helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

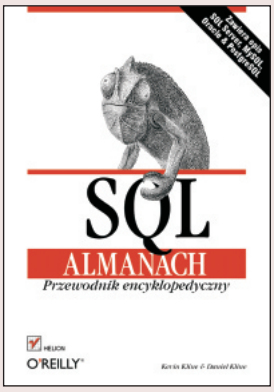
TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

SQL. Almanach.

Opis poleceń języka



Autorzy: Kevin Kline, Daniel Kline
Tłumaczenie: Paweł Janociński
ISBN: 83-7197-595-3
Tytuł oryginału: SQL in a Nutshell
Format: B5, stron: około 200

SQL (Structured Query Language - strukturalny język zapytań) jest standardowym językiem zapytań przeznaczonym do pobierania informacji z baz danych. Historycznie, był to język systemów zarządzania bazami danych działających na minikomputerach i komputerach mainframe. Z czasem został jednak zaadoptowany do systemów PC obsługujących rozproszone bazy danych i pozwalających użytkownikom sieci lokalnych na jednoczesny dostęp do tych samych danych, pomimo istnienia różnych dialektów języka SQL,
SQL. Almanach. Opis poleceń języka jest praktycznym i użytecznym leksykonem poleceń najnowszej wersji standardu języka SQL (SQL99). Pozwoli czytelnikowi poznać sposób, w jaki jego ulubiony system baz danych obsługuje dowolne polecenie standardu SQL. Niniejsza książka prezentuje każdą instrukcję języka SQL i opisuje jej użycie zarówno w implementacjach komercyjnych (Microsoft SQL Server 2000 i Oracle 8i) jak i open source (MySQL i PostgreSQL 7.0). Opis każdego polecenia zawiera jego składnię, opis i przykłady ilustrujące najważniejsze pojęcia i zastosowania.

SQL. Almanach. Opis poleceń języka jest czymś więcej niż tylko leksykonem dla doświadczonych programistów SQL, analityków czy administratorów baz danych. Jest również wspaniałym źródłem wiedzy dla początkujących użytkowników SQL i tych, dla których bazy danych są narzędziem pomocniczym. Dotyczy to administratorów systemów, użytkowników pakietów produktów klient/serwer i konsultantów, którzy muszą znać różne dialekty SQL na wielu platformach.

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

Spis treści

Wprowadzenie *7*

Rozdział 1. *SQL, implementacje dostawców i zarys historii* *11*

Relacyjny model baz danych 11
Bazy danych opisane w tej książce 12
Standard SQL 12
Klasy instrukcji w SQL99 15
Dialekty języka SQL 16
Kryteria relacyjności baz danych 17

Rozdział 2. *Podstawowe pojęcia* *19*

Przetwarzanie rekordów kontra przetwarzanie zbiorów 19
Model relacyjny 20
SQL99 i typy danych wprowadzone przez dostawców 21
Obsługa wartości NULL 29
Kategorie składni 29
Stosowanie języka SQL 34
Podsumowanie 37

Rozdział 3. *Polecenia języka SQL* *39*

Jak korzystać z tego rozdziału 39
Krótki spis poleceń języka SQL 39
Instrukcje DROP 104
Podsumowanie 168

Rozdział 4. *Funkcje języka SQL* *169*
Funkcje deterministyczne i niedeterministyczne 169
Typy funkcji 169
Rozszerzenia dostawców 180

Rozdział 5. *Niezaimplementowane polecenia SQL99* *205*

Dodatek A *Słowa kluczowe SQL99 i wprowadzone przez dostawców* *209*

Skorowidz *219*

3

Polecenia języka SQL

Niniejszy rozdział stanowi główną część książki *SQL. Almanach*. Są w nim wypisane w porządku alfabetycznym polecenia języka SQL wraz z dokładnym omówieniem i przykładami zastosowania. Każde polecenie jest opisane jako „obsługiwane", „obsługiwane ze zmianami", „obsługiwane z ograniczeniami" lub „nie obsługiwane" dla każdego z czterech opisanych w niniejszej książce dialektów języka SQL: SQL Server, MySQL, Oracle i PostgreSQL. Po krótkim opisie standardu SQL99 umieszczono zwięzłe, ale dokładne omówienie implementacji każdego z dostawców wraz z przykładami i fragmentami kodu.

Jak korzystać z tego rozdziału

Czytanie opisu konkretnego polecenia SQL warto rozpocząć od wstępnego akapitu zawierającego tabelę z informacjami o sposobie obsługi przez dostawców i podpunktu zawierającego składnię i opis polecenia w standardzie SQL99. Jest to ważne, ponieważ wszystkie cechy wspólne standardu i implementacji konkretnego producenta są omówione w opisie SQL99. Dlatego podpunkt dotyczący dostawcy może nie zawierać wszystkich aspektów stosowania polecenia, gdyż niektóre z nich są opisane wcześniej.

Krótki spis poleceń języka SQL

Poniższa lista zawiera użyteczne wskazówki dotyczące czytania tabeli 3.1 oraz pochodzenia stosowanych w niej skrótów. Poniżej tabeli następuje szczegółowe omówienie zawartych w niej poleceń.

1. Pierwsza kolumna zawiera alfabetyczny spis poleceń języka SQL.
2. W drugiej kolumnie przedstawiono klasę, do której należy dane polecenie.
3. Trzecia kolumna zawiera informację na temat obsługi polecenia w SQL99.
4. Kolejne kolumny opisują sposób obsługi polecenia w implementacjach dostawców:

Obsługiwane (O)
: Polecenie jest obsługiwane zgodnie ze standardem.

Obsługiwane ze zmianami (OZ)

Dostawca wspiera standard SQL99, ale używa własnego kodu albo składni.

Obsługiwane z ograniczeniami (OO)

Dostawca obsługuje niektóre, ale nie wszystkie funkcje określone w SQL99 dla tego polecenia.

Nie obsługiwane (NO)

Dostawca nie obsługuje danego polecenia zgodnie ze standardem SQL99.

5. Warto pamiętać, że nawet jeśli polecenie jest oznaczone jako „nie obsługiwane", istnieje zazwyczaj stworzona przez dostawcę alternatywna metoda wykonywania tych samych działań czy funkcji. Należy zatem przeczytać także omówienie i przykłady stosowania tego polecenia w dalszej części niniejszego rozdziału.

Tabela 3.1. Krótki, alfabetyczny spis poleceń języka SQL

Polecenie	Klasa polecenia	SQL 99	Microsoft SQL Server	MySQL	Oracle	Postgre SQL
ALTER PROCEDURE	SQL-schemat	tak	OZ	NO	OZ	NO
ALTER TABLE	SQL-schemat	tak	OZ	OO	OZ	OZ
ALTER TRIGGER	SQL-schemat	nie	OZ	NO	OZ	NO
ALTER VIEW	SQL-schemat	nie	OZ	NO	OZ	NO
CALL	SQL-kontrola	tak	NO	NO	O	O
CASE	SQL-dane	tak	O	O	NO	O
CAST	SQL-dane	tak	O	NO	NO	O
CLOSE CURSOR	SQL-dane	tak	O	NO	O	O
COMMIT TRANSACTION	SQL-transakcje	tak	OZ	NO	O	O
operatory konkatenacji	SQL-dane	tak	OZ	OZ	O	O
CONNECT	SQL-połączenia	tak	OO	NO	O	NO
CREATE DATABASE	SQL-schemat	nie	OZ	O	O	OZ
CREATE FUNCTION	SQL-schemat	tak	OZ	OZ	OZ	OZ
CREATE INDEX	SQL-schemat	tak	OZ	OZ	OZ	OZ
CREATE PROCEDURE	SQL-schemat	tak	O	NO	O	NO
CREATE ROLE	SQL-schemat	tak	NO	NO	OZ	NO
CREATE SCHEMA	SQL-schemat	tak	O	NO	O	NO
CREATE TABLE	SQL-schemat	tak	OZ	OZ	OZ	OZ
CREATE TRIGGER	SQL-schemat	tak	OZ	NO	OZ	OZ
CREATE VIEW	SQL-schemat	tak	OZ	NO	OZ	OZ
DECLARE CURSOR	SQL-dane	tak	O	NO	O	O
DELETE	SQL-dane	tak	OZ	OZ	O	O
DISCONNECT	SQL-połączenia	tak	OO	NO	OZ	NO
DROP DATABASE	SQL-schemat	tak	OZ	OZ	NO	OZ

Tabela 3.1. Krótki, alfabetyczny spis poleceń języka SQL (ciąg dalszy)

Polecenie	Klasa polecenia	SQL 99	Microsoft SQL Server	MySQL	Oracle	Postgre SQL
DROP FUNCTION	SQL-schemat	tak	OZ	OZ	OZ	OZ
DROP INDEX	SQL-schemat	tak	OZ	OZ	OZ	OZ
DROP PROCEDURE	SQL-schemat	tak	O	NO	O	NO
DROP ROLE	SQL-schemat	tak	NO	NO	OZ	NO
DROP TABLE	SQL-schemat	tak	OZ	OZ	OZ	OZ
DROP TRIGGER	SQL-schemat	tak	OZ	NO	OZ	OZ
DROP VIEW	SQL-schemat	tak	O	NO	O	O
FETCH	SQL-dane	tak	O	NO	O	OZ
GRANT	SQL-schemat	tak	OZ	OZ	OZ	OZ
INSERT	SQL-schemat	tak	OZ	OZ	O	O
klauzula JOIN	SQL-dane	tak	O	OO	NO (obsługa złączeń theta)	OZ (obsługa złączeń theta)
operator LIKE	SQL-schemat	tak	OZ	OZ	OZ	OZ
OPEN	SQL-schemat	tak	O	NO	O	O
OPERATORS	SQL-schemat	tak	OZ	OZ	OZ	OZ
RETURN	SQL-kontrola	tak	O	O	O	O
REVOKE	SQL-schemat	tak	OZ	OZ	OZ	OZ
ROLLBACK	SQL-transakcje	tak	OZ	NO	O	O
SAVEPOINT	SQL-transakcje	tak	OZ	NO	O	NO
SELECT	SQL-dane	tak	OZ	OZ	OZ	OZ
SET CONNECTION	SQL-połączenia	tak	OO	NO	NO	NO
SET ROLE	SQL-sesje	tak	NO	NO	OZ	NO
SET TIME ZONE	SQL-sesje	tak	NO	NO	OZ	NO
SET TRANSACTION	SQL-sesje	tak	OZ	NO	OO	O
START TRANSACTION	SQL-transakcje	tak	NO (obsługuje *BEGIN TRAN*)	NO	NO	NO (obsługuje *BEGIN TRAN*)
TRUNCATE TABLE	SQL-dane	tak	O	NO	OZ	O
UPDATE	SQL-dane	tak	OZ	OZ	OZ	O

ALTER PROCEDURE

Instrukcja *ALTER PROCEDURE* pozwala na wprowadzenie zmian w istniejących składowanych procedurach. Zakres i stopień możliwych zmian jest bardzo zróżnicowany w zależności od dostawcy.

W SQL Server składowana procedura (utworzona wcześniej przy użyciu *CREATE PROCEDURE*) zostaje zmieniona. Instrukcja ta nie wpływa na uprawnienia, procedury zależne czy wyzwalacze.

W Oracle polecenie to po prostu rekompiluje składowaną procedurę PL/SQL, ale nie pozwala na zmianę jej kodu. Do przeprowadzenia tego typu zmian Oracle używa polecenia *CREATE OR REPLACE PROCEDURE*.

Dostawca	Polecenie
SQL Server	obsługiwane ze zmianami
MySQL	nie obsługiwane
Oracle	obsługiwane ze zmianami
PostgreSQL	nie obsługiwane

Opis i składnia SQL99

```
ALTER PROCEDURE nazwa_procedury {CASCADE | RESTRICT}
[LANGUAGE | PARAMETER STYLE | <dostęp do danych SQL> | <klauzula obsługi null>
| DYNAMIC RESULT SETS | NAME]
[typ parametru [,..n]
```

Jak opisano przy *CREATE PROCEDURE*, można zmienić: *LANGUAGE*, *PARAMETER STYLE*, dostęp do danych SQL (np. *NO SQL*, *CONTAINS SQL* itp.), klauzulę obsługi wartości null (np. *CALL ON NULL INPUT*), *DYNAMIC RESULT SET*, czy nazwę procedury.

Polecenia *ALTER PROCEDURE* można użyć także do zmiany liczby i typów parametrów.

Składnia i zmiany w SQL Server

```
ALTER PROC[EDURE] nazwa procedury [; numer]
[{@typ parametru } [VARYING] [=wartość domyślna] [OUTPUT][,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ]
[FOR REPLICATION]
AS
blok_T-SQL
```

W SQL Server polecenie to pozwala na zmianę dowolnych istniejących parametrów utworzonej wcześniej składowanej procedury. W rezultacie jest ono krótszym sposobem wykonania operacji *DROP PROCEDURE* i zmienionej instrukcji *CREATE PROCEDURE*. Przyznane pozwolenia czy uprawnienia dotyczące tej procedury nie muszą być ponownie tworzone. Polecenie to może być używane przez właściciela procedury lub przez osobę, której przyznano jedną z ustalonych ról: **db_owner** lub db_**ddladmin**.

Składnia i zmiany w Oracle

```
ALTER PROCEDURE [użytkownik.]nazwa_procedury COMPILE [DEBUG];
```

Nazwa pakietu lub procedury, która wymaga ponownego skompilowania, musi być podana. Słowo kluczowe *COMPILE* jest wymagane. Opcja *COMPILE [DEBUG]* odtwarza informacje PL/SQL. Polecenie może być wydane tylko przez właściciela procedury lub osobę, która ma specjalne uprawnienia do zmian procedur (*ALTER ANY PROCEDURE*).

Przykład

W poniższym przykładzie użyto składni Microsoft SQL Server do stworzenia procedury o nazwie **wez_nastepna_liczbe**, która tworzy unikatowy łańcuch wyjściowy *CHAR(22)*. Następnie procedurę tę zmieniono przy użyciu *ALTER TABLE*, aby jej wynikiem była unikatowa liczba całkowita *INT*.

```
--składowana procedura w Microsoft SQL Server
CREATE PROCEDURE wez_nastepna_liczbe
  @nastepna_liczba CHAR(22) OUTPUT
AS
BEGIN
  DECLARE @liczba_losowa INT
  SELECT @liczba_losowa = RAND() * 1000000

SELECT @nastepna_liczba =
  RIGHT('000000' + CAST(RAND(@liczba_losowa)*1000000 AS CHAR(6)), 6) +
  RIGHT('0000' + CAST(DATEPART(yy, GETDATE() )AS CHAR(4)), 2) +
  RIGHT('000' + CAST(DATEPART(dy, GETDATE() ) AS CHAR(3)), 3) +
  RIGHT('00' + CAST(DATEPART(hh, GETDATE() ) AS CHAR(2)), 2) +
  RIGHT('00' + CAST(DATEPART(mi, GETDATE() ) AS CHAR(2)), 2) +
  RIGHT('00' + CAST(DATEPART(ss, GETDATE() ) AS CHAR(2)), 2) +
  RIGHT('000' + CAST(DATEPART(ms, GETDATE() ) AS CHAR(3)), 3)
END
GO
ALTER PROCEDURE wez_nastepna_liczbe
  @nastepna_liczba INT OUTPUT
AS
BEGIN
  DECLARE @do_konwersji CHAR(22)
  DECLARE @liczba_losowa INT
  SELECT @liczba_losowa = RAND() * 1000000

SELECT @do_konwersji =
  RIGHT('000000' + CAST(RAND(@liczba_losowa)*1000000 AS CHAR(6)), 6) +
  RIGHT('0000' + CAST(DATEPART(yy, GETDATE() )AS CHAR(4)), 2) +
  RIGHT('000' + CAST(DATEPART(dy, GETDATE() ) AS CHAR(3)), 3) +
  RIGHT('00' + CAST(DATEPART(hh, GETDATE() ) AS CHAR(2)), 2) +
  RIGHT('00' + CAST(DATEPART(mi, GETDATE() ) AS CHAR(2)), 2) +
  RIGHT('00' + CAST(DATEPART(ss, GETDATE() ) AS CHAR(2)), 2) +
  RIGHT('000' + CAST(DATEPART(ms, GETDATE() ) AS CHAR(3)), 3)

  SELECT @nastepna_liczba = CAST(@do_konwersji AS INT)

END
GO
```

ALTER TABLE

Instrukcja *ALTER TABLE* pozwala na modyfikację istniejącej tabeli bez konieczności usuwania tabeli, czy zmiany dotyczących jej uprawnień. Dzięki temu można łatwo przeprowadzić pewne zmiany w istniejącej tabeli.

Zarówno Oracle, jak i Microsoft SQL Server obsługują to polecenie z wieloma zmianami, które pozwalają na obsługę stosowanych w tych systemach różnych fizycznych metod alokacji plików.

Dostawca	Polecenie
SQL Server	obsługiwane ze zmianami
MySQL	obsługiwane z ograniczeniami
Oracle	obsługiwane ze zmianami
PostgreSQL	obsługiwane ze zmianami

Opis i składnia SQL99

```
ALTER TABLE nazwa_tabeli
[ADD [COLUMN] nazwa_kolumny typ_danych atrybuty]
| [ALTER [COLUMN] nazwa_kolumny SET DEFAULT domyślna_wartość]
| [ALTER [COLUMN] nazwa_kolumny DROP DEFAULT]
| [ALTER [COLUMN] nazwa_kolumny ADD SCOPE nazwa_tabeli
| [ALTER [COLUMN] nazwa_kolumny DROP SCOPE {RESTRICT | CASCADE}]
| [DROP [COLUMN] nazwa_kolumny {RESTRICT | CASCADE}]
| [ADD nazwa_więzów_tabeli]
| [DROP CONSTRAINT nazwa_więzów_tabeli {RESTRICT | CASCADE}]
```

Instrukcja *ALTER TABLE* w SQL99 umożliwia przeprowadzenie wielu użytecznych modyfikacji istniejącej tabeli. To wszechstronne polecenie pozwala na dodanie kolumny (*ADD COLUMN*) lub więzów, dodanie lub usunięcie wartości domyślnej (*DEFAULT*), dodanie (lub usunięcie) zakresu (*SCOPE*) do kolumn, posiadających typy zdefiniowane przez użytkownika, oraz usunięcie kolumny lub więzów tabeli. Konstrukcja *DROP RESTRICT* pozwala systemowi DBMS na anulowanie operacji w razie stwierdzenia zależności pomiędzy usuwanym obiektem a innymi obiektami bazy danych. Dodatkowe wyjaśnienia elementów składowych tego polecenia można znaleźć w opisie instrukcji *CREATE TABLE*.

Składnia i zmiany w Microsoft SQL Server

```
ALTER TABLE nazwa_tabeli
[ALTER COLUMN nazwa_kolumny nowy_typ_danych atrybuty { ADD | DROP }
ROWGUIDCOL]
| [ADD [COLUMN] nazwa_kolumny typ_danych atrybuty] [,...n]
| [WITH CHECK | WITH NOCHECK] ADD więzy_tabeli] [,...n]
| [DROP { [CONSTRAINT ] nazwa_więzów | COLUMN nazwa_kolumny }] [,...n]
| [{ CHECK | NOCHECK } CONSTRAINT { ALL | nazwa_więzów [,...n] }]
| [{ ENABLE | DISABLE} TRIGGER { ALL | nazwa_wyzwalacza [,...n] }]
```

Microsoft SQL Server oferuje w swojej implementacji polecenia *ALTER TABLE* wiele funkcji. *ALTER COLUMN* pozwala na zmiany dla istniejącej kolumny cech takich jak typ danych, możliwość umieszczania wartości null, funkcji [tożsamości] itp. *ADD* umieszcza w tabeli nową kolumnę, kolumnę wyliczaną lub więzy jako ostatnią kolumnę tabeli. (W tym momencie nie ma jeszcze metody wstawiania nowej kolumny na innej pozycji.) Opcjonalne słowo *COLUMN* wprowadzono

nie z konieczności, lecz dla większej czytelności. Nowa kolumna musi być zdefiniowana w taki sam sposób, jak w poleceniu *CREATE TABLE*, włącznie z więzami, wartościami domyślnymi i zestawieniami.

Klauzule *WITH CHECK* i *WITH NOCHECK* pozwalają ustalić, czy tabela powinna być sprawdzona z uwzględnieniem dodanych więzów czy kluczy. Jeśli więzy są dodawane z klauzulą *WITH NOCHECK*, optymizer zapytań ignoruje je do momentu uaktywnienia ich przez wykonanie instrukcji *ALTER TABLE nazwa_tabeli CHECK CONSTRAINT ALL*. Więzy mogą być usunięte przez polecenie *DROP CONSTRAINT* (słowo *CONSTRAINT* nie jest wymagane) i aktywowane/dezaktywowane przez *CHECK CONSTRAINT* i *NOCHECK CONSTRAINT* odpowiednio.

Podobnie wyzwalacze mogą być aktywowane i dezaktywowane przez klauzule *ENABLE TRIGGER* i *DISABLE TRIGGER*. Wszystkie wyzwalacze tabeli mogą być aktywowane lub dezaktywowane za pomocą klauzuli *ALL*, użytej wraz z nazwą tabeli, jak w zapytaniu *ALTER TABLE pracownicy DISABLE TRIGGER ALL*.

Składnia i zmiany w MySQL

```
ALTER [IGNORE] TABLE nazwa_tabeli
[ADD [COLUMN] nazwa_kolumny typ_danych atrybuty]
  [FIRST | AFTER nazwa_kolumny]][,...n]
| [ADD INDEX [nazwa_indeksu] (nazwa_kolumny_indeksu,...)] [,...n]
| [ADD PRIMARY KEY (nazwa_kolumny_indeksu,...)] [,...n]
| [ADD UNIQUE [nazwa_indeksu] (nazwa_kolumny_indeksu,...)] [,...n]
| [ALTER [COLUMN] nazwa_kolumny {SET DEFAULT literał | DROP DEFAULT}] [,...n]
| [CHANGE [COLUMN] stara_nazwa_kolumny definicja_tworzenia] [,...n]
| [MODIFY [COLUMN] nazwa_kolumny typ_danych atrybuty] [,...n]
| [DROP [COLUMN] nazwa_kolumny] [,...n]
| [DROP PRIMARY KEY] [,...n]
| [DROP INDEX nazwa_indeksu] [,...n]
| [RENAME [AS] nowa_nazwa_tabeli] [,...n]
| [opcje_tabeli]
```

Więcej informacji na temat dostępnych atrybutów kolumn i więzów tabel można znaleźć w opisie instrukcji *CREATE TABLE*.

Podanie opcji *IGNORE* pozwala na usunięcie każdego następnego rekordu, który w kolumnie tworzonego klucza ma wartość już napotkaną. Jeśli ta opcja nie zostanie podana, stwierdzenie powtarzającej się wartości spowoduje anulowanie operacji tworzenia nowego klucza.

Opcja *FIRST* powoduje, że dodawana kolumna jest umieszczana w tabeli jako pierwsza. *AFTER nazwa_kolumny* pozwala na wstawienie nowej kolumny po podanej w **nazwa_kolumny**.

Polecenie *ALTER TABLE* w MySQL jest dość elastyczne, pozwala bowiem użytkownikowi stosować w jednej instrukcji wiele klauzul *ADD*, *ALTER*, *DROP* czy *CHANGE*. Należy jednak zwrócić uwagę na fakt, że klauzule *CHANGE nazwa_kolumny* i *DROP INDEX* są własnymi rozszerzeniami MySQL i nie występują w standardzie SQL99. MySQL obsługuje również polecenie *MODIFY nazwa_kolumny*, które jest rozszerzeniem Oracle. Klauzula *ALTER COLUMN* pozwala na ustalenie lub usunięcie domyślnej wartości kolumny.

Nazwa tabeli może zostać zmieniona przy użyciu *CHANGE*. Poniższy przykład ilustruje zmianę nazwy tabeli i kolumny:

```
ALTER TABLE pracownicy RENAME AS prac;
ALTER TABLE pracownicy CHANGE pesel_pracownika pesel_prac INTEGER;
```

Ponieważ MySQL pozwala na tworzenie indeksów przy użyciu części kolumny (na przykład na pierwszych dziesięciu znakach kolumny), polecenia *CHANGE* czy *MODIFY* nie mogą tworzyć kolumn o długości mniejszej niż ich indeksy. Użycie *DROP COLUMN* powoduje usunięcie kolumny zarówno z tabeli, jak i ze wszystkich indeksów, w skład których wchodzi.

Wydanie polecenia *DROP PRIMARY KEY*, gdy taki klucz nie istnieje, nie powoduje błędu. MySQL usuwa wtedy pierwszy unikatowy indeks tabeli.

MySQL pozwala na zmianę typu danych istniejącej kolumny bez utraty danych. Wartości zawarte w kolumnie muszą być kompatybilne z nowym typem danych. Przykładowo, kolumna zawierająca datę może być przedefiniowana na typ znakowy, ale typ znakowy nie może być zmieniony na liczbę całkowitą. Na przykład:

```
ALTER TABLE moja_tabela MODIFY moja_kolumna LONGTEXT
```

MySQL pozwala na użycie klauzul *FOREIGN KEY*, *CHECK* i *REFERENCES*, ale są one puste. Polecenia zawierające powyższe klauzule mogą być stosowane, ale nie wykonują żadnej akcji. Klauzule te zostały zastosowane głównie dla osiągnięcia kompatybilności.

Składnia i zmiany w Oracle

```
ALTER TABLE [nazwa_właściciela.]nazwa_tabeli
[ADD nazwa_kolumny typ_danych atrybuty]
| [MODIFY {nazwa_kolumny typ_danych
   | więzy_kolumny
   | atrybuty_fizycznego_składowania [LOGGING | NOLOGGING]
   | zagnieżdżone_atrybuty_tabeli}]
| [MODIFY CONSTRAINT {nazwa_więzów {stan_więzów}
   | kaluzula_usuwania_więzów
   | klauzula_usuwania_kolumny
   | [ALLOCATE | DEALLOCATE klauzula_zakresu]
   | [CACHE | NOCACHE]
   | [MONITORING | NOMONITORING] ]
| [DROP {[COLUMN] nazwa_kolumny | nazwa więzów}]
| [ALLOCATE EXTENT szczegóły]
| [DEALLOCATE UNUSED szczegóły]
| [RENAME TO nowa_nazwa_tabeli]
| [OVERFLOW atrybuty_fizycznego_składowania]
| [ADD OVERFLOW atrybuty_fizycznego_składowania]
| [{ADD | DROP | MODIFY | MOVE | TRUNCATE | SPLIT | EXCHANGE | MODIFY}
   PARTITION szczegóły_partycji]
```

Instrukcja *ALTER TABLE* pokazuje, jak wiele potężnych funkcji kontroli fizycznego składowania tabel i manipulacji tabelami (takich jak obsługa zakresów danych, obsługa przekroczenia zakresu, partycjonowanie tabel dla lepszej obsługi przy dużym obciążeniu) zawiera Oracle. Więcej informacji na temat wymienionych wyżej specyficznych opcji, takich jak *więzy_kolumny*, *atrybuty_fizycznego-_składowania* i *zagnieżdżone_atrybuty_tabeli* można znaleźć w opisie polecenia *CREATE TABLE* dla Oracle.

Polecenie to może być używane do dodawania (*ADD*), modyfikowania (*MODIFY*) i usuwania (*DROP*) istniejących kolumn i więzów. Dodawana kolumna powinna być definiowana jako *NULL*, jeśli tabela zawiera jakiekolwiek rekordy. Słowo kluczowe *MODIFY* pozwala na zmianę właściwości uprzednio stworzonej tabeli. *MODIFY CONSTRAINT* pozwala na usunięcie lub zmodyfikowanie więzów tabeli zarówno wtedy, gdy są aktywne opcje *LOGGING*, *CACHE* czy *MONITOR*, jak i przy zakresach podanych w *ALLOCATE* czy *DEALLOCATE*. Obsługuje także słowa kluczowe *ENABLE* i *DISABLE* służące do aktywowania i dezaktywowania więzów tabeli.



Implementacja *ALTER TABLE* w Oracle jest bardzo złożona i wyrafinowana. Pełne omówienie wszystkich klauzul tego polecenia znajduje się w opisie instrukcji *CREATE TABLE*.

Na przykład poniższy kod dodaje do tabeli nową kolumnę i tworzy nowe, unikatowe więzy z tą tabelą:

```
ALTER TABLE tytuly
ADD podtytul VARCHAR(32) NULL
CONSTRAINT unikalny_podtytul UNIQUE;
```

Podczas dodawania nowych więzów klucza obcego system DBMS sprawdza, czy wszystkie istniejące w tabeli dane spełniają warunki więzów. Jeśli nie, wykonanie polecenia *ALTER TABLE* nie powiedzie się.



Wszystkie aplikacje, które używają polecenia SELECT * będą zawierać w wyniku zapytania nowe kolumny, nawet jeśli nie było to planowane. Z drugiej strony, obiekty wstępnie skompilowane, takie jak składowane procedury, mogą nie zwracać żadnej z nowych kolumn.

Oracle pozwala, aby wiele poleceń, m.in. *ADD* i *MODIFY*, działało na wielu kolumnach, co osiąga się przez umieszczenie tych kolumn w nawiasach. Poniższy przykład ilustruje dodawanie do tabeli kilku kolumn przy użyciu jednej instrukcji:

```
ALTER TABLE tytuly
ADD (podtytul VARCHAR(32) NULL,
   rok_nabycia_praw_autorskich INT,
   data_powstania DATE);
```

Składnia i zmiany w PostgreSQL

```
ALTER TABLE tabela [*]
[ADD [COLUMN] nazwa_kolumny typ_danych atrybuty]
| [ALTER [COLUMN] nazwa_kolumny {SET DEFAULT wartość | DROP DEFAULT}]
| [RENAME [COLUMN] nazwa_kolumny TO nowa_nazwa_kolumny]
| [RENAME TO nowa_nazwa_tabeli]
```

Implementacja *ALTER TABLE* w PostgreSQL pozwala na dodawanie nowych kolumn przy użyciu słowa kluczowego *ADD*. Istniejącym kolumnom można nadać nowe wartości domyślne przy użyciu *ALTER COLUMN ... SET DEFAULT* lub usunąć całkowicie wartości istniejące przez *ALTER COLUMN ... DROP DEFAULT*. Dodatkowo klauzula *ALTER* pozwala na nadanie nowych wartości domyślnych, które będą stosowane tylko do wstawianych później rekordów. Klauzula *RENAME* pozwala na zmianę nazw istniejących kolumn i tabel.

ALTER TRIGGER

Instrukcja *ALTER TRIGGER* modyfikuje istniejącą definicję wyzwalacza bez zmieniania uprawnień czy zależności.

Dostawca	Polecenie
SQL Server	obsługiwane ze zmianami
MySQL	nie obsługiwane
Oracle	obsługiwane ze zmianami
PostgreSQL	nie obsługiwane

Opis i składnia SQL99

Nie istnieje jeszcze standard SQL99 dla tego polecenia.

Składnia i zmiany w Microsoft SQL Server

```
ALTER TRIGGER nazwa_wyzwalacza
ON { nazwa_tabeli | nazwa_perspektywy }
[WITH ENCRYPTION]
{FOR | AFTER | INSTEAD OF} {[DELETE] [,] [INSERT] [,] [UPDATE]}
[WITH APPEND]
[NOT FOR REPLICATION]
AS
  blok_T-SQL
| [FOR { [INSERT] [,] [UPDATE] }
[NOT FOR REPLICATION]
AS
  { IF UPDATE(kolumna) [{AND | OR} UPDATE(kolumna)] [...n]
    |
    IF (COLUMNS_UPDATED() {bitwise_operator} updated_bitmask}
    { operator_porównania} column_bitmask [...n] }
    blok_T-SQL ] } ]
```

Specyfikacja Microsoft SQL Server zezwala na użycie *FOR | AFTER | INSTEAD OF { [DELETE] [,] [UPDATE] [,] [INSERT] } | { [INSERT] [,] [UPDATE] }* dla określenia, do której instrukcji modyfikowania danych odnosi się wyzwalacz. Wymagane jest podanie co najmniej jednej, ale dozwolona jest każda kombinacja opcji oddzielonych przecinkami. Opcje *FOR* i *AFTER* działają w ten sam sposób, powodując wywołanie kodu wyzwalacza po wykonaniu operacji manipulacji danymi. W przeciwieństwie do poprzednich, fraza *INSTEAD OF* powoduje całkowite zastąpienie wywołania operacji manipulacji danymi kodu wyzwalacza.

Fraza *WITH APPEND* pozwala dołączyć dodatkowy wyzwalacz określonego typu do tabeli bazowej. Ta opcja jest dozwolona tylko dla wyzwalaczy *FOR*. Fraza *NOT FOR REPLICATION* informuje serwer, aby nie wykonywał kodu wyzwalacza, jeśli akcja jest wywołana przez replikator taki jak *sqlrepl*. Klauzula *IF UPDATE(kolumna)* sprawdza, czy na podanej kolumnie jest wykonywana operacja *INSERT* lub *UPDATE*. Jest użyteczna podczas wykonywania operacji na rekordach z wykorzystaniem kursora. Operatory *{AND | OR}* pozwalają na testowanie w jednej frazie kilku kolumn. Instrukcja *IF (COLUMNS_UPDATED())* sprawdza, czy podane kolumny podlegały zmianie podczas wywołania wyzwalacza typu *INSERT* lub *UPDATE*. Rezultat jest zwracany jako operator bitowy.

Składnia i zmiany w Oracle

```
ALTER TRIGGER [użytkownik.]nazwa_wyzwalacza [ENABLE | DISABLE | COMPILE
[DEBUG] ];
```

Oracle nie pozwala na całkowitą zmianę kodu wyzwalacza przy użyciu tego polecenia (jakkolwiek można to osiągnąć za pomocą instrukcji *CREATE OR REPLACE TRIGGER* w implementacji Oracle). *ALTER TRIGGER* pozwala na aktywowanie, dezaktywowanie lub ponowne skompilowanie wyzwalacza. *Opcja COMPILE [DEBUG]* generuje informacje PL/SQL.



Oracle pozwala na tworzenie wyzwalaczy *wyłącznie* dla tabel (chociaż dla perspektyw dozwolone są wyzwalacze *INSTEAD OF*). Microsoft SQL Server umożliwia tworzenie wyzwalaczy zarówno dla tabel, jak i dla aktualizowanych perspektyw.

ALTER VIEW

Pomimo że nie istnieje standard SQL99 dla tego polecenia, należy odnotować fakt, że instrukcja ta działa zupełnie inaczej u każdego z głównych dostawców. W Oracle jest ono używane do rekompilacji perspektywy; w Microsoft SQL Server służy ono do modyfikowania perspektywy bez zmieniania zależnych składowanych procedur, wyzwalaczy czy uprawnień.

Dostawca	**Polecenie**
SQL Server	obsługiwane ze zmianami
MySQL	nie obsługiwane
Oracle	obsługiwane ze zmianami
PostgreSQL	nie obsługiwane

Opis i składnia SQL99

Nie istnieje jeszcze standard SQL99 dla tego polecenia.

Składnia i zmiany w Microsoft SQL Server

```
ALTER VIEW nazwa_perspektywy [(kolumna [,...n])]
[WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA]
AS
instrukcja_select
[WITH CHECK OPTION]
```

Instrukcja *ALTER VIEW*, analogicznie do *CREATE VIEW*, pozwala zdefiniować aliasy będące nazwami kolumn w perspektywie oraz podać instrukcję *SELECT* stanowiącą główny składnik perspektywy.

Inne klauzule instrukcji *ALTER VIEW* objaśniono w opisie instrukcji *CREATE VIEW*.

Microsoft SQL Server może zachować uprawnienia do kolumn tylko wtedy, jeśli nazwa kolumny nie zostanie zmieniona podczas wykonywania polecenia. Słowo kluczowe *ENCRYPTION* pozwala na zaszyfrowanie kodu perspektywy wewnątrz systemowej tabeli syscomments. Klauzula *CHECK OPTION* wymusza sprawdzanie przy każdej modyfikacji danych w perspektywie warunków zdefiniowanych w *instrukcji_select*. Jeśli perspektywa zawierała któreś z tych opcji, aby pozostały aktywne, muszą być powtórzone w *ALTER VIEW*.

Składnia i zmiany w Oracle

```
ALTER VIEW [użytkownik.]nazwa_perspektywy COMPILE
```

Instrukcja *ALTER VIEW* w Oracle rekompiluje perspektywę. Służy do zatwierdzenia perspektywy po dokonaniu zmian w tabeli bazowej, gdyż niewykonanie tej operacji powoduje nieprawidłowe działanie perspektywy.

Przykład

Poniższy przykład tworzy przy użyciu SQL Server perspektywę *autorzy_z_kalifornii*, która zawiera dane autorów z Kalifornii. Następnie perspektywa zostaje rozszerzona i zmieniona przy użyciu *ALTER VIEW*.

```
CREATE VIEW autorzy_z_kalifornii
AS
SELECT au_nazwisko, au_imie, miasto, stan
FROM autorzy
WHERE stan='CA'
WITH CHECK OPTION
GO
ALTER VIEW autorzy_z_kalifornii
AS
SELECT au_imie, au_nazwisko, adres, miasto, stan, kod
FROM pubs..autorzy
WHERE stan = 'CA'
GO
```

CALL

Instrukcja CALL wywołuje składowaną procedurę.

Dostawca	Polecenie
SQL Server	nie obsługiwane
MySQL	nie obsługiwane
Oracle	obsługiwane
PostgreSQL	obsługiwane

Opis i składnia SQL99

```
CALL nazwa_procedury [(parametr [,...n] )]
```

Instrukcja *CALL* pozwala na łatwe wywołanie składowanej procedury. Wystarczy podać jej nazwę oraz (w nawiasie) wszystkie konieczne parametry. Jeżeli procedura ma tylko parametry *OUT* lub nie ma żadnych, można dołączyć pusty nawias.



Microsoft SQL Server nie obsługuje instrukcji *CALL*. Niemal identyczny efekt osiągnąć można jednak przy użyciu polecenia *EXECUTE*. Pełny opis tego rozszerzenia można znaleźć w dokumentacji dostawcy dla SQL Server.

Składnia i zmiany w Oracle

```
CALL [schemat.][{nazwa_typu | nazwa_pakietu}.]nazwa_procedury@dblink
[(parametr [,...n] )]
[INTO :nazwa_zmiennej [INDICATOR :nazwa_wskaznika] ]
```

Oracle pozwala na wywołanie przy użyciu instrukcji *CALL* samodzielnej procedury, funkcji czy modułu, ale także procedury lub funkcji będącej częścią typu albo pakietu. Jeżeli funkcja czy procedura należy do innej bazy danych, wystarczy podać w argumencie dblink stworzone wcześniej połączenie z bazą danych, do której należy procedura.

Jeżeli wywoływany podprogram jest funkcją, Oracle wymaga klauzuli *INTO* i odwrotnie, klauzula ta może wystąpić tylko podczas wywołania funkcji. Wymagane jest podanie zmiennej, która ma przechowywać wartość zwracaną przez funkcję. Jeśli funkcja jest prekompilowanym podprogramem Pro*C/C++, można podać wskaźnik, który pozwoli na uzyskanie stanu zmiennej programu-gospodarza.

Przykład

W poniższym przykładzie stworzono i następnie niezależnie wywołano prostą składowaną procedurę.

```
CREATE PROCEDURE zmien_zarobki_pracownika
(id_prac NUMBER, nowe_zarobki NUMBER)
IS
BEGIN
  UPDATE pracownicy SET zarobki = nowe_zarobki WHERE id_pracownika = id_prac
END;

CALL zmien_zarobki_pracownika(1517, 95000);
```

CASE

Funkcja *CASE* pozwala na stosowanie instrukcji warunkowej typu *IF-THEN-ELSE* wewnątrz instrukcji *SELECT* lub *UPDATE*. Instrukcja ta, na podstawie zadanych warunków zwraca jedną z kilku możliwych wartości.

Dostawca	**Polecenie**
SQL Server	obsługiwane
MySQL	obsługiwane
Oracle	nie obsługiwane (podobną rolę spełnia funkcja *DECODE*; szczegóły w dokumentacji dostawcy)
PostgreSQL	Obsługiwane

Istnieją dwa tryby stosowania instrukcji *CASE*: prosty i wyszukiwawczy. Tryb prosty bazuje na porównywaniu: dana wejściowa jest kolejno porównywana z wartościami z listy instrukcji *CASE*. Wynikiem zaś jest wartość odpowiadająca temu elementowi listy, przy którym nastąpiła pierwsza równość. Tryb wyszukiwawczy pozwala na analizę kilku wyrażeń logicznych; wynikiem jest wartość powiązana z pierwszym prawdziwym wyrażeniem.

Opis i składnia SQL99

```
--tryb prosty
CASE wartość_wejsciowa
WHEN wartość_when THEN wartość_wynikowa
[...n]
[ELSE domyślna_wartość_wynikowa]
END

--tryb wyszukiwawczy
CASE
WHEN warunek_logiczny THEN wartość_wynikowa
[...n]
[ELSE domyślne_wyrażenie_wynikowe]
END
```

W prostej funkcji *CASE*, w każdej klauzuli *WHEN* wykonywane jest porównanie *wartość_wejściowa = wartość_when*. *wartość_wynikowa* jest zwracana dla pierwszego porównania, które w wyniku da TRUE. Jeśli żadna równość nie jest prawdziwa, zwracana jest *domyślna_wartość_wynikowa*. Jeśli nie podano wartości domyślnej, funkcja *CASE* zwraca wartość NULL.

Bardziej złożona funkcja wyszukiwawcza ma w zasadzie tę samą strukturę, co funkcja prosta, ale każda klauzula *WHEN* ma własną logiczną operację porównania.

W obu trybach stosuje się wiele klauzul *WHEN*, konieczna jest zaś tylko jedna klauzula *ELSE*.

Przykłady

Poniższy przykład ilustruje użycie prostej instrukcji *CASE* do zmiany sposobu wyświetlania kolumny kontrakt:

```
SELECT au_imie,
       au_nazwisko,
       CASE kontrakt
           WHEN 1 THEN 'Tak'
           ELSE 'Nie'
      END 'kontrakt'
FROM autorzy
WHERE stan = 'CA'
```

Drugi przykład pokazuje użycie wyszukiwawczej funkcji *CASE* w instrukcji *SELECT*, która wypisuje jak wiele tytułów sprzedano w ciągu roku w różnych zakresach liczby sprzedanych książek.

```
SELECT CASE
        WHEN roczna_sprzedaz IS NULL THEN 'Nieznana'
        WHEN roczna_sprzedaz <=  200 THEN 'Nie więcej niż 200'
        WHEN roczna_sprzedaz <= 1000 THEN 'Pomiędzy 201 a 1000'
        WHEN roczna_sprzedaz <= 5000 THEN 'Pomiędzy 1001 a 5000'
        WHEN roczna_sprzedaz <=10000 THEN 'Pomiędzy 5001 a 10000'
        ELSE 'Powyżej 10000'
    END 'Roczna sprzedaż',
    COUNT(*) 'Liczba tytułów'
FROM tytuly
GROUP BY CASE
        WHEN roczna_sprzedaz IS NULL THEN 'Nieznana'
        WHEN roczna_sprzedaz <=  200 THEN 'Nie więcej niż 200'
        WHEN roczna_sprzedaz <= 1000 THEN 'Pomiędzy 201 a 1000'
        WHEN roczna_sprzedaz <= 5000 THEN 'Pomiędzy 1001 a 5000'
        WHEN roczna_sprzedaz <=10000 THEN 'Pomiędzy 5001 a 10000'
        ELSE 'Powyżej 10000'
      END
ORDER BY MIN( roczna_sprzedaz )
```

Rezultat będzie wyglądał następująco:

```
Roczna sprzedaż         Liczba tytułów
----------------------- -------------
Nieznana                2
Nie więcej niż 200      1
Pomiędzy 201 a 1000     2
Pomiędzy 1001 a 5000    9
Pomiędzy 5001 a 10000   1
Powyżej 10000           3
```

W poniższym przykładzie zastosowano instrukcję *UPDATE* do obniżenia cen książek. Skomplikowane polecenie obniża ceny książek komputerowych o 25%, innych o 10%, zaś książek, których roczna sprzedaż wyniosła powyżej 10000 egzemplarzy tylko o 5%.

Do zmiany cen zastosowano wyszukiwawczą instrukcję *CASE*:

```
UPDATE tytuly
SET    cena = cena *
       CASE
           WHEN roczna_sprzedaz > 10000 THEN 0.95 -- 5% rabatu
           WHEN rodzaj = 'komputerowa'  THEN 0.75 -- 25% rabatu
           ELSE 0.9                               -- 10% rabatu
       END
WHERE data_wydania IS NOT NULL
```

W ten sposób przeprowadzono w jednym wyrażeniu trzy odrębne zmiany danych.

CAST

Polecenie *CAST* konwertuje jawnie wyrażenie jednego typu na inny typ danych.

Dostawca	**Polecenie**
SQL Server	obsługiwane
MySQL	nie obsługiwane
Oracle	nie obsługiwane
PostgreSQL	obsługiwane

Opis i składnia SQL99

```
CAST(wyrażenie AS typ_danych [(długość)])
```

Funkcja *CAST* konwertuje dowolne wyrażenie, takie jak wartość kolumny lub zmiennej, na inny zdefiniowany typ danych. Dozwolone jest podanie długości dla wszystkich typów danych, które go obsługują.



Należy zwrócić uwagę na fakt, że niektóre rodzaje konwersji, takie jak z *DECIMAL* na *INTEGER*, powodują zaokrąglenie wartości. Niektóre operacje konwersji mogą zaś powodować błąd, jeśli konwertowana wartość nie mieści się w nowym typie danych.

Przykład

Poniższy kod pobiera z bazy wartość rocznej sprzedaży jako *CHAR*, łączy go z łańcuchem znaków i częścią tytułu książki. Konwertuje *roczna_sprzedaz* do *CHAR(5)* i skraca *tytul* dla większej czytelności:

```
SELECT CAST(roczna_sprzedaz AS CHAR(5)) + „ egzemplarzy ” + CAST(tytul AS
VARCHAR(30))
FROM tytuly
WHERE roczna_sprzedaz IS NOT NULL
  AND roczna_sprzedaz > 10000
ORDER BY roczna_sprzedaz DESC
```

A oto wynik:

```
-----------------------------------------------
22246 egzemplarzy The Gourmet Microwave
18772 egzemplarzy You Can Combat Computer Stress
15096 egzemplarzy Fifty Years in Buckingham Pala
```

CLOSE CURSOR

Polecenie *CLOSE CURSOR* zamyka kursor stworzony na serwerze przy użyciu instrukcji *DECLARE CURSOR*. MySQL nie obsługuje kursorów serwera, ale wspiera wiele rozszerzeń programistycznych języka C.

Dostawca	Polecenie
SQL Server	obsługiwane
MySQL	nie obsługiwane
Oracle	obsługiwane
PostgreSQL	obsługiwane

Opis i składnia SQL99

```
CLOSE { nazwa_kursora }
```

nazwa_kursora to nazwa kursora stworzonego przy użyciu polecenia *DECLARE CURSOR*.

Przykład

Poniższy przykład z Microsoft SQL Server otwiera kursor i wybiera wszystkie rekordy.

```
DECLARE kursor_pracownika CURSOR FOR
  SELECT nazwisko, imie
  FROM pubs.dbo.autorzy
  WHERE nazwisko LIKE 'K%'

OPEN kursor_pracownika

FETCH NEXT FROM kursor_pracownika

WHILE @@FETCH_STATUS = 0
BEGIN
```

```
  FETCH NEXT FROM kursor_pracownika
END

CLOSE kursor_pracownika

DEALLOCATE kursor_pracownika
```



Instrukcja *DEALLOCATE* służy w Microsoft SQL Server do zwolnienia zasobów i struktur danych używanych przez kursor. Oracle, PostgreSQL i MySQL nie używają tego polecenia

COMMIT TRANSACTION

Instrukcja *COMMIT TRANSACTION* jawnie kończy transakcję otwartą jawnie za pomocą *BEGIN* lub niejawnie jako część instrukcji *INSERT*, *UPDATE* lub *DELETE*. Polecenie to pozwala na ręczne i ostateczne zakończenie operacji manipulacji danymi.

Dostawca	**Polecenie**
SQL Server	obsługiwane ze zmianami
MySQL	nie obsługiwane
Oracle	obsługiwane
PostgreSQL	obsługiwane

Opis i składnia SQL99

```
COMMIT [WORK]
```

Oprócz finalizowania jednej lub grupy operacji manipulacji danymi, *COMMIT* ma interesujący wpływ na inne aspekty transakcji. Po pierwsze, zamyka wszystkie powiązane z nią otwarte kursory. Po drugie, usuwa dane z wszystkich tymczasowych tabel stworzonych z klauzulą *ON COMMIT DELETE ROWS*. Po trzecie, zwalnia wszystkie blokady stworzone przez transakcję. Po czwarte, sprawdzane są wszystkie odroczone więzy. Jeśli któreś z nich są naruszone, cała transakcja jest cofana.

Należy zwrócić uwagę na fakt, że w SQL99 transakcja jest *otwarta w sposób jawny* po wykonaniu jednego z poniższych poleceń:

ALTER
CLOSE
COMMIT AND CHAIN (nowe w SQL99)
CREATE
DELETE
DROP
FETCH
FREE LOCATOR
GRANT
HOLD LOCATOR
INSERT
OPEN
RETURN

REVOKE
ROLLBACK AND CHAIN (nowe w SQL99)
SELECT
START TRANSACTION (nowe w SQL99)
UPDATE

SQL99 oferuje nowe, opcjonalne słowa kluczowe *AND CHAIN*. Żaden z dostawców nie obsługuje jeszcze tego polecenia. Nowa składnia wygląda następująco:

```
COMMIT [WORK] [AND [NO] CHAIN]
```

Opcja *AND CHAIN* informuje system DBMS, aby traktował niniejszą transakcję jako część poprzedniej. W rezultacie dwie osobne transakcje wykonują swoje zadania osobno, ale mają wspólne środowisko, takie jak poziom izolacji transakcji. Opcja *AND NO CHAIN* po prostu kończy pojedynczą transakcję. Polecenie *COMMIT* jest równoznaczne z *COMMIT WORK AND NO CHAIN*.

Składnia i zmiany w Microsoft SQL Server

```
COMMIT [TRAN[SACTION] [nazwa_transakcji | @zmienna_z_nazwą_transakcji] ]
|
COMMIT [WORK]
GO
```

Microsoft SQL Server pozwala na stałe wykonywanie specyficznych, nazwanych transakcji. Polecenie *COMMIT* musi wystąpić razem z poleceniem *BEGIN TRAN*. Składnia *COMMIT TRANSACTION* pozwala programiście na jawne podanie nazwy transakcji do zakończenia lub na umieszczenie nazwy transakcji w zmiennej. Co ciekawe, SQL Server zamyka (bez względu na podaną nazwę) ostatnio otwartą transakcję. Przy użyciu *COMMIT WORK* podanie nazwy transakcji czy zmiennej zawierającej nazwę nie jest wymagane.

Powyższa składnia wprowadza w błąd w przypadku zagnieżdżonych wyzwalaczy, ponieważ zamyka najbardziej zewnętrzną transakcję. Transakcje w SQL Server są identyfikowane przy użyciu globalnej zmiennej *@@TRANCOUNT*. Wszystkie transakcje są zakończone tylko wtedy, jeśli *@@TRANCOUNT* jest równa 0.

Składnia i zmiany w Oracle

```
COMMIT [WORK];
```

Oracle nie pozwala na nadawanie nazw transakcjom (obsługuje jednak punkty zapisu stanu), zatem polecenie *COMMIT* po prostu zatwierdza wszystkie operacje manipulacji danymi od ostatniego jawnego lub niejawnego wykonania instrukcji *COMMIT*. Oracle dopuszcza słowo kluczowe *WORK*, ale jest ono całkowicie opcjonalne.

Składnia i zmiany w PostgreSQL

```
COMMIT [WORK | TRANSACTION];
```

W PostgreSQL zarówno *WORK*, jak i *TRANSACTION* są opcjonalne. Polecenie działa tak samo bez nich, jak i z którymkolwiek z nich. Po wykonaniu polecenia wszystkie zakończone transakcje są zapisywane na dysk i widoczne dla innych użytkowników.

Przykład

```
INSERT INTO sprzedaz VALUES ('7896','JR3435','Oct 28 1997',25,'Net
60','BU7832');

COMMIT WORK;
```

Operatory konkatenacji

Operatory konkatenacji zdefiniowane w systemie DBMS pozwalają na łączenie w jedną kolumnę danych z kilku kolumn w zbiorze wyników polecenia *SELECT*.

Dostawca	Polecenie
SQL Server	obsługiwane ze zmianami
MySQL	obsługiwane ze zmianami
Oracle	obsługiwane
PostgreSQL	obsługiwane

Przykład i opis

```
SELECT nazwisko || ', ' || imie FROM klienci WHERE nr_klienta = 41;
```

Operator konkatenacji w standardzie ANSI to dwie pionowe kreski (||), jak pokazano w powyższym przykładzie. Jest on obsługiwany przez Oracle i PostgreSQL.

Symbolem konkatenacji w Microsoft SQL Server jest znak plusa (+).

MySQL używa w tym celu funkcji *CONCAT(łańcuch1, liczba1, łańcuch2, liczba2[,...n])*.

CONNECT

Instrukcja *CONNECT* tworzy połączenie z systemem DBMS i konkretną bazą danych wewnątrz tego systemu.

Dostawca	Polecenie
SQL Server	obsługiwane z ograniczeniami
MySQL	nie obsługiwane
Oracle	obsługiwane
PostgreSQL	nie obsługiwane

Opis i składnia SQL99

```
CONNECT [TO] DEFAULT
| {[specyfikacja_serwera] [AS nazwa_połączenia] [USER nazwa_użytkownika ] }
```

Okres pomiędzy wywołaniem *CONNECT* a *DISCONNECT* jest zazwyczaj nazywany *sesją*. Jeżeli instrukcja *CONNECT* zostanie wywołana bez jawnego rozłączenia poprzedniej sesji, sesja ta przechodzi w stan uśpienia, a nowa staje się aktywna. Zazwyczaj użytkownik pracuje z systemem baz danych za pomocą jawnie utworzonej sesji.



SQL*Plus, narzędzie systemu Oracle, używa polecenia *CONNECT* w inny sposób: do tworzenia połączenia pomiędzy użytkownikiem a schematem.

Instrukcja *CONNECT TO DEFAULT* daje odmienne wyniki, ponieważ różni dostawcy implementują ją w różny sposób. Jednak zgodnie ze standardem, polecenie to powinno otworzyć domyślną sesję z serwerem, w której domyślne są upoważnienia użytkownika i bieżąca baza danych.

W odróżnieniu od powyższej, instrukcja *CONNECT TO nazwa_serwera* pozwala na jawne podanie nazwy serwera, z którym tworzymy połączenie. Dodatkowo, przy użyciu *AS* połączeniu można nadać nazwę, zaś za pomocą *USER* podać nazwę użytkownika.

Składnia i zmiany w Oracle

```
CONN[ECT] [[nazwa_użytkownika/hasło] [AS [SYSOPER | SYSDBA] ] ]
```

Klauzula *CONNECT* pozwala na stworzenie połączenia z podaniem użytkownika. Połączenie ze specjalnymi uprawnieniami może być utworzone przez *AS SYSOPER* lub *AS SYSDBA*. Jeśli jakieś połączenie jest już otwarte, polecenie *CONNECT* zatwierdza wszystkie otwarte transakcje, zamyka tę sesję i tworzy nową.



PostgreSQL nie obsługuje otwarcie polecenia *CONNECT*. Jednakże obsługuje instrukcję *SPI_CONNECT* w interfejsie SPI (Server Programming Interface) i *PG_CONNECT* w pakiecie programistycznym PG/tcl.

Przykłady

Do połączenia przy użyciu podanego identyfikatora użytkownika można użyć:

```
CONNECT TO USER pubs_admin
```

Jeśli system DBMS wymaga nazwania połączenia, można użyć składni:

```
CONNECT TO USER pubs_admin AS pubs_administrative_session;
```

Microsoft SQL Server obsługuje *CONNECT TO* tylko w osadzonym SQL (*ESQL* — Embedded SQL):

```
EXEC SQL CONNECT TO new_york.pubs USER pubs_admin
```