SQL TABLICE INFORMATYCZNE

Marcin Szeliga

Helion

STANDARD ANSI ORAZ IMPLEMENTACJE MS SQL SERVER I POSTGRESQL

Znaczniki języka SQL można podzielić na dziewięć grup:

Dyrektywy

Koniec instrukcji

Serwery baz danych ignorują występujące w instrukcjach SQL znaki końca wiersza. Koniec instrukcji oznacza się symbolem ;.

MS SQL Server

Średnik zastępuje dyrektywa GO - powoduje ona wysłanie do serwera bazy danych i wykonanie wszystkich zapisanych od momentu ostatniego wystąpienia dyrektywy instrukcji.

Wskazówki

1. Dyrektywa GO nie może występować w tym samym wierszu co jakakolwiek instrukcja języka SQL. W wierszu z dyrektywą GO można umieszczać jedynie komentarze.
2. Zakres zmiennych użytkownika ograniczony jest do bloku skryptu zapisanego pomiędzy poszczególnymi dyrektywami GO.
3. Następujące instrukcje muszą zostać oddzielone dyrektywą GO od pozostałych: CREATE PROCEDURE, CREATE VIEW, CREATE TRIGGER, CREATE RULE, CREATE DEFAULT oraz (o ile nie użyto dyrektywy EXEC) wywołania procedur.
4. Dyrektywa GO jest poprawnie interpretowana tylko przez programy MS Query Analyzer i OSQL.

Wywołanie procedury

Do wywołania procedury składowanej służy dyrektywa CALL. Umożliwia ona przekazanie do wywoływanej procedury parametrów i odczytanie wyniku jej działania.

MS SQL Server

Dyrektywa CALL została zastąpioną dyrektywą EXEC (EXECUTE). Oprócz wywołania procedury umożliwia ona wykonanie ciągu znaków, będącego instrukcją języka SQL (wykonywanie tworzonych dynamicznie instrukcji SQL).
Składnia:

```
[[EXEC [UTE]]
    {
        [@zwrócony_stan_wykonania =]
        {nazwa_procedury [;liczba]
        | @nazwa_procedury
    }
    [[@parametr =] {wartość | @zmienna
    [OUTPUT] | [DEFAULT]]
        [,...n]
[WITH RECOMPILE]
```

gdzie:
@zwrócony_stan_wykonania jest opcjonalną zmienną lokalną, przechowującą informacje o stanie wykonania wywoływanej procedury lub funkcji,
@nazwa_procedury jest nazwą zmiennej lokalnej, przechowującej nazwę wywoływanego obiektu,
OUTPUT określa, że wywoływana procedura jest funkcją zwracającą jakąś wartość,
DEFAULT przekazuje do procedury domyślne, określone podczas jej tworzenia wartości parametrów wywołania,
WITH RECOMPILE wymusza ponowną kompilację kodu wywoływanej procedury lub funkcji, co wiąże się z obliczeniem nowego planu jej wykonania, bazującego na aktualnych statystykach.

PostgreSQL

Procedury napisane w językach proceduralnych mogą zostać wywołane dyrektywą CALL. Zwracające dane funkcje wywołuje się instrukcją SELECT, przy czym funkcje zwracające wartości skalarne będą wywoływane w klauzuli SELECT, WHERE, ORDER BY, GROUP BY lub HAVING, a funkcje zwracające zbiory - w klauzuli FROM.

Funkcje

Funkcje języka SQL można podzielić na cztery kategorie:

1. Funkcje skalarne, które zwracają pojedynczą wartość obliczoną na podstawie zera lub większej liczby argumentów skalarnych.
2. Funkcje grupujące, które zwracają pojedynczą wartość dla zbioru argumentów wywołania.
3. Zwracające zbiory danych funkcje typu RowSet, do których można się odwoływać jak do tabel, w klauzuli FROM.
4. Funkcje sterujące wykonaniem programu.

Decyzja o zaimplementowaniu poszczególnych funkcji należy do twórcy serwera baz danych. Wyjątkiem jest ujęta w standardzie ANSI funkcja CASE.

Funkcja sterująca wykonaniem programu CASE

Sprawdza wartość argumentu wywołania i zwraca jeden z wielu predefiniowanych wyników.
Składnia:

```
CASE argument
    WHEN warunek THEN wynik
        [...n]
    [ ELSE wynik_n ]
END
```

Przykład (zapytanie zwracające dane o podatku przypisanym do danej kategorii produktów):

```
SELECT ProductID, CategoryID,
   CASE CategoryID
      WHEN 1 THEN 5.10
      WHEN 2 THEN 1
      WHEN 3 THEN 5.50
      WHEN 4 THEN 1.55
      WHEN w THEN 0
      WHEN 6 THEN 9.05
      WHEN 7 THEN 5
      WHEN 8 THEN 9.05
   END AS Podatek
FROM Products
```

Identyfikatory

Określające obiekty i umożliwiające odwoływanie się do przechowywanych w nich danych lub do ich metod, zdarzeń czy właściwości Identyfikatory muszą być zgodne z przyjętą konwencją nazewniczą:

- Identyfikatory mogą składać się z nie więcej niż 128 znaków.
- Pierwszym znakiem identyfikatora musi być litera alfabetu.
- Identyfikator nie może być słowem zastrzeżonym języka SQL.
- Identyfikator może zawierać litery, cyfry oraz symbole: @, #, _. Wynika z tego, że identyfikatory nie mogą zawierać spacji ani pozostałych symboli specjalnych (PostgreSQL zezwala na stosowanie znaku $).
- Identyfikatory niezgodne z konwencją nazewniczą muszą być wyróżniane za pomocą cudzysłowów.

MS SQL Server

Identyfikatory powinny być wyróżniane za pomocą nawiasów kwadratowych.

Komentarze

Komentarze są ignorowanymi przez kompilator ciągami znaków. Tekst komentarza wyróżniany jest za pomocą znaków /* */. Kompilator ignoruje wszystkie znaki (z wyjątkiem dyrektywy GO) znajdujące się pomiędzy tymi znacznikami. Zdefiniowanym w standardzie ANSI znakiem komentarza są dwa myślniki (--). Kompilator ignoruje znajdujące się po prawej stronie myślników znaki.

Operatory

Operatory pełnią rolę spójników języka SQL i choć w większości mogą zostać zastąpione odpowiednią funkcją, to poprawiają czytelność kodu programu. Funkcja każdego operatora zależy od kontekstu jego wystąpienia.

Operatory arytmetyczne
W języku SQL występują następujące operatory arytmetyczne:

- iloczyn (*),
- iloraz (/),
- modulo (%),
- suma (+),
- różnica (-).

Operator ciągów znaków

Jedynym operatorem ciągów znaków jest operator konkatenacji (||). Pozostałe operacje na ciągach znaków przeprowadzane są przez odpowiednie funkcje ciągu znaków.

MS SQL Server

Operatorem konkatenacji jest znak +.

Operatory logiczne

Operatory logiczne wykorzystywane są najczęściej do łączenia warunków umieszczonych w klauzulach WHERE i HAVING. Występują trzy operatory logiczne:

- koniunkcji (AND), zwracający wartość 1 (prawda), jeżeli oba wyrażenia są prawdziwe,
- alternatywy (OR), zwracający wartość 1 (prawda), jeżeli którekolwiek z wyrażeń jest prawdziwe,
- negacji (NOT), zwracający wartość 1 (prawda), jeżeli wyrażenie było nieprawdziwe i 0 w przeciwnym wypadku.

Operatory porównania

Do operatorów porównania języka SQL zaliczamy:

- równy (=),
- mniejszy niż (<),
- większy niż (>),
- mniejszy lub równy (<=),
- większy lub równy (>=),
- różny (<>),
- należy do zbioru (IN),
- należy do przedziału (BETWEEN ... AND),
- jest zgodny ze wzorcem (LIKE).

Operatory poziomu bitowego

Język SQL umożliwia wykonywanie podstawowych operacji na bitach:

- mnożenie logiczne (&),
- alternatywa logiczna (|),
- negacja logiczna (~),
- alternatywa wyłączna (operatorem XOR serwera MS SQL Server jest ^, PostgreSQLa - #).

PostgreSQL

Serwer PostgreSQL umożliwia dodatkowo wykonanie następujących operacji na poziomie bitów:

- przesunięcie w lewo (<<),
- przesunięcie w prawo (>>).

Hierarchia operatorów

Jeżeli w jednym wyrażeniu posłużono się kilkoma operatorami, serwer baz danych obliczy jego wartość w następującej (mogącej wpłynąć na wynik) kolejności:

1. Operatory umieszczone w nawiasach.
2. Mnożenie, dzielenie, modulo.
3. Dodawanie, odejmowanie.
4. Operatory porównania.
5. Operatory poziomu bitowego.
6. Konkatenacja.
7. Negacja.
8. Koniunkcja.
9. Alternatywa.
10. Przypisanie wartości zmiennej (=).

Słowa kluczowe

Słowa kluczowe są ciągami znaków, mających ściśle określone znaczenie i interpretowanych przez serwer baz danych w sposób charakterystyczny dla słowa kluczowego. Niedopuszczalne jest używanie słów kluczowych niezgodnie z określonymi dla nich zasadami syntaktycznymi (na przykład w niewłaściwej dla słowa klauzuli) oraz z określonymi dla nich zasadami semantycznymi (ich znaczeniem).

Wyrażenia

Wyrażenia są połączeniem symboli i operatorów, reprezentującym pojedynczą wartość. Typ reprezentowanych danych zależy od typu elementów wyrażenia.

Zmienne

W przypadku MS SQL Servera opisywane znaczniki zostały dodane do słownika języka T-SQL, w przypadku PostgreSQL wchodzą one w skład dodatkowego, osobno instalowanego języka PL/pgSQL
Zmienne charakteryzują się nazwą, typem i wartością.
Dzięki zmiennym możemy:

- przechowywać wartości zwrócone przez funkcje w celu ich późniejszego wykorzystania (w standardzie SQL funkcjonalnym odpowiednikiem zmiennych są podzapytania),
- wielokrotnie wykorzystać ten sam kod do wykonywania operacji na różnych danych wejściowych.

Instrukcja DECLARE

Wykonanie instrukcji spowoduje utworzenie zmiennej o określonej nazwie i typie. Wartość zadeklarowanej zmiennej ustawiana jest na NULL.
Składnia:

```
DECLARE
    {{@zmienna typ}
        | {@kursor CURSOR}
        | {typ_tabeli}
    } [,...n]
< typ_tabeli > ::=
    TABLE ({< definicja_kolumny >
    | < zawężenie >} [,...]
        )
```

gdzie:
CURSOR określa, że zmienna będzie kursorem.

Instrukcja SET

Nadaje wartość zdefiniowanej zmiennej.
Składnia:

```
SET @zmienna = wyrażenie
```

Przykład:

```
DECLARE @znajdz varchar(30)
SET @znajdz = 'Fra%'
SELECT CompanyName, Address, Phone
FROM Customers
WHERE CompanyName LIKE @znajdz
```

Znaczniki sterujące wykonaniem programu

W przeciwieństwie do standardu ANSI SQL, implementacje języka posiadają kilka wzorowanych na językach proceduralnych znaczników sterujących wykonaniem programu.
W przypadku MS SQL Servera opisywane znaczniki zostały dodane do słownika języka T-SQL, w przypadku PostgreSQL wchodzą one w skład dodatkowego, osobno instalowanego języka PL/pgSQL.

Instrukcje BEGIN ... END

BEGIN ... END wydzielają zbiór instrukcji SQL traktowanych jako funkcjonalna całość (blok instrukcji).
W serwerze PostgreSQL instrukcja BEGIN rozpoczyna transakcję.

Instrukcja GOTO

Powoduje zmianę sposobu wykonania programu i przejście do oznaczonej etykietą instrukcji. Instrukcje znajdujące się za poleceniem GOTO nie zostaną wykonane.

Instrukcja IF ... ELSE

Instrukcja IF ... ELSE pozwala na warunkowe wykonanie instrukcji SQL. Instrukcja zostanie wykonana, jeżeli warunek zostanie spełniony, w przeciwnym wypadku może zostać wykonana instrukcja umieszczona w opcjonalnej klauzuli ELSE.

Instrukcja RETURN

Powoduje natychmiastowe przerwanie wykonywania procedury lub bloku instrukcji SQL.

Instrukcja WAITFOR

Określa czas, po którym określona instrukcja zostanie wykonana.

Instrukcja WHILE

Polecenie WHILE pozwala na wielokrotne (aż do momentu spełnienia określonego warunku logicznego) wykonanie instrukcji SQL. Dodatkowo możemy sterować wykonaniem instrukcji za pomocą słów kluczowych BREAK i CONTINUE, gdzie:

- BREAK powoduje natychmiastowe przerwanie wykonywania instrukcji SQL,
- CONTINUE powoduje restart pętli WHILE. Instrukcje znajdujące się za poleceniem CONTINUE nie zostaną wykonane.

TYPY DANYCH

Standard ANSI SQL-3 definiuje następujące struktury danych:

Typy liczbowe

bigint	reprezentuje liczby całkowite z zakresu od -2^{63} (-9 223 372 036 854 775 808) do $2^{63}-1$ (9 223 372 036 854 775 807).
int	reprezentuje liczby całkowite z zakresu od -2^{31} (-2 147 483 648) do $2^{31}-1$ (2 147 483 647).
smallint	reprezentuje liczby całkowite z zakresu od -2^{15} (-32 768) do $2^{15}-1$ (32 767).
tinyinit	(MS SQL Server) reprezentuje liczby całkowite z zakresu od 0 do 255.
decimal	reprezentuje liczby o określonej skali i precyzji z zakresu od $-10^{38}+1$ do $10^{38}-1$. Typ numeric jest synonimem typu decimal.
float	reprezentuje liczby o zmiennej precyzji z zakresu od 1,79E+308 do 1,79E+308.
real	reprezentuje liczby o zmiennej precyzji z zakresu od -3,40E+38 do 3,40E+38. Typy zmiennoprzecinkowe charakteryzują się tym, że wartości tak zapisanych danych przechowywane są z pewną skończoną dokładnością.
bit	(MS SQL Server) reprezentuje 0 lub 1. Ten typ często stosowany jest jako odpowiednik niezaimplementowanego w tym serwerze typu logicznego.

Typy daty i czasu

datetime	(MS SQL Server) reprezentuje datę i czas z zakresu od 1 stycznia 1753 do 31 grudnia 9999, z dokładnością do jednej milisekundy.
smalldatetime	(MS SQL Server) reprezentuje datę i czas z zakresu od 1 stycznia 1900 do 6 czerwca 2079, z dokładnością do jednej minuty.
date	(PostgreSQL) reprezentuje datę z dokładnością do jednego dnia.
time	(PostgreSQL) reprezentuje czas.
timestamp	(PostgreSQL) reprezentuje datę i czas.

SQL (strukturalny język zapytań)

TYPY DANYCH cd.

Typy znakowe

`char`	reprezentuje ciąg znaków o określonej, nie większej niż 8 000 znaków, długości.
`nchar`	(MS SQL Server) reprezentuje ciąg znaków o określonej, nie większej niż 4 000 znaków, długości zapisanej w postaci Unicode.
`varchar`	reprezentuje ciąg znaków o zmiennej, nie większej niż 8 000 znaków, długości.
`nvarchar`	(MS SQL Server) reprezentuje ciąg znaków o zmiennej, nie większej niż 4 000 znaków, długości zapisanej w postaci Unicode.
`text`	reprezentuje ciąg znaków o określonej, nie większej niż 2^{31} - 1 (2 147 483 647), długości.
`ntext`	(MS SQL Server) reprezentuje ciąg znaków o zmiennej, nie większej niż 2^{30} - 1 (1 073 741 823), długości zapisanej w postaci Unicode.

Typy binarne

Typy binarne są bardzo podobne do typów znakowych, z tą różnicą, że dane zapisane są jako poszczególne bity:

`binary`	(MS SQL Server) i `bit` (PostgreSQL) reprezentują ciąg bitów o określonej, nie większej niż 8 000 bitów, długości.
`varbinary`	(MS SQL Server) i `varbit` (PostgreSQL) reprezentują ciąg bitów o zmiennej, nie większej niż 8 000 bitów, długości.
`image`	(MS SQL Server) i `bytea` (PostgreSQL) reprezentują dane binarne, na przykład grafikę, o wielkości od 0 do 2^{31} - 1 (2 147 483 647).

Identyfikatory

`uniqueidentifier`	(MS SQL Server) reprezentuje globalny identyfikator (GUID), który jest 16-bajtową liczbą heksadecymalną.

Typy logiczne

`bool`	(PostgrSQL) reprezentuje prawdę lub fałsz.

Wartości nieznane lub nieokreślone

Każdy system zarządzania bazami danych obsługuje specjalną wartość `NULL`, która reprezentuje dane brakujące lub o nieokreślonej wartości. Wartość `NULL` jest wartością specjalną, różną od wszystkich innych wartości, w tym od `0` i pustego ciągu znaków

Wskazówki

1. Wszelkie operacje z wartością `NULL` (np. łączenie ciągu znaków lub mnożenie liczb) powinny zwracać `NULL`.
2. Jedynym dopuszczalnym operatorem porównania z tą wartością jest operator `IS [NOT] NULL` - chociaż nie wiemy, jaką wartość reprezentuje `NULL`, to możemy sprawdzić, czy dana wartość jest nieokreślona.

Typy użytkowników

Tworzenie

Podstawą (szablonem) wszystkich typów danych użytkownika jest predefiniowany typ danych. Zakresem stosowania typu danych użytkownika jest określona baza danych.

MS SQL Server

Możemy utworzyć własny typ danych, wywołując procedurę systemową `sp_addtype`.
Składnia:

```
sp_addtype [@typename =] typ,
    [@phystype =] typ_predefiniowany
    [, [@nulltype =] 'NULL']
    [, [@owner =] 'właściciel']
```

PostgreSQL

Do tworzenia nowych typów służy instrukcja `CREATE TYPE`.
Składnia:

```
CREATE TYPE typ AS
( nazwa_atrybutu
  typ_danych [, ...n ] )
```

Usuwanie

MS SQL Server

Wywołanie procedury `sp_droptype` spowoduje usunięcie wybranego typu danych użytkownika.
Składnia:

```
sp_droptype [@typename =] 'typ'
```

PostgreSQL

Składnia:

```
DROP TYPE typ [, ...n]
    [ CASCADE | RESTRICT ]
```

gdzie:

- użycie opcji `CASADE` powoduje automatyczne usunięcie wszystkich zależnych od danego typu obiektów,
- użycie opcji `RESTRICT` powoduje zgłoszenie błędu, jeżeli istnieją jakiekolwiek zależne od danego typu obiekty (opcja domyślna).

POBIERANIE DANYCH

Informacje przechowywane w bazach danych mogą zostać pobrane za pomocą instrukcji języka SQL `SELECT`.
Instrukcja `SELECT` (zapytanie) określa, **jakie dane mają zostać zwrócone w wyniku jej wykonania, a nie w jaki sposób serwer baz danych ma wykonać tę instrukcję** - SQL jest językiem strukturalnym, a nie proceduralnym.

Instrukcja SELECT

Instrukcja `SELECT` służy do pobierania danych z bazy. Instrukcja musi zawierać (z wyjątkiem polecenia `SELECT`, odwołującego się wyłącznie do stałych, zmiennych lub wyrażeń arytmetycznych) co najmniej jedną klauzulę: za pomocą polecenia `SELECT` określamy interesujące nas kolumny (dokonujemy operacji **selekcji pionowej, projekcji)**, za pomocą klauzuli `FROM` wskazujemy tabelę, z której pobieramy dane. Z reguły ogranicza się również, za pomocą klauzuli `WHERE`, liczbę zwracanych wierszy do rekordów spełniających określone kryteria (operacja **selekcji poziomej, selekcji)**.
Składnia:

```
SELECT [ALL | DISTINCT]
    [TOP n [PERCENT] [WITH TIES]]
    < lista_wyboru >
< lista_wyboru > ::=
    {   *
        | {nazwa_tabeli
        | nazwa_widoku | alias).*
        |    {nazwa_kolumny | wyrażenie}
            [[AS] alias_kolumny]
        | alias_kolumny = wyrażenie
    }   | [LIMIT { n | ALL )] [OFFSET n]
[,...n]
```

gdzie:
`ALL` (wartość domyślna) oznacza, że zwrócone zostaną wszystkie wybrane wiersze,
`DISTINCT` oznacza, że z wyniku zostaną usunięte duplikaty wybranych wierszy,
`TOP` (MS SQL Server) oznacza, że zwrócona zostanie tylko określona liczba wierszy,
`PERCENT` pozwala na względne określenie liczby zwracanych wierszy. Jeżeli użyto znacznika `PERCENT`, wartość n musi należeć do zakresu <0, 100>,
`WITH TIES` oznacza, że zostaną zwrócone wszystkie wiersze, których wartość kolumny wymienionej w klauzuli `ORDER BY` będzie równa najmniejszej wartości wiersza wyniku. Np. jeżeli wybieramy 5 towarów o najwyższej marży, a w tabeli znajduje się 7 towarów o tej samej, najwyższej marży, zapytanie zwróci 7, a nie 5 wierszy,
`LIMIT` (PostgreSQL) jest odpowiednikiem klauzuli `TOP`,
`OFFSET` (PostgreSQL) określa, ile wierszy zostanie pominiętych, zanim zwrócony zostanie pierwszy wiersz wyniku.
Przykłady:

```
SELECT OrderDate, Freight/2 AS wyrażenie
FROM Orders
SELECT TOP 3 'autor: ' AS literał,
   au_fname + ' '+ au_lname AS konkatencja
FROM authors
```

Wskazówki

1. Wartości są zwracane w kolejności określonej w poleceniu `SELECT`. Jeżeli użyto symbolu wielokrotnego (`*`), dane będą zwrócone w kolejności odpowiadającej kolejności kolumn w tabeli.
2. Nazwy kolumn należy oddzielać przecinkami.
3. Ze względu na wydajność należy unikać stosowania symbolu `*` w zapytaniach.
4. Słowa kluczowe `TOP` i `LIMIT` są używane razem z klauzulą `ORDER BY`, ponieważ w przeciwnym wypadku serwer zwróci pierwszych *n* wierszy z przypadkowo uporządkowanego wyniku.

Klauzula FROM

Określa, z których obiektów będą pobierane dane źródłowe.
Składnia:

```
[FROM {< tabela >}
< tabela > ::=
    nazwa_tabeli [[AS] alias_tabeli]
    [WITH (< optymalizacja > [,...n])]
    | nazwa_widoku [[AS] alias_widoku]
    | funkcja_rowset [[AS] alias_funkcji]
    | OPENXML
    | wynik_podzapytania [AS] alias_tabeli
    [(alias_kolumny [,...n])]
```

gdzie:
`optymalizacja` (MS SQL Server) wymusza użycie przez optymalizator konkretnej metody skanowania tabeli, indeksu czy trybu blokowania,
`OPENXML` (MS SQL Server) spowoduje zapisanie wyniku w formacie XML.
Przykład:

```
SELECT CategoryID, CategoryName
FROM Categories
```

Klauzula WHERE

Określa, które wiersze zostaną zwrócone przez zapytanie.
Składnia:

```
[WHERE < warunek_wyszukiwania > | <
   złączenie_starego_typu >]
< złączenie_starego_typu > ::=
    nazwa_kolumny {*= | =*} nazwa_kolumny
< warunek_wyszukiwania > ::=
    {   [NOT] < predykat > |
    (< warunek_wyszukiwania >)}
        [{AND | OR} [NOT] {< predykat
          > | (< warunek_wyszukiwania >)}]
    }   [,...n]
< predykat > ::=
    {   wyrażenie {= | < > | ! = | >
      | > = | ! > | < | < = | ! <}
      wyrażenie
        | wyrażenie znakowe [NOT]
        LIKE wyrażenie_znakowe
            [ESCAPE 'wzorzec']
        | wyrażenie [NOT] BETWEEN
        wyrażenie AND wyrażenie
        | wyrażenie IS [NOT] NULL
        | wyrażenie [NOT] IN
        (podzapytanie | wyrażenie
         [,...n])
        | wyrażenie {= | < > | ! = |
        > | > = | ! > | < | < = | ! <}
            {ALL | SOME | ANY}
            (podzapytanie)
        | EXISTS (podzapytanie)
    }
```

gdzie:
`złączenie_starego_typu` jest niestandardowym określeniem warunków złączenia tabel.
Symbol `*` = odpowiada złączeniu lewostronnemu,
symbol `=` `*` - prawostronnemu,
`predykat` jest wyrażeniem o wartości `1` (prawda), `0` (fałsz) lub `NULL`,
`ESCAPE` 'wzorzec' (MS SQL Server) określa znak eliminujący z wyniku wiersze, które go zawierają.
Przykład:

```
SELECT lastname, firstname, title
FROM employees
WHERE employeeid = 1
```

Wybieranie danych na podstawie warunków różnego rodzaju

Język SQL nie nakłada żadnych ograniczeń na liczbę argumentów wyszukiwania wymienionych w klauzuli `WHERE`. W zależności od rodzaju kryteriów i typu wyszukiwanych danych, możemy skorzystać z:

1. Operatorów porównania - jeżeli chcemy sprawdzić, czy wartość zapisana w tabeli odpowiada wartości warunku wyszukiwania. Na przykład aby wybrać nazwiska i nazwy miast pracowników ze Stanów Zjednoczonych urodzonych przed 1 sierpnia 1955 roku, napiszemy:

```
SELECT lastname, city
FROM employees
WHERE country = 'USA' AND BirthDate
                    <'8/1/55'
```

2. Operatora `LIKE` - jeżeli chcemy sprawdzić zgodność tekstu z podanym wzorcem. Operator `LIKE` może zostać użyty dla danych tekstowych, binarnych oraz daty i czasu. Operator `LIKE` dopuszcza stosowanie następujących symboli wieloznacznych:
   a. `%` - odpowiada dowolnemu ciągowi znaków o długości równej 0 lub większej,
   b. `_` - odpowiada dowolnemu pojedynczemu znakowi,
   c. `[]` - odpowiada pojedynczemu znakowi z określonego zbioru lub zakresu,
   d. `[^]` - odpowiada pojedynczemu znakowi nienależącemu do określonego zbioru lub zakresu.
   Na przykład aby wyświetlić nazwy firm zawierające w nazwie słowo `Restaurant`, których nazwiska przedstawicieli zaczynają się na literę `M`, napiszemy:

```
SELECT companyname
FROM customers
WHERE companyname LIKE '%Restaurant%'
      AND ContactName LIKE 'M%'
```

3. Operatorów logicznych - jeżeli chcemy połączyć kilka warunków wyszukiwania. Na przykład aby wyświetlić informacje o tych towarach, których nazwy nie zaczynają się na literę `T` lub `R` lub zostały dostarczone przez dostawcę o identyfikatorze równym `21` i jednocześnie kosztowały mniej niż `15 $`, napiszemy:

```
SELECT productname, supplierid, unitprice
FROM products
WHERE (productname NOT LIKE 'T%' AND
       productname NOT LIKE 'R%')
OR (supplierid = 21 AND unitprice < 15.00)
```

4. Operatora `BETWEEN ... AND` - jeżeli chcemy sprawdzić, czy wartość zapisana w tabeli należy (lub nie) do określonego przedziału zamkniętego. Na przykład aby wyświetlić nazwy towarów, których cena nie należy do przedziału <5, 55>, napiszemy:

```
SELECT productname
FROM products
WHERE unitprice NOT BETWEEN 5 AND 55
```

5. Operatora `IN` - jeżeli chcemy sprawdzić, czy wartość zapisana w tabeli należy do określonego zbioru. Na przykład aby wyświetlić nazwy firm ze Stanów lub Włoch, napiszemy:

```
SELECT companyname, country
FROM suppliers
WHERE country IN ('USA', 'Italy')
```

6. Operatora `IS NULL` - jeżeli chcemy sprawdzić, czy wartość zapisana w tabeli jest wartością nieokreśloną. Na przykład aby wyświetlić nazwy i numery telefonów tych firm, dla których nie znamy numeru faksu, napiszemy:

```
SELECT companyname, phone
FROM suppliers
WHERE fax IS NULL
```

Klauzula ORDER BY

Wynik zapytania może zostać sformatowany zgodnie z wymaganiami użytkownika:

- Za pomocą słowa kluczowego `DISTINCT` możemy wyeliminować duplikaty.
- Za pomocą aliasów możemy zmienić nazwy kolumn.
- Za pomocą literałów możemy dodać do wyniku dowolne wyrażenia.
- Za pomocą klauzuli `ORDER BY` możemy posortować wiersze wyniku.

Składnia:

```
[ORDER BY {wyrażenie [ASC | DESC]) [,...n]]
```

gdzie:
`wyrażenie` określa kolumnę, według której będą sortowane dane,
`ASC` (wartość domyślna) określa, że dane będą sortowane w porządku rosnącym (od wartości najmniejszej do największej),
`DESC` określa, że dane będą sortowane w porządku malejącym.
Przykład:

```
SELECT OrderID, CustomerID,
   EmployeeID, OrderDate
FROM Orders
ORDER BY OrderDate DESC
```

Kolejny przykład pokazuje, jak uporządkować wynik zapytania przez posortowanie wierszy, eliminacje duplikatów oraz wykorzystanie aliasów i literałów:

```
SELECT DISTINCT 'Nazwa kraju: ',
   Country AS 'Kraj'
FROM suppliers
ORDER BY country
```

ŁĄCZENIE ZBIORÓW DANYCH

W klauzuli `FROM` możemy odwołać się do dowolnej liczby obiektów - tabel, widoków, funkcji typu *RowSet* czy wyników podzapytań. Z reguły łączy się obiekty na podstawie wartości wspólnego atrybutu, na przykład wartości pary klucz podstawowy - klucz obcy. W takim przypadku musimy użyć jednoznacznego identyfikatora obiektu (kolumny). Ponieważ nazwy kolumn zawierających klucz podstawowy i obcy najczęściej są takie same, musimy poprzedzać nazwy kolumn nazwami tabel. Jeżeli dodatkowo stosujemy konwencję, w myśl której poprzedzamy nazwy obiektów nazwą ich właściciela, zapytanie wybierające dane z kilku obiektów staje się mało czytelne. Rozwiązaniem tego problemu są **aliasy**.
Składnia:

```
 [FROM {< złączona_tabela >} [,...n]]
< złączona_tabela > ::=
     < tabela > < typ_złączenia >
     < tabela >
ON < warunek > | USING (kolumna, ...n)
    | < tabela > CROSS JOIN < tabela >
    | < złączona_tabela >
< typ_złączenia > ::=
 [INNER | {{LEFT | RIGHT | FULL}
   [OUTER]}]
 [< optymalizacja_złączenia >]
JOIN
```

gdzie:
`CROSS JOIN` określa, że wynik będzie obliczony jako iloczyn kartezjański złączonych tabel,

Minimalną liczbę warunków użytych do łączenia tabel można wyliczyć ze wzoru: minimalna liczba warunków = liczba złączonych tabel - 1.

`INNER JOIN` określa, że wynik będzie obliczony jako złączenie naturalnie złączonych tabel. Wynikiem złączenia naturalnego są tylko te wiersze, w których w powiązanych kolumnach zapisane były te same wartości,
`LEFT OUTER JOIN` określa, że wynik będzie zawierał wszystkie wiersze z lewej tabeli i wyłącznie te wiersze z prawej tabeli, w których w powiązanych kolumnach zapisane były te same wartości,
`RIGHT OUTER JOIN` określa, że wynik będzie zawierał wszystkie wiersze z prawej tabeli i wyłącznie te wiersze z lewej tabeli, w których w powiązanych kolumnach zapisane były te same wartości. Złączenie prawostronne jest rzadko stosowane przez osoby przyzwyczajone do czytania od strony lewej do prawej,
`FULL OUTER JOIN` określa, że wynik będzie zawierał wszystkie wiersze z obu złączonych tabel.
`optymalizacja_złączenia` (MS SQL Server) wymusza użycie przez optymalizator konkretnej metody złączenia tabel (złączenie przez połączenie, złączenie na podstawie zgodności funkcji skrótu lub złączenie przez pętlę wyszukania).

SQL (strukturalny język zapytań)

ŁĄCZENIE ZBIORÓW DANYCH cd.

`USING` (PostgreSQL) umożliwia podanie oddzielonych przecinkami nazw kolumn występujących w łączonych tabelach. Dane z tych kolumn będą wykorzystane do sprawdzenia warunku złączenia. Np. `USING (kolumna1, kolumna2)` jest odpowiednikiem `ON (tabela1.kolumna1 = tabela2.kolumna1 AND tabela1.kolumna2 = tabela2.kolumna2)`.
Przykład (pobieramy informacje o nazwiskach autorów i tytułach napisanych przez nich książek. Nazwiska autorów, którzy nie napisali żadnej książki, oraz tytuły książek nieznanych autorów nie zostaną zwrócone):

```
SELECT au_lname, title
FROM authors AS a
INNER JOIN titleauthor ta
   ON a.au_id = ta.au_id
INNER JOIN titles t ON
   ta.title_id = t.title_id
```

Wskazówki

1. Należy łączyć tabele za pomocą kolumn przechowujących parę kluczy podstawowy-obcy.
2. Do złączenia należy wykorzystywać całe klucze podstawowe tabel. Jeżeli dla jakiejś tabeli zdefiniowano złożony (składający się z kilku atrybutów) klucz podstawowy, łącząc taką tabelę, trzeba odwołać się do całego klucza.
3. Obiekty należy łączyć za pomocą kolumn tego samego typu.
4. Nazwy kolumn należy poprzedzać aliasem nazwy obiektu źródłowego, nawet jeżeli ich nazwy są unikatowe - w ten sposób poprawia się czytelność zapytania.
5. Należy ograniczać liczbę łączonych obiektów do niezbędnego minimum. Na przykład jeżeli wymagany jest tylko identyfikator obiektu, z reguły nie trzeba odwoływać się do przechowującej informacje o tym obiekcie tabeli.

Złączenie naturalne

Wynikiem złączenia naturalnego jest zbiór wierszy łączonych tabel, dla których wartości kolumn określonych jako warunek złączenia są takie same. Ponieważ w relacyjnych bazach danych informacje są podzielone pomiędzy tabele zawierające dane o obiektach jednego typu, złączenie naturalne jest najczęściej wykorzystywanym (i domyślnym) złączeniem obiektów. Łącząc w ten sposób obiekty, należy pamiętać o następujących zasadach:

1. Ponieważ złączenie naturalne jest domyślnym typem złączenia, można pominąć słowo kluczowe `INNER`.
2. Należy podać pełną nazwę atrybutu, za pomocą którego łączy się obiekty.
3. Łączenie obiektów za pomocą atrybutów, które przyjmują wartość `NULL`, zakończy się niespodziewanymi wynikami.
4. Serwery baz danych nie sortują wyniku złączenia naturalnego - aby uzyskać posortowane dane, należy użyć klauzuli `ORDER BY`.

Złączenia zewnętrzne

Wynikiem lewo- lub prawostronnego złączenia zewnętrznego jest zbiór wierszy łączonych tabel, dla których wartości kolumn określonych jako warunek złączenia są takie same; zbiór ten uzupełniony jest o pozostałe wiersze z lewej lub prawej łączonej tabeli. Nieistniejące wartości reprezentowane są w wyniku złączenia przez wartość `NULL`.
Przykład:

```
SELECT au_lname, title
FROM authors AS a
LEFT OUTER JOIN titleauthor ta
   ON a.au_id = ta.au_id
LEFT OUTER JOIN titles t
   ON ta.title_id = t.title_id
```

Wskazówki

1. Należy korzystać ze złączeń zewnętrznych do pobierania kompletnych informacji o wszystkich obiektach danego typu, nawet jeżeli nie istnieją powiązane z nimi obiekty innego typu.
2. Wynikiem pełnego złączenia zewnętrznego jest zbiór wierszy łączonych tabel, dla których wartości kolumn określonych jako warunek złączenia są takie same, uzupełniony o pozostałe wiersze z obu łączonych tabel.

Złączenie krzyżowe

Wynikiem złączenia krzyżowego jest iloczyn kartezjański łącznych obiektów. W przeciwieństwie do innych typów złączeń, w tym wypadku łączone obiekty nie muszą mieć wspólnych atrybutów.
Przykład:

```
SELECT au_lname, title
FROM authors AS a CROSS JOIN
   titles AS ta
```

Wskazówka

Złączenia tego typu są rzadko stosowane w znormalizowanych bazach danych i służą raczej do generowania danych testowych niż do pobierania istniejących danych.

Złączenie tabeli z nią samą

Złączenie tabeli z nią samą stosujemy, kiedy chcemy wybrać rekordy z tabeli na podstawie wspólnych wartości atrybutów rekordów tej samej tabeli.
Przykład:

```
SELECT a.employeeid, a.lastname,
   b.employeeid, b.lastname
FROM employees AS a INNER JOIN
   employees AS b
ON a.title = b.title
WHERE a.employeeid > b.employeeid
```

Wskazówki

1. Należy utworzyć różne aliasy dla łączonej tabeli i konsekwentnie odwoływać się do aliasów, a nie do nazwy tabeli w ramach zapytania.
2. Każdy rekord, w którym wartości atrybutu złączenia będą sobie równe, zostanie dodany do wyniku złączenia, co spowoduje powstanie duplikatów rekordów. Rozwiązaniem tego problemu jest użycie asymetrycznego (np. większy niż) warunku logicznego w klauzuli `WHERE`.

Złączenie wyników - operator UNION

Za pomocą operatora teoriomnogościowego `UNION` możemy dodać wyniki poszczególnych zapytań. Łączone wyniki muszą składać się z takiej samej liczby kolumn, a poszczególne kolumny muszą być tego samego typu, poza tym konieczne jest, aby występowały one w tej samej kolejności w obu wynikach.
Tabele są reprezentacjami zbiorów, tak więc na sumę składają się wiersze obu tabel:
Tabela1 $\cup$ Tabela2 = {wiersz: wiersz $\in$ Tabela1 lub wiersz $\in$ Tabela2}.
Przykład:

```
SELECT (firstname + ' ' + lastname)
   AS nazwa, city
FROM employees
UNION
SELECT companyname, city
FROM customers
ORDER BY nazwa DESC
```

Wskazówki

1. Domyślnie duplikaty są usuwane ze zbioru powstałego przez złączenie wyników zapytań. Użycie słowa kluczowego `ALL` po operatorze `UNION` spowoduje, że nie będą one usuwane (dotyczy to wszystkich operatorów teoriomnogościowych).
2. Dzielenie skomplikowanych zapytań i łączenie ich wyników jest jednym ze sposobów poprawy wydajności zapytań.

Część wspólna - operator INTERSEC (PostgreSQL)

W wyniku użycia operatora `INTERSEC` uzyskujemy wiersze wspólne dla wszystkich tabel wchodzących w skład instrukcji `SELECT`: Tabela1 $\cap$ Tabela2 = {wiersz: wiersz $\in$ Tabela1 i wiersz $\in$ Tabela2}.
Przykład:

```
SELECT description
FROM products
WHERE Id_G=1
INTERSEC
SELECT description
FROM products
WHERE Id_G=2
```

Wskazówka

W serwerach baz danych nieobsługujących operatora `INTERSEC` tam sam wynik można uzyskać za pomocą podzapytania.

Różnica - operator MINUS (PostgreSQL)

W wyniku użycia operatora `MINUS` uzyskujemy te wiersze, które zostały zwrócone tylko przez pierwszą instrukcję `SELECT` i nie zostały zwrócone przez którąkolwiek z następnych instrukcji `SELECT`: Tabela1 \ Tabela2 = {wiersz: wiersz $\in$ Tabela1 i wiersz $\notin$ Tabela2}.
Przykład:

```
SELECT description
FROM products
WHERE Id_G=3
MINUS
SELECT description
FROM products
WHERE Id_G=1
```

Wskazówka

W serwerach baz danych nieobsługujących operatora `INTERSEC` tam sam wynik można uzyskać za pomocą podzapytania.

GRUPOWANIE DANYCH

W języku SQL dostępnych jest kilka funkcji grupujących. Wykorzystując funkcje tego typu, uzyskujemy pojedynczy wynik obliczony na podstawie wielu argumentów. Na przykład możemy w tabeli policzyć liczbę wierszy spełniających określone kryteria lub możemy wyliczyć wartość średnią dla wszystkich wartości z wybranej kolumny. Użycie tych funkcji zwykle związane jest z operacją na jednej kolumnie (na której wykonywane są obliczenia), a jako wynik zwracany jest tylko jeden wiersz.

Funkcje grupujące

Wszystkie opisywane funkcje domyślnie nie eliminują powtarzających się wierszy, co odpowiada użyciu operatora `ALL` jako pierwszego argumentu wywołania. Jeżeli chcemy ograniczyć dziedzinę funkcji do unikatowych wartości wierszy, należy użyć operatora `DISTINCT`.

COUNT()

Funkcja zawraca liczbę wystąpień wartości wyrażeń różnych od `NULL`, chyba że jako argumentu użyto znaku `*` - takie wywołanie funkcji spowoduje obliczenie liczby wszystkich wierszy, łącznie z duplikatami i wartościami `NULL`.
Przykład:

```
SELECT COUNT (*) AS [liczba
   wszystkich wierszy w tabeli]
FROM Categories
```

Wskazówka

Wykonanie powyższego zapytania dla bardzo dużych tabel może być czasochłonne. Ponieważ serwery baz danych przechowują (w postaci niezbędnych dla optymalizacji zapytań statystyk) informacje o liczbie wierszy każdej tabeli, lepszym rozwiązaniem jest odwołanie się do specyficznej dla danego serwera tabeli systemowej.

SUM()

Funkcja zwraca sumę wszystkich argumentów wywołania. W przeciwieństwie do funkcji `COUNT`, która działa dla wszystkich typów danych, funkcja `SUM` działa tylko dla argumentów liczbowych lub dających się automatycznie na liczby skonwertować. Funkcja `SUM` nie uwzględnia wartości `NULL`.
Przykład:

```
SELECT SUM (UnitPrice)
FROM Products
```

AVG()

Funkcja zwraca wartość średnią przekazanych argumentów. Wartości `NULL` nie są uwzględniane. Argumentami funkcji `AVG` muszą być dane liczbowe.
Przykład:

```
SELECT AVG (UnitPrice)
FROM Products
```

MIN() i MAX()

Funkcja `MIN` służy do znajdowania wartości najmniejszej w zbiorze wartości, a funkcja `MAX` największej. Obie funkcje, podobnie jak funkcja `COUNT`, mogą być użyte dla różnych typów danych.
Przykład:

```
SELECT MAX (OrderDate)
FROM Orders
```

Klauzula GROUP BY

Określa grupy wierszy, dla których będą obliczane wyniki funkcji grupujących wymienionych w poleceniu `SELECT`. Jeżeli została użyta, na liście argumentów klauzuli `GROUP BY` muszą znaleźć się wszystkie niebędące wynikiem funkcji grupującej wyrażenia wymienione w poleceniu `SELECT`.
Składnia:

```
[GROUP BY [ALL] wyrażenie [,...n]
        [WITH {CUBE | ROLLUP}]
]
```

gdzie:
`ALL` (MS SQL Server) powoduje, że do wyniku zostaną dodane wiersze niespełniające warunku określonego w klauzuli `WHERE`. Wynik funkcji grupowych dla tych wierszy jest wartością `NULL`,
`CUBE` (MS SQL Server) powoduje zwrócenie dodatkowego wiersza, zawierającego sumę wyników obliczonych dla wszystkich możliwych kombinacji poszczególnych grup,
`ROLLUP` (MS SQL Server) powoduje zwrócenie dodatkowego wiersza, zawierającego sumę wyników obliczonych grup wartości.

Wskazówki

1. Serwer baz danych zwróci jeden wiersz dla każdej grupy wartości.
2. Kolumny wymienione w klauzuli `GROUP BY` muszą zostać wymienione w poleceniu `SELECT`.
3. Maksymalna liczba kolumn ograniczona jest typem poszczególnych danych zwróconych przez funkcje grupujące.

Przykłady:

```
SELECT CustomerID, SUM(Freight)
FROM orders
GROUP BY ALL CustomerID

SELECT au_lname, COUNT(title_id)
FROM authors AS a
LEFT OUTER JOIN titleauthor AS ta
   ON a.au_id = ta.au_id
GROUP BY au_lname

SELECT orderid, CustomerID,
   SUM(Freight) AS Suma
FROM orders
GROUP BY  orderid, CustomerID
WITH CUBE
ORDER BY  orderid, CustomerID
```

Klauzula HAVING

Określa warunki wyszukiwania dla grup lub funkcji grupujących. Za pomocą klauzuli `HAVING` określa się warunki, które muszą zostać spełnione przez grupy wymienione w klauzuli `GROUP BY`, tak jak za pomocą klauzuli `WHERE` określa się warunki, które muszą spełnić wiersze wymienione w klauzuli `SELECT`.
Składnia:

```
[HAVING < warunek_wyszukiwania >]
```

Przykład:

```
SELECT au_lname, COUNT(title_id)
FROM authors AS a
LEFT OUTER JOIN titleauthor AS ta
   ON a.au_id = ta.au_id
GROUP BY au_lname
HAVING COUNT(title_id)>2
```

Wskazówki

1. Używanie klauzuli `HAVING` w zapytaniu, w którym nie występuje klauzula `GROUP BY`, choć dopuszczalne, jest niczym nieuzasadnione.
2. Warunek wyszukiwania może dotyczyć dowolnych danych wymienionych w poleceniu `SELECT`.
3. Warunek umieszczony w klauzuli `HAVING` wyrażony jest za pomocą dowolnej funkcji grupującej.
4. Klauzula `HAVING` ma wyższy priorytet niż słowo kluczowe `ALL` i jeżeli wystąpi w zapytaniu zawierającym to słowo, zapytanie zwróci wyłącznie grupy spełniające podany warunek wyszukiwania.

Klauzule COMPUTE i COMPUTE BY (MS SQL Server)

Obie klauzule zwracają dodatkowy wiersz, zawierający obliczone dla wyniku podsumowania. Klauzula `COMPUTE BY` dodatkowo zwraca obliczone sumy częściowe. Obie klauzule mogą być wielokrotnie użyte w tym samym zapytaniu.
Składnia:

```
[COMPUTE
    ({AVG | COUNT | MAX | MIN | STDEV
    | STDEVP | VAR | VARP | SUM)
            (wyrażenie)) [,...n]
    [BY wyrażenie [,...n]]
]
```

Przykład:

```
SELECT CategoryID, ProductName, UnitPrice
FROM Products
ORDER BY CategoryID
COMPUTE SUM(UnitPrice) BY CategoryID
COMPUTE SUM(UnitPrice)
```

Wskazówki

1. Niemożliwe jest sformatowanie danych zwracanych przez obie klauzule.
2. Niemożliwe jest obliczenie wyniku na podstawie danych typu text, ntext lub image.
3. Dane, na podstawie których będzie obliczony wynik, muszą zostać wymienione w poleceniu `SELECT`.
4. Należy używać klauzuli `COMPUTE BY` łącznie z klauzulą `ORDER BY` - w innym wypadku obliczone sumy częściowe będą nieuporządkowane i nieczytelne.
5. Lista i kolejność kolumn wymieniona w klauzuli `COMPUTE BY` musi dokładnie odpowiadać liście kolumn wymienionych w klauzuli `ORDER BY`.

PODZAPYTANIA

Podzapytaniem nazywamy instrukcję `SELECT` umieszczoną w ramach innej instrukcji języka SQL (instrukcji `SELECT`, `INSERT`, `UPDATE` lub `DELETE`). Podzapytania z reguły mogą zostać zastąpione złączeniami lub wyrażeniami. Najczęstszymi powodami tworzenia podzapytań są:

1. Chęć uproszczenia zapytania przez podzielenie go na kilka prostszych zapytań.
2. Wykorzystanie wyniku jednego zapytania jako danych źródłowych dla innego zapytania.

Wskazówki

1. Podzapytanie musi zostać zapisane w nawiasie.
2. Wynik podzapytania nie może zawierać danych typów `BLOB` lub `CLOB`.
3. Serwery pozwalają na zagnieżdżanie zapytań do określonego poziomu - w przypadku skomplikowanych lub zwracających duże liczby danych podzapytań maksymalny poziom zagnieżdżenia może wynosić kilkanaście, a nawet kilka zapytań.

Podzapytania niepowiązane

Podzapytania niepowiązane w pierwszej kolejności wykonują zagnieżdżoną instrukcję `SELECT`, a zwrócone przez nią dane przekazują do zapytania zewnętrznego i wykonują je. W tego typu podzapytaniach zapytanie wewnętrzne wykonywane jest tylko raz.

Podzapytania jako źródła danych

Wynik podzapytania może być zbiorem danych źródłowych dla innego zapytania. W takim przypadku podzapytanie znajduje się w klauzuli `FROM` zapytania nadrzędnego.

SQL (strukturalny język zapytań)

PODZAPYTANIA cd.

Przykład:
```
SELECT wynik.*
FROM (SELECT CategoryID, CategoryName
FROM Categories
WHERE CategoryID<3) AS wynik
```

Podzapytania jako wyrażenia

Wynik podzapytania może zostać potraktowany jako lista wartości wyrażenia, pod warunkiem, że podzapytanie zwróciło wartość skalarną.
```
SELECT ProductName, UnitPrice,
  (SELECT AVG(UnitPrice)
  FROM Products) AS [średnia],
  UnitPrice-(SELECT AVG(UnitPrice)
  FROM Products) AS [różnica]
FROM Products
```

Podzapytania powiązane

W tego typu podzapytaniach zapytanie wewnętrzne wykonywane jest dla każdego wiersza wyniku zapytania zewnętrznego i dzięki temu wyniki obu zapytań mogą być ze sobą porównywane. Podzapytanie powiązane jest przykładem dynamicznego złączenia wyniku zapytania z każdym kolejnym wierszem wyniku zapytania zewnętrznego. Serwery baz danych wykonują podzapytania powiązane według następującego schematu:

1. Zapytanie zewnętrzne zwraca wiersz, który jest z wynikiem zapytania wewnętrznego.
2. Podzapytanie jest wykonywane, a jego wynik zostaje przekazany do zapytania zewnętrznego.
3. Zapytanie zewnętrzne zwraca kolejny wiersz i cały proces się powtarza tyle razy, ile wierszy zwróciło zewnętrzne zapytanie.

Przykład (odczytujemy informacje o wszystkich produktach należących do tej samej kategorii co produkt o podanej nazwie - a więc nie podajemy nazwy kategorii):
```
SELECT CategoryName, ProductName
FROM Categories AS C
INNER JOIN Products AS P
   ON C.CategoryID=P.CategoryID
WHERE  C.CategoryID
   IN (SELECT CategoryID
   FROM Products
   WHERE ProductName LIKE 'Chai')
```

Wskazówki

1. Podzapytania powiązane, choć są funkcjonalnym odpowiednikiem złączenia tabel, z reguły są wykonywane dłużej i wymagają przetworzenia większej ilości danych.
2. Jeżeli jest to możliwe, należy zastąpić podzapytanie powiązane klauzulą HAVING - poprawia to czytelność i wydajność zapytania.

Podzapytania z operatorem EXISTS

Za pomocą operatorów EXISTS i NOT EXISTS możemy ograniczyć wynik zapytania zewnętrznego do wierszy odpowiednio zwróconych (lub nie) przez podzapytanie powiązane. Operator EXIST zwraca wartość logiczną I (prawda), jeżeli określony wiersz istnieje w wyniku podzapytania, operator NOT EXISTS - jeżeli wynik podzapytania nie zawiera określonego wiersza.
Przykład (odczytujemy informacje o tych autorach, którzy mają to samo imię i nazwisko, a więc dane, które prawdopodobnie zostały przez pomyłkę kilkakrotnie wprowadzone do bazy):
```
SELECT au_id, au_fname,au_lname,phone
FROM authors as t1
WHERE EXISTS (SELECT au_id
              FROM authors AS t2
              WHERE t2.au_lname =
                t1.au_lname
              AND  t2.au_fname =
                t1.au_fname
              AND  t1.au_id <>
                t2.au_id)
```

Wskazówka

Podzapytania z operatorem EXISTS lub NOT EXISTS nie zwracają żadnych danych, a jedynie prawdę lub fałsz.

MODYFIKOWANIE DANYCH

Instrukcja INSERT

Dodać pojedynczy wiersz do tabeli możemy za pomocą instrukcji INSERT, zawierającej klauzulę VALUES.
Składnia:
```
INSERT [INTO]
    {tabela WITH (<optymalizacja> [,...n])
          | widok
          | funkcja_typu_RowSet
    }
     {   [(lista_kolumn)]
         {VALUES
             ({DEFAULT | NULL |
             wyrażenie} [,...n])
             | instrukcja_SELECT
             | dyrektywa_EXECUTE
         }
     }
     | DEFAULT VALUES
```
gdzie:
lista_kolumn jest opcjonalnym określeniem nazw kolumn, do których dopisywane będą dane. Niepodanie listy kolumn spowoduje dodawanie wartości do kolumn według porządku ich występowania w definicji tabeli docelowej.
Przykład:
```
INSERT INTO authors (au_id, au_fname,
    au_lname, contract)
VALUES ('999-99-9999', 'Szczepan',
   'Szczęśliwy', 1)
```

Wskazówki

1. Wstawiane dane muszą uwzględniać zawężenia nałożone na tabelę.
2. Dane poszczególnych typów muszą być wyszczególnione w klauzuli VALUES zgodnie z konwencją nazewniczą serwera baz danych.

Wstawianie danych wybranych w zapytaniu

Instrukcja INSERT z klauzulą SELECT pozwala na dodanie do tabeli dowolnej liczby wierszy wybranych w zapytaniu.
Przykład:
```
INSERT Customers (CustomerID,
   ContactName, CompanyName)
SELECT EmployeeID, FirstName + ' ' +
   LastName, Title
FROM Employees
WHERE EmployeeID < 3
```

Wskazówki

1. Wynik zapytania w całości zostanie dodany do tabeli docelowej.
2. Tabela docelowa musi zostać wcześniej stworzona.
3. Próba dodania danych innych typów niż typy kolumn tabeli docelowej zakończy się błędem.
4. Wstawiane dane muszą uwzględniać zawężenia nałożone na tabelę.

Tworzenie tabeli za pomocą instrukcji SELECT INTO

Instrukcja SELECT INTO umożliwia utworzenie tabeli, do której zostaną wstawione dane wybrane w zapytaniu.
Przykład:
```
SELECT ProductID, ProductName,
   UnitPrice, UnitPrice*0.22 AS Vat
INTO Podatki
FROM Products
```

Wskazówki

1. Instrukcja SELECT INTO tworzy tabelę docelową o podanej nazwie, więc próba jej wykonania dla istniejącej tabeli skończy się błędem.
2. Konieczne jest podanie nazwy kolumny lub stworzenie aliasu dla wszystkich wyrażeń wymienionych w poleceniu SELECT.
3. Przechowywanie wyników zapytania w tymczasowej tabeli jest często spotykanym sposobem na podzielenie skomplikowanego zapytania pomiędzy kilka prostszych i szybciej wykonywanych zapytań.

Wstawianie wartości domyślnych

DEFAULT

Jeżeli dla danej kolumny tabeli docelowej utworzono zawężenie DEFAULT w instrukcji INSERT, możemy wstawić wartość określoną jako wartość domyślna dla kolumny.
Przykład:
```
INSERT INTO authors (au_id, au_lname,
   au_fname, phone, address, contract)
VALUES ('172-32-7776', 'Gibson',
   'Fiodor', DEFAULT, DEFAULT,0)
```

Wskazówki

1. Próba wstawienia wartości domyślnej dla kolumny, dla której nie została ona określona, spowoduje wstawienie wartości nieokreślonej.
2. Próba wstawienia wartości domyślnej dla kolumn typów autonumerowania (IDENTITY w przypadku MS SQL Servera i SERIAL w przypadku PostgreSQL) skończy się błędem.
3. Niewymienienie słowa kluczowego DEFAULT spowoduje wstawienie wartości NULL dla kolumn, dla których nie określono wartości domyślnej, lub wstawienie wartości domyślnej dla kolumn, dla których została ona określona.

DEFAULT VALUES

Język SQL umożliwia nam również, za pomocą słów kluczowych DEFAULT VALUES, wstawienie wiersza zawierającego wyłącznie wartości domyślne, bez konieczności wielokrotnego wymieniania słowa kluczowego DEFAULT.
Przykład:
```
INSERT INTO Products
DEFAULT VALUES
```

Wskazówki

1. Do wszystkich kolumn, dla których nie zdefiniowano zawężenia DEFAULT, zostaną wstawione wartości nieokreślone.
2. Próba wykonania instrukcji dla tabeli, która zawiera kolumny niezezwalające na wstawienie wartości NULL i nieposiadające zawężenia DEFAULT, skończy się błędem.

Instrukcja DELETE

Instrukcja DELETE usuwa wybrane wiersze tabeli.
Składnia:
```
DELETE
    [FROM]
        {tabela WITH (<optymalizacja> [,...n])
          | widok
          | funkcja_typu_RowSet
        }
         [FROM {<tabela_źródłowa>} [,...n]]
    [WHERE
        {<warunek_wyszukiwania>}
    ]
    [OPTION (<optymalizacja> [,...n])]
```

Wskazówka

Próba usunięcia wierszy, które są powiązane z wierszami w innych tabelach, może skończyć się niepowodzeniem. W takim wypadku najpierw należy usunąć odpowiednie wiersze z tabel bazowych

MS SQL Server

Serwer umożliwia usuwanie wierszy wybranych w wyniku złączenia tabel oraz zwróconych przez podzapytanie powiązane.
Przykłady:
```
DELETE Products
WHERE CategoryID <6
  (Usuwamy informacje o pięciu
  najdroższych produktach)
DELETE Products
FROM (SELECT TOP 5 *
FROM Products
ORDER BY UnitPrice DESC) AS t1
WHERE Products.ProductID = t1.ProductID
```

Usuwanie danych wybranych na podstawie danych zapisanych w innej tabeli

Warunek określony w klauzuli WHERE określa, które wiersze zostaną usunięte z tabeli. Język Transact-SQL umożliwia, przez wymienienie dodatkowej klauzuli FROM, wybór usuwanych wierszy na podstawie danych zapisanych w innej tabeli.
Przykład:
```
  DELETE FROM [order details]
FROM Products AS p
INNER JOIN [order details] AS od ON
  p.ProductID = od.ProductID
WHERE p.ProductName = 'Chang'
```

Instrukcja TRUNCATE

Wszystkie dane możemy usunąć z wybranej tabeli, wykonując instrukcję TRUNCATE TABLE. Chociaż funkcjonalnym odpowiednikiem wykonania instrukcji TRUNCATE jest wykonanie instrukcji DELETE bez klauzuli WHERE, to pierwsza operacja zostanie wykonana szybciej, a informacje o poszczególnych usuwanych wierszach nie zostaną zapisane w dzienniku transakcyjnym.
Składnia:
```
TRUNCATE TABLE tabela
```

Wskazówki

1. Próba obcięcia tabeli powiązanej z innymi tabelami przez klucze obce skończy się błędem (wykonanie instrukcji DELETE bez klauzuli WHERE spowodowałoby usunięcie danych).
2. Obcięcie tabeli jest najszybszym sposobem usunięcia danych.
3. Obcięcie danych jest operacją niezapisywaną w dzienniku transakcyjnym, co oznacza, że po jej przeprowadzeniu należy wykonać kopię zapasową pliku bazy danych.
4. Wartości danych przechowywanych w kolumnie typu autonumerowania zostaną zresetowane.

Instrukcja UPDATE

Za pomocą instrukcji UPDATE możemy aktualizować istniejące dane.
Składnia:
```
UPDATE
        {
         tabela WITH (<optymalizacja>
         [,...n])
         | widok
         | funkcja_typu_RowSet
        }
        SET
        {kolumna = {wyrażenie |
        DEFAULT | NULL} [,...n]
    {{[FROM {<tabela_źródłowa>} [,...n]]
        [WHERE
            <warunek_wyszukiwania>]}
```
Przykład:
```
UPDATE authors
SET au_lname = UPPER (au_lname)
WHERE State = 'CA'
```

Wskazówki

1. Zaktualizowane zostaną wszystkie dane spełniające warunek podany w klauzuli WHERE. Brak tej klauzuli spowoduje zmianę wartości wszystkich danych zapisanych w tabeli.
2. Próba zmiany typu danych zakończy się błędem, chyba że serwer baz danych będzie potrafił dokonać niejawnej konwersji nowego typu do typu zdefiniowanego dla danej kolumny.
3. Aktualizowane dane muszą uwzględniać zawężenia nałożone na tabelę.
4. Niemożliwe jest jednoczesne aktualizowanie danych zapisanych w kilku tabelach.

Aktualizowanie danych na podstawie danych zapisanych w innej tabeli (MS SQL Server)

Język Transact-SQL pozwala na aktualizowanie danych do wartości wybranych lub obliczonych na podstawie danych zapisanych w innej tabeli.
Przykład:
```
UPDATE titles
SET ytd_sales = (SELECT SUM(qty)
FROM sales
WHERE sales.title_id = titles.title_id
AND sales.ord_date IN (SELECT
   MAX(ord_date) FROM sales))
FROM titles, sales
```

Aktualizowanie danych wybranych na podstawie danych zapisanych w innej tabeli (MS SQL Server)

Tak jak możemy ustawić nową wartość aktualizowanych danych na podstawie danych zapisanych w innej tabeli, dane z innej tabeli mogą posłużyć jako kryterium wyboru aktualizowanych wierszy.
Przykład:
```
UPDATE Products
SET unitprice = unitprice * 1.1
FROM Products AS p
INNER JOIN Categories AS c
ON p.CategoryID = c.CategoryID
WHERE c.CategoryName LIKE 'C%'
```

TRANSAKCJE

Transakcja jest sekwencją operacji traktowanych jak pojedyncza, niepodzielna jednostka, która musi zostać w całości przeprowadzona lub w całości wycofana. Przetwarzanie transakcji przebiega w dwóch fazach. Pierwsza z nich to inicjalizacja - po zainicjalizowaniu transakcji wszystkie kolejne wyrażenia SQL traktowane są jako część transakcji, aż do momentu jej zakończenia. Drugą fazą jest zatwierdzenie lub wycofanie transakcji. Jest ono równoznaczne z jej zakończeniem i sprawia, że wszelkie wprowadzone przez nią modyfikacje są na stałe zapisywane lub anulowane w bazie danych.
Do momentu, kiedy transakcja nie zostanie zatwierdzona, wprowadzone przez nią zmiany nie są widoczne dla innych użytkowników.

Rozpoczynanie transakcji

Aby jawnie rozpocząć nową transakcję, należy wykonać instrukcję:
```
BEGIN [TRAN | TRANSACTION]
      [nazwa_transakcji]
```
gdzie nazwa_transakcji umożliwi w przyszłości jej zatwierdzenie, anulowanie lub odtworzenie z kopii zapasowej.

Wskazówki

1. Większość serwerów baz danych posiada specjalny tryb pracy (ang. *Autocommit mode*), w którym wykonanie każdej instrukcji powoduje niejawne rozpoczęcie i zatwierdzenie transakcji.
2. Rozpoczęcie transakcji związane jest z zablokowaniem określonych zasobów (wierszy, kluczy indeksów, stron, tabel lub całej bazy danych).

Zatwierdzanie transakcji

Aby jawnie zatwierdzić transakcję, należy wykonać instrukcję:
```
COMMIT [WORK | TRAN | TRANSACTION]
[nazwa_transakcji]
```

Wskazówki

1. Zatwierdzając transakcję, zapiszemy i udostępnimy zmienione dane pozostałym użytkownikom.
2. Zatwierdzenie transakcji powoduje zdjęcie wszystkich założonych blokad i punktów zachowania.
3. W niektórych serwerach baz danych możemy zatwierdzić transakcję, wykonując instrukcję CREATE, ALTER lub DROP albo poprawnie kończąc sesję, w ramach której rozpoczęliśmy transakcję.

Wycofywanie transakcji

Aby jawnie wycofać rozpoczętą transakcję, należy wykonać instrukcję:
```
ROLLBACK [WORK | TRAN | TRANSACTION]
[nazwa_transakcji]
```

Wskazówki

1. Wycofując transakcję, anulujemy wszystkie dokonane od jej rozpoczęcia zmiany.
2. Wycofanie transakcji powoduje zdjęcie wszystkie wszystkich założonych blokad i punktów zachowania.
3. Przekroczenie czasu transakcji lub przerwanie sesji powoduje wycofanie transakcji (MS SQL Server). W przypadku serwera PostgreSQL 7.4 zerwanie sesji powoduje powstanie nieaktywnych, ale wciąż blokujących zasoby transakcji.

SQL (strukturalny język zapytań)

UPRAWNIENIA

Każdy serwer baz danych umożliwia identyfikację użytkownika, z reguły przez podanie przez niego identyfikatora i hasła (uwierzytelnienie). Następnie serwer określa zakres operacji dozwolonych danemu użytkownikowi (autoryzacja). Ponieważ modele bezpieczeństwa poszczególnych serwerów znacznie się między sobą różnią, przedstawione zostaną tylko najważniejsze informacje dotyczące tworzenia kont użytkowników.

Dodawanie kont użytkowników

Do utworzenia konta nowego użytkownika służy instrukcja CREATE USER.

```
CREATE USER nazwa_użytkownika
```

MS SQL Server

Procedura sp_grantlogin

Wykonanie procedury powoduje dodanie loginu dla użytkownika lub grupy użytkowników systemu operacyjnego.
Składnia:

```
sp_grantlogin [@loginame =]
    'login'
```

Procedura sp_addlogin

Wykonanie procedury spowoduje utworzenie loginu użytkownika serwera baz danych.
Składnia:

```
sp_addlogin [@loginame =] 'login'
    [, [@passwd =] 'hasło']
```

Procedura sp_grantdbaccess

Wykonanie procedury spowoduje dodanie do bieżącej bazy danych konta użytkownika reprezentowanego przez określony login.
Składnia:

```
sp_grantdbaccess [@loginame =] 'login'
  [,[@name_in_db =]
    'konto_użytkownika'
```

Usuwanie kont użytkowników

Do usunięcia konta użytkownika służy instrukcja DROP USER.

```
DROP USER nazwa_użytkownika
```

MS SQL Server

Procedura sp_revokelogin

Wykonanie procedury usuwa login powiązany z użytkownikiem lub grupą użytkowników systemu operacyjnego. Ponieważ konto użytkownika może jednocześnie należeć do wielu grup, usunięcie loginu nie jest jednoznaczne z odmową dostępu.
Składnia:

```
sp_revokelogin [@loginame =]
    'login'
```

Procedura sp_denylogin

Wykonanie procedury spowoduje odmówienie użytkownikowi lub grupie użytkowników systemu operacyjnego dostępu do serwera.
Składnia:

```
sp_denylogin [@loginame =]
    'login'
```

Procedura sp_revokedbaccess

Wykonanie procedury spowoduje usunięcie konta użytkownika z bieżącej bazy danych.
Składnia:

```
sp_revokedbaccess [@name_in_db =]
    'konto_użytkownika'
```

Nadawanie uprawnień

Wyróżnia się trzy typy uprawnień:

1. Uprawnienia do wykonania instrukcji języka SQL - na ich podstawie użytkownicy mogą tworzyć nowe obiekty (np. wykonywać instrukcję CREATE TABLE).
2. Uprawnienia obiektowe - na ich podstawie użytkownicy mogą odwoływać się i wykonywać określone operacje związane z wybranymi obiektami bazy danych.
3. Uprawnienia predefiniowane - użytkownicy należący do predefiniowanych grup serwera lub bazy danych oraz właściciele obiektów na podstawie predefiniowanych (niejawnych) uprawnień mogą wykonywać określone operacje.

Instrukcja GRANT

Nadać uprawnienia rolom lub użytkownikom możemy za pomocą instrukcji języka GRANT.

Składnia:

```
Uprawnienia do wykonania
instrukcji:
GRANT {ALL | instrukcja [,...n]}
TO użytkownik [,...n]
Uprawnienia obiektowe:
GRANT
    {ALL [PRIVILEGES]
    | uprawnienie [,...n]}
    {
        [(kolumna [,...n])] ON
        {tabela | widok}
        | ON {tabela | widok}
        [(kolumna [,...n])]
        | ON {procedura}
        | ON {funkcja}
    }
TO użytkownik [,...n]
[WITH GRANT OPTION]
[AS {grupa | rola}]
```

gdzie:
ALL oznacza wszystkie możliwe uprawnienia danego typu,
WITH GRANT OPTION może być użyta wyłącznie w przypadku nadawania uprawnień obiektowych i przekazuje użytkownikowi prawo do nadania otrzymanego uprawnienia innym użytkownikom.
Przykład:

```
GRANT SELECT, INSERT
ON Categories
TO Jacek
```

Odbieranie uprawnień

Instrukcja DENY (MS SQL Server)

Wykonanie instrukcji spowoduje jawne odebranie określonych uprawnień użytkownikowi.
Składnia:

```
Uprawnienia do wykonania
instrukcji:
DENY {ALL | instrukcja [,...n]}
TO użytkownik [,...n]
Uprawnienia obiektowe:
DENY
    {ALL [PRIVILEGES]
    | uprawnienie [,...n]}
    {
        [(kolumna [,...n])] ON
        {tabela | widok}
        | ON {tabela | widok}
        [(kolumna [,...n])]
        | ON {procedura}
    }
TO użytkownik [,...n]
[CASCADE]
```

gdzie:
CASCADE powoduje odebranie uprawnień użytkownikowi i wszystkim użytkownikom, którzy otrzymali od niego dane uprawnienie i jest wymaganą klauzulą podczas odbierania uprawnień nadanych z klauzulą WITH GRANT OPTION.
Przykład:

```
DENY ALL
ON Categories
TO Agatka
```

Instrukcja REVOKE

Usunąć wcześniej nadane uprawnienia (w przypadku SQL Servera zarówno nadane instrukcją GRANT, jak i DENY) możemy przez wykonanie instrukcji REVOKE.
Składnia:

```
Uprawnienia do wykonania
instrukcji:
REVOKE {ALL | instrukcja [,...n]}
FROM użytkownik[,...n]
Uprawnienia obiektowe:
REVOKE [GRANT OPTION FOR]
    {ALL [PRIVILEGES]
    | uprawnienie [,...n]}
    {
        [(kolumna [,...n])] ON
        {tabela | widok}
        | ON {tabela | widok}
        [(kolumna [,...n])]
        | {procedura}
    }
{TO | FROM}
    użytkownik [,...n]
[CASCADE]
[AS {grupa | rola}]
```

Przykład:

```
REVOKE SELECT
ON Categories
FROM Jacek
```

Wskazówka

Ponieważ brak nadanych uprawnień uniemożliwia wykonanie operacji, użytkownicy, którym odebrano uprawnienia, będą mogli wykonać daną operację tylko w przypadku, gdy należą do roli, której nadano dane uprawnienie.

BAZY DANYCH

Instrukcja CREATE DATABASE

Tworzenie nowej bazy danych polega na podaniu jej nazwy. Opcjonalnie podczas wykonywania instrukcji CREATE DATABASE możemy ustawić wartość opcji bazodanowych związanych z jej fizyczną strukturą. Standard ANSI nie definiuje sposobów tworzenia, modyfikowania i usuwania baz danych.

MS SQL Server

Składnia:

```
CREATE DATABASE nazwa
[ON [PRIMARY]
        [<plik> [,...n]]
        [, <grupa_plików> [,...n]]
]
[LOG ON (<plik> [,...n])]
[COLLATE porządek]
[FOR LOAD | FOR ATTACH]
<plik> ::=
([NAME = nazwa_logiczna,]
    FILENAME = 'nazwa_fizyczna'
    [, SIZE = wielkość]
    [, MAXSIZE = (maksymalna_wielkość
        | UNLIMITED)]
    [, FILEGROWTH = przyrost]) [,...n]
<grupa_plików> ::=
FILEGROUP nazwa <plik> [,...n]
```

PostgreSQL

Składnia:

```
CREATE DATABASE nazwa
[ [ WITH ] [ OWNER [=] właściciel ]
[ LOCATION [=] lokalizacja ]
[ TEMPLATE [=] szablon ]
[ ENCODING [=] kodowanie ] ]
```

Instrukcja ALTER DATABASE

Strukturę bazy danych możemy zmienić, wykonując instrukcję ALTER DATABASE.

MS SQL Server

Składnia:

```
ALTER DATABASE nazwa
{ADD FILE <plik> [,...n]
  [TO FILEGROUP grupa_plików]
| ADD LOG FILE <plik> [,...n]
| REMOVE FILE nazwa_pliku
  [WITH DELETE]
| ADD FILEGROUP nazwa_grupy_plików
| REMOVE FILEGROUP nazwa_grupy_plików
| MODIFY FILE <plik>
| MODIFY NAME = nowa_nazwa
| MODIFY FILEGROUP nazwa_grupy_plików
  {właściwość | NAME =
  nowa_nazwa_grupy_plików}
| SET <opcja> [,...n] [WITH
  <przerwanie>]
| COLLATE <porządek>
}
```

PostgreSQL

Składnia:

```
ALTER DATABASE nazwa SET parametr
   { TO | = } { wartość | DEFAULT }
ALTER DATABASE nazwa RESET parametr
ALTER DATABASE nazwa RENAME
   TO nowa_nazwa
```

Instrukcja DROP DATABASE

Wykonanie instrukcji spowoduje usunięcie wybranych baz danych.
Składnia:

```
DROP DATABASE nazwa [,...n]
```

Wskazówka

MS SQL Server umożliwia jednoczesne usunięcie wielu baz danych.

TABELE

Instrukcja CREATE TABLE

Wykonanie instrukcji spowoduje dodanie nowej tabeli do bazy danych.
Składnia:

```
CREATE TABLE
    [nazwa_bazy_danych. |
      właściciel.] nazwa_tabeli
    ((<definicja_kolumny>
        | nazwa_kolumny AS
          wynik_wyrażenia
        | <zawężenie_tabeli>) [,...n]
| LIKE tabela_wzorcowa [ ( INCLUDING
  | EXCLUDING ) DEFAULTS ] ) [,...n ]
  )
[ INHERITS (tabela_wzorcowa [,...n] ) ]
[ON {grupa_plików | DEFAULT}]
<definicja_kolumny> ::= {nazwa typ}
    [[DEFAULT wyrażenie]
        | [IDENTITY [(wartość_początkowa,
        przyrost) [NOT FOR REPLICATION]]]
  ]
    [<zawężenie_kolumny>] [,…n]
```

gdzie:
IDENTITY (MS SQL Server) oznacza, że wartości poszczególnych pól kolumny będą niepowtarzalnymi, generowanymi przez serwer identyfikatorami. Dla jednej tabeli można zdefiniować tylko jedną kolumnę typu IDENTITY,
LIKE (PostgreSQL) określa tabelę, której struktura zostanie powielona w tworzonej tabeli,
INHERITS (PostgreSQL) określa tabelę, której tworzona tabela będzie kopią,
ON (MS SQL Server) pozwala na podanie grupy plików, w której zostanie zapisana tabela.
Przykład:

```
CREATE TABLE Categories (
   CategoryID int IDENTITY
          (1, 1) NOT NULL ,
   CategoryName varchar (15) NOT NULL ,
   Description ntext  ,
   Picture image )
```

Kolumny zawierające dane pochodne (MS SQL Server)

Jednym ze sposobów poprawy wydajności jest zapisywanie w tabelach danych wyliczonych na podstawie innych informacji. Ze względu na spójność danych dane tego typu muszą być automatycznie modyfikowane przy każdej zmianie danych bazowych.
Przykład:

```
CREATE TABLE City (
  id_C int IDENTITY,
  city varchar (30),
  citizens int,
  category AS CASE
   WHEN citizens < 1000 THEN 'A'
   WHEN citizens BETWEEN 1000 AND 5000 THEN 'B'
   WHEN citizens > 50000 THEN 'C'
   END )
```

Instrukcja ALTER TABLE

Za pomocą instrukcji ALTER TABLE możemy dodawać i usuwać kolumny z definicji tabeli oraz włączać i wyłączać sprawdzanie zawężeń i uruchamianie wyzwalaczy.
Składnia:

```
ALTER TABLE tabela
({ALTER COLUMN kolumna
    {typ [(precyzja [, skala])]
        [NULL | NOT NULL] )
  ]
    | ADD [COLUMN]
        {[<definicja_kolumny>]
        |  kolumna AS wynik_wyrażenia
      } [,...n]
    | [WITH CHECK | WITH NOCHECK] ADD
        {<zawężenie_tabeli>} [,...n]
    | DROP
        {[CONSTRAINT] zawężenie_kolumny
            | COLUMN kolumna} [,...n]
    | {CHECK | NOCHECK} CONSTRAINT
        {ALL | zawężenie [,...n]}
    | {ENABLE | DISABLE} TRIGGER
        {ALL | wyzwalacz [,...n]}
    | RENAME TO nowa_nazwa
}
```

gdzie:
{CHECK | NOCHECK} CONSTRAINT (MS SQL Server) pozwala na czasowe wyłączenie zawężeń typu CHECK i FOREIGN KEY,
{ENABLE | DISABLE} TRIGGER (MS SQL Server) pozwala na czasowe wyłączenie wyzwalaczy,
RENAME TO (PostgreSQL) pozwala zmienić nazwę tabeli.
Przykład:

```
ALTER TABLE Categories
ADD ModTime smalldatetime
```

Zmiana nazwy obiektu (MS SQL Server)

Zmienić nazwę tabeli oraz kolumny możemy przez wywołanie procedury składowanej sp_rename.
Składnia:

```
sp_rename [ @objname = ] 'nazwa' ,
    [ @newname = ] 'nowa_nazwa'
    [ , [ @objtype = ] 'typ_obiektu' ]
```

Instrukcja DROP TABLE

Nieużywane tabele powinny zostać usunięte. Usunięcie tabeli spowoduje utratę wszystkich zapisanych w niej danych.
Składnia:

```
DROP TABLE nazwa_tabeli [,...n]
  [ CASCADE | RESTRICT ]
```

gdzie:
CASCADE (PostgreSQL) spowoduje automatyczne usunięcie powiązanych obiektów.
RESTRICT (PostgreSQL) spowoduje zgłoszenie błędu, jeżeli istnieją powiązane z tabelą obiekty.

Wskazówka

Usunięcie tabeli powoduje automatyczne usunięcie powiązanych z nią indeksów, zawężeń i wyzwalaczy.

SQL (strukturalny język zapytań)

ZAWĘŻENIA

Podstawowym sposobem zapewniania spójności danych jest utworzenie zawężeń - regul określających, jakie dane mogą być zapisywane w tabelach. Zawężenia można definiować, wykonując instrukcje CREATE TABLE oraz ALTER TABLE dla poszczególnych kolumn lub całej tabeli.
Składnia:

```
<zawężenie_kolumny> ::= [CONSTRAINT nazwa]
    {[NULL | NOT NULL]
        | [{PRIMARY KEY | UNIQUE}
            [CLUSTERED | NONCLUSTERED]
            [WITH FILLFACTOR =
          współczynnik_wypełnienia]
          [ON {grupa_plików | DEFAULT}]]
| CHECK (wyrażenie_logiczne)
    }
        | [[FOREIGN KEY]
            REFERENCES
            powiązana_tabela
            [(powiązana_kolumna)]
            [ON DELETE {CASCADE |
             NO ACTION}]
            [ON UPDATE {CASCADE |
             NO ACTION}]
            [NOT FOR REPLICATION]
        ]
  }
<zawężenie_tabeli> ::= [CONSTRAINT nazwa]
    {[{PRIMARY KEY | UNIQUE}
        [CLUSTERED | NONCLUSTERED]
        ((kolumna [ASC | DESC] [,...n]))
        [WITH FILLFACTOR =
         współczynnik_wypełnienia]
        [ON {grupa_plików | DEFAULT}]
| CHECK (wyrażenie_logiczne)
  ]
    | FOREIGN KEY
        [(kolumna [,...n])]
        REFERENCES powiązana_tabela
        [(powiązana_kolumna [,...n])]
          [ON DELETE {CASCADE |
           NO ACTION | SET NULL}]
          [ON UPDATE {CASCADE |
           NO ACTION | SET NULL}]
          [NOT FOR REPLICATION]
  }
```

gdzie:
WITH FILLFACTOR (MS SQL Server) pozwala na określenie współczynnika wypełnienia stron indeksu,
[NOT FOR REPLICATION] (MS SQL Server) definiuje zawężenie, które nie będzie replikowane do powiązanych serwerów,
[CLUSTERED | NONCLUSTERED] (MS SQL Server) określa, czy automatycznie utworzony indeks będzie indeksem grupującym.

NULL i NOT NULL

Język SQL umożliwia określenie, czy wartości przechowywane w danych kolumnach mogą przyjmować wartość nieokreśloną (wartość NULL). Domyślną wartością parametru jest NULL.
Przykład:

```
CREATE TABLE distributors (
name varchar(40) NOT NULL )
```

DEFAULT

Powoduje automatyczne wprowadzenie określonej wartości do kolumny, jeżeli użytkownik pozostawi ją nieuzupełnioną.
Przykład:

```
CREATE TABLE Region (
rowguid uniqueidentifier ROWGUIDCOL
CONSTRAINT DF__Region__rowguid
    DEFAULT (NewId()))
```

Wskazówki

1. Dla kolumny można utworzyć tylko jedno zawężenie DEFAULT.
2. Wartością zawężenia może być wyrażenie, o ile nie odwołuje się ono do kolumn z innych tabel.
3. Użyte wyrażenie musi być proste - niedozwolone jest używanie podzapytań.

PRIMARY KEY

Zawężenie PRIMARY KEY definiuje główny klucz tabeli. Dla tabeli może być zdefiniowany tylko jeden klucz główny, ale w jego skład może wchodzić wiele kolumn.
Przykład:

```
CREATE TABLE films (
code char(5) CONSTRAINT firstkey
     PRIMARY KEY)
```

Wskazówki

1. Kolumny, dla których utworzono zawężenie, nie mogą zawierać wartości NULL.
2. Przed usunięciem zawężenia należy usunąć wszystkie powiązane z nim zawężenia klucza obcego.

FOREGIN KEY

W wyniku nadania tego zawężenia ograniczymy listę dopuszczalnych dla kolumny wartości do wartości przechowywanych w odpowiadającej jej kolumnie w innej tabeli. Wcześniej należy utworzyć tabelę, do której zawężenie będzie się odwoływało. Dodatkowo w tabeli tej musi być utworzona odpowiednia kolumna (lub kolumny) tego samego typu oraz musi być na nich nałożone jedno z dwóch zawężeń: PRIMARY KEY lub UNIQUE.
Przykład:

```
ALTER TABLE films
ADD CONSTRAINT FK_films_pers FOREIGN
  KEY (id_pers) REFERENCES person
  (id_pers)
ON UPDATE CASCADE
```

Wskazówki

1. Nadanie kolumnie zawężenia FOREGIN KEY nie uniemożliwia przechowywania w niej wartości NULL.
2. Można zażądać automatycznego modyfikowania lub usuwania rekordów w powiązanych tabelach. Służą do tego dyrektywy ON UPDATE i ON DELETE.

UNIQUE

W kolumnach, dla których zdefiniujemy to zawężenie, przechowywane będą wyłącznie niepowtarzalne (unikatowe) wartości. W ramach jednej tabeli można zdefiniować dowolną liczbę warunków UNIQUE.
Przykład:

```
CREATE TABLE Sales (
SalesRegion CHAR(2) UNIQUE)
```

Wskazówka

Zawężenia UNIQUE i PRIMARY KEY implementowane są za pomocą automatycznie tworzonych indeksów.

CHECK

Umożliwia sprawdzenie, czy wartość wstawiana do każdego wiersza spełnia określone przez warunek CHECK kryteria. Kryteria określa się przez podanie zawracającego fałsz lub prawdę wyrażenia.
Przykład:

```
ALTER TABLE Region
ADD CONSTRAINT DF__Region CHECK
   (RegionID < 999)
```

Wskazówki

1. Wyrażenie nie może odwoływać się do kolumn z innych tabel.
2. Niedozwolone jest używanie funkcji niedeterministycznych, np. funkcji zwracającej aktualny czas.
3. Wyrażenie nie może być podzapytaniem.
4. Dla jednej kolumny można zdefiniować dowolną liczbę zawężeń CHECK, a każde z nich może zawierać dowolną liczbę operatorów logicznych.

INDEKSY

Jedyną funkcją indeksów jest poprawa wydajności instrukcji języka SQL. Uzyskuje się to przez zmniejszenie liczby odczytywanych danych oraz przechowywanie uporządkowanych danych. Ponieważ każdy serwer baz danych przechowuje dane w innym formacie, nie istnieje uniwersalna, zdefiniowana w standardzie języka strategia tworzenia indeksów.

Instrukcja CREATE INDEX

Wykonanie instrukcji spowoduje utworzenie indeksu powiązanego z wybraną tabelą.

MS SQL Server

Składnia:

```
CREATE [UNIQUE] [CLUSTERED |
  NONCLUSTERED] INDEX indeks
    ON {tabela (kolumna [ASC | DESC]
      [,...n])
[WITH
        [PAD_INDEX]
        [[,] FILLFACTOR =
          współczynnik_wypełnienia]
        [[,] IGNORE_DUP_KEY]
        [[,] DROP_EXISTING]
        [[,] STATISTICS_NORECOMPUTE]
        [[,] SORT_IN_TEMPDB]
]
[ON grupa_plików]
```

gdzie:
PAD_INDEX oznacza utworzenie indeksu, którego środkowe poziomy (poziomy pomiędzy korzeniem a liściem) zostaną zapisane na stronach niewypełnionych w 100 proc., dzięki czemu dodanie lub modyfikacja danych nie będą oznaczały konieczności przebudowy tego indeksu. Parametry wykorzystywane są w połączeniu z parametrem FILLFACTOR. Domyślnie SQL Server na każdej stronie indeksu zostawi przynajmniej tyle wolnej przestrzeni, ile wystarcza, aby możliwe było zapisanie dwóch dodatkowych wierszy indeksu,
FILLFACTOR określa procent wolnej przestrzeni na stronach przechowujących wartości liści indeksu,
IGNORE_DUP_KEY (opcja dostępna tylko dla indeksów unikatowych) sprawi, że próba wstawienia wartości już istniejącej w jednej z kolumn indeksu unikatowego spowoduje wyświetlenie ostrzeżenia, a naruszająca warunki integralności instrukcja zostanie zignorowana, ale transakcja będzie kontynuowana. Brak tego parametru powoduje przy próbie wstawienia istniejących wartości wyświetlenie komunikatu o błędzie i wycofanie całej transakcji,
DROP_EXISTING zastępuje istniejący indeks nowym o tej samej nazwie,
STATISTICS_NORECOMPUTE wyłącza automatyczne aktualizowanie statystyk dla indeksu,
SORT_IN_TEMPDB powoduje, że wewnętrzne operacje sortowania danych indeksu będą przeprowadzane w bazie tempdb.
Przykład:

```
CREATE CLUSTERED INDEX
  idx_autothor_lf ON authors
  (au_ln, au_fn)
WITH
    PAD_INDEX
   ,FILLFACTOR = 80
```

PostrgeSQL

Składnia:

```
CREATE [ UNIQUE ] INDEX nazwa
  ON tabela [ USING metoda ]
( { kolumna | ( wyrażenie ) } [,...n] )
[ WHERE predykat ]
```

gdzie:
USING wskazuje na typ indeksu i może przyjmować jedną z następujących wartości: btree, hash, rtree, gist.

Instrukcja DROP INDEX

Wykonanie instrukcji spowoduje usunięcie określonego indeksu. Przestrzeń zajmowana przez indeks zostaje zwrócona systemowi.

Wskazówka

Indeksy utworzone automatycznie dla kolumn podczas nadawania im zawężeń PRIMARY KEY oraz UNIQUE nie mogą zostać w ten sposób usunięte - należy je usunąć, wykonując odpowiednią instrukcję ALTER TABLE.

MS SQL Server

Składnia:

```
DROP INDEX 'tabela.indeks' [,...n]
```

PostrgeSQL

Składnia:

```
DROP INDEX nazwa [,...n] [ CASCADE
  | RESTRICT ]
```

WIDOKI

Instrukcja CREATE VIEW

Wykonanie instrukcji spowoduje utworzenie wirtualnej tabeli lub widoku pobierających dane z określonych tabel lub widoków. Tabele bazowe muszą zostać utworzone przed utworzeniem widoku.

MS SQL Server

Składnia:

```
CREATE [<właściciel>] VIEW widok
   [(kolumna [,...n])]
[WITH <atrybut> [,...n]]
AS
instrukcja_SELECT
[WITH CHECK OPTION]
<atrybut> ::=
    {[ENCRYPTION | SCHEMABINDING |
     VIEW_METADATA)
```

gdzie:
instrukcja_SELECT jest poprawną instrukcją języka SQL SELECT, z kilkoma dodatkowymi ograniczeniami:
- Instrukcja nie może zawierać klauzuli COMPUTE lub COMPUTE BY.
- Instrukcja nie może zawierać klauzuli ORDER BY, chyba że w klauzuli SELECT zostało wymienione słowo kluczowe TOP.
- Instrukcja nie może zawierać klauzuli INTO.
- Instrukcja nie może pobierać danych z tabel tymczasowych.

WITH CHECK OPTION uniemożliwi takie zmodyfikowanie danych, które spowodowałoby usunięcie wiersza z widoku,
ENCRYPTION powoduje zapisanie definicji widoku w postaci zaszyfrowanej. Zaszyfrowane widoki nie mogą być replikowane,
SCHEMABINDING powoduje powiązanie definicji widoku ze schematem bazy danych, w rezultacie czego niemożliwe będzie usunięcie lub takie zmodyfikowanie obiektów bazowych (tabel lub widoków), które wpłynęłoby na utworzony widok,
VIEW_METADATA powoduje, że klientom łączącym się przez interfejs OLE DB zwracane będą metainformacje o widoku, zamiast metainformacji o obiektach bazowych.
Przykład:

```
CREATE VIEW w_ Products (
  Price, Vat)
AS
AS SELECT ProductID,
  UnitPrice, UnitPric
FROM Products
WHERE CategoryId IN (2,5,9)
WITH CHECK OPTION
```

PostrgeSQL

Składnia:

```
CREATE [ OR REPLACE ] VIEW widok
  [ ( kolumna [, ...n] ) ]
  AS instrukcja_SELECT
```

Wskazówka

Widoki umożliwiają jedynie odczyt danych.

Instrukcja ALTER VIEW (MS SQL Server)

Zmienić definicję widoku możemy przez wykonanie instrukcji ALTER VIEW. W przypadku PostgreSQL należy wykonać instrukcję CREATE OR REPLACE VIEW.
Składnia:

```
ALTER VIEW widok [(kolumna [,...n])]
[WITH <atrybut> [,...n]]
AS
instrukcja_SELECT
[WITH CHECK OPTION]
<atrybut> ::= {ENCRYPTION |
    SCHEMABINDING | VIEW_METADATA}
```

Przykład:

```
ALTER VIEW w_ Products (Name, Price, Vat)
AS
AS SELECT ProductName, UnitPrice,
   UnitPrice*0.22
FROM Products
```

Wskazówka

Zmiana definicji obiektu tym różni się od jego usunięcia i odtworzenia, że zachowane zostaną nadane użytkownikom uprawnienia.

Instrukcja DROP VIEW

Niepotrzebne widoki mogą zostać usunięte z bazy danych.

MS SQL Server

Składnia:

```
DROP VIEW widok [,...n]
```

Wskazówka

Ponieważ usunięcie tabeli bazowej nie spowoduje automatycznego usunięcia powiązanych z nią widoków, z reguły instrukcje DROP TABLE i DROP VIEW wykonywane są łącznie.

PostgreSQL

Składnia:

```
DROP VIEW widok [,...n] [ CASCADE
  | RESTRICT ]
```


Wydawnictwo Helion
ul. Chopina 6, 44-100 Gliwice
44-100 Gliwice, skr. poczt. 462
(32) 230-98-63
http://helion.pl e-mail: helion@helion.pl
Informatyka w najlepszym wydaniu

Zamów najnowszy katalog: http://helion.pl/katalog
Zamów informacje o nowościach: http://helion.pl/nowosci
Zamów cennik: http://helion.pl/cennik

Aby ocenić tę tablicę, zajrzyj pod adres:
http://helion.pl/user/opinie?tisql

helion.pl księgarnia internetowa

ISBN 83-7361-873-2