji deklarujemy za pomocą instrukcji `COMMIT`. Polecenie to zdejmuje blokady z tabel założone na czas trwania transakcji. Natomiast użycie instrukcji `ROLLBACK` odwołuje transakcję i odrzuca wszystkie zmiany dokonane podczas trwania transakcji. Polecenie to również zdejmuje blokady z tabel założone na czas trwania transakcji.

Przykład 3.92

Załóżmy, że od nowego roku ceny książek wydanych po roku 2017 wzrosły w księgarni internetowej o 5%, a ceny książek wydanych przed rokiem 2015 zmalały o 2%. Wykonywanie operacji zmiany cen książek zostanie zabezpieczone transakcją.

```
BEGIN TRANSACTION

UPDATE Ksiazki

SET cena = cena + cena * 0.05

WHERE rok_wydania > 2017

IF @@ERROR <> 0
```

```

BEGIN

    RAISERROR ('Błąd, Operacja zakończona niepowodzeniem !', 16, -1)

    ROLLBACK

END

UPDATE Ksiazki

SET cena = cena - cena *0.02

WHERE rok_wydania < 2015

IF @@ERROR <> 0

BEGIN

    RAISERROR ('Błąd, Operacja zakończona niepowodzeniem !', 16, -1)

    ROLLBACK

END

COMMIT;

```

Pierwsza instrukcja (BEGIN TRANSACTION) uruchomiła transakcję, następna (UPDATE) przeprowadziła operację na danych. Kolejny blok to instrukcje sprawdzające, czy transakcja się powiodła. Ponieważ zmienna systemowa @@ERROR standardowo zwraca informację z numerem ostatniego błędu, może zostać użyta do sprawdzenia, czy w trakcie operacji zmiany ceny książek wystąpił błąd. Jeśli błąd nie wystąpił, zmienna ma wartość zero. Jeśli błąd wystąpił, zostanie wyświetlony komunikat (instrukcja RAISERROR) i transakcja zostanie odwołana (ROLLBACK). Jeżeli operacja przebiegła pomyślnie, przechodzimy do kolejnej operacji zmiany ceny książek.

Pojedyncza instrukcja UPDATE (dotyczy to również instrukcji DELETE i INSERT) nie musiałaby być poprzedzona uruchomieniem transakcji, ponieważ dla niej transakcja zostanie uruchomiona automatycznie. W podanym wyżej przykładzie modyfikowane dane są ze sobą wzajemnie powiązane, czyli niepowodzenie przy modyfikacji jednego zbioru danych powinno spowodować anulowanie modyfikacji drugiego zbioru danych — dlatego należy użyć transakcji.

Transakcje Autocommit

Transakcje *Autocommit*, podobnie jak transakcje *Implicit*, inicjuje SQL Server oraz MySQL za każdym razem, gdy zostanie wydane polecenie modyfikowania danych. Ale po wykonaniu polecenia serwer automatycznie zatwierdza lub odrzuca transakcję.

Dla serwerów tryb automatyczny jest domyślnym trybem zarządzania transakcjami. Należy pamiętać, że jeżeli pracujemy w trybie *Autocommit*, każde wydane polecenie jest osobną transakcją.

Przykład 3.93

Wykorzystując funkcję systemową @@TRANCOUNT, sprawdzimy działanie trybu automatycznego transakcji. Funkcja ta zwraca liczbę otwartych w danym momencie transakcji.

```
SELECT @@TRANCOUNT;

UPDATE Ksiazki SET cena=20

WHERE rok_wydania<2018;

SELECT @@TRANCOUNT;
```

W wyniku wykonania kodu zobaczymy, że przed rozpoczęciem wykonywania instrukcji UPDATE i po jej zakończeniu żadne transakcje nie były otwarte.

Transakcje Implicit

Transakcje *Implicit* inicjuje SQL Server, wydając niejawnie polecenie BEGIN TRANSACTIONS. Transakcje niejawne są przeprowadzane, gdy wykonywana jest jedna z następujących komend: ALTER TABLE, CREATE, DELETE, DROP, FETCH, GRANT, INSERT, OPEN, REVOKE, SELECT, TRUNCATE, UPDATE. Naszym zadaniem jest zakończenie transakcji i jej zatwierdzenie lub wycofanie. Aby serwer pracował w trybie transakcji niejawnych, należy je włączyć za pomocą polecenia:

```
SET IMPLICIT_TRANSACTIONS ON
```

Po wykonaniu tego polecenia do końca sesji będą realizowane transakcje niejawne. W celu wcześniejszego zakończenia pracy w trybie transakcji niejawnych trzeba wpisać polecenie:

```
SET IMPLICIT_TRANSACTIONS OFF
```

Ten tryb pracy może stwarzać problemy z bazą danych, gdy zapomnimy zatwierdzić lub wycofać transakcję. Pozostanie ona otwarta i będzie blokowała dostęp do danych. Jego zaletą jest możliwość wycofywania przypadkowych lub błędnych modyfikacji danych.

Przykład 3.94

Wykorzystując polecenie SET IMPLICIT_TRANSACTIONS ON, ustawimy tryb niejawny transakcji. Następnie, stosując zmienną systemową @@TRANCOUNT, sprawdzimy działanie trybu niejawnego transakcji.

```
SET IMPLICIT_TRANSACTIONS ON;

SELECT @@TRANCOUNT;

UPDATE Ksiazki SET cena=30

WHERE rok_wydania<2017;

SELECT @@TRANCOUNT;
```

Przed rozpoczęciem wykonywania instrukcji `UPDATE` nie było otwartych żadnych transakcji. Natomiast po jej zakończeniu pozostała rozpoczęta jedna transakcja, ponieważ nie została ona automatycznie zamknięta. Użytkownik powinien zamknąć transakcję, zatwierdzając zmiany lub je wycofując. Aby zamknąć transakcję, należy wykonać polecenie:

```
COMMIT;

SET IMPLICIT_TRANSACTIONS OFF;
```

Dopóki transakcja nie zostanie zamknięta, dopóty dostęp do tabeli *Książki* nie będzie możliwy.

Zagnieżdżanie transakcji

W ramach już rozpoczętej transakcji można umieścić kolejną instrukcję `BEGIN TRANSACTION`. Wynikiem takiego działania jest zwiększenie liczby otwartych transakcji, a nie rozpoczęcie nowej transakcji.

Mechanizm zagnieżdżonych transakcji wygląda następująco: wykonanie kolejnej instrukcji `BEGIN TRANSACTION` zwiększa o 1 liczbę otwartych transakcji, wykonanie instrukcji `COMMIT` zmniejsza o 1 liczbę otwartych transakcji, natomiast wykonanie instrukcji `ROLLBACK` zamyka transakcje i ustawia liczbę otwartych transakcji na 0.

Przykład 3.95

```
BEGIN TRANSACTION;

SELECT @@TRANCOUNT;

UPDATE Książki SET cena=10

WHERE rok_wydania<2015;

BEGIN TRANSACTION;

SELECT @@TRANCOUNT;

    UPDATE Książki SET cena=12

WHERE rok_wydania=2016;

BEGIN TRANSACTION;

SELECT @@TRANCOUNT;

    UPDATE Książki SET cena=18

WHERE rok_wydania=2017;

COMMIT;
```

```
SELECT @@TRANCOUNT;

ROLLBACK;

SELECT @@TRANCOUNT;
```

W podanym przykładzie po pierwszym odczycie zmiennej @@TRANCOUNT otrzymamy jedną otwartą transakcję, następnie dwie, później trzy, znowu dwie i na końcu zero otwartych transakcji.

Punkty przywracania

W większości serwerów bazodanowych można wycofać nie tylko całą transakcję, ale także jej część. W tym celu należy utworzyć punkty przywracania za pomocą instrukcji `SAVE TRANSACTION`.

Przykład 3.96

```
BEGIN TRANSACTION;

INSERT INTO Klient (nazwisko, imie, PESEL)
VALUES ('Gordon', 'Michał', '79020908765');

SAVE TRANSACTION P1;

INSERT INTO Klient (nazwisko, imie, PESEL)
VALUES ('Zieliński', 'Tomasz', '89110803456');

ROLLBACK TRANSACTION P1;
```

W podanym przykładzie instrukcja `BEGIN TRANSACTION` uruchomi transakcję. Po dodaniu do tabeli *Klient* danych jednego klienta instrukcja `SAVE TRANSACTION P1` utworzy punkt przywracania. Po dodaniu do tabeli *Klient* danych kolejnego klienta transakcja zostanie odwołana (instrukcja `ROLLBACK`), ale nie zostaną odrzucone wszystkie zmiany dokonane podczas trwania transakcji, tylko zmiany wprowadzone po zdefiniowanym punkcie przywracania. W rezultacie, mimo że transakcja została odwołana, w tabeli *Klient* zostaną zapisane dane klienta o nazwisku *Gordon*, natomiast dane klienta o nazwisku *Zieliński* zostaną anulowane.

UWAGA

Należy pamiętać o zamknięciu transakcji. Inaczej dostęp do tabeli *Klient* będzie niemożliwy.

3.14. Współbieżność

Współbieżność to zdolność do jednoczesnego realizowania wielu procesów (wątków) opartych na wspólnych danych. Polega ona na przełączaniu między procesami w bardzo krótkich odstępach czasu, co sprawia wrażenie, że procesy wykonywane są równocześnie.

Ten mechanizm znajduje szerokie zastosowanie w serwerach bazodanowych, które muszą obsługiwać wielu użytkowników jednocześnie. Aby każdy z użytkowników mógł pracować tak, jakby był jedynym użytkownikiem bazy danych, konieczne jest odizolowanie operacji (transakcji) wykonywanych przez tych użytkowników.

3.14.1. Kontrola współbieżności

Kontrola współbieżności w serwerach bazodanowych może odbywać się na podstawie:

- modelu optymistycznego,
- modelu pesymistycznego.

Model optymistyczny

Model optymistyczny opiera się na założeniu, że modyfikowanie i odczytywanie tych samych danych przez różnych użytkowników jest mało prawdopodobne, chociaż nie niemożliwe. W związku z tym można przeprowadzić transakcję bez blokowania zasobów. Jedynie w sytuacji modyfikowania danych zasoby bazy są sprawdzane w celu wykrycia konfliktów.

Model pesymistyczny

Model pesymistyczny zakłada wystąpienie konfliktów, dlatego dane są blokowane za każdym razem, gdy jakaś transakcja próbuje je odczytać lub modyfikować.

3.14.2. Blokowanie danych

Blokowanie danych stosujemy w celu zagwarantowania integralności i spójności danych w trakcie realizowania transakcji. Dzięki temu użytkownicy nie mogą odczytywać danych, które są właśnie zmieniane przez innych użytkowników, oraz nie mogą równocześnie modyfikować tych samych danych. Bez blokad dane znajdujące się w bazie bardzo szybko stałyby się niespójne, a definiowane na nich zapytania dawałyby błędne wyniki.

Serwery bazodanowe automatycznie ustawiają blokady, ale na potrzeby własnej aplikacji można modyfikować ich standardowe ustawienia.

Tryby blokad

Istnieją dwa tryby blokad. Decydują one, czy możliwe jest założenie blokady na dane, które wcześniej zostały zablokowane przez inny proces.

Blokady współdzielone S (ang. *Shared*) są zakładane domyślnie na odczytywanych obiektach tylko na czas wykonania zapytania. Jeżeli dane zostały zablokowane w trybie *S*, to możliwe jest założenie na nie blokady *S* przez inne procesy.

Blokady wyłączne X (ang. *eXclusive*) są zakładane na modyfikowanych obiektach i domyślnie utrzymywane do zakończenia całej transakcji. Użytkownicy modyfikujący dane blokują innych użytkowników.

Zakresy blokad

Blokady mogą być zakładane na różnym poziomie szczegółowości — na poziomie wierszy, kluczy, indeksów, stron, tabel, zakresów lub bazy. Dla każdej transakcji dynamicznie jest określany odpowiedni poziom, na którym należy założyć blokadę. Poziomy, na których zakładane są blokady, są ustawiane i kontrolowane przez serwer bazodanowy. To on sprawdza, czy blokady przydzielone na jednym poziomie respektują blokady ustawione na innym poziomie.

Im większe obiekty są blokowane, tym dłużej nie są one dostępne dla użytkownika (mniejsza współbieżność), ale dzięki temu zmniejsza się liczba blokad, którymi musi zarządzać serwer.

Zakleszczenia

Zakleszczenie (ang. *deadlock*) powstaje, gdy jeden proces próbuje założyć blokadę powodującą konflikt z blokadą, którą próbuje założyć inny proces. W wyniku nie mogą zostać założone blokady wymagane do ukończenia rozpoczętych procesów. Serwery bazodanowe posiadają algorytmy, które automatycznie wykrywają zakleszczenia i przerywają wykonywanie jednej z transakcji.

3.14.3. Izolowanie transakcji

Wiemy, że transakcja musi być izolowana, czyli próba odczytania danych z tabeli, na której przeprowadzana jest transakcja przez innego użytkownika, zakończy się niepowodzeniem. Dopiero pojawienie się polecenia `COMMIT` powoduje właściwą modyfikację danych, zdjęcie blokad z danych i umożliwia dostęp do nich innym użytkownikom.

W zależności od istniejącego na serwerze bazodanowym poziomu izolowania transakcji może pojawić się jeden z następujących problemów:

- **Utrata aktualizacji** (ang. *lost or buried updates*) — problem pojawi się, gdy dwie transakcje modyfikują te same dane. Żadna z transakcji nie wie, że druga transakcja korzysta z tych samych danych. W efekcie transakcja, która zakończy się później, zmodyfikuje dane, niszcząc zmiany dokonane przez transakcję, która zakończyła się wcześniej. Domyślnie skonfigurowany serwer nie dopuści do utraty aktualizacji.
- **Brudne odczyty** (ang. *dirty reads*) — problem pojawia się, gdy jedna transakcja dokonuje zmian danych, a druga w tym czasie odczytuje te dane. W efekcie transakcja odczytuje dane, które nie zostały jeszcze zatwierdzone i mogą zostać wycofane. Domyślnie skonfigurowany serwer nie dopuści do brudnych odczytów.
- **Niepowtarzalne odczyty** (ang. *non-repeatable reads*) — występują, gdy powtórzenie w ramach transakcji tego samego odczytu daje inny wynik. Może to być spowodowane tym, że po pierwszym odczycie (a nie po zakończeniu transakcji) zostaną zdjęte blokady założone na odczytywane dane. Niezablokowane dane mogą zostać zmienione przez inne działania, a powtórne ich odczytanie da inny wynik. Domyślnie skonfigurowany serwer dopuszcza niepowtarzalne odczyty.

- **Odczyty widma** (ang. *phantom reads*) — występują, gdy pomiędzy dwoma odczytami tych samych danych w ramach jednej transakcji zmieni się liczba odczytywanych wierszy (na przykład w wyniku wykonania w międzyczasie instrukcji `INSERT` lub `DELETE` na tych danych). Domyślnie skonfigurowany serwer dopuszcza odczyty widma.

3.14.4. Poziomy izolowania transakcji

Blokad używamy w celu zabezpieczenia danych w trakcie współbieżnego wykonywania transakcji. Pozwala to na wykonywanie transakcji w pełnej izolacji od innych transakcji i zapewnia przez cały czas poprawność danych w bazie.

Dzięki temu możliwe jest wykonywanie kilku transakcji w tym samym czasie, tak jakby były wykonywane jedna po drugiej, czyli szeregowo.

Transakcje nie zawsze wymagają pełnej izolacji. Możemy wpływać na sposób zakładania blokad przez serwery bazodanowe, zmieniając poziom izolacji transakcji.

Poziom izolacji określa stopień, do którego dana transakcja musi być izolowana od innych transakcji. Najniższy poziom izolacji to maksymalny stopień współbieżności, ale jest on okupiony najmniejszym stopniem spójności danych. Natomiast wyższy stopień izolacji transakcji zwiększa poprawność danych, ale zmniejsza stopień współbieżności. Najwyższy poziom izolacji gwarantuje najwyższy poziom spójności danych kosztem ograniczenia do minimum współbieżności.

Serwery bazodanowe pozwalają ustawiać na poziomie serwera, baz danych lub pojedynczych sesji poziom izolowania transakcji.

Standard SQL3 definiuje cztery poziomy izolacji transakcji:

- *Read Uncommitted*,
- *Read Committed*,
- *Repeatable Read*,
- *Serializable*.

Read Uncommitted

Read Uncommitted to tryb niezatwierdzonego odczytu. Jest to najniższy poziom izolacji transakcji. Odczyt danych nie powoduje założenia blokady współdzielonej. Tylko fizycznie uszkodzone dane nie będą odczytywane. Na tym poziomie pojawiają się brudne odczyty, niepowtarzalne odczyty i odczyty widma. Nie występuje jedynie problem z utratą aktualizacji. Ten tryb może być stosowany do odczytywania danych, o których wiemy, że w czasie odczytywania nie będą modyfikowane.

Ćwiczenie 3.18

W ramach jednej sesji w oknie edytora SQL rozpoczniemy transakcję, której zadaniem będzie zmodyfikowanie danych klienta w tabeli *Klient*.

```
BEGIN TRANSACTION;

UPDATE Klient

SET ulica = 'Mickiewicza', nr_domu=94

WHERE nazwisko='Nowak' AND imie='Andrzej';
```

Nie zamykając tej transakcji, w ramach kolejnej sesji, jako inny użytkownik (nowe okno edytora SQL), zmienimy poziom izolowania transakcji i spróbujemy odczytać dane modyfikowane przez pierwszego użytkownika:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

GO

SELECT miejscowosc, ulica, nr_domu

FROM Klient

WHERE nazwisko='Nowak' AND imie='Andrzej';
```

Mimo że pierwszy użytkownik nie zamknął transakcji, udało się odczytać zmienione dane klienta.

Kończąc przykład, należy zamknąć transakcję bez zatwierdzania zmian.

Read Committed

Read Committed to tryb odczytu zatwierdzonego. Jest to domyślny poziom izolacji stosowany przez MS SQL Server. Podczas odczytu danych zostanie założona na nie blokada współdzielona. Założona blokada eliminuje brudne odczyty. Natomiast nadal występują niepowtarzalne odczyty oraz odczyty widma.

Ćwiczenie 3.19

Otwieramy dwa okna edytora SQL. W pierwszym oknie ustawiamy tryb odczytu zatwierdzonego, rozpoczynamy transakcję i odczytujemy dane klienta:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

BEGIN TRANSACTION;

SELECT miejscowosc, ulica, nr_domu

FROM Klient

WHERE nazwisko='Nowak' AND imie='Andrzej';
```

Pozostawiamy otwartą transakcję i w drugim oknie zmieniamy adres klienta:

```
UPDATE Klient

SET ulica = 'Sienkiewicza', nr_domu=21

WHERE nazwisko='Nowak' AND imie='Andrzej';
```

Ponownie wracamy do okna pierwszego. W ramach tej samej transakcji drugi raz odczytujemy dane klienta:

```
SELECT miejscowosc, ulica, nr_domu
FROM Klient
WHERE nazwisko='Nowak' AND imie='Andrzej';
COMMIT;
```

Adres ponownie odczytany jest inny niż odczytany wcześniej. Wystąpił efekt niepowtarzalnych odczytów.

Repeatable Read

Repeatable Read to tryb powtarzalnego odczytu. Założona na dane blokada współdzielona jest utrzymywana aż do zakończenia całej transakcji. Dzięki temu, że inny proces nie może zmodyfikować odczytywanych danych, zostały wyeliminowane niepowtarzalne odczyty. Natomiast nadal występują odczyty widma.

Ćwiczenie 3.20

Otwieramy dwie sesje edytora SQL. W pierwszej ustawiamy tryb powtarzalnego odczytu, rozpoczynamy transakcję i odczytujemy nazwiska klientów mieszkających w Poznaniu:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN TRANSACTION;

SELECT nazwisko, imie
FROM Klient
WHERE miejscowosc='Poznań';
```

Pozostawiamy otwartą transakcję i w drugiej sesji zmieniamy na Poznań miejsce zamieszkania jednego z klientów, który nie mieszkał w Poznaniu:

```
UPDATE Klient
SET miejscowosc = 'Poznań'
WHERE nazwisko='Kowalski' AND imie='Jan';
```

Ponownie wracamy do pierwszej sesji. W ramach tej samej transakcji drugi raz odczytujemy nazwiska klientów mieszkających w Poznaniu:

```
SELECT nazwisko, imie
FROM Klient
WHERE miejscowosc='Poznań';
```

Tym razem liczba odczytywanych wierszy uległa zmianie. Jest ich więcej. Pojawił się wiersz widmo.

Zmiana danych w drugiej sesji była możliwa, ponieważ dotyczyła danych, które nie były odczytywane w transakcji otwartej w pierwszej sesji.

Na zakończenie przykładu należy wykonać instrukcję zamknięcia transakcji `COMMIT`.

Sytuacja ulegnie zmianie, gdy w drugiej sesji będziemy próbowali zmienić dane odczytywane w pierwszej sesji.

Ćwiczenie 3.21

Otwieramy dwie sesje edytora SQL. W pierwszej ustawiamy tryb powtarzalnego odczytu, rozpoczynamy transakcję i odczytujemy nazwiska klientów mieszkających w Gdańsku:

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
BEGIN TRANSACTION;

SELECT nazwisko, imie
FROM Klient
WHERE miejscowosc='Gdańsk';
```

Pozostawiamy otwartą transakcję i w drugiej sesji zmieniamy na Kraków miejsce zamieszkania jednego z klientów, który mieszkał w Gdańsku:

```
UPDATE Klient
SET miejscowosc = 'Kraków'
WHERE miejscowosc = 'Gdańsk';
```

Ta instrukcja będzie oczekiwać na zakończenie transakcji w pierwszej sesji i zdjęcie blokady z danych.

Aby aktualizacja danych była możliwa, w pierwszej sesji należy wykonać instrukcję zamknięcia transakcji `COMMIT`. Efekt niepowtarzalnego odczytu został wyeliminowany.

Serializable

Serializable to tryb szeregowania. Transakcje odwołujące się do tych samych danych są szeregowane, czyli wykonywane jedna po drugiej. Jest to najwyższy poziom izolacji transakcji, gdzie transakcje są w pełni izolowane od siebie. Na czas trwania transakcji ulegają zablokowaniu całe obiekty, a nie tylko odczytywane dane, co pozwala wyeliminować odczyty widma, ale powoduje, że inni użytkownicy nie mogą modyfikować przechowywanych w obiekcie danych. Na przykład odczytując jeden wiersz tabeli, blokujemy możliwość modyfikowania danych w całej tabeli.

Ćwiczenie 3.22

Otwieramy dwie sesje edytora SQL. W pierwszej ustawiamy tryb szeregowania, rozpoczynamy transakcję i odczytujemy informacje o wybranym kliencie:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRANSACTION;

SELECT miejscowosc, ulica, nr_domu
FROM Klient
WHERE nazwisko='Adamek' AND imie='Marek';
```

Jeżeli w drugiej sesji spróbujemy zmienić dane innego klienta, to okaże się, że aktualizacja została zablokowana:

```
UPDATE Klient
SET miejscowosc = 'Warszawa'
WHERE nazwisko='Borewicz' AND imie='Adam';
```

Aktualizacja będzie możliwa dopiero po zakończeniu transakcji w pierwszej sesji.

Na zakończenie przykładu należy zamknąć obydwie sesje bez zatwierdzania transakcji.

W tym trybie odczytywane dane zawsze są takie same. Jednak w trakcie transakcji pozostali użytkownicy nie mogą modyfikować zablokowanych tabel. Powoduje to znaczne wydłużenie czasu dostępu do danych i w praktyce, jeśli zmian nie jest dużo, zaleca się przełączenie do modelu optymistycznego.

3.15. T-SQL

T-SQL (*Transact-SQL*) jest rozszerzeniem języka SQL stosowanym dla serwerów MS SQL Server. W jego skład, oprócz standardowych poleceń języka SQL, wchodzi instrukcje tworzenia pętli, instrukcje warunkowe, zmienne, wyzwalacze, funkcje i procedury.

W języku T-SQL ciąg poleceń bezpośrednio kierowany do serwera powinien kończyć się słowem kluczowym GO. Ich wykonanie na poziomie edytora nastąpi po wybraniu przycisku *Execute* lub wciśnięciu klawisza *F5* na klawiaturze.

Pojedyncza instrukcja może kończyć się średnikiem, ale brak średnika nie jest błędem. Instrukcje mogą być pisane w jednej linii. Jednak dla większej czytelności kodu zaleca się umieszczanie każdej z nich w oddzielnej linii i stosowanie wcięć.

3.15.1. Zmienne

Zmienne deklarowane przez użytkownika są zmiennymi lokalnymi i istnieją tylko w obrębie skryptu. Muszą być poprzedzone znakiem @. Deklaracja zmiennych jest realizowana za pomocą instrukcji:

```
DECLARE @zmienna typ_danych
```

W jednej instrukcji można deklarować wiele zmiennych.

Przykład 3.97

```
DECLARE @Imie nvarchar (10), @Nazwisko nvarchar (30);
```

W podanym przykładzie zostały zadeklarowane dwie zmienne lokalne: *@Imie* i *@Nazwisko*.

Zmienne systemowe

Zmienne systemowe oznaczone są dwoma znakami @@.

Przykłady zmiennych systemowych:

- @@ERROR — zwraca numer ostatniego błędu (0 — brak błędu),
- @@FETCH_STATUS — ustala, czy cursor pobrał wiersz (0 — pobrał),
- @@IDENTITY — zwraca ostatnio wygenerowaną wartość,
- @@ROWCOUNT — zwraca liczbę wierszy, dla których została wykonana instrukcja SQL,
- @@VERSION — zwraca informację o wersji SQL,
- @@TRANCOUNT — zwraca liczbę otwartych transakcji.

Zmiennej można przypisać wartość za pomocą instrukcji SELECT:

```
SELECT @zmienna=wrażenie [FROM ...]
```

Przykład 3.98

```
SELECT @Imie='Jan', @Nazwisko='Kowalski';
```

Innym sposobem przypisania wartości do zmiennej jest użycie instrukcji SET. Instrukcją SET można przypisać tylko jedną wartość.

Przykład 3.99

```
SET @Imie='Jan';
```

```
SET @Nazwisko='Kowalski';
```

Zmiennym można przypisywać wartości zwrócone przez zapytanie. Zapytanie takie musi jednak zwracać dokładnie jeden wiersz.

Przykład 3.100

```
SELECT @Imie=imie, @Nazwisko=nazwisko
FROM Klient
WHERE id_klienta=1;
```

Jeśli zapytanie zwróci więcej wierszy, to zmiennym zostaną przypisane wartości z ostatniego wiersza. Natomiast jeśli zapytanie nie zwróci żadnego wiersza, zmienne zachowają swoje dotychczasowe wartości.

Wartość przypisaną zmiennej można odczytać za pomocą instrukcji SELECT.

Przykład 3.101

```
INSERT INTO Ksiazki (tytul) VALUES ('Dolina Issy')
SELECT @@IDENTITY;
```

Instrukcja INSERT INTO doda do tabeli *Ksiazki* nowy rekord i w polu *tytul* umieści podaną wartość. Dodatkowo dla pola *id_ksiazki*, które nie może pozostać puste, zostanie wygenerowana wartość liczbowa. Zapytanie SELECT @@IDENTITY zwróci wartość wygenerowaną dla tego pola.

3.15.2. Instrukcja warunkowa

Instrukcja warunkowa używana jest do określania sposobu wykonania kodu. Ogólna postać instrukcji wygląda następująco:

```
IF warunek_logiczny
    polecenie
ELSE
    polecenie
```

Przykład 3.102

```
DECLARE @Imie nvarchar (10), @Nazwisko nvarchar (30);
SELECT @Imie='Adam', @Nazwisko='Nowak';
IF NOT EXISTS (SELECT TOP 1 * FROM Klient WHERE imie=@Imie AND nazwisko=
@Nazwisko)
INSERT INTO Klient (imie, nazwisko)
VALUES ('Adam', 'Nowak');
```

Wynikiem będzie sprawdzenie, czy klient o nazwisku Adam Nowak został zarejestrowany w bazie, a jeżeli nie został zarejestrowany — dodanie go do tabeli *Klient*.

Przykład 3.103

```
IF object_id ('Zbior_ksiazek') IS NOT NULL
DROP VIEW Zbior_ksiazek;
GO
CREATE VIEW Zbior_ksiazek AS
SELECT tytul, nazwisko, imie, cena
FROM Autor INNER JOIN Ksiazki
ON Autor.id_autora= Ksiazki.id_autora;
```

Funkcja systemowa `object_Id (nazwa_obiektu)` zwraca numer identyfikacyjny obiektu. Została tutaj użyta do sprawdzenia, czy podany obiekt istnieje. Jeżeli tak jest, obiekt ten zostanie usunięty.

3.15.3. Wyrażenie CASE

Wyrażenie CASE może być użyte wewnątrz niektórych poleceń języka SQL. Należą do nich polecenia: SELECT, UPDATE, DELETE, SET oraz klauzule: IN, WHERE, ORDER BY, HAVING. Wyrażenie CASE sprawdza dla każdego wiersza wartość otrzymaną w zapytaniu i porównuje ją z wynikiem wyrażenia. W zależności od wyniku zwraca wartość zapisaną po słowie kluczowym THEN.

```
CASE
{WHEN wyrażenie_logiczne THEN wyrażenie_wynikowe}
[ELSE wyrażenie_wynikowe]
END
```

Przykład 3.104

```
SELECT [cena],
CASE
WHEN [cena] < 20 THEN 'Tania książka'
WHEN [cena] < 80 THEN 'Książka w średniej cenie'
ELSE 'Książka z najwyższej półki'
END
FROM Ksiazki;
```

Wyrażenie CASE sprawdza dla każdego wiersza tabeli *Ksiazki* wartość pola [*cena*] i porównuje ją z wartościami *20* i *80*. W zależności od wyniku porównania zwraca wartość zapisaną po słowie kluczowym THEN.

3.15.4. Wyrażenia tablicowe (CTE)

Wyrażenia tablicowe umożliwiają definiowanie wirtualnych tabel, do których można się odwoływać wielokrotnie za pomocą ich nazwy. W celu utworzenia wyrażenia tablicowego należy użyć polecenia WITH w postaci:

```
WITH nazwa_wirtualnej_tabeli
AS
(zapytanie)
```

Po wykonaniu zapytania zostanie utworzona wirtualna tabela, z którą można pracować jak ze zwykłą tabelą.

Wyrażenia tablicowe mogą być stosowane:

- do tworzenia zapytań cyklicznych,
- jako odpowiedniki widoków, jeżeli nie ma potrzeby przechowywania definicji metadanych,
- przy wielokrotnym odwoływaniu się do tabeli wynikowej w tym samym zapytaniu.

Zaletą stosowania wyrażeń tablicowych jest większa czytelność złożonych zapytań. Mogą one służyć do budowania prostych bloków logicznych, z których tworzone są bardziej złożone zapytania. Mogą być definiowane i używane w funkcjach i procedurach składowanych oraz w wyzwalaczach i widokach.

Wykonanie wyrażenia tablicowego następuje po podaniu polecenia `SELECT`:

```
SELECT lista_kolumn
FROM nazwa_wirtualnej_tabeli
```

Przykład 3.105

```
WITH CTE_Klient
AS
(SELECT id_klienta, imie + ' ' + nazwisko AS Dane,
kod_pocztowy + ' ' + miejscowosc AS Adres,
ulica + ' ' + nr_domu AS Ul
FROM Klient)
SELECT id_klienta, Dane, Adres, Ul
FROM CTE_Klient
WHERE id_klienta > 20;
```

Najpierw została utworzona wirtualna tabela `CTE_Klient`, a następnie na podstawie tej tabeli w zapytaniu zostały wybrane dane do pokazania.

Jeżeli połączymy wyniki dwóch wyrażeń tabelarycznych, uzyskamy możliwość wykonywania rekurencyjnych zapytań.

3.15.5. Procedury składowane

Procedury składowane to polecenie lub kilka poleceń, które są wykonywane jako całość w jednym bloku. Mogą mieć zadeklarowane zarówno parametry wejściowe, jak i wyjściowe. Mogą również zawierać polecenia kontrolowania kodu, takie jak `IF` lub `WHILE`. Zaleca się, aby wszystkie powtarzalne czynności w bazie danych były definiowane przy użyciu procedur składowanych.

Zdefiniowana przez użytkownika procedura jest tworzona w bieżącej bazie danych. Wyjątkiem są procedury tymczasowe — one zawsze są tworzone w folderze `tempdb`.

Definiowanie procedury składowanej wygląda następująco:

```
CREATE PROCEDURE Nazwa_procedury
(
    @nazwa_parametru_1 typ_parametru, ..., @nazwa_parametru_n typ_parametru
)
AS
Treść_procedury
```

Przykład 3.106

Utworzona procedura będzie pobierała parametr z ceną książek i zwracała listę tylko tych książek, których ceny są niższe niż cena podana jako parametr.

```
CREATE PROCEDURE cena_ksiazki
(@cena_ks money)
AS
BEGIN
    Print 'Książki o cenie niższej niż' + CAST (@cena_ks AS varchar (10));
    SELECT tytuł, [cena]
    FROM Ksiazki
    WHERE [cena] < @cena_ks;
END
GO
```

Pierwsza instrukcja tworzy procedurę o nazwie `cena_ksiazki` z parametrem wejściowym `@cena_ks` typu `money`.

Kolejne dwie instrukcje to instrukcje procedury. Pierwsza instrukcja to polecenie wydrukowania podanego tekstu połączonego z wartością parametru wejściowego, którego typ został zmieniony z `money` na `varchar (10)`. Druga to zapytanie, w którym jako warunek klauzuli `WHERE` występuje parametr wejściowy, zwracające listę wszystkich książek, które kosztują mniej niż wartość parametru wejściowego.

Procedury składowane mogą być wykonywane w następujący sposób:

```
EXECUTE nazwa_procedury;
```

lub:

```
EXEC nazwa_procedury;
```

Przykład 3.107

Aby uruchomić procedurę dla książek tańszych niż 90 zł, należy wpisać polecenie `EXECUTE` w podanej postaci:

```
EXECUTE cena_ksiazki 90.00;
```

```
GO
```

Przykład 3.108

Mamy utworzyć procedurę, która po podaniu nazwiska i imienia klienta będzie zwracała jego dane.

```
CREATE PROCEDURE Dane_klienta
(
    @Nazwisko nvarchar (50), @Imie nvarchar (40)
)
AS
SET NOCOUNT ON;

SELECT nazwisko, imie, kod_pocztowy, miejscowosc, ulica, nr_domu, telefon
FROM Klient
WHERE nazwisko=@Nazwisko AND imie=@Imie;

GO
```

Procedura może zostać wykonana w następujący sposób:

```
EXECUTE Dane_klienta 'Kowalski', 'Jan';
```

lub:

```
EXEC Dane_klienta @Nazwisko = 'Kowalski', @Imie = 'Jan';
```

Parametry do procedury można przekazywać anonimowo (bez określania, któremu parametrowi zostanie przypisana wartość). Jest to dobre rozwiązanie, gdy procedura ma jeden parametr wejściowy. Natomiast w przypadku, gdy tych parametrów jest więcej (jak w podanym przykładzie), lepiej stosować jawne przypisanie wartości do parametru.

Jeżeli procedura jest wywoływana jako pierwszy element w skrypcie, słowo kluczowe `EXECUTE` może zostać pominięte.

Użyta w procedurze składowanej opcja `SET NOCOUNT ON` zapobiega wyświetlaniu komunikatu pokazującego, na ilu wierszach działa instrukcja.

UWAGA

Komunikat `DONE_IN_PROC` jest wysyłany do użytkownika dla każdej instrukcji umieszczonej w procedurze składowanej. W przypadku procedur składowanych zawierających kilka instrukcji, które nie zwracają danych, oraz procedur zawierających pętle ustawienie opcji `SET NOCOUNT = ON` może znacznie przyspieszyć wykonywanie zapytań.

Przykład 3.109

```

CREATE PROCEDURE Sprzedaz_ksiazek
(
    @Tyt_ksiazki nvarchar (100)
)
AS
SET NOCOUNT ON;

SELECT Ksiazki.tytul, SUM(liczba_egz) AS [Liczba książek]
FROM Ksiazki INNER JOIN Rejestracja_zamowienia
ON Ksiazki.id_ksiazki = Rejestracja_zamowienia.id_ksiazki
WHERE Ksiazki.tytul=@Tyt_ksiazki
GROUP BY Ksiazki.tytul;

```

Po podaniu tytułu książki procedura zwróci liczbę sprzedanych książek o podanym tytule. Procedura może zostać wielokrotnie użyta w analizie statystycznej dotyczącej sprzedaży poszczególnych tytułów.

Wbudowane procedury

Oprócz definiowania własnych procedur pracę z bazą danych może usprawnić stosowanie procedur wbudowanych.

- `sys.sp_addrolemember` — dodaje użytkownika do roli bazy danych. Parametry to `@rolename` i `@membername`.
- `sys.sp_adduser` — dodaje użytkownika do bazy danych. Parametry to: `@loginame`, `@name_in_db`, `@grpname`.
- `sys.sp_catalogs` — zwraca listę katalogów serwera. Parametr to `@server_name`.
- `sys.sp_columns` — zwraca szczegółowe informacje dotyczące kolumn wybranej tabeli. Parametry to: `@table_name`, `@table_owner`, `@table_qualifier`, `@column_name`, `@ODBCVer`.
- `sys.sp_databases` — zwraca listę wszystkich baz danych dostępnych na serwerze wraz z informacją o ich rozmiarze.
- `sys.sp_help` — zwraca szczegółowe informacje (typ, precyzja itp.) wskazanego obiektu bazodanowego (na przykład tabeli).

3.15.6. Funkcje składowane

Funkcje składowane działają podobnie jak procedury składowane, ale sposób ich wykorzystania jest zupełnie inny. Jedną z cech takiej funkcji jest to, że należy zadeklarować w niej typ zwracanego wyniku. Instrukcja `SELECT` wykorzystywana w procedurze może zwracać pewną wartość, ale nie musi, natomiast w funkcji jest to wymagane. Dlatego

zalecane jest, aby procedury były wykorzystywane do zapisywania, modyfikowania i usuwania rekordów, czyli działań, które nie zwracają wyniku, funkcje zaś do obliczeń typu: liczenie średniej, przeliczanie wartości itp.

Podstawowa postać funkcji to:

```
CREATE FUNCTION nazwa_funkcji
(
    @nazwa_parametru_1 typ_parametru, ..., @nazwa_parametru_n typ_parametru
)
RETURNS typ_zwracany
AS
BEGIN
    Treść_funkcji
RETURN wartość_zwracana
END;
```

Podczas definiowania funkcji trzeba zdefiniować nagłówek (nazwa, lista parametrów, typ zwracanej wartości) i treść funkcji. Ostatnim poleceniem treści powinna być instrukcja RETURN, która określa wartości zwracane przez funkcję.

Przykład 3.110

```
CREATE FUNCTION Data_pl
(
    @pl_data datetime,
    @odstep varchar (8)
)
RETURNS nvarchar (32)
AS
BEGIN
RETURN
    CONVERT (nvarchar (32), DATEPART (dd, @pl_data))
    +@odstep
    + CONVERT (nvarchar (32), DATEPART (mm, @pl_data))
    +@odstep
    + CONVERT (nvarchar (32), DATEPART (yy, @pl_data))
END
```

Funkcja zmienia format daty z 2012-10-20 na bardziej przyjazny 20-10-2012.

Jeżeli w tabeli *Klient* zostanie umieszczone pole [*data urodzenia*], to wywołanie funkcji może mieć postać:

```
SELECT nazwisko, imie, data_pl([data urodzenia], '-')
FROM Klient;
```

Przykład tej funkcji pokazuje, jak duże możliwości daje stworzenie własnej biblioteki ze skryptami zawierającymi funkcje, które będziemy mogli wykorzystywać w wielu tworzonych projektach.

Funkcje wbudowane

Podobnie jak w przypadku procedur, użytkownik może skorzystać z istniejących funkcji wbudowanych. Najczęściej wykorzystywane funkcje to:

- AVG — obliczanie wartości średniej,
- SUM — sumowanie wartości,
- COUNT — zliczanie liczby wystąpień,
- GETDATE — zwracanie wartości bieżącej daty i czasu,
- DAY — zwracanie dnia miesiąca jako liczby całkowitej,
- YEAR — zwracanie roku jako liczby całkowitej,
- MONTH — zwracanie miesiąca jako liczby całkowitej.

3.15.7. Wyzwalacze

Wyzwalacze (ang. *triggers*) to specjalny rodzaj procedur składowanych, które są automatycznie wykonywane, gdy na serwerze wystąpi określone zdarzenie. Tymi zdarzeniami są operacje przeprowadzane przez użytkownika. Dopiero po wykonaniu instrukcji uruchamiany jest wyzwalacz. Polecenia wykonywane w ramach wyzwalacza są traktowane jako transakcje rozpoczęte jawnie lub niejawnie przez użytkownika.

Głównym przeznaczeniem wyzwalaczy jest wymuszenie integralności danych. Tylko właściciel tabeli może utworzyć powiązany z nią wyzwalacz. Prawo do tworzenia wyzwalaczy nie można nikomu przekazać. Możliwe jest definiowanie dla tabeli dowolnej liczby powiązanych z nią wyzwalaczy. Nie można definiować wyzwalaczy tabel tymczasowych. Wyzwalacze nie zwracają żadnych danych. Są automatycznie uruchamiane w odpowiedzi na działania użytkownika.

Do tworzenia wyzwalaczy służy instrukcja `CREATE TRIGGER`.

W języku SQL występują trzy rodzaje wyzwalaczy:

- wyzwalacze DML,
- wyzwalacze DDL,
- wyzwalacze logowania.

Wyzwalacze DML

Wyzwalacze DML są wywoływane, gdy zostanie wykonane jedno z poleceń języka DML, na przykład `INSERT`, `UPDATE` lub `DELETE`. Takie wyzwalacze są użyteczne:

- przy modyfikowaniu danych w powiązanych tabelach (jednak lepsze efekty uzyskamy, stosując w tej sytuacji kaskadowe więzy integralności);
- przy egzekwowaniu warunków ograniczających zakres danych wprowadzanych do kolumny zdefiniowanej w instrukcjach `INSERT`, `UPDATE` i `DELETE` (w przeciwieństwie do warunków ograniczających, definiowanych przy użyciu atrybutu `CHECK`, wyzwalacze DML mogą odwoływać się do kolumn w innych tabelach);
- przy podejmowaniu działań na podstawie odczytu stanu tabeli przed modyfikacją i po modyfikacji.

Można wyróżnić następujące typy wyzwalaczy DML:

- Wyzwalacze *After* — są uruchamiane po instrukcji generującej zdarzenie. Mogą być definiowane tylko dla tabel. Wyzwalacz zostanie uruchomiony tylko wtedy, gdy wszystkie operacje generujące zdarzenie zostały wykonane pomyślnie.
- Wyzwalacze *Instead of* — są wykonywane zamiast instrukcji generującej zdarzenie. Są definiowane dla widoków opartych na tabelach i aktualizują zawartość tych widoków.
- Wyzwalacze *CLR* — mogą być wyzwalaczami *After* lub *Instead of*. Mogą być również wyzwalaczami DLL. Zamiast wykonywania transakcji dla procedury składowanej wykonują kod zewnętrzny utworzony w .NET Framework i przesłany do serwera SQL Server.

Postać instrukcji `CREATE TRIGGER` dla wyzwalaczy DML:

```
CREATE TRIGGER nazwa_wyzwalacza ON nazwa_tabeli/widoku
{FOR/AFTER/INSTEAD OF} {INSERT, DELETE, UPDATE}
AS
Polecenia/EXTERNAL NAME nazwa_wywoływanej_procedury
```

gdzie:

nazwa_tabeli/widoku — tabela lub widok, dla którego wykonywany jest wyzwalacz. Podanie nazwy tabeli lub wyzwalacza jest opcjonalne. Widok może wystąpić tylko w wyzwalaczu *Instead of*. Wyzwalacze DML nie można tworzyć dla tabel tymczasowych.

`FOR/AFTER` — wyzwalacz zostanie uruchomiony tylko wtedy, gdy wszystkie operacje generujące zdarzenie zostały wykonane pomyślnie. Zdefiniowane kaskadowe aktualizowanie więzów integralności oraz warunki ograniczające również muszą dać pozytywne efekty.

`INSTEAD OF` — wyzwalacz jest wykonywany zamiast generującej go instrukcji. Tylko jeden wyzwalacz tego typu może zostać zdefiniowany dla tabeli lub widoku. Natomiast można zdefiniować widoki dla widoku i każdy z nich może mieć swój wyzwalacz typu `INSTEAD OF`.

INSERT, UPDATE i DELETE — określają instrukcje modyfikacji danych, które mogą uruchomić wyzwalacz dla określonej tabeli lub dla określonego widoku. Co najmniej jedna z instrukcji musi zostać wybrana. W wyzwalaczach typu `INSTEAD OF` instrukcja DELETE nie jest dozwolona dla tabel, które mają zdefiniowaną opcję kaskadowego usuwania danych. Podobnie jak instrukcja INSERT, nie jest ona dozwolona w przypadku tabel, które mają zdefiniowaną opcję kaskadowego aktualizowania danych.

Przykład 3.111

```
CREATE TRIGGER modyfikuj
ON dbo.Klient
AFTER INSERT, UPDATE
AS RAISERROR ('Uwaga! Zmiana danych', 16, 10);
```

Wyzwalacz wysyła do użytkownika komunikat, gdy ktoś próbuje dodać dane do tabeli *Klient* lub je zmienić.

`RAISERROR(...)` to funkcja generująca komunikat o błędzie.

Wyzwalacze DDL

Wyzwalacze DDL są uruchamiane w wyniku zdarzeń zachodzących po wykonaniu poleceń języka DDL. Dotyczy to poleceń `CREATE`, `ALTER` i `DROP`. Często są wykorzystywane do sprawdzania stanu odpowiedzi wysyłanych przez procedury składowane do systemu. Mogą być wykorzystywane do zadań administracyjnych, takich jak audyt czy regulowanie operacji bazodanowych.

Są użyteczne, gdy trzeba:

- uniemożliwić wprowadzenie zmian w schemacie bazy danych,
- wykonać działania w bazie danych w odpowiedzi na zmiany w schemacie bazy,
- zapamiętać zmiany w schemacie bazy danych.

Wyzwalacze DDL są uruchamiane tylko po poleceniach języka DDL. Nie mogą być używane wyzwalacze `INSTEAD OF`.

Postać instrukcji `CREATE TRIGGER` dla wyzwalaczy DDL:

```
CREATE TRIGGER nazwa_wyzwalacza ON ALL SERVER/DATABASE
{FOR /AFTER} {nazwa_zdarzenia}
AS
Polecenia/EXTERNAL NAME nazwa_wywoływanej_procedury
```

gdzie:

DATABASE — działanie wyzwalacza dotyczy bieżącej bazy danych,

ALL SERVER — działanie wyzwalacza dotyczy bieżącego serwera,

nazwa_zdarzenia — nazwa zdarzenia, po którym nastąpi wywołanie wyzwalacza.

Przykładowe zdarzenia wykorzystywane przez wyzwalacze DDL:

- CREATE_TABLE,
- ALTER_TABLE,
- CREATE_SCHEMA,
- GRANT_DATABASE,
- CREATE_INDEX,
- DROP_INDEX.

Słowa kluczowe zawsze są oddzielone znakiem podkreślenia `_`.

Przykład 3.112

```
CREATE TRIGGER blokuj
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
PRINT 'Musisz wyłączyć wyzwalacz "blokuj", aby zmienić lub usunąć tabelę!'
ROLLBACK;
```

Zdefiniowany wyzwalacz zadziała zawsze, gdy w bazie danych wystąpi zdarzenie `DROP_TABLE` lub `ALTER_TABLE`. Zdarzenia te są generowane przez instrukcje `DROP TABLE` i `ALTER TABLE`.

Wyzwalacze logowania

Wyzwalacze logowania uruchamiają się w odpowiedzi na zdarzenia związane z logowaniem. Uruchamiane są po zakończeniu fazy uwierzytelnienia logowania, ale przed rozpoczęciem sesji użytkownika. Nie są uruchamiane, gdy logowanie się nie powiedzie.

Postać instrukcji `CREATE TRIGGER` dla wyzwalaczy logowania:

```
CREATE TRIGGER nazwa_wyzwalacza ON ALL SERVER
{FOR /AFTER} LOGON
AS
Polecenia/EXTERNAL NAME nazwa_wywoływanej_procedury
```

Uprawnienia

Do tworzenia wyzwalaczy DML wymagane są uprawnienia `ALTER` dla tabeli lub widoku, dla którego tworzony jest wyzwalacz.

Do tworzenia wyzwalaczy DDL dla serwera (`ON ALL SERVER`) oraz wyzwalaczy logowania wymagane są uprawnienia `CONTROL SERVER` na serwerze.

Do tworzenia wyzwalaczy DDL dla bazy danych (ON DATABASE) wymagane są uprawnienia ALTER DATABASE.



3.16. Pytania i zadania

3.16.1. Pytania

1. Podaj podstawowe cechy języka SQL.
2. Przedstaw hierarchię obiektów bazy danych w systemie SQL Server.
3. Jakiego typu działania są realizowane za pomocą języka DDL?
4. Jakiego typu działania są realizowane za pomocą języka DML?
5. Za pomocą jakich klauzul języka SQL realizowane są połączenia zewnętrzne?
6. Jaką rolę w zapytaniu spełniają podzapytania (zapytania zagnieżdżone)?
7. Kiedy należy stosować transakcje?
8. Wymień właściwości transakcji.
9. Na czym polega współbieżność w dostępie do bazy danych?
10. Na czym polega izolowanie transakcji?
11. Co to jest procedura składowana?
12. Jaką rolę w języku T-SQL spełniają wyzwalacze?

3.16.2. Zadania

Zadanie 1.

Korzystając z wybranego serwera baz danych i języka SQL, utwórz dla bazy danych *Moja_szkoła* zapytania określające sposób wybierania i przetwarzania danych zgromadzonych w tabelach. Wykorzystując narzędzia języka SQL, zbuduj pełną obsługę tej bazy.

Załącznik 1.

Tabele wchodzące w skład bazy danych księgarnia

Tabela Klient

- *id_klienta*,
- *nazwisko*,
- *imie*,
- *kod_pocztowy*,
- *miescowosc*,
- *ulica*,

- *nr_domu*,
- *PESEL*,
- *telefon*,
- *adres_e_mail*.

Tabela Ksiazki

- *id_ksiazki*,
- *tytul*,
- *id_autora*,
- *cena*,
- *wydawnictwo*,
- *temat*,
- *miejsce_wydania*,
- *jezyk_ksiazki*,
- *opis*,
- *rok_wydania*.

Tabela Zamowienia

- *id_zamowienia*,
- *id_klienta*,
- *data_zlozenia_zamowienia*,
- *data_wyslania*,
- *koszt_wyslki*,
- *id_faktury*.

Tabela Autor

- *id_autora*,
- *nazwisko*,
- *imie*,
- *narodowosc*,
- *okres_tworzenia*,
- *jezyk*,
- *rodzaj_tworczosci*,
- *osiagniecia*.

Tabela Rejestracja_zamowienia

- *id_zamowienia*,
- *id_ksiazki*,
- *liczba_egz*.

Tabela Faktura

- *nr_faktury*,
- *sposob_platnosci*,
- *data_wystawienia_faktury*.

Skrypty tworzące tabele dla bazy ksiegarnia (SQL Server)

Tabela Autor

```
CREATE TABLE Autor
(
    id_autora int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    nazwisko nvarchar (50) NOT NULL,
    imie nvarchar (30) NOT NULL,
    narodowosc nvarchar (30),
    okres_tworzenia nvarchar (35),
    jezyk nvarchar (30),
    rodzaj_tworczosci nvarchar (35),
    osiagniecia nvarchar (200)
);
```

Tabela Faktura

```
CREATE TABLE Faktura
(
    nr_faktury int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    sposob_platnosci nvarchar (30),
    data_wystawienia_faktury date
);
```

Tabela Klient

```
CREATE TABLE Klient
(
    id_klienta int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    nazwisko nvarchar (60) NOT NULL,
    imie nvarchar (40) NOT NULL,
    kod_pocztowy nvarchar (6),
    miejscowosc nvarchar (50) DEFAULT 'Warszawa',
    ulica nvarchar (50),
```

```

        nr_domu nvarchar (7),
        PESEL nvarchar (11) NOT NULL,
        telefon nvarchar (12) UNIQUE,
        adres_e_mail nvarchar (70)
    );

```

Tabela Ksiazki

```

CREATE TABLE Ksiazki
(
    id_ksiazki int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    tytul nvarchar (100) NOT NULL,
    id_autora int NOT NULL,
    cena money,
    wydawnictwo nvarchar (20),
    temat nvarchar (30),
    miejsce_wydania nvarchar (28),
    jezyk_ksiazki nvarchar (15),
    opis nvarchar (100),
    rok_wydania nvarchar (4)
);

```

Tabela Rejestracja_zamowienia

```

CREATE TABLE Rejestracja_zamowienia
(
    id_zamowienia int,
    id_ksiazki int,
    liczba_egz int
);

```

Tabela Zamowienia

```

CREATE TABLE Zamowienia
(
    id_zamowienia int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    data_zlozenia_zamowienia date,
    data_wyslania date,
    koszt_wysylki money,

```

```
id_klienta int NOT NULL,  
id_faktury int NOT NULL  
);
```

Skrypty tworzące tabele dla bazy księgarnia (MySQL)

Tabela Autor

```
CREATE TABLE Autor  
(  
    id_autora int AUTO_INCREMENT NOT NULL PRIMARY KEY,  
    nazwisko varchar (50) NOT NULL,  
    imie varchar (30) NOT NULL,  
    narodowosc varchar (30),  
    okres_tworzenia varchar (35),  
    jezyk varchar (30),  
    rodzaj_tworczosci varchar (35),  
    osiagniecia varchar (200)  
);
```

Tabela Faktura

```
CREATE TABLE Faktura  
(  
    nr_faktury int AUTO_INCREMENT NOT NULL PRIMARY KEY,  
    sposob_platnosci varchar (30),  
    data_wystawienia_faktury date  
);
```

Tabela Klient

```
CREATE TABLE Klient  
(  
    id_klienta int AUTO_INCREMENT NOT NULL PRIMARY KEY,  
    nazwisko varchar (60) NOT NULL,  
    imie varchar (40) NOT NULL,  
    kod_pocztowy varchar (6),  
    miejscowosc varchar (50) DEFAULT 'Warszawa',  
    ulica varchar (50),  
    nr_domu varchar (7),
```



```

        PESEL varchar (11) NOT NULL,
        telefon varchar (12) UNIQUE,
        adres_e_mail varchar (70)
    );

```

Tabela Ksiazki

```

CREATE TABLE Ksiazki
(
    id_ksiazki int AUTO_INCREMENT NOT NULL PRIMARY KEY,
    tytul varchar (100) NOT NULL,
    id_autora int NOT NULL,
    cena decimal,
    wydawnictwo varchar (20),
    temat varchar (30),
    miejsce_wydania varchar (28),
    jezyk_ksiazki varchar (15),
    opis varchar (100),
    rok_wydania varchar (4)
);

```

Tabela Rejestracja_zamowienia

```

CREATE TABLE Rejestracja_zamowienia
(
    id_zamowienia int,
    id_ksiazki int,
    liczba_egz int
);

```

Tabela Zamowienia

```

CREATE TABLE Zamowienia
(
    id_zamowienia int AUTO_INCREMENT NOT NULL PRIMARY KEY,
    data_zlozenia_zamowienia date,
    data_wyslania date,
    koszt_wysylki decimal,
    id_klienta int NOT NULL,

```

```
id_faktury int NOT NULL  
);
```

Skrypty tworzące tabele dla bazy księgarnia (wykorzystane do tworzenia diagramu bazy danych)

```
CREATE TABLE Autor  
(  
    id_autora int IDENTITY (1, 1) NOT NULL PRIMARY KEY,  
    nazwisko varchar (50) NOT NULL,  
    imie varchar (30) NOT NULL,  
    narodowosc varchar (30),  
    okres_tworzenia varchar (35),  
    jezyk varchar (30),  
    rodzaj_tworczosci varchar (35),  
    osiagniecia varchar (200)  
);  
  
CREATE TABLE Ksiazki  
(  
    id_ksiazki int IDENTITY (1, 1) NOT NULL PRIMARY KEY,  
    tytul varchar (100) NOT NULL,  
    id_autora int REFERENCES Autor (id_autora),  
    cena money,  
    wydawnictwo varchar (20),  
    temat varchar (30),  
    miejsce_wydania varchar (28),  
    jezyk_ksiazki varchar (15),  
    opis varchar (100),  
    rok_wydania varchar (4)  
);  
  
CREATE TABLE Klient  
(  
    id_klienta int IDENTITY (1, 1) NOT NULL PRIMARY KEY,  
    nazwisko varchar (60) NOT NULL,
```

```
    imie varchar (40) NOT NULL,
    kod_pocztowy varchar (6),
    miejscowosc varchar (50) DEFAULT 'Warszawa',
    ulica varchar (50),
    nr_domu varchar (7),
    PESEL varchar (11) NOT NULL,
    telefon varchar (12) UNIQUE,
    adres_e_mail varchar (70)
);

CREATE TABLE Faktura
(
    nr_faktury int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    sposob_platnosci varchar (30),
    data_wystawienia_faktury date
);

CREATE TABLE Zamowienia
(
    id_zamowienia int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    data_zlozenia_zamowienia date,
    data_wyslania date,
    koszt_wysylki money,
    id_klienta int REFERENCES Klient (id_klienta),
    id_faktury int REFERENCES Faktura (nr_faktury)
);

CREATE TABLE Rejestracja_zamowienia
(
    id_zamowienia int REFERENCES Zamowienia (id_zamowienia),
    id_ksiazki int REFERENCES Ksiazki (id_ksiazki),
    liczba_egz int
);
```

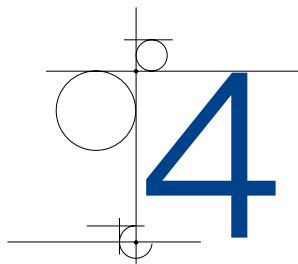
Widoki

Sprzedaż książek

```
SELECT dbo.klient.nazwisko AS K_Nazw, dbo.klient.imie AS K_Im, dbo.Ksiazki.
tytul, dbo.Autor.nazwisko AS A_Nazw, dbo.Autor.imie AS A_Im, dbo.Ksiazki.cena
FROM dbo.Klient INNER JOIN dbo.Zamowienia
ON dbo.Klient.id_klienta = dbo.Zamowienia.id_klienta
INNER JOIN dbo.Rejestracja_zamowienia
ON dbo.Zamowienia.id_zamowienia = dbo.Rejestracja_zamowienia.id_zamowienia
INNER JOIN dbo.Ksiazki
ON dbo.Rejestracja_zamowienia.id_ksiazki = dbo.Ksiazki.id_ksiazki
INNER JOIN dbo.Autor
ON dbo.Ksiazki.id_autora = dbo.Autor.id_autora
```

Książki i autorzy

```
SELECT dbo.Autor.nazwisko, dbo.Autor.imie, dbo.Ksiazki.tytul, dbo.Ksiazki.
cena
FROM dbo.Autor INNER JOIN
        dbo.Ksiazki ON dbo.Autor.id_autora = dbo.Ksiazki.id_autora
WHERE (dbo.Ksiazki.rok_wydania > 2010)
```



Administrowanie serwerami baz danych

4.1. Wprowadzenie

Serwer baz danych to program, który zarządza bazami danych. Jego zadaniem jest bezpieczne przechowywanie danych, kontrolowanie dostępu do danych, umożliwienie wielu użytkownikom jednoczesnego korzystania z przechowywanych danych oraz umożliwienie przetwarzania danych za pomocą języka SQL.

4.1.1. Zadania administratora baz danych

Podstawowym celem działań administratora baz danych powinno być zapewnienie niezawodności i bezawaryjnego działania systemu. Wiąże się to z zarządzaniem dostępem użytkowników do baz danych, zabezpieczeniem przed częściową lub całkowitą utratą danych oraz monitorowaniem pracy serwera baz danych.

Bezpieczeństwo danych to:

- uwierzytelnienie użytkownika,
- autoryzacja użytkownika,
- zarządzanie dostępem do baz danych,
- replikacja bazy danych,
- tworzenie kopii zapasowych.

Administrowanie bazami danych na poziomie użytkownika to:

- dodawanie nowego konta z różnymi uprawnieniami,
- usuwanie konta,

- wprowadzanie ograniczeń i nadawanie przywilejów dla konta,
- zarządzanie bazą danych przez definiowanie ról,
- dołączanie lub odłączanie baz danych,
- zakładanie baz danych,
- konserwacja i aktualizacja baz danych,
- monitorowanie bieżącego działania baz danych.

4.2. MS SQL Server

4.2.1. Środowisko SQL Server

Środowisko SQL Server składa się z wielu narzędzi, które można wybrać podczas instalacji serwera. Wybrane i zainstalowane składniki środowiska tworzą tzw. instancję. W ramach instancji można tworzyć wiele baz danych i nimi zarządzać. Fizycznie na serwerze można zainstalować wiele instancji SQL Server, chociaż najczęściej jest to jedna lub dwie instancje. Podstawową usługą SQL Server jest usługa serwera, która realizuje zadania związane z obsługą baz danych. Posiada mechanizmy zabezpieczeń, autoryzacji i autentykacji (uwierzytelnienia). Odpowiada za przetwarzanie zapytań, zarządza składowaniem i ochroną danych.

4.2.2. Tryby uwierzytelniania

Domyślnie tylko administratorzy komputera mają pełny dostęp do serwera SQL Server.

SQL Server obsługuje dwa tryby uwierzytelniania, tryb uwierzytelniania Windows (*Windows Authentication*) i tryb mieszany (*Mixed Mode*).

Tryb uwierzytelniania Windows akceptuje uwierzytelnienie systemu Windows i wyłącza uwierzytelnienie przez SQL Server. Logowanie w trybie *Windows Authentication* jest automatyczne. Nie trzeba wprowadzać nazwy użytkownika i hasła. Jeżeli użytkownik należy do grupy administratorów lokalnych komputera, zostanie mu przyznana automatycznie rola **sysadmin**. Uwierzytelnienia systemu Windows nie można wyłączyć.

Tryb mieszany akceptuje uwierzytelnienie systemu Windows i uwierzytelnienie przez SQL Server. Ten tryb umożliwia logowanie na specjalne konto SQL Server. Domyślnie zawsze istniejącym w bazie użytkownikiem jest sa (*System Administrator*), który należy do roli **sysadmin**.

Uwierzytelnienie za pomocą systemu Windows jest bezpieczniejsze od uwierzytelnienia SQL Server. Uwierzytelnienie to wykorzystuje specjalny protokół zabezpieczeń (*Kerberos*), który zapewnia egzekwowanie zasad haseł, wsparcie dla blokady kont i obsługuje wygaśnięcia hasła. Połączenie dokonywane za pomocą uwierzytelnienia systemu Windows jest też nazywane zaufanym połączeniem.

Podczas korzystania z uwierzytelnienia przez SQL Server (tryb mieszany) loginy są tworzone i przechowywane na serwerze SQL Server. Użytkownicy korzystający z takiego uwierzytelnienia muszą podawać swój login i hasło za każdym razem, gdy chcą się połączyć z bazą.

Rodzaj uwierzytelnienia dla użytkownika bazy danych zależy od konfiguracji systemu komputerowego.

4.2.3. Autoryzacja i uwierzytelnianie

Za bezpieczeństwo danych przechowywanych w bazach, jak również za bezpieczny dostęp do nich, odpowiadają autoryzacja i uwierzytelnianie (autentykacja).

Aby użytkownik mógł się zalogować i połączyć z bazą danych, musi mieć konto na serwerze. SQL Server umożliwia dostęp do bazy użytkownikom posiadającym lokalne lub domenowe konto w systemie Windows. W takiej sytuacji podczas logowania do systemu operacyjnego użytkownik jest identyfikowany, a podczas łączenia się z serwerem SQL Server uwierzytelnianie jest pomijane.

Drugą metodą logowania jest posiadanie loginu na serwerze SQL Server. W takim przypadku łączenie z instancją SQL Server wymaga użycia tego loginu.

Model zabezpieczeń w serwerze SQL Server opiera się na następującej zasadzie: tylko uwierzytelnieni użytkownicy mogą się połączyć z serwerem bazodanowym.

Rozpoczynając pracę z bazą danych, użytkownik musi podać swój login i wpisać odpowiednie hasło. Jest to mechanizm *uwierzytelniania użytkownika*. Ponadto, jeżeli użytkownik próbuje wykonać jakąkolwiek operację na bazie danych, serwer sprawdza, czy ma on do tego wystarczające uprawnienia. Zabronienie korzystania lub zezwolenie użytkownikowi na korzystanie z niektórych zasobów, czyli *autoryzacja użytkownika*, są przeprowadzane automatycznie.

Podczas instalowania serwera bazodanowego automatycznie tworzone są loginy dla grupy lokalnych administratorów systemu operacyjnego (wszyscy administratorzy systemu), dla konta użytkownika, na którym to koncie instalowany jest serwer, oraz gdy został wybrany tryb mieszany dla użytkownika sa (*System Administrator*).

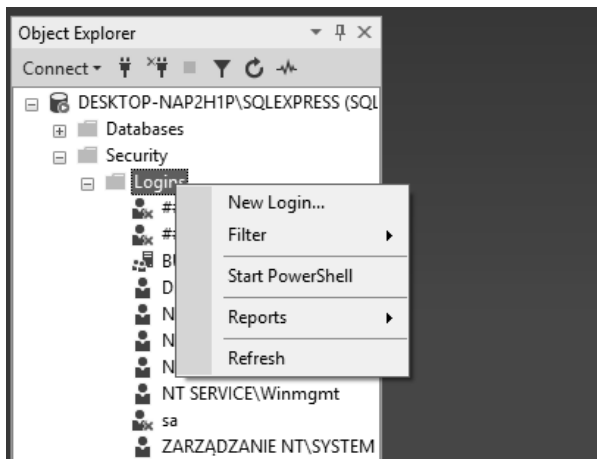
Wszyscy oni mają nieograniczone uprawnienia na serwerze bazodanowym. Pozostali użytkownicy serwera bazodanowego muszą mieć zdefiniowany login.

Oto instrukcja tworząca login na serwerze bazodanowym:

```
CREATE LOGIN [Komp\User1]
FROM WINDOWS;
```

W tak zapisanej instrukcji *Komp* to nazwa komputera, a *User1* to nazwa konta użytkownika.

Login można też utworzyć z poziomu SQL Server Management Studio (rysunek 4.1).



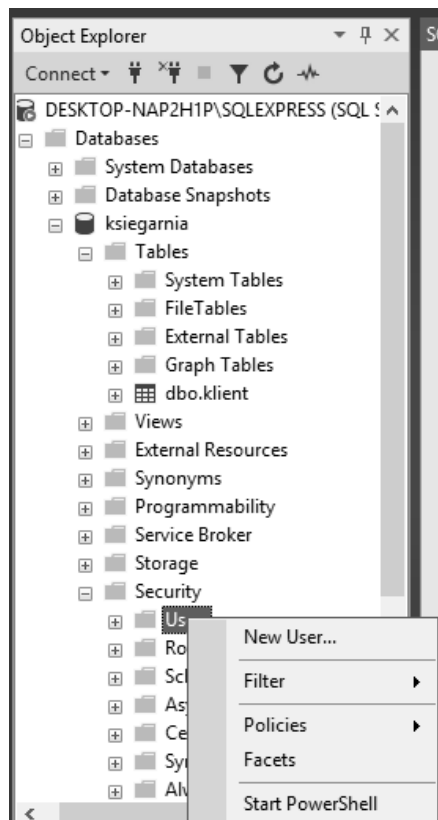
Rysunek 4.1. Tworzenie nowego loginu z poziomu SQL Server Management Studio

Po zdefiniowaniu loginu użytkownika ma on dostęp do serwera baz danych, ale nie ma uprawnień, aby uzyskać dostęp do bazy danych. Aby uzyskać dostęp do bazy danych, należy z poziomu wybranej bazy utworzyć użytkownika tej bazy i połączyć go z loginem. Oto instrukcja tworząca użytkownika bazy:

```
CREATE USER Janek
FOR LOGIN [Komp\User1]
WITH PASSWORD = 'hasło'
CHECK_POLICY = ON
```

Użytkownika bazy danych można utworzyć za pomocą SQL Server Management Studio. Należy pamiętać, aby tworzyć go z poziomu wybranej bazy danych (rysunek 4.2).

Login i user to dwa odrębne elementy systemu baz danych. Login zapewnia dostęp do instancji SQL Server, natomiast user to użytkownik, któremu można nadać uprawnienia do wybranych obiektów konkretnej bazy danych. Określony login może mieć przypisanych wielu użytkowników. Dzięki temu przy użyciu jednego konta logowania użytkownik może mieć zdefiniowane różne uprawnienia do różnych baz danych.



Rysunek 4.2.

Tworzenie użytkownika bazy danych w SQL Server Management Studio

Do tworzonego loginu możemy przypisać domyślną bazę danych, z którą użytkownik będzie mógł automatycznie się połączyć:

```
CREATE LOGIN [Komp\User1]
FROM WINDOWS
WITH DEFAULT_DATABASE=Nazwa_bazy;
GO
```

Po utworzeniu loginu powiązanego z kontem wystarczy, że użytkownik zaloguje się do systemu, aby miał dostęp do bazy.

Z poziomu serwera bazodanowego można też utworzyć konto użytkownika w systemie Windows, wykonując instrukcję `CREATE USER`.

Przykład 4.1

```
USE master;
CREATE USER Pracownik1
FOR LOGIN [Komp\Pracownik1];
```

Jeden login może zostać powiązany z wieloma dostępnymi na serwerze bazami danych.

4.2.4. Zarządzanie bazami danych

SQL Server przechowuje informacje o istniejących bazach użytkowników w specjalnej bazie systemowej *master*. Z poziomu tej bazy jest realizowane zarządzanie pozostałymi bazami danych. Aby połączyć się z dowolną bazą, należy wykonać instrukcję `USE`. Połączenie z bazą *master* uzyskamy, wpisując:

```
USE master;
```

Do połączenia z bazą *master* wymagane są uprawnienia administratora.

Systemowe bazy danych

Podczas instalowania serwera SQL Server tworzone są systemowe bazy danych, którymi można zarządzać za pomocą narzędzi serwera.

master

Baza *master* jest najważniejszą bazą danych w systemie. Zawiera podstawowe dane dla serwera oraz informacje o wszystkich utworzonych w systemie bazach danych. Ze względu na bezpieczeństwo powinno się regularnie tworzyć jej kopie zapasowe.

model

Baza *model* jest szablonem bazy danych. W momencie tworzenia nowej bazy danych jest tworzona kopia bazy *model*, a następnie kopia ta jest dostosowywana do wymagań tworzonej bazy. Jeżeli baza *model* zostanie zmodyfikowana, na przykład przez dodanie jakiegoś obiektu, to wszystkie tworzone nowe bazy danych będą zawierały ten obiekt.

tempdb

Baza *tempdb* jest przeznaczona do wykonywania operacji, które wymagają tymczasowej przestrzeni. Jest wykorzystywana przy operacjach typu sortowanie, złączenie itp.

msdb

Baza *msdb* przechowuje informacje na temat replikacji, zdarzeń, zadań (na przykład tworzenia kopii zapasowych, odzyskiwania danych).

Tworzenie bazy danych

Do tworzenia bazy danych służy polecenie `CREATE DATABASE`. W trakcie tworzenia bazy danych można określić takie parametry jak: rozmiar bazy, jej nazwa logiczna oraz lokalizacja pliku z danymi.

Przykład 4.2

```
CREATE DATABASE Towary ON PRIMARY
(NAME= Sprzedaz_towarow,
FILENAME = "C:\Program Files\Microsoft SQL Server\MSSQL\DATA\Towary.mdf",
SIZE = 10MB,
MAXSIZE = 30MB,
FILEGROWTH = 5MB)
```

Została utworzona baza danych *Towary*. Dla powstałej bazy określono nazwę logiczną, lokalizację pliku z danymi i rozmiar bazy.

Zmiana parametrów bazy danych

Parametry bazy danych, takie jak rozmiar lub nazwa, można zmienić poleceniem `ALTER DATABASE`.

Przykład 4.3

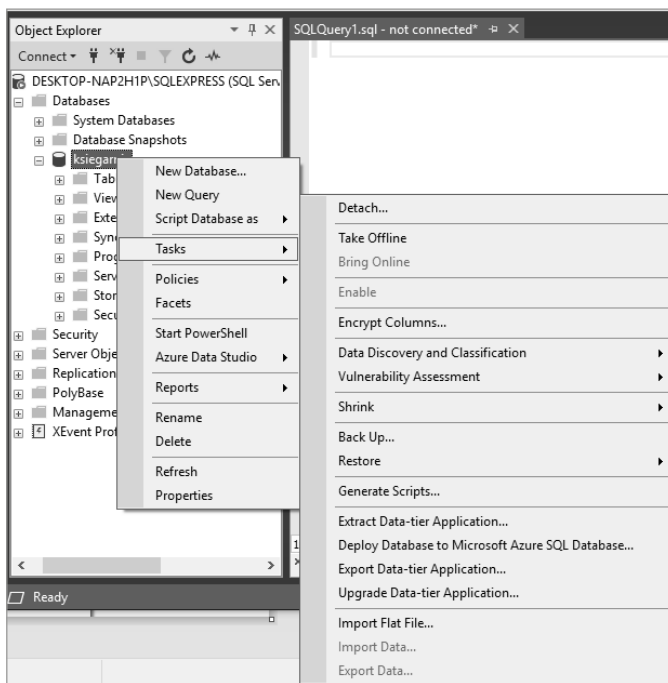
```
ALTER DATABASE Towary
ADD FILE
(NAME = Sprzedaz_towarow1,
FILENAME = "C:\Program Files\Microsoft SQL Server\MSSQL\DATA\Towary1.mdf",
SIZE = 5,
MAXSIZE = 15,
FILEGROWTH = 3)
```

Poleceniem `ALTER DATABASE` można do istniejącej bazy danych dodać dziennik transakcji.

Przykład 4.4

```
ALTER DATABASE Towary
ADD LOG FILE
(NAME = Sprzedaz_Dziennik,
FILENAME = "C:\Program Files\Microsoft SQL Server\MSSQL\DATA\Towary.mdf",
SIZE = 2,
MAXSIZE = 6,
FILEGROWTH = 1)
```

W SQL Server Management Studio w menu kontekstowym wybranej bazy danych znajduje się opcja *Tasks* (rysunek 4.3) zawierająca polecenia, które pomogą w administrowaniu bazą danych.



Rysunek 4.3. Narzędzia służące do zarządzania bazą danych

Detach... służy do odłączania bazy danych od danej instancji serwera. Jest to narzędzie przydatne, gdy zachodzi potrzeba przeniesienia bazy danych z jednego serwera na inny.

Shrink służy do zmniejszenia rozmiarów bazy danych. Przy jego użyciu można zmniejszyć rozmiary albo pojedynczych plików używanych przez bazę, albo całej bazy. Zmniejszanie rozmiaru bazy danych ma na celu zwolnienie miejsca niewykorzystywanego przez bazę.

Back Up... umożliwia zapisanie bazy danych na dysku lokalnym serwera.

Restore służy do przywrócenia bazy danych na serwerze SQL z pliku kopii zapasowej.

Generate Scripts... to polecenie, które po uruchomieniu kreatora pozwala na wygenerowanie skryptów bazy danych. W zależności od wybranych opcji można określić, jakie elementy bazy danych powinny znaleźć się w skrypcie.

Poza grupą *Tasks* w menu kontekstowym bazy danych znajdują się następujące opcje:

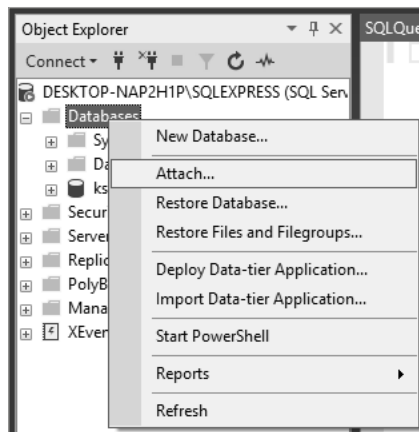
Rename — służy do zmiany nazwy bazy danych;

Delete — powoduje usunięcie bazy danych;

Refresh — odświeża wyświetlane w drzewie obiekty bazy;

Properties — powoduje wyświetlenie okna właściwości bazy danych.

Odłączoną bazę danych można dołączyć do wybranej instancji poleceniem *Attach...* (rysunek 4.4).



Rysunek 4.4.
Narzędzie dołączania bazy danych

4.3. Prawa dostępu do serwera

Serwery bazodanowe mają mechanizmy, które nie pozwalają korzystać z ich zasobów niezidentyfikowanym użytkownikom.

MS SQL Server w chwili tworzenia bazy danych wraz z nią tworzy konta domyślnych użytkowników oraz przypisuje im domyślne role i uprawnienia. Aby lepiej zabezpieczyć bazę danych, administrator może zmienić domyślne ustawienia serwera baz danych.

Tylko uwierzytelnieni użytkownicy mają dostęp do wybranych baz danych. Kolejny poziom zabezpieczeń tworzą uprawnienia nadawane poszczególnym użytkownikom.

4.3.1. Role

Utworzone dla użytkownika konta umożliwiają połączenie się z bazą danych, ale nie pozwalają na odczytanie danych czy ich zmodyfikowanie. Użytkownik domyślnie nie ma żadnych uprawnień. Wynika to z zasad bezpieczeństwa stosowanych przez serwery bazodanowe. Aby użytkownik mógł wykonywać jakiejkolwiek operacje w bazie danych, musi mieć nadane odpowiednie uprawnienia.

Podczas nadawania uprawnień powinno się stosować zasadę minimalnych uprawnień. Nie należy nadawać żadnych uprawnień poza minimalnymi, niezbędnymi do prawidłowej pracy z bazą. Uprawnienia można nadawać użytkownikom albo grupom użytkowników. SQL Server umożliwia zarządzanie uprawnieniami poprzez tworzenie ról, czyli grupowanie użytkowników według potrzeb i przypisywanie im uprawnień.

W serwerze SQL Server wyróżniamy trzy rodzaje ról:

- serwerowe,
- bazodanowe,
- definiowane przez użytkownika.

Role pozwalają grupować użytkowników, którzy powinni mieć takie same prawa dostępu do obiektów bazy danych. Role nadawane na poziomie instancji serwera służą do nadawania uprawnień operatorom serwera. Role bazodanowe służą do nadawania użytkownikom uprawnień dostępu do bazy danych. Role definiowane przez użytkownika to uprawnienia do wybranych obiektów bazy lub uprawnienia mieszane, na przykład prawo tylko do odczytu w jednej tabeli, a w innej do odczytu i do zapisu.

W serwerze SQL Server istnieją predefiniowane role serwerowe i role bazodanowe.

Role serwerowe

Są przeznaczone do administrowania serwerem. Mają stały zestaw uprawnień dotyczących całego serwera. Nie można zmienić uprawnień przypisanych do tych ról.

`sysadmin` — użytkownicy przypisani do tej roli mają prawo wykonywać każdą operację na serwerze i są właścicielami każdej bazy danych. Nie można im zabronić wykonania żadnej operacji. Rola ta jest przydzielona użytkownikowi `sa` i nie można mu jej odebrać.

`serveradmin` — użytkownicy przypisani do tej roli to administratorzy serwera. Nie administrują oni bazami danych, ale mają prawo do konfigurowania systemu i do zarządzania serwerem.

`setupadmin` — ta rola jest przypisywana administratorom, którzy konfiguruje system i nim zarządzają.

`securityadmin` — ta rola jest przypisywana administratorom, którzy zarządzają bezpieczeństwem systemu (tworzą konta, przyznają prawo tworzenia baz danych, wykonują procedury związane z bezpieczeństwem).

`processadmin` — ta rola jest przypisywana administratorom, którzy kontrolują procesy uruchomione na serwerze, na przykład kasowanie działających zapytań.

`dbcreator` — użytkownicy przypisani do tej roli mogą tworzyć, modyfikować, usuwać i odzyskiwać bazy danych.

`diskadmin` — użytkownicy przypisani do tej roli zarządzają plikami.

`bulkadmin` — użytkownicy przypisani do tej roli mogą uruchamiać polecenie `BULK INSERT` (operację masowego wstawiania danych).

Role bazodanowe

Role bazodanowe to uprawnienia przyznawane użytkownikowi do określonej bazy danych.

`db_owner` — rola przypisywana właścicielowi bazy danych. Użytkownicy przypisani do tej roli mogą wykonywać operacje DDL (wyjątkiem są polecenia `GRANT`, `DENY` i `REVOKE`).

`db_accessadmin` — użytkownicy przypisani do tej roli decydują, które konta logowania mają prawa dostępu do bazy danych.

Istnieją także inne role, ale ich opis wykracza poza zakres niniejszego podręcznika.

Właściciel bazy danych należy do roli `db_owner` i ma do niej wszystkie prawa. Każda baza danych ma tylko jednego właściciela (`dbo`).

Właścicielem obiektu bazy danych (`dbo`) jest użytkownik, który utworzył ten obiekt. Użytkownicy należący do ról `db_owner` lub `db_ddladmin` mogą tworzyć obiekty pod swoją nazwą lub jako `dbo`.

Tworząc konto logowania (`login`) i użytkownika (`user`), który będzie wykorzystywany podczas pracy z bazą danych, należy przypisać użytkownikowi odpowiednie role serwerowe i bazodanowe.

Tworzenie i usuwanie ról

Rola w bazie danych jest tworzona za pomocą polecenia `CREATE ROLE`.

Przykład 4.5

```
CREATE ROLE Blok1;
CREATE ROLE Pracownik;
CREATE ROLE Operator;
```

Istniejącą rolę można usunąć, wpisując polecenie `DROP ROLE`.

```
DROP ROLE Pracownik;
```

Można usuwać tylko role użytkowników. Natomiast nie mogą być usuwane wbudowane role serwera i bazy danych.

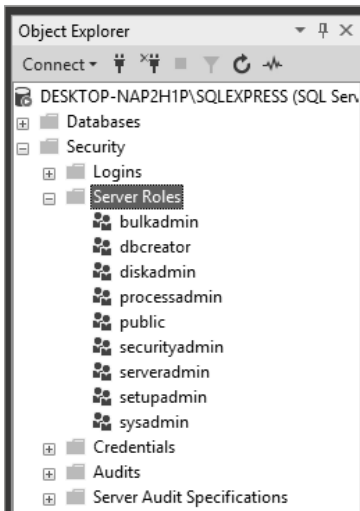
Dostęp do ról serwerowych i bazodanowych z poziomu SQL Server Management Studio został pokazany na rysunkach 4.5 i 4.6.

Przypisywanie do ról

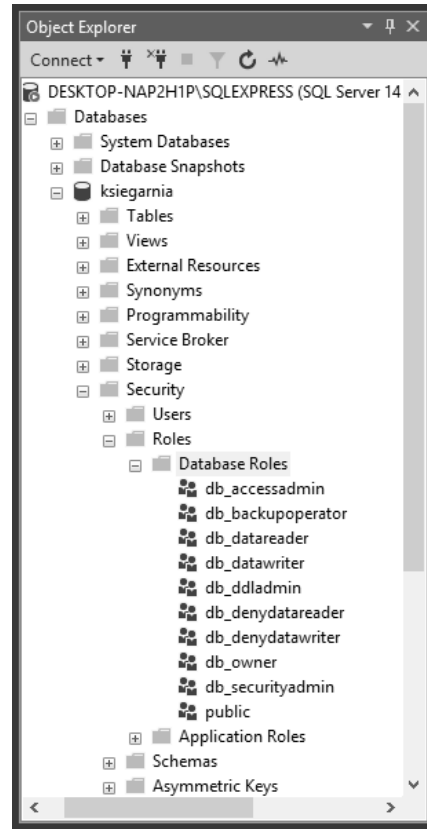
Do utworzonych ról można przypisać dowolną liczbę użytkowników. Użytkownicy są przypisywani do ról poleceniem `GRANT ... TO ...`.

Przykład 4.6

```
GRANT Blok1 TO Maciej;
GRANT Blok1 TO Michał;
GRANT Blok1 TO Marcin;
```

**Rysunek 4.5.**

Role serwerowe dostępne z poziomu SQL Server Management Studio

**Rysunek 4.6.**

Role bazodanowe dostępne z poziomu SQL Server Management Studio

Ten sam użytkownik może zostać przypisany do różnych ról.

```
GRANT Operator TO Michał;
```

Każdy z użytkowników jest automatycznie przypisany do roli PUBLIC. Jest to specjalna grupa, z której nie można usunąć żadnego użytkownika. Pozwala ona nadać lub odebrać uprawnienia wszystkim użytkownikom bazy danych.

4.3.2. Uprawnienia

Uprawnienia nadawane użytkownikom dzielimy na dwie kategorie.

1. Uprawnienia do wykonania określonych instrukcji. Są to:

- uprawnienia do modyfikowania ról (ALTER ANY ROLE),
- uprawnienia do modyfikowania kont użytkowników (ALTER ANY USER),

- uprawnienia do wykonywania kopii zapasowych bazy danych (BACKUP DATABASE),
 - uprawnienia do tworzenia funkcji (CREATE FUNCTION),
 - uprawnienia do tworzenia procedur (CREATE PROCEDURE),
 - uprawnienia do tworzenia schematów (CREATE SCHEMA),
 - uprawnienia do tworzenia tabel (CREATE TABLE),
 - uprawnienia do przejmowania obiektów na własność (TAKE OWNERSHIP),
 - uprawnienia do odczytywania metadanych obiektów (VIEW DEFINITION).
- 2.** Uprawnienia obiektowe — użytkownicy mogą odwoływać się do wybranych obiektów bazy danych i wykonywać określone operacje. Możliwe do nadania uprawnienia zależą od typu obiektu:
- Do tabel i widoków można nadawać uprawnienia SELECT, INSERT, UPDATE, DELETE, REFERENCE. Uprawnienia te pozwalają na pobieranie, wstawianie, modyfikowanie i usuwanie danych oraz na tworzenie kluczy obcych.
 - Do kolumn można nadawać uprawnienia SELECT i UPDATE.
 - Do procedur składowanych i funkcji można nadać uprawnienie EXECUTE.
 - Do schematów można nadać wszystkie wymienione wyżej uprawnienia.

Nadawanie uprawnień — instrukcja GRANT

Uprawnienia do obiektów bazy danych nadawane są rolom lub użytkownikom instrukcją GRANT.

```
GRANT {uprawnienia}
ON Obiekt
TO Użytkownicy/Role
```

Przykład 4.7

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON dbo.Ksiazki
TO Blok1;
```

Zamiast nadawać uprawnienia poszczególnym użytkownikom, należy nadawać je rolom. W podanym przykładzie wszyscy członkowie roli *Blok1* mogą odczytywać i modyfikować dane zapisane w tabeli *Ksiazki*.

Odbieranie uprawnień — instrukcja REVOKE

Instrukcją REVOKE można usunąć wcześniej nadane lub odebrane uprawnienia do obiektu.

Przykład 4.8

```
REVOKE INSERT, UPDATE
ON dbo.Ksiazki
FROM Michał;
```

W podanym przykładzie użytkownikowi *Michał* zostały odebrane uprawnienia do wstawiania i modyfikowania danych w tabeli *Ksiazki*. Ale *Michał* należy do roli *Blok1*, która posiada odebrane mu uprawnienia. Z tego powodu będzie on nadal mógł wstawiać i modyfikować dane w tabeli *Ksiazki*.

Instrukcja DENY

Jawne odebranie użytkownikowi uprawnień dostępu do obiektów bazy danych nastąpi po użyciu instrukcji DENY.

```
DENY {Uprawnienia}
ON Obiekt
TO Uzytkownicy/Role
```

Przykład 4.9

```
DENY INSERT, DELETE
ON Ksiazki
TO Michał;
```

Po wykonaniu tej operacji użytkownik *Michał*, mimo przynależności do ról, nie będzie mógł dodać wiersza do tabeli *Ksiazki* ani go z niej usunąć.

Dziedziczenie uprawnień

W serwerach bazodanowych funkcjonuje mechanizm dziedziczenia uprawnień. Obiekty bazy danych tworzą hierarchię, zgodnie z którą działają mechanizmy dziedziczenia.

Najwyżej w hierarchii znajdują się obiekty serwera bazodanowego (bazy danych, loginy), poniżej są obiekty bazy danych (konta użytkowników, role, schematy), jeszcze niżej — obiekty schematów (tabele, widoki, procedury, funkcje), a na samym dole hierarchii są obiekty tabel, czyli kolumny.

Uprawnienia nadane do obiektu znajdującego się wyżej w hierarchii są domyślnie dziedziczone przez obiekty położone niżej. Jednak uprawnienia mogą być nadawane i odbierane na dowolnym poziomie.

Przykład 4.10

```
REVOKE UPDATE
ON dbo.Ksiazki (tytul)
FROM Maciej;
```

W podanym przykładzie użytkownik *Maciej* może aktualizować wszystkie kolumny tabeli *Ksiazki* oprócz kolumny *tytul*.

Właściciel obiektu

Każdy obiekt zdefiniowany w bazie danych ma swojego właściciela. Domyślnie właścicielem obiektu jest użytkownik, który go utworzył. Ma on nieograniczone uprawnienia do obiektu. Możliwe jest jawne określenie innego właściciela obiektu, jeśli podczas definiowania obiektu jego nazwę poprzedzimy nazwą nowego właściciela. Możliwa jest również zmiana właściciela istniejącego obiektu. Właściciel obiektu może udzielać uprawnień do tego obiektu.

Zaleca się, aby właścicielem wszystkich obiektów bazy danych był ten sam użytkownik.

Przykład 4.11

```
USE ksiegarnia GRANT CREATE VIEW TO Blok1;

GO

CREATE VIEW Marcin.widok1 AS

SELECT Autor.nazwisko, Autor.imie, Ksiazki.tytul, Ksiazki.cena

FROM Autor INNER JOIN Ksiazki

ON Autor.id_autora=Ksiazki.id_autora

WHERE Ksiazki.rok_wydania>2017;

GO
```

W podanym przykładzie pierwsze polecenie nadaje prawo tworzenia widoków w bazie *ksiegarnia* użytkownikom należącym do roli *Blok1*. Następne polecenie tworzy widok, którego właścicielem zostaje użytkownik *Marcin*.

Podsumowanie

Jedną z podstawowych zasad bezpieczeństwa danych jest nadawanie minimalnych uprawnień. Oznacza to, że należy nadawać użytkownikowi tylko te uprawnienia, które są niezbędne do wykonywania jego obowiązków.

Jeżeli chcemy ograniczyć użytkownikom dostęp do poszczególnych kolumn w tabelach, można nadawać uprawnienia do tych kolumn, ale lepszym rozwiązaniem jest tworzenie widoków i nadawanie odpowiednich uprawnień do widoków.

4.4. Kopie bezpieczeństwa

Jednym z podstawowych działań administratora baz danych jest zabezpieczenie przed utratą danych. Jedynie posiadanie kopii bezpieczeństwa bazy danych gwarantuje odzyskanie utraconych danych.

4.4.1. Tworzenie kopii bezpieczeństwa

Strategię wykonywania kopii bezpieczeństwa wypracowuje administrator serwera. Niezależnie od przyjętej strategii kopia bezpieczeństwa bazy danych powinna być wykonana zawsze, gdy:

- utworzono lub zmodyfikowano strukturę bazy danych,
- utworzono indeks,
- usunięto nieaktywną część dziennika.

Kopie bezpieczeństwa bazy danych mogą tworzyć administrator serwera i właściciel bazy danych. SQL Server umożliwia wykonanie kopii bezpieczeństwa całej bazy danych, wybranych plików (lub grup plików) bazy danych lub kopii dziennika transakcyjnego. Kopię bezpieczeństwa każdego typu można utworzyć, wykonując instrukcję `BACKUP`.

Instrukcją `BACKUP LOG` można wykonać kopię dziennika transakcyjnego wybranej bazy danych.

Pełna kopia bazy danych

Aby zachować wszystkie elementy bazy danych niezbędne do jej odtworzenia, należy wykonywać pełną kopię bazy danych. Przechowuje ona wszystkie informacje zapisane zarówno w plikach bazy danych (strukturę obiektów bazodanowych oraz dane tabel i indeksów), jak i w plikach dziennika transakcyjnego oraz wszystkie dane z aktywnej części dziennika.

Pełna kopia bazy danych jest wymagana do odtworzenia kopii przyrostowej lub kopii dziennika transakcyjnego bazy danych.

Przykład 4.12

```
BACKUP DATABASE ksiegarnia
TO DISK = 'E:\SQLServerBackups\Ksiegarnia_kopia.bak'
WITH FORMAT;
```

W podanym przykładzie została utworzona pełna kopia bazy danych *ksiegarnia*. Kopia ta została zapisana na dysku *E:* w pliku *Ksiegarnia_kopia*.

Przyrostowa kopia bazy danych

Tworzenie przyrostowych kopii baz danych pozwala skrócić czas potrzebny na odtworzenie bazy z plików kopii dziennika transakcyjnego oraz czas potrzebny na wykonanie kopii plików bazodanowych. Do pliku przyrostowej kopii bazy danych zostaną zapisane wszystkie dane, które zostały zmodyfikowane od momentu wykonania ostatniej pełnej kopii bazy danych.

Przykład 4.13

```
BACKUP DATABASE ksiegarnia
TO Ksiegarnia_kopia_przyrost WITH DIFFERENTIAL
```

W podanym przykładzie została utworzona przyrostowa kopia bazy danych *ksiegarnia*. Będzie ona zawierała tylko zapis zmian, które nastąpiły od momentu wykonania pełnej kopii bazy.

W przypadku bardzo dużych baz danych codzienne wykonywanie kopii całej bazy może być trudne. W tym przypadku, zamiast kopii zapasowej całej bazy, można utworzyć kopię wybranego pliku (lub grupy plików).

Przykład 4.14

```
BACKUP DATABASE ksiegarnia
FILE = 'Dane', FILEGROUP = 'Grupa1' TO Kopia_ksiegarnia
```

W podanym przykładzie została wykonana kopia zapasowa tylko wybranych plików bazy danych *ksiegarnia*.

4.4.2. Sprawdzanie spójności bazy danych

Przed wykonaniem kopii bezpieczeństwa należy sprawdzić spójność bazy danych. Utworzenie kopii niespójnej bazy danych może uniemożliwić późniejsze jej odtworzenie. Spójność bazy można sprawdzić za pomocą polecenia `DBCC CHECKDB`. Polecenie `DBCC CHECKCATALOG` sprawdza integralność referencyjną tabel systemowych, polecenie `DBCC CHECKALLOC` — spójność alokowania struktur na dysku twardym, a polecenie `DBCC CHECKTABLE` — spójność na poziomie jednej tabeli.

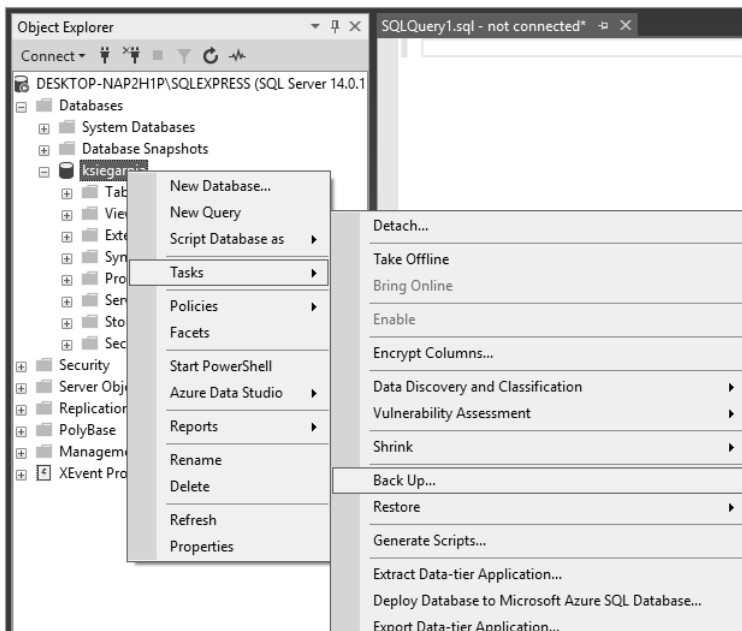
Polecenie `DBCC CHECKDB` ma następujące opcje:

- `Repair_Allow_Data_Loss` — naprawia zgłoszone błędy. Uszkodzone dane są czyszczone, nawet jeśli spowoduje to ich utratę.
- `Repair_Fast` — opcja dla zachowania kompatybilności wstecznej, nie wykonuje żadnej naprawy.
- `Repair_Rebuild` — naprawia uszkodzone indeksy. Wykonuje szybkie naprawy (naprawianie zgubionych wierszy) lub dłuższe naprawy (odbudowanie indeksu).

Przykład 4.15

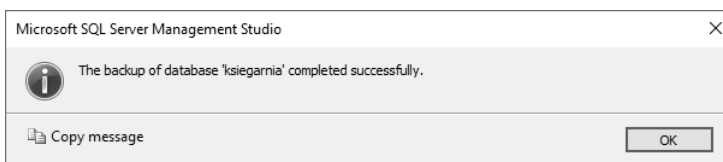
```
DBCC CHECKDB ('ksiegarnia', Repair_fast)
```

Kopię zapasową w programie SQL Server Management Studio wykonuje się, wybierając w oknie *Object Explorer* określoną bazę. Następnie klikamy bazę prawym przyciskiem myszy i wybieramy opcję *Tasks/Back Up...* (rysunek 4.7).



Rysunek 4.7. Wykonanie kopii zapasowej bazy danych

W otwartym oknie należy sprawdzić ustawienia dotyczące bazy danych. Można zmienić miejsce docelowe oraz nazwę pliku z kopią bazy. Po kliknięciu **OK** zostanie utworzona kopia zapasowa bazy (rysunek 4.8).



Rysunek 4.8. Potwierdzenie utworzenia kopii zapasowej

4.4.3. Przywracanie bazy danych z kopii bezpieczeństwa

Przywracanie bazy danych z kopii bezpieczeństwa odbywa się przy użyciu polecenia `RESTORE DATABASE`. Istnieje kilka metod przywracania bazy:

- przywracanie całej bazy danych z kopii zapasowej (pełne przywracanie),
- przywracanie części bazy danych (częściowe przywracanie),
- przywracanie wybranych plików lub grup plików do bazy danych (plik przywracania),
- przywracanie dziennika transakcji do bazy danych (plik dziennika transakcji).

Przykład 4.16

Przywracanie pełnej bazy danych:

```
RESTORE DATABASE ksiegarnia
FROM Ksiegarnia_kopia.bak;
```

Przykład 4.17

Nadpisywanie istniejącej bazy danych:

```
RESTORE DATABASE ksiegarnia
FROM Ksiegarnia_kopia.bak WITH replace;
```



4.5. Import i eksport danych

Podczas instalowania SQL Server Management Studio instalowany jest program DTS (*Data Transformation Service*), który może być bardzo przydatnym narzędziem umożliwiającym import danych do serwera SQL Server lub eksport danych z niego.

Źródłem danych dla serwera SQL Server mogą być dane z: *.NET Framework*, *OLE DB*, *SQL Server Native Client*, *ADO.NET*, *MS Excel*, *MS Access*. W zależności od pochodzenia danych można ustawić opcje określające tryb uwierzytelnienia, nazwę serwera, nazwę bazy danych, nazwę pliku. Dane mogą zostać wyeksportowane do: *.NET Framework*, *OLE DB*, *SQL Server Native Client*, *ADO.NET*, *MS Excel*, *MS Access*.

Aby uruchomić program DTS, należy z menu *Start* systemu Windows wybrać *Microsoft SQL Server 2017/ SQL Server 2017 Import and Export Data*. Zostanie uruchomiony kreator importu i eksportu danych.

4.5.1. Eksport danych

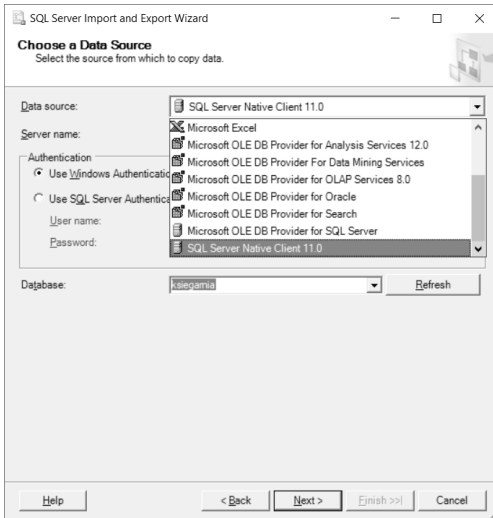
Po zaznaczeniu w oknie nawigacyjnym bazy, która będzie eksportowana, i wybraniu opcji *Tasks/Export Data...* w pierwszym oknie kreatora należy wybrać typ źródła eksportowanych danych, *SQL Server Native Client 11.0* (rysunek 4.9), oraz bazę danych, a w kolejnym miejscu docelowe danych, *Flat File Destination* (rysunek 4.10), oraz podać nazwę pliku, do którego zostaną wyeksportowane dane z bazy.

W kolejnym oknie importowania plików można określić, czy będą przenoszone dane z tabel lub widoków, czy wyniki zapytania (rysunek 4.11).

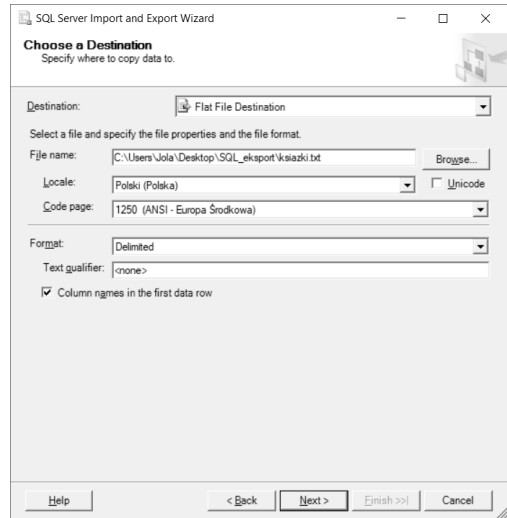
Następne okno zawiera listę tabel, z których należy wybrać tę, z której dane zostaną wyeksportowane. Po wykonaniu tych czynności dane zostaną zapisane w pliku docelowym.

4.5.2. Import danych

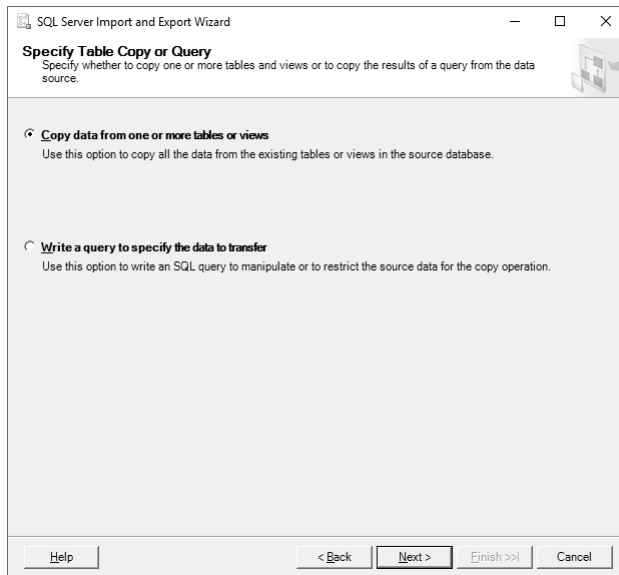
W ten sam sposób za pomocą kreatora importu i eksportu danych można wykonać operację importu danych, określając jako źródło danych odpowiednio przygotowany plik tekstowy, a jako miejsce docelowe bazę SQL Server.



Rysunek 4.9.
Wybór źródła importowanych danych



Rysunek 4.10.
Wybór miejsca docelowego importowanych danych



Rysunek 4.11. Wybór rodzaju importowanych danych

Polecenie BULK INSERT

Inną metodą importu danych z pliku tekstowego do istniejącej tabeli w bazie danych jest wykorzystanie polecenia `BULK INSERT`. Polecenie to jest rozszerzeniem języka T-SQL i może służyć do utworzenia skryptu importu danych z plików tekstowych do bazy.

Przykład 4.18

```

USE ksiegarnia

CREATE TABLE Ksiazki

(
    id_ksiazki int IDENTITY (1, 1) NOT NULL PRIMARY KEY,
    tytul varchar (100) NOT NULL,
    id_autora int REFERENCES Autor (id_autora),
    cena money,
    wydawnictwo varchar (20),
    temat varchar (30),
    miejsce_wydania varchar (28),
    jezyk_ksiazki varchar (15),
    opis varchar (100),
    rok_wydania varchar (4)
);

GO

BULK INSERT ksiazki
FROM 'C:\Dane\ksiazki.txt'
WITH
{
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n',
    CODEPAGE = 'ACP'
}

GO

```

gdzie:

FIELDTERMINATOR — to symbol znacznika oddzielającego kolejne wartości w wierszu,

ROWTERMINATOR — to symbol końca wiersza.

Opcjonalnie w bloku WITH może wystąpić parametr CODEPAGE w postaci:

```
CODEPAGE = 'strona_kodowa'
```

który określa stronę kodową znaków, np. ACP oznacza kodowanie ANSI. Polecenie BULK nie obsługuje (dla wersji starszych niż 2017) kodowania UTF-8.

4.6. MySQL

4.6.1. Konfigurowanie serwera

Konfigurowanie serwera MySQL przez administratora można przeprowadzić za pomocą wiersza poleceń lub odpowiednich narzędzi, na przykład programu phpMyAdmin. Narzędzie to jest dostępne zarówno na platformie Windows, jak i Linux.

4.6.2. Tryby uwierzytelnienia

Aby połączyć się z serwerem, użytkownik musi potwierdzić swoją tożsamość, podając login i hasło. Jest to proces *uwierzytelnienia użytkownika*. Jeżeli użytkownik zalogowany do bazy danych próbuje wykonać jakąkolwiek operację (na przykład odczytanie lub modyfikowanie danych), serwer sprawdza, czy ma on wystarczające uprawnienia do wykonania tej operacji. Jeżeli użytkownik nie ma wymaganych uprawnień, nastąpi przerwanie wykonywania operacji. Ten proces nazywany jest *autoryzacją*.

Podczas instalowania serwera MySQL tworzone jest wbudowane konto *root*. Ma ono pełne uprawnienia i w codziennej pracy nie powinno być używane nawet przez administratora. Korzystając z konta *root*, wyłączamy cały mechanizm autoryzacji i obniżamy poziom bezpieczeństwa serwera. Założenie konta użytkownika i nadanie użytkownikowi minimalnych uprawnień potrzebnych do wykonywania jego pracy powinno być podstawową zasadą bezpieczeństwa podczas korzystania z serwera baz danych.

Serwer MySQL powinien być uruchamiany z konta zwykłego użytkownika, specjalnie w tym celu utworzonego, z minimalnymi uprawnieniami, które są potrzebne do pracy. Aby serwer uruchamiał się z tego konta, w pliku *my.cnf* należy dodać wpis:

```
[mysqld]
user=mysql
```

W serwerze MySQL konta użytkownika związane są z nazwą komputera, z którego użytkownik korzysta. Jeżeli użytkownik łączy się z serwerem z wielu komputerów, to dla każdego z nich możliwe jest ustawienie innego hasła i innych praw dostępu.

Do utworzenia użytkownika służy polecenie `CREATE USER`.

Przykład 4.19

```
CREATE USER adam IDENTIFIED BY 'haslo';
CREATE USER adam IDENTIFIED BY PASSWORD 'tajne_haslo';
```

W wyniku wykonania polecenia w tabeli *mysql.user* zostanie utworzony wpis oznaczający, że użytkownik *adam* może korzystać z dowolnego komputera do połączenia się z serwerem.

Aby usunąć użytkownika, należy użyć polecenia `DROP USER`.

Przykład 4.20

```
DROP USER adam;
```

Nazwę bieżącego użytkownika można uzyskać po wpisaniu polecenia:

```
SELECT CURRENT_USER();
```

Zmianę hasła bieżącego użytkownika uzyskamy po wpisaniu polecenia:

```
SET PASSWORD = PASSWORD('haslo');
```

Zmiana hasła dla innego konta wymaga odpowiednich uprawnień i ma postać:

```
SET PASSWORD FOR adam = PASSWORD('haslo_adama');
```

Zadanie 4.1

Skonfiguruj zainstalowany w wybranym przez siebie systemie operacyjnym serwer MySQL. Utwórz trzech nowych użytkowników: *Operator* z hasłem, *Instruktor* z hasłem oraz *Gracz* bez hasła.

4.6.3. Zarządzanie bazami danych

Podczas instalowania serwera MySQL automatycznie tworzone są dwie bazy systemowe: *information_schema* oraz *mysql*. Baza *information_schema* zawiera metadane na temat pozostałych baz danych. Baza *mysql* zawiera między innymi informacje o użytkownikach serwera i ich uprawnieniach.

Tworzenie bazy danych

Do tworzenia bazy danych służy polecenie `CREATE DATABASE`, do zmiany parametrów bazy — polecenie `ALTER DATABASE`, zaś do usunięcia bazy — polecenie `DROP DATABASE`.

Przykład 4.21

```
CREATE DATABASE Towary;
```

```
ALTER DATABASE Towary CHARACTER SET latin2;
```

```
DROP DATABASE Towary;
```

Informację o bazie danych można wyświetlić, używając polecenia `SHOW DATABASES`. Informację o tabelach w aktualnej bazie danych wyświetlimy, wpisując polecenie `SHOW TABLES`. Informacje o wybranej tabeli w aktualnej bazie danych można wyświetlić za pomocą polecenia `DESCRIBE Nazwa_tabeli`.

4.6.4. Typy tabel w MySQL

W serwerze MySQL występuje wiele mechanizmów obsługi tabel. Każdy z nich ma inne zastosowanie. Przy użyciu polecenia `CREATE DATABASE` i jego klauzul `ENGINE`

lub `TYPE` można zdefiniować mechanizm przechowywania danych, który zostanie zastosowany w definiowanej tabeli.

MyISAM

Typ tabeli używany domyślnie w MySQL do wersji 5.5.4. Tego typu tabele pozwalają na szybkie odczytywanie danych, ale nie obsługują transakcji. Powinny być wykorzystywane do przechowywania rzadko zmienianych danych.

MEMORY

Tabele tego typu są przechowywane w pamięci operacyjnej, a dane w nich zapisane zostają bezpowrotnie utracone po wyłączeniu serwera. Wykorzystywane są w nich indeksy mieszane (ang. *hash*), dlatego modyfikacje danych przebiegają bardzo szybko. Tabele tego typu powinny być używane jako tabele tymczasowe. Każda tabela typu `MEMORY` przechowywana jest w postaci pojedynczego pliku z rozszerzeniem *frm*, który zawiera jedynie definicję tabeli.

BLACKHOLE

Dane zapisywane w tego typu tabelach nie są przechowywane. Zapisywane w nich wiersze są automatycznie usuwane. Ze względu na to, że informacje o wykonywanych na nich operacjach są zapisywane w dzienniku zdarzeń serwera MySQL, tabele tego typu są przydatne podczas diagnozowania i testowania baz danych.

CSV

Tabele tego typu służą do przechowywania danych w plikach tekstowych typu *csv*. Są używane do importowania i eksportowania danych pomiędzy serwerem MySQL a innymi serwerami lub programami.

ARCHIVE

Tabele tego typu służą do przechowywania dużych ilości danych. Dane są przechowywane bez indeksów i są kompresowane przed umieszczeniem w tabeli. Podczas odczytu danych następuje ich dekompresja. Zmienianie lub usuwanie tych danych jest niedozwolone. Tabele typu `ARCHIVE` stosuje się głównie do przechowywania zarchiwizowanych danych lub danych diagnostycznych.

InnoDB

Domyślne tabele od wersji 5.5.4 MySQL. Są to tabele obsługujące transakcje. Serwer MySQL automatycznie blokuje odczytywane i modyfikowane wiersze tych tabel. Tabele `InnoDB` pozwalają również definiować i sprawdzać ograniczenia klucza obcego.

Aby wyświetlić informacje o dostępnych na serwerze typach tabel, należy użyć polecenia:

```
SHOW ENGINES;
```

4.7. Prawa dostępu do serwera

Podczas pracy z serwerem MySQL należy przestrzegać zasad bezpieczeństwa. Trzeba pamiętać o ustawieniu hasła dla konta administratora oraz konta anonimowego (`User=""`). Jeżeli po wpisaniu polecenia `mysql -u root` nie pojawi się komunikat z prośbą o podanie hasła, znaczy to, że hasło nie zostało ustawione. Nie należy pracować na koncie `root`. Nie należy definiować dostępu do bazy `mysql` innym użytkownikom poza administratorem. Użytkownikom należy nadawać jak najmniejsze uprawnienia — tylko te, które są konieczne. Nie należy nadawać praw dostępu do wszystkich baz danych, lecz jedynie do wybranych.

Prawa dostępu do serwera MySQL mogą być nadawane użytkownikom na poziomie:

- całego serwera,
- bazy danych (na przykład: `CREATE`, `ALTER`, `DROP`),
- obiektów bazy danych (na przykład: `SELECT`, `INSERT`, `UPDATE`, `DELETE`).

Informacje na temat praw dostępu do serwera MySQL są przechowywane w tabelach słownikowych bazy danych `mysql`. Są to tabele: `host`, `db`, `user`, `tables_priv`, `columns_priv` i `procs_priv`.

- `user` przechowuje informacje o prawach dostępu użytkownika niezależnie od bazy danych.
- `db` przechowuje informacje o prawach dostępu użytkownika w zależności od bazy danych.
- `host` przechowuje informacje w kontekście komputera, z którego łączy się użytkownik.

Pozostałe tabele przechowują informacje szczegółowe dotyczące praw dostępu do tabel, kolumn itp.

Po zalogowaniu się do bazy `mysql` można wyświetlić listę dostępnych tabel.

Przykład 4.22

```
USE mysql;

SHOW TABLES;

SELECT host, user FROM user WHERE user = 'adam';
```

Użyte w przykładzie polecenie `SELECT` pozwoli wyświetlić prawa dostępu użytkownika `adam`.

W serwerze MySQL prawa są dziedziczone. Oznacza to, że prawa nadane do obiektu znajdującego się wyżej w hierarchii są domyślnie dziedziczone przez obiekty znajdujące się niżej. Jednak prawa mogą być nadawane i odbierane na dowolnym poziomie.

4.7.1. Uprawnienia

Nadawanie praw

Prawa są nadawane użytkownikom instrukcją GRANT. Składnia polecenia ma postać:

```
GRANT [prawo] ON baza_danych.* TO '[użytkownik]','[host]' IDENTIFIED BY '[hasło]';
```

Prawa mogą być nadawane na różnych poziomach.

Prawa nadawane globalnie:

```
GRANT UPDATE ON *.* TO Marcin;
```

Prawa nadawane na poziomie domyślnej bazy danych:

```
GRANT UPDATE ON ksiegarnia.* TO Marcin;
```

lub:

```
GRANT UPDATE ON * TO Marcin;
```

Prawa nadawane na poziomie obiektów bazy danych (na przykład do wybranej tabeli):

```
GRANT INSERT ON ksiegarnia.ksiazki TO Marcin;
```

Prawa nadawane na poziomie wybranych kolumn tabeli:

```
GRANT UPDATE(tytul),INSERT ON ksiegarnia.ksiazki TO Marcin;
```

Do przyznania wszystkich przywilejów służy w poleceniu GRANT opcja ALL PRIVILEGES lub ALL.

Przykład 4.23

```
GRANT ALL PRIVILEGES on *.* TO Marcin;
```

Polecenie GRANT może zawierać dodatkowe klauzule:

- MAX_QUERIES_PER_HOUR — ogranicza liczbę zapytań,
- MAX_UPDATES_PER_HOUR — ogranicza liczbę zmian wprowadzanych do bazy,
- MAX_CONNECTIONS_PER_HOUR — ogranicza liczbę logowań użytkownika w ciągu godziny,
- MAX_USER_CONNECTIONS — ogranicza liczbę jednoczesnych połączeń uzyskiwanych z jednego konta.

Przykład 4.24

```
GRANT USAGE ON *.* TO Marcin WITH MAX_QUERIES_PER_HOUR 1;
```

Prawo USAGE oznacza, że użytkownikowi nie zostały nadane żadne prawa.

Odbieranie praw

Instrukcją `REVOKE` można usunąć wcześniej nadane prawa.

Przykład 4.25

```
REVOKE UPDATE ON ksiegarnia.Ksiazki FROM Marcin;
```

W podanym przykładzie użytkownikowi *Marcin* zostały odebrane uprawnienia do modyfikowania danych w tabeli *Ksiazki*.

Lista uprawnień

- `ALL` — użytkownik otrzymuje wszystkie prawa poza `GRANT OPTION`.
- `CREATE` — użytkownik może tworzyć tabele.
- `SELECT` — użytkownik może wyświetlać zawartość tabel.
- `INSERT` — użytkownik może dodawać nowe dane do tabel.
- `SHOW DATABASES` — użytkownik może przeglądać listy dostępnych baz danych.
- `USAGE` — umożliwia tworzenie użytkownika bez uprawnień.
- `GRANT OPTION` — użytkownik może nadawać uprawnienia innym użytkownikom.

Aby ułatwić przypisywanie uprawnień użytkownikom, w MySQL została wprowadzona koncepcja ról administracyjnych. Za pomocą ról można w szybki sposób przyznawać użytkownikowi zestaw uprawnień potrzebnych do pracy na serwerze. Użytkownikowi można nadać jedną lub kilka ról.

Dostępne role

DBA — wszystkie uprawnienia.

MaintenanceAdmin — uprawnienia do utrzymania serwera.

ProcessAdmin — uprawnienia do monitorowania i zatrzymywania procesów użytkownika.

UserAdmin — uprawnienia do tworzenia użytkowników i ustawiania haseł.

SecurityAdmin — uprawnienia do zarządzania kontami oraz nadawania i odbierania uprawnień serwera.

MonitorAdmin — uprawnienia do monitorowania serwera.

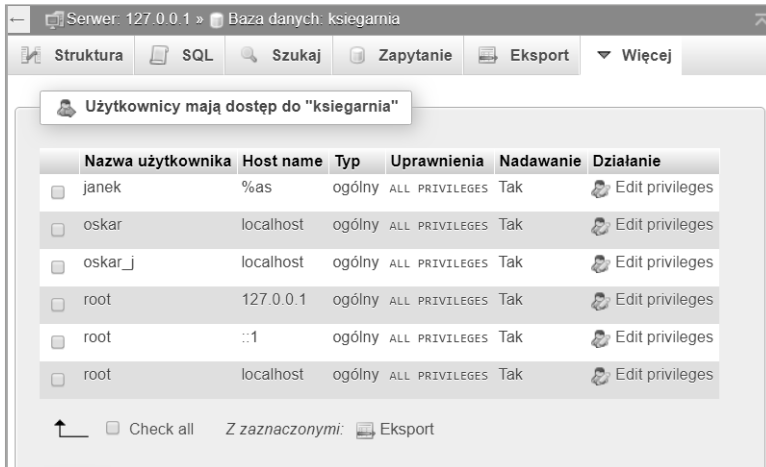
DBManager — uprawnienia do zarządzania bazami danych.

DBDesigner — uprawnienia do tworzenia i modyfikowania schematów baz danych.

ReplicationAdmin — uprawnienia do tworzenia replikacji i zarządzania nią.

BackupAdmin — uprawnienia do tworzenia kopii zapasowych baz danych.

Nadawane użytkownikom uprawnienia można kontrolować z poziomu phpMyAdmin. W zakładce **Uprawnienia** można odczytać nazwy użytkowników, którzy posiadają uprawnienia do wybranej bazy danych (rysunek 4.12).



Rysunek 4.12. Użytkownicy z dostępem do bazy księgarnia

Uprawnienia określonego użytkownika (rysunek 4.13) otrzymamy po wpisaniu polecenia:

```
SHOW GRANTS FOR nazwa_użytkownika;
```



Rysunek 4.13. Lista uprawnień użytkownika do bazy księgarnia

Po wybraniu systemowej bazy **mysql** i wpisaniu polecenia:

```
SELECT * FROM user;
```

uzyskamy informację o tym, jacy użytkownicy baz danych istnieją w systemie (rysunek 4.14).

| + Opcje | | | | Host | User | Password | Select_priv | Insert_priv | Update_priv | Delete_priv |
|--------------------------|--------|---------|------|-----------|---------|---|-------------|-------------|-------------|-------------|
| <input type="checkbox"/> | Edytuj | Kopiuuj | Usuń | localhost | root | | Y | Y | Y | Y |
| <input type="checkbox"/> | Edytuj | Kopiuuj | Usuń | 127.0.0.1 | root | | Y | Y | Y | Y |
| <input type="checkbox"/> | Edytuj | Kopiuuj | Usuń | :::1 | root | | Y | Y | Y | Y |
| <input type="checkbox"/> | Edytuj | Kopiuuj | Usuń | localhost | | | N | N | N | N |
| <input type="checkbox"/> | Edytuj | Kopiuuj | Usuń | localhost | pma | | N | N | N | N |
| <input type="checkbox"/> | Edytuj | Kopiuuj | Usuń | %as | janeek | *45B86621143A92EA9D915F23BBEC93B217E1CCDC | Y | Y | Y | Y |
| <input type="checkbox"/> | Edytuj | Kopiuuj | Usuń | localhost | oskar | | Y | Y | Y | Y |
| <input type="checkbox"/> | Edytuj | Kopiuuj | Usuń | localhost | oskar_j | | Y | Y | Y | Y |

Rysunek 4.14. Użytkownicy baz danych na serwerze MySQL

Ćwiczenie 4.1

Utwórz użytkownika *Janek* i nadaj mu pełne prawa do wszystkich bazy danych.

Rozwiązanie

W okienku nawigacyjnym narzędzia phpMyAdmin zaznaczamy bazę *ksiegarnia* i w zakładce SQL wpisujemy polecenie:

```
GRANT ALL ON * TO Janek IDENTIFIED BY 'Janek20';
```

Uzyskane przez użytkownika prawa możemy odczytać, wybierając zakładkę *Uprawnienia*.

Ćwiczenie 4.2

Ogranicz uprawnienia użytkownika *Janek* do tworzenia użytkowników i nadawania im uprawnień.

Rozwiązanie

```
REVOKE USAGE, GRANT OPTION ON * FROM Janek;
```

Ćwiczenie 4.3

Utwórz użytkownika *Michał*, który będzie użytkownikiem bazy *ksiegarnia* i nie będzie posiadał żadnych uprawnień do tej bazy. Sprawdź jego uprawnienia. Sprawdź, czy taki użytkownik istnieje.

Rozwiązanie

```
GRANT USAGE ON ksiegarnia.* TO Michał IDENTIFIED BY 'michal';
```

Ćwiczenie 4.4

Nadaj użytkownikowi *Michał* uprawnienia do bazy danych *ksiegarnia* do dodawania nowych danych i wyświetlania zawartości tabel. Sprawdź jego uprawnienia.

Rozwiązanie

```
GRANT SELECT, INSERT ON ksiegarnia.* TO Michał;
```


Zadanie 4.2

Utworzonym wcześniej użytkownikom przypisz prawa do bazy danych *ksiegarnia*. *Operator* powinien mieć pełne prawa do bazy. *Instruktor* powinien mieć prawo do przeglądania zawartości wszystkich tabel. Dodatkowo powinien mieć prawo dodawania nowych danych do tabeli rejestrującej nowych klientów. *Gracz* powinien mieć prawo przeglądania tabeli rejestrującej klientów. Sprawdź prawa dostępu różnych użytkowników do bazy danych.



4.8. Kopie bezpieczeństwa

Jednym z podstawowych działań administratora jest zapewnienie bezpieczeństwa danych przez tworzenie ich kopii.

4.8.1. Sprawdzanie spójności bazy danych

Przed wykonaniem kopii bezpieczeństwa należy sprawdzić, czy baza danych nie zawiera błędów i — w razie potrzeby — wykonać jej naprawę. Do sprawdzania poprawności tabel MyISAM, InnoDB i ARCHIVE służy polecenie `CHECK TABLE` w postaci:

```
CHECK TABLE Tabela1, Tabela2;
```

Polecenie sprawdzania poprawności tabel może mieć następujące opcje:

- **QUICK** — sprawdzanie poprawności tabel bez sprawdzania, czy wiersze mają właściwe referencje;
- **FAST** — sprawdzanie dotyczy tylko tabel, które nie zostały poprawnie zamknięte;
- **CHANGED** — sprawdzanie dotyczy tylko tabel, które zmieniły się od ostatniej kontroli lub nie zostały poprawnie zamknięte;
- **MEDIUM** — sprawdzanie dotyczy poprawności wierszy i niektórych referencji;
- **EXTENDED** — sprawdzane są wszystkie referencje; używana, gdy potrzebna jest pewność co do spójności danych w tabeli.

Podane opcje mogą być łączone w jednym poleceniu.

Przykład 4.26

```
CHECK TABLE Ksiazki FAST QUICK;
CHECK TABLE Autor MEDIUM;
```

4.8.2. Naprawa bazy danych

Do naprawienia uszkodzonej bazy danych z tabelami typu MyISAM lub ARCHIVE może być użyte polecenie `REPAIR TABLE` w postaci:

```
REPAIR TABLE Tabela1;
```

Do naprawy tylko pliku indeksu można użyć polecenia w postaci:

```
REPAIR TABLE Tabela1 QUICK;
```

4.8.3. Tworzenie pełnej kopii danych

Strategia tworzenia kopii bezpieczeństwa zaleca tworzenie w określonych momentach pełnych kopii baz danych oraz częste tworzenie różnicowych kopii bezpieczeństwa.

Utworzenie pełnej kopii baz danych umożliwia program narzędziowy `mysqldump`. W wyniku użycia tego narzędzia tworzony jest plik *sql* zawierający komendy źródłowej bazy danych.

Przed wykonaniem kopii należy zablokować bazę poleceniem `FLUSH TABLES`.

Składnia polecenia do wykonania kopii zapasowej ma postać:

```
mysqldump -u użytkownik -p[hasło] nazwa_bazy > plik.sql
```

Składnia polecenia do przywracania bazy danych wygląda podobnie:

```
mysql -u użytkownik -p[hasło] nazwa_bazy < plik.sql
```

Podstawowe parametry polecenia `mysqldump` to:

- `databases` — kopiowana jest zawartość bazy danych i jej struktura. Opcja pozwala na zapisanie wielu baz danych w jednym pliku.
- `all-databases` (lub `-A`) — kopiowana jest zawartość wszystkich baz danych i tabel dostępnych na serwerze. Nie ma możliwości wybierania pojedynczych baz.
- `no-create-info` — w utworzonej kopii nie zostanie zapisana informacja o strukturze tabel (nazwy pól, nazwy tabel, indeksów). Opcja tworzy kopię samych danych.
- `no-data` — tworzona jest kopia struktury bazy i tabel.
- `default-character-set=charset_name` — możliwość ustawienia kodowania znaków podczas wykonywania kopii bezpieczeństwa.
- `opt nazwa_bazy` — tworzona jest kopia bazy danych wraz z rozszerzonymi informacjami MySQL na przykład na temat blokowania tabel.
- `single-transaction` — tworzona jest spójna kopia działającej bazy danych.
- `flush-logs` — przed wykonaniem kopii bezpieczeństwa nastąpi zapisanie dziennika transakcji.
- `master-data` — w pliku kopii zostaną zapisane informacje o bieżącym dzienniku transakcji, takie jak: nazwa, stan, pozycja, potrzebne do ustawienia parametrów replikacji dla serwera *slave*.
- `add-drop-database` — przywraca bazę danych z pliku przy jednoczesnym usunięciu istniejącej bazy.
- `add-drop-table` — przywraca tabelę (strukturę i dane lub tylko strukturę) z pliku przy jednoczesnym usunięciu istniejącej tabeli.

- `host` (lub `-h`) — nazwa lub adres IP komputera, z którego mają zostać skopiowane lub przywrócone dane.
- `user` (lub `-u`) — deklaracja użytkownika, który ma prawo do wykonywania kopii zapasowych bazy.
- `password` (lub `-p`) — opcja wykorzystywana do uwierzytelnienia użytkownika na serwerze bazodanowym.

Przykład 4.27

```
mysqldump -u user -phaslo databases > kopia_baz.sql
```

W wyniku wykonania polecenia podanego w przykładzie zostanie utworzona kopia wszystkich baz danych serwera.

Przykład 4.28

```
mysqldump -u user -phaslo ksiegarnia > kopia_ksiegarnia.sql
```

W wyniku wykonania polecenia zostanie utworzona kopia bazy danych *ksiegarnia*.

Przykład 4.29

```
mysql -u root -phaslo
```

```
mysql -u root -phaslo ksiegarnia < /katalog/kopia_ksiegarnia.sql
```

W wyniku wykonania poleceń po zalogowaniu się do serwera zostanie przywrócona z pliku *kopia_ksiegarnia.sql* baza danych *ksiegarnia*.

Przykład 4.30

```
mysqldump -u root -p --single-transaction --all-databases > kopia_baz.sql
```

W podanym przykładzie zostanie utworzona kopia wszystkich tabel przy użyciu jednej transakcji.

Przykład 4.31

```
mysqldump -u root -p --single-transaction --flush-logs --master-data --all-databases > kopia_baz.sql
```

W wyniku wykonania polecenia z przykładu zostanie utworzona spójna kopia działającej bazy danych, a także zapisane zostaną dziennik transakcji i informacje o nim.

Tworzenie kopii bezpieczeństwa dla tabel typu `MyISAM` polega na skopiowaniu plików typu *frm*, *MYD* i *MYI*.

Inną metodą tworzenia kopii bezpieczeństwa dla tabel typu `InnoDB` może być zatrzymanie serwera MySQL i skopiowanie plików *ibd*, *ib_logfile**, *frm*, *my.cnf*.

4.8.4. Tworzenie przyrostowej kopii danych

Tworzenie przyrostowej kopii bezpieczeństwa jest związane z dziennikiem transakcji (ang. *binary log*). Są w nim zapisywane wszystkie operacje dotyczące modyfikowania bazy danych wykonywane od momentu utworzenia ostatniej kopii bezpieczeństwa. W przypadku awarii istnieje możliwość odtworzenia bazy danych na podstawie tych zapisów. Dzienniki transakcji są traktowane jako kopie przyrostowe bazy danych. Aby w MySQL operacje były zapisywane do dziennika transakcji, należy uruchomić serwer z opcją `--log-bin` lub ustawić tę opcję w pliku konfiguracyjnym. Przy tworzeniu nowego pliku dziennika do jego nazwy dodawany jest kolejny numer. Nowy dziennik transakcji jest tworzony zawsze, gdy:

- uruchamiany jest serwer,
- zostanie wykonane polecenie `FLUSH LOGS`,
- aktualny plik dziennika osiągnie maksymalny rozmiar.

Do lokalizacji plików dzienników transakcji służy polecenie:

```
SHOW BINLOG EVENTS;
```

Aby odtworzyć stan bazy danych sprzed awarii, potrzebne są pełna kopia bazy i plik dziennika transakcji.

4.8.5. Odzyskiwanie danych

Odzyskiwanie bazy danych z kopii bezpieczeństwa odbywa się w dwóch etapach:

- odzyskanie bazy danych z pełnej kopii (na przykład utworzonej programem `mysqldump`) poleceniem:

```
mysql -u root -p < kopia.sql
```

- odzyskanie wszystkich danych zmodyfikowanych po utworzeniu pełnej kopii bezpieczeństwa poleceniem:

```
mysqlbinlog -u root -p binlog.2 | mysql -u root -p
```



4.9. Import i eksport danych

W programie phpMyAdmin import i eksport danych są realizowane za pomocą zakładek *Import* i *Eksport* dostępnych w oknie głównym narzędzia.

4.9.1. Eksport danych

Po zaznaczeniu w oknie nawigacyjnym bazy, która będzie eksportowana, należy wybrać zakładkę *Eksport* i w otwartym oknie określić metodę eksportu (opcja *Dostosuj* — wyświetli wszystkie możliwe opcje). Następnie należy wybrać format pliku (SQL) i określić informacje bazy danych, które zostaną wyeksportowane do pliku (rysunek 4.15). Na koniec należy kliknąć przycisk *Wykonaj*.

Eksportowanie tabeli z bazy "ksiegarnia"

Export templates:

New template:

Template name: **Utwórz**

Existing templates:

Template: -- Select a template -- **Update** **Usun**

Export method:

☐ Szybko - wyświetlane są tylko minimalne opcje

☒ Dostosuj - wyświetli wszystkie możliwe opcje

Format:

SQL

Tabele:

| Tabele | Struktura | Dane |
|---|-------------------------------------|-------------------------------------|
| Select all | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> autor | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> faktura | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> klient | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |

Console

Rysunek 4.15. Okno eksportu bazy danych

Zostanie utworzony plik o nazwie takiej jak wybrana baza danych, który będzie zawierał informacje podobne do pokazanych na rysunku 4.16.

```
ksiegarnia.sql - not connected - X
-- phpMyAdmin SQL Dump
-- version 4.5.1
-- http://www.phpmyadmin.net
--
-- Host: 127.0.0.1
-- Czas generowania: 27 Cze 2019, 21:16
-- Wersja serwera: 10.1.19-MariaDB
-- Wersja PHP: 5.6.28

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: `ksiegarnia`
--

--
-- Struktura tabeli dla tabeli `autor`
--

CREATE TABLE `autor` (
  `nazwisko` varchar(50) NOT NULL,
  `imie` varchar(30) NOT NULL,
  `narodowosc` varchar(30) DEFAULT NULL,
  `okres_tworzenia` varchar(35) DEFAULT NULL,
  `jezyk` varchar(30) DEFAULT NULL,
  `rodzaj_tworczości` varchar(35) DEFAULT NULL,
```

Rysunek 4.16. Plik wyeksportowanej bazy danych

4.9.2. Import danych

Aby zaimportować dane zapisane w pliku, należy utworzyć nową bazę danych i po jej zaznaczeniu w oknie nawigacyjnym wybrać zakładkę **Import** (rysunek 4.17). W otwartym oknie należy kliknąć przycisk **Wybierz plik** i po podaniu ścieżki do pliku można zaimportować wcześniej wyeksportowany projekt (lub dowolny inny projekt), klikając przycisk **Wykonaj**.

Rysunek 4.17. Okno importu bazy danych

4.10. Optymalizacja wydajności SZBD

4.10.1. Optymalizacja wydajności systemu bazodanowego

Wydajność SZBD określana jest najczęściej za pomocą następujących parametrów:

- liczba operacji przeprowadzanych na sekundę, czyli przepustowość systemu,
- czas potrzebny na zwrócenie wyniku, czyli czas reakcji systemu.

Wartość pierwszego parametru zależy przede wszystkim od dostępnych dla serwera baz danych zasobów systemu komputerowego (pamięci operacyjnej, mocy obliczeniowej, wydajności systemu wejścia-wyjścia, przepustowości lokalnej sieci). Wartość drugiego

zależy głównie od logicznej i fizycznej struktury bazy danych i od aplikacji klienckich. Wpływ na wydajność systemu bazodanowego ma również stosowana przez serwer i klienty metoda pobierania i przetwarzania danych. Mała wydajność jednego z podanych składników może negatywnie wpływać na wyniki monitorowania wydajności pozostałych składników.

Monitorowanie wydajności systemu bazodanowego należy przeprowadzać, zaczynając od sprawdzenia wydajności komputera. Następnie trzeba prześledzić pracę systemu operacyjnego, sprawdzić wykorzystanie zasobów systemowych przez serwer baz danych oraz przeanalizować strukturę bazy danych. Powinno się również monitorować wydajność aplikacji klienckiej.

4.10.2. Optymalizacja bazy danych

Jednym z czynników wpływających na wydajność pracy serwera jest odpowiednie skonstruowanie logicznej i fizycznej struktury bazy danych i aplikacji klienckich. Najczęstszymi przyczynami małej efektywności pracy z bazą danych są:

- nieefektywne projekty schematów baz,
- źle utworzone indeksy,
- źle skonstruowane zapytania,
- nieoptymalna struktura przechowywania danych.

Większość baz pracuje na ogromnej ilości danych, dlatego powinny one być projektowane tak, aby jak najmniej obciążały serwer oraz aby dane nie zajmowały zbyt dużo miejsca na dysku. Wykonanie optymalizacji bazy danych może pomóc w zwiększeniu jej wydajności i zmniejszeniu jej rozmiarów. Podstawowymi działaniami, które powinny zostać wykonane, są:

- normalizacja bazy danych,
- prawidłowe tworzenie indeksów,
- optymalizacja zapytań i operacji wykonywanych w bazie danych.

Normalizacja

Prawidłowo opracowany projekt schematu bazy danych jest pierwszym elementem optymalizacji. Zasady poprawnej budowy schematu bazy są definiowane za pomocą postaci normalnych. Normalizacja chroni przed powielaniem tej samej informacji w tabelach. Brak normalizacji prowadzi do wielu utrudnień (anomalii bazy danych) w trakcie korzystania z bazy i jej modyfikowania. W celu zwiększenia wydajności bazy danych należy zadbać o to, aby schemat baz danych spełniał wymogi normalizacji.

Indeksy

Indeksy w bazie danych pozwalają na szybkie wyszukiwanie danych, co powoduje znacznie szybsze wykonywanie zapytań do bazy.

Zaletą stosowania indeksów jest ograniczenie ilości danych odczytywanych z bazy, przyspieszenie wyszukiwania informacji oraz sortowanie danych. Dostęp do rekordu odszukanego przez indeks jest bardzo szybki i efektywny. Wadą indeksów jest to, że zajmują na dysku dodatkowe miejsce, a ponadto muszą być na bieżąco aktualizowane. Każde wstawienie, usunięcie lub zaktualizowanie danych w tabeli wiąże się z aktualizacją wszystkich zdefiniowanych dla niej indeksów. Zastosowanie indeksowania daje najlepsze rezultaty przy wybieraniu małej liczby rekordów z dużego zbioru. Wtedy najlepiej widać korzyść z ograniczenia ilości danych, które muszą zostać odczytane z dysku. Przyjmuje się, że indeks jest opłacalny, gdy z tabeli odczytywane jest nie więcej niż około 15% rekordów.

Zastosowanie indeksu przyspiesza odczyt danych. Spowalnia jednak modyfikowanie danych, ponieważ wymaga aktualizowania informacji w indeksach. Jeżeli tabela zawiera kilka indeksów, to czas modyfikacji danych może być wielokrotnie dłuższy w porównaniu z aktualizacją tej tabeli bez indeksów. Jeżeli na przykład tylko dodajemy nowe dane do bazy, użycie indeksów może znacznie spowolnić jej działanie.

Poniżej został pokazany przykład wpływu indeksów na wydajność bazy danych.

Przykład 4.32

Utworzone zostały dwa indeksy, każdy dla innego pola:

```
CREATE INDEX indeks_imie ON Klient (imie);
CREATE INDEX indeks_nazw ON Klient (nazwisko);
```

Przykład 4.33

Utworzony został jeden indeks dla dwóch kolumn jednocześnie:

```
CREATE INDEX indeks_imie_nazw ON Klient (imie, nazwisko);
```

Po wykonaniu zapytania:

```
SELECT * FROM Klient WHERE imie='Jan' AND nazwisko='Kowalski';
```

w pierwszym przypadku wykonane zostaną następujące czynności:

- wyszukane zostaną wszystkie rekordy dla *imie* = Jan,
- wyszukane zostaną wszystkie rekordy dla *nazwisko* = Kowalski,
- zostanie wybrana wspólna część zbiorów rekordów z pierwszego i drugiego wyszukiwania i zwrócona jako wynik zapytania.

W drugim przypadku potrzebne dane zostaną wyszukane w jednym kroku, w którym równocześnie sprawdzone zostaną wartości w polach *imie* i *nazwisko*.

Każdy indeks zajmuje pewną ilość miejsca na dysku, dlatego nie należy tworzyć indeksów dla każdej kolumny w bazie. Należy przeanalizować zapytania, które będą wykonywane w bazie, i podjąć decyzję, gdzie i jakie indeksy utworzyć. Indeksy należy tworzyć dla kluczy głównych i dla kluczy obcych, gdyż te pola są najczęściej przeszukiwane.

Optymalizacja zapytań i operacje wykonywane w bazie danych

Podczas wykonywania zapytania, oprócz uzyskania interesujących nas informacji, bardzo ważna jest szybkość wykonania tego zapytania. W systemach zarządzania bazą danych występuje narzędzie służące do optymalizacji zapytań (ang. *query optimizer*). Aby zapewnić wydajność bazy danych, optymalizator analizuje zapytanie i tworzy możliwie optymalny plan jego wykonania (ang. *execution plan*). Plan ten zawiera opis użytych indeksów, sposób dostępu do danych z tabeli, sposób i kolejność łączenia tabel, uwzględnienie warunków z sekcji `WHERE`.

Aby przyspieszyć wykonywanie zapytań, należy je konstruować tak, by były wykonywane na jak najmniejszej ilości danych. Zmniejszenie ilości przetwarzanych danych zmniejsza ilość potrzebnych zasobów oraz zwiększa efektywność działania indeksów.

Jedną z metod zmniejszenia ilości przetwarzanych danych jest ograniczenie listy kolumn w zapytaniu `SELECT`.

Przykład 4.34

Zamiast stosować polecenie w postaci:

```
SELECT * FROM Klient WHERE imie='Jan' AND nazwisko='Kowalski';
```

można zapisać:

```
SELECT id_klienta, imie, nazwisko FROM Klient WHERE imie='Jan' AND
nazwisko='Kowalski';
```

Inną metodą prowadzącą do zmniejszenia ilości przetwarzanych danych jest ich filtrowanie za pomocą klauzuli `WHERE`:

```
SELECT nazwisko, imie, miejscowosc, ulica, nr_domu
FROM Klient
WHERE miejscowosc='Kraków';
```

Nie powinno się stosować porządkowania danych, jeżeli nie jest to konieczne (klauzula `ORDER BY`). Dane należy pobierać w takiej kolejności, w jakiej są zapisane w bazie. Trzeba unikać zagnieżdżonych zapytań i klauzuli `GROUP BY`.

Transakcje

Sama transakcja nie jest narzędziem optymalizacji, natomiast może mieć na nią wpływ. Zatwierdzenie transakcji instrukcją `COMMIT` powoduje zapisanie zmian dokonanych w bazie danych. Natomiast brak zatwierdzenia transakcji prowadzi do utrzymywania rosnącego zbioru wartości danych na wypadek wycofania zmian instrukcją `ROLLBACK`. Najczęściej sposób obsługi transakcji jest wymuszany względami bezpieczeństwa danych i częstsze wykonywanie instrukcji `COMMIT` jest możliwe tylko w nielicznych przypadkach.

Procedury składowane w bazie danych

Inną możliwością optymalizacji bazy danych jest tworzenie procedur i funkcji składowanych. Dzięki temu, że są one przechowywane i wykonywane w bazie danych, zostają wyeliminowane opóźnienia związane z przesyłaniem danych poza bazę. W związku z tym możliwy jest wzrost wydajności.

Struktura bazy danych

Wielkość bazy danych, jej uporządkowanie oraz operacje wykonywane na plikach bazodanowych mają istotny wpływ na jej wydajność. Aby zapewnić maksymalną wydajność, należy:

- umieścić pliki bazy danych na oddzielnym dysku twardym,
- pliki dziennika transakcyjnego umieścić na oddzielnym, wydajnym dysku twardym,
- utworzyć dla bazy danych kilka plików i umieścić je na oddzielnych dyskach twardych,
- dla tabel i powiązanych z nimi indeksów utworzyć grupy plików i zapisać je na oddzielnych dyskach,
- wykonywać regularne defragmentowanie dysków,
- uporządkować logiczną strukturę plików bazy danych.

4.10.3. Optymalizacja wydajności serwera

SQL Server

Tak jak wydajność każdego serwera bazodanowego, również wydajność serwera MS SQL Server zależy w dużej mierze od zasobów systemu komputerowego. Na przykład mała ilość pamięci operacyjnej spowoduje częstsze odwoływanie się do danych zapisanych na dysku, a to skutkuje obniżeniem wydajności systemu. Powstaną kolejki procesów czekających na obsłużenie. Wskutek powstałej sytuacji bieżące wysyłanie danych do klientów i ich odbieranie stanie się niemożliwe.

SQL Server działa w środowisku systemu Windows, dlatego wszystkie zasoby systemowe można monitorować, korzystając z narzędzi *Monitor systemu* oraz *Dzienniki wydajności i alerty*. Domyślna konfiguracja serwera SQL Server umożliwia dynamiczne przydzielanie dostępnych zasobów sprzętowych potrzebnych do pracy serwera i w większości przypadków nie ma potrzeby jej zmieniania.

Pamięć operacyjna

Menedżer pamięci w programie Microsoft SQL Server eliminuje konieczność ręcznego zarządzania pamięcią dostępną dla serwera SQL Server. Po uruchomieniu SQL Server dynamicznie określa ilość potrzebnej pamięci na podstawie ilości pamięci systemu operacyjnego i innych aktualnie używanych aplikacji. Jest to optymalny przydział, zapewniający najlepszą wydajność pracy serwera.

4.10.4. Poprawa wydajności serwera MySQL

Czynnikami wpływającymi na wydajność pracy serwera MySQL mogą być sprzęt lub konfiguracja serwera. Na przykład zwiększenie ilości pamięci operacyjnej może znacząco poprawić wydajność systemu.

Konfigurację serwera można sprawdzić i zmodyfikować w pliku konfiguracyjnym *my.ini* lub *my.cnf*. Dostępne są następujące parametry:

- `key_buffer_size` — opisuje ilość pamięci dostępnej do przechowywania kluczy indeksu. Domyślna wartość to 8 MB, ale parametr może zostać ustawiony nawet na 25% pamięci RAM.
- `query_cache_size` — opisuje ilość pamięci zarezerwowanej na zapytania. Domyślna wartość jest równa 0. Jeżeli w bazie danych występuje dużo powtarzających się zapytań, wartość tę należy zwiększyć.
- `table_open_cache` — określa liczbę deskryptorów tabeli MySQL przechowywanych w pamięci podręcznej. Domyślną wartością jest 64. Jeśli wielu użytkowników korzysta jednocześnie z tabel, wartość ta powinna zostać zwiększona.

Statystyki działania serwera można uzyskać po uruchomieniu programu narzędziowego `mysqlreport`. Wyświetli on raport z informacją o konfiguracji serwera oraz o wykorzystaniu zasobów przez serwer. Na podstawie tego raportu można ustalić obciążenie serwera, sprawdzić, czy konfiguracja jest poprawna, a także czy struktura bazy danych jest prawidłowa i czy zapytania są optymalne.

Zadanie 4.3

W wybranym przez siebie serwerze baz danych (SQL Server lub MySQL) przeprowadź optymalizację bazy danych *ksiegarnia*. Przeanalizuj poprawność schematu bazy danych. Sprawdź zdefiniowane klucze podstawowe, klucze obce i utworzone indeksy. Przeanalizuj i zmodyfikuj konstrukcję utworzonych zapytań.



4.11. Pytania i zadania

4.11.1. Pytania

1. Co należy do podstawowych zadań administratora baz danych?
2. Na czym polega tryb uwierzytelnienia na serwerze MS SQL Server?
3. Na czym polega autoryzacja użytkownika na serwerze MS SQL Server?
4. W jaki sposób na serwerze SQL Server nadawane są uprawnienia użytkownikom?
5. Kiedy należy wykonywać pełną kopię bazy danych?
6. Na czym polega sprawdzanie spójności bazy danych?
7. Jak działa tryb uwierzytelnienia na serwerze MySQL?
8. W jaki sposób na serwerze MySQL nadawane są uprawnienia użytkownikom?

- 9. Kiedy powinna być tworzona kopia przyrostowa bazy danych?
- 10. Na czym polega optymalizacja wydajności Systemu Zarządzania Bazą Danych?

4.11.2. Zadania

Zadanie 1.

Korzystając z wybranego serwera baz danych, skonfiguruj dla bazy danych *Moja_szkoła* konta użytkowników. Przypisz im odpowiednie role i uprawnienia do bazy.

Zadanie 2.

Utwórz pełną kopię bezpieczeństwa bazy danych *Moja_szkoła*. Określ miejsce jej przechowywania.

Zadanie 3.

Przeprowadź optymalizację bazy danych *Moja_szkoła*. Przeanalizuj poprawność schematu bazy danych. Sprawdź zdefiniowane klucze podstawowe, klucze obce i utworzone indeksy. Przeanalizuj i zmodyfikuj konstrukcję utworzonych zapytań.

Bibliografia

Literatura

Jolanta Pokorska, *Kwalifikacja E.14. Tworzenie baz danych i administrowanie bazami. Podręcznik do nauki zawodu technik informatyk*, Helion, Gliwice 2014.

Jolanta Pokorska, *Kwalifikacja EE.09. Programowanie, tworzenie i administrowanie stronami internetowymi i bazami danych. Część 3. Tworzenie i administrowanie bazami danych. Podręcznik do nauki zawodu technik informatyk*, Helion, Gliwice 2019.

Źródła internetowe

<http://www.w3big.com/pl/mysql/mysql-select-query.html>.

<https://dev.mysql.com/doc/refman/5.7/en/introduction.html>.

[https://docs.microsoft.com/pl-pl/previous-versions/hh125811\(v=msdn.10\)](https://docs.microsoft.com/pl-pl/previous-versions/hh125811(v=msdn.10)).

<https://www.sqlpedia.pl/kurs-sql/>.

https://www.w3schools.com/sql/sql_datatypes.asp.

<https://docs.microsoft.com/pl-pl/dotnet/framework/data/adonet/sql/overview-of-sql-server-security>.

Skorowidz

A

- abstrakcja danych, 48
- administrator baz danych, 165
- administrowanie serwerami, 165
- aktualizowanie danych, 117
- algebra relacji, 13
- aliasy, 111
- ANSI, 71
- architektura
 - 3-warstwowa, 53
 - klient-serwer, 52
 - systemu baz danych, 52
- atrybut, 19, 22
 - AUTO_INCREMENT, 87
 - CHECK, 91
 - DEFAULT, 89
 - IDENTITY, 87
 - NOT NULL, 86
 - UNIQUE, 90, 92
- atrybuty kolumn, 85
- autoryzacja, 167

B

- bazy danych
 - architektura 3-warstwowa, 53
 - architektura klient-serwer, 52
 - hierarchia obiektów, 76
 - indeksy, 199
 - konceptualne projektowanie, 21
 - modele, 8
 - naprawa, 193
 - normalizacja, 199
 - operacje wykonywane, 201
 - optymalizacja, 199
 - pełna kopia, 179, 194
 - planowanie, 19
 - prawidłowy projekt, 38
 - procedury składowane, 202
 - projektowanie, 7, 18
 - przyrostowa kopia, 179, 196
 - przywracanie, 181
 - relacyjne, 7, 11
 - schemat, 29
 - sprawdzanie spójności, 180, 193
 - struktura, 202
 - systemowe, 169

- systemy zarządzania, 51
- tworzenie, 170
- typy danych, 43
- zarządzanie, 169
- zmiana parametrów, 170
- bezpieczeństwo danych, 47
- blokady
 - tryby, 137
 - współdzielone, 137
 - wyłączne, 137
 - zakresy, 138
- blokowanie danych, 137
- brudne odczyty, 138

C

- CASE, Computer Aided Software Engineering, 25
- cechy relacyjnej bazy danych, 46

D

- DBMS, Database Management System, 53
- DCL, Data Control Language, 72, 73
- DDL, Data Definition Language, 72, 73
- definiowanie
 - klucza obcego, 114
 - klucza podstawowego, 85
 - makr, 96
 - relacji, 11
- diagram
 - ERD, 21, 24
 - związków encji, 23
- DML, Data Manipulation Language, 72, 73
- dodanie kolumny, 84
- DQL, Data Querying Language, 73
- druga postać normalna, 35
- dziedziczenie uprawnień, 177
- dziedzina, 19

E

- eksportowanie
 - danych, 182, 196
 - projektu, 29
- encja, 19, 21

- ERD, Entity Relationship Diagram, 21

F

- funkcje
 - agregujące, 104
 - składowane, 150
 - wbudowane, 152

G

- graficzna reprezentacja
 - encji, 21
 - opcjonalności związku, 23
 - związków, 22
- grupowanie danych, 104

H

- hierarchia obiektów, 76

I

- iloczyn kartezjański, 14, 110
- importowanie danych, 182, 198
- indeksy, 130, 199
 - mieszane, 187
- instalowanie serwera
 - MySQL, 55, 57
 - SQL Server 2017, 62
- instrukcja
 - COMMIT, 201
 - DELETE, 95
 - DENY, 177
 - DQL, 73
 - EXCEPT, 120
 - GRANT, 176, 189
 - INSERT, 92
 - INTERSECT, 120
 - REVOKE, 176
 - SELECT, 96
 - klauzule, 97
 - wrażenia, 98
 - UNION, 119
 - UPDATE, 94
- instrukcje
 - DCL, 72, 73
 - DDL, 72, 73
 - DML, 72, 73
 - języka SQL, 73

TCL, 73
warunkowe, 145
integralność, 46
 danych, 13, 17
 encji, 13
 referencyjna, 13
izolowanie transakcji, 138, 139

J

język
 definiowania danych, DDL, 77
 SQL, 71

K

kaskadowe
 usuwanie danych, 117
 aktualizowanie danych, 118
klauzula
 CONSTRAINT, 116
 DISTINCT, 97
 FROM
 podzapytania, 126
 GROUP BY, 106
 HAVING, 106
 ORDER BY, 99
 REFERENCES, 116
 TOP, 103
 WHERE, 100
 podzapytania, 122
klauzule dla instrukcji SELECT, 97
klient-serwer, 52
klucz, 13
 obcy, 114
 podstawowy, 16, 85
 sztuczny, 17
kod SQL wygenerowany automatycznie, 30
konceptualne projektowanie, 21
konstrukcja zapytania, 97
kontrola współbieżności, 137
kopia bezpieczeństwa, 178
 pełna, 179, 194
 przyrostowa, 179, 196

L

lista uprawnień, 190

Ł

łączenie
 tabel, 108
 wyników zapytań, 119

M

manipulowanie danymi, DML, 92
model baz danych, 8
 hierarchiczny, 9
 obiektowy, 10
 relacyjny, 10, 11
 sieciowy, 9
modele konceptualne, 21
modyfikowanie
 kolumny, 84
 widoku, 129
MS SQL Server, 61
MySQL, 185
 konfigurowanie serwera, 185
 kopie bezpieczeństwa, 193
 optymalizacja wydajności, 198
 pełne kopie, 194
 poprawa wydajności serwera, 203
 prawa dostępu, 188
 przyrostowe kopie, 196
 tryby uwierzytelnienia, 185
 typy tabel, 186
 zarządzanie bazami danych, 186

N

nadawanie
 praw, 189
 uprawnień, 176
naprawa bazy danych, 193
narzędzia CASE, 25
narzędzie, *Patrz:* program
niepowtarzalne odczyty, 138
niezależność danych, 48
normalizacja, 199
 tabel, 34

O

odbieranie
 praw, 190
 uprawnień, 176
odczyty widma, 139
odzyskiwanie danych, 196
okno
 edytowania połączenia, 28
 edytowania tabeli, 26
 narzędzia phpMyAdmin, 60
operator
 ALL, 126
 ANY, 125
 EXISTS, 124
 SOME, 125
operatory podzapytań, 124

optymalizacja
 bazy danych, 199
 wydajności serwera, 202
 wydajności SZBD, 198
 zapytań, 201

P

pakiet
 MySQL, 57
 XAMPP, 59
pamięć operacyjna, 202
perspektywy, *Patrz:* widoki
phpMyAdmin, 59
pierwsza postać normalna, 34
podzapytania, 121
 klauzuli FROM, 126
 klauzuli WHERE, 122
 skorelowane, 128
 w instrukcjach
 modyfikujących dane, 127
połączenie
 BULK INSERT, 183
 mysqldump, 194
 REPAIR TABLE, 193
połączenia, 17
 krzyżowe, 110
 wewnętrzne, 108
 wielokrotne, 110
 zewnętrzne, 108, 109
poziomy izolowania transakcji, 139
prawa dostępu do serwera, 172, 188
procedury
 składowane, 147, 202
 wbudowane, 150
program
 DBDesigner4, 27
 phpMyAdmin, 59
 SQL Server Management Studio, 66
projekcja, 14
projektowanie bazy danych, 18
przypisywanie do ról, 174
przywracanie bazy danych, 181
punkty przywracania, 136

R

Read Committed, 140
Read Uncommitted, 139
relacja, 11
 „jeden do jednego”, 18, 31
 „jeden do wielu”, 27
 „wiele do jednego”, 18, 32
 „wiele do wielu”, 28, 33
relacyjna baza danych, 7, 11

Repeatable Read, 141
 role, 172, 190
 bazodanowe, 173
 serwerowe, 173

S

schemat, 81
 bazy danych, 29
 relacji, 12
 selekcja, 13
 serializable, 142
 serwery bazodanowe, 54, 165
 MySQL, 55, 57
 SQL Server, 202
 składnia języka SQL, 72
 skrypty tworzące tabele, 160, 162
 słowo kluczowe INNER JOIN, 108
 sortowanie, 99
 spójność bazy danych, 180, 193
 SQL, Structured Query Language, 71
 dialekty języka, 72
 składnia języka, 72
 standardy języka, 71
 terminatory, 72
 typy danych, 74
 SQL Server Management Studio, 66
 optymalizacja wydajności, 202
 standard
 ANSI, 71
 SQL99, 71
 struktura
 bazy danych, 202
 tabeli, 16, 61
 systemowe bazy danych, 169
 systemy zarządzania bazą danych, 8, 51
 SZBD, Database Management Systems, 8, 51

Ś

środowisko SQL Server, 166

T

tabele, 15
 klucz podstawowy, 16
 łączenie, 108
 pochodne, 126
 tworzenie, 79
 usuwanie, 79

 w MySQL, 186
 zmiana struktury, 84
 TCL, Transaction Control Language, 73
 terminatory SQL, 72
 transakcje, 131, 201
 Autocommit, 133
 Explicit, 132
 Implicit, 134
 izolowanie, 138
 poziomy izolowania, 139
 zagnieżdżanie, 135
 transformacja
 encji do tabeli, 31
 modelu konceptualnego, 31
 trwałość danych, 46
 tryb
 niezatwierdzonego odczytu, 139
 odczytu zatwierdzonego, 140
 powtarzalnego odczytu, 141
 szeregowania, 142
 tryby uwierzytelniania, 166, 185
 trzecia postać normalna, 36
 tworzenie
 bazy danych, 170, 186
 kopii bezpieczeństwa, 179, 194
 modelu konceptualnego, 21
 ról, 174
 tabel, 79
 widoków, 129
 typy
 danych, 43, 55
 binarne, 45
 daty i czasu, 45
 liczbowe, 44
 specjalne, 46
 tekstowe, 43
 walutowe, 45
 połączeń, 18

U

uprawnienia, 155, 175
 dziedziczenie, 177
 nadawanie, 176
 odbieranie, 176
 usuwanie
 kolumn, 84
 ról, 174
 tabel, 79
 widoku, 129
 utrata aktualizacji, 138
 uwierzytelnianie, 166, 167

W

wartość
 domyślna, 59, 64
 NULL, 14, 76
 widoki, 128, 164
 modyfikowanie, 129
 tworzenie, 129
 usuwanie, 129
 więzy
 integralności, 114
 ogólne, 13
 właściciel obiektu, 178
 właściwości transakcji, 131
 współbieżność, 136
 współdzielenie danych, 48
 wybór
 attributów, 14
 krotek, 13
 wierszy, 100, 103
 wydajność SZBD, 198
 wyrażenia
 CASE, 146
 tablicowe, CTE, 146
 w instrukcji SELECT, 98
 wyzwalacze, triggers, 152
 DDL, 154
 DML, 153
 logowania, 155

X

XAMPP, 59

Z

zadania administratora, 165
 zagnieżdżanie transakcji, 135
 zakleszczenia, 138
 zakresy blokad, 138
 zarządzanie bazami danych, 8, 51
 zbiór encji, 40, 41
 złączenia, *Patrz:* połączenia, 109
 złączenie dwóch relacji, 14
 zmiana struktury tabeli, 84
 zmienne, 143
 systemowe, 144
 związki
 „jeden do jednego”, 18, 31
 „jeden do wielu”, 27
 „wiele do jednego”, 18, 32
 „wiele do wielu”, 28, 33
 opcjonalny, 23
 wymagany, 23
 związki między zbiorami, 42

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 



KOMPLEKSOWO SZKOLIMY NOWOCZESNY BIZNES



IT



BIZNES



PROJEKTY



PROCESY

NASZE SZKOLENIA SĄ PROWADZONE
ZGODNIE Z METODĄ

BLENDED LEARNING

modelem kształcenia, który łączy tradycyjne szkolenie
z dostępem do nowoczesnych narzędzi - wideokursów,
e-booków i audiobooków

T: 609 850 372 E: SZKOLENIA@HELION.PL

WWW.HELIONSZKOLENIA.PL