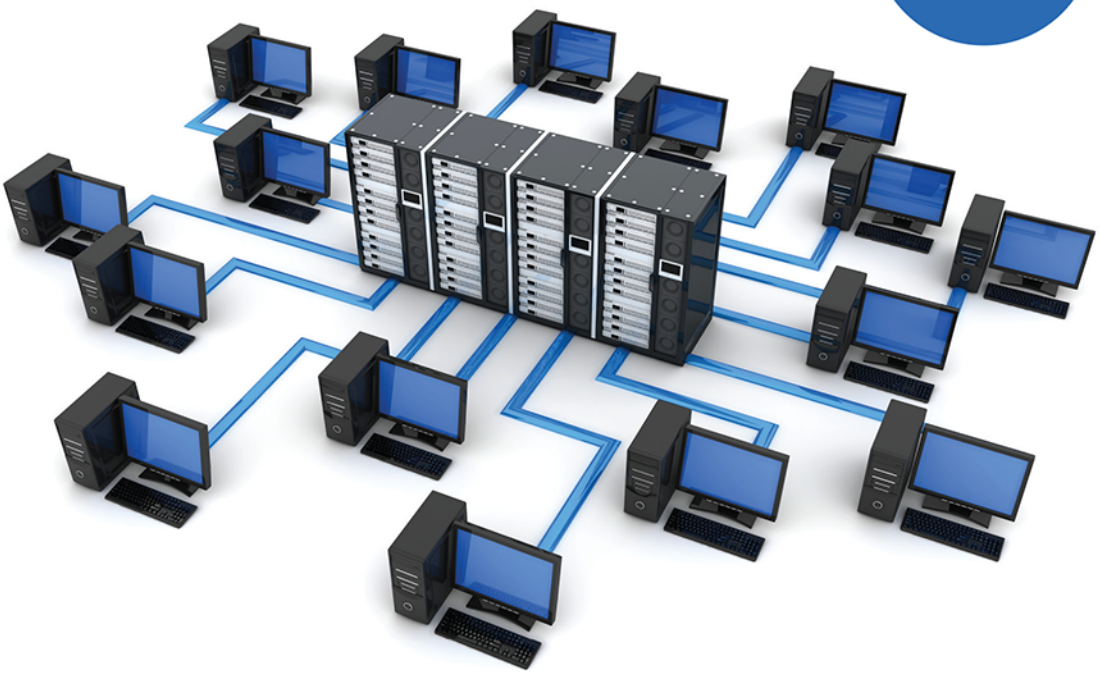
Wydanie VI



Ryan Stephens, Arie D. Jones, Ron Plew

SQL

w 24 godziny

SAMS

Helion

Tytuł oryginału: Sams Teach Yourself SQL in 24 Hours, Sixth Edition

Tłumaczenie: Grzegorz Kowalczyk
Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-283-2590-6

Authorized translation from the English language edition: SQL IN 24 HOURS, SAMS TEACH YOURSELF, Sixth Edition,; ISBN 0672337592; by Ryan Stephens; and by Arie D. Jones; and by Ron Plew; published by Pearson Education, Inc, publishing as SAMS Publishing.

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)
Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/sqlw24.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/sqlw24
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Godzina 3.

Zarządzanie obiektami bazy danych

W ciągu tej godziny dowiesz się:

- **Obiekty bazy danych**
- **Schematy**
- **Tabele**
- **Natura i atrybuty tabel**
- **Przykłady tworzenia i modyfikowania tabel**
- **Opcje przechowywania tabel**
- **Więzy integralności i spójność danych**

W ciągu tej godziny będziemy mówić o obiektach bazy danych: czym są, jak się zachowują, jak są przechowywane i jakie zachodzą między nimi relacje. Obiekty bazy danych to jednostki logiczne tworzące elementy składowe bazy danych. Co prawda większość zagadnień omawianych w tej godzinie będzie dotyczyła tabel, ale pamiętaj, że w bazie danych istnieją również inne obiekty. Wiele z nich będziemy bardziej szczegółowo omawiać w kolejnych godzinach.

Obiekty bazy danych i schematy

Obiekt bazy danych to każdy obiekt zdefiniowany w bazie danych, który jest wykorzystywany do przechowywania danych lub odwoływania się do nich. Przykładami obiektów bazy danych są tabele, widoki, klastry, sekwencje, indeksy i synonimy. W ciągu tej godziny skoncentrujemy się głównie na tabelach, ponieważ jest to podstawowa i najprostsza forma przechowywania danych w relacyjnych bazach danych.

Schemat (ang. *schema*) to kolekcja obiektów bazy danych zazwyczaj powiązanych z jednym, konkretnym kontem użytkownika. Taki użytkownik jest nazywany **właścicielem schematu** (ang. *schema owner*), a inaczej mówiąc, jest on właścicielem odpowiadającej mu grupy obiektów. W bazie danych może znajdować się jeden lub wiele schematów. Konto użytkownika jest powiązane ze schematem o tej samej nazwie i bardzo często oba pojęcia są używane wymienne. Jeżeli użytkownik tworzy jakiś obiekt, to taki obiekt jest domyślnie tworzony w schemacie tego użytkownika, o ile w poleceniu nie znalazła się

odpowiednia instrukcja stanowiąca inaczej. Możemy zatem powiedzieć, że w oparciu o uprawnienia w bazie danych użytkownik ma pełną kontrolę nad tworzeniem, modyfikowaniem i usuwaniem obiektów. Schemat może się składać zarówno z jednego obiektu, jak i z wielu obiektów, przy czym liczba obiektów wchodzących w skład schematu teoretycznie jest nieograniczona, o ile oczywiście takich ograniczeń nie wprowadza określona implementacja bazy danych.

Załóżmy, że administrator bazy danych utworzył dla Ciebie konto użytkownika o nazwie USER1 i ustawił odpowiednie hasło dostępu. Po uzyskaniu dostępu zalogowałeś się do bazy danych i utworzyłeś tabelę o nazwie EMPLOYEE_TBL. Zgodnie z informacjami w bazie danych Twoja nowo utworzona tabela nosi nazwę USER1.EMPLOYEE_TBL. Nazwa schematu dla tej tabeli to USER1 i ten użytkownik jest jednocześnie właścicielem tej tabeli. Właśnie utworzyłeś w swoim schemacie pierwszą tabelę.

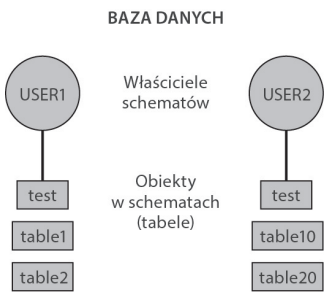
Dobrą cechą schematów jest to, że kiedy korzystasz z tabel w swoim schemacie, których jesteś właścicielem, to odwołując się do nich, nie musisz podawać nazwy schematu. Na przykład do naszej tabeli możesz odwoływać się na jeden z dwóch sposobów:

```
EMPLOYEE_TBL
USER1.EMPLOYEE_TBL
```

Oczywiście preferowanym rozwiązaniem jest pierwsza z tych opcji, ponieważ jej wpisanie wymaga mniejszej liczby naciśniętych klawiszy. Jeżeli jednak jakiś inny użytkownik będzie chciał się odwołać do tej samej tabeli, to będzie musiał podać jej pełną nazwę zawierającą nazwę schematu:

```
USER1.EMPLOYEE_TBL
```

W godzinie 20. „Tworzenie i używanie widoków i synonimów” będziemy omawiać zagadnienia związane z nadawaniem użytkownikom odpowiednich uprawnień umożliwiających dostęp do tabel innych użytkowników. Dowiesz się również, jak działają synonimy, które pozwalają przypisywać tabelom inne nazwy, dzięki którym korzystając z takich tabel, możesz uniknąć konieczności podawania nazw schematów. Rysunek 3.1 przedstawia dwa schematy utworzone w relacyjnej bazie danych.



RYSUNEK 3.1. Schematy w bazie danych

Rysunek 3.1 przedstawia dwa konta użytkowników, USER1 i USER2, będących właścicielami tabel. Każdy z użytkowników posiada swój własny schemat. Oto kilka przykładów ilustrujących, w jaki sposób użytkownicy mogą się odwoływać do swoich tabel i do tabel drugiego użytkownika:

```
USER1 odwołuje się do swojej tabeli TABLE1:                    TABLE1
USER1 odwołuje się do swojej tabeli TEST:                      TEST
USER1 odwołuje się do tabeli TABLE10 użytkownika USER2:        USER2.TABLE10
USER1 odwołuje się do tabeli TEST użytkownika USER2:           USER2.TEST
```

W naszym przykładzie obaj użytkownicy posiadają swoje tabele o nazwie TEST. Tabele w bazie danych mogą mieć takie same nazwy, o ile znajdują się w różnych schematach. Jest to możliwe, ponieważ z punktu widzenia bazy nazwa schematu jest częścią nazwy tabeli, co powoduje, że pełna nazwa tabeli jest zawsze unikatowa. Na przykład tabela USER1.TEST to zupełnie inna tabela niż USER2.TEST. Jeżeli odwołując się do tabeli, pominiesz nazwę schematu, serwer bazy danych domyślnie będzie jej poszukiwał wśród tabel, których jesteś właścicielem. Inaczej mówiąc: jeżeli użytkownik USER1 będzie się odwoływał do tabeli TEST, to serwer bazy danych najpierw będzie jej poszukiwał wśród tabel, których ten użytkownik jest właścicielem, a dopiero później wśród innych obiektów użytkownika USER1, takich jak synonimy wskazujące tabele w innych schematach. W godzinie 21. „Praca z katalogiem systemowym" znajdziesz omówienie wielu zagadnień, które pozwolą Ci lepiej zrozumieć, jak działają synonimy.

Musisz dobrze zrozumieć, na czym polega różnica między obiektami w Twoim schemacie a obiektami w schematach innych użytkowników. Jeżeli nie określisz docelowego schematu podczas wykonywania operacji modyfikujących tabele, takich jak polecenie DROP, baza danych zakłada, że miałeś na myśli tabelę z Twojego własnego schematu, co w pewnych okolicznościach może doprowadzić na przykład do niezamierzonego usunięcia niewłaściwego obiektu. Pracując z bazą danych, zawsze pamiętaj, na jakie konto użytkownika jesteś zalogowany.

Ostrzeżenie

Ostrzeżenie

> **Reguły tworzenia nazw obiektów w poszczególnych systemach mogą być różne**
>
> Każdy serwer baz danych ma odrębny zestaw wytycznych określających reguły nadawania nazw obiektom i elementom składowym tych obiektów, takich jak nazwy pól. Szczegółowe informacje na temat konwencji nadawania nazw znajdziesz w dokumentacji danej implementacji.

Tabele — podstawowe miejsce przechowywania danych

Tabela to podstawowy obiekt służący do przechowywania danych w relacyjnej bazie danych. W swojej najprostszej formie tabela składa się z wierszy i kolumn, w których przechowywane są dane. Tabele fizycznie zajmują miejsce w bazie danych i mogą być stałe lub tymczasowe.

Kolumny

Pole (ang. *field*), nazywane również **kolumną**, to część tabeli, w której przechowywane są dane określonego typu. Typ danych przypisany do kolumny determinuje rodzaj danych, jakie mogą być w niej przechowywane. Takie rozwiązanie ułatwia projektantowi tabeli utrzymanie spójności przechowywanych w niej danych.

Każda tabela bazy danych musi się składać z co najmniej jednej kolumny. Kolumny to elementy tabeli, które przechowują dane określonego typu, takie jak nazwiska czy adresy. Na przykład poprawną kolumną tabeli będzie kolumna zawierająca nazwisko klienta. Rysunek 3.2 przedstawia przykładową kolumnę tabeli.

EMP_ID	LAST_NAME	FIRST_NAME	MIDDLE_NAME
213764555	GLASS	BRANDON	SCOTT
220984332	WALLACE	MARIAH	NULL
311549302	STEPHENS	TINA	DAWN
313782439	GLASS	JACOB	NULL
442346889	PLEW	LINDA	CAROL
443679012	SPURGEON	TIFFANY	NULL

RYSUNEK 3.2. Przykładowa kolumna tabeli

Ogólnie rzecz biorąc, nazwa kolumny musi mieć postać pozbawionego spacji łańcucha znaków, a jej maksymalny rozmiar może podlegać różnym ograniczeniom w poszczególnych implementacjach SQL. Powszechnie przyjętą praktyką jest zastępowanie spacji w nazwach kolumn znakami podkreślenia. Na przykład kolumna, w której przechowywane są nazwiska klientów, może zamiast `CUSTOMERNAME` nosić nazwę `CUSTOMER_NAME`. Nietrudno zauważyć, że takie rozwiązanie znacząco wpływa na zwiększenie czytelności nazw obiektów bazy danych. Istnieje również wiele innych konwencji nadawania nazw, takich jak choćby notacja CamelCase, często stosowana także w innych językach programowania. Z tego względu niezmiernie ważne jest, aby zespół deweloperów uzgodnił konwencję nadawania nazw obiektom bazy danych i konsekwentnie jej przestrzegał podczas projektowania bazy i aplikacji.

Najczęściej spotykaną formą danych przechowywanych w kolumnach tabeli są ciągi znaków. Takie dane mogą być zapisywane w polach znakowych zarówno przy użyciu małych, jak i wielkich liter. Wielkość liter używanych do zapisywania danych tekstowych w bazie danych jest kwestią umowną i powinna być wybierana pod kątem tego, w jaki sposób takie dane będą później wykorzystywane. Bardzo często zdarza się na przykład, że w celu uproszczenia całego procesu i zachowania spójności dane tekstowe są przechowywane od razu w postaci wielkich liter. Z drugiej strony, jeżeli w bazie danych znajdują się dane tekstowe zapisane przy użyciu zarówno małych, jak i wielkich liter, w razie potrzeby możemy użyć odpowiednich funkcji konwertujących ciągi znaków wielkich czy małych liter. Więcej szczegółowych informacji na temat takich funkcji znajdziesz w godzinie 11. „Modyfikowanie wyglądu wyników działania zapytania”.

Kolumny mogą być również zdefiniowane jako NULL lub NOT NULL. Jeżeli kolumna zostanie oznaczona jako NOT NULL, to nie będzie możliwe pozostawienie w niej pustego pola. Jeżeli zaś kolumna zostanie oznaczona jako NULL, to wprowadzanie do niej danych nie będzie wymagane. Wartość NULL jest diametralnie inna od wartości pustej, takiej jak na przykład pusty ciąg znaków i zajmuje specjalne miejsce w procesie tworzenia baz danych. Do wartości NULL można się odnosić jak do braku jakichkolwiek danych w polu.

Wiersze

Wiersz to inaczej rekord danych zapisany w tabeli. Na przykład wiersz danych w tabeli klientów może składać się z numeru identyfikacyjnego klienta, jego nazwy, adresu, numeru telefonu i numeru faksu. Wiersz składa się z szeregu pól zawierających dane jednego rekordu w tabeli. W tabeli równie dobrze może znajdować się tylko jeden rekord jak i kilka czy kilkanaście milionów rekordów (czy, jak kto woli, wierszy danych). Rysunek 3.3 przedstawia przykładowy wiersz tabeli.

EMP_ID	LAST_NAME	FIRST_NAME	MIDDLE_NAME
213764555	GLASS	BRANDON	SCOTT
220984332	WALLACE	MARIAH	NULL
311549902	STEPHENS	TINA	DAWN
313782439	GLASS	JACOB	NULL
442346889	PLEW	LINDA	CAROL
443679012	SPURGEON	TIFFANY	NULL

RYSUNEK 3.3. Przykładowy wiersz tabeli

Polecenie CREATE TABLE

Polecenie CREATE TABLE języka SQL jest używane do tworzenia tabel. Choć sam proces tworzenia tabeli jest relatywnie prosty, to jednak przed samym wykonaniem polecenia CREATE TABLE należy poświęcić wystarczającą ilość czasu i zasobów na odpowiednie zaprojektowanie struktury tabeli. Dzięki takiemu podejściu możemy później uniknąć konieczności rekonfiguracji tabeli podczas uruchamiania i wdrażania aplikacji.

Uwaga

Typy danych, których będziemy używać w tej godzinie

W przykładach omawianych w tej godzinie będziemy korzystać z popularnych typów danych, takich jak CHAR (typ znakowy o stałej długości), VARCHAR (typ znakowy o zmiennej długości), NUMBER (wartości numeryczne, liczby dziesiętne i liczby całkowite) oraz DATE (wartości daty i czasu).

Podczas tworzenia tabeli musimy sobie odpowiedzieć na szereg podstawowych pytań:

- Jakie rodzaje danych będą wprowadzane do tabeli?
- Jaka będzie nazwa tabeli?
- Która kolumna (lub kolumny) będzie pełnić rolę klucza głównego?
- Jakie będą nazwy poszczególnych kolumn (pól)?
- Jakie typy danych będą przypisane do poszczególnych kolumn?
- Jakie będą rozmiary poszczególnych kolumn?
- W których kolumnach będą dopuszczalne wartości puste (NULL)?

Uwaga

> **Istniejący system zazwyczaj ma już przyjętą konwencję nadawania nazw**
>
> Przed rozpoczęciem nadawania nazw obiektów i innych elementów istniejącej bazy danych należy zawsze sprawdzić zaimplementowane w niej reguły. Bardzo często administratorzy baz danych przyjmują określoną *konwencję nazewnictwa obiektów bazy danych*, która szczegółowo objaśnia i reguluje zasady nadawania nazw obiektom w takiej bazie.

Po udzieleniu odpowiedzi na wymienione wcześniej pytania przygotowanie odpowiedniego polecenia CREATE TABLE nie jest trudne. Podstawowa składania tego polecenia jest następująca:

```
CREATE TABLE nazwa_tabeli
(pole1 typ_danych [NOT NULL],
pole2 typ_danych [NOT NULL],
pole3 typ_danych [NOT NULL],
pole4 typ_danych [NOT NULL],
pole5 typ_danych [NOT NULL]);
```

Zauważ, że ostatnim znakiem naszego polecenia jest średnik. Nawiasy kwadratowe wskazują opcjonalne części składni polecenia. Większość implementacji SQL wykorzystuje określone znaki do oznaczania końca polecenia czy do przesłania polecenia do serwera bazy danych. Oracle, Microsoft SQL Server oraz MySQL używają do tego celu średnika. W przypadku języka Transact-SQL, czyli zgodnej ze standardem ANSI wersji języka SQL, wykorzystywanej w implementacjach Microsoft SQL Server, nie ma co prawda takiego wymagania, ale stosowanie takiej konwencji zapisu poleceń jest uznawane za dobrą praktykę. W naszej książce wszystkie polecenia SQL będą się kończyły średnikiem.

W przykładzie przedstawionym poniżej tworzymy tabelę o nazwie EMPLOYEE_TBL (składnia polecenia dla bazy MySQL):

```
CREATE TABLE EMPLOYEE_TBL
   (EMP_ID        VARCHAR (9)    NOT NULL,
    EMP_NAME      VARCHAR (40)   NOT NULL,
    EMP_ST_ADDR   VARCHAR (20)   NOT NULL,
    EMP_CITY      VARCHAR (15)   NOT NULL,
    EMP_ST        VARCHAR (2)    NOT NULL,
```

```
    EMP_ZIP       INTEGER(5)     NOT NULL,
    EMP_PHONE     INTEGER(10)    NULL,
    EMP_PAGER     INTEGER(10)    NULL);
```

Polecenie przedstawione poniżej będzie kompatybilne z bazami Microsoft SQL Server i Oracle:

```
CREATE TABLE EMPLOYEE_TBL
   (EMP_ID        VARCHAR (9)    NOT NULL,
    EMP_NAME      VARCHAR (40)   NOT NULL,
    EMP_ST_ADDR   VARCHAR (20)   NOT NULL,
    EMP_CITY      VARCHAR (15)   NOT NULL,
    EMP_ST        VARCHAR (2)    NOT NULL,
    EMP_ZIP       INTEGER        NOT NULL,
    EMP_PHONE     INTEGER        NULL,
    EMP_PAGER     INTEGER        NULL);
```

Nasza tabela składa się z ośmiu kolumn. Zwróć uwagę, że dla zwiększenia czytelności nazw kolumn użyliśmy znaku podkreślenia „udającego" spację (na przykład kolumna reprezentująca identyfikator pracownika, `EMPLOYEE ID`, nosi nazwę `EMP_ID`). Każda kolumna ma przypisany typ danych, maksymalny rozmiar oraz ograniczenie `NULL` lub `NOT NULL`, określające, w których kolumnach mogą się znajdować puste pola, a które muszą być wypełnione w każdym rekordzie danych. Na przykład kolumna `EMP_PHONE` została zdefiniowana jako `NULL`, co oznacza, że mogą się w niej znajdować puste pola, ponieważ nie każdy pracownik musi posiadać swój numer telefonu. Definicje poszczególnych kolumn są od siebie oddzielone przecinkami, a cała definicja wszystkich kolumn została dodatkowo ujęta w nawiasy okrągłe (nawias otwierający znajduje się przed definicją pierwszej kolumny, a nawias zamykający następuje po definicji ostatniej kolumny).

Ostrzeżenie

> **Ograniczenia typów danych w poszczególnych systemach mogą być różne**
>
> Sprawdź w dokumentacji konkretnej bazy danych, jakie są maksymalne rozmiary poszczególnych typów danych i jakie zakresy danych mogą przyjmować. Ograniczenia typów danych w poszczególnych systemach mogą znacząco różnić się od siebie.

Każdy rekord (wiersz danych) naszej tabeli będzie się składał z następujących kolumn:

```
EMP_ID, EMP_NAME, EMP_ST_ADDR, EMP_CITY, EMP_ST, EMP_ZIP, EMP_PHONE, EMP_PAGER
```

W tej tabeli każde pole jest kolumną. Kolumna `EMP_ID` może składać się z jednego identyfikatora pracownika, ale równie dobrze może zawierać bardzo wiele takich identyfikatorów.

Konwencje nazewnictwa

Wybierając nazwy dla obiektów bazy danych, a zwłaszcza dla tabel i kolumn, należy je tak dobierać, aby odzwierciedlały rodzaje przechowywanych w nich danych. Na przykład tabela, w której przechowywane są informacje o pracownikach, może nosić nazwę

EMPLOYEE_TBL. Nazwy poszczególnych kolumn powinny być tworzone według takich samych zasad. Przykładowo jeżeli w danej kolumnie będzie przechowywany numer telefonu pracownika, to dosyć oczywistą nazwą dla tej kolumny będzie PHONE_NUMBER.

Polecenie ALTER TABLE

Po utworzeniu tabeli możesz ją modyfikować za pomocą polecenia ALTER TABLE. Polecenie to umożliwia dodawanie nowych kolumn do tabeli, usuwanie istniejących kolumn, zmianę definicji kolumn, dodawanie i usuwanie ograniczeń oraz, w niektórych implementacjach, modyfikowanie parametrów STORAGE tabeli. Oto standardowa składnia polecenia ALTER TABLE:

```
ALTER TABLE nazwa_tabeli [MODIFY] [COLUMN nazwa_kolumny ][ typ_danych | NULL NOT NULL]
[RESTRICT|CASCADE]
[DROP] [CONSTRAINT nazwa_ograniczenia ]
[ADD] [COLUMN] definicja_kolumny
```

Modyfikowanie elementów tabeli

Atrybuty kolumny odnoszą się do reguł i zachowania danych w kolumnie. Atrybuty kolumny możesz modyfikować za pomocą polecenia ALTER TABLE. W tym przypadku słowo *atrybuty* odnosi się do:

- typu danych kolumny;
- rozmiaru, precyzji bądź skali kolumny;
- określenia, czy w kolumnie mogą się znajdować wartości NULL.

W przykładzie przedstawionym poniżej używamy polecenia ALTER TABLE do zmodyfikowania atrybutów kolumny EMP_ID tabeli EMPLOYEE_TBL:

```
ALTER TABLE EMPLOYEE_TBL MODIFY EMP_ID VARCHAR(10);
Table altered.
```

Kolumna EMP_ID była już wcześniej co prawda zdefiniowana jako typ VARCHAR (typ tekstowy o zmiennej długości), ale teraz zwiększyliśmy jej maksymalny rozmiar z 9 do 10 znaków.

Dodawanie obowiązkowych kolumn do tabeli

Jedną z podstawowych reguł dodawania kolumn do istniejącej tabeli jest to, że jeżeli w tabeli zostały już wcześniej zapisane jakieś dane, to dodawana kolumna nie może być zdefiniowana jako NOT NULL. Klauzula NOT NULL oznacza, że w danej kolumnie we wszystkich rekordach muszą się znajdować jakieś wartości, stąd jeżeli dodajesz do istniejącej tabeli nową kolumnę z klauzulą NOT NULL, to od samego początku takie ograniczenie nie może być spełnione, o ile w istniejących wcześniej wierszach danych nie ma odpowiednich wartości dla nowej kolumny.

Istnieje jednak sposób na dodanie takiej obowiązkowej kolumny do tabeli:

1. Dodaj do tabeli nową kolumnę z klauzulą NULL (czyli taka kolumna może mieć puste pola).

2. Wstaw odpowiednie wartości kolumny dla każdego rekordu w tabeli.
3. Zmień atrybut tej kolumny na NOT NULL.

Dodawanie kolumn z autoinkrementacją do tabeli

W pewnych sytuacjach bardzo przydatna może się okazać możliwość utworzenia kolumny, która dla każdego nowego wiersza automatycznie inkrementuje swoją wartość, dając tym samym dla każdego wiersza unikatowy numer sekwencji. Takie rozwiązanie może być przydatne w wielu sytuacjach, na przykład kiedy nie posiadasz innego, naturalnego klucza lub chcesz wykorzystać numery sekwencji do sortowania danych. Tworzenie automatycznie inkrementowanych kolumn tabeli jest dosyć proste. W bazach MySQL możemy tego dokonać za pomocą typu SERIAL, który tworzy unikatowe wartości dla tabeli. Oto przykład takiego polecenia:

```
CREATE TABLE TEST_INCREMENT(
   ID SERIAL,
   TEST_NAME VARCHAR(20));
```

Uwaga

> **Zastosowanie atrybutu NULL do tworzenia tabel**
>
> Klauzula NULL jest domyślnym atrybutem kolumny, stąd w przeciwieństwie do klauzuli NOT NULL nie musisz jej używać w poleceniu CREATE TABLE.

W przypadku baz Microsoft SQL Server mamy do dyspozycji kolumny typu IDENTITY. Oto przykład polecenia tworzącego tabelę z wykorzystaniem takiej kolumny:

```
CREATE TABLE TEST_INCREMENT(
   ID INT IDENTITY(1,1) NOT NULL,
   TEST_NAME VARCHAR(20));
```

W bazach Oracle nie mamy żadnej bezpośredniej metody tworzenia kolumny z autoinkrementacją. Zamiast tego możemy jednak zasymulować podobne działanie za pomocą obiektów SEQUENCE i TRIGGER. Technikę tę opiszemy bardziej szczegółowo podczas omawiania wyzwalaczy w godzinie 22. „SQL dla zaawansowanych".

Po utworzeniu takiej tabeli możemy do niej wstawiać dane bez konieczności podawania wartości dla kolumny z automatyczną inkrementacją:

```
INSERT INTO TEST_INCREMENT(TEST_NAME) VALUES ('FRED'),('JOE'),('MIKE'),('TED');
SELECT * FROM TEST_INCREMENT;
| ID |    TEST_NAME  |
|  1 |    FRED       |
|  2 |    JOE        |
|  3 |    MIKE       |
|  4 |    TED        |
```

Modyfikowanie kolumn

Chcąc dokonać modyfikacji istniejących kolumn w tabeli, należy najpierw rozważyć kilka bardzo istotnych spraw. Oto najważniejsze reguły modyfikowania kolumn:

- Rozmiar kolumny może być zwiększany aż do maksymalnego rozmiaru przypisanego do niej typu danych.
- Rozmiar kolumny może być zmniejszany tylko wtedy, kiedy rozmiar największego elementu zapisanego w tej kolumnie jest równy nowemu rozmiarowi kolumny lub od niego mniejszy.
- Liczba cyfr znaczących dla kolumn numerycznych może być zawsze zwiększana.
- Liczba cyfr znaczących dla kolumn numerycznych może być zmniejszana tylko wtedy, kiedy liczba cyfr wartości o największej liczbie cyfr znaczących jest równa nowej liczbie cyfr znaczących modyfikowanej kolumny lub od niej mniejsza.
- Liczba miejsc dziesiętnych w kolumnie numerycznej może być dowolnie zmniejszana lub zwiększana.
- Typ danych przypisany do kolumny może być zmieniany.

W niektórych implementacjach baz danych użycie wybranych opcji polecenia `ALTER TABLE` może podlegać pewnym ograniczeniom. Na przykład możesz nie mieć możliwości usuwania kolumn tabeli. Aby to zrobić, trzeba w takiej sytuacji usunąć całą tabelę, a następnie ponownie ją utworzyć bez niepotrzebnej już kolumny. Usunięcie kolumny z tabeli, która jest zależna od kolumny w innej tabeli, lub usunięcie kolumny, do której odwołuje się kolumna z innej tabeli, może spowodować mniej lub bardziej poważne problemy. Pamiętaj, aby przed wykonaniem takiej operacji uważnie zapoznać się z dokumentacją implementacji bazy danych.

Uwaga

> **Tworzenie tabel, których będziemy używać w przykładach i ćwiczeniach**
>
> Tabele, których będziemy używać w ćwiczeniach i przykładach omawianych w tej książce, możesz utworzyć, wykonując szczegółowe polecenia znajdujące się w sekcji „Ćwiczenia" na końcu tej godziny. W godzinie 5. „Operowanie danymi", dowiesz się, jak wypełnić nowo utworzone tabele odpowiednimi danymi.

Tworzenie nowej tabeli na podstawie istniejącej tabeli

Używając kombinacji poleceń `CREATE TABLE` i `SELECT`, możesz utworzyć kopię istniejącej tabeli. Nowa tabela będzie miała takie same definicje kolumn jak tabela, która była jej pierwowzorem. W zależności od potrzeb możesz wybrać wszystkie lub tylko niektóre kolumny tabeli wzorcowej. Rozmiar nowych kolumn, które utworzysz za pomocą funkcji lub jako kombinacje innych kolumn, zostanie dobrany automatycznie tak, aby pomieścić wszystkie dane. Podstawowa składnia polecenia tworzącego nową tabelę na podstawie innej tabeli jest następująca:

```
CREATE TABLE nazwa_nowej_tabeli AS
   SELECT [ *|kolumna1, kolumna2]
   FROM nazwa_tabeli
      [WHERE warunki]
```

Ostrzeżenie

> **Modyfikowanie lub usuwanie tabel może być niebezpieczne**
>
> Usuwając lub modyfikując istniejące tabele, należy zachować szczególną ostrożność. Jeżeli popełnisz jakiś błąd logiczny w poleceniu lub nawet prostą literówkę w nazwie tabeli, możesz spowodować niezamierzoną utratę ważnych danych.

Zwróć uwagę, że w składni naszego polecenia pojawiły się nowe słowa kluczowe, a w szczególności polecenie SELECT. Polecenie SELECT służy do tworzenia zapytań i omówimy je szczegółowo w godzinie 7. „Wprowadzenie do tworzenia zapytań". W tym momencie powinna nam wystarczyć wiedza, że możemy tworzyć nowe tabele na podstawie wyników zwracanych przez zapytanie.

Bazy MySQL i Oracle umożliwiają tworzenie nowych tabel na podstawie istniejących tabel za pomocą polecenia CREATE TABLE AS SELECT. Microsoft SQL Server używa do tego celu innego polecenia. W przypadku tej implementacji bazy danych należy użyć polecenia SELECT ... INTO. Składnia tego polecenia wygląda następująco:

```
SELECT [ *|kolumna1, kolumna2]
INTO nazwa_nowej_tabeli
FROM nazwa_tabeli
[WHERE warunki]
```

Poniżej przedstawimy kilka przykładów zastosowania takich poleceń.

Najpierw wykonamy proste zapytanie, które wyświetli dane przechowywane w tabeli FlightStatuses:

```
select * from FlightStatuses;
STATUSCODE   STATUSNAME
--------------------------
CAN          Cancelled
COM          Completed
DEL          Delayed
ONT          On-Time
```

Następnie na bazie powyższego zapytania utworzymy nową tabelę o nazwie Flight
↪StatusesNew:

```
create table FlightStatusesNew as select * from FlightStatuses;
Table created.
```

W przypadku bazy Microsoft SQL Server podobne polecenie będzie wyglądało następująco:

```
select * into FlightStatusesNew from FlightStatuses;
Table created.
```

Teraz, jeżeli wykonamy zapytanie na tabeli `FlightStatusesNew`, otrzymamy takie same wyniki jak w przypadku zapytania na oryginalnej tabeli:

```
select * from FlightStatusesNew;
STATUSCODE   STATUSNAME
--------------------------
CAN          Cancelled
COM          Completed
DEL          Delayed
ONT          On-Time
```

Wskazówka

> **Co oznacza * w zapytaniach?**
>
> Polecenie `SELECT *` zwraca dane ze wszystkich pól odpytywanej tabeli. Symbol gwiazdki (*) reprezentuje cały wiersz danych lub, inaczej mówiąc, cały rekord tabeli.

Usuwanie tabel

Usuwanie tabeli jest w praktyce jednym z łatwiejszych zadań. Jeżeli polecenie zostanie wykonane z opcją `RESTRICT`, a tabela jest używana przez widoki lub ma więzy integralności, próba jego wykonania zakończy się niepowodzeniem i wystąpieniem błędu. Jeśli w poleceniu zostanie użyta opcja `CASCADE`, tabela zostanie pomyślnie usunięta, a wraz z nią wszystkie powiązane z nią widoki i więzy integralności. Składnia polecenia usuwającego tabelę wygląda następująco:

```
DROP TABLE nazwa_tabeli [RESTRICT | CASCADE]
```

Microsoft SQL Server nie pozwala na stosowanie opcji `CASCADE`. W przypadku tej implementacji musisz samodzielnie się upewnić, że usunąłeś wszystkie obiekty odwołujące się do usuwanej tabeli tak, aby nie pozostawić w bazie nieprawidłowego obiektu.

Poniższe polecenie powoduje usunięcie tabeli o nazwie `products_tmp`:

```
drop table products_tmp;
Table dropped.
```

Ostrzeżenie

> **Zachowaj ostrożność podczas usuwania tabel**
>
> Usuwając tabelę, należy zawsze umieszczać w poleceniu nazwę schematu lub właściciela tabeli, co pozwoli zminimalizować ryzyko przypadkowego usunięcia niewłaściwej tabeli. Jeżeli masz dostęp do wielu kont użytkowników, upewnij się przed usunięciem tabeli, że zalogowałeś się do bazy na właściwym koncie użytkownika.

Więzy integralności

Więzy integralności to zbiór reguł, które gwarantują poprawność i logiczną spójność danych wprowadzanych i przechowywanych w relacyjnej bazie danych. Spójność danych w bazie jest realizowana za pomocą tak zwanych **więzów integralności referencyjnej** (ang. *referential integrity*), które wykorzystują wiele rodzajów reguł integralnościowych. Więzy integralności referencyjnej składają się z szeregu zasad i warunków zdefiniowanych w bazie danych, które muszą być spełnione przez określony podzbiór danych, wymuszając tym samym poprawność i spójność danych przechowywanych w tabelach.

Więzy klucza głównego (PRIMARY KEY)

Klucz główny to określenie kolumny, która zapewnia unikatowość poszczególnych wierszy danych i umożliwia ich jednoznaczną identyfikację. Choć zazwyczaj rolę klucza głównego spełnia jedna, wybrana kolumna tabeli, to jednak klucz główny może się również składać z kilku kolumn. Na przykład rolę klucza głównego w tabeli przechowującej dane pracowników może spełniać zarówno kolumna zawierająca numery PESEL, jak i kolumna zawierająca numery pracowników. Każdy rekord tabeli powinien mieć unikatową wartość klucza głównego i taką rolę może spełniać unikatowy numer pracownika. Ponieważ w tabeli pracowników z reguły nie ma potrzeby tworzenia więcej niż jednego rekordu dla każdego pracownika, unikatowy numer pracownika wydaje się być logicznym kandydatem na klucz główny tabeli. Wyboru klucza głównego dokonujemy podczas tworzenia tabeli.

Oto przykład polecenia, które tworzy tabelę `EMPLOYEE_TBL` i na jej klucz główny wybiera kolumnę o nazwie `EMP_ID`:

```
CREATE TABLE EMPLOYEE_TBL
   (EMP_ID       VARCHAR(9)     NOT NULL PRIMARY KEY,
    EMP_NAME     VARCHAR(40)    NOT NULL,
    EMP_ST_ADDR  VARCHAR(20)    NOT NULL,
    EMP_CITY     VARCHAR(15)    NOT NULL,
    EMP_ST       VARCHAR(2)     NOT NULL,
    EMP_ZIP      INTEGER(5)     NOT NULL,
    EMP_PHONE    INTEGER(10)    NULL,
    EMP_PAGER    INTEGER(10)    NULL);
```

W tym przypadku więzy klucza głównego tworzone są w sposób pośredni, poprzez użycie klauzuli `PRIMARY KEY` w definicji kolumny. Więzy klucza głównego możesz również zdefiniować bezpośrednio, dodając do polecenia tworzącego tabelę odpowiednią klauzulę:

```
CREATE TABLE EMPLOYEE_TBL
   (EMP_ID       VARCHAR(9)     NOT NULL PRIMARY KEY,
    EMP_NAME     VARCHAR(40)    NOT NULL,
    EMP_ST_ADDR  VARCHAR(20)    NOT NULL,
    EMP_CITY     VARCHAR(15)    NOT NULL,
    EMP_ST       VARCHAR(2)     NOT NULL,
    EMP_ZIP      INTEGER(5)     NOT NULL,
    EMP_PHONE    INTEGER(10)    NULL,
    EMP_PAGER    INTEGER(10)    NULL,
    PRIMARY KEY (EMP_ID));
```

W tym przykładzie definicja więzów klucza głównego została dodana na końcu polecenia CREATE TABLE, zaraz za definicją ostatniej kolumny tabeli.

Za pomocą obu metod możesz również utworzyć klucz główny, który będzie się składał z więcej niż jednej kolumny. Oto przykład poleceń definiujących więzy klucza głównego na dwóch kolumnach tabeli w bazie danych Oracle:

```
CREATE TABLE PRODUCT_TST
   (PROD_ID       VARCHAR (10)       NOT NULL,
    VEND_ID       VARCHAR (10)       NOT NULL,
    PRODUCT       VARCHAR (30)       NOT NULL,
    COST          NUMBER(8,2)        NOT NULL,
    PRIMARY KEY (PROD_ID, VEND_ID));

ALTER TABLE PRODUCTS_TST
   ADD CONSTRAINT PRODUCTS_PK PRIMARY KEY (PROD_ID, VEND_ID);
```

Więzy klucza jednoznacznego (UNIQUE)

Klucz jednoznaczny (ang. *unique column constraint*) ma właściwości bardzo podobne do klucza głównego, gdzie każda z wartości w kolumnie musi być unikatowa. Różnica polega na tym, że tabela może mieć tylko jeden klucz główny, a kluczy unikatowych może być więcej. Przykładowo możemy założyć więzy klucza jednoznacznego na kilku kolumnach, które nie są używane jako klucz główny tabeli.

Przyjrzyjmy się następującemu przykładowi:

```
CREATE TABLE EMPLOYEE_TBL
   (EMP_ID          VARCHAR (9)     NOT NULL     PRIMARY KEY,
    EMP_NAME        VARCHAR (40)    NOT NULL,
    EMP_ST_ADDR     VARCHAR (20)    NOT NULL,
    EMP_CITY        VARCHAR (15)    NOT NULL,
    EMP_ST          VARCHAR (2)     NOT NULL,
    EMP_ZIP         INTEGER(5)      NOT NULL,
    EMP_PHONE       INTEGER(10)     NULL         UNIQUE,
    EMP_PAGER       INTEGER(10)     NULL);
```

Kluczem głównym naszej tabeli jest kolumna EMP_ID, co oznacza, że polem gwarantującym unikatowość każdego rekordu jest numer pracownika. Klucz główny to kolumna, do której zazwyczaj odwołujemy się w zapytaniach, zwłaszcza podczas łączenia danych z różnych tabel. Kolumna EMP_PHONE została zdefiniowana jako klucz jednoznaczny UNIQUE, co oznacza, że dwóch pracowników nie może mieć takiego samego numeru telefonu. Oba klucze nie różnią się zbytnio od siebie, z tym że to klucz główny określa kolejność danych w tabeli i pozwala łączyć powiązane ze sobą tabele.

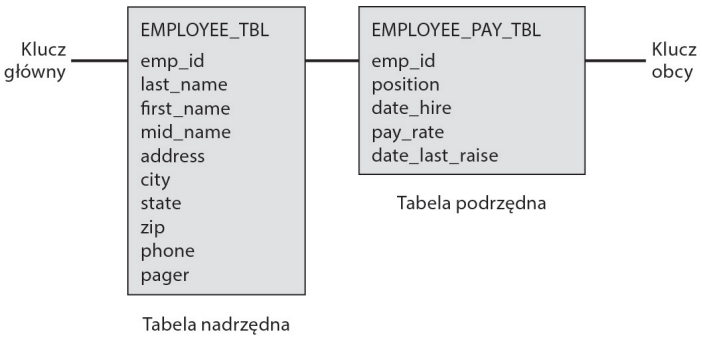
Więzy klucza obcego (FOREIGN KEY)

Klucz obcy (ang. *foreign key*) to kolumna w tabeli podrzędnej odwołująca się do klucza głównego tabeli nadrzędnej. Więzy klucza obcego (ang. *foreign key constraint*) to główny mechanizm wymuszający integralność referencyjną między tabelami w relacyjnej bazie danych. Kolumna zdefiniowana jako klucz obcy odwołuje się do kolumny zdefiniowanej jako klucz główny innej tabeli.

Tworzenie klucza obcego zostało pokazane w poniższym przykładzie:

```
CREATE TABLE EMPLOYEE_PAY_TBL
   (EMP_ID             VARCHAR (9)       NOT NULL,
    POSITION           VARCHAR (15)      NOT NULL,
    DATE_HIRE          DATE              NULL,
    PAY_RATE           NUMBER(4,2)       NOT NULL,
    DATE_LAST_RAISE    DATE              NULL,
    CONSTRAINT EMP_ID_FK FOREIGN KEY (EMP_ID) REFERENCES EMPLOYEE_TBL (EMP_ID));
```

Kolumna EMP_ID tabeli EMPLOYEE_PAY_TBL została zdefiniowana jako klucz obcy odwołujący się do kolumny EMP_ID tabeli EMPLOYEE_TBL. Taki klucz obcy gwarantuje, że dla każdego rekordu z identyfikatorem EMP_ID w tabeli EMPLOYEE_PAY_TBL istnieje odpowiadający mu rekord z identyfikatorem EMP_ID w tabeli EMPLOYEE_ID. Takie rozwiązanie jest nazywane **relacją tabela nadrzędna – tabela podrzędna** (ang. *parent/child relationship*). W tym przypadku tabelą nadrzędną jest tabela EMPLOYEE_TBL, a tabelą podrzędną EMPLOYEE_PAY_TBL. Relacja ta została zilustrowana na rysunku 3.4.



RYSUNEK 3.4. Relacja tabela nadrzędna — tabela podrzędna

Na rysunku kolumna EMP_ID w tabeli podrzędnej odwołuje się do kolumny EMP_ID w tabeli nadrzędnej. Aby można było wstawić nową wartość EMP_ID do tabeli podrzędnej, musi istnieć odpowiadająca jej wartość EMP_ID w tabeli nadrzędnej. Analogicznie, aby usunąć wybraną wartość EMP_ID z tabeli nadrzędnej, musimy usunąć wszystkie odpowiadające jej wartości EMP_ID z tabeli podrzędnej. W taki właśnie sposób działają więzy integralności referencyjnej.

Aby dodać klucz obcy do tabeli, należy użyć polecenia ALTER TABLE, tak jak to zostało pokazane w poniższym przykładzie:

```
alter table employee_pay_tbl
add constraint id_fk foreign key (emp_id)
references employee_tbl (emp_id);
```

Uwaga

> **Odmiany polecenia ALTER TABLE**
>
> Opcje polecenia ALTER TABLE w różnych implementacjach SQL mogą być inne, zwłaszcza w przypadku więzów integralności. Ponadto sposób użycia i definiowania więzów w poszczególnych implementacjach może być różny, ale samo pojęcie integralności referencyjnej powinno być takie samo w przypadku wszystkich relacyjnych baz danych.

Więzy NOT NULL

W poprzednich przykładach wykorzystywaliśmy słowa kluczowe NULL i NOT NULL w wierszach definiujących poszczególne kolumny tabeli. W kolumnie, dla której zdefiniowane są więzy NOT NULL, nie dopuszcza się wystąpienia wartości NULL, a zatem poszczególne pola w takiej kolumnie nie mogą być puste. Pamiętaj, że domyślnie podczas tworzenia tabeli wszystkie kolumny są deklarowane jako NULL, o ile nie zostaną jawnie zadeklarowane jako NOT NULL.

Więzy CHECK

Więzy CHECK pozwalają zdefiniować warunek sprawdzający poprawność danych wprowadzanych do kolumny tabeli. Zastosowanie takich więzów umożliwia kontrolę poprawności danych na poziomie bazy, choć zwykle odbywa się to na poziomie aplikacji użytkownika, gdzie zazwyczaj istnieje szereg ściśle określonych reguł ograniczających zakresy danych, które mogą być wprowadzane do poszczególnych kolumn. Zastosowanie więzów CHECK pozwala dostarczyć kolejną warstwę ochronną, umożliwiającą wymuszenie poprawności wprowadzanych danych.

Oto przykład zastosowania więzów CHECK w tabeli bazy danych Oracle:

```
CREATE TABLE EMPLOYEE_CHECK_TST
   (EMP_ID          VARCHAR (9)       NOT NULL,
    EMP_NAME        VARCHAR (40)      NOT NULL,
    EMP_ST_ADDR     VARCHAR (20)      NOT NULL,
    EMP_CITY        VARCHAR (15)      NOT NULL,
    EMP_ST          VARCHAR (2)       NOT NULL,
    EMP_ZIP         NUMBER(5)         NOT NULL,
    EMP_PHONE       NUMBER(10)        NULL,
    EMP_PAGER       NUMBER(10)        NULL,
    PRIMARY KEY (EMP_ID),
    CONSTRAINT CHK_EMP_ZIP CHECK ( EMP_ZIP = '46234'));
```

W naszej przykładowej tabeli na kolumnie EMP_ZIP zostało nałożone ograniczenie CHECK gwarantujące, że dla wszystkich pracowników, których dane znajdą się w tej tabeli, kod pocztowy będzie miał wartość 46234. Jak widać, jest to dosyć mocne ograniczenie, ale za to dobrze ilustruje sposób działania więzów CHECK.

Jeżeli zechcesz użyć więzów CHECK do sprawdzenia, czy podawany kod pocztowy znajduje się na liście dozwolonych kodów, to definicja takich więzów powinna wyglądać na przykład tak:

```
CONSTRAINT CHK_EMP_ZIP CHECK ( EMP_ZIP in ('46234','46227','46745') );
```

Innym przykładem może być sytuacja, w której stawka godzinowa pracownika nie może być mniejsza niż określona wartość minimalna. W tym wypadku możemy użyć następującej definicji więzów CHECK:

```
CREATE TABLE EMPLOYEE_PAY_TBL
   (EMP_ID             VARCHAR (9)    NOT NULL,
    POSITION           VARCHAR (15)   NOT NULL,
    DATE_HIRE          DATE           NULL,
    PAY_RATE           NUMBER(4,2)    NOT NULL,
    DATE_LAST_RAISE    DATE           NULL,
    CONSTRAINT EMP_ID_FK FOREIGN KEY (EMP_ID) REFERENCES EMPLOYEE_TBL (EMP_ID),
    CONSTRAINT CHK_PAY CHECK ( PAY_RATE > 12.50 ) );
```

W tym przykładzie wszyscy pracownicy, których dane są wprowadzane do tabeli, muszą mieć wpisaną stawkę wynagrodzenia większą niż 12.50 za godzinę. W więzach CHECK możesz zamieścić praktycznie dowolny warunek, podobnie jak to ma miejsce w zapytaniach SQL. Więcej szczegółowych informacji na temat warunków znajdziesz w godzinach 5. i 7.

Usuwanie więzów

Za pomocą polecenia ALTER TABLE z opcją DROP CONSTRAINT możesz usunąć dowolne więzy integralności zdefiniowane wcześniej w tabeli. Na przykład aby usunąć więzy klucza głównego z tabeli EMPLOYEES, użyj poniższego polecenia:

```
ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;
Table altered.
```

Niektóre implementacje baz danych umożliwiają stosowanie uproszczonej składni poleceń usuwających niektóre więzy integralności. Na przykład aby w bazie MySQL usunąć więzy klucza głównego tabeli, możesz użyć następującego polecenia:

```
ALTER TABLE EMPLOYEES DROP PRIMARY KEY;
Table altered.
```

Wskazówka

Wskazówka

Inne metody pracy z więzami integralności

W niektórych implementacjach baz danych zamiast całkowicie usuwać wybrane więzy integralności z tabeli możesz je tymczasowo zablokować i później w razie potrzeby odblokować.

Podsumowanie

W tej godzinie dowiedziałeś się nieco o obiektach bazy danych, ale głównie skoncentrowaliśmy się tutaj na zagadnieniach związanych tabelami. Tabela jest najprostszą formą przechowywania danych w relacyjnej bazie danych. W tabelach przechowywane są różnego rodzaju informacje, takie jak dane pracowników, dane klientów czy informacje o produktach. Tabele składają się z szeregu kolumn, a każda kolumna ma przypisane odpowiednie atrybuty, na przykład typ danych czy więzy i ograniczenia nałożone na kolumnę, takie jak klauzula `NOT NULL`, klucz główny, klucze obce czy więzy klucza jednoznacznego.

Poznałeś również polecenie `CREATE TABLE`, za pomocą którego możemy tworzyć tabele, oraz szereg jego opcji dostępnych w różnych implementacjach baz danych. Dowiedziałeś się też, jak za pomocą polecenia `ALTER TABLE` możemy modyfikować struktury istniejących tabel.

Choć zarządzanie tabelami nie jest takie proste, jak mogłoby się na pierwszy rzut oka wydawać, to jednak kiedy poznasz strukturę i sposób funkcjonowania tabel w środowisku relacyjnej bazy danych, inne tematy, takie jak tworzenie zapytań czy modyfikowanie danych, z pewnością staną się łatwiejsze do opanowania. W kolejnych godzinach będziemy omawiać zagadnienia związane z zarządzaniem innymi obiektami baz danych, takimi jak indeksy tabel czy widoki.

Pytania i odpowiedzi

P: Czy nadając nazwę nowej tabeli, muszę używać przyrostków takich jak `_TBL`?

O: Absolutnie nie. Nie musisz używać żadnych przyrostków, przedrostków ani innych tego typu elementów. Na przykład tabela, w której przechowywane są dane pracowników, może nosić dowolną nazwę, aczkolwiek dobrą praktyką jest nadawanie nazw sugerujących przeznaczenie lub rodzaj przechowywanych danych. Oto kilka propozycji nazw dla takiej tabeli:

```
PRACOWNIK
PRACOWNIK_TBL
EMPLOYEE_TBL
EMPLOYEE_TABLE
WORKER
```

P: Dlaczego podczas usuwania tabeli podawanie nazwy schematu jest takie ważne?

O: W ramach odpowiedzi na to pytanie przytoczymy prawdziwą historię pewnego administratora bazy danych, który usunął tabelę. Jeden z programistów utworzył w swoim schemacie tabelę roboczą o takiej samej nazwie jak tabela produkcyjna. Kilka tygodni później ten programista odszedł z firmy. Administrator bazy danych rozpoczął usuwanie konta tego pracownika, ale próba wykonania polecenia `DROP USER` zakończyła się niepowodzeniem, ponieważ w schemacie programisty znajdowała się wspomniana tabela. Po przeprowadzeniu małego dochodzenia okazało się, że tabela nie jest już do niczego potrzebna i można ją usunąć. Administrator wykonał zatem polecenie `DROP TABLE`, które zadziałało bez żadnych problemów i tabela została usunięta.

Problem polegał jednak na tym, że podczas usuwania tabeli nasz administrator był zalogowany do schematu produkcyjnego. Wykonując polecenie DROP TABLE, powinien był podać nazwę schematu lub właściciela tabeli, którą chciał usunąć. Ponieważ tego nie zrobił, a nazwa tabeli roboczej w schemacie programisty była taka sama jak nazwa tabeli produkcyjnej, nieopatrznie wykonane polecenie usunęło tabelę ze schematu produkcyjnego, w którym był zalogowany administrator. Usunięcie szkód i odtworzenie bazy danych z kopii zapasowej zajęło prawie 8 godzin.

Warsztaty

Warsztaty w tej godzinie składają się z serii pytań testowych i ćwiczeń praktycznych. Quiz został przygotowany tak, aby sprawdzić ogólną znajomość zagadnień omawianych w tej godzinie. Ćwiczenia zaś dadzą Ci możliwość praktycznego zastosowania wiedzy zdobytej podczas pracy z książką. Zanim przejdziesz do kolejnej godziny, poświęć trochę czasu na wykonanie ćwiczeń i odpowiedzenie na pytania quizowe. Odpowiedzi do wszystkich pytań i ćwiczeń znajdziesz w dodatku C „Odpowiedzi na pytania i rozwiązania ćwiczeń".

Quiz

1. Czy polecenie CREATE TABLE przedstawione poniżej będzie działać poprawnie? Jeżeli nie, to co musimy zrobić, aby usunąć problem(y)? Czy istnieją jakieś ograniczenia co do implementacji bazy danych, w której takie polecenie może zadziałać (MySQL, Oracle czy SQL Server)?

```
Create table EMPLOYEE_TABLE as:
   (ssn           number(9)       not null,
    last_name     varchar(20)     not null,
    first_name    varchar(20)     not null,
    middle_name   varchar(20)     not null,
    st address    varchar(30)     not null,
    city          varchar(20)     not null,
    state         varchar(2)      not null,
    zip           number(4)       not null,
    date hired    date);
```

2. Czy możesz usunąć wybraną kolumnę z tabeli?
3. Jakiego polecenia użyjesz do utworzenia klucza głównego w tabeli EMPLOYEE_TABLE?
4. Jakiego polecenia użyjesz do zmiany tabeli EMPLOYEE_TABLE tak, aby w kolumnie middle_name można było używać wartości NULL?
5. Jakiego polecenia użyjesz do zmiany tabeli EMPLOYEE_TABLE tak, aby można było do niej wprowadzać tylko dane pracowników mieszkających w stanie NY (kolumna state)?
6. Jakiego polecenia użyjesz, aby dodać do tabeli EMPLOYEE_TABLE automatycznie inkrementowaną kolumnę EMPID? Podaj składnię dla baz MySQL i SQL Server.

Ćwiczenie

W tej sekcji utworzymy w bazie danych wszystkie tabele, z których będziemy korzystać w ćwiczeniach i przykładach omawianych w dalszej części książki, a następnie wykonamy kilka poleceń, za pomocą których zapoznasz się ze strukturą tabel przechowywanych w bazie. Ponieważ każda z używanych przez nas implementacji (Oracle i Microsoft SQL Server) wymaga nieco innego podejścia, poniżej zamieszczamy szczegółowe instrukcje wykonania tego ćwiczenia dla każdej z tych implementacji osobno.

Microsoft SQL Server

Uruchom okno wiersza poleceń i zaloguj się do lokalnej instancji bazy danych SQL Server, wykonując polecenie pokazane poniżej. Pamiętaj, aby podać odpowiednią nazwę konta użytkownika i hasło dostępu. Upewnij się, że pomiędzy opcją `-p` i hasłem dostępu nie pozostawiłeś spacji.

```
SQLCMD -S localhost -U nazwa_użytkownika -Phasło_dostępu
```

Po zalogowaniu na ekranie pojawi się znak zachęty 1>. Wpisz kolejne polecenie, które poinformuje serwer, że chcesz użyć bazy danych o nazwie `learnsql`. Pamiętaj, że pracując z konsolą SQLCMD, do uruchomienia wpisanego polecenia musisz użyć komendy GO, tak jak to zostało pokazane poniżej:

```
1>use learnsql;
2>GO
```

Teraz przejdź do dodatku D „Ćwiczenia dodatkowe", gdzie znajdziesz zestawienie poleceń DDL dla tabel używanych w tej książce. W wierszu poleceń konsoli, gdzie pojawił się znak zachęty 1>, wpisz kolejne polecenia `CREATE TABLE`. Upewnij się, że zakończyłeś każde polecenie znakiem średnika, i uruchom każde polecenie za pomocą komendy GO. Uważne wykonanie tych poleceń spowoduje utworzenie tabel, z których będziemy korzystać w kolejnych godzinach.

Po utworzeniu tabel przejdź do wiersza poleceń konsoli i po pojawieniu się znaku zachęty 1> wpisz przedstawione poniżej polecenie, które spowoduje wyświetlenie listy istniejących tabel. Pamiętaj o uruchomieniu tego polecenia komendą GO.

```
Select name from sys.tables;
```

Kiedy znak zachęty 1> ponownie pojawi się na ekranie, użyj procedury składowanej (ang. *stored procedure*) o nazwie `sp_help` do wyświetlenia listy kolumn i ich atrybutów dla jednej z nowo utworzonych tabel, na przykład:

```
Sp_help_ trips;
Sp_help flights;
```

Jeżeli podczas próby wykonania takiego polecenia pojawią się błędy, należy po prostu ponownie utworzyć odpowiednie tabele. Jeżeli tabela została poprawnie utworzona, ale w definicji kolumn pojawiły się błędy (na przykład popełniłeś literówkę w nazwie kolumny lub zapomniałeś dodać jakiejś kolumny), usuń całą tabelę i ponownie utwórz ją za pomocą odpowiedniego polecenia `CREATE TABLE`. Składnia polecenia `DROP TABLE`, za pomocą którego możesz usunąć wybraną tabelę, jest następująca:

```
drop table nazwa_tabeli;
```

Oracle

Przejdź do okna wiersza poleceń konsoli i zaloguj się do lokalnej instancji bazy danych Oracle, wykonując polecenie przedstawione poniżej. Program poprosi Cię o podanie nazwy konta użytkownika i hasła dostępu.

```
sqlplus
```

Teraz przejdź do dodatku D, gdzie znajdziesz zestawienie poleceń DDL dla tabel używanych w tej książce. W wierszu poleceń konsoli SQLPlus, gdzie pojawił się znak zachęty SQL>, wpisz kolejne polecenia `CREATE TABLE`. Upewnij się, że zakończyłeś każde polecenie znakiem średnika. Uważne wykonanie tych poleceń spowoduje utworzenie tabel, z których będziemy korzystać w kolejnych godzinach.

Po zakończeniu tworzenia tabeli w wierszu poleceń SQL> wpisz przedstawione poniżej polecenie, które spowoduje wyświetlenie listy istniejących tabel:

```
Select * from cat;
```

Jeżeli wszystkie tabele zostały utworzone poprawnie, wyniki działania tego polecenia powinny być następujące:

```
SQL> SELECT * FROM CAT;

TABLE_NAME                     TABLE_TYPE
------------------------------ -----------
TRIPS                          TABLE
TRIPITINERARY                  TABLE
ROUTES                         TABLE
RICH_EMPLOYEES                 TABLE
PASSENGERS                     TABLE
HIGH_SALARIES                  TABLE
FLIGHTSTATUSES                 TABLE
FLIGHTS                        TABLE
EMPLOYEE_MGR                   TABLE
EMPLOYEES                      TABLE
EMPLOYEEPOSITIONS              TABLE
COUNTRIES                      TABLE
AIRPORTS                       TABLE
AIRCRAFTFLEET                  TABLE
AIRCRAFT                       TABLE

15 rows selected.
```

Teraz w wierszu poleceń SQL> wyświetl za pomocą polecenia `DESCRIBE` listę kolumn i ich atrybutów dla jednej z nowo utworzonych tabel (zamiast tego możesz użyć skróconej wersji tego polecenia, `desc`). Na przykład wykonanie polecenia

```
DESCRIBE FLIGHTS;
```

daje wyniki przedstawione poniżej:

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
FLIGHTID                                  NOT NULL NUMBER(10)
FLIGHTSTART                                        DATE
FLIGHTEND                                          DATE
```

```
FLIGHTDURATION                  NUMBER(5)
ROUTEID                         NUMBER(10)
AIRCRAFTFLEETID                 NUMBER(10)
STATUSCODE                      CHAR(3 CHAR)
```

Jeżeli pojawią się błędy, należy po prostu ponownie utworzyć odpowiednie tabele. Jeżeli tabela została poprawnie utworzona, ale w definicji kolumn pojawiły się błędy (na przykład popełniłeś literówkę w nazwie kolumny lub zapomniałeś dodać jakiejś kolumny), usuń całą tabelę i ponownie utwórz ją za pomocą odpowiedniego polecenia `CREATE TABLE`. Składnia polecenia `DROP TABLE`, za pomocą którego możesz usunąć wybraną tabelę, jest następująca:

```
drop table nazwa_tabeli;
```

Skorowidz

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

T

U

V

W

Z

Aby w pełni wykorzystać możliwości wielu najlepszych baz danych, takich jak Oracle czy MS SQL Server, trzeba nauczyć się języka SQL. To narzędzie stworzone specjalnie na potrzeby baz danych: pozwala na budowanie ich struktury i wypełnianie tabel danymi, na wyszukiwanie i pobieranie informacji, a także na zarządzanie wszystkimi aspektami działania bazy. Mogłoby się wydawać, że musi to być język bardzo skomplikowany i trudny do opanowania!

Książka, którą trzymasz w dłoni, została pomyślana jako podręcznik składający się z dwudziestu czterech godzinnych lekcji. Dzięki niej błyskawicznie zaczniesz korzystać z zaawansowanych technik bazodanowych. Nauczysz się używać widoków, wykonywać transakcje, konfigurować połączenia sieciowe i stosować rozszerzenia języka SQL dostępne w bazach danych Oracle i MS SQL Server. Szczegółowe instrukcje i wskazówki, praktyczne przykłady, a także liczne quizy i ćwiczenia pozwolą Ci na solidne przyswojenie materiału i natychmiastowe wypróbowanie zdobytej wiedzy w praktyce.

Najważniejsze zagadnienia ujęte w książce:

- **projektowanie efektywnych struktur baz danych i normalizacja danych**
- **grupowanie, sortowanie i modyfikowanie danych**
- **transakcje i efektywna optymalizacja zapytań**
- **zarządzanie bazami danych i kontami użytkowników**
- **kwestie bezpieczeństwa baz i danych**
- **korzystanie z SQL w internecie i rozszerzenia języka SQL dla Oracle i SQL Server**

Przekonaj się, jak szybko można nauczyć się swobodnie używać SQL!

Ryan Stephens — prowadzi zajęcia z języka SQL i baz danych Oracle na Indiana University – Purdue University. Pracował jako analityk i programista w Gwardii Narodowej stanu Indiana. Jest autorem wielu książek o bazach danych.

Arie D. Jones — często występuje na różnych konferencjach technicznych. Jest autorem kilku książek i artykułów dotyczących baz danych.

Ronald Plew — prowadził zajęcia z baz danych i języka SQL na Indiana University – Purdue University, a później pracował jako analityk i programista w Gwardii Narodowej stanu Indiana. Jest współautorem wielu książek o bazach danych.

sięgnij po WIĘCEJ

KOD KORZYŚCI

Helion

SAMS

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

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

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

ISBN 978-83-283-2590-6

9 788328 325906

Informatyka w najlepszym wydaniu

cena: 69,00 zł