SQL
dla analityków danych

Skutecznie analizuj dane, wyciągaj wartościowe wnioski i opanuj zaawansowany SQL na potrzeby praktycznych zastosowań

Wydanie IV

Jun Shan | Haibin Li | Matt Goldwasser
Upom Malik | Benjamin Johnston

Helion

packt

Tytuł oryginału: SQL for Data Analytics: Analyze data effectively, uncover insights and master advanced SQL for real-world applications, 4th Edition

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-289-3857-1

First published in the English language under the title
'SQL for Data Analytics - Fourth Edition – (9781836646259)'

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/sqlan4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

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

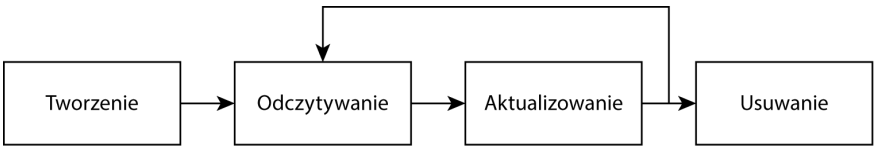
Printed in Poland.

Spis treści

Wyświetlanie danych z użyciem instrukcji SELECT

Rozdział 5

W poprzednich rozdziałach omówiliśmy podstawowy cykl operacji na danych. Potrafisz już tworzyć i usuwać tabele oraz wstawiać do nich dane. Wiesz także, jak zaimportować dane do relacyjnej bazy danych oraz jak je z niej wyeksportować. Wszystkie te operacje dotyczą początkowego i końcowego etapu cyklu operacji CRUD (rysunek 5.1).



Rysunek 5.1. Operacje CRUD

Ostatecznym celem zarządzania danymi jest wykorzystanie ich do generowania informacji na potrzeby analiz. Podstawowym krokiem w tym procesie jest odczyt zapisanych danych. W tym rozdziale zaczniesz się uczyć, jak wczytywać dane z istniejących tabel. W następnym rozdziale poznasz kolejne operacje związane z przekształcaniem danych oraz etap aktualizowania w cyklu życia danych.

W tym rozdziale omawiamy następujące zagadnienia:

- użycie wyrażeń `SELECT`:
  - aliasy wyrażeń,
  - klauzula `LIMIT`,
  - klauzula `ORDER BY`,
  - funkcje `DISTINCT` i `DISTINCT ON`;
- filtrowanie wyników kwerend:
  - klauzule `AND`, `OR` i `NOT`,
  - klauzule `IN` i `NOT IN`,
  - klauzule `IS NULL` i `IS NOT NULL`.

Po zapoznaniu się z tymi tematami będziesz wiedzieć, jak pobierać dane z bazy za pomocą SQL-a z wykorzystaniem warunków i filtrów.

Wymogi techniczne

Aby wykonać ćwiczenia z tego rozdziału, należy skonfigurować na komputerze serwer PostgreSQL i zaimportować bazę `sqlda` zgodnie z instrukcjami z rozdziału 1.

Stosowanie wyrażeń SELECT

Do tej pory korzystaliśmy z podstawowych kwerend `SELECT` o następującym wzorze:

```
SELECT * FROM <nazwa_tabeli>;
```

Taka kwerenda pobiera wszystkie wiersze z jednej tabeli i wyświetla wszystkie kolumny każdego wiersza. Można użyć tej kwerendy do sprawdzenia, jakie dane znajdują się w tabeli. Jednak bezrefleksyjne pobieranie wszystkich kolumn zazwyczaj nie jest konieczne i może prowadzić do poważnych problemów z wydajnością. W wielu sytuacjach można wskazać listę kolumn. W tym celu należy zastąpić znak * listą nazw kolumn, na przykład w następujący sposób:

```
SELECT product_id, model, base_msrp FROM products;
```

Ta instrukcja zwraca informacje o kolumnach `product_id`, `model` i `base_msrp` dla wszystkich produktów z tabeli. Możesz podać kolumny w dowolnej kolejności niezależnie od ich uporządkowania w definicji tabeli. PostgreSQL wyświetla kolumny zgodnie z ich układem w instrukcji `SELECT`:

```
 product_id |         model          | base_msrp
------------+------------------------+-----------
          1 | Lemon                  |    399.99
          2 | Lemon Limited Edition  |    799.99
          3 | Lemon                  |    499.99
          5 | Blade                  |    699.99
          7 | Bat                    |    599.99
          8 | Bat Limited Edition    |    699.99
         12 | Lemon Zester           |    349.99
          4 | Model Chi              | 115000.00
          6 | Model Sigma            |  65500.00
          9 | Model Epsilon          |  35000.00
         10 | Model Gamma            |  85750.00
         11 | Model Chi              |  95000.00
```

Warto zauważyć, że zapytanie `SELECT` jest stosowane do wszystkich wierszy w tabeli. Jest ono wykonywane dla każdego wiersza niezależnie od efektów uruchomienia go dla pozostałych rekordów. Wyjaśnia to częste zaskoczenie związane z uruchomieniem następującego polecenia SQL:

```
SELECT 1 FROM products;
```

Wielu początkujących programistów uważa, że to zapytanie zwróci jedną wartość 1. W rzeczywistości operacja `SELECT 1` zostanie zastosowana do wszystkich 12 wierszy z tabeli `products`. Wynik tej operacji, czyli `1`, pojawi się więc 12 razy. W efekcie zobaczysz 12 wierszy zawierających wartość 1:

```
?column?
----------
         1
         1
         1
         1
         1
         1
         1
         1
         1
         1
         1
         1
(12 wierszy)
```

Możesz pobierać nie tylko kolumny z tabel, ale także wyniki obliczeń wykonywanych na kolumnach (zarówno z użyciem stałej, jak i wartości z różnych kolumn). Takie obliczenia są zapisywane w formie **wyrażeń**. Na przykład następujące wyrażenie odejmuje 10-procentowy rabat od ceny `base_msrp` produktów (czyli mnoży tę cenę przez 0,9):

```
SELECT model, base_msrp * 0.9 FROM products;
```

Oto wynik wykonania tego kodu:

```
         model          |  ?column?
------------------------+------------
 Lemon                  |    359.991
 Lemon Limited Edition  |    719.991
 Lemon                  |    449.991
 Blade                  |    629.991
 Bat                    |    539.991
 Bat Limited Edition    |    629.991
 Lemon Zester           |    314.991
 Model Chi              | 103500.000
 Model Sigma            |  58950.000
 Model Epsilon          |  31500.000
 Model Gamma            |  77175.000
 Model Chi              |  85500.000
(12 rows)
```

Wyniki to pierwotne ceny pomnożone przez 0,9. Operacja ta jest stosowana do ceny w każdym wierszu.

Dane przedstawione powyżej to wartości wyrażenia, ale nagłówkiem kolumny jest `?column?`. Dzieje się tak dlatego, że wyrażeniu nie przypisano nazwy. W następnym punkcie zobaczysz, jak przypisać nazwę do takiego wyrażenia.

Aliasy wyrażeń

W instrukcji `SELECT` można użyć słowa kluczowego `AS`, by przypisać do wyrażenia zrozumiałą nazwę. Na przykład w poniższej instrukcji do wyników obliczeń przypisywana jest nazwa `Discounted_Price`:

```
SELECT model, base_msrp * 0.9 AS Discounted_Price
FROM products;
```

Oto kilka pierwszych zwróconych wierszy:

```
         model          | discounted_price
------------------------+------------------
 Lemon                  |          359.991
 Lemon Limited Edition  |          719.991
 Lemon                  |          449.991
 Blade                  |          629.991
```

Alias można przypisać do dowolnego wyrażenia. Dotyczy to zarówno obliczeń na kolumnach, jak i pojedynczych kolumn oraz stałych. Ponadto można pominąć słowo kluczowe `AS`. Oto przykładowa instrukcja i kilka pierwszych zwróconych wierszy:

```
SELECT
  Model Product_Name,
  0.9 Discount
  base_msrp * 0.9 Discounted_Price
FROM products;
```

Wyniki wyświetlane są w następujący sposób:

```
     product_name       | discount | discounted_price
------------------------+----------+------------------
 Lemon                  |      0.9 |          359.991
 Lemon Limited Edition  |      0.9 |          719.991
 Lemon                  |      0.9 |          449.991
```

Teraz, gdy wiesz już, jak określać wyświetlane kolumny, kolejnym krokiem jest zarządzanie listą wynikowych wierszy. Zaczniesz od ograniczenia liczby wyników.

Klauzula LIMIT

Większość tabel w bazach SQL-owych jest dość duża, dlatego nie trzeba zwracać wszystkich wierszy. Czasem potrzebnych jest tylko kilka pierwszych wierszy. W takim scenariuszu przydatne jest słowo kluczowe `LIMIT`. W poprzednich rozdziałach kilkakrotnie używaliśmy już następującej kwerendy:

```
SELECT * FROM products LIMIT 5;
```

Jeśli programista nie zna danej tabeli lub kwerendy, często może się obawiać, że instrukcja `SELECT` nieoczekiwanie zwróci wiele wierszy, co może wymagać dużo czasu i wielu zasobów maszyny. Dlatego zgodnie z ogólną regułą zwykle warto używać słowa kluczowego `LIMIT` w tabelach i kwerendach, z których wcześniej nie korzystałeś.

Warto jednak zauważyć, że słowo kluczowe `LIMIT` nie zmniejsza obciążenia związanego z wykonywaniem kwerendy po stronie serwera. Ogranicza jedynie liczbę wierszy przesyłanych z systemu zarządzania bazą danych do narzędzia klienckiego. SZBD nadal przetwarza cały zbiór danych. W związku z tym technika ta nie pomaga w optymalizacji wydajności serwera. Redukcja liczby wierszy jest odczuwalna jedynie po stronie klienta.

Kolejną ważną kwestią jest sposób porządkowania wierszy. Jak wspomnieliśmy wcześniej, operacje relacyjne nie określają kolejności danych. Jeśli więc chcesz, aby wyniki były posortowane według Twoich instrukcji, musisz bezpośrednio nakazać SQL-owi, by to zrobił. Jest to temat następnego punktu.

Klauzula ORDER BY

Kwerendy SQL-owe *nie* porządkują wierszy. Jeśli kwerenda nie otrzyma konkretnych instrukcji, uporządkuje wiersze tak, jak zostały znalezione przez bazę danych. W wielu scenariuszach jest to akceptowalne. Jednak często pożądane jest zwracanie wierszy w określonym porządku.

Wyobraź sobie, że chcesz pobrać wszystkie produkty uporządkowane od najstarszych do najnowszych według daty rozpoczęcia produkcji. W SQL-u umożliwia to klauzula `ORDER BY`:

```
SELECT model, production_start_date
FROM products
ORDER BY production_start_date
LIMIT 5;
```

Jak ilustrują to poniższe dane wyjściowe, produkty są uporządkowane według pola `production_start_date`:

```
         model          | production_start_date
------------------------+-----------------------
 Lemon                  | 2015-10-28 00:00:00
 Lemon Limited Edition  | 2016-08-30 00:00:00
 Lemon                  | 2018-12-27 00:00:00
 Blade                  | 2020-02-17 00:00:00
 Model Chi              | 2020-02-17 00:00:00
(5 rows)
```

Kolumna podana w klauzuli `ORDER BY` musi pochodzić z używanej tabeli, ale nie jest konieczne, by była używana w klauzuli `SELECT`. Jeśli kolejność sortowania nie jest bezpośrednio określona, zwracane wiersze będą uporządkowane rosnąco. Oznacza to, że wiersze są porządkowane od najmniejszej do największej wartości z wybranej kolumny (lub ze zbioru kolumn). W przypadku tekstu uwzględniana jest kolejność alfabetyczna. Aby jawnie zażądać kolejności rosnącej, użyj słowa kluczowego `ASC`. W poprzedniej kwerendzie można to zrobić tak:

```
SELECT model FROM products
ORDER BY production_start_date ASC
LIMIT 5;
```

Ta kwerenda SQL-owa zwróci te same wyniki w tej samej kolejności co wcześniejszy kod.

Jeśli chcesz pobrać dane w porządku malejącym, użyj słowa kluczowego `DESC`. Jeżeli zamierzasz wczytać modele uporządkowane od najnowszych do najstarszych, zastosuj następujący kod:

```
SELECT model FROM products
ORDER BY production_start_date DESC
LIMIT 5;
```

Wyniki będą posortowane malejąco według pola `production_start_date` (na początku znajdzie się najnowszy produkt).

Ponadto zamiast podawać nazwę kolumny, według której chcesz sortować dane, możesz podać numer tej kolumny z klauzuli `SELECT` kwerendy. Załóżmy, że chcesz zwrócić wszystkie modele z tabeli `products` uporządkowane według identyfikatorów produktów (`product_id`). Możesz zapisać kwerendę tak:

```
SELECT product_id, model FROM products
ORDER BY product_id;
```

Ponieważ jednak `product_id` to pierwsza kolumna w instrukcji `SELECT`, możesz też zastosować taki zapis:

```
SELECT product_id, model FROM products
ORDER BY 1;
```

Ta kwerenda SQL-owa zwróci te same wyniki co wcześniejsza.

Możesz także sortować dane według kilku kolumn. W tym celu należy podać po klauzuli `ORDER BY` dodatkowe kolumny rozdzielone przecinkami (opcjonalnie z modyfikatorami `ASC` lub `DESC` przy każdej z nich). Załóżmy, że chcesz uporządkować wszystkie wiersze tabeli najpierw według roku produkcji modelu (od najnowszych do najstarszych), a następnie według **sugerowanej ceny detalicznej** (pole `base_msrp`) od najniższej do najwyższej. Możesz zapisać kod tak:

```
SELECT * FROM products
ORDER BY year DESC, base_msrp ASC;
```

Możesz również użyć pozycji kolumn zamiast nazw:

```
SELECT * FROM products
ORDER BY 3 DESC, 5 ASC;
```

Wynik będzie taki sam.

Za pomocą klauzul `LIMIT` i `ORDER BY` można wczytywać dane w ich pierwotnej postaci. Zdarza się jednak, że dane przed wyświetleniem trzeba zmodyfikować. Pierwszy z takich scenariuszy dotyczy pobierania z tabeli unikatowych wartości, co omawiamy w następnym punkcie.

Funkcje DISTINCT i DISTINCT ON

W pokazanych wcześniej wynikach nazwy niektórych produktów się powtarzają. W trakcie analizowania zbioru danych często przydatne jest ustalenie unikatowych wartości w kolumnie lub w grupie kolumn. Jest to główne zastosowanie słowa kluczowego `DISTINCT`.

W tabeli `products` znajdują się dwa modele o nazwie `Lemon`. Jeśli chcesz poznać wszystkie unikatowe modele przechowywane w tej tabeli, możesz użyć następującej kwerendy:

```
SELECT DISTINCT model FROM products;
```

Otrzymasz następujące wyniki:

```
        model
----------------------
 Model Epsilon
```

```
 Model Chi
 Model Gamma
 Lemon
 Lemon Limited Edition
 Lemon Zester
 Bat Limited Edition
 Bat
 Model Sigma
 Blade
(10 rows)
```

W tych wynikach nazwa modelu `Lemon` występuje tylko raz. Możesz zastosować te funkcje do wielu kolumn, aby uzyskać wszystkie unikatowe kombinacje wartości z tych kolumn. Na przykład aby znaleźć wszystkie unikatowe lata i rodzaje produktów wprowadzonych na rynek w tych latach, możesz zastosować taki kod:

```
SELECT DISTINCT year, product_type FROM products;
```

Powinny pojawić się następujące dane wyjściowe:

```
 year | product_type
------+--------------
 2025 | automobile
 2020 | automobile
 2016 | scooter
 2025 | scooter
 2023 | automobile
 2019 | scooter
 2021 | automobile
 2020 | scooter
 2022 | scooter
 2017 | scooter
 2023 | scooter
(11 rows)
```

Z `DISTINCT` powiązane jest słowo kluczowe `DISTINCT ON`, które pozwala zagwarantować zwrócenie tylko po jednym wierszu z każdą wartością podanego wyrażenia. Dla tych pojedynczych wierszy można zwrócić określoną pierwszą wartość z pozostałych kolumn. Oto ogólna składnia kwerend `DISTINCT ON`:

```
SELECT DISTINCT ON (unikatowa_kolumna)
  unikatowa_kolumna,
  kolumna_1,
  kolumna_2,
FROM tabela
ORDER BY kolumna_sortowania;
```

Tu `unikatowa_kolumna` to kolumna (lub zbiór kolumn), z której wartości mają być unikatowe, kolumny `kolumna_1`, `kolumna_2` itd. to kolumny, które mają znaleźć się w wynikach kwerendy, a `kolumna_sortowania` to kolumna służąca do określenia pierwszego wiersza zwróconego przez kwerendę `DISTINCT ON`, jeśli w kilku wierszach w kolumnie `unikatowa_kolumna` znajduje się ta sama wartość. Jako `kolumna_sortowania` pierwsza powinna być wymieniona `unikatowa_kolumna`. Jeśli kwerenda nie zawiera klauzuli `ORDER BY`, pierwszy wiersz zostanie wybrany losowo.

Załóżmy, że chcesz pobrać listę salonów z najdłużej pracującym sprzedawcą z każdego z nich. Potrzebna kwerenda wygląda następująco:

```
SELECT DISTINCT ON (dealership_id)
  dealership_id, first_name, last_name
FROM salespeople
ORDER BY dealership_id , hire_date;
```

W tym przykładzie wiersze wyjściowe są uporządkowane według kolumn `dealership_id` i `hire_date`. Dla każdego salonu sprzedawcy są sortowani według kolumny `hire_date`, a jako pierwszy umieszczany jest wiersz z najwcześniejszą datą zatrudnienia, reprezentujący osobę najdłużej pracującą w danym salonie. Klauzula `DISTINCT ON` powoduje wybranie pierwszego wiersza, czyli najdłużej zatrudnionego sprzedawcy, dla każdej wartości z kolumny `dealership_id`. Tak więc dla każdego salonu otrzymasz imię i nazwisko najdłużej pracującej osoby. Kod powinien zwrócić następujące dane wyjściowe (pokazanych jest kilka pierwszych wierszy):

```
 dealership_id | first_name | last_name
---------------+------------+-----------
             1 | Dukie      | Oxteby
             2 | Adrianne   | Otham
             3 | Benn       | Jakobssen
             4 | Benji      | Balsillie
             5 | Hewie      | Hutcheon
```

Ten kod gwarantuje, że w każdym wierszu tabeli występuje unikatowa wartość `dealership_id`. Jeśli w jednym salonie pracuje wielu sprzedawców, kwerenda zwróci osobę, która została zatrudniona w danym salonie najdawniej.

Dotychczas w tym rozdziale koncentrowaliśmy się na pobieraniu danych bez stosowania kryteriów filtrowania. W następnym podrozdziale zobaczysz, jak odfiltrować niepotrzebne wiersze.

Filtrowanie wyników kwerend

W większości sytuacji konieczna jest analiza tylko części danych, spełniającej określone kryteria. Nie trzeba uwzględniać całej tabeli. Dlatego sensowne jest stosowanie warunków filtrowania do instrukcji `SELECT`, by pobrać tylko potrzebne dane. Klauzula `WHERE` pozwala dodać warunek ograniczający ilość zwracanych danych. Wszystkie wiersze zwracane przez kwerendę `SELECT` z klauzulą `WHERE` spełniają warunek z tej klauzuli. W kwerendzie `SELECT` klauzula `WHERE` zwykle jest umieszczana po klauzuli `FROM`.

Warunkiem w klauzuli `WHERE` zazwyczaj jest wyrażenie logiczne, które dla każdego wiersza przyjmuje wynik `true` lub `false`. Gdy używane są kolumny liczbowe, tekstowe lub z datą i czasem, w wyrażeniu logicznym można stosować operatory równości (`=`), większości (`>`), większe lub równe (`>=`), mniejszości (`<`) albo mniejsze lub równe (`<=`), aby porównywać wartość z kolumny z podaną wartością. Można też użyć operatora nierówności. Istnieją dwa takie operatory: `<>` i `!=`. Możesz stosować dowolny z nich.

Oto ilustrujący to przykład. Załóżmy, że chcesz sprawdzić nazwy modeli z 2017 roku z przykładowego zbioru danych. W tym celu możesz napisać następującą kwerendę:

```
SELECT model FROM products WHERE year=2017;
```

Oto dane wyjściowe tego kodu w SQL-u:

```
         model
----------------------
 Lemon Limited Edition
(1 row)
```

Klauzula `WHERE` może również posłużyć do ograniczenia liczby zwracanych wierszy, jeśli potrzebujesz tylko części zbioru danych. Pod tym względem działa podobnie do klauzuli `LIMIT`. Jednak klauzula `WHERE` stosuje do zbioru danych warunki logiczne, natomiast klauzula `LIMIT` powoduje pobranie określonej liczby pierwszych zwróconych wierszy. Dlatego klauzula `WHERE` jest wykonywana po stronie serwera bazodanowego i zmniejsza ilość danych przetwarzanych przez bazę. Zazwyczaj pomaga to zwiększyć wydajność.

Za pomocą klauzuli `WHERE` udało się przefiltrować produkty spełniające określony warunek. Jeśli chcesz otrzymać listę produktów sprzed 2017 roku, wystarczy zmodyfikować klauzulę `WHERE` na postać `year<2017`. Z kolei by uzyskać produkty, których produkcja rozpoczęła się w dowolnym roku innym niż 2017, można zastosować klauzulę `WHERE` z warunkiem `year!=2017`. Co jednak zrobić, jeśli zechcesz przefiltrować wiersze na podstawie wielu kryteriów jednocześnie? Możliwe też, że chcesz otrzymać wiersze spełniające jeden z dwóch (lub więcej) warunków. Możesz uzyskać taki efekt przez dodanie do kwerendy klauzuli `AND` lub `OR`.

Klauzule AND i OR

Aby zastosować kilka warunków do klauzuli `SELECT`, można połączyć zestaw wyrażeń `WHERE` za pomocą klauzul `AND` i `OR`. Klauzula `AND` pozwala pobrać tylko te wiersze, które spełniają co najmniej dwa warunki. Z kolei klauzula `OR` służy do pobierania wierszy, które pasują do co najmniej jednego warunku ze zbioru obejmującego przynajmniej dwa warunki.

Wyobraź sobie, że chcesz zwrócić modele, które nie tylko zostały zbudowane w 2017 roku, ale dodatkowo mają sugerowaną cenę detaliczną producenta niższą niż 1000 dolarów. Możesz napisać następującą kwerendę:

```
SELECT model, year, base_msrp
FROM products
WHERE year=2017 AND msrp<=1000;
```

Oto wyniki:

```
         model          | year | base_msrp
------------------------+------+-----------
 Lemon Limited Edition | 2017 |    799.99
(1 row)
```

Widać tu, że produkty pochodzą z roku (`year`) 2017, a sugerowana cena (`base_msrp`) jest niższa niż 1000 dolarów. Właśnie o takie dane chodziło.

Ponadto możesz używać klauzuli OR do wyszukiwania wierszy, które spełniają przynajmniej jeden warunek z ich listy. Teraz załóżmy, że chcesz zwrócić dowolne modele, które zostały wprowadzone na rynek w 2017 roku lub których sugerowana cena jest niższa niż 1000 dolarów. Umożliwia to poniższa kwerenda:

```
SELECT model, year, base_msrp
FROM products
WHERE year=2017 OR msrp<=1000;
```

Oto wyniki:

```
         model          | year | base_msrp
------------------------+------+-----------
 Lemon                  | 2016 |    399.99
 Lemon Limited Edition  | 2017 |    799.99
 Lemon                  | 2019 |    499.99
 Blade                  | 2020 |    699.99
 Bat                    | 2022 |    599.99
 Bat Limited Edition    | 2023 |    699.99
 Lemon Zester           | 2025 |    349.99
(7 rows)
```

Wiesz już, że istnieje tylko jeden produkt, Lemon Limited Edition, z roku (year) 2017. Pozostałe produkty w tym przykładzie mają sugerowaną cenę poniżej 1000 dolarów. Widoczny jest połączony zbiór danych dla warunków year=2017 i msrp<=1000. Tak właśnie działa operator OR.

Gdy używasz więcej niż jednej klauzuli AND lub OR, dodaj nawiasy, aby odpowiednio oddzielić i rozmieścić wyrażenia logiczne. To gwarantuje, że kwerenda będzie działać zgodnie z oczekiwaniami i że będzie czytelna. Jeśli na przykład chcesz pobrać wszystkie produkty z lat od 2019 do 2021, a także produkty typu 'scooter', możesz użyć następującej kwerendy:

```
SELECT *
FROM products
WHERE year>2019 AND year<2021
OR product_type='scooter';
```

Wyniki obejmują wszystkie skutery, a dodatkowo także samochody z lat od 2019 do 2021.

Jednak aby klauzula WHERE była bardziej czytelna, lepiej jest użyć następującego zapisu:

```
SELECT *
FROM products
WHERE (year>2019 AND year<2021)
OR product_type='scooter';
```

Otrzymasz te same wyniki co wcześniej, jednak logika tego kodu w SQL-u jest bardziej zrozumiała.

PostgreSQL udostępnia ponadto klauzulę NOT. Służy ona do określania wierszy, które nie są zgodne z podanym warunkiem. Na przykład poniższy kod zwraca wiersze, które nie spełniają warunku OR podanego w nawiasie:

```
SELECT model, year, base_msrp
FROM products
WHERE NOT (year=2017 OR base_msrp<=1000);
```

Wyniki obejmują wszystkie modele z lat innych niż 2017 i mające sugerowaną cenę powyżej 1000 dolarów:

```
     model      | year | base_msrp
----------------+------+-----------
 Model Chi      | 2020 | 115000.00
 Model Sigma    | 2021 |  65500.00
 Model Epsilon  | 2023 |  35000.00
 Model Gamma    | 2023 |  85750.00
 Model Chi      | 2025 |  95000.00
(5 rows)
```

Dzięki połączeniu operatorów porównania, takich jak > i <, oraz operatorów logicznych, takich jak `AND` i `OR`, można wykonać większość operacji filtrowania wartości liczbowych i alfabetycznych. Jednak istnieją również specjalne kryteria filtrowania, służące na przykład do wybierania elementów z listy lub wykonywania operacji z uwzględnieniem wartości `NULL`. Takie kryteria omawiamy w następnym podrozdziale.

Klauzule IN i NOT IN

Skoro umiesz już pisać kwerendy obejmujące wiele warunków, możesz doprecyzować kryteria i pobrać wiersze, które zawierają (bądź nie) jedną lub więcej określonych wartości we wskazanych kolumnach. W takim scenariuszu przydatne są klauzule `IN` i `NOT IN`.

Załóżmy, że interesują Cię wszystkie modele z lat `2017`, `2019` i `2022`. Możesz zapisać taką kwerendę tak:

```
SELECT model, year
FROM products
WHERE year = 2017
OR year = 2019
OR year = 2022;
```

W wynikach widoczne są trzy modele z podanych lat:

```
         model          | year
------------------------+------
 Lemon Limited Edition | 2017
 Lemon                 | 2018
 Bat                   | 2022
(3 rows)
```

Jednak kod z wieloma klauzulami `OR` jest długi i żmudny w pisaniu. Za pomocą klauzuli `IN` możesz zapisać go tak:

```
SELECT model, year
FROM products
WHERE year IN (2017, 2019, 2022);
```

Tę wersję znacznie łatwiej jest zapisać, a także zrozumieć. Zwraca ona te same wyniki co wcześniejszy kod.

Możesz też zastosować klauzulę NOT IN, aby zwrócić wszystkie wartości poza podanymi na liście. Jeśli chcesz otrzymać wszystkie modele, które nie zostały wyprodukowane w latach 2017, 2019 lub 2022, możesz użyć tej kwerendy:

```
SELECT model, year
FROM products
WHERE year NOT IN (2017, 2019, 2022);
```

Teraz wyświetlane są produkty z lat innych niż trzy wymienione w pokazanej instrukcji SQL-a.

Dotychczas wszystkie kryteria filtrowania dotyczyły rzeczywistych wartości, na przykład równych lub większych od podanych. Co jednak zrobić w sytuacji, gdy chcesz wykrywać brak wartości? Często w kolumnie brakuje niektórych danych. Może to wynikać z wielu powodów. Możliwe, że dane nie zostały pobrane lub były niedostępne w momencie ich wprowadzania. Możliwe też, że brak wartości reprezentuje określony stan wiersza i stanowi cenną informację. Niezależnie od powodu analityk często chce znaleźć wiersze, w których w określonej kolumnie brakuje wartości. W SQL-u brak wartości nieraz jest reprezentowany za pomocą NULL. Na przykład w tabeli products wartość NULL w kolumnie production_end_date oznacza, że produkt wciąż jest wytwarzany. Dlatego wykrywanie wartości NULL jest bardzo istotne w operacjach biznesowych. SQL udostępnia specjalną funkcję do sprawdzenia występowania wartości NULL, czyli braku rzeczywistych danych.

Klauzule IS NULL i IS NOT NULL

Jeśli chcesz wyświetlić wszystkie nadal produkowane towary, musisz wykryć wiersze, w których kolumna production_end_date zawiera wartość NULL. Możesz w tym celu użyć następującej kwerendy:

```
SELECT model, production_end_date
FROM products
WHERE production_end_date IS NULL;
```

Oto dane wyjściowe tej kwerendy:

```
        model        | production_end_date
---------------------+---------------------
 Bat                 |
 Bat Limited Edition |
 Lemon Zester        |
 Model Epsilon       |
 Model Gamma         |
 Model Chi           |
(6 rows)
```

Ten kod wyświetla wszystkie produkty z wartością NULL w kolumnie production_end_date.

Natomiast jeśli interesują Cię tylko produkty, które już nie są wytwarzane, możesz użyć klauzuli IS NOT NULL, tak jak w tej kwerendzie:

```
SELECT *
FROM products
WHERE production_end_date IS NOT NULL;
```

Zwróć uwagę, że wartość NULL nie jest równa 0 ani pustemu łańcuchowi znaków. Można ją wykryć wyłącznie przy użyciu operatorów IS NULL lub IS NOT NULL. Na przykład następujące klauzule WHERE nie zwrócą wierszy zawierających wartości NULL:

- base_msrp = 0,
- name = ''.

Nawet porównanie z NULL nie spowoduje zwrócenia wierszy z nazwami o wartości NULL:

```
name = NULL
```

Żadne z tych porównań nie wykrywa wierszy z wartościami NULL. Dzieje się tak dlatego, że każde porównanie logiczne obejmujące wartość NULL daje w wyniku NULL, a nie True lub False. Dlatego gdy wartością kolumny jest NULL, wszystkie wymienione porównania zwrócą NULL.

Wiesz już, że gdy system zarządzania bazą danych wykonuje instrukcję SELECT, jest ona uruchamiana dla każdego wiersza. Dotyczy to również sprawdzania, czy dany wiersz spełnia warunek podany w klauzuli WHERE. Jeśli taki warunek dla danego wiersza ma wartość False lub NULL, wiersz nie jest pobierany do końcowego zbioru wyników. Dlatego wymienione porównania nie wykrywają poprawnie wartości NULL. Wartość NULL powoduje, że takie porównania zwracają NULL i wiersze nie są pobierane. Do operacji związanych z wartościami NULL należy używać operatora IS NULL lub IS NOT NULL.

W następnym ćwiczeniu wykorzystasz zdobytą w tym rozdziale wiedzę do wczytania danych z bazy. Zastosujesz poznane formatowanie i warunki.

Ćwiczenie 5.1. Wczytywanie danych z bazy

Omówiliśmy już najważniejsze elementy klauzuli SELECT. Teraz wykorzystasz zdobyte umiejętności w ćwiczeniu.

Zaczynasz pracę analityka danych w firmie ZoomZoom. Właśnie przyznano Ci dostęp do produkcyjnej bazy danych sqlda. W rozdziale 1. przedstawiliśmy listę tabel z tej bazy i omówiliśmy ich przeznaczenie. Teraz wykorzystasz umiejętności z zakresu SQL-a, aby lepiej zrozumieć dane znajdujące się w tych tabelach. Na początek zbadasz niewielką próbkę danych, aby zidentyfikować podstawowe cechy zbioru danych. Następnie użyjesz klauzuli WHERE do odfiltrowania niepotrzebnych danych, tak aby analizy były bardziej specyficzne. Zastosujesz serię filtrów, aby przyjrzeć się danym z różnych perspektyw. Ćwiczenie obejmuje wszystkie te zagadnienia.

Oto kroki niezbędne do wykonania tego ćwiczenia:

1. Otwórz aplikację psql i nawiąż połączenie z bazą danych sqlda.
2. Zacznij od tabeli customers. Najpierw przyjrzyj się pierwszym trzem wierszom, aby ustalić, jakie kolumny zawiera ta tabela:

   ```
   SELECT * FROM customers LIMIT 3;
   ```

   Wyniki wyglądają następująco:

   ```
   customer_id | title | first_name | last_name | suffix
   |          email          | gender |   ip_address    | phone |
   ```

```
street_address | city | state | postal_code | latitude | longitude
|     date_added
-------------+-------+------------+-----------+--------+------------
-------------+--------+----------------+-------+---------------+-
-----+-------+-------------+----------+-----------+----------------
-----
         716 |       | Jarred     | Bester    |        | jbesterjv@
nih.gov       | M      | 216.51.110.28   |       |                |
      |       |             |          |           | 2021-09-19
00:00:00
         644 |       | Aeriell    | Zanicchi  |        |
azanicchihv@a8.net      | F      | 154.245.228.134 |       |
               |      |       |            |          |
           | 2019-04-03 00:00:00
         709 |       | Chane      |  McGrory   |        |
cmcgroryjo@gravatar.com | M      |  182.125.205.96  |       |
               |      |       |            |          |
           | 2022-07-02 00:00:00
(3 rows)
```

3. Pierwszym elementem, który zwraca uwagę, jest kolumna `date_added`. Możesz wyświetlić trzech najnowszych klientów za pomocą następującej kwerendy:

```
SELECT first_name, last_name, date_added
FROM customers
ORDER BY date_added DESC
LIMIT 3;
```

Ten kod porządkuje wyniki na podstawie pola `date_added`. Dane są porządkowane malejąco (`DESC`), ponieważ celem jest wyświetlenie trzech ostatnich („największych") dat. Oto wyniki:

```
 first_name | last_name |     date_added
------------+-----------+---------------------
 Patrizius  | Feetham   | 2025-02-23 00:00:00
 John       | Manclark  | 2025-02-23 00:00:00
 Athena     | Rickerby  | 2025-02-23 00:00:00
(3 rows)
```

4. Zawsze gdy zaczynasz pracę z nowymi zbiorami danych, należy sprawdzić jakość danych. Czy są one wiarygodne? Możesz na przykład posortować dane klientów według kolumny `date_added`, ale czy zawiera ona poprawne wartości? A może obejmuje wartości `NULL`, które mogą wskazywać na błędy w procesie wprowadzania danych? Aby odpowiedzieć na to pytanie, możesz uruchomić następującą kwerendę:

```
SELECT first_name, last_name, date_added
FROM customers
WHERE date_added IS NULL
LIMIT 3;
```

Wynik wskazuje, że w kolumnie `date_added` nie brakuje żadnych danych:

```
 first_name | last_name | date_added
------------+-----------+------------
(0 rows)
```

Należy sprawdzać jakość wszystkich otrzymywanych zbiorów danych, jednak kontrolowane aspekty mogą być różne. Na przykład w kolumnie `product_end_date` w tabeli `products` występuje wartość `NULL`. Oznacza ona jednak, że produkcja danego pojazdu nadal trwa. W związku z tym wartość `NULL` w tej kolumnie nie wskazuje na żadne problemy z jakością danych.

5. Kolejnym interesującym aspektem jest rozmieszczenie geograficzne klientów. W tym przypadku możesz zacząć od ogólnego ujęcia przez analizę stanów, w których mieszkają klienci. Można się spodziewać, że z każdego stanu będzie pochodziło wiele osób. Dlatego warto użyć słowa kluczowego `DISTINCT`, aby usunąć powtarzające się nazwy stanów:

   ```
   SELECT DISTINCT state FROM customers;
   ```

   Oto kilka pierwszych wierszy wyników:

   ```
   state
   -------
    KS

    CA
    NH
    OR
   ```

   Wcześniej wspomnieliśmy, że jeśli nie ma klauzuli `ORDER BY`, SQL nie wymusza sortowania. W związku z tym pierwszych kilka wierszy wyświetlonych na ekranie może różnić się od tych pokazanych w tym miejscu. Mimo to wyniki powinny wskazywać, że w tabeli znajduje się 50 stanów i jedna wartość `NULL`.

6. Interesuje Cię analiza zachowań klientów w zależności od lokalizacji geograficznej i czasu, dlatego chcesz uruchamiać zapytania z jednoczesnymi ograniczeniami dotyczącymi miejsca i dat. Można to osiągnąć za pomocą operatorów `AND` i `OR`. Na przykład następująca kwerenda zwróci trzech klientów mieszkających w Kalifornii lub na Florydzie, którzy zarejestrowali się po 01.01.2025:

   ```
   SELECT first_name, last_name, state, date_added
   FROM customers
   WHERE state IN ('CA', 'FL')
   AND date_added >= '01/01/2025'
   LIMIT 3;
   ```

   Ta kwerenda zwraca następujące wyniki:

   ```
    first_name | last_name | state |     date_added
   ------------+-----------+-------+---------------------
    Ferrell    | Beese     | CA    | 2025-01-07 00:00:00
    Cecile     | Metzig    | FL    | 2025-01-18 00:00:00
    Randee     | West      | CA    | 2025-01-11 00:00:00
   (3 rows)
   ```

W tym ćwiczeniu należało wykorzystać zdobyte w niniejszym rozdziale umiejętności do przeprowadzenia wstępnej analizy tabeli `customers`. W rzeczywistym systemie będziesz wykonywać podobne operacje na wszystkich tabelach i kolumnach. Zrozumienie danych jest pierwszym krokiem w analizie danych, a instrukcja `SELECT` może być w tym bardzo pomocna. Niektóre operacje wypróbujesz w kolejnym zadaniu.

Zadanie 5.

W tym rozdziale omówiliśmy różne aspekty instrukcji `SELECT`. Teraz wykorzystaj nową wiedzę w praktyce i wykonaj następujące zadania:

1. Napisz kwerendę pobierającą wszystkich sprzedawców zatrudnionych w latach 2024 i 2025, którzy nie zostali zwolnieni (wartość `hire_date` jest równa lub późniejsza niż 2024-01-01, a `termination_date` ma wartość `NULL`). Dane posortuj według kolumny `hire_date` malejąco (od najnowszych).
2. Napisz kwerendę, która pobiera imiona i nazwiska oraz adresy e-mail klientów firmy ZoomZoom z miasta Nowy Jork w stanie Nowy Jork. Wyniki powinny być posortowane alfabetycznie (najpierw według nazwisk, a następnie według imion).
3. Napisz kwerendę, która zwraca wszystkich klientów posiadających numer telefonu. Wyniki posortuj według daty dodania klienta do bazy danych.

Podsumowanie

W tym rozdziale skupiliśmy się na wyświetlaniu danych za pomocą instrukcji `SELECT`. Zaczęliśmy od wprowadzenia do wczytywania danych z istniejących tabel przy użyciu najprostszej formy instrukcji `SELECT`. Następnie wyjaśniliśmy, jak określać listę kolumn, nadawać aliasy, ograniczać liczbę wyników, sortować wyniki oraz pobierać unikatowe wartości. Wprowadziliśmy klauzulę `WHERE`, która służy do filtrowania wyników zapytań na podstawie operatorów porównania (`=`, `>`, `>=`, `<`, `<=`, `<>`, `!=`) oraz operatorów logicznych (`AND`, `OR`, `NOT`), a także `IN`, `NOT IN` oraz `IS NULL` i `IS NOT NULL`. Omówiliśmy również logikę kwerend związaną z obsługą wartości `NULL`. Dzięki tym umiejętnościom będziesz w stanie pisać kwerendy pobierające na podstawie odpowiednich warunków określone informacje z dowolnej tabeli, aby zapoznać się z danymi.

Często trzeba przekształcić dane, aby było łatwiej je zrozumieć. Na przykład możesz chcieć ustalić bieżącą datę lub przefiltrować pola tekstowe na podstawie kombinacji słów. Wymaga to dalszych działań związanych z transformacją danych. Ponadto konieczne może być zaktualizowanie istniejących danych z użyciem preferowanych wartości, a czasami nawet zmodyfikowanie struktury tabeli. Wszystkie te zagadnienia omawiamy w następnym rozdziale.

Od podstaw SQL-a do zaawansowanej analityki danych z PostgreSQL

SQL pozostaje fundamentalnym narzędziem w nowoczesnej analityce danych, a jego znajomość należy do najbardziej pożądanych na rynku IT. W erze big data i podejmowania decyzji opartych na danych ważna jest umiejętność efektywnego wydobywania informacji z relacyjnych baz danych. Czwarte wydanie tej książki, zaktualizowane o najnowsze funkcje PostgreSQL i dostosowane do współczesnych przepływów pracy, łączy tradycyjne podejście SQL-owe z nowoczesnymi narzędziami, jak Python, i technikami uczenia maszynowego.

Książka prowadzi czytelnika przez kompletną ścieżkę – od podstaw tworzenia i zarządzania bazami danych, przez zaawansowane techniki agregacji i funkcje okna, aż po analizy statystyczne i przetwarzanie złożonych typów danych. Autorzy kładą nacisk na praktyczne zastosowania, prezentując nie tylko składnię SQL-a, ale przede wszystkim kontekst biznesowy i rzeczywiste scenariusze analityczne. Każdy rozdział zawiera praktyczne ćwiczenia i studia przypadku, które pozwalają natychmiast zastosować zdobytą wiedzę.

Najważniejsze zagadnienia:

- Tworzenie i zarządzanie strukturami baz danych PostgreSQL z wykorzystaniem operacji CRUD
- Zaawansowane techniki pobierania danych: złączenia, podkwerendy, widoki i wyrażenia WITH
- Analiza statystyczna i testowanie hipotez bezpośrednio w SQL-u
- Przetwarzanie JSON, tablic, danych geoprzestrzennych i szeregów czasowych
- Optymalizacja wydajności SQL-a przez indeksy i plany wykonywania kwerend
- Integracja SQL-a z Pythonem do automatyzacji procesów analitycznych

Jun Shan jest głównym doradcą do spraw rozwiązań chmurowych z ponad 20-letnim doświadczeniem w zarządzaniu danymi. Tworzył rozwiązania dla Amazon i Bank of America. **Haibin Li** ma 10-letnie doświadczenie w danologii w branży ubezpieczeniowej jako lider zespołu modelowania predykcyjnego. **Matt Goldwasser** jest wiceprezesem do spraw AI i danologii w T. Rowe Price, specjalizuje się w MLOps i we wdrażaniu rozwiązań AI na dużą skalę. **Upom Malik** jest danologiem i analitykiem z ponad ośmioletnim doświadczeniem. **Benjamin Johnston** jest starszym danologiem w jednej z czołowych firm z branży medyczno-technologicznej. Ma ponad 10 lat doświadczenia w zakresie projektowania i rozwoju urządzeń medycznych.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-3857-1
9 788328 938571
Cena: 79,00 zł
packt