DO NOWEJ PODSTAWY PROGRAMOWEJ

Część 2

Kwalifikacja E.14
Tworzenie baz danych i administrowanie bazami

Podręcznik do nauki zawodu
technik informatyk

Jolanta Pokorska

Helion Edukacja

Podręcznik dopuszczony do użytku szkolnego przez ministra właściwego do spraw oświaty i wychowania i wpisany do wykazu podręczników przeznaczonych do kształcenia w zawodzie technik informatyk, na podstawie opinii rzeczoznawców: mgr Marii Dziurzyńskiej-Ścibior, mgr Elżbiety Leszczyńskiej, dr. inż. Stanisława Szabłowskiego.

Nazwa kwalifikacji: Kwalifikacja E-14. Część 2. Tworzenie baz danych i administrowanie bazami.
Typ szkoły: technikum, szkoła policealna, kurs kwalifikacyjny.
Rok dopuszczenia 2013.

Numer ewidencyjny w wykazie: 50/2013

Redaktorzy prowadzący: Marcin Borecki
Projekt okładki: Maciej Pasek

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?e14te2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-246-6510-5

Printed in Poland.

Spis treści

5

SQL — strukturalny język zapytań

5.1. Wprowadzenie

Język SQL (ang. *Structured Query Language*) jest 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.

5.2. 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) oraz Międzynarodowa Organizacja Normalizacyjna (ang. *International Organization for Standardization*— ISO) opracowują i publikują standardy języka SQL. W roku 1999 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 roku 2003 został opublikowany czwarty standard języka SQL, który jest rozszerzeniem SQL3. Zawiera on następujące rozszerzenia:

- obsługa nowych typów danych,

- rozszerzenia w sposobie wywoływania procedur,
- poszerzona instrukcja CREATE TABLE,
- funkcje rankingu,
- retrospektywne sprawdzanie więzów integralności.

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

Najbardziej znane dialekty języka SQL to:

- T-SQL (ang. *Transact-SQL*) — stosowany na serwerach Microsoft SQL Server i Sybase Adaptive Server;
- 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.

5.3. Składnia języka SQL

Język SQL realizuje trzy podstawowe typy zadań — definiowanie danych, manipulowanie danymi i kontrolowanie danych. W związku z tym jego instrukcje można podzielić na trzy kategorie:

- 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.

UWAGA

Żaden z systemów bazodanowych nie jest w pełni zgodny ze standardem języka SQL. Na potrzeby podręcznika został wybrany SQL Server 2008 Express Edition firmy Microsoft. Przedstawiane w przykładach polecenia języka SQL w większości wykorzystują jego standardowe cechy i prawdopodobnie będą obsługiwane również w innych systemach bazodanowych.

5.3.1. Instrukcje języka SQL

Instrukcje języka SQL składają się z wyrażeń, klauzul i warunków. 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 5.1

```
SELECT nazwisko, Imię
FROM Klient
WHERE miejscowosc="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. Taką składnię języka już poznaliśmy w rozdziale 2., definiując w programie Access kwerendy z użyciem języka SQL.

Składnia języka jest zbiorem reguł, których należy przestrzegać. W języku SQL stosuje się trzy kategorie pojęć składniowych: identyfikatory, literały i operatory.

Identyfikator jednoznacznie definiuje obiekt bazy danych. Każdy obiekt bazy danych (baza, tabela, kolumna) musi posiadać 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 dopuszczalną 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) — `IN` (przynależność do listy wartości), `BETWEEN...AND...` (przynależność do domkniętego przedziału), `LIKE` (zgodność ze wzorem).

Słowa kluczowe to wyrazy zastrzeżone, interpretowane w ściśle określony sposób. W systemie baz danych mają dokładnie określone znaczenie. Do słów kluczowych należą:

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

5.3.2. Typy danych

Typy danych, jak wiemy, 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 (tabela 5.1) na:

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

Tabela 5.1. Typy danych w MS SQL Server

Kategoria	Typ danych
typy liczbowe	`int`, `smallint`, `bigint`, `tinyint`, `float`, `real`, `decimal`, `numeric`
typy daty i czasu	`datetime`, `smalldatetime`
typy znakowe	`char`, `varchar`, `nchar`, `ntext`, `nvarchar`
typy waluty	`money`, `smallmoney`
typy binarne	`binary`, `varbinary`
typy specjalne	`text`, `image`, `xml`, `bit`

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`.

5.3.3. 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 pełna 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*.

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.

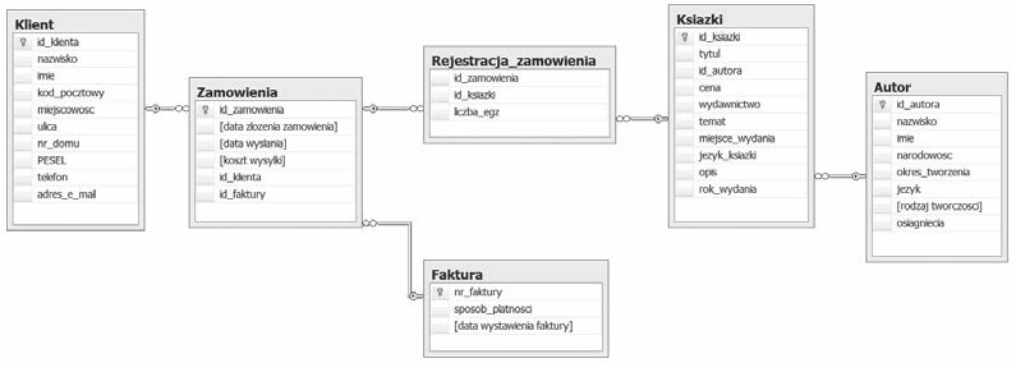
5.4. Język definiowania danych (DDL)

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 używanego 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 internetowa* po niewielkich modyfikacjach (rozdział 2., przykład 1.2). Struktura bazy danych *Księgarnia internetowa* została pokazana na rysunku 5.1.



Rysunek 5.1. Baza danych Księgarnia internetowa

Tworzenie bazy danych jest realizowane za pomocą instrukcji:

```
CREATE DATABASE nazwa_bazy
```

Przykład 5.2

```
CREATE DATABASE ksiegarnia_internetowa;
```

Tak samo nieskomplikowanie wygląda polecenie usuwające bazę danych:

```
DROP DATABASE nazwa_bazy
```

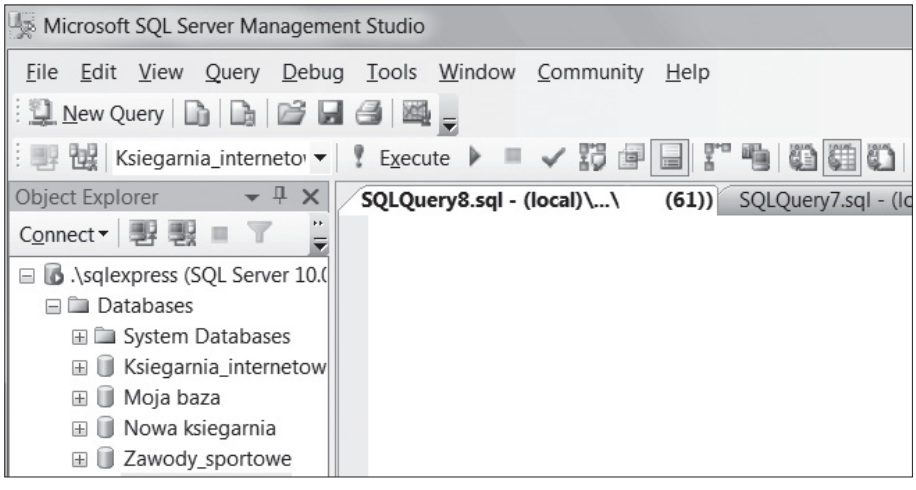
Przykład 5.3

```
DROP DATABASE ksiegarnia_internetowa;
```

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.

W praktyce te i wszystkie inne operacje można wykonać za pomocą SQL Server Management Studio bez znajomości poleceń języka SQL.

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 5.2).



Rysunek 5.2. Okno tworzenia zapytań dla bazy danych

Przykład 5.4

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

Klikamy przycisk *New Query* i w otwartym oknie wprowadzamy poniższy skrypt:

```
CREATE DATABASE ksiegarnia_internetowa;
```

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

5.4.1. Tworzenie i usuwanie tabel

Po utworzeniu bazy danych następnym etapem pracy jest tworzenie tabel oraz 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],
)
```

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.

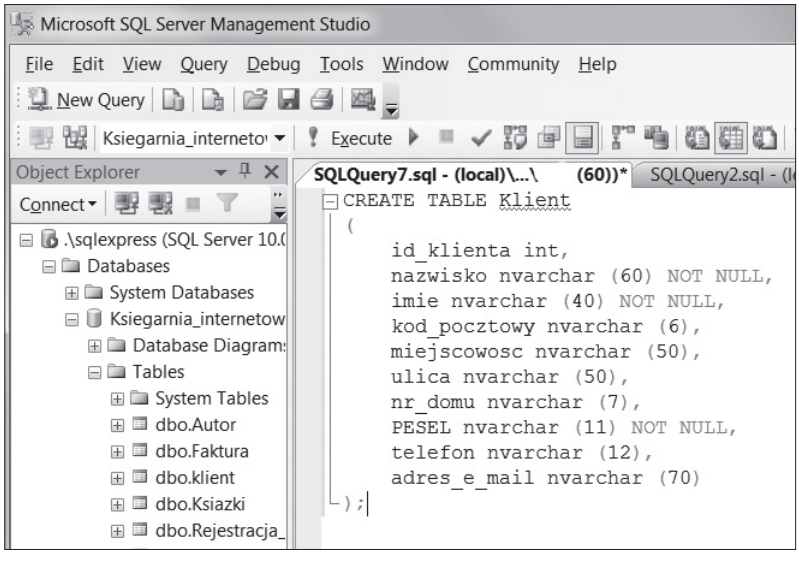
Przykład 5.5

Dla bazy danych *ksiegarnia_internetowa* utwórz tabelę *Klient* z kolumnami takimi jak widoczne na rysunku 5.1. Ustaw odpowiednie typy dla kolumn tworzonej tabeli.

Podobnie jak w przykładzie 5.4, klikamy przycisk *New Query* i w otwartym oknie wprowadzamy podany niżej skrypt:

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

Po wprowadzeniu skryptu klikamy przycisk *Execute* (lub *F5* — rysunek 5.3). 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`.



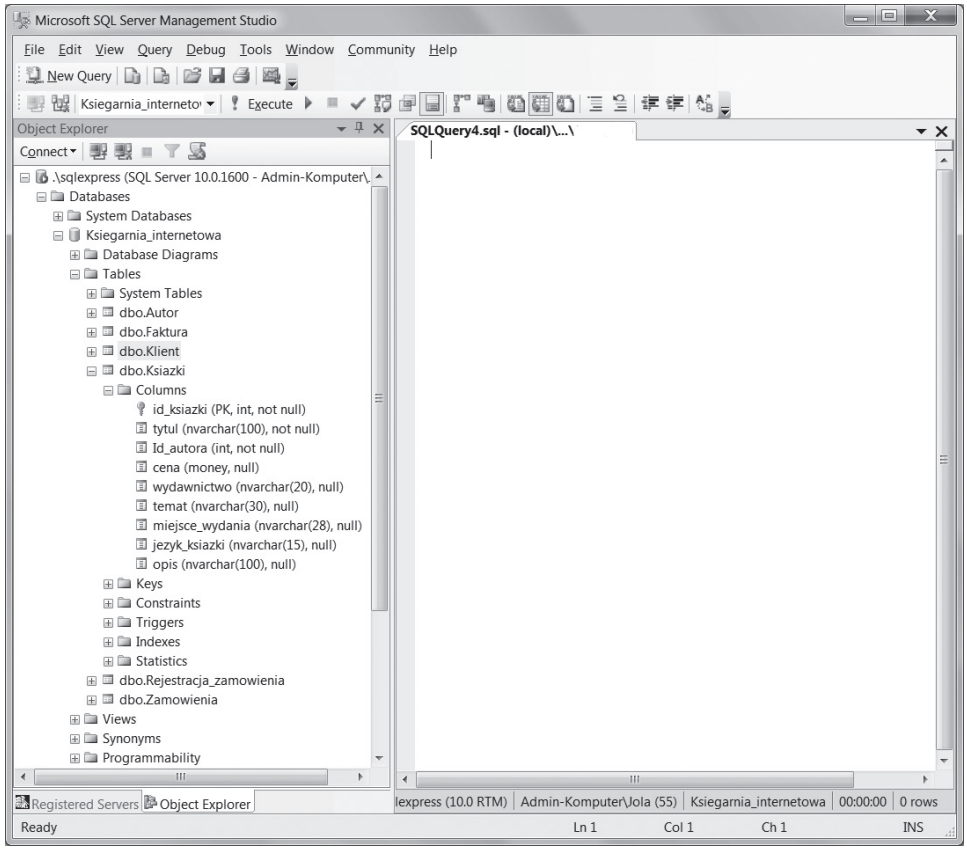
Rysunek 5.3. Definiowanie polecenia tworzącego tabelę Klient

Przykład 5.6

Utwórz tabelę *Ksiazki* z kolumnami jak na rysunku 5.1. Polecenie tworzące tabelę będzie miało postać:

```
CREATE TABLE 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),
);
```

Rysunek 5.4 pokazuje strukturę bazy danych i tabel w oknie *Server Management Studio*.



Rysunek 5.4. Struktura bazy danych ksiegarnia_internetowa

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

```
DROP TABLE nazwa_tabeli
```

Przykład 5.7

```
DROP TABLE Klient;
```

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

5.4.2. Schematy

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 5.8

```
CREATE SCHEMA Zasoby;
```

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

Przykład 5.9

```
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*. Ale gdyby zalogowany użytkownik znajdował się domyślnie w schemacie *Zasoby*, to tabela zostałaby dodana do schematu *Zasoby*. Niejawne przypisywanie do schematów nie jest zalecane.

Przykład 5.10

```
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:

```
DEFAULT_SCHEMA CREATE USER
```

lub:

```
ALTER USER
```

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

5.4.3. 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
```

Przykład 5.11

Dodanie kolumny do tabeli *Ksiazki*:

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

Przykład 5.12

Zmiana definicji istniejącej kolumny w tabeli *Ksiazki*:

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

Przykład 5.13

Usunięcie kolumny z tabeli *Ksiazki*:

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

Przykład 5.14

Usunięcie z tabeli *ksiegarnia_internetowa* kolumny *autor* i dodanie kolumny *id_autora*:

```
Use ksiegarnia_internetowa;
GO
ALTER TABLE Ksiazki
DROP COLUMN Autor;
ALTER TABLE Ksiazki
ADD id_autora int NOT NULL;
```

5.4.4. 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.

Przykład 5.15

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
    );
```

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`.

Można również jawnie zezwolić na wprowadzanie do kolumny wartości NULL, wpisując po jej nazwie słowo NULL.

Przykład 5.16

Blokowanie wartości NULL podczas tworzenia tabeli *Ksiazki*:

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

IDENTITY

Atrybut IDENTITY oznacza automatyczny wzrost wartości w kolumnie, dla której został zdefiniowany. Na przykład IDENTITY (1,1) oznacza wzrost wartości kolumny o 1, począwszy od wartości 1. Niemożliwe jest nadawanie atrybutu IDENTITY istniejącej kolumnie.

Przykład 5.17

Definiowanie ograniczeń dla kolumn podczas tworzenia tabeli *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 (100));
```

DEFAULT

Atrybut DEFAULT jest stosowany do wprowadzania do kolumny wartości domyślnej.

Przykład 5.18

Definiowanie wartości domyślnej podczas tworzenia tabeli *Ksiazki*:

```
CREATE TABLE Ksiazki
(
    ....
    rok_wydania nvarchar (4) NOT NULL DEFAULT '2012'
    ....
);
```

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`.

Przykład 5.19

Definiowanie wartości unikatowych podczas tworzenia tabeli *Ksiazki*:

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

    ....
);
```

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 5.20

Definiowanie ograniczeń podczas tworzenia tabeli *Ksiazki*:

```
CREATE TABLE Ksiazki
(
    . . . .
    rok_wydania int CHECK (BETWEEN 2010 AND 2014)
    . . . .
);
```

Przykład 5.21

Definiowanie atrybutów ograniczających podczas tworzenia tabeli *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)
);
```

Przykład 5.22

Definiowanie atrybutów ograniczających podczas tworzenia tabeli *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),
);
GO
ALTER TABLE Ksiazki
ADD rok_wydania nvarchar (4) NOT NULL DEFAULT '2012';
```

Zadanie 5.1

W bazie danych *ksiegarnia_internetowa* utwórz tabele zgodnie ze schematem zamieszczonym na rysunku 5.1. Określ dopuszczalne wartości w kolumnach tabel i zdefiniuj klucze podstawowe.

5.5. Manipulowanie danymi (DML)

Jednym z zadań realizowanych przez język SQL jest pobieranie i modyfikowanie danych. Instrukcje do tego służące tworzą tak zwany język manipulacji 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.

5.5.1. 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 5.23

```
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 wstawiony wiersz odczytamy, 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 podobnie 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 nieznanej (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 5.24

```
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 5.25

```
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.

5.5.2. Instrukcja UPDATE

Do aktualizowania danych służy instrukcja UPDATE w postaci:

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

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 5.26

```
UPDATE Ksiazki SET [jezyk ksiazki] ='polski'
WHERE [rok wydania]>=2012;
```

W podanym przykładzie dla tabeli *Ksiazki* kolumnie *jezyk ksiazki* zostanie przypisana wartość *polski*, ale tylko dla książek wydanych w roku 2012 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 5.27

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

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

5.5.3. Instrukcja DELETE

Instrukcja DELETE usuwa wiersze z wybranej tabeli.

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

Przykład 5.28

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 5.29

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.

5.5.4. Instrukcja SELECT

Instrukcja SELECT określa, jakie dane zostaną zwrócone w wyniku jej wykonania. Ogólną postać instrukcji SELECT definiującej kwerendę wybierającą w programie Access poznaliśmy w rozdziale 3. Jej najprostsza postać dla serwera SQL Server wygląda następująco:

```
SELECT [ALL | DISTINCT] [TOP n [PERCENT] ]
{nazwa_kolumny | wyrażenie | IDENTITYCOL | ROWGUIDCOL} [[AS] nazwa_kolumny]
| nazwa_kolumny = wyrażenie } [, ... n]
FROM Nazwa_tabeli
```

gdzie:

`IDENTITYCOL` zwraca wartość kolumny *IDENTITY*,

`ROWGUIDCOL` zwraca wartość globalnego identyfikatora.

Przykład 5.30

```
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 wypisywania listy wszystkich pól tabeli można użyć symbolu `*`.

Przykład 5.31

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.

Klauzula DISTINCT

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

Przykład 5.32

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

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

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 5.33

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

```
SELECT tytul, cena, cena*0.07 AS [Marża]
FROM Ksiazki;
```

Wynik zostanie obliczony dla wszystkich wierszy tabeli *Ksiazki*.

Przykład 5.34

Chcemy uzyskać ostateczną cenę książki, uwzględniającą marżę.

```
SELECT tytul, cena+cena*0.07 AS [Cena sprzedaży]
FROM Ksiazki;
```

Przykład 5.35

Chcemy połączyć dane klienta z kilku kolumn tabeli *Klient*.

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

Sortowanie

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

Przykład 5.36

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

Książki zostaną posortowane według kolumny *tytul*.

Inny przykład:

```
SELECT tytul, 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 5.37

```
SELECT tytul, 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 5.38

```
SELECT tytul, rok_wydania, cena
FROM Ksiazki
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`.

Wybieranie wierszy — klauzula WHERE

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 5.39

```
SELECT tytul, cena
FROM Ksiazki
WHERE rok_wydania=2012;
```

lub:

```
SELECT tytul, cena
FROM Ksiazki
WHERE cena BETWEEN 50 AND 100;
```

Przykład 5.40

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

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

Przykład 5.41

Chcemy odczytać nazwiska klientów, którzy nie podali swojego adresu e-mail.

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

Przykład 5.42

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` oraz `DELETE`. Składnia jej jest we wszystkich przypadkach taka sama.

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.

Przykład 5.43

```
SELECT TOP 1 tytul, 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 tylko wtedy, gdy w bazie danych jest tylko jedna książka z tą ceną. A jeżeli książek z tą samą ceną jest więcej?

Wtedy 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 5.44

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

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

5.5.5. Grupowanie danych

Funkcje agregujące

Funkcje agregujące działają na wartościach wybranego pola w grupie wierszy. Wynikiem może być suma, średnia, liczba wierszy, wybranie wartości maksymalnej lub minimalnej. Funkcje te zwracają pojedyncze wartości i są wywoływane w instrukcji `SELECT`. Funkcje agregujące są jednym 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 5.45

```
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_internetowa*?”. 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 5.46

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

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

Przykład 5.47

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

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 5.48

```
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 5.49

```
SELECT COUNT (tytul) AS 'Liczba tytułów'
FROM Ksiazki
WHERE rok_wydania>2008;
```

Wynikiem będzie liczba książek wydanych po roku 2008.

Funkcje agregujące mogą być częścią wyrażeń.

Przykład 5.50

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

W wyniku otrzymamy różnicę między maksymalną i minimalną ceną książki.

Klauzula GROUP BY

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

Przykład 5.51

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.

Klauzula HAVING

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

Przykład 5.52

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

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

Istotne jest zrozumienie różnicy między klauzulami `WHERE` oraz `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.

5.6. 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".

5.6.1. 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 5.53

Mamy prześledzić historię zamówień w księgarni internetowej w czerwcu 2012 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 '2012-06-1' AND
'2012-06-30';
```

Aby prawidłowo utworzyć zapytanie, należy zdefiniować połączenia między tabelami bazy danych. Definicji połączeń można używać wielokrotnie, chociaż może to wpłynąć 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 w 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.

Gdybyśmy chcieli uzyskać listę wszystkich zamówień zrealizowanych w księgarni internetowej wraz z numerami wystawionych faktur, to przy zastosowaniu domyślnego połączenia między tabelami (`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 5.54

```
SELECT Zamowienia.[data zlozenia zamowienia], Zamowienia.[koszt wysylki],
Faktura.nr_faktury, Faktura.sposob_platnosci
FROM Zamowienia LEFT OUTER JOIN Faktura
ON Zamowienia.nr_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.

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

Przykład 5.55

Jeżeli przyjmiemy, że w tabeli *Zamowienia* znajdują się zamówienia, dla których nie zostały wystawione faktury, a w tabeli *Faktury* 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],
Faktura.nr_faktury, Faktura.sposob_platnosci
FROM Zamowienia FULL OUTER JOIN Faktura
ON Zamowienia.nr_faktury=Faktura.nr_faktury;
```

5.6.2. 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 5.56

```
SELECT Klient.nazwisko, Klient.imie, Zamowienia.[data zlozenia zamowienia],
Zamowienia.[koszt wysylki]
FROM Klient CROSS JOIN Zamowienia;
```

5.6.3. 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 obsługiwania 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.

5.6.4. 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 5.57

```
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 5.58

```
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;
```

5.7. Więzy integralności

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

5.7.1. Definiowanie klucza obcego

Jeżeli w bazie danych *ksiegarnia_internetowa* 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;

Skorowidz

E

F

G

H

I

J

K

S

T

U

V

W

X

Z

Kwalifikacja E.14

Tworzenie baz danych i administrowanie bazami

Część 2

Podręcznik do nauki zawodu technik informatyk

Technik informatyk nie jest zwykłym użytkownikiem komputerów. Jeśli uczeń wybiera szkołę kształcącą w tym zawodzie, z czasem staje się prawdziwym komputerowym ekspertem.

Kwalifikacja E.14 dotyczy tematów związanych z tworzeniem aplikacji internetowych oraz projektowaniem i tworzeniem baz danych. Pierwszy podręcznik z tej serii zawierał wiedzę z zakresu projektowania i tworzenia stron internetowych. Drugi, *Tworzenie baz danych i administrowanie bazami*, wyjaśnia zasady projektowania baz danych i zarządzania tymi bazami. Podręcznik jest zgodny z najnowszą podstawą programową. Dzięki realizacji zawartych w nim zadań uczniowie zaczną bez problemu identyfikować terminy związane z relacyjnymi bazami danych, będą stosować zasady normalizacji tabel oraz poznają funkcje programu MS Access. Przyszli technicy informatycy poznają pojęcia związane z architekturą systemów bazodanowych. Nauczą się także instalować i konfigurować serwery baz danych oraz nimi zarządzać. Informacje dotyczące języka SQL ułatwią im tworzenie bazy danych oraz pozwolą na odczytywanie i modyfikowanie zawartych w niej danych, a także kontrolowanie dostępu do nich. Uczniowie zapoznają się też z zasadami stosowania transakcji, procedur składowanych oraz wyzwalaczy. Podręcznik składa się z sześciu części, a jego budowa pozwala na realizowanie treści programowych w sposób wybrany przez nauczyciela.

Technik informatyk to doskonały, charakteryzujący się wysoką jakością, kompletny zestaw edukacyjny, przygotowany przez dysponującego ogromnym doświadczeniem lidera na rynku książek informatycznych – wydawnictwo Helion.

W skład zestawu *Technik informatyk* wchodzą także:

Kwalifikacja E.12. Montaż i eksploatacja komputerów osobistych oraz urządzeń peryferyjnych. Podręcznik do nauki zawodu technik informatyk

Kwalifikacja E.13. Projektowanie lokalnych sieci komputerowych i administrowanie sieciami. Podręcznik do nauki zawodu technik informatyk

Kwalifikacja E.14. Część 1. Tworzenie stron internetowych. Podręcznik do nauki zawodu technik informatyk

Kwalifikacja E.14. Część 3. Aplikacje internetowe. Podręcznik do nauki zawodu technik informatyk

Podręczniki oraz inne pomoce naukowe należące do tej serii zostały opracowane z myślą o wykształceniu kompetentnych techników, którzy świetnie poradzą sobie z trudnymi zadaniami w świecie współczesnej informatyki. Według nowych przepisów, aby otrzymać tytuł technika informatyka, należy potwierdzić trzy kwalifikacje wyodrębnione w tym zawodzie – to niewątpliwie wyzwanie i dla adeptów nauki o komputerach, i dla ich pedagogów. Ta książka pozwoli zarówno przygotować się do egzaminów, jak i uzyskać wiedzę i umiejętności oczekiwane na rynku pracy lub przydatne w przyszłej pracy.

helion.pl
księgarnia internetowa

Nr katalogowy: 12163

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-246-6510-5

Informatyka w najlepszym wydaniu