

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. Receptury

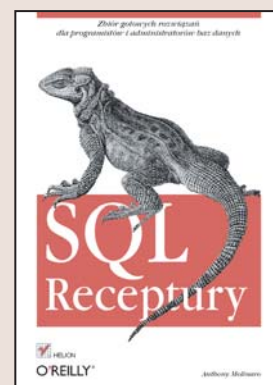
Autor: Anthony Molinaro

Tłumaczenie: Mikołaj Szczepaniak

ISBN: 83-246-0450-2

Tytuł oryginału: [SQL Cookbook](#)

Format: B5, stron: 624



SQL jest językiem programowania używanym we wszystkich najpopularniejszych systemach zarządzania bazami danych. Oczywiście, każdy z nich ma specyficzne dla siebie polecenia i parametry, jednak rdzeń języka jest ustandaryzowany.

SQL wykorzystuje się do tworzenia baz i tabel, wprowadzania danych do bazy i manipulowania nimi oraz do administrowania serwerem bazy danych.

Mimo stosunkowo niewielkiego zbioru słów kluczowych, jest niezwykle elastyczny i uniwersalny.

Książka „SQL. Receptury” to zestawienie rozwiązań problemów, z jakimi programiści baz danych spotykają się w swojej pracy. Przedstawia zagadnienia związane z wybieraniem rekordów z bazy, grupowaniem, sortowaniem ich i tworzeniem złożonych zapytań wykorzystujących kilka tabel. Opisuje metody wprowadzania danych do bazy, tworzenia raportów i przetwarzania wyników zapytań. Każde z rozwiązań zaprezentowane jest w formie polecenia SQL opatrzonego szczegółowym komentarzem.

- Odczytywanie danych z bazy
- Sortowanie wyników zapytań
- Łączenie tabel w zapytaniach
- Wprowadzanie i aktualizowanie danych w tabelach
- Usuwanie rekordów
- Operacje na liczbach i datach
- Zapytania złożone
- Tworzenie raportów
- Polecenia SQL specyficzne dla określonych systemów baz danych

Optymalna konstrukcja zapytań SQL jest jednym z warunków szybkiego działania aplikacji bazodanowych. Dzięki wiadomościom z tej książki każdy programista baz danych wykorzysta wszystkie możliwości języka SQL.



Spis treści

Przedmowa	11
1. Odczytywanie rekordów	29
Odczytywanie wszystkich wierszy i kolumn tabeli	29
Odczytywanie podzbioru wierszy tabeli	30
Odnajdywanie wierszy spełniających wiele warunków	30
Odczytywanie podzbioru kolumn tabeli	31
Definiowanie znaczących nazw kolumn	32
Odwołania do aliasów kolumn w klauzuli WHERE	33
Konkatenacja wartości kolumn	34
Stosowanie logiki warunkowej w wyrażeniu SELECT	35
Ograniczanie liczby zwracanych wierszy	36
Zwracanie n losowych rekordów tabeli	38
Odnajdywanie wartości pustych	39
Tłumaczenie wartości pustych na wartości rzeczywiste	40
Poszukiwanie wzorców	41
2. Sortowanie wyników zapytań	43
Zwracanie wyników zapytań posortowanych w określonym porządku	43
Sortowanie zbioru wynikowego według zawartości wielu pól	44
Sortowanie według podłańcuchów	45
Sortowanie wymieszanych danych alfanumerycznych	46
Obsługa wartości pustych w zapytaniach sortujących	49
Sortowanie według klucza zależnego od danych	55
3. Praca z wieloma tabelami	57
Umieszczanie jednego zbioru wierszy ponad drugim	57
Łączenie wzajemnie powiązanych wierszy	59
Odnajdywanie wspólnych wierszy pomiędzy dwiema tabelami	60
Uzyskiwanie z jednej tabeli tylko tych wartości, które nie występują w innej tabeli	62

Uzyskiwanie z jednej tabeli tylko tych wierszy, dla których nie istnieją odpowiedniki w innej tabeli	66
Dodawanie złączeń do zapytań bez konieczności modyfikowania pozostałych, już istniejących złączeń	68
Określanie, czy dwie tabele zawierają te same dane	70
Identyfikowanie i eliminowanie iloczynów kartezjańskich	76
Stosowanie złączeń w zapytaniach wykorzystujących funkcje agregujące	78
Stosowanie złączeń zewnętrznych w zapytaniach wykorzystujących funkcje agregujące	82
Zwracanie brakujących danych z wielu tabel	85
Wykorzystywanie wartości NULL w operacjach i wyrażeniach warunkowych	89
4. Wstawianie, aktualizowanie i usuwanie	91
Wstawianie nowych rekordów	92
Wstawianie wartości domyślnych	92
Przykrywanie wartości domyślnych wartością NULL	94
Kopiowanie wierszy pomiędzy tabelami	95
Kopiowanie definicji tabel	95
Wstawianie wierszy do wielu tabel jednocześnie	97
Blokowanie możliwości wstawiania wartości do wybranych kolumn	99
Modyfikowanie rekordów tabeli	100
Aktualizowanie danych pod warunkiem zawierania w tabeli określonych wierszy	101
Aktualizowanie wartości według zawartości innej tabeli	102
Scalanie rekordów	105
Usuwanie wszystkich rekordów z tabeli	107
Usuwanie rekordów spełniających określone kryteria	107
Usuwanie pojedynczych rekordów	108
Usuwanie wierszy naruszających integralność odwołań	109
Usuwanie powtarzających się rekordów	110
Usuwanie wierszy będących przedmiotem odwołań rekordów składowanych w innej tabeli	111
5. Zapytania przetwarzające metadane	113
Generowanie listy tabel wchodzących w skład schematu bazy danych	113
Generowanie listy kolumn danej tabeli	114
Generowanie listy indeksowanych kolumn danej tabeli	115
Generowanie listy ograniczeń zdefiniowanych dla tabeli	117
Generowanie listy kluczy obcych pozbawionych indeksów	118
Generowanie kodu języka SQL za pomocą wyrażen tego języka	121
Opisywanie perspektyw słowników danych w bazie danych Oracle	123

6. Praca z łańcuchami	125
Przechodzenie pomiędzy znakami łańcucha	126
Umieszczanie apostrofów w stałych łańcuchowych	128
Zliczanie wystąpień znaku w łańcuchu wejściowym	129
Usuwanie z łańcucha niechcianych znaków	130
Oddzielanie danych numerycznych od danych znakowych	131
Określanie, czy łańcuch jest ciągiem alfanumerycznym	135
Określanie inicjałów na podstawie całych imion i nazwisk	140
Sortowanie kolumn według wybranych fragmentów łańcuchów	144
Sortowanie danych według liczb zapisanych w łańcuchach	145
Tworzenie listy wartości oddzielonych przecinkami z danych zawartych w wierszach tabeli	151
Konwertowanie danych oddzielonych przecinkami na wielowartościową listę IN	157
Sortowanie znaków w łańcuchach w porządku alfabetycznym	162
Identyfikowanie łańcuchów, które można traktować jak liczby	168
Odnajdywanie n-tego podłańcucha na liście oddzielonej przecinkami	174
Przetwarzanie adresów IP	180
7. Praca z liczbami	183
Wyznaczanie wartości średniej	183
Identyfikacja minimalnej i maksymalnej wartości w kolumnie	185
Sumowanie wartości składowanych w kolumnie	187
Zliczanie wierszy tabeli	188
Zliczanie różnych wartości w kolumnie	190
Generowanie sum bieżących	191
Generowanie iloczynów bieżących	194
Wyznaczanie różnic bieżących	197
Wyznaczanie wartości modalnej (dominanty)	198
Wyznaczanie mediany	201
Określanie procentowego udziału w wartości łącznej	205
Agregowanie kolumn zawierających wartości NULL	207
Wyznaczanie wartości średnich z wyłączeniem wartości spoza określonego przedziału	209
Konwertowanie łańcuchów alfanumerycznych na liczby	210
Modyfikowanie wartości uwzględnianych w sumach bieżących	213
8. Działania na datach	215
Dodawanie i odejmowanie dni, miesięcy i lat	215
Określanie liczby dni pomiędzy dwiema datami	218
Określanie liczby dni roboczych pomiędzy dwiema datami	220
Określanie liczby miesięcy lub lat dzielących dwie daty	224
Określanie liczby sekund, minut lub godzin dzielących dwie daty	227

Zliczanie wystąpień poszczególnych dni tygodnia w roku	228
Określanie różnicy dzielącej datę reprezentowaną w bieżącym rekordzie i datę reprezentowaną w rekordzie następnym	239
9. Przetwarzanie dat	245
Określanie, czy dany rok jest rokiem przestępnym	246
Określanie liczby dni w roku	252
Wydobywanie jednostek czasu z dat wejściowych	255
Określanie pierwszego i ostatniego dnia miesiąca	257
Określanie wszystkich dat dla konkretnego dnia tygodnia w ciągu danego roku	260
Określanie dat pierwszego i ostatniego wystąpienia określonego dnia tygodnia w danym miesiącu	267
Tworzenie kalendarza	274
Generowanie dat rozpoczynających i kończących poszczególne kwartały danego roku	291
Określanie daty początkowej i końcowej dla danego na wejściu kwartału	296
Odnajdywanie brakujących dat	303
Przeszukiwanie według określonych jednostek czasu	312
Porównywanie rekordów według określonych fragmentów dat	314
Identyfikacja wzajemnie pokrywających się przedziałów czasowych	317
10. Praca z przedziałami	323
Lokalizowanie przedziałów w ramach ciągów wartości	323
Odnajdywanie różnic pomiędzy wierszami należącymi do tej samej grupy lub wycinka danych	328
Lokalizowanie początków i końców przedziałów wartości następujących bezpośrednio po sobie	337
Uzupełnianie brakujących wartości w przedziale	342
Generowanie kolejnych wartości liczbowych	346
11. Zaawansowane przeszukiwanie	351
Podział zbioru wynikowego na strony	351
Pomijanie n wierszy tabeli	354
Stosowanie logiki alternatywy w zapytaniach wykorzystujących złączenia zewnętrzne	357
Identyfikacja par odwrotnych w przetwarzanym zbiorze wierszy	360
Wybieranie n najlepszych rekordów	361
Odnajdywanie rekordów z największymi i najmniejszymi wartościami	363
Badanie „przyszłych” wierszy	365
Przenoszenie wartości nieprzetworzonych	369
Przypisywanie ocen do wierszy zbioru wynikowego	372
Eliminowanie powtórzeń	373

Odnajdywanie wartości skoczka	375
Generowanie prostych prognoz	382
12. Raportowanie i magazynowanie danych	391
Konwertowanie zbioru wynikowego do postaci pojedynczego wiersza	391
Konwertowanie zbioru wynikowego do postaci zbioru wielowierszowego	393
Odwrotna transpozycja zbioru wynikowego	401
Odwrotna transpozycja zbioru danych do postaci zbioru jednokolumnowego	403
Eliminowanie powtórzeń ze zbioru wynikowego	406
Obracanie zbioru wynikowego celem wykonywania obliczeń w ramach skonsolidowanych wierszy	409
Tworzenie bloków danych tej samej wielkości	411
Tworzenie predefiniowanej liczby bloków danych	414
Tworzenie histogramów poziomych	419
Tworzenie histogramów pionowych	421
Zwracanie zbiorów wynikowych bez kolumn wykorzystywanych w procesie grupowania	424
Wyznaczanie prostych sum częściowych	429
Wyznaczanie sum częściowych dla wszystkich możliwych kombinacji wyrażeń	433
Identyfikowanie wierszy niebędących sumami częściowymi	442
Konwertowanie wierszy na wersję bitową za pomocą wyrażeń CASE	444
Tworzenie tzw. macierzy rzadkich	446
Grupowanie wierszy według określonych jednostek czasu	447
Jednoczesne agregowanie danych według różnych grup i bloków	451
Agregowanie zmiennych (ruchomych) przedziałów wartości	453
Obracanie zbioru wynikowego zawierającego sumy częściowe	460
13. Zapytania hierarchiczne	465
Wyrażanie relacji rodzic-potomek	466
Wyrażanie relacji potomek-rodzic-dziadek	469
Tworzenie hierarchicznych perspektyw dla istniejących tabel	475
Odnajdywanie wszystkich wierszy potomnych dla danego wiersza rodzica	483
Określanie wierszy występujących w rolach liści, gałęzi i korzeni	486
14. Rozmaitości	495
Tworzenie raportów krzyżowych za pomocą operatora PIVOT systemu SQL Server	495
Przywracanie oryginalnego kształtu obróconych raportów krzyżowych za pomocą operatora UNPIVOT systemu SQL Server	497
Transponowanie zbiorów wynikowych za pomocą klauzuli MODEL systemu Oracle	499
Wydobywanie interesujących nas elementów z różnych części łańcucha	503
Znajdowanie liczby dni w roku (rozwiązanie alternatywne tylko dla systemu Oracle)	506

Przeszukiwanie danych wejściowych pod kątem zawierania łańcuchów alfanumerycznych	507
Konwertowanie liczb całkowitych na reprezentacje binarne w systemie Oracle	509
Obracanie zbioru wynikowego z wartościami rankingowymi	512
Wstawianie nagłówków kolumn w dwukrotnie obróconych zbiorach wynikowych	516
Konwertowanie podzapytań skalarnych na podzapytania kompozytowe w systemie Oracle	526
Konwertowanie uszeregowanych danych do postaci osobnych wierszy	528
Wyznaczanie procentowych stosunków poszczególnych wartości względem sumy wszystkich wartości	533
Tworzenie w systemie Oracle list wartości oddzielonych przecinkami	535
Odnajdywanie kolejnych niepasujących wzorców w systemie Oracle	540
Transformacja danych za pomocą perspektywy wbudowanej	542
Testowanie występowania wartości w grupie	544
A Przypomnienie funkcji okienkowania	549
B Jeszcze raz o Rozenshteinie	575
Skorowidz	611

Odczytywanie rekordów

W niniejszym rozdziale skupimy się na najbardziej podstawowych wyrażeniach `SELECT`. Dobrze ich zrozumienie jest o tyle ważne, że znaczna część prezentowanych tutaj zagadnień będzie wykorzystywana nie tylko w kolejnych rozdziałach, ale też w Twojej codziennej pracy z językiem SQL.

Odczytywanie wszystkich wierszy i kolumn tabeli

Problem

Dysponujemy tabelą bazy danych i chcemy się zapoznać ze wszystkimi zawartymi w niej danymi.

Rozwiązanie

Należy użyć dla interesującej nas tabeli wyrażenia `SELECT` ze znakiem specjalnym gwiazdki (*):

```
1 select *  
2   from emp
```

Omówienie

Znak gwiazdki (*) ma w języku SQL znaczenie specjalne. Użycie tego znaku w wyrażeniu `SELECT` spowoduje zwrócenie wszystkich kolumn wskazanej tabeli. Ponieważ w prezentowanym rozwiązaniu nie ma klauzuli `WHERE`, zwrócony wynik będzie zawierał także wszystkie wiersze tabeli `EMP`. Alternatywnym rozwiązaniem byłoby zdefiniowanie wprost listy wszystkich kolumn tej tabeli:

```
select empno, ename, job, sal, mgr, hiredate, comm, deptno  
from emp
```

W przypadku zapytań budowanych ad hoc i wykonywanych w sposób interaktywny dużo prostszym rozwiązaniem jest stosowanie wyrażenia `SELECT *`. Podczas pisania trwałego kodu programu warto jednak wprost wymieniać odczytywane kolumny. Wydajność obu zapytań będzie identyczna, ale obecność konkretnych nazw kolumn ułatwi identyfikację pobieranych danych i będzie stanowiła pewne zabezpieczenie programu na wypadek zmian w bazie danych. Co więcej, tak zapisywane zapytania łatwiej zrozumieć osobom analizującym nasz kod (które nie muszą przecież znać wszystkich kolumn tworzących przetwarzaną tabelę).

Odczytywanie podzbioru wierszy tabeli

Problem

Dysponujemy tabelą bazy danych i chcemy się zapoznać wyłącznie z zawartością tych wierszy, które spełniają określony warunek.

Rozwiązanie

Należy użyć klauzuli `WHERE` definiującej warunek kwalifikowania wierszy do generowanego zbioru wynikowego. Przykładowo, aby uzyskać listę wszystkich pracowników zatrudnionych w dziesiątym dziale firmy, powinniśmy wykonać następujące wyrażenie:

```
1 select *
2   from emp
3  where deptno = 10
```

Omówienie

Klauzula `WHERE` umożliwia nam uzyskiwanie tylko tych wierszy, które nas interesują. Jeśli wyrażenie zdefiniowane w klauzuli `WHERE` okaże się prawdziwe dla danego wiersza, wiersz ten zostanie uwzględniony w zbiorze wynikowym.

Większość producentów oferuje w swoich systemach takie operatory porównawcze jak `=`, `<`, `>`, `<=`, `>=`, `!` oraz `<>`. Dodatkowo mamy możliwość definiowania klauzul, które będą wymagały od wierszy kwalifikowanych do zbioru wynikowego spełniania wielu warunków — można to osiągnąć, stosując operatory `AND`, `OR` oraz nawiasy (patrz następny przepis).

Odnajdywanie wierszy spełniających wiele warunków

Problem

Chcemy uzyskać wiersze, które spełniają wiele warunków.

Rozwiązanie

Należy użyć klauzuli `WHERE` zawierającej operatory `OR` i (lub) `AND`. Przykładowo, jeśli będziemy zainteresowani listą wszystkich pracowników dziesiątego działu firmy, pracowników, którzy są wynagradzani w trybie prowizyjnym, oraz wszystkich pracowników dwudziestego działu firmy zarabiających nie więcej niż 2000 dolarów miesięcznie, powinniśmy zastosować następujące wyrażenie `SELECT`:

```
1 select *
2   from emp
3  where deptno = 10
4     or comm is not null
5     or sal <= 2000 and deptno = 20
```

Omówienie

Możemy użyć kombinacji operatorów AND i OR oraz nawiasów, aby zakwalifikować do zbioru wynikowego wiersze spełniające wiele warunków. W prezentowanym przykładzie klauzula WHERE powinna wskazywać na wiersze, które:

- w kolumnie DEPTNO zawierają wartość 10,
- w kolumnie COMM zawierają wartość NULL,
- w kolumnie SAL zawierają wartość mniejszą lub równą 2000 i w kolumnie DEPTNO wartość 20.

Stosując nawiasy, wymusimy łączne wyznaczenie wartości warunków zawartych w tych nawiasach.

Przykładowo, zastanówmy się, jak zmieni się zbiór wynikowy, jeśli w naszym zapytaniu umieścimy nawiasy tak jak w poniższym przykładzie:

```
select *
from emp
where (    deptno = 10
        or comm is not null
        or sal <= 2000
      )
and deptno = 20
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980	800		20
7876	ADAMS	CLERK	7788	12-JAN-1983	1100		20

Odczytywanie podzbioru kolumn tabeli

Problem

Dysponujemy tabelą bazy danych i chcemy się zapoznać z wartościami określonych kolumn zamiast z zawartością wszystkich kolumn danej tabeli.

Rozwiązanie

Należy wymienić w klauzuli SELECT zapytania wszystkie interesujące nas kolumny. Przykładowo, aby uzyskać nazwisko, numer działu i wynagrodzenie pracowników, powinniśmy wykonać zapytanie SELECT w postaci:

```
1 select ename, deptno, sal
2   from emp
```

Omówienie

Wymieniając poszczególne kolumny w klauzuli SELECT, możemy zagwarantować, że zbiór wynikowy nie będzie zawierał żadnych dodatkowych, niespodziewanych danych. Tego rodzaju pewność jest szczególnie istotna w sytuacji, gdy żądania są kierowane do bazy danych za

pośrednictwem sieci, ponieważ eliminuje to opóźnienia wynikające z wyszukiwania i przesyłania niepotrzebnych danych.

Definiowanie znaczących nazw kolumn

Problem

Chcielibyśmy tak zmienić nazwy kolumn uwzględnianych w zbiorze wynikowym wygenerowanym przez nasze zapytanie, aby były bardziej czytelne i zrozumiałe. Przeanalizujmy zapytanie zwracające wysokość wynagrodzeń i prowizji otrzymywanych przez wszystkich pracowników:

```
1 select sal, comm
2   from emp
```

Czym jest sal? Skróconą wersją słowa sale? Czyimś nazwiskiem? Czym jest comm? Skróconą wersją słowa communication? Chcemy, aby etykiety kolumn tabeli wynikowej były bardziej zrozumiałe.

Rozwiązanie

Aby zmienić nazwy prezentowane w wynikach zapytań, należy użyć słowa kluczowego AS w następującej formie: *nazwa_oryginalna AS nowa_nazwa*. Niektóre bazy danych nie wymagają stosowania słowa kluczowego AS, ale wszystkie produkty akceptują i prawidłowo interpretują to słowo:

```
1 select sal as salary, comm as commission
2   from emp
```

SALARY	COMMISSION
--------	------------

800	
1600	300
1250	500
2975	
1250	1300
2850	
2450	
3000	
5000	
1500	0
1100	
950	
3000	
1300	

Omówienie

Stosowanie słowa kluczowego AS do nadawania nowych nazw kolumnom zwracanym przez nasze zapytanie często jest nazywane *nadawaniem aliasów* (ang. *aliasing*) tym kolumnom. Nowe nazwy są określane mianem *aliasów*. Zdefiniowanie dobrych aliasów może bardzo ułatwić ewentualnym odbiorcom wyników zapytania ich prawidłową interpretację.

Odwołania do aliasów kolumn w klauzuli WHERE

Problem

Używaliśmy już aliasów do definiowania bardziej zrozumiałych nazw kolumn dla generowanych zbiorów wynikowych; tym razem chcielibyśmy dodatkowo wyłączyć część wierszy za pomocą klauzuli WHERE. Okazuje się jednak, że próba bezpośredniego odwołania do nazw aliasów w tej klauzuli kończy się niepowodzeniem:

```
select sal as salary, comm as commission
      from emp
     where salary < 5000
```

Rozwiązanie

Opakowując nasze zapytanie w sposób, który przekształci je w wewnętrzną perspektywę, możemy sobie zapewnić możliwość odwołań do kolumn reprezentowanych przez aliasy:

```
1 select *
2   from (
3     select sal as salary, comm as commission
4       from emp
5     ) x
6  where salary < 5000
```

Omówienie

W tym prostym przykładzie moglibyśmy oczywiście uniknąć konieczności stosowania perspektywy wewnętrznej i użyć w klauzuli WHERE bezpośrednich odwołań do kolumn COMM i SAL (efekt byłby identyczny). Przedstawione rozwiązanie wprowadza mechanizm, którego będziemy potrzebowali podczas stosowania odwołań do następujących konstrukcji w klauzuli WHERE:

- funkcji agregujących,
- podzapytań skalarnych,
- funkcji okienkowania,
- aliasów.

Wróćmy do naszego przykładowego zapytania, w którym użyliśmy aliasu. Okazuje się, że aliasy przypisane kolumnom w perspektywie wewnętrznej mogą być przedmiotem odwołań w zapytaniu zewnętrznym. Dlaczego aliasy definiowane w perspektywie wewnętrznej są dla nas takie ważne? Otóż klauzula WHERE jest przetwarzana przed klauzulą SELECT, zatem aliasy SALARY i COMMISSION nie istnieją w momencie wyznaczania klauzuli WHERE oryginalnego zapytania (klauzuli z warunkiem SALARY < 5000). Z drugiej strony, klauzula FROM jest przetwarzana przed klauzulą WHERE. Umieszczając oryginalne zapytanie w klauzuli FROM, spowodujemy, że niezbędne aliasy zostaną zdefiniowane przed osiągnięciem skrajnie zewnętrznej klauzuli WHERE, zatem będą „widoczne” dla tej klauzuli. Opisana technika jest szczególnie przydatna w sytuacji, gdy nazwy kolumny tabeli są niezrozumiałe dla osób zmuszonych do analizy uzyskiwanych danych.



Perspektywa wewnętrzna w prezentowanym rozwiązaniu jest reprezentowana przez alias X. Nie wszystkie bazy danych wymagają reprezentowania wewnętrznych perspektyw za pomocą definiowanych wprost aliasów, ale wszystkie bazy danych akceptują tego rodzaju zabiegi składniowe.

Konkatenacja wartości kolumn

Problem

Chcemy zwracać wartości składowane w wielu kolumnach w formie pojedynczej kolumny. Wyobraźmy sobie na przykład, że chcemy skonstruować zapytanie, które na podstawie tabeli EMP wygeneruje następujący zbiór wynikowy:

```
CLARK WORKS AS A MANAGER
KING WORKS AS A PRESIDENT
MILLER WORKS AS A CLERK
```

Warto jednak pamiętać, że dane potrzebne do wygenerowania tego rodzaju wyników pochodzą z dwóch różnych kolumn tabeli EMP — ENAME oraz JOB:

```
select ename, job
from emp
where deptno = 10
```

ENAME	JOB
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

Rozwiązanie

Należy odnaleźć wbudowaną funkcję naszego systemu zarządzania bazą danych, która umożliwia konkatenację wartości, i użyć jej.

DB2, Oracle, PostgreSQL

Systemy zarządzania bazami danych DB2, Oracle, PostgreSQL wykorzystują w roli operatorów konkatenacji pary znaków pionowej linii:

```
1 select ename||' WORKS AS A '||job as msg
2   from emp
3  where deptno = 10
```

MySQL

Baza danych MySQL oferuje funkcję nazwaną CONCAT:

```
1 select concat(ename, ' WORKS AS A ', job) as msg
2   from emp
3  where deptno = 10
```

SQL Server

Użytkownicy systemu SQL Server mogą konkatelować wartości wielu tabel za pomocą standardowego operatora dodawania (+):

```
1 select ename + ' WORKS AS A ' + job as msg
2   from emp
3  where deptno = 10
```

Omówienie

Konkatenacja wartości pochodzących z wielu kolumn wymaga użycia funkcji CONCAT. Operator || jest skrótem funkcji CONCAT stosowanym w systemach baz danych DB2, Oracle i PostgreSQL, natomiast operator + jest skrótem funkcji CONCAT w systemie SQL Server.

Stosowanie logiki warunkowej w wyrażeniu SELECT

Problem

Chcemy wykonywać operacje IF-ELSE na wartościach zwróconych przez wyrażenie SELECT. Przykładowo, chcielibyśmy utworzyć zbiór wynikowy, w którym kolumna STATUS będzie zawierała wartość UNDERPAID dla pracowników zarabiających nie więcej niż 2000 dolarów, wartość OVERPAID dla pracowników zarabiających co najmniej 4000 dolarów lub wartość OK dla pracowników, których zarobki mieszczą się w przedziale od 2000 do 4000 dolarów. Zbiór wynikowy powinien mieć następującą postać:

ENAME	SAL	STATUS
SMITH	800	UNDERPAID
ALLEN	1600	UNDERPAID
WARD	1250	UNDERPAID
JONES	2975	OK
MARTIN	1250	UNDERPAID
BLAKE	2850	OK
CLARK	2450	OK
SCOTT	3000	OK
KING	5000	OVERPAID
TURNER	1500	UNDERPAID
ADAMS	1100	UNDERPAID
JAMES	950	UNDERPAID
FORD	1000	OK
MILLER	1300	UNDERPAID

Rozwiązanie

Musimy użyć wyrażenia CASE, za pośrednictwem którego możemy zdefiniować logikę warunkową bezpośrednio w naszym zapytaniu SELECT:

```
1 select ename, sal,
2        case when sal <= 2000 then 'UNDERPAID'
3             when sal >= 4000 then 'OVERPAID'
4             else 'OK'
5        end as status
6   from emp
```

Omówienie

Wyrażenie CASE umożliwia definiowanie rozmaitych elementów logiki warunkowej, której zachowania są uzależnione od wartości zwracanych przez zapytanie. Istnieje możliwość przypisania do wyrażenia CASE aliasu, który zapewni większą czytelność generowanego zbioru wyników. W prezentowanym przykładzie nadano wynikowi wyrażenia CASE alias STATUS. Klauzula ELSE ma charakter opcjonalny. Jeśli zrezygnujemy z tej klauzuli i dany wiersz nie zostanie dopasowany do żadnego z warunków testowych, wyrażenie CASE zwróci wartość NULL.

Ograniczanie liczby zwracanych wierszy

Problem

Chcemy ograniczyć liczbę wierszy zwracanych przez nasze zapytanie. Nie interesuje nas porządek wierszy w zbiorze wynikowym — chcemy tylko, by ich liczba wynosiła dokładnie *n*.

Rozwiązanie

Należy użyć odpowiedniej funkcji oferowanej przez nasz system zarządzania bazą danych umożliwiającej kontrolę liczby zwracanych wierszy.

DB2

W systemie bazy danych DB2 powinniśmy użyć klauzuli `FETCH FIRST`:

```
1 select *
2   from emp fetch first 5 rows only
```

MySQL i PostgreSQL

Ten sam efekt osiągniemy w systemach MySQL i PostgreSQL, stosując klauzulę `LIMIT`:

```
1 select *
2   from emp limit 5
```

Oracle

W systemie zarządzania bazą danych Oracle ograniczenia liczby wierszy zwracanych przez zapytania powinny mieć postać klauzul `WHERE` definiujących wartość progową funkcji `ROWNUM`:

```
1 select *
2   from emp
3  where rownum <= 5
```

SQL Server

W systemie SQL Server można ograniczyć liczbę zwracanych wierszy, stosując słowo kluczowe `TOP`:

```
1 select top 5 *
2   from emp
```

Omówienie

Wielu producentów oferuje obsługę klauzul podobnych do `FETCH FIRST` oraz `LIMIT`, dzięki którym możemy określać liczbę wierszy zwracanych przez zapytanie. Inaczej jest w systemie Oracle, gdzie jesteśmy zmuszeni użyć funkcji nazwanej `ROWNUM`, która zwraca liczbę wierszy wchodzących w skład zbioru wynikowego (licznik jest automatycznie zwiększany o jeden wraz z każdym wierszem kwalifikowanym do zbioru wynikowego).

Poniżej opisano procedurę wykonywania zapytania zawierającego w klauzuli `WHERE` warunek `ROWNUM <= 5`, który ogranicza liczbę zwracanych wierszy do pięciu:

1. System Oracle zaczyna wykonywać nasze zapytanie.
2. System Oracle odczytuje ze wskazanej tabeli pierwszy wiersz i przypisuje licznikowi wartość 1.
3. Czy osiągnęliśmy już limit pięciu wierszy? Jeśli nie, system Oracle zwróci dany wiersz, ponieważ zdefiniowane w klauzuli `WHERE` kryterium liczby wierszy mniejszej lub równej 5 jest spełnione. Jeśli tak, system Oracle nie zwróci kolejnego wiersza.
4. System Oracle odczytuje kolejny wiersz ze wskazanej tabeli i zwiększa wartość licznika wierszy (do 2, do 3, do 4 itd.).
5. Wróć do kroku 3.

Z powyższego opisu wynika, że funkcja `ROWNUM` systemu Oracle przypisuje wartość odpowiedniemu licznikowi *po* odczytaniu kolejnego wiersza. Sekwencja działań jest w tym przypadku niezwykle istotna. Wielu programistów baz danych Oracle próbuje określać, że zbiór wynikowy powinien zawierać np. tylko pięć wierszy za pomocą wyrażenia `ROWNUM = 5`. Stosowanie warunku równościowego dla wartości funkcji `ROWNUM` nie jest jednak dobrym rozwiązaniem. Poniżej opisano procedurę wykonywania zapytania zawierającego w klauzuli `WHERE` warunek `ROWNUM <= 5`:

1. System Oracle zaczyna wykonywać nasze zapytanie.
2. System Oracle odczytuje ze wskazanej tabeli pierwszy wiersz i przypisuje licznikowi wartość 1.
3. Czy osiągnęliśmy już limit pięciu wierszy? Jeśli nie, system Oracle pomija dany wiersz, ponieważ zdefiniowane w klauzuli `WHERE` kryterium liczby wierszy równej 5 jest spełnione. Jeśli tak, system Oracle zwróci dany wiersz. Warto jednak pamiętać, że odpowiedź na to pytanie nigdy nie będzie pozytywna!
4. System Oracle odczytuje kolejny wiersz ze wskazanej tabeli i przypisuje mu numer 1. Wynika to z faktu, że pierwszy wiersz zwracany przez zapytanie zawsze musi być oznaczany numerem 1.
5. Wróć do kroku 3.

Jeśli dokładnie przeanalizujesz opisaną powyżej procedurę, przekonasz się, dlaczego stosowanie warunku równościowego `ROWNUM = 5` nie przyniesie spodziewanych rezultatów. Nie możemy przecież otrzymać zbioru wynikowego złożonego z pięciu wierszy, jeśli wcześniej nie dodamy do tego zbioru wierszy od pierwszego do czwartego!

Być może się domyśliłeś, że warunek `ROWNUM = 1` spowoduje — zgodnie z oczekiwaniami — wygenerowanie jednowierszowego zbioru wynikowego, co pozornie jest sprzeczne z powyższymi wyjaśnieniami. Powodem prawidłowej interpretacji warunku `ROWNUM = 5` jest to, że określenie w systemie Oracle, czy dana tabela zawiera choć jeden wiersz, wymaga od tego systemu odczytania tego wiersza. Jeśli ponownie przeanalizujesz przedstawioną powyżej pięciopunktową procedurę, zastępując testowaną liczbę 5 liczbą 1, zrozumiesz, dlaczego warunek `ROWNUM = 1` prawidłowo ogranicza licznosc zbioru wynikowego do jednego wiersza.

Zwracanie n losowych rekordów tabeli

Problem

Chcemy zwrócić określoną liczbę losowo wybranych rekordów tabeli. Naszym celem jest taka modyfikacja poniższego wyrażenia, która spowoduje, że następujące po sobie wywołania będą zwracały różne zbiory, z których każdy będzie się składał z pięciu losowo wybranych wierszy:

```
select ename, job
  from emp
```

Rozwiązanie

Należy użyć funkcji wbudowanej w wykorzystywany system zarządzania bazą danych, która zwraca wartości losowe. Powinniśmy zastosować tę funkcję w klauzuli `ORDER BY`, aby posortować wiersze w sposób przypadkowy. Następnie należy wykorzystać opisaną w poprzednim podrozdziale technikę ograniczania liczby losowo posortowanych wierszy, które znajdują się w zbiorze wynikowym.

DB2

Należy użyć wbudowanej funkcji `RAND` łącznie z klauzulą `ORDER BY` i wspominaną wcześniej funkcją `FETCH`:

```
1 select ename, job
2   from emp
3  order by rand() fetch first 5 rows only
```

MySQL

Należy użyć wbudowanej funkcji `RAND` łącznie z funkcją `FETCH` wywoływaną w ciele klauzuli `ORDER BY`:

```
1 select ename, job
2   from emp
3  order by rand() limit 5
```

PostgreSQL

Powinniśmy użyć wbudowanej funkcji `RANDOM` łącznie z funkcją `FETCH` i klauzulą `ORDER BY`:

```
1 select ename, job
2   from emp
3  order by random() limit 5
```

Oracle

Należy użyć wbudowanej funkcji `VALUE` będącej częścią wbudowanego pakietu `DBMS_RANDOM` łącznie z klauzulą `ORDER BY` oraz wywołaniem funkcji `ROWNUM` klauzuli `WHERE`:

```
1 select *
2   from (
3     select ename, job
4       from emp
5     order by dbms_random.value()
6           )
7   where rownum <= 5
```

SQL Server

Aby uzyskać losowy pięcioelementowy zbiór wynikowy, należy użyć wbudowanej funkcji `NEWID` w połączeniu ze słowem kluczowym `TOP` oraz klauzulą `ORDER BY`:

```
1 select top 5 ename, job
2   from emp
3   order by newid()
```

Omówienie

Klauzula `ORDER BY` może otrzymywać wartość zwróconą przez funkcję i na jej podstawie modyfikować porządek elementów w zbiorze wynikowym. Wszystkie zapytania przedstawione w punkcie „Rozwiązanie” sprawdzają liczbę wierszy w zbiorze wynikowym *po* wykonaniu funkcji wywoływanej w klauzuli `ORDER BY`. Użytkownicy systemów zarządzania bazą danych innych niż Oracle powinni skorzystać z okazji i przeanalizować rozwiązanie zaproponowane właśnie dla tego systemu, ponieważ właśnie to zapytanie najlepiej ilustruje ideę stojącą także za pozostałymi zapytaniami.

W żadnym razie nie należy mylić stosowania funkcji w klauzuli `ORDER BY` ze stosowaniem stałych numerycznych. Podczas określania stałej numerycznej w klauzuli `ORDER BY` tak naprawdę żądamy, by sortowanie uwzględniało zawartość określonej kolumny wymienionej w klauzuli `SELECT`. Jeśli w klauzuli `ORDER BY` umieścimy wywołanie jakiejś funkcji, operacja sortowania będzie wykonywana na wynikach tej funkcji wyznaczanych dla każdego kolejnego wiersza.

Odnajdywanie wartości pustych

Problem

Chcemy odnaleźć wszystkie wiersze, które zawierają wartości puste w określonej kolumnie.

Rozwiązanie

Aby określić, czy w danym wierszu określona kolumna zawiera wartość pustą, musimy użyć wyrażenia `IS NULL`.

```
1 select *
2   from emp
3  where comm is null
```

Omówienie

Wartość NULL nigdy nie jest ani równa, ani różna od czegokolwiek (nawet od samej siebie), zatem nie możemy sprawdzać ewentualnego zawierania tej wartości w wierszu za pomocą standardowych `=` lub `!=`. Aby określić, czy dany wiersz zawiera wartość NULL, musimy użyć wyrażenia `IS NULL`. Odnajdywanie wierszy, które nie zawierają wartości pustych w określonej kolumnie, wymaga zastosowania w klauzuli `WHERE` wyrażenia `IS NOT NULL`.

Tłumaczenie wartości pustych na wartości rzeczywiste

Problem

Mamy do czynienia z wierszami zawierającymi wartości puste i chcielibyśmy, aby w ich miejsce były zwracane inne, konkretne wartości.

Rozwiązanie

Należy użyć funkcji `COALESCE`, aby zastąpić wartości puste właściwymi wartościami (w tym przypadku zerami):

```
1 select coalesce(comm, 0)
2   from emp
```

Omówienie

Funkcja `COALESCE` otrzymuje za pośrednictwem swoich argumentów jedną lub wiele wartości wejściowych. Funkcja zwraca pierwszą wartość różną od NULL z tej listy. Oznacza to, że w prezentowanym przykładzie wartość kolumny `COMM` jest zwracana pod warunkiem, że nie jest wartością pustą. W przeciwnym razie funkcja zwraca zero.

Podczas pracy z wartościami pustymi zawsze warto korzystać z wbudowanej funkcjonalności stosowanego systemu zarządzania bazą danych (DBMS). Znaczna część tych systemów oferuje wiele funkcji, które można z powodzeniem używać do realizacji tego zadania. W prezentowanym rozwiązaniu wykorzystaliśmy funkcję `COALESCE`, ponieważ działa ona we wszystkich systemach zarządzania bazami danych. Drugą uniwersalną konstrukcją języka SQL obsługiwaną przez wszystkie systemy jest klauzula `CASE`:

```
select case
      when comm is null then 0
      else comm
    end
   from emp
```

Chociaż wyrażenie `CASE` może być wykorzystywane do tłumaczenia wartości NULL na inne wartości, powyższy przykład dowodzi, że funkcja `COALESCE` jest zdecydowanie prostsza w użyciu i bardziej zwięzła.

Poszukiwanie wzorców

Problem

Chcemy zwrócić wiersze spełniające określone kryteria — wiersze, których zawartość można dopasować do określonego podłańcucha lub wzorca. Przeanalizujmy poniższe zapytanie i zbiór wynikowy:

```
select ename, job
  from emp
 where deptno in (10, 20)
```

ENAME	JOB
SMITH	CLERK
JONES	MANAGER
CLARK	MANAGER
SCOTT	ANALYST
KING	PRESIDENT
ADAMS	CLERK
FORD	ANALYST
MILLER	CLERK

Chcemy, aby zbiór wynikowy zawierał nazwiska i stanowiska pracy tylko tych pracowników dziesiątego i dwudziestego działu, których nazwiska zawierają literę I lub których nazwa stanowiska pracy kończy się literami ER.

ENAME	JOB
SMITH	CLERK
JONES	MANAGER
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

Rozwiązanie

Należy użyć operatora LIKE łącznie ze stosowanym w języku SQL symbolem wieloznacznym %:

```
1 select ename, job
2   from emp
3  where deptno in (10, 20)
4     and (ename like '%I%' or job like '%ER')
```

Omówienie

Operator % stosowany łącznie z uniwersalnym operatorem dopasowywania wzorców LIKE jest dopasowywany do dowolnej sekwencji znaków. Większość implementacji standardu SQL dodatkowo oferuje operator podkreślenia (), który jest dopasowywany do pojedynczych znaków. Umieszczając prosty, jednoliterowy wzorec przeszukiwania I pomiędzy dwoma operatorami %, określamy, że do zbioru wynikowego będą kwalifikowane łańcuchy zawierające literę I (na dowolnej pozycji). Gdybyśmy użyli tylko jednego operatora %, zawartość zbioru wynikowego naszego zapytania byłaby uzależniona od miejsca umieszczenia tego operatora. Przykładowo, aby otrzymać stanowiska pracy o nazwach kończących się literami ER, powinniśmy szukany łańcuch ER poprzedzić operatorem %; gdybyśmy chcieli odnaleźć nazwy stanowisk pracy rozpoczynające się od tych liter, powinniśmy użyć wyrażenia regularnego %ER.