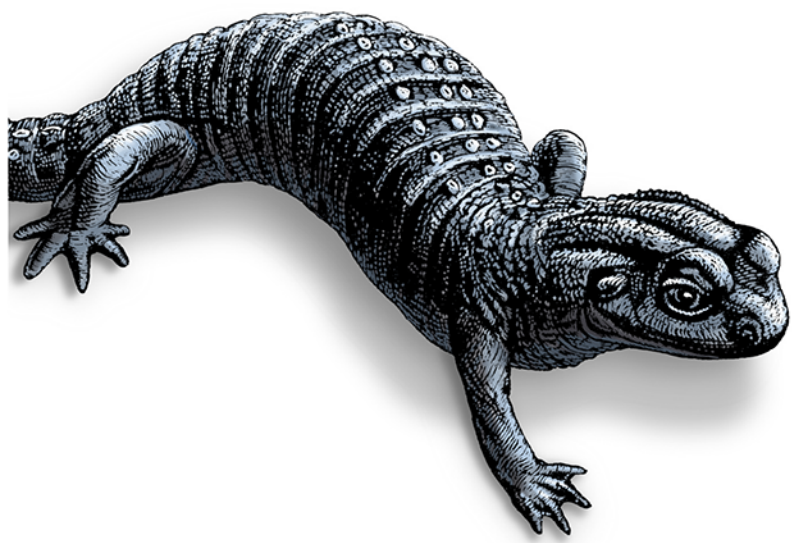
O'REILLY®

Wydanie IV

SQL

Leksykon kieszonkowy



Helion

Alice Zhao

Tytuł oryginału: SQL Pocket Guide: A Guide to SQL Usage, 4th Edition

Tłumaczenie: Artur Dorda

ISBN: 978-83-283-8928-1

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/sqllk4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

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)

Printed in Poland.

Spis treści

Rozdział 4
Podstawy zapytań

Zapytanie jest inną nazwą polecenia `SELECT`, które to składa się z sześciu głównych klauzul. Każdy podrozdział tego rozdziału omawia szczegółowo jedną z klauzul:

1. `SELECT`
2. `FROM`
3. `WHERE`
4. `GROUP BY`
5. `HAVING`
6. `ORDER BY`

Ostatni podrozdział tego rozdziału omawia klauzulę `LIMIT`, która jest obsługiwana przez MySQL, PostgreSQL oraz SQLite.

Przykłady kodu w tym rozdziale odwołują się do czterech tabel:

`wodospady`
: wodospady Górnego Półwyspu Michigan

`zarzadcy`
: zarządcy wodospadów

`hrabstwa`
: hrabstwa, w których znajdują się wodospady

`trasy`
: trasy wycieczek, które składają się z wielu przystanków przy wodospadach

Oto przykładowe zapytanie, które używa sześciu głównych klauzul. Po nim następują wyniki zapytania, które znane są również jako **zbiór wynikowy** (ang. *result set*).

```
-- Trasy z co najmniej 2 ogólnodostępnymi wodospadami
SELECT   t.nazwa AS nazwa_trasy,
         COUNT(*) AS ile_wodospadow
```

```
FROM     trasy t LEFT JOIN wodospady w
         ON t.przystanek = w.id
WHERE    w.ogolnodostepny = 't'
GROUP BY t.nazwa
HAVING   COUNT(*) >= 2
ORDER BY nazwa_trasy;

nazwa_trasy  ile_wodospadow
------------ ------------------
M-28                        6
Munising                    6
US-2                        4
```

Wykonać *zapytanie* do bazy danych znaczy pobrać dane z bazy danych, zazwyczaj z tabeli lub wielu tabel.

> **ZANOTUJ**
>
> Możliwe jest także wykonanie zapytania do **widoku** zamiast do tabeli. Widoki wyglądają jak tabele i są pochodnymi tabel, ale same w sobie nie przechowują żadnych danych. Więcej informacji na temat widoków znajdziesz w podrozdziale „Widoki” w rozdziale 5.

Klauzula SELECT

Klauzula `SELECT` wymienia kolumny, które mają być zwrócone poleceniem.

W klauzuli `SELECT` po słowie kluczowym `SELECT` następuje lista nazw kolumn i/lub wyrażeń oddzielonych przecinkami. Każda nazwa kolumny i/lub wyrażenie staje się wtedy kolumną w wynikach.

Pobieranie kolumn

Najprostsze zapytanie `SELECT` wypisuje wartości jednej lub więcej kolumn z tabel wymienionych w klauzuli `FROM`:

```
SELECT id, nazwa
FROM zarzadcy;

id      nazwa
------- ---------------
      1 Pictured Rocks
      2 Michigan Nature
      3 AF LLC
      4 MI DNR
      5 Horseshoe Falls
```

Pobieranie wszystkich kolumn

Aby zwrócić wszystkie kolumny z tabeli, możesz użyć pojedynczej gwiazdki, zamiast wypisywać każdą nazwę kolumny:

```
SELECT *
FROM zarzadcy;

id    nazwa             telefon        typ
----- ----------------- -------------- --------
    1 Pictured Rocks     906.387.2607  publiczny
    2 Michigan Nature    517.655.5655  prywatny
    3 AF LLC                           prywatny
    4 MI DNR             906.228.6561  publiczny
    5 Horseshoe Falls    906.387.2635  prywatny
```

UWAGA

Gwiazdka jest pomocnym skrótem podczas testowania zapytań, ponieważ może zaoszczędzić Ci sporo pisania. Jednakże używanie gwiazdki w kodzie produkcyjnym jest ryzykowne, ponieważ kolumny w tabeli mogą ulec zmianie, powodując błędy, gdy będzie ich mniej lub więcej, niż oczekiwano.

Pobieranie wyrażeń

Poza zwykłym wylistowaniem kolumn w klauzuli `SELECT` możesz również użyć bardziej złożonych wyrażeń, które będą zwrócone jako kolumny w wynikach.

Poniższe zapytanie zawiera wyrażenie obliczające spadek populacji o 10%, zaokrąglone do zera miejsc po przecinku:

```
SELECT nazwa, ROUND(populacja * 0.9, 0)
FROM hrabstwa;

nazwa      ROUND(populacja * 0.9, 0)
---------- --------------------------
Alger                           8876
Baraga                          7871
Ontonagon                       7036
...
```

Pobieranie funkcji

Wyrażenia na liście SELECT zazwyczaj odnoszą się do kolumn w tabelach, z których pobierasz dane, ale są od tego wyjątki. Na przykład często spotykaną funkcją, która nie odwołuje się do żadnej tabeli, jest funkcja zwracająca aktualną datę:

```
SELECT CURRENT_DATE;

CURRENT_DATE
------------
2021-12-01
```

Powyższy kod działa w MySQL, PostgreSQL oraz SQLite. Odpowiedniki kodu działające w innych RDBMS-ach znajdziesz w podrozdziale „Funkcje daty i czasu" w rozdziale 7.

> **ZANOTUJ**
>
> Większość zapytań zawiera zarówno klauzulę SELECT, jak i FROM, ale w przypadku niektórych funkcji bazodanowych, takich jak CURRENT_DATE, wymagana jest tylko klauzula SELECT.

Możliwe jest również użycie wewnątrz klauzuli SELECT wyrażeń, które są **podzapytaniami** (zapytanie zagnieżdżone wewnątrz innego zapytania). Więcej szczegółów znajdziesz w sekcji „Pobieranie podzapytań".

Aliasy kolumn

Celem **aliasu kolumny** jest nadanie tymczasowej nazwy dowolnej kolumnie lub wyrażeniu wymienionym w klauzuli SELECT. Ta tymczasowa nazwa, czyli alias kolumny, jest następnie wyświetlana jako nazwa kolumny w wynikach.

Zauważ, że nie jest to trwała zmiana nazwy, ponieważ nazwy kolumn w oryginalnych tabelach pozostają niezmienione. Alias istnieje tylko w obrębie zapytania.

Poniższy kod wyświetla trzy kolumny.

```
SELECT id, nazwa,
       ROUND(populacja * 0.9, 0)
FROM hrabstwa;
```

```
id    nazwa      ROUND(populacja * 0.9, 0)
----- ---------- --------------------------
    2 Alger                           8876
    6 Baraga                          7871
    7 Ontonagon                       7036
...
```

Załóżmy, że chcemy zmienić nazwy kolumn w wynikach. `id` jest niejasne i chcielibyśmy nadać mu bardziej opisową nazwę. `ROUND(populacja * 0.9, 0)` jest z kolei zbyt długie i chcielibyśmy nadać mu prostszą nazwę.

Aby utworzyć alias kolumny, za nazwą kolumny lub wyrażenia dajemy albo (1) nazwę aliasu, albo (2) słowo kluczowe `AS` i nazwę aliasu.

```
-- nazwa_aliasu
SELECT id hrabstwo_id, nazwa,
       ROUND(populacja * 0.9, 0) szacowana_pop
FROM hrabstwa;
```

lub:

```
-- AS nazwa_aliasu
SELECT id AS hrabstwo_id, nazwa,
       ROUND(populacja * 0.90, 0) AS szacowana_pop
FROM hrabstwa;

hrabstwo_id  nazwa      szacowana_pop
------------ ---------- --------------
           2 Alger                8876
           6 Baraga               7871
           7 Ontonagon            7036
...
```

Podczas tworzenia aliasów używane są w praktyce obie opcje. W klauzuli `SELECT` druga opcja jest bardziej popularna, ponieważ słowo kluczowe `AS` sprawia, że wizualnie łatwiej jest odróżnić nazwy kolumn i aliasów od długiej listy nazw kolumn.

> **ZANOTUJ**
>
> Starsze wersje PostgreSQL wymagają używania AS podczas tworzenia aliasów kolumn.

Chociaż aliasy kolumn nie są wymagane, są one zalecane podczas pracy z wyrażeniami, aby nadać sensowne nazwy kolumnom w wynikach.

Aliasy z uwzględnieniem wielkości liter i interpunkcji

Jak widać na przykładzie aliasów `hrabstwo_id` i `szacowana_pop`, konwencją nazywania aliasów kolumn jest używanie małych liter ze znakiem podkreślenia zamiast spacji.

Możesz także tworzyć aliasy zawierające wielkie litery, spacje i znaki interpunkcyjne, używając podwójnego cudzysłowu, jak pokazano w poniższym przykładzie:

```
SELECT id AS "Wodospad #",
  nazwa AS "Nazwa wodospadu"
FROM wodospady;

Wodospad #  Nazwa Wodospadu
----------- ---------------
          1 Munising Falls
          2 Tannery Falls
          3 Alger Falls
...
```

Dookreślanie kolumn

Załóżmy, że piszesz zapytanie, które pobiera dane z dwóch tabel, i obie zawierają kolumnę `nazwa`. Jeśli zawarłbyś w klauzuli `SELECT` tylko `nazwa`, zapytanie nie wiedziałoby, do której tabeli się odnosisz.

Aby rozwiązać ten problem, możesz **dookreślić** nazwę kolumny przez jej nazwę tabeli. Innymi słowy, używając **notacji kropkowej**, jak w `nazwa_tabeli.nazwa_kolumny`, możesz nadać kolumnie prefiks, aby określić, do której tabeli należy.

Poniższy przykład zapytuje pojedynczą tabelę, więc choć dookreślanie kolumn nie jest tu konieczne, pokazujemy to dla celów demonstracyjnych. W ten sposób dookreślisz kolumnę po nazwie tabeli:

```
SELECT zarzadcy.id, zarzadcy.nazwa
FROM zarzadcy;
```

WSKAZÓWKA

Jeśli otrzymasz błąd w SQL odwołujący się do **niejednoznacznej nazwy kolumny** (ang. *ambiguous column name*), oznacza to, że wiele tabel w Twoim zapytaniu ma kolumnę o tej samej nazwie, a Ty nie określiłeś, do której kombinacji tabeli i kolumny się odwołujesz. Możesz rozwiązać ten problem poprzez dookreślenie nazwy kolumny.

Dookreślanie tabel

Jeśli dookreślisz nazwę kolumny poprzez nazwę tabeli, możesz również dookreślić nazwę tej tabeli poprzez nazwę bazy danych lub schematu. Poniższe zapytanie pobiera dane konkretnie z tabeli `zarzadcy` w schemacie `ksiazkasql`:

```
SELECT ksiazkasql.zarzadcy.id, ksiazkasql.zarzadcy.nazwa
FROM ksiazkasql.zarzadcy;
```

Powyższy kod jest długi, ponieważ fragment `ksiazkasql.zarzadcy` jest powtarzany wielokrotnie. Aby zaoszczędzić sobie pisania, możesz utworzyć **alias tabeli**. Poniższy przykład tworzy alias z dla tabeli `zarzadcy`:

```
SELECT z.id, z.nazwa
FROM ksiazkasql.zarzadcy z;
```

lub:

```
SELECT z.id, z.nazwa
FROM zarzadcy z;
```

> **ALIASY KOLUMN KONTRA ALIASY TABEL**
>
> *Aliasy kolumn* są definiowane w klauzuli `SELECT`, aby zmienić nazwę kolumny w wynikach. Z reguły dołącza się do nich `AS`, choć nie jest to wymagane.
>
> ```
> -- Alias kolumny
> SELECT num AS nowa_kol
> FROM moja_tabela;
> ```
>
> *Aliasy tabel* są definiowane w klauzuli `FROM`, aby stworzyć tymczasowy pseudonim dla tabeli. Z reguły `AS` jest tu pomijane, choć dołączenie `AS` również zadziała.
>
> ```
> -- Alias tabeli
> SELECT *
> FROM moja_tabela mt;
> ```

Pobieranie podzapytań

Podzapytanie jest zapytaniem, które jest zagnieżdżone wewnątrz innego zapytania. Podzapytania mogą być umieszczone w różnych klauzulach, włączając w to klauzulę `SELECT`.

W poniższym przykładzie powiedzmy, że poza `id`, `nazwa` i `populacja` chcemy zobaczyć średnią populację wszystkich hrabstw. Poprzez dołączenie podzapytania tworzymy w wynikach nową kolumnę dla średniej populacji.

```
SELECT id, nazwa, populacja,
       (SELECT AVG(populacja) FROM hrabstwa)
       AS srednia_pop
FROM hrabstwa;

id    nazwa      populacja  srednia_pop
----- ---------- ---------- ------------
    2 Alger            9862        18298
    6 Baraga           8746        18298
    7 Ontonagon        7818        18298
...
```

Kilka rzeczy, na które warto zwrócić uwagę:

- Podzapytanie musi być otoczone nawiasami.
- Pisząc podzapytanie w klauzuli `SELECT`, zaleca się nadawanie aliasu kolumny, którym w tym przypadku jest `srednia_pop`. W ten sposób kolumna będzie miała prostą nazwę w wynikach.
- W kolumnie `srednia_pop` jest tylko jedna wartość, która powtarza się we wszystkich wierszach. Kiedy piszesz podzapytanie w klauzuli `SELECT`, wynik podzapytania musi zwracać pojedynczą kolumnę i albo zero, albo jeden wiersz, tak jak pokazuje to poniższe podzapytanie obliczające średnią populację.

  ```
  SELECT AVG(populacja) FROM hrabstwa;
  AVG(populacja)
  ----------------
            18298
  ```

- Jeśli podzapytanie zwróci zero wierszy, to nowa kolumna zostanie wypełniona wartościami `NULL`.

NIESKORELOWANE A SKORELOWANE PODZAPYTANIA

Powyższy przykład jest **podzapytaniem nieskorelowanym**, co oznacza, że podzapytanie nie odnosi się do zapytania zewnętrznego. Podzapytanie może być uruchamiane samodzielnie, niezależnie od zapytania zewnętrznego.

Drugi typ podzapytania nazywany jest **podzapytaniem skorelowanym**, czyli takim, które odwołuje się do wartości w zewnętrznym zapytaniu. To często znacznie spowalnia czas przetwarzania zapytania, więc najlepiej przepisać takie zapytanie, używając klauzuli `JOIN`. Poniżej znajduje się przykład podzapytania skorelowanego wraz z bardziej wydajnym kodem.

Problemy z wydajnością skorelowanych podzapytań

Poniższe zapytanie zwraca liczbę wodospadów dla każdego zarządcy. Zauważ, że warunek `z.id = w.zarzadca_id` w podzapytaniu odwołuje się do tabeli `zarzadcy` w zewnętrznym zapytaniu, co czyni je podzapytaniem skorelowanym.

```
SELECT z.id, z.nazwa,
       (SELECT COUNT(*) FROM wodospady w
       WHERE z.id = w.zarzadca_id) AS ile_wodospadow
FROM zarzadcy z;

id    name             ile_wodospadow
----- ---------------- ---------------
    1 Pictured Rocks                 3
    2 Michigan Nature                3
    3 AF LLC                         1
    4 MI DNR                         1
    5 Horseshoe Falls                0
```

Lepszym pomysłem byłoby przepisanie zapytania za pomocą klauzuli `JOIN`. W ten sposób tabele są najpierw łączone, a następnie wykonywana jest reszta zapytania, co jest znacznie szybsze niż wykonywanie podzapytania dla każdego wiersza danych. Więcej na temat złączeń znajdziesz w podrozdziale „Łączenie tabel” w rozdziale 9.

```
SELECT   z.id, z.nazwa,
         COUNT(w.id) AS ile_wodospadow
FROM     zarzadcy z LEFT JOIN wodospady w
         ON z.id = w.zarzadca_id
GROUP BY z.id, z.nazwa;

id    nazwa            ile_wodospadow
----- ---------------- ---------------
    1 Pictured Rocks                 3
    2 Michigan Nature                3
    3 AF LLC                         1
    4 MI DNR                         1
    5 Horseshoe Falls                0
```

DISTINCT

Kiedy kolumna jest wymieniona w klauzuli SELECT, domyślnie zwracane są wszystkie wiersze. Aby być bardziej precyzyjnym, możesz dołączyć słowo kluczowe ALL, ale jest ono czysto opcjonalne. Poniższe zapytania zwracają każdą kombinację kolumn typ/ogolnodostepny.

```
SELECT z.typ, w.ogolnodostepny
FROM zarzadcy z
JOIN wodospady w ON z.id = w.zarzadca_id;
```

lub:

```
SELECT ALL z.typ, w.ogolnodostepny
FROM zarzadcy z
JOIN wodospady w ON z.id = w.zarzadca_id;

typ        ogolnodostepny
---------- --------------
publiczny  y
publiczny  y
publiczny  y
prywatny   y
prywatny   y
prywatny   y
prywatny   y
publiczny  y
```

Jeśli chcesz usunąć duplikaty wierszy z wyników, możesz użyć słowa kluczowego DISTINCT. Poniższe zapytanie zwraca listę unikalnych kombinacji kolumn typ/ogolnodostepny.

```
SELECT DISTINCT z.type, w.ogolnodostepny
FROM zarzadcy z
JOIN wodospady w ON z.id = w.zarzadca_id;

typ        ogolnodostepny
---------- --------------
publiczny  y
prywatny   y
```

COUNT i DISTINCT

Aby policzyć liczbę unikalnych wartości w obrębie *pojedynczej kolumny*, możesz połączyć słowa kluczowe COUNT i DISTINCT w obrębie klauzuli SELECT. Poniższe zapytanie zwraca liczbę unikalnych wartości kolumny typ.

```
SELECT COUNT(DISTINCT typ) AS unikalne
FROM zarzadcy;
```

```
unikalne
---------
        2
```

Aby policzyć liczbę unikalnych kombinacji *wielu kolumn*, możesz zrobić z zapytania DISTINCT podzapytanie, a następnie wykonać na nim operację COUNT. Poniższe zapytanie zwraca liczbę unikalnych kombinacji typ/ogolnodostepny.

```
SELECT COUNT(*) AS ile_unikalnych
FROM (SELECT DISTINCT z.typ, w.ogolnodostepny
      FROM zarzadcy z JOIN wodospady w
      ON z.id = w.zarzadca_id) moje_podzapytanie;

ile_unikalnych
-----------
          2
```

MySQL oraz PostgreSQL obsługują użycie składni COUNT(DISTINCT) na wielu kolumnach. Poniższe dwa zapytania są równoważne powyższemu zapytaniu, bez potrzeby stosowania podzapytania:

```
-- wariant MySQL
SELECT COUNT(DISTINCT z.typ, w.ogolnodostepny)
       AS ile_unikalnych
       FROM zarzadcy z JOIN wodospady w
            ON z.id = w.zarzadca_id;

-- wariant PostgreSQL
SELECT COUNT(DISTINCT (z.typ, w.ogolnodostepny))
       AS ile_unikalnych
       FROM zarzadcy z JOIN wodospady w
            ON z.id = w.zarzadca_id;

ile_unikalnych
---------------
              2
```

Klauzula FROM

Klauzula FROM jest używana do określenia źródła danych, które chcesz pobrać. Najprostszym przypadkiem jest podanie nazwy pojedynczej tabeli lub widoku w klauzuli FROM.

```
SELECT nazwa
FROM wodospady;
```

Możesz dookreślić tabelę lub widok za pomocą nazwy bazy danych lub schematu, używając notacji kropkowej. Poniższe zapytanie pobiera dane z tabeli `wodospady` ze schematu `ksiazkasql`:

```
SELECT nazwa
FROM ksiazkasql.wodospady;
```

Pobieranie danych z wielu tabel

Zamiast pobierania danych z jednej tabeli często będziesz potrzebował pobrać naraz dane z wielu tabel. Najczęściej stosowanym sposobem na to jest użycie klauzuli `JOIN` wewnątrz klauzuli `FROM`. Poniższe zapytanie pobiera dane zarówno z tabeli `wodospady`, jak i `trasy` i wyświetla jedną tabelę wynikową.

```
SELECT *
FROM wodospady w JOIN trasy t
     ON w.id = t.przystanek;

id    nazwa           ... nazwa     przystanek  ...
----- --------------      --------- -----------
    1  Munising Falls     M-28                1
    1  Munising Falls     Munising            1
    2  Tannery Falls      Munising            2
    3  Alger Falls        M-28                3
    3  Alger Falls        Munising            3
...
```

Przeanalizujmy każdą część tego bloku kodu.

Aliasy tabel

```
wodospady w JOIN trasy t
```

Tabele `wodospady` i `trasy` otrzymały aliasy tabel `w` i `t`, które są tymczasowymi nazwami tabel w zapytaniu. Aliasy tabel nie są wymagane w klauzuli `JOIN`, ale są bardzo pomocne w skracaniu nazw tabel, do których trzeba się odwoływać w klauzulach `ON` i `SELECT`.

JOIN… ON…

```
wodospady w JOIN trasy t
ON w.id = t.przystanek
```

Te dwie tabele są złączone razem za pomocą słowa kluczowego `JOIN`. Po klauzuli `JOIN` zawsze następuje klauzula `ON`, która opisuje, jak tabele

powinny być ze sobą połączone. W tym przypadku id wodospadu w tabeli wodospady musi odpowiadać wartości kolumny przystanek (z wodospadem) w tabeli trasy.

ZANOTUJ

Możesz zobaczyć klauzule FROM, JOIN i ON wcięte lub w różnych liniach. Nie jest to wymagane, ale zwiększa czytelność zapytania, zwłaszcza gdy złączasz ze sobą wiele tabel.

Tabela wyników

Wynikiem zapytania jest zawsze jedna tabela. Tabela wodospady ma 12 kolumn, a tabela trasy ma 3 kolumny. Po złączeniu tych tabel w całość tabela wyników ma 15 kolumn.

```
 id   nazwa           ... nazwa     przystanek  ...
----- ---------------     --------- -----------
    1 Munising Falls      M-28                1
    1 Munising Falls      Munising            1
    2 Tannery Falls       Munising            2
    3 Alger Falls         M-28                3
    3 Alger Falls         Munising            3
...
```

Zauważysz, że w tabeli wyników znajdują się dwie kolumny o nazwie nazwa. Pierwsza z nich pochodzi z tabeli wodospady, a druga z tabeli trasy. Aby odwołać się do nich w klauzuli SELECT, potrzebujesz dookreślić nazwy kolumn.

```
SELECT w.nazwa, t.nazwa
FROM wodospady w JOIN trasy t
     ON w.id = t.przystanek;

nazwa           nazwa
--------------- ---------
Munising Falls  M-28
Munising Falls  Munising
Tannery Falls   Munising
...
```

Aby rozróżnić te dwie kolumny, potrzebujesz również nadać ich nazwom aliasy:

```
SELECT w.nazwa AS nazwa_wodospadu,
       t.nazwa AS nazwa_trasy
```

```
FROM wodospady w JOIN trasy t
     ON w.id = t.przystanek;

nazwa_wodospadu  nazwa_trasy
---------------- ---------
Munising Falls   M-28
Munising Falls   Munising
Tannery Falls    Munising
Alger Falls      M-28
Alger Falls      Munising
...
```

Warianty JOIN

W powyższym przykładzie jeśli wodospad nie jest wymieniony na żadnej trasie, nie pojawi się w tabeli wyników. Gdybyś chciał zobaczyć wszystkie wodospady w wynikach, musiałbyś użyć innego typu złączenia.

> **DOMYŚLNY JOIN TO INNER JOIN**
>
> Ten przykład używa zwykłego `JOIN`, aby pobrać dane z dwóch tabel naraz, jednak dobrą praktyką jest wyraźne określenie typu złączenia, którego używasz. `JOIN` sam w sobie domyślnie wykonuje złączenie wewnętrzne, `INNER JOIN`, co oznacza, że tylko rekordy znajdujące się w obu tabelach są zwracane w wynikach.

Istnieje wiele typów złączeń używanych w SQL, które są omówione bardziej szczegółowo w podrozdziale „Łączenie tabel" w rozdziale 9.

Pobieranie danych z podzapytań

Podzapytanie jest zapytaniem, które jest zagnieżdżone wewnątrz innego zapytania. Podzapytania w klauzuli `FROM` powinny być niezależnymi (nieskorelowanymi) zapytaniami `SELECT`, co oznacza, że nie odwołują się w ogóle do zewnętrznego zapytania i mogą być uruchamiane samodzielnie.

> **ZANOTUJ**
>
> Podzapytanie w klauzuli `FROM` nazywane jest również **tabelą pochodną**, ponieważ podzapytanie podczas trwania zapytania zachowuje się tak naprawdę jak tabela.

Poniższe zapytanie zwraca wszystkie wodospady będące własnością publiczną. Fragment kodu zawierający podzapytanie jest pogrubiony.

```
SELECT w.nazwa AS nazwa_wodospadu,
       z.nazwa AS nazwa_zarzadcy
FROM (SELECT * FROM zarzadcy WHERE typ = 'publiczny') z
     JOIN wodospady w
     ON z.id = w.zarzadca_id;

nazwa_wodospadu  nazwa_zarzadcy
---------------- ---------------
Little Miners    Pictured Rocks
Miners Falls     Pictured Rocks
Munising Falls   Pictured Rocks
Wagner Falls     MI DNR
```

Ważne jest, aby zrozumieć kolejność, w jakiej wykonywane jest to zapytanie.

Krok 1: Wykonaj podzapytanie

Podzapytanie jest wykonywane w pierwszej kolejności. Możesz zobaczyć, że w wyniku tego otrzymamy tabelę zawierającą tylko publicznych zarządców:

```
SELECT * FROM zarzadcy WHERE typ = 'publiczny';

id    nazwa           telefon       typ
----- --------------- ------------- ----------
    1 Pictured Rocks  906.387.2607  publiczny
    4 MI DNR          906.228.6561  publiczny
```

Wracając do oryginalnego zapytania, możesz zauważyć, że zaraz po podzapytaniu występuje litera z. Jest to tymczasowa nazwa, czy też alias, do której przypisujemy wyniki podzapytania.

ZANOTUJ

Aliasy są wymagane dla podzapytań w klauzuli `FROM` w bazach MySQL, PostgreSQL oraz SQL Server, ale nie w Oracle oraz SQLite.

Krok 2: Wykonaj całe zapytanie

Teraz możemy skupić się na literze z zajmującej miejsce podzapytania. Zapytanie jest teraz wykonywane jak zwykle.

```
SELECT w.nazwa AS nazwa_wodospadu,
       z.nazwa AS nazwa_zarzadcy
FROM z JOIN wodospady w
     ON z.id = w.zarzadca_id;

nazwa_wodospadu  nazwa_zarzadcy
---------------- ---------------
Little Miners    Pictured Rocks
Miners Falls     Pictured Rocks
Munising Falls   Pictured Rocks
Wagner Falls     MI DNR
```

PODZAPYTANIA A KLAUZULA WITH

Alternatywą dla pisania podzapytania jest napisanie wspólnego wyrażenia tabelarycznego (CTE, ang. *common table expression*) z użyciem klauzuli `WITH`. Zaletą klauzuli `WITH` jest to, że podzapytanie jest od razu nazwane, co czyni kod czystszym oraz daje możliwość wielokrotnego odwoływania się do podzapytania.

```
WITH z AS (SELECT * FROM zarzadcy
           WHERE typ = 'public')

SELECT w.nazwa AS nazwa_wodospadu,
z.nazwa AS nazwa_zarzadcy
FROM z JOIN wodospady w
     ON z.id = w.zarzadca_id;
```

Klauzula `WITH` jest obsługiwana przez MySQL 8.0+ (2018 i nowsze), PostgreSQL, Oracle, SQL Server oraz SQLite. Podrozdział „Wspólne wyrażenia tabelaryczne" w rozdziale 9. zawiera jej więcej przykładów.

Dlaczego warto używać podzapytania w klauzuli FROM?

Główną zaletą używania podzapytań jest to, że możesz rozbić większy problem na części. Oto dwa przykłady:

Przykład 1: Parę kroków prowadzących do wyników

Załóżmy, że chcesz znaleźć średnią liczbę przystanków na trasie. Najpierw musiałbyś znaleźć liczbę przystanków na każdej trasie, a następnie wyliczyć średnią z wyników.

Poniższe zapytanie wyszukuje liczbę przystanków na każdej trasie:

```
SELECT nazwa, MAX(przystanek) as ile_przystankow
FROM trasy
GROUP BY nazwa;

nazwa     ile_przystankow
--------- ----------------
M-28                   11
Munising                6
US-2                   14
```

Możesz wtedy zamienić to zapytanie w podzapytanie i napisać wokół niego kolejne zapytanie, aby znaleźć średnią:

```
SELECT AVG(ile_przystankow) FROM
(SELECT nazwa, MAX(przystanek) as ile_przystankow
FROM trasy
GROUP BY name) przystanki_na_trasie;

AVG(ile_przystankow)
--------------------
10.3333333333333
```

Przykład 2: Tabela w klauzuli FROM jest zbyt duża

Pierwotnym celem było wylistowanie wszystkich wodospadów będących własnością publiczną. To akurat można zrobić bez podzapytania, za pomocą `JOIN`:

```
SELECT w.nazwa AS nazwa_wodospadu,
       z.nazwa AS nazwa_zarzadcy
FROM zarzadcy z
       JOIN wodospady w ON z.id = w.zarzadca_id
WHERE   z.typ = 'publiczny';

nazwa_wodospadu  nazwa_zarzadcy
---------------- ---------------
Little Miners    Pictured Rocks
Miners Falls     Pictured Rocks
Munising Falls   Pictured Rocks
Wagner Falls     MI DNR
```

Załóżmy, że zapytanie trwa naprawdę długo. Może się tak zdarzyć, gdy łączysz ze sobą ogromne tabele (mówimy o dziesiątkach milionów wierszy). Jest wiele sposobów na przepisanie takiego zapytania, aby je przyspieszyć, a jednym z nich jest użycie podzapytania.

Skoro interesują nas tylko publiczni zarządcy, możemy najpierw napisać podzapytanie, które odfiltruje wszystkich prywatnych zarządców.

Odchudzona tabela zarzadcy zostanie następnie złączona z tabelą wodospady, co zajmie mniej czasu, a da takie same wyniki.

```
SELECT w.nazwa AS nazwa_wodospadu,
       z.nazwa AS nazwa_zarzadcy
FROM   (SELECT * FROM zarzadcy
        WHERE typ = 'publiczny') z
        JOIN wodospady w ON z.id = w.zarzadca_id;

nazwa_wodospadu  nazwa_zarzadcy
---------------- ---------------
Little Miners    Pictured Rocks
Miners Falls     Pictured Rocks
Munising Falls   Pictured Rocks
Wagner Falls     MI DNR
```

To tylko dwa z wielu przykładów na to, jak podzapytania mogą być użyte do rozbicia większego zapytania na mniejsze fragmenty.

Klauzula WHERE

Klauzula WHERE służy do ograniczenia wyników zapytania do interesujących nas wierszy, czyli mówiąc prościej, jest miejscem filtrowania danych. Rzadko kiedy będziesz chciał wyświetlić wszystkie wiersze z tabeli, a raczej wiersze, które spełniają określone kryteria.

WSKAZÓWKA

Przeszukując tabelę z milionami wierszy, nigdy nie wyszukuj poprzez SELECT * FROM moja_tabela; ponieważ wykonanie takiego zapytania niepotrzebnie zajmie dużo czasu.

Zdecydowanie lepszym pomysłem jest przefiltrowanie danych. Dwa przykłady filtrowania:

Filtrowanie po kolumnie w klauzuli WHERE

By wyszukiwanie było jeszcze szybsze, najlepiej jest filtrować po kolumnie, która jest indeksowana.

```
SELECT *
FROM moja_tabela
WHERE year_id = 2021;
```

Zwrócenie pierwszych (w tym przypadku 10) wierszy danych za pomocą klauzuli `LIMIT` *(lub* `WHERE ROWNUM <= 10` *w Oracle, a* `SELECT TOP 10 *` *w SQL Server)*

```
SELECT *
FROM moja_tabela
LIMIT 10;
```

Poniższe zapytanie wyszukuje wszystkie wodospady, które nie zawierają w nazwie fragmentu *Falls*[1]. Więcej na temat słowa kluczowego `LIKE` można znaleźć w rozdziale 7.

```
SELECT id, nazwa
FROM wodospady
WHERE nazwa NOT LIKE '%Falls%';
x
id    nazwa
----- ----------------
    7 Little Miners
   14 Rapid River Fls
```

Pogrubiony fragment jest często nazywany instrukcją warunkową lub predykatem. Predykat dokonuje logicznego porównania dla każdego wiersza danych, którego wynikiem jest `TRUE`/`FALSE`/`UNKNOWN`.

Tabela `wodospady` ma 16 wierszy. Dla każdego wiersza sprawdzane jest, czy nazwa wodospadu zawiera `Falls`, czy nie. Jeśli nie zawiera, wtedy predykat `nazwa NOT LIKE '%Falls%'` zwraca `TRUE` i wiersz jest zwracany w wynikach, tak jak to było w przypadku dwóch pokazanych wyżej w tabelce wierszy.

Wielokrotne predykaty

Możliwe jest także łączenie wielu predykatów za pomocą **operatorów** takich jak `AND` lub `OR`. Poniższe zapytanie wyszukuje wszystkie wodospady, które nie zawierają w nazwie fragmentu *Falls* oraz nie mają zarządcy:

```
SELECT id, nazwa
FROM wodospady
WHERE nazwa NOT LIKE '%Falls%'
      AND zarzadca_id IS NULL;
```

[1] *Falls* występujące jako odrębne słowo lub jako część słowa *waterfalls* (pol. wodospady), to odpowiednik „spad” w słowie „wodospad” — *przyp. tłum.*

```
id    nazwa
----- ----------------
   14 Rapid River Fls
```

Więcej szczegółów na temat operatorów znajdziesz w podrozdziale „Operatory” w rozdziale 7.

Filtrowanie po podzapytaniach

Podzapytanie jest zapytaniem zagnieżdżonym wewnątrz innego zapytania, a klauzula `WHERE` jest miejscem, w którym można je często znaleźć. Poniższy przykład wyszukuje publicznie dostępne wodospady znajdujące się w hrabstwie Alger:

```
SELECT w.nazwa
FROM   wodospady w
WHERE  w.ogolnodostepny = 't'
       AND w.hrabstwo_id IN (
           SELECT h.id FROM hrabstwa h
           WHERE h.nazwa = 'Alger');

nazwa
---------------
Munising Falls
Tannery Falls
Alger Falls
...
```

> **ZANOTUJ**
>
> W przeciwieństwie do podzapytań w klauzuli `SELECT` lub `FROM` podzapytania w klauzuli `WHERE` nie wymagają aliasu. Co więcej, jeśli zamieścisz w niej alias, otrzymasz błąd.

Dlaczego warto używać podzapytania w klauzuli WHERE?

Pierwotnym celem było wyszukanie publicznie dostępnych wodospadów znajdujących się w hrabstwie Alger. Gdybyś miał napisać to zapytanie od zera, prawdopodobnie zacząłbyś od czegoś w tym stylu:

```
SELECT w.nazwa
FROM   wodospady w
WHERE  w.ogolnodostepny = 't';
```

Na tym etapie mamy wszystkie wodospady, które są ogólnodostępne. Ostatnim krokiem jest znalezienie tych, które znajdują się w hrabstwie

Alger. Wiemy, że tabela `wodospady` nie ma kolumny z nazwą hrabstwa, ale tabela `hrabstwa` już tak.

Masz więc dwie opcje, aby dołączyć nazwę hrabstwa do wyników. Możesz albo (1) napisać podzapytanie w klauzuli `WHERE`, które pobiera informacje o hrabstwie Alger, albo (2) złączyć tabele `wodospady` i `hrabstwa`:

```
-- Podzapytanie w klauzuli WHERE
SELECT w.nazwa
FROM   wodospady w
WHERE  w.ogolnodostepny = 't'
       AND w.hrabstwo_id IN (
           SELECT h.id FROM hrabstwa h
           WHERE h.nazwa = 'Alger');
```

lub:

```
-- Klauzula JOIN
SELECT w.nazwa
FROM   wodospady w INNER JOIN hrabstwa h
       ON w.hrabstwo_id = h.id
WHERE  w.ogolnodostepny = 't'
       AND h.nazwa = 'Alger';

nazwa
---------------
Munising Falls
Tannery Falls
Alger Falls
...
```

Oba zapytania dają takie same wyniki. Zaletą pierwszego podejścia jest to, że podzapytania są często łatwiejsze do zrozumienia niż złączenia. Zaletą drugiego podejścia jest to, że złączenia zazwyczaj wykonują się szybciej niż podzapytania.

DZIAŁAJĄCY > ZOPTYMALIZOWANY

Podczas pisania kodu SQL z reguły istnieje wiele sposobów na zrobienie tej samej rzeczy.

Twoim głównym priorytetem powinno być napisanie *działającego* kodu. Jeśli jego uruchomienie zajmuje dużo czasu lub jest brzydki, to nie ma znaczenia… on działa!

Następnym krokiem, jeśli masz czas, jest *zoptymalizowanie* kodu poprzez poprawienie wydajności, być może przepisanie go za pomocą `JOIN`, uczynienie go bardziej czytelnym dzięki wcięciom i wielkim literom itp.

Nie stresuj się pisaniem od razu najbardziej optymalnego kodu, ale staraj się raczej pisać kod, który działa. Pisanie eleganckiego kodu przychodzi z doświadczeniem.

Inne sposoby filtrowania danych

Klauzula `WHERE` nie jest jedynym miejscem w zapytaniu typu `SELECT`, w którym można filtrować wiersze danych.

- Klauzula `FROM`: Podczas złączania tabel klauzula `ON` określa, w jaki sposób powinny one zostać złączone. W tym miejscu możesz zawrzeć instrukcje warunkowe, aby ograniczyć wiersze danych zwróconych przez zapytanie. Więcej szczegółów znajdziesz w podrozdziale „Łączenie tabel” w rozdziale 9.
- Klauzula `HAVING`: Jeśli w zapytaniu typu `SELECT` występują agregacje, klauzula `HAVING` jest miejscem, gdzie możesz określić, jak agregacje powinny być przefiltrowane. Więcej szczegółów znajdziesz w podrozdziale „Klauzula HAVING”.
- Klauzula `LIMIT`: Aby wyświetlić określoną liczbę wierszy, możesz użyć klauzuli LIMIT. W Oracle robi się to za pomocą `WHERE ROWNUM`, a w *SQL Server* za pomocą `SELECT TOP`. Więcej szczegółów znajdziesz w ostatnim podrozdziale tego rozdziału, „Klauzula LIMIT”.

Klauzula GROUP BY

Celem klauzuli `GROUP BY` jest zbieranie wierszy w grupy, zsumowanie w dany sposób wierszy w obrębie grup i zwrócenie ostatecznie tylko jednego wiersza na grupę. Czasami jest to określane jako „krojenie” wierszy na grupy i „zwijanie” wierszy w każdej grupie.

Poniższe zapytanie zlicza liczbę wodospadów wzdłuż każdej z tras:

```
SELECT   t.nazwa AS nazwa_trasy,
         COUNT(*) AS ile_wodospadow
FROM     wodospady w INNER JOIN trasy t
         ON w.id = t.przystanek
GROUP BY t.nazwa;
```

```
nazwa_trasy  ile_wodospadow
------------ ---------------
M-28                       6
Munising                   6
US-2                       4
```

Mamy tu dwie główne części, na których należy się skupić:

- *zbieranie wierszy* w grupy, co jest określone w klauzuli `GROUP BY`;
- *zsumowywanie wierszy* w obrębie grup, co jest określone w klauzuli `SELECT`.

Krok 1: Zbieranie wierszy w grupy

W klauzuli `GROUP BY`:

```
GROUP BY t.nazwa
```

stwierdzamy, że chcielibyśmy wziąć wszystkie wiersze z danymi i umieścić wodospady z trasy M-28 w jednej grupie, wodospady z trasy Munising w drugiej grupie i tak dalej. Pod spodem dane są grupowane w następujący sposób:

```
nazwa_trasy  nazwa_wodospadu
------------ ----------------
M-28         Munising Falls
M-28         Alger Falls
M-28         Scott Falls
M-28         Canyon Falls
M-28         Agate Falls
M-28         Bond Falls

Munising     Munising Falls
Munising     Tannery Falls
Munising     Alger Falls
Munising     Wagner Falls
Munising     Horseshoe Falls
Munising     Miners Falls

US-2         Bond Falls
US-2         Fumee Falls
US-2         Kakabika Falls
US-2         Rapid River Fls
```

Krok 2: Zsumowywanie wierszy

W klauzuli SELECT:

```
SELECT t.nazwa AS nazwa_trasy,
       COUNT(*) AS ile_wodospadow
```

stwierdzamy, że dla każdej grupy, czy też — każdej trasy, chcemy policzyć liczbę wierszy danych w grupie. Ponieważ każdy wiersz reprezentuje wodospad, w rezultacie otrzymamy całkowitą liczbę wodospadów na każdej trasie.

Funkcja COUNT() jest bardziej formalnie nazywana jako **funkcja agregująca** (ang. *aggregate function*), czyli funkcja, która zsumowuje wiele wierszy danych w jedną wartość. Więcej funkcji agregujących znajdziesz w podrozdziale „Funkcje agregujące” w rozdziale 7.

UWAGA

W powyższym przykładzie COUNT(*) zwraca liczbę wodospadów wzdłuż każdej trasy. Dzieje się tak jednak tylko dlatego, że każdy wiersz danych w tabelach wodospady i trasy reprezentuje jeden wodospad.

Jeśli jeden wodospad znajdowałby się w wielu wierszach, funkcja COUNT(*) zwróciłaby większą liczbę niż oczekiwana. W takim przypadku zamiast tego moglibyśmy potencjalnie użyć COUNT(DISTINCT nazwa_wodospadu), aby znaleźć unikalne wodospady. Więcej szczegółów znajdziesz w sekcji „COUNT i DISTINCT”.

Kluczowym wnioskiem do zapamiętania jest to, że ważne jest, by ręcznie sprawdzać wyniki funkcji agregującej, aby upewnić się, że zsumowuje ona dane taki w sposób, jaki zamierzałeś.

Teraz, gdy grupy zostały utworzone za pomocą klauzuli GROUP BY, funkcja agregująca zostanie zastosowana po jednym razie dla każdej grupy:

```
nazwa_trasy  COUNT(*)
------------ ---------
M-28                 6
M-28
M-28
M-28
M-28
M-28
```

```
Munising             6
Munising
Munising
Munising
Munising
Munising

US-2                 4
US-2
US-2
US-2
```

Wszystkie kolumny, do których nie została zastosowana funkcja agregująca, czyli w tym przypadku kolumna `nazwa_trasy`, są teraz zwinięte w jedną wartość:

```
nazwa_trasy  COUNT(*)
------------ ---------
M-28                 6
Munising             6
US-2                 4
```

ZANOTUJ

Zwijanie wielu wierszy w jeden zagregowany wiersz oznacza, że kiedy używamy klauzuli `GROUP BY`, klauzula `SELECT` powinna zawierać *tylko*:

- wszystkie kolumny wymienione w klauzuli `GROUP BY`: `t.nazwa`
- agregacje: `COUNT(*)`

```
SELECT t.nazwa AS nazwa_trasy,
       COUNT(*) AS ile_wodospadow
...
GROUP BY t.nazwa;
```

Niezastosowanie się do tego zalecenia może spowodować wyświetlenie komunikatu o błędzie lub zwrócenie niedokładnych wartości.

GROUP BY W PRAKTYCE

Oto kroki, które powinieneś podjąć, gdy używasz `GROUP BY`:

1. Zastanów się, jakiej kolumny (jakich kolumn) chcesz użyć, aby wyodrębnić czy pogrupować swoje dane (np. nazwa trasy).
2. Zastanów się, jak chciałbyś zsumować dane w każdej grupie (np. policzyć wodospady wzdłuż każdej trasy).

Kiedy już podejmiesz decyzję:

1. w klauzuli SELECT wymień kolumny, które chcesz pogrupować (np. nazwa trasy), oraz agregacje, które chcesz obliczyć w każdej grupie (np. liczba wodospadów);
2. w klauzuli GROUP BY wymień wszystkie kolumny, które nie są agregatami (np. nazwa trasy).

Aby poznać bardziej złożone sytuacje związane z grupowaniem, takie jak ROLLUP, CUBE i GROUPING SETS, przejdź do podrozdziału „Grupowanie i zsumowywanie" w rozdziale 8.

Klauzula HAVING

Klauzula HAVING nakłada ograniczenia na wiersze zwrócone z zapytania GROUP BY. Innymi słowy, pozwala Ci przefiltrować wyniki po zastosowaniu GROUP BY.

ZANOTUJ

Klauzula HAVING zawsze występuje bezpośrednio po klauzuli GROUP BY. Bez klauzuli GROUP BY nie może być klauzuli HAVING.

Oto zapytanie, które zwraca liczbę wodospadów na każdej trasie, używając klauzuli GROUP BY:

```
SELECT   t.nazwa AS nazwa_trasy,
         COUNT(*) AS ile_wodospadow
FROM     wodospady w INNER JOIN trasy t
         ON w.id = t.przystanek
GROUP BY t.nazwa;

nazwa_trasy  ile_wodospadow
------------ ---------------
M-28                       6
Munising                   6
US-2                       4
```

Załóżmy, że chcemy wypisać tylko te trasy, które mają dokładnie sześć przystanków. Aby to zrobić, musisz dodać klauzulę HAVING po klauzuli GROUP BY:

```
SELECT   t.nazwa AS nazwa_trasy,
         COUNT(*) AS ile_wodospadow
```

```
FROM     wodospady w INNER JOIN trasy t
         ON w.id = t.przystanek
GROUP BY t.nazwa
HAVING   COUNT(*) = 6;

nazwa_trasy  ile_wodospadow
------------ ---------------
M-28                       6
Munising                   6
```

WHERE KONTRA HAVING

Celem obu klauzul jest filtrowanie danych. Jeśli chcesz:

- filtrować po poszczególnych kolumnach, wpisz swoje warunki w klauzuli `WHERE`;
- filtrować po agregatach, wpisz swoje warunki w klauzuli `HAVING`.

Zawartości klauzuli `WHERE` i `HAVING` nie mogą być zamienione.

- Nigdy nie umieszczaj warunku dotyczącego agregacji w klauzuli `WHERE`. Otrzymasz błąd.
- Nigdy nie umieszczaj w klauzuli `HAVING` warunku, który nie dotyczy agregacji. Takie warunki są przetwarzane znacznie wydajniej w klauzuli `WHERE`.

Zauważ, że klauzula `HAVING` odnosi się do agregacji `COUNT(*)`,

```
SELECT COUNT(*) AS ile_wodospadow
...
MAJĄC COUNT(*) = 6;
```

a nie do aliasu,

```
# ten kod nie zadziała
SELECT COUNT(*) AS ile_wodospadow
...
HAVING ile_wodospadow = 6;
```

Powodem tego jest kolejność wykonywania klauzul. Klauzula `SELECT` jest napisana przed klauzulą `HAVING`. Jednak klauzula `SELECT` jest w rzeczywistości wykonywana *po* klauzuli `HAVING`.

To oznacza, że alias `ile_wodospadow` w klauzuli `SELECT` nie istnieje w momencie wykonywania klauzuli `HAVING`. Klauzula `HAVING` musi odnosić się do czystej agregacji `COUNT(*)`.

ZANOTUJ

MySQL i SQLite są wyjątkami i pozwalają na aliasy (`ile_wodospadow`) w klauzuli `HAVING`.

Klauzula ORDER BY

Klauzula `ORDER BY` jest używana do określenia, w jaki sposób wyniki zapytania mają zostać posortowane.

Poniższe zapytanie zwraca listę zarządców i wodospadów, bez żadnego sortowania:

```
SELECT COALESCE(o.nazwa, 'Nieznany') AS zarzadca,
       w.nazwa AS nazwa_wodospadu
FROM   wodospady w
       LEFT JOIN zarzadcy z ON w.zarzadca_id = z.id;

zarzadca         nazwa_wodospadu
---------------- ---------------
Pictured Rocks   Munising Falls
Michigan Nature  Tannery Falls
AF LLC           Alger Falls
MI DNR           Wagner Falls
Nieznany         Horseshoe Falls
...
```

FUNKCJA COALESCE

Funkcja `COALESCE` zastępuje wszystkie wartości `NULL` w danej kolumnie inną wartością. W tym przypadku zamieniła ona wartości `NULL` w kolumnie `z.nazwa` na tekst `Nieznany`.

Gdyby nie użyto tutaj funkcji `COALESCE`, wszystkie wodospady bez zarządców zostałyby pominięte w wynikach. Zamiast tego są one teraz oznaczone jako takie, których zarządca jest `Nieznany`, i mogą być posortowane oraz uwzględnione w wynikach.

Więcej szczegółów znajdziesz w rozdziale 7.

Poniższe zapytanie zwraca tę samą listę, ale najpierw posortowaną alfabetycznie według zarządców, a następnie według wodospadów:

```
SELECT   COALESCE(o.nazwa, 'Nieznany') AS zarzadca,
         w.nazwa AS nazwa_wodospadu
```

```
FROM     wodospady w
         LEFT JOIN zarzadcy z ON w.zarzadca_id = z.id
ORDER BY zarzadca, nazwa_wodospadu;

zarzadca         nazwa_wodospadu
---------------- ---------------
AF LLC           Alger Falls
MI DNR           Wagner Falls
Michigan Nature  Tannery Falls
Michigan Nature  Twin Falls #1
Michigan Nature  Twin Falls #2
...
```

Domyślnym sortowaniem jest sortowanie rosnące (ang. *ascending*), co oznacza, że tekst będzie ułożony od A do Z, a liczby od najniższej do najwyższej. Możesz użyć słów kluczowych `ASCENDING` i `DESCENDING` (które mogą być skrócone do `ASC` i `DESC`), aby kontrolować sortowanie każdej kolumny.

Poniższe polecenie jest modyfikacją poprzedniego sortowania, tym razem sortuje nazwy zarządców w odwrotnej kolejności:

```
SELECT COALESCE(o.nazwa, 'Nieznany') AS zarzadca,
       w.nazwa AS nazwa_wodospadu
...
ORDER BY zarzadca DESC, nazwa_wodospadu ASC;

zarzadca         nazwa_wodospadu
---------------- ---------------
Nieznany         Agate Falls
Nieznany         Bond Falls
Nieznany         Canyon Falls
...
```

Możesz sortować po kolumnach i wyrażeniach, które nie znajdują się na liście w klauzuli `SELECT`:

```
SELECT   COALESCE(o.nazwa, 'Nieznany') AS zarzadca,
         w.nazwa AS nazwa_wodospadu
FROM     wodospady w
         LEFT JOIN zarzadcy z ON w.zarzadca_id = z.id
ORDER BY z.id DESC, w.id;

zarzadca         nazwa_wodospadu
---------------- ---------------
MI DNR           Wagner Falls
AF LLC           Alger Falls
Michigan Nature  Tannery Falls
...
```

Możesz również sortować po numerze porządkowym kolumny:

```
SELECT COALESCE(o.nazwa, 'Nieznany') AS zarzadca,
       w.nazwa AS nazwa_wodospadu
...
ORDER BY 1 DESC, 2 ASC;

zarzadca         nazwa_wodospadu
---------------- ---------------
Nieznany         Agate Falls
Nieznany         Bond Falls
Nieznany         Canyon Falls
...
```

Ponieważ wiersze tabeli SQL nie są domyślnie uporządkowane w żaden sposób, jeśli nie zawrzesz klauzuli `ORDER BY` w zapytaniu, za każdym razem, gdy je wykonasz, wyniki mogą być wyświetlone w innej kolejności.

> **ORDER BY NIE MOŻE BYĆ UŻYTE W PODZAPYTANIU**
>
> Z sześciu głównych klauzul tylko klauzula `ORDER BY` nie może być użyta w podzapytaniu. Niestety, nie możesz zmusić wierszy podzapytania do bycia uporządkowanymi.
>
> Aby uniknąć tego problemu, musiałbyś przepisać swoje zapytanie tak, by nie użyć klauzuli `ORDER BY` wewnątrz podzapytania, ale użyć jej tylko w zewnętrznym zapytaniu.

Klauzula LIMIT

Dobrą praktyką przy przeglądaniu szybko tabeli jest zwrócenie ograniczonej liczby wierszy zamiast całej tabeli.

MySQL, PostgreSQL oraz SQLite wspierają klauzulę `LIMIT`. Oracle i SQL Server używają natomiast innej składni z tą samą funkcjonalnością:

```
-- MySQL, PostgreSQL oraz SQLite
SELECT *
FROM zarzadcy
LIMIT 3;

-- Oracle
SELECT *
FROM zarzadcy
WHERE ROWNUM <= 3;
```

```
-- SQL Server
SELECT TOP 3 *
FROM zarzadcy;

id  nazwa            telefon       typ
--- ---------------- ------------- ----------
  1 Pictured Rocks   906.387.2607  publiczny
  2 Michigan Nature  517.655.5655  prywatny
  3 AF LLC                         prywatny
```

Innym sposobem na ograniczenie liczby zwracanych wierszy jest filtrowanie po kolumnie w klauzuli `WHERE`. Filtrowanie będzie wykonane jeszcze szybciej, jeśli kolumna jest indeksowana.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

Mijają lata, a w pracy z danymi to SQL wciąż jest najważniejszy!

Język SQL jest istotnym narzędziem nie tylko dla programistów, ale także dla analityków biznesowych i inżynierów danych. Nawet jeśli nieźle znasz składnię SQL, może się zdarzyć, że podczas pracy poczujesz potrzebę odświeżenia wiedzy czy też sprawdzenia jakiegoś szczegółu działania swojego zapytania SQL. W takich wypadkach nie potrzebujesz drobiazgowej specyfikacji technicznej ani opasłego podręcznika: po prostu zależy Ci na szybkim i pewnym odnalezieniu potrzebnej informacji bez przebijania się przez dogłębne wyjaśnienia.

To czwarte, poprawione i zaktualizowane wydanie cenionego leksykonu poświęconego SQL. Przemyślany układ zawartych w nim treści zdecydowanie ułatwia i przyspiesza wynajdywanie informacji — bez konieczności odrywania się od pracy. Poza zwięzłymi objaśnieniami dotyczącymi składni SQL znajdziesz tu opis kluczowych aspektów języka SQL używanego w Microsoft SQL Server, MySQL, Oracle Database, PostgreSQL i SQLite. Ułatwi Ci to stosowanie tych systemów zarządzania bazami danych. Oprócz tego w książce szybko odszukasz szczegóły dotyczące typów danych i ich konwersji, składni wyrażeń regularnych, funkcji okna, pivotingu i unpivotingu, a także wielu innych zagadnień.

Dzięki książce:

- szybko sprawdzisz, jak wykonać konkretne zadania za pomocą SQL
- znajdziesz przydatne przykłady składni
- sprawisz, aby zapytania SQL działały w różnych systemach zarządzania bazami danych
- zastosujesz kod Pythona i R do pracy z relacyjną bazą danych
- znajdziesz odpowiedzi na często zadawane pytania dotyczące SQL

Alice Zhao jest inżynierem danych. Prowadziła liczne kursy z zakresu SQL i Pythona, jest też autorką cenionych samouczków technicznych opublikowanych na YouTube. Współzałożycielka Best Fit Analytics, wcześniej analityk danych w Metis. Prelegentka na wielu prestiżowych konferencjach dotyczących sztucznej inteligencji.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-8928-1

9 788328 389281

Cena: 59,00 zł