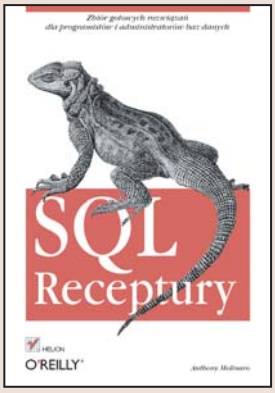
SQL. Receptury



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.

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 1.

Odczytywanie rekordów

W niniejszym rozdziale skupimy się na najbardziej podstawowych wyrażenia `SELECT`. Dobre 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 wyrażeń `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 konkatenować 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 wynikowego. 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 liczność 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 znajdą 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()
8         )
9   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 wzorzec 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`.