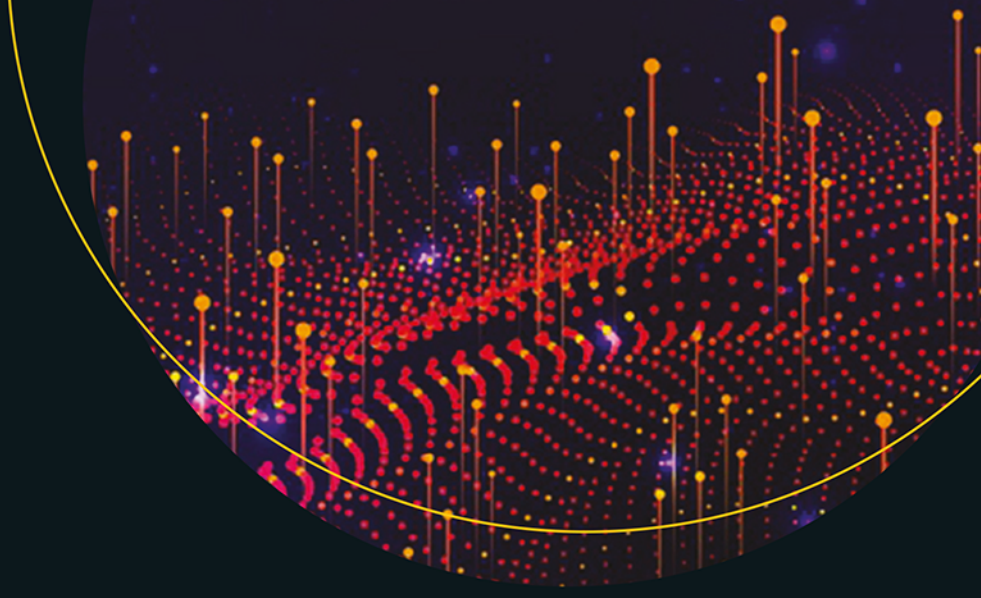


Praktyczna nauka SQL dla Oracle

Wykorzystaj ogromne możliwości bazy danych Oracle

—

Kim Berg Hansen

Helion

Apress

Tytuł oryginału: Practical Oracle SQL: Mastering the Full Power of Oracle Database

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-8733-1

First published in English under the title Practical Oracle SQL: Mastering the Full Power of Oracle Database by Kim Berg Hansen, edition: 1

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/prnsql.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/prnsql
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 5.

Funkcje zdefiniowane w języku SQL

Jedną z pięknych stron języka SQL w Oracle jest możliwość jego łatwej rozbudowy przez tworzenie funkcji wywoływanych w kodzie SQL. Zwykle to będą funkcje PL/SQL, choć w niektórych przypadkach mogą być utworzone również w językach C lub Java. Dzięki nowemu silnikowi w przyszłości będzie możliwe tworzenie procedur składowanych i funkcji w wielu różnych językach.

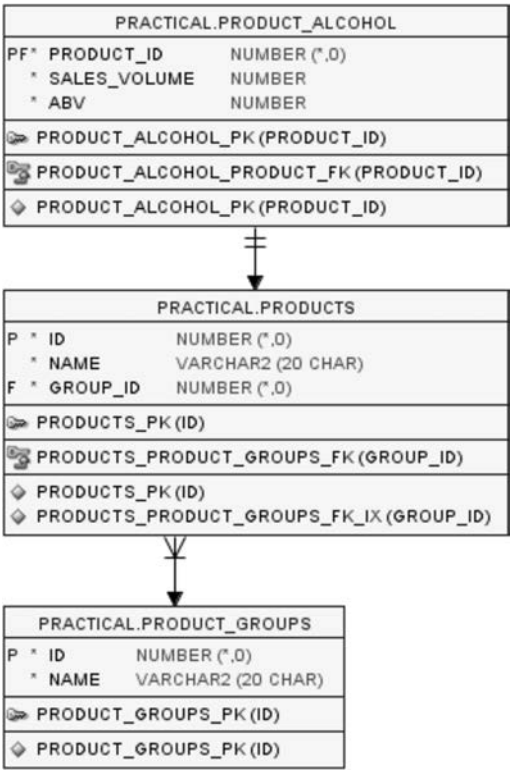
Trzeba w tym miejscu wyraźnie podkreślić, że kod SQL i PL/SQL jest wykonywany przez dwa oddzielne silniki, między którymi istnieją niewielkie różnice w zakresie np. sposobu obsługi zmiennych, typów danych i pamięci. W trakcie każdej operacji wywołania przez SQL funkcji PL/SQL lub na odwrót, gdy PL/SQL wykonuje statyczny bądź dynamiczny kod SQL, dane są przekazywane między wspomnianymi silnikami. Podczas tej operacji może się odbywać również konwersja — to nazywamy przełączaniem kontekstu.

Operacja przełączania kontekstu jest bardzo krótka i zwykle nie ma powodów do zastanawiania się nad nią. Jeżeli jednak funkcja będzie wywoływana z poziomu kodu SQL tysiące razy w ciągu sekundy, wówczas czasy poszczególnych operacji przełączania kontekstu zsumują się i ostatecznie będą miały pokaźny udział w całkowitym czasie wykonania zapytania. W wersji 12.1 serwera Oracle istnieje możliwość minimalizacji liczby operacji przełączania kontekstu, a tym samym staje się ono ledwo zauważalne.

Tabela z danymi zawartości alkoholu w piwie

Aby pokazać minimalną funkcję przełączania kontekstu w kodzie SQL, posłużę się tabelą `product_alcohol` o strukturze pokazanej na rysunku 5.1.

W tej tabeli dla każdego rodzaju piwa jest przechowywana objętość (wyrażona w mililitrach) jednostki sprzedaży (butelka lub inny pojemnik) oraz procentowa zawartość alkoholu. Kod przedstawiony na listingu 5.1 powoduje wyświetlenie danych dla piwa w grupie 142, którą jest Stout (to względnie mocne i ciemne piwo).



Rysunek 5.1. *Tabela product_alcohol zawiera dane niezbędne do obliczania zawartości alkoholu w piwie*

Listing 5.1. *Dotyczące alkoholu dane dla piwa w grupie Stout*

```
SQL> select
  2     p.id as p_id
  3   , p.name
  4   , pa.sales_volume as vol
  5   , pa.abv
  6  from products p
  7  join product_alcohol pa
  8     on pa.product_id = p.id
  9  where p.group_id = 142
 10  order by p.id;
```

Piwo Reindeer Fuel jest rozlewane w butelkach półlitrowych (500 ml), a jego procentowa zawartość alkoholu wynosi tylko 6%. Dwa pozostałe piwa w tej grupie są rozlewane w standardowych butelkach 330 ml, a ich procentowa zawartość alkoholu jest większa.

```
P_ID  NAME              VOL  ABV
----  ----------------  ---  ---
4040  Coalminers Sweat  330  8.5
4160  Reindeer Fuel     500  6
4280  Hoppy Crude Oil   330  7
```

Te dane można wykorzystać do ustalenia, ile czystego alkoholu zawiera butelka danego piwa. Dzięki temu można obliczyć stężenie alkoholu we krwi po wypiciu danego piwa.

Stężenie alkoholu we krwi

Firma Good Beer Trading Co musi stosować się do przepisów, zgodnie z którymi na każdym opakowaniu napoju trzeba podać maksymalne stężenie alkoholu we krwi po wypiciu danej butelki piwa. Ta wartość będzie różna dla kobiet i mężczyzn. Zależy od masy ciała — musi być podana dla kobiety o wadze 60 kg i mężczyzny o wadze 80 kg.

Stężenie alkoholu we krwi musi być obliczone jako liczba gramów alkoholu w mililitrze krwi; zwykle jest podawane w procentach. Dlatego też stężenie 0,04 oznacza, że 0,04% krwi to czysty alkohol. Do obliczenia tej wartości można skorzystać ze wzoru Widmarka.

■ **Wzór Widmarka** Wyrażona w mililitrach ilość wypitego napoju * procentowa zawartość alkoholu / 100 = mililitry alkoholu. Mililitry alkoholu * 0,789 (ciężar właściwy alkoholu) = gramy alkoholu. Waga ciała * 1000 * procentowa zawartość wody w ciele danej płci = mililitry płynu w ciele. (W przypadku kobiet procentowa zawartość wody wynosi 55%, a w przypadku mężczyzn to 68%). 100 * gramy alkoholu / mililitry płynu we krwi = stężenie alkoholu we krwi.

Po zastosowaniu w kodzie SQL wzoru Widmarka można obliczyć żądane wartości stężenia alkoholu we krwi, jak pokazałem na listingu 5.2.

Listing 5.2. *Obliczanie stężenia alkoholu we krwi dla kobiet i mężczyzn*

```
SQL> select
  2     p.id as p_id
  3   , p.name
  4   , pa.sales_volume as vol
  5   , pa.abv
  6   , round(
  7         100 * (pa.sales_volume * pa.abv / 100 * 0.789)
  8          / (80 * 1000 * 0.68)
  9       , 3
 10     ) bac_m
 11   , round(
 12         100 * (pa.sales_volume * pa.abv / 100 * 0.789)
 13          / (60 * 1000 * 0.55)
 14       , 3
 15     ) bac_f
 16  from products p
 17  join product_alcohol pa
 18     on pa.product_id = p.id
 19  where p.group_id = 142
 20  order by p.id;
```

W wierszach od 6. do 10. następuje obliczenie stężenia alkoholu we krwi dla mężczyzny o wadze 80 kg, a w wierszach od 11. do 15. te same obliczenia są przeprowadzane dla kobiety o wadze 60 kg. Mężczyzna ma więcej wody w organizmie (zarówno z powodu płci, jak i większej wagi), więc szybciej spala alkohol i ma mniejsze jego stężenie we krwi.

Te obliczenia dostarczają kolumnom `bac_m` i `bac_f` wartości, które firma Good Beer Trading Co musi później umieszczać na etykietach butelek piwa i na kartonach.

```
P_ID  NAME              VOL  ABV  BAC_M  BAC_F
----  ----------------  ---  ---  -----  -----
4040  Coalminers Sweat  330  8.5  0.041  0.067
4160  Reindeer Fuel     500  6    0.044  0.072
4280  Hoppy Crude Oil   330  7    0.034  0.055
```

Dzięki obliczonym wcześniej wartościom można sprawdzić, czy np. dany kraj to jeden z tych, w których można prowadzić pojazdy mechaniczne przy stężeniu alkoholu we krwi wynoszącym 0,05%. (W niektórych krajach tę wartość podaje się w promilach, wówczas wynosi ona 0,5‰). Wypicie takiego piwa przez kobietę o wadze 60 kg spowoduje przekroczenie przez nią dozwolonego stężenia alkoholu we krwi i narazi ją na odpowiedzialność karną za jazdę po spożyciu. Natomiast w przypadku mężczyzny o wadze 80 kg stężenie alkoholu we krwi będzie poniżej dozwolonej granicy.

■ **Uwaga** To są przykładowe dane niezbędne do zilustrowania wzoru zapisanego w kodzie SQL. Rzeczywiste stężenie alkoholu we krwi będzie zależało od wielu innych czynników, m.in. budowy ciała i metabolizmu, więc nie należy go używać do ustalenia, czy można zasiąść za kierownicą po wypiciu kilku butelek piwa. Te przykłady są przeznaczone jedynie do nauki języka SQL — nie biorę odpowiedzialności za mandaty, które możesz otrzymać, a także apeluję o rozsądne spożywanie napojów alkoholowych.

Jako programista, prawdopodobnie wyraźnie dostrzegasz, że kod przedstawiający wzór Widmarka powinien zostać umieszczony w funkcji, a nie powielony z nieco innymi wartościami w zapytaniu.

Zdefiniowanie funkcji za pomocą PRAGMA UDF

W kodzie zamieszczonym na listingu 5.3 najpierw została utworzona zwykła (no prawie zwykła) funkcja PL/SQL wykorzystująca wzór Widmarka do obliczenia stężenia alkoholu we krwi. Wprawdzie to nie ma tutaj żadnego znaczenia, ale zastosowałem się do najlepszej praktyki, polegającej na umieszczaniu funkcji w pakiecie zamiast definiowania jej jako samodzielnej funkcji. Dlatego też postanowiłem utworzyć pakiet `formulas` przeznaczony dla tego rodzaju funkcji.

Listing 5.3. *Tworzenie pakietu wzoru dla funkcji obliczającej stężenie alkoholu we krwi*

```
SQL> create or replace package formulas
  2  is
  3     function bac (
  4        p_volume in number
  5      , p_abv    in number
  6      , p_weight in number
  7      , p_gender in varchar2
  8     ) return number deterministic;
  9  end formulas;
 10  /

Package FORMULAS compiled
```

```
SQL> create or replace package body formulas
  2  is
  3     function bac (
  4        p_volume in number
  5      , p_abv    in number
  6      , p_weight in number
  7      , p_gender in varchar2
  8     ) return number deterministic
  9     is
 10        PRAGMA UDF;
 11     begin
 12        return round(
 13           100 * (p_volume * p_abv / 100 * 0.789)
 14            / (p_weight * 1000 * case p_gender
 15                                    when 'M' then 0.68
 16                                    when 'F' then 0.55
 17                                 end)
 18         , 3
 19        );
 20     end bac;
 21  end formulas;
 22  /

Package Body FORMULAS compiled
```

Przeznaczenie tego kodu jest dość oczywiste i niejasny może być jedynie wiersz 10. **`PRAGMA UDF`** (ang. *user-definied function*, czyli funkcja zdefiniowana przez użytkownika) to dyrektywa kompilatora wprowadzona w wersji 12.1. Jej celem jest wskazanie kompilatorowi, że dana funkcja ma być *przede wszystkim* wywoływana z poziomu kodu SQL, a nie PL/SQL.

Jeżeli funkcja zostanie zdefiniowana *bez* tej dyrektywy, zostanie skompilowana w standardowy sposób, co oznacza przełączenie kontekstu podczas wywoływania danej funkcji. Natomiast zdefiniowanie funkcji *razem* z podaną dyrektywą powoduje je skompilowanie w inny sposób, który może zmniejszyć obciążenie związane z przełączaniem kontekstu. Poziom (ewentualnego) zmniejszenia obciążenia może być niezależny od programisty. Do tego zagadnienia jeszcze powrócę, ale najpierw pokażę, jak używać naszej funkcji.

Z funkcją zdefiniowaną przez użytkownika pracuje się w dokładnie taki sam sposób jak ze zwykłą funkcją. Na listingu 5.4 pokazałem przykład zapytania pobierającego wartości stężenia alkoholu we krwi za pomocą wywołań funkcji w pakiecie.

Listing 5.4. *Pobieranie wartości stężenia alkoholu we krwi dla kobiety i mężczyzny*

```
SQL> select
  2     p.id as p_id
  3   , p.name
  4   , pa.sales_volume as vol
  5   , pa.abv
  6   , formulas.bac(pa.sales_volume, pa.abv, 80, 'M') bac_m
  7   , formulas.bac(pa.sales_volume, pa.abv, 60, 'F') bac_f
  8  from products p
  9  join product_alcohol pa
 10     on pa.product_id = p.id
 11  where p.group_id = 142
 12  order by p.id;
```

Wygenerowane zostają takie same dane wyjściowe jak w przypadku listingu 5.2, więc nie ma tutaj żadnych niespodzianek.

To rozwiązanie jest łatwe w użyciu, ponieważ kod funkcji jest tworzony w zwykły sposób. Jeżeli wiadomo, że dana funkcja będzie wywołana przede wszystkim z poziomu kodu SQL i rzadziej (o ile w ogóle) z PL/SQL, wówczas wystarczy dodać dyrektywę `PRAGMA UDF`, a kompilator zajmie się resztą. Potencjalną korzyścią jest zmniejszenie w trakcie wykonywania kodu obciążenia związanego z przełączaniem kontekstu.

Wielkość korzyści z użycia dyrektywy `PRAGMA UDF` będzie zależała od wielu czynników. Jeżeli kod zdefiniowany w funkcji PL/SQL składa się z poleceń, które *można* wyrazić bezpośrednio za pomocą czystego kodu SQL (takich jak funkcja `formulas.bac`), wówczas korzyści prawdopodobnie będą większe. Natomiast zastosowanie znacznie bardziej skomplikowanych funkcji, zawierających sporo funkcjonalności PL/SQL bądź osadzonego kodu SQL, przynosi tylko niewielkie korzyści lub nie przynosi żadnych. Zawsze należy przeprowadzać odpowiednie testy. Ogólnie rzecz biorąc, dyrektywa `PRAGMA UDF` nie zaszkodzi, a może nieco pomóc, gdy *wiadomo*, że funkcja będzie używana praktycznie wyłącznie z poziomu kodu SQL.

Podczas kompilowania funkcji zdefiniowanej razem z dyrektywą `PRAGMA UDF` staram się możliwie zmniejszyć koszt jej wywołania z poziomu kodu SQL. To jednocześnie oznacza, że nie przejmuję się *potencjalnie* większym kosztem jej wywołania z poziomu PL/SQL. W zależności od wielu czynników może tutaj wystąpić niewielki efekt uboczny, ponieważ funkcja zdefiniowana z `PRAGMA UDF` oczekuje danych w formacie dostarczanym przez silnik SQL. Ten efekt może być niezauważalny, wszystko zależy od konkretnych okoliczności.

Zamiast używać funkcji `PRAGMA UDF`, można pominąć tworzenie funkcji składowej w bazie danych i po prostu podać tę funkcję w samym zapytaniu.

Zdefiniowanie funkcji za pomocą klauzuli WITH

Wersja 12.1 bazy danych Oracle pozwala umieszczać kod funkcji PL/SQL (a także procedury, co rzadko okazuje się użyteczne) bezpośrednio w klauzuli `WITH` zapytania, jak pokazałem na listingu 5.5.

Listing 5.5. *Obliczanie stężenia alkoholu we krwi za pomocą klauzuli WITH*

```
SQL> with
  2     function bac (
  3        p_volume in number
  4      , p_abv    in number
  5      , p_weight in number
  6      , p_gender in varchar2
  7     ) return number deterministic
  8     is
  9     begin
 10        return round(
 11           100 * (p_volume * p_abv / 100 * 0.789)
 12            / (p_weight * 1000 * case p_gender
 13                                    when 'M' then 0.68
 14                                    when 'F' then 0.55
 15                                 end)
```

```
16         , 3
17        );
18     end;
19  select
20     p.id as p_id
21   , p.name
22   , pa.sales_volume as vol
23   , pa.abv
24   , bac(pa.sales_volume, pa.abv, 80, 'M') bac_m
25   , bac(pa.sales_volume, pa.abv, 60, 'F') bac_f
26  from products p
27  join product_alcohol pa
28     on pa.product_id = p.id
29  where p.group_id = 142
30  order by p.id
31  /
```

Na początku kodu znajduje się słowo kluczowe `WITH`, podobnie jak w przykładach zamieszczonych w rozdziale 3. Jednak zamiast podzapytania wiersze od 2. do 18. zawierają funkcję `bac()`, podobną do zdefiniowanej w pakiecie `formulas`. Może być ona wywoływana z poziomu kodu SQL, jak pokazałem w wierszach 24. i 25. Dane wyjściowe tego zapytania są identyczne z wygenerowanymi przez kod przedstawiony na listingu 5.2.

Funkcja zdefiniowana w klauzuli `WITH` jest kompilowana w taki sam sposób jak funkcja z dyrektywą `PRAGMA UDF`, przy czym nie jest przechowywana w słowniku danych, jak w przypadku obiektu PL/SQL. Taka funkcja zostanie razem z zapytaniem zapisana jedynie w puli współdzielonej i *nie* może być wywoływana z poziomu innego zapytania SQL lub PL/SQL.

■ **Uwaga** Wiersz 31. na listingu 5.5 kończy się ukośnikiem / zamiast średnikiem. Gdy analizator składni ustali, że klauzula `WITH` zawiera kod PL/SQL, nie będzie w stanie (przynajmniej obecnie) stwierdzić, czy ukośnik wskazuje koniec polecenia czy też fragment kodu PL/SQL. To może się zmienić w przyszłych wersjach bazy danych. Obecnie rozwiązaniem jest użycie ukośnika, który pozwala wskazać SQLcl lub SQL*Plus koniec polecenia.

Istnieje możliwość zdefiniowania wielu funkcji w pojedynczej klauzuli `WITH`. Na przykład mogę się zdecydować na refaktoryzację kodu i utworzenie dwóch funkcji pomocniczych, przeznaczonych do obliczania gramów alkoholu i gramów płynu w ciele (wartości wyrażone w mililitrach). Następnie obie funkcje pomocnicze będą używane w funkcji `bac()`. Przykład takiego rozwiązania zaprezentowałem na listingu 5.6, który jest nieco dłuższy od wcześniejszych, ale jednocześnie znacznie bardziej samodokumentujący się.

Listing 5.6. *Przykład zdefiniowania wielu funkcji w klauzuli WITH*

```
SQL> with
  2     function gram_alcohol (
  3        p_volume in number
  4      , p_abv    in number
  5     ) return number deterministic
  6     is
  7     begin
```

```
 8        return p_volume * p_abv / 100 * 0.789;
 9     end;
10     function gram_body_fluid (
11        p_weight in number
12      , p_gender in varchar2
13     ) return number deterministic
14     is
15     begin
16        return p_weight * 1000 * case p_gender
17                                    when 'M' then 0.68
18                                    when 'F' then 0.55
19                                 end;
20     end;
21     function bac (
22        p_volume in number
23      , p_abv    in number
24      , p_weight in number
25      , p_gender in varchar2
26     ) return number deterministic
27     is
28     begin
29        return round(
30           100 * gram_alcohol(p_volume, p_abv)
31            / gram_body_fluid(p_weight, p_gender)
32         , 3
33        );
34     end;
35  select
...
```

Wiele funkcji zdefiniowanych w klauzuli `WITH` nie ma wpływu na generowane dane wyjściowe — pozostają bez zmian.

Do programisty należy decyzja, czy chce używać pojedynczej funkcji czy ich większej liczby. Jeżeli funkcja ma być wywołana w wielu zapytaniach SQL, wówczas należy utworzyć funkcję składowaną (z dyrektywą `PRAGMA UDF` lub bez niej), co do tego nie ma żadnych wątpliwości. Natomiast w innych przypadkach może być zasadne pytanie o sens umieszczania funkcji w klauzuli `WITH` zamiast jej zdefiniowania razem z dyrektywą `PRAGMA UDF`.

Jednym z takich przypadków jest sytuacja, w której np. nie można zdefiniować funkcji lub procedury składowej, ponieważ baza danych jest w trybie tylko do odczytu bądź tworzone są polecenia i narzędzia przeznaczone do uruchamiania bez konieczności instalowania kodu w bazach danych klientów.

Jeszcze inna sytuacja to taka, w której funkcja w rzadkich przypadkach wykonuje dynamiczny kod SQL i z jakiegokolwiek powodu nie może używać wiązania zmiennych, a zamiast tego do przygotowania kodu SQL korzysta z konkatenacji ciągu tekstowego. Zdefiniowanie funkcji w zapytaniu zapewnia pełną kontrolę nad argumentami używanymi podczas jej wywoływania, co z kolei pozwala zabezpieczyć się przed atakami typu SQL injection. Taka funkcja nie będzie mogła być wywołana z innego miejsca.

Trzeci przypadek zachodzi wtedy, gdy funkcjonalność jest ściśle związana z pojedynczym celem, i wówczas można zdecydować się na inny sposób hermetyzacji kodu.

Hermetyzacja kodu w widoku

W przypadku omawianej tutaj aplikacji rozsądne jest stwierdzenie, że obliczanie stężenia alkoholu we krwi nie ma sensu poza kontekstem rekordu w tabeli `product_alcohol`. Uciekając się do analogii programowania zorientowanego obiektowo, można by powiedzieć, że to jest metoda *egzemplarza*, a nie *statyczna*.

Podobny efekt można osiągnąć przez utworzenie widoku, jak pokazałem na listingu 5.7.

Listing 5.7. *Tworzenie widoku przeznaczonego do obliczania stężenia alkoholu we krwi*

```
SQL> create view product_alcohol_bac
  2  as
  3  with
  4     function gram_alcohol (
...
 12     function gram_body_fluid (
...
 23     function bac (
...
 37  select
 38     pa.product_id
 39   , pa.sales_volume
 40   , pa.abv
 41   , bac(pa.sales_volume, pa.abv, 80, 'M') bac_m
 42   , bac(pa.sales_volume, pa.abv, 60, 'F') bac_f
 43  from product_alcohol pa
 44  /

View PRODUCT_ALCOHOL_BAC created.
```

W tym widoku użyłem klauzuli `WITH` razem z trzema funkcjami pochodzącymi z listingu 5.6. Zapytanie zostało zdefiniowane w wierszach od 37. do 43. — używa jedynie tabeli `product_alcohol` i pobiera z niej wszystkie kolumny plus wartości dwóch obliczonych kolumn, `bac_m` i `bac_f`.

Teraz można wykonać zapytanie złączające tabelę `products` z widokiem `product_alcohol_bac`, jak pokazałem na listingu 5.8. W ten sposób żądane dane pobieramy bezpośrednio i bez problemów.

Listing 5.8. *Pobieranie za pomocą widoku danych dotyczących stężenia alkoholu we krwi*

```
SQL> select
  2     p.id as p_id
  3   , p.name
  4   , pab.sales_volume as vol
  5   , pab.abv
  6   , pab.bac_m
  7   , pab.bac_f
  8  from products p
  9  join product_alcohol_bac pab
 10     on pab.product_id = p.id
 11  where p.group_id = 142
 12  order by p.id;
```

Otrzymujemy te same dane wyjściowe.

```
P_ID  NAME              VOL  ABV  BAC_M  BAC_F
----  ----------------  ---  ---  -----  -----
4040  Coalminers Sweat  330  8.5  0.041  0.067
4160  Reindeer Fuel     500  6    0.044  0.072
4280  Hoppy Crude Oil   330  7    0.034  0.055
```

Ta metoda pozwala wielokrotnie używać logiki przez wykonywanie zapytań SQL do widoku zamiast do tabeli. Mimo to logika będzie znajdowała się w pojedynczym miejscu — w definicji widoku.

Zamiast definiować funkcje w widoku, ten sam efekt można osiągnąć, gdy wynik wywołuje funkcję pakietu `formulas.bac()`. Jeżeli jednak funkcjonalność jest *ściśle* związana jedynie z określonym zapytaniem lub definicją widoku, wówczas lepsze będzie przechowywanie kodu w jednym miejscu i niezaśmiecanie słownika danych funkcjami składowymi, które i tak nigdy nie powinny być wywoływane poza określonym kodem SQL.

Podsumowanie

Wprawdzie tematem tej książki nie jest PL/SQL jako taki, ale możliwość jeszcze ściślejszej niż wcześniej integracji kodu PL/SQL i SQL jest funkcjonalnością, która powinna być znana programistom SQL. Dzięki informacjom zamieszczonym w rozdziale potrafisz:

- Ustalać, czy funkcja jest wywołana przede wszystkim z poziomu SQL. W takim przypadku korzystne będzie dodanie dyrektywy `PRAGMA UDF` do jej definicji.
- Osadzać funkcje „jednorazowe” w zapytaniach SQL z użyciem klauzuli `WITH`.
- Sprawdzać, czy konkretna funkcjonalność może działać lepiej po jej hermetyzacji w widoku za pomocą klauzuli `WITH` zamiast w tradycyjnych funkcjach składowanych.

Podczas codziennej pracy prawdopodobnie najczęściej będziesz korzystać z funkcji zawierających dyrektywy `PRAGMA UDF`. Mimo to technika oparta na klauzuli `WITH` również może być niezwykle użyteczna, zwłaszcza w sytuacjach, w których nie można instalować funkcji lub procedur składowanych.

Skorowidz

F

G

H

I

K

L

Ł

M

N

O

P

R

S

T

U

V

W

Z

SQL. Opanuj sztukę pisania naprawdę zaawansowanych zapytań!

Mimo upływu lat SQL jest bezkonkurencyjnym narzędziem do przetwarzania danych. Bazy danych Oracle wciąż imponują możliwościami. W ciągu ostatnich dekad bowiem obie te technologie były konsekwentnie unowocześniane i usprawniane. W efekcie nawet za pomocą jednego, choć niekiedy dość złożonego zapytania SQL można przeprowadzić operacje, które w innym języku wymagałyby napisania długich bloków kodu. Jednak do uzyskania spektakularnej wydajności i szybkości aplikacji niezbędne jest duże doświadczenie w posługiwaniu się zaawansowanymi konstrukcjami SQL.

To książka przeznaczona dla osób, które dobrze poznały podstawy języka SQL i chcą nabrać biegłości w praktycznym zastosowaniu jego zaawansowanych funkcji. Poszczególne zagadnienia zostały zaprezentowane poprzez stopniową rozbudowę i zwiększanie złożoności prostych zapytań SQL. Omówiono takie techniki jak korelacja widoku osadzonego, operacje na zbiorach, analiza dzienników zdarzeń, a także sposoby używania klauzul, między innymi MODEL czy MATCH_RECOGNIZE. Znalazło się tu mnóstwo przykładów kodu SQL, skonstruowanego tak, aby maksymalnie ułatwić zrozumienie prezentowanych treści. To pomoże Ci zdobyć umiejętności, dzięki którym wydajność i wygoda użytkowania Twoich aplikacji istotnie się zwiększą!

W książce:

- stosowanie zaawansowanych funkcji języka SQL w bazie danych Oracle
- stopniowe usprawnianie zapytań SQL
- przetwarzanie większej ilości danych za pomocą mniejszej liczby zapytań
- korzystanie z funkcji analitycznych
- dopasowywanie wzorca rekordu
- rekurencyjna faktoryzacja podzapytania

Kim Berg Hansen jest duńskim programistą baz danych. Od ponad 20 lat zajmuje się technologiami Oracle SQL i PL/SQL. Podczas pisania kodu zwraca baczną uwagę na jego efektywność i niskie wykorzystanie zasobów komputera, a także na uzyskanie najlepszych wrażeń użytkownika gotowej aplikacji. Posiada tytuły Oracle Certified Expert (OCE) w zakresie SQL i Oracle ACE Director.

ISBN 978-83-283-8733-1