SQL. Szybki start

Autor: Chris Fehily
Tłumaczenie: Jarosław Gierlicki
ISBN: 83-7361-037-5
Tytuł oryginału: SQL: Visual QuickStart Guide
Format: B5, stron: 416

SQL jest uznawanym za standard językiem programowania służącym do tworzenia, modyfikowania oraz pobierania informacji przechowywanych przez systemy zarządzania relacyjnymi bazami danych. Chociaż każdy system bazodanowy używa nieco innego dialektu tego języka, często rozbudowując go o dodatkowe funkcje, podstawowe instrukcje SQL są wspólne dla wszystkich systemów: od Accessa do Oracle'a.

Bogata w ilustracje, napisana przystępnym językiem książka „SQL. Szybki start" pozwoli Ci nauczyć się SQL-a i zacząć pracę z relacyjną bazą danych, niezależnie do tego, który system wybierzesz. Autor kładzie szczególny nacisk na realizację konkretnych zadań, przedstawiając kolejne kroki, jakie należy wykonać, by rozwiązać dany problem. Dzięki temu książka jest nie tylko przewodnikiem po języku SQL i jego odmianach, ale także doskonałą pomocą, po którą będziesz często sięgał w swojej praktyce programistycznej.

W książce omówiono m.in.:

- Najpopularniejsze systemy bazodanowe, ich wady i zalety
- Relacyjny model danych
- Podstawy języka SQL: składnię i najważniejsze typy danych
- Pobieranie danych za pomocą zapytań
- Operatory i funkcje SQL
- Filtrowanie, grupowanie i sortowanie wyników zapytań
- Dodawanie, usuwanie i modyfikowanie danych
- Korzystanie z indeksów
- Używanie perspektyw
- Stosowanie transakcji

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Podstawy SQL-a 3

Może zauważyłeś, że w poprzednim rozdziale SQL był rzadko wspomniany. Zapamiętaj:

SQL ≠ model relacyjny

SQL jest *oparty* na modelu relacyjnym, ale nie jest w pełni jego implementacją. Odstępstwem od modelu jest na przykład to, że istnienie kluczy głównych w SQL-u jest opcjonalne, a nie obligatoryjne. W konsekwencji tabele bez kluczy głównych mogą zawierać dublujące się wiersze, co z kolei może uniemożliwiać dostęp do niektórych danych. Tematyka niniejszej książki nie obejmuje wyczerpującego opisu wspomnianych różnic (jeśli chcesz uzyskać więcej informacji, poszukaj w Internecie artykułów na temat SQL-a autorstwa E.F. Codda, Chrisa Date'a lub Fabiana Pascala). Rezultatem tych rozbieżności jest fakt, że to na użytkownikach systemów DBMS, a nie na samych systemach, spoczywa obowiązek zapewnienia struktury relacyjnej. Inną konsekwencją jest nieidealna wymienność terminów przedstawionych w tabeli 2.1 w rozdziale 2.

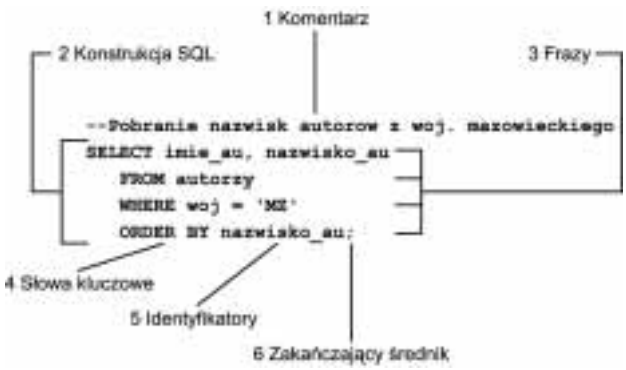
Przystępując do nauki SQL-a należy mieć na względzie powyższe uwagi.

Program SQL jest to ciąg instrukcji SQL wykonywanych w podanej kolejności. Aby napisać program, trzeba znać zasady składni SQL-a. Reguły tworzenia instrukcji SQL przedstawia niniejszy rozdział. Zawarto w nim również opis typów danych i wartości *null*.

Składnia SQL-a

Rysunek 3.1 przedstawia przykład konstrukcji SQL. Nie zwracaj na razie uwagi na jej znaczenie (semantykę) — zadaniem tego przykładu jest wyjaśnienie zasad składni SQL-a.

1. **Komentarz.** *Komentarz* jest opcjonalnym tekstem wpisywanym w osobnej linii w celu wyjaśnienia działania programu. Początek komentarza oznaczają dwa łączniki — następujący po nich tekst jest ignorowany przez system DBMS. Komentarz kończy się wraz z końcem linii.
2. **Konstrukcja SQL.** Konstrukcja SQL jest określoną kombinacją słów poprzedzonych słowem kluczowym. *Słowa* są niepodzielnymi elementami języka SQL — pod względem gramatycznym nie można rozłożyć ich na mniejsze części. Do tej grupy należą słowa kluczowe, identyfikatory, operatory, wzorce oraz inne słowa (opisane w dalszej części książki).
3. **Frazy.** Konstrukcja SQL składa się z jednej lub kilku fraz. Mówiąc ogólnie, *fraza* to fragment konstrukcji SQL rozpoczynający się słowem kluczowym. Frazy mogą być wymagane lub opcjonalne i muszą występować w określonej kolejności. Przedstawiony przykład zawiera cztery frazy: `SELECT`, `FROM`, `WHERE` i `ORDER BY`.
4. **Słowa kluczowe.** *Słowa kluczowe*, zwane także słowami zastrzeżonymi, są zarezerwowane przez SQL, ponieważ mają w tym języku specjalne znaczenie. Próba użycia słowa kluczowego w sposób inny niż pozwala na to jego kontekst (np. jako identyfikator) spowoduje wystąpienie błędu. Tabela 3.1 zawiera słowa kluczowe języka SQL. W tabeli 3.2 umieszczono potencjalne słowa kluczowe języka SQL, które na razie nie są zarezerwowane, ale w przyszłości mogą się takimi stać.
5. **Identyfikatory.** *Identyfikatory* są to słowa, których Ty (lub projektant bazy danych) używasz do nazywania obiektów bazy danych takich jak tabele, kolumny, aliasy, indeksy oraz perspektywy (widoki). Identyfikatorami nie mogą być słowa kluczowe, a ich długość może wynosić maksymalnie 128 znaków. SQL dopuszcza używanie w identyfikatorach wszystkich znaków i symboli zestawu znaków systemu Windows (włączając znaki zestawu Western oraz ideogramy), jednak zanim zaczniesz ich używać, zapoznaj się ze wskazówkami. W przedstawionym przykładzie identyfikatorami są `imie_au`, `nazwisko_au`, `autorzy` oraz `woj`.
6. **Zakańczający średnik.** Każda konstrukcja SQL kończy się średnikiem.



Rysunek 3.1. *Konstrukcja SQL zawierająca komentarz*

Tabela 3.1. *Słowa kluczowe języka SQL*

ABSOLUT	CROSS	GET	NOT	SPACE
ACTION	CURRENT	GLOBAL	NULL	SQL
ADD	CURRENT_DATE	GO	NULLIF	SQLCODE
ALL	CURRENT_TIME	GOTO	NUMERIC	SQLERROR
ALLOCATE	CURRENT_TIMESTAMP	GRANT	OCTET_LENGTH	SQLSTATE
ALTER	CURRENT_USER	GROUP	OF	SUBSTRING
AND	CURSOR	HAVING	ON	SUM
ANY	DATE	HOUR	ONLY	SYSTEM_USER
ARE	DAY	IDENTITY	OPEN	TABLE
AS	DEALLOCATE	IMMEDIATE	OPTION	TEMPORARY
ASC	DEC	IN	OR	THEN
ASSERTION	DECIMAL	INDICATOR	ORDER	TIME
AT	DECLARE	INITIALLY	OUTER	TIMESTAMP
AUTHORIZATION	DEFAULT	INNER	OUTPUT	TIMEZONE_HOUR
AVG	DEFERRABLE	INPUT	OVERLAPS	TIMEZONE_MINUTE
BEGIN	DEFERRED	INSENSITIVE	PAD	TO
BETWEEN	DELETE	INSERT	PARTIAL	TRAILING
BIT	DESC	INT	POSITION	TRANSACTION
BIT_LENGTH	DESCRIBE	INTEGER	PRECISION	TRANSLATE
BOTH	DESCRIPTOR	INTERSECT	PREPARE	TRANSLATION
BY	DIAGNOSTICS	INTERVAL	PRESERVE	TRIM
CASCADE	DISCONNECT	INTO	PRIMARY	TRUE
CASCADED	DISTINCT	IS	PRIOR	UNION
CASE	DOMAIN	ISOLATION	PRIVILEGES	UNIQUE
CAST	DOUBLE	JOIN	PROCEDURE	UNKNOWN
CATALOG	DROP	KEY	PUBLIC	UPDATE
CHAR	ELSE	LANGUAGE	READ	UPPER
CHARACTER	END	LAST	REAL	USAGE
CHAR_LENGTH	END-EXEC	LEADING	REFERENCES	USER
CHARACTER_LENGTH	ESCAPE	LEFT	RELATIVE	USING
CHECK	EXCEPT	LEVEL	RESTRICT	VALUE
CLOSE	EXCEPTION	LIKE	REVOKE	VALUES
COALESCE	EXEC	LOCAL	RIGHT	VARCHAR
COLLATE	EXECUTE	LOWER	ROLLBACK	VARYING
COLLATION	EXISTS	MATCH	ROWS	VIEW
COLUMN	EXTERNAL	MAX	SCHEMA	WHEN
COMMIT	EXTRACT	MIN	SCROLL	WHENEVER
CONNECT	FALSE	MINUTE	SECOND	WHERE
CONNECTION	FETCH	MODULE	SECTION	WITH
CONSTRAINT	FIRST	MONTH	SELECT	WORK
CONSTRAINTS	FLOAT	NAMES	SESSION	WRITE
CONTINUE	FOR	NATIONAL	SESSION_USER	YEAR
CONVERT	FOREIGN	NATURAL	SET	ZONE
CORRESPONDING	FOUND	NCHAR	SIZE	
COUNT	FROM	NEXT	SMALLINT	
CREATE	FULL	NO	SOME	

Tabela 3.2. *Potencjalne słowa kluczowe języka SQL*

AFTER	EQUALS	OLD	RETURN	TEST
ALIAS	GENERAL	OPERATION	RETURNS	THERE
ASYNC	IF	OPERATORS	ROLE	TRIGGER
BEFORE	IGNORE	OTHERS	ROUTINE	TYPE
BOOLEAN	LEAVE	PARAMETERS	ROW	UNDER
BREADTH	LESS	PENDANT	SAVEPOINT	VARIABLE
COMPLETION	LIMIT	PREORDER	SEARCH	VIRTUAL
CALL	LOOP	PRIVATE	SENSITIVE	VISIBLE
CYCLE	MODIFY	PROTECTED	SEQUENCE	WAIT
DATA	NEW	RECURSIVE	SIGNAL	WHILE
DEPTH	NONE	REF	SIMILAR	WITHOUT
DICTIONARY	OBJECT	REFERENCING	SQLEXCEPTION	
EACH	OFF	REPLACE	SQLWARNING	
ELSEIF	OID	RESIGNAL	STRUCTURE	

```
select imie_au
,           NAZWISKO_AU
              FROM
 autorzy WhErE         woj
= 'MZ' order
                 bY
nazwiskO_AU
    ;
```

Rysunek 3.2. *Nie istnieją reguły, według których trzeba formatować konstrukcję SQL. Powyższa konstrukcja jest analogiczna do konstrukcji przedstawionej na rysunku 3.1*

SQL nie definiuje precyzyjnie formy konstrukcji. Konstrukcje SQL:

- Mogą być pisane małymi lub dużymi literami (np. `SELECT` i `select` oznaczają to samo słowo kluczowe).
- Mogą być kontynuowane w kolejnych liniach. Niedopuszczalne jest tylko dzielenie słów oraz łańcuchów znaków zawartych w cudzysłowach.
- Mogą być umieszczone w jednej linii z innymi konstrukcjami.
- Mogą rozpoczynać się w dowolnej kolumnie.

Pomimo przedstawionej dowolności najlepiej jednak trzymać się konsekwentnie jednego stylu (rysunek 3.2). Ja zazwyczaj piszę słowa kluczowe dużymi literami, a każdą frazę wpisuję w osobnej linii — szczegóły wykorzystywanego przeze mnie stylu opisałem w podrozdziałach „Konwencje typograficzne" oraz „Konwencje składniowe" zawartych we Wprowadzeniu.

Częstymi błędami towarzyszącymi programowaniu w SQL-u są:

- Pomyłka w słowie kluczowym lub identyfikatorze.
- Pominięcie zakańczającego średnika.
- Zmiana kolejności wpisanych fraz.
- Nieobjęcie cudzysłowami łańcuchów znaków oraz wyrażeń oznaczających czas i datę.
- Objęcie cudzysłowami wzorców liczbowych.
- Niepoprawne skojarzenie kolumn z tabelami (np. wpisanie `SELECT procent_tantiem FROM autorzy;` zamiast `SELECT procent_tantiem FROM autorzy_tytulow;`).

Wskazówki

- Wprowadzające słowo kluczowe konstrukcji SQL nazywa się często *czasownikiem*, ponieważ wskazuje ono na wykonywaną przez konstrukcję czynność.
- Pomimo że słów kluczowych nie można używać jako identyfikatorów, to jednak można je w identyfikatorach zawierać. Np. `group` czy `max` nie są prawidłowymi identyfikatorami, ale `groups` czy `max_cena` są poprawne.
- W identyfikatorach można zawierać spacje, należy wtedy identyfikator objąć apostrofami (`'numer telefonu'`). Tworzenie nazw obiektów zawierających spacje nie jest jednak zalecane. Do oddzielania wyrazów lepiej stosować znaki podkreślenia (`numer_telefonu`) lub kombinacje małych i dużych liter (`NumerTelefonu`).
- Wyrażenie jest dowolną poprawną kombinacją symboli, która daje w wyniku pojedynczą wartość. Może zawierać operatory matematyczne i logiczne, identyfikatory, stałe, funkcje, nazwy kolumn itp. Tabela 3.3 przedstawia kilka rodzajów popularnych wyrażeń oraz ich przykłady. Typ każdego z wyrażeń zostanie szczegółowo omówiony przy okazji opisywania typów danych.
- DBMS Systemy DBMS określają własne wymagania dotyczące długości identyfikatorów oraz dostępnego dla nich zestawu znaków — szczegółowe informacje możesz znaleźć w dokumentacjach systemów pod hasłem *identyfikatory* (ang. *identifiers*) lub *nazwy* (ang. *names*).

  Systemy DBMS posiadają także dodatkowe słowa kluczowe, których nie można używać w roli identyfikatorów. Szczegóły w dokumentacjach systemów pod hasłem *słowa kluczowe* (ang. *keywords*) lub *słowa zastrzeżone* (ang. *reserved words*).
- Microsoft SQL Server, Oracle, MySQL oraz PostgreSQL akceptują zarówno komentarze jedno- jak i wieloliniowe zawarte pomiędzy symbolami komentarza `/*` i `*/`. Szczegóły w dokumentacjach systemów pod hasłem *komentarze* (ang. *comments*). MySQL dopuszcza także używanie jako oznaczenia początku komentarza znaku `#`.

Tabela 3.3. *Typy wyrażeń*

Wyrażenie	Przykład
Przypadek (`Case`)	`CASE WHEN n <> 0 THEN x/n ELSE 0 END`
Rzutowanie (`Cast`)	`CAST(data_wyd AS CHARACTER)`
Czas i data (`Datetime`)	`czas_poczatku + '01:30'`
Okres (`Interval`)	`INTERVAL '7' DAY * 2`
Liczbowe (`Numeric`)	`(sprzedanych * cena) / 12`
Łańcuch znaków (`String`)	`'Szanowny Panie ' || nazwisko_au || '.'`

Tabela 3.4. *Kategorie typów danych*

Typ danych	Przechowywany rodzaj danych
Łańcuch znaków	Ciągi znaków
Łańcuch bitowy	Ciągi bitów
Liczba dokładna	Liczby całkowite i dziesiętne
Liczba przybliżona	Liczby zmiennoprzecinkowe
Data i czas	Wartości czasu i daty
Okres	Okresy czasu

Wskazówki

- Do określania i zmiany typu danych kolumny oraz dotyczących jej ograniczeń służą konstrukcje `CREATE TABLE` oraz `ALTER TABLE` — patrz rozdział 10.
- Projektanci baz danych przywiązują bardzo dużą wagę do określania typów danych dla kolumn. Konsekwencją złego wyboru może być sytuacja, w której niemożliwe będzie wpisanie do kolumny określonych wartości, a zmiana typu danych kolumny pociągnie za sobą utratę danych już istniejących w kolumnie.
- DBMS Wiele szczegółów dotyczących typów danych ustalają producenci systemów DBMS. Stąd typy danych SQL-a nie są wiernie odwzorowane w konkretnych systemach DBMS, nawet, jeśli nazywają się identycznie. Nazwy analogicznych lub zbliżonych do teoretycznych typów danych systemów DBMS przedstawiono we wskazówkach w następnych podrozdziałach. Systemy DBMS często mają rozszerzony zestaw typów danych pozwalający przechowywać specyficzne wartości np. logiczne czy walutowe. Szczegóły dotyczące standardowych i rozszerzonych typów danych znajdziesz w dokumentacjach systemów DBMS pod hasłem *typy danych* (ang. *data types*). Informacje na temat sposobu sortowania znajdują się w podrozdziale „Sortowanie wierszy za pomocą frazy ORDER BY" w rozdziale 4.

Typy danych

W podrozdziale „Tabele, kolumny i wiersze" w rozdziale 2. stwierdzono, że dziedzina kolumny ogranicza wartości, które mogą być w danej kolumnie przechowywane. Praktyczną realizacją dziedzin są *typy danych*. Poniżej przedstawiono ich cechy.

- Każda kolumna w tabeli ma określony pojedynczy typ danych.
- Typ danych mieści się w jednej z kategorii przedstawionych w tabeli 3.4 (a opisanych w następnych sześciu podrozdziałach).
- Typ danych określa wartości, jakie dana kolumna akceptuje, oraz operacje, jakie można na niej wykonać. Np. całkowity typ danych (`integer`) reprezentuje każdą liczbę całkowitą mieszczącą się w granicach określonych przez system DBMS i pozwala na wykonywanie zwykłych operacji arytmetycznych, m.in.: dodawania, odejmowania, mnożenia i dzielenia. Ale typ całkowity nie może reprezentować wartości nieliczbowych, np. `'shadenfreude'` oraz nie umożliwia przeprowadzania operacji znakowych (np. zamiana liter małych na duże).
- Typ danych narzuca sposób sortowania kolumny. Liczby całkowite 1, 2 i 10 są sortowane arytmetycznie: `1`, `2`, `10`. Łańcuchy znaków `'1'`, `'2'` i `'10'` będą natomiast posortowane słownikowo i dadzą w wyniku kolejność: `'1'`, `'10'`, `'2'`. *Sortowanie słownikowe* polega na kolejnym porównywaniu pojedynczych znaków. Stąd `'10'` jest przed `'2'` — znak `'1'` w porządku słownikowym występuje przed znakiem `'2'`.
- Wartości wzorców przechowywane są w kolumnach. Wzorzec jest stałą, a nie wynikiem wyliczenia wartości wyrażenia liczbowego. Przykładami wzorców liczbowych są `40` i `12.34`, wzorcami znakowymi są np. `'40'` i `'zielony'`, natomiast `DATE '2002-05-10'` i `TIME '09:45:00'` to wzorce daty i czasu.

Typy łańcuchów znaków

Łańcuchy znaków używane są do reprezentowania tekstu. *Łańcuch znaków* — lub po prostu *łańcuch* — posiada następujące cechy:

- Jest uporządkowanym ciągiem zawierającym (lub nie) pewną ilość znaków.
- Jego długość może być stała lub zmienna.
- Rozróżnia wielkość liter (w czasie sortowania litera 'A' umieszczana jest przed 'a').
- W konstrukcjach SQL wzorce łańcuchowe objęte są apostrofami.
- Jest jednym z typów przedstawionych w tabeli 3.5.

Tabela 3.5. *Typy łańcuchów znaków*

Typ	Opis
CHARACTER (znakowy)	Reprezentuje stałą liczbę znaków. Łańcuch przechowywany w kolumnie zdefiniowanej jako CHARACTER(*dlugosc*) może zawierać maksymalnie *dlugosc* znaków, gdzie *dlugosc* jest liczbą całkowitą większą lub równą 1. Maksymalna długość zależy od systemu DBMS. Jeśli w kolumnie typu CHARACTER(*dlugosc*) umieszczony jest łańcuch o długości mniejszej od *dlugosc*, system DBMS wypełni brakujące miejsca spacjami tak, żeby powstał łańcuch o długości równej dokładnie *dlugosc*. Np. łańcuch 'Jacek ' w kolumnie typu CHARACTER(7) przechowywany jest jako 'Jacek '. Typy CHARACTER oraz CHAR są równoważne.
CHARACTER VARYING (znakowy o zmiennej długości)	Reprezentuje zmienną liczbę znaków. Łańcuch przechowywany w kolumnie zdefiniowanej jako CHARACTER VARYING(*dlugosc*) może zawierać maksymalnie *dlugosc* znaków, gdzie *dlugosc* jest liczbą całkowitą większą lub równą 1. Maksymalna długość zależy od systemu DBMS. W przeciwieństwie do typu CHARACTER, jeśli umieścimy w kolumnie zdefiniowanej jako CHARACTER VARYING(*dlugosc*) łańcuch o długości mniejszej niż *dlugosc*, system DBMS przechowa go w pierwotnej postaci nie wypełniając brakujących miejsc spacjami. Np. łańcuch 'Jacek ' w kolumnie typu CHARACTER VARYING(7) przechowywany jest jako 'Jacek '. Typy CHARACTER VARYING, CHAR VARYING oraz VARCHAR są równoważne.
NATIONAL CHARACTER (znakowy narodowy)	Ten typ danych odpowiada typowi CHARACTER, z tym że jest on w stanie przechowywać ustandaryzowane znaki wielobajtowe czy unikodowe (patrz ramka). W konstrukcjach SQL łańcuchy typu NATIONAL CHARACTER zapisuje się tak samo jak łańcuchy typu CHARACTER, umieszczając jedynie przed pierwszym apostrofem literę N, np. N' ą'. Typy NATIONAL CHARACTER, NATIONAL CHAR oraz NCHAR są równoważne.
NATIONAL CHARACTER VARYING (znakowy narodowy o zmiennej długości)	Ten typ danych odpowiada typowi CHARACTER VARYING z tym, że może on przechowywać ustandaryzowane znaki wielobajtowe czy unikodowe (patrz typ NATIONAL CHARACTER). Typy NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING oraz NCHAR VARYING są równoważne.

Unikod

Komputery przechowują znaki (litery, cyfry, znaki interpunkcyjne, znaki kontrolne oraz inne symbole) w swojej pamięci, przypisując im wartości liczbowe. Sposób zamiany znaków na liczby określa *kodowanie*. Różne języki i różne systemy operacyjne stosują wiele specyficznych sposobów kodowania. Standardowe łańcuchy w języku angielskim wykorzystują kodowanie ASCII, które umożliwia przypisanie liczbom 256 (2^8) znaków. Jest to niewiele — za mało, żeby przechować wszystkie litery używane w nowożytnych językach europejskich i stanowczo za mało, żeby przechować chińskie ideogramy.

Unikod jest zestawem pojedynczych znaków reprezentującym znaki prawie wszystkich języków pisanych występujących na świecie. Jest on w stanie zakodować 65536 (2^{16}) znaków. Standard unikodu ogłosiła i zarządza nim instytucja o nazwie Unicode Consortium. Aktualna specyfikacja unikodu jest zawsze dostępna w najnowszym wydaniu (internetowym lub drukowanym) standardu (*The Unicode Standard*) pod adresem *www.unicode.org*.

Wskazówki

- Jeżeli chcesz w łańcuchu wpisać znak apostrofu, musisz wpisać go dwukrotnie. Np. łańcuchem odpowiadającym angielskiemu słowu *it's* jest `'it''s'`. Znak cudzysłowu (`"`) jest oddzielnym znakiem i nie wymaga specjalnego traktowania.
- Długością łańcucha jest liczba całkowita większa lub równa 0 i mniejsza lub równa `length`. Łańcuch niezawierający żadnych znaków — `''` (dwa apostrofy bez spacji między nimi) — nazywany jest *łańcuchem pustym* lub *łańcuchem o zerowej długości*. Łańcuch pusty traktowany jest jako `VARCHAR` o długości 0.
- Systemy DBMS szybciej niż łańcuchy o zmiennej długości sortują i przetwarzają łańcuchy o stałej długości.
- DBMS W Microsoft Accessie łańcuchowymi typami danych są `tekst` oraz `nota`. W Microsoft SQL Serverze są nimi `char`, `varchar`, `text`, `nchar`, `nvarchar` oraz `ntext`. Oracle oferuje typy: `char`, `varchar`, `varchar2`, `nchar`, `nvarchar` oraz `nvarchar2`. W MySQL-u są to: `char`, `varchar`, `nchar`, `nvarchar`, `text`, `tinytext`, `mediumtext` i `longtext`, a w PostgreSQL-u `char`, `varchar` i `text`.

  Oracle puste łańcuchy traktuje jako wartości *null* — patrz podrozdział „Wartości null".

Typy łańcuchów bitowych

Łańcuch bitowy reprezentuje liczbę binarną i posiada następujące cechy:

- Jest uporządkowanym ciągiem zawierającym pewną liczbę bitów (lub niezawierającą żadnego).
- Każdy bit ma wartość 0 lub 1.
- Zazwyczaj jest wykorzystywany do przechowywania danych binarnych, takich jak pliki programów, cyfrowe dźwięki czy obrazy. Długi łańcuch bitowy nazywany jest często BLOB (ang. *binary large object*).
- W konstrukcjach SQL łańcuch znaków objęty jest apostrofami.
- Jest jednym z typów przedstawionych w tabeli 3.6.

Wskazówki

- Systemy DBMS nie próbują interpretować łańcuchów bitowych, ich znaczenie jest ważne z punktu widzenia wykorzystujących je aplikacji.
- Uwaga dla doświadczonych programistów: oprócz używania postaci binarnej (o podstawie 2) można także stosować dla łańcuchów znaków zapis szesnastkowy (ang. *hex* lub *hexadecimal*) — o podstawie 16. System szesnastkowy wykorzystuje cyfry od 0 do 9 oraz litery (duże lub małe) od A do F. Jeden znak szesnastkowy odpowiada czterem bitom. W konstrukcjach SQL łańcuchy bitowe w zapisie szesnastkowym zawierają przed pierwszym apostrofem literę `X`. Np. łańcuch szesnastkowy `X'4B'` jest odpowiednikiem łańcucha binarnego `B'01001011'`.
- DBMS W Microsoft Accessie binarnymi typami danych są `tak/nie` oraz `obiekt OLE`. W Microsoft SQL Serverze występują `binary`, `varbinary` i `image`. Oracle oferuje typy: `raw`, `long raw`, `blob` oraz `bfile`. W MySQL-u istnieją `blob`, `tinyblob`, `mediumblob` i `longblob`, a w PostgreSQL-u `bit` i `varbit`.

Tabela 3.6. *Typy łańcuchów bitowych*

Typ	Opis
`BIT` (bitowy)	Reprezentuje stałą liczbę bitów. Łańcuch bitowy przechowywany w kolumnie zdefiniowanej jako `BIT(`*dlugosc*`)` może zawierać maksymalnie *dlugosc* bitów, gdzie *dlugosc* jest liczbą całkowitą większą lub równą 1. Maksymalna długość zależy od systemu DBMS. W odróżnieniu od łańcuchów znakowych typu `CHARACTER`, próba umieszczenia w kolumnie `BIT(`*dlugosc*`)` łańcucha bitowego o długości mniejszej niż *dlugosc* spowoduje wystąpienie błędu. W konstrukcjach SQL łańcuchy bitowe zapisywane są podobnie, jak łańcuchy znakowe, jedynie przed pierwszym apostrofem umieszczona jest litera B. Np. łańcuchem typu `BIT(8)` jest `B'01001011'`.
`BIT VARYING` (bitowy o zmiennej długości)	Reprezentuje zmienną liczbę bitów. Łańcuch bitowy przechowywany w kolumnie zdefiniowanej jako `BIT VARYING(`*dlugosc*`)` może zawierać maksymalnie *dlugosc* bitów, gdzie *dlugosc* jest liczbą całkowitą większą lub równą 1. Maksymalna długość zależy od systemu DBMS. Podobnie jak w przypadku łańcuchów znakowych typu `CHAR VARYING`, jeśli umieścimy w kolumnie zdefiniowanej jako `BIT VARYING(`*dlugosc*`)` łańcuch o długości mniejszej niż *dlugosc*, system DBMS przechowa go w pierwotnej postaci nie wypełniając brakujących miejsc spacjami. Np. łańcuch `B'0101'` typu `BIT VARYING(8)` przechowywany jest jako `B'0101'`.

Typy liczb dokładnych

Typy liczb dokładnych reprezentują dokładne wartości liczbowe. *Dokładna wartość liczbowa* posiada następujące cechy:

- Może być ujemna, dodatnia lub równa zero.
- Jest liczbą całkowitą lub dziesiętną. *Liczby całkowite* nie zawierają przecinka dziesiętnego (kropki w zapisie komputerowym), np. –39, 0, 62262. *Liczby dziesiętne* zawierają także cyfry po prawej stronie przecinka np. –22.06, 0.0, 0.0003, 12.34.
- Posiada stałą rozdzielczość i dokładność. *Rozdzielczością* określa się liczbę cyfr znaczących wykorzystywanych do przedstawiania liczby — liczba ta obejmuje zarówno cyfry z prawej, jak i z lewej strony przecinka dziesiętnego. *Dokładność* jest to liczba cyfr po przecinku dziesiętnym. Dokładność oczywiście nie może przekroczyć rozdzielczości. Ustalenie dokładności na poziomie zero powoduje przedstawienie liczby w postaci całkowitej. We wskazówkach zawarto kilka przykładów.
- Należy do jednego z typów przedstawionych w tabeli 3.7.

Tabela 3.7. *Typy liczb dokładnych*

Typ	Opis
NUMERIC (liczbowa)	Reprezentuje liczbę dziesiętną. Liczby dziesiętne są przechowywane w kolumnach zdefiniowanych jako NUMERIC(*rozdzielczosc*[, *dokladnosc*]). *rozdzielczosc* jest liczbą większą lub równą 1 — maksymalna wielkość zależy od systemu DBMS. *dokladnosc* jest liczbą z przedziału od 0 do *rozdzielczosc*. Jeśli *dokladnosc* nie jest jawnie określona, domyślnie przyjmowana jest jako 0 (co powoduje, że w rzeczywistości typ staje się całkowity).
DECIMAL (dziesiętna)	Ten typ danych jest analogiczny do typu NUMERIC, niektóre systemy DBMS traktują je równoważnie. Różnica jest taka, że system DBMS może określić *rozdzielczosc* liczby jako większą niż ustalono to w definicji DECIMAL(*rozdzielczosc*[, *dokladnosc*]) — *rozdzielczosc* jest minimalnym, a nie rzeczywistym rozmiarem przedstawianej liczby, tak jak było to w przypadku typu NUMERIC. Słowa DEC i DECIMAL są synonimami.
INTEGER (całkowita)	Reprezentuje liczbę całkowitą. Minimalna i maksymalna wartość liczby, jaka może być przechowywana w kolumnie zdefiniowanej jako INTEGER, zależy od systemu DBMS. Definicja typu INTEGER nie wymaga określania argumentów. Słowa INTEGER oraz INT są synonimami.
SMALLINT (całkowita krótka)	Ten typ jest takim samym typem jak INTEGER, posiada tylko węższy zakres dopuszczalnych wartości — zależny od systemu DBMS. Definicja tego typu nie wymaga określania argumentów.

Wskazówki

- Tabela 3.8 przedstawia sposób, w jaki liczba 123.89 jest przechowywana dla różnych wartości rozdzielczości i dokładności.
- Wartości liczbowych nie umieszcza się w cudzysłowach.
- Jeśli liczby nie są stosowane do żadnych obliczeń (np. numery telefonów czy kody pocztowe), lepiej przechowywać je jako łańcuchy znaków. Zapobiegnie to ewentualnej utracie informacji — na przykład przechowanie kodu `'00950'` jako liczby całkowitej spowoduje utratę początkowych zer.
- Obliczenia wykonywane tylko na liczbach całkowitych są znacznie szybsze od obliczeń, których argumentami są liczby dziesiętne lub zmiennoprzecinkowe.
- DBMS Typami liczb dokładnych w Microsoft Accessie są `dziesiętne`, `liczba całkowita`, `bajt` oraz `liczba całkowita długa`. Microsoft SQL Server oferuje typy: `numeric`, `decimal`, `integer`, `smallint`, `bigint` oraz `tinyint`. W Oracle'u wystepują: `numeric`, `decimal`, `integer`, `smallint` oraz `number`. MySQL oferuje typy `numeric`, `decimal`, `integer`, `smallint`, `bigint`, `mediumint` i `tinyint`, natomiast PostgreSQL `numeric`, `decimal`, `integer`, `smallint` oraz `bigint`.

Tabela 3.8. *Przykłady rozdzielczości i dokładności dla liczby 123.89*

Definicja typu	Przechowywana wartość
`NUMERIC(5)`	124
`NUMERIC(5,0)`	124
`NUMERIC(5,1)`	123.9
`NUMERIC(5,2)`	123.89
`NUMERIC(4,0)`	124
`NUMERIC(4,1)`	123.9
`NUMERIC(4,2)`	Poza zakresem rozdzielczości
`NUMERIC(2,0)`	Poza zakresem rozdzielczości

Typy liczb przybliżonych

Typy liczb przybliżonych służą do reprezentowania przybliżonych wartości liczbowych. *Przybliżona wartość liczbowa* posiada następujące cechy:

- Może być liczbą ujemną, dodatnią lub równą zero.
- Jest traktowana jako przybliżenie liczby zmiennoprzecinkowej (rzeczywistej).
- Zazwyczaj wykorzystywana jest do reprezentowania liczb bardzo dużych lub bardzo małych oraz do obliczeń naukowych.
- Wyrażana jest w postaci naukowej. Wartość w *notacji naukowej* zapisuje się jako liczbę dziesiętną mnożoną przez całkowitą potęgę liczby 10. Duża litera `E` jest symbolem eksponenta np. 2.5E2 = $2.5*10^2 = 250$. *Mantysą* w takim zapisie są cyfry znaczące (w podanym przykładzie 2.5), natomiast *eksponentem* jest wykładnik potęgi liczby 10 (w podanym przykładzie 2). Zarówno mantysa, jak i eksponent, mogą mieć określony znak: $-2.5E-2 = -2.5*10^{-2} = -0.025$.
- Posiada stałą rozdzielczość, ale nie posiada dokładności w ścisłym sensie tego słowa (określają ją znak oraz wartość eksponenta). Rozdzielczość jest liczbą bitów wykorzystywanych do zapisu mantysy. Aby przekształcić rozdzielczość bitową na dziesiętną, należy ją pomnożyć przez liczbę 0.30103. Przekształcenie rozdzielczości dziesiętnej na bitową można uzyskać, mnożąc ją przez liczbę 3.32193. Na przykład 24 bity oznaczają rozdzielczość 7-cyfrową, a 53 bity to rozdzielczość 15-cyfrowa.
- Należy do jednego z typów wymienionych w tabeli 3.9.

Wskazówki

- Wartości liczbowych nie umieszcza się w cudzysłowie.
- DBMS Typami liczb przybliżonych w Microsoft Accessie są `pojedyncza precyzja` i `podwójna precyzja`. W Microsoft SQL Serverze występują typy `float` i `real`. Oracle oferuje typy `float`, `real`, `double precision` oraz `number`. W MySQL-u istnieją typy `float`, `real` i `double`, natomiast w PostgreSQL-u `real` i `double precision`.

Tabela 3.9. *Typy liczb przybliżonych*

Typ	Opis
`FLOAT` (zmiennoprzecinkowa)	Reprezentuje liczbę zmiennoprzecinkową. Liczba zmiennoprzecinkowa przechowywana w kolumnie zdefiniowanej jako `FLOAT(`*`rozdzielczosc`*`)` posiada określoną w definicji rozdzielczość — liczbę większą lub równą 1 wyrażającą ilość bitów, a nie cyfr. Maksymalna rozdzielczość zależy od systemu DBMS.
`REAL` (rzeczywista)	Ten typ jest analogiczny do typu `FLOAT`, z tym że rozdzielczość ustala system DBMS. Liczby typu `REAL` często nazywa się liczbami pojedynczej precyzji. Definicja typu `REAL` nie wymaga określania argumentów.
`DOUBLE PRECISION` (podwójnej precyzji)	Jest to taki sam typ, jak `FLOAT`. Różnica jest tylko taka, że rozdzielczość typu `DOUBLE PRECISION` ustalana przez system DBMS jest większa od rozdzielczości typu `REAL`. Definicja typu `DOUBLE PRECISION` nie wymaga określania argumentów.

Typy daty i czasu

Typy daty i czasu, jak sama nazwa wskazuje, służą do reprezentowania *dat i czasu*. Wartości tych typów posiadają następujące cechy:

- Są określane względem czasu UTC (ang. *Universal Coordinated Time*) zwanego oficjalnie czasem uniwersalnym Greenwich (ang. GMT — *Greenwich Mean Time*). Standard SQL-92 wymaga, aby każda sesja SQL posiadała domyślne przesunięcie czasowe względem czasu UTC, które jest wykorzystywane w trakcie trwania sesji. Na przykład przesunięciem dla strefy czasowej San Francisco w Kalifornii jest –8.
- Reguły rządzące kalendarzem gregoriańskim określają w naturalny sposób format zapisu dat. Systemy DBMS odrzucają wartości, których nie mogą zinterpretować jako daty.
- Wartości czasowe oparte są na 24-godzinnym formacie czasu (np. 13:00 zamiast 1:00 po południu).
- Łączniki (-) oddzielają kolejne części dat, natomiast dwukropki (:) oddzielają kolejne części wartości czasowych. Jeśli jedna wartość zawiera jednocześnie datę i czas, to są one oddzielone spacją.
- Należą do jednego z typów przedstawionych w tabeli 3.10.

Tabela 3.10. *Typy daty i czasu*

Typ	Opis
DATE (data)	Reprezentuje datę. Data przechowywana w kolumnie zdefiniowanej jako DATE zawiera trzy liczby całkowite: rok (YEAR), miesiąc (MONTH) i dzień (DAY) sformatowane według schematu *rrrr-mm-dd* (długość 10), np. 2002-06-14. Tabela 3.11 przedstawia poprawne wartości dla pól tego typu. Definicja typu DATE nie wymaga określania argumentów.
TIME (czas)	Reprezentuje czas dnia. Czas przechowywany w kolumnie zdefiniowanej jako TIME zawiera trzy pola: godzinę (HOUR), minuty (MINUTE) oraz sekundy (SECOND) sformatowane według szablonu *gg:mm:ss* (długość 8), np. 22:06:57. Opcjonalnie można zdefiniować kolumnę jako TIME(*rozdzielczosc*), gdzie *rozdzielczosc* będzie liczbą większą lub równą zero oznaczającą liczbę cyfr określających części sekundy. Maksymalna *rozdzielczosc* zależy od systemu DBMS, natomiast minimalna wynosi 6. Wartości HOUR oraz MINUTE są liczbami całkowitymi, a SECOND jest liczbą dziesiętną. Szablonem wartości czasu zawierającym ułamkowe części sekund jest *gg:mm:ss.ssss...* (długość 9 + liczba cyfr ułamkowych), np. '22:06:57.1333'. Tabela 3.11 przedstawia poprawne wartości pól tego typu.
TIMESTAMP (stempel czasowy)	Reprezentuje kombinację wartości DATE i TIME oddzielonych spacją. Formatem wartości tego typu jest *rrrr-mm-dd gg:mm:ss* (długość 19), np. 2002-06-14 22:06:57. Można także określić za pomocą definicji TIMESTAMP(*rozdzielczosc*) ułamkowe części sekund — format zapisu wygląda wtedy następująco: *rrrr-mm-dd gg:mm:ss.ssss...* (długość 20 + liczba cyfr ułamkowych).
TIME WITH TIME ZONE (czas ze strefą czasową)	Ten typ jest typem analogicznym do typu DATA, z tym że zawiera dodatkowe pole — TIME_ZONE_OFFSET (przesunięcie strefy czasowej) — określające przesunięcie w godzinach względem czasu UTC. Wartość TIME_ZONE_OFFSET sformatowana jest jako INTERVAL HOUR TO MINUTE (patrz następny podrozdział) i może zawierać wartości przedstawione w tabeli 3.11. Uwzględnienie strefy czasowej osiąga się poprzez dodanie do wartości typu TIME zwrotu: AT TIME ZONE *time_zone_offset*, np. 22:06:57 AT TIME ZONE -08:00. Drugą możliwością jest dodanie zwrotu AT LOCAL, co wskazuje, że strefa czasowa powinna być domyślna dla całej sesji, np. 22:06:57 AT LOCAL. Jeśli fraza AT nie jest jawnie określona, wszystkie wartości czasu są domyślnie przyjmowane jako AT LOCAL.
TIMESTAMP WITH TIME ZONE (stempel czasowy ze strefą czasową)	Ten typ odpowiada typowi TIMESTAMP, zawiera tylko dodatkowe pole TIME_ZONE_OFFSET określające przesunięcie w godzinach względem czasu UTC. Zasady składniowe wyrażeń tego typu są analogiczne, co w przypadku typu TIME WITH TIME ZONE, trzeba jedynie na początku umieścić datę np. 2002-06-14 22:06:57 AT TIME ZONE -08:00.

Tabela 3.11. *Pola typu daty i czasu*

Pole	Poprawna wartość
YEAR	od 0001 do 9999
MONTH	od 01 do 12
DAY	od 01 do 31
HOUR	od 00 do 23
MINUTE	od 00 do 59
SECOND	od 00 do 61.999... (patrz wskazówki)
TIME_ZONE_OFFSET	od -12:59 do +13:00

- DBMS Typem daty i czasu w Microsoft Accessie jest `data/godzina`. Microsoft SQL Server oferuje typy `datetime` oraz `smalldatetime`. W Oracle'u są to `date` i `timestamp`. MySQL oferuje typy `date`, `time`, `datetime` i `timestamp`, natomiast PostgreSQL `date`, `time` i `timestamp`.

  Systemy DBMS pozwalają na wpisywanie wartości dat w formatach *miesiąc-dzień-rok*, *dzień-miesiąc-rok* oraz w innych, a wartości czasu w postaci *A.M./P.M.*. Format, w jakim wartości dat i czasu są wyświetlane może różnić się od tego, w jakim zostały wprowadzone.

  W Microsoft Accessie wzorców dat i czasu nie umieszcza się w apostrofach, lecz pomiędzy znakami # oraz nie wpisuje się nazwy typu.

  W 4Microsoft SQL Serverze także nie wpisuje się typu danych przed wzorcami dat i czasu.

Wskazówki

- Opis pobierania czasu systemowego znajdziesz w podrozdziale „Pobieranie bieżącej daty i czasu" w rozdziale 5.
- Wartości daty i czasu można porównywać, jeśli zawierają te same pola (patrz „Filtrowanie wierszy za pomocą frazy WHERE" w rozdziale 4. oraz „Obliczenia operujące na wartościach daty i czasu").
- Pole `SECOND` może zawierać wartości mniejsze lub równe 61.999... (zamiast 59.999...), co pozwala na uwzględnianie w określonej minucie „sekund przestępnych" w celu zachowania zgodności zegara zdefiniowanego przez człowieka z naturalnym zegarem Ziemi.
- Wzorce daty i czasu wpisuje się, rozpoczynając od słowa oznaczającego typ, potem następuje spacja i wartość daty lub czasu umieszczona w apostrofach, czyli `DATE 'rrrr-mm-dd'`, `TIME 'gg:mm:ss'` lub `TIMESTAMP 'rrrr-mm-dd gg:mm:ss'`.
- Standard SQL-92 nie przewiduje możliwości przechowywania dat *p.n.e.* – przed naszą erą czy przed Chrystusem (*p. Ch*), ale Twój system DBMS może taką możliwość oferować.
- Stemple czasowe są w praktyce często wykorzystywane do konstruowania unikalnych kluczy lub zaznaczania zdarzeń związanych z wierszem, w którym występują.
- Typ danych `TIME WITH TIME ZONE` właściwie nie ma sensu, gdyż w rzeczywistości strefy czasowe nie są jednoznaczne bez określonej daty (przesunięcie czasowe zmienia się w ciągu roku). Lepiej jest stosować typ `TIMESTAMP WITH TIME ZONE`.

Typy interwałowe (okresowe)

Zgodność systemów DBMS ze standardem SQL-92 w zakresie typów okresowych jest niewielka, stąd niniejszy rozdział należy traktować jako w dużej mierze teoretyczny. Systemy DBMS posiadają własne rozszerzone typy danych oraz funkcje służące do obliczania okresów i wykonywania obliczeń na wartościach daty i czasu.

Typy okresowe reprezentują okresy pomiędzy określonymi datami lub godzinami. *Wartość okresowa* posiada następujące cechy:

- Przechowuje okres czasu, jaki upływa pomiędzy dwoma wartościami daty lub czasu. Np. pomiędzy godzinami 09:00 a 13:30 okres wynosi 04:30. Okres powstaje po odjęciu od siebie dwóch wartości typu daty lub czasu.
- Może być dodawana lub odejmowana od wartości daty lub czasu — patrz „Obliczenia operujące na wartościach daty i czasu" w rozdziale 5.
- Zawiera takie same pola, jak wartości daty i czasu (`YEAR`, `HOUR`, `SECOND` itd.), ale może być poprzedzona znakiem + (w przód) lub - (w tył) w celu określenia kierunku upływu czasu. Znaki oddzielające kolejne pola są takie same, jak dla wartości daty i czasu.
- Występuje w dwóch postaciach: jako okres *rok-miesiąc* lub jako *dzień-czas*. Okres pierwszego typu wyrażony jest liczbą lat i całkowitą liczbą miesięcy, natomiast okres typu *dzień-czas* wyraża liczbę dni, godzin, minut i sekund.
- Może posiadać kwalifikator jedno- lub kilkupolowy. Kwalifikator jednopolowy określany jest po prostu jako `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE` lub `SECOND`. Natomiast kwalifikator wielopolowy zapisuje się w następującej postaci:

  ```
  pole_poczatkowe TO pole_koncowe
  ```

- *pole_poczatkowe* może być polem `YEAR`, `DAY`, `HOUR` lub `MINUTE`, a *pole_koncowe* jednym z pól: `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE` lub `SECOND`. *pole_koncowe* musi oznaczać mniejszą jednostkę czasu, niż *pole_poczatkowe*.

Tabela 3.12. *Typy okresowe*

Typ	Opis
Year-month (rok-miesiąc)	Takie okresy zawierają tylko ilość lat, ilość miesięcy lub obie z nich. Poprawnymi kolumnami są: INTERVAL YEAR, INTERVAL YEAR(*rozdzielczosc*), INTERVAL MONTH, INTERVAL MONTH(*rozdzielczosc*), INTERVAL YEAR TO MONTH, INTERVAL YEAR(*rozdzielczosc*) TO MONTH.
Day-time (dzień-czas)	Okresy tego typu mogą zawierać ilość dni, godzin, minut, sekund lub różne ich kombinacje. Przykładami poprawnych typów kolumn są: INTERVAL MINUTE, INTERVAL DAY(*rozdzielczosc*), INTERVAL DAY TO HOUR, INTERVAL DAY(*rozdzielczosc*) TO SECOND, INTERVAL MINUTE(*rozdzielczosc*) TO SECOND(*rozdzielczosc_ulamkowa*).

Kolumna jednopolowa zdefiniowana jako INTERVAL HOUR może przechowywać okresy typu „4 godziny" czy „25 godzin". Kolumna wielopolowa zdefiniowana jako INTERVAL DAY TO MINUTE jest zdolna do przechowywania wartości typu „2 dni, 5 godzin i 10 minut".

- Kolumna jednopolowa może posiadać rozdzielczość określającą długość (liczbę pozycji) pola, np. INTERVAL HOUR(2). Jeśli rozdzielczość nie jest jawnie określona, przyjmowana jest domyślnie jako 2. Pole typu SECOND może mieć dodatkowo określoną rozdzielczość ułamkową, czyli liczbę cyfr po przecinku dziesiętnym — np. INTERVAL SECOND(5, 2). Jeśli nie jest ona jawnie określona, przyjmowana jest domyślnie jako 6.

  Kolumna wielopolowa może mieć określoną rozdzielczość dla pola *pole_poczatkowe*, ale nie dla *pole_koncowe* (chyba, że *pole_końcowe* jest typu SECOND, wtedy można określić rozdzielczość ułamkową), np. INTERVAL DAY(3) TO MINUTE lub INTERVAL MINUTE(2) TO SECOND(4).

- Należy do jednego z typów przedstawionych w tabeli 3.12.

Wskazówki

- Aby wprowadzić wzorzec okresu, należy wpisać słowo INTERVAL, następnie spację, a potem wartość okresu umieszczoną w apostrofach, np. INTERVAL '15-3' dla wartości 15 lat i 3 miesięcy lub INTERVAL '-22:06:5.5' dla określenia okresu 22 godzin, 6 minut i 5,5 sekundy temu.
- DBMS Microsoft Access i Microsoft SQL Server oraz MySQL nie akceptują okresowych typów danych. Oracle i PostgreSQL oferują typ interval.

Wartości null

Kiedy zestaw Twoich danych nie jest kompletny, jako oznaczenia brakującej czy nieznanej wartości możesz użyć wartości pustej — *null*. Wartość *null* posiada następujące cechy:

- W konstrukcjach SQL wartość *null* jest reprezentowana przez słowo kluczowe `NULL`.
- Wartość określana jest jako *null*, jeśli nigdy nie będzie ona znana, jeśli istnieje możliwość, że będzie określona później lub jeśli jest nieodpowiednia (wartość *null* można sobie wyobrazić raczej jako zaznaczenie wolnego miejsca niż jako rzeczywistą wartość).
- Wartość *null* to nie to samo co zero, łańcuch zawierający tylko puste miejsca czy łańcuch pusty (`''`). Wartość *null* w kolumnie `cena` nie oznacza, że dana pozycja nie ma ceny lub że cena wynosi zero — oznacza to, że cena jest nieznana lub nie została jeszcze ustalona (wyjątkiem jest Oracle, który akceptuje puste łańcuchy — patrz wskazówka **DBMS**).
- Wartości *null* nie należą do żadnego typu danych i mogą być umieszczane we wszystkich kolumnach z wyjątkiem tych zdefiniowanych jako `NOT NULL` — patrz „Nie akceptowanie w kolumnach wartości null za pomocą wyrażenia NOT NULL" w rozdziale 10.
- Wartości *null* mogą być wykrywane za pomocą wyrażenia `IS NULL` — patrz „Sprawdzanie występowania wartości null za pomocą słowa kluczowego IS NULL" w rozdziale 4.
- Wartości *null* nie są równe żadnym innym. Nie można określić czy wartość *null* jest równa jakiejś innej wartości, nawet innej wartości *null*. Taka sytuacja jest punktem wyjścia do logiki trójwartościowej — patrz „Łączenie i negacja warunków za pomocą operatorów AND, OR i NOT" w rozdziale 4.
- Chociaż wartości *null* nigdy nie są sobie równe, to jednak słowo kluczowe `DISTINCT` traktuje wszystkie wartości *null* znajdujące się w danej kolumnie jako duplikaty — patrz „Eliminowanie powtarzających się wierszy za pomocą słowa kluczowego DISTINCT" w rozdziale 4.
- W czasie sortowania kolumny zawierającej wartości *null* mogą one być większe lub mniejsze od wszystkich innych wartości występujących w kolumnie — zależy to od systemu DBMS. Patrz „Sortowanie wierszy za pomocą frazy ORDER BY" w rozdziale 4.
- Wartości *null* „propagują się" poprzez obliczenia. Wynik każdego wyrażenia zawierającego wartość *null* wynosi *null*: `(12 * NULL) / 4 = NULL`. Patrz rozdział 5.
- Funkcje agregujące, takie jak `SUM()`, `AVG()` czy `MAX()` ignorują w trakcie obliczeń wartości *null*. Patrz rozdział 6.
- Podczas grupowania za pomocą frazy `GROUP BY` kolumn zawierających wartości *null*, są one wszystkie umieszczane w jednej grupie — patrz „Grupowanie wierszy za pomocą frazy GROUP BY" w rozdziale 6.
- Wartości *null* mają wpływ na wynik złączeń — patrz „Złączenia" w rozdziale 7.
- Wartości *null* mogą sprawiać kłopoty w przypadku podzapytań — patrz „Wartości null w podzapytaniach" w rozdziale 8.

```
SELECT MAX(id_au)
  FROM autorzy
  WHERE nazwisko_au = 'XXX';

MAX(id_au)
-----------
NULL
```

Rysunek 3.3. *Otrzymanie wartości null z kolumny, która nie pozwala na istnienie wartości null*

- DBMS Sposób wyświetlania w wynikach wartości *null* zależy od systemu DBMS. Wartości puste mogą być wyświetlane np. jako `NULL`, `(NULL)`, `<NULL>` lub jako puste pola.

  Obecna wersja Oracle'a traktuje puste łańcuchy (`''`) jako wartości *null*. Takie rozwiązanie w kolejnych wersjach raczej nie będzie stosowane, stąd Oracle już teraz radzi nie opierać się na takiej jednoznaczności. Może to spowodować problemy z konwersjami danych do innych systemów DBMS. W przykładowej bazie danych kolumna `imie_au` w tabeli `autorzy` zdefiniowana jest jako `NOT NULL`. W Oracle'u imieniem autora o nazwisku Sitkowski (autor `A06`) jest pojedyncza spacja (`' '`), natomiast w innych systemach w tym miejscu występuje pusty łańcuch (`''`). Informacje na temat przykładowej bazy danych znajdują się w rozdziale 2.

Wskazówki

- Z wartościami *null* związanych jest tyle problemów i komplikacji — ważniejsze z nich zostały opisane w niniejszym podrozdziale — że niektórzy znawcy baz danych zalecają użytkownikom rezygnację z ich używania (zamiast tego proponują stosowanie wartości domyślnych lub innych znaczników braku danych). Chociaż więc do niektórych zastosowań wartości *null* są niezbędne, to jednak zalecana jest jak największa powściągliwość w ich używaniu. Konkluzja jest taka: wyniki zawierające wartości *null* należy interpretować ostrożnie.
- Przeczytaj także podrozdziały „Sprawdzanie występowania wartości null za pomocą funkcji COALESCE()" oraz „Porównywanie wyrażeń za pomocą funkcji NULLIF()" w rozdziale 5.
- Określenie *wartość null* nie jest precyzyjne — *null* oznacza brak wartości.
- Słowa kluczowego `NULL` nie należy umieszczać w apostrofach — system DBMS zinterpretuje je wtedy jako łańcuch znaków `'NULL'`, a nie jako wartość pustą.
- Wartość *null* można także otrzymać z kolumny, która na istnienie wartości *null* nie pozwala. Kolumna `id_au` w tabeli `autorzy` nie pozwala na istnienie wartości *null*, ale konstrukcja `SELECT` przedstawiona na rysunku 3.3 zwraca jako wartość maksymalną `id_au` właśnie wartość pustą.