

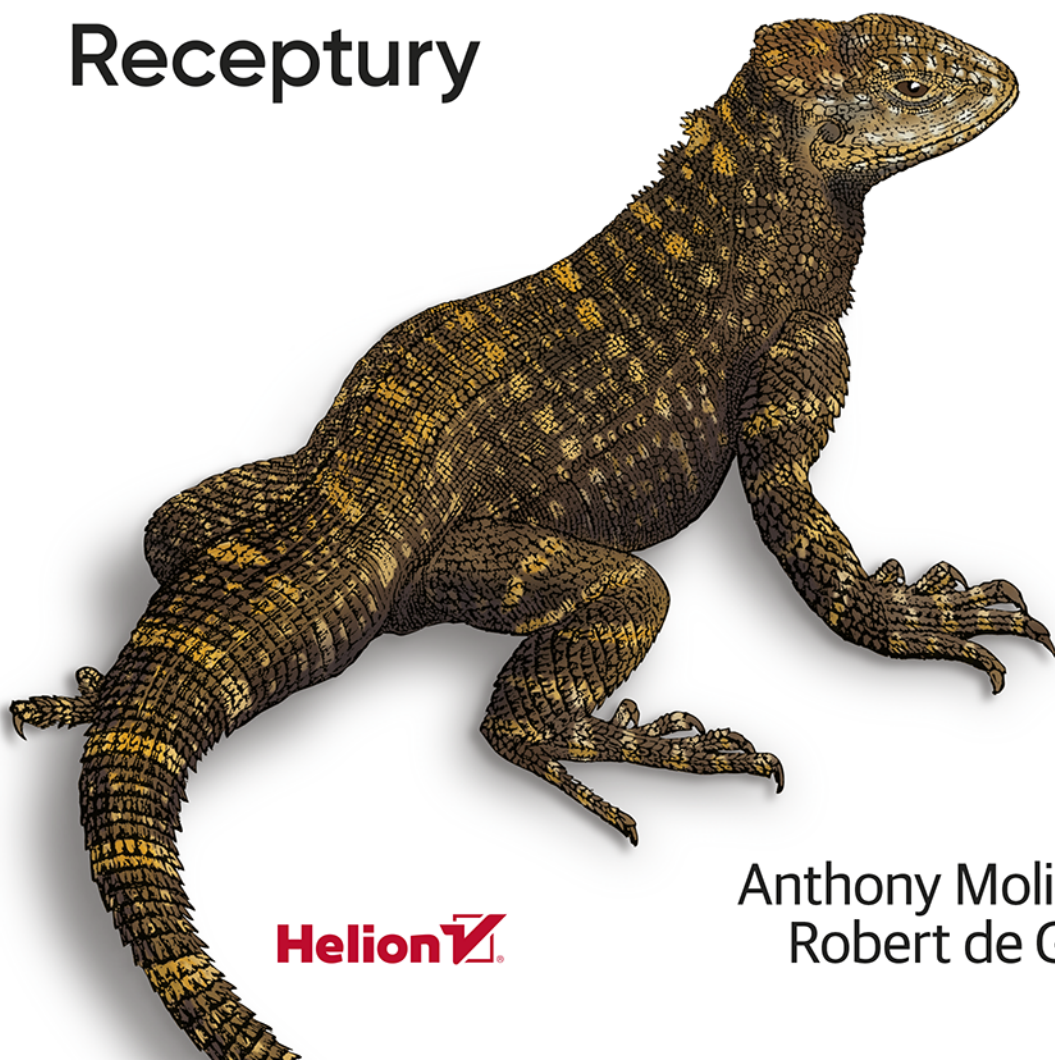
O'REILLY®

Wydanie II

# SQL

Zapytania i techniki  
dla bazodanowców

Receptury



Helion 

Anthony Molinaro  
Robert de Graaf

Tytuł oryginału: SQL Cookbook: Query Solutions and Techniques for All SQL Users, 2nd Edition

Tłumaczenie: Mikołaj Szczepaniak, Piotr Cieślak

ISBN: 978-83-283-7847-6

© 2021 Helion S.A.

Authorized Polish translation of the English edition of SQL Cookbook, 2E

ISBN 9781492077442 © 2021 Robert de Graaf.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również adnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/sqlzr2>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<https://ftp.helion.pl/przyklady/sqlzr2.zip>

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

<b>Przedmowa .....</b>	<b>11</b>
<b>1. Odczytywanie rekordów .....</b>	<b>21</b>
1.1. Odczytywanie wszystkich wierszy i kolumn tabeli	21
1.2. Odczytywanie podzbioru wierszy tabeli	22
1.3. Odnajdywanie wierszy spełniających wiele warunków	22
1.4. Odczytywanie podzbioru kolumn tabeli	23
1.5. Definiowanie sensownych nazw kolumn	24
1.6. Odwołania do aliasów kolumn w klauzuli WHERE	25
1.7. Konkatenacja wartości kolumn	26
1.8. Stosowanie logiki warunkowej w wyrażeniu SELECT	27
1.9. Ograniczanie liczby zwracanych wierszy	28
1.10. Zwracanie n losowych rekordów tabeli	30
1.11. Odnajdywanie wartości pustych (NULL)	31
1.12. Przekształcanie wartości pustych w rzeczywiste	32
1.13. Poszukiwanie wzorców	32
1.14. Podsumowanie	33
<b>2. Sortowanie wyników zapytań .....</b>	<b>35</b>
2.1. Zwracanie wyników zapytań posortowanych w określonym porządku	35
2.2. Sortowanie zbioru wynikowego według zawartości wielu pól	36
2.3. Sortowanie według podłańcuchów	37
2.4. Sortowanie wymieszanych danych alfanumerycznych	38
2.5. Obsługa wartości pustych w zapytaniach sortujących	41
2.6. Sortowanie według klucza zależnego od danych	47
2.7. Podsumowanie	48
<b>3. Praca z wieloma tabelami .....</b>	<b>49</b>
3.1. Umieszczanie jednego zbioru wierszy ponad drugim	49
3.2. Łączenie wzajemnie powiązanych wierszy	51
3.3. Odnajdywanie wspólnych wierszy pomiędzy dwiema tabelami	53
3.4. Uzyskiwanie z jednej tabeli tylko tych wartości, które nie występują w innej tabeli	54

3.5. Uzyskiwanie z jednej tabeli tylko tych wierszy, dla których nie istnieją odpowiedniki w innej tabeli	60
3.6. Dodawanie złączeń do zapytań bez konieczności modyfikowania pozostałych, już istniejących złączeń	62
3.7. Określanie, czy dwie tabele zawierają te same dane	64
3.8. Identyfikowanie i eliminowanie iloczynów kartezjańskich	71
3.9. Stosowanie złączeń w zapytaniach wykorzystujących funkcje agregujące	72
3.10. Stosowanie złączeń zewnętrznych w zapytaniach wykorzystujących funkcje agregujące	77
3.11. Zwracanie brakujących danych z wielu tabel	80
3.12. Wykorzystywanie wartości NULL w operacjach i porównaniach	83
3.13. Podsumowanie	84
<b>4. Wstawianie, aktualizowanie i usuwanie .....</b>	<b>85</b>
4.1. Wstawianie nowych rekordów	86
4.2. Wstawianie wartości domyślnych	87
4.3. Zastępowanie wartości domyślnych wartością NULL	88
4.4. Kopiowanie wierszy pomiędzy tabelami	89
4.5. Kopiowanie definicji tabel	90
4.6. Wstawianie wierszy do wielu tabel jednocześnie	91
4.7. Blokowanie możliwości wstawiania wartości do wybranych kolumn	93
4.8. Modyfikowanie rekordów tabeli	94
4.9. Aktualizowanie danych pod warunkiem istnienia w tabeli określonych wierszy	95
4.10. Aktualizowanie wartości według zawartości innej tabeli	96
4.11. Scalanie rekordów	100
4.12. Usuwanie wszystkich rekordów z tabeli	102
4.13. Usuwanie rekordów spełniających określone kryteria	102
4.14. Usuwanie pojedynczych rekordów	103
4.15. Usuwanie wierszy naruszających integralność referencyjną	104
4.16. Usuwanie powtarzających się rekordów	105
4.17. Usuwanie rekordów na podstawie danych z innej tabeli	107
4.18. Podsumowanie	108
<b>5. Zapytania przetwarzające metadane .....</b>	<b>109</b>
5.1. Generowanie listy tabel wchodzących w skład schematu bazy danych	109
5.2. Generowanie listy kolumn danej tabeli	110
5.3. Generowanie listy indeksowanych kolumn danej tabeli	111
5.4. Generowanie listy ograniczeń zdefiniowanych dla tabeli	113
5.5. Generowanie listy kluczy obcych pozbawionych indeksów	114
5.6. Generowanie kodu języka SQL za pomocą wyrażeń tego języka	118
5.7. Opisywanie widoków słownika danych w bazie danych Oracle	120
5.8. Podsumowanie	121

<b>6. Praca z łańcuchami .....</b>	<b>123</b>
6.1. Przechodzenie pomiędzy znakami łańcucha	124
6.2. Umieszczanie apostrofów w stałych łańcuchowych	126
6.3. Zliczanie wystąpień znaku w łańcuchu wejściowym	127
6.4. Usuwanie z łańcucha niechcianych znaków	128
6.5. Oddzielanie danych numerycznych od danych znakowych	130
6.6. Określanie, czy łańcuch jest ciągiem alfanumerycznym	134
6.7. Określanie inicjałów na podstawie całych imion i nazwisk	138
6.8. Sortowanie kolumn według wybranych fragmentów łańcuchów	143
6.9. Sortowanie danych według liczb zapisanych w łańcuchach	144
6.10. Tworzenie listy wartości oddzielonych przecinkami z danych zawartych w wierszach tabeli	150
6.11. Konwertowanie danych oddzielonych przecinkami na wielowartościową listę IN	154
6.12. Sortowanie znaków w łańcuchach w porządku alfabetycznym	160
6.13. Identyfikowanie łańcuchów, które można traktować jak liczby	164
6.14. Odnajdywanie n-tego podłańcucha na liście oddzielonej przecinkami	171
6.15. Przetwarzanie adresów IP	177
6.16. Porównywanie łańcuchów znaków na podstawie brzmienia	180
6.17. Wyszukiwanie tekstu niepasującego do wzorca	181
6.18. Podsumowanie	184
<b>7. Praca z liczbami .....</b>	<b>185</b>
7.1. Wyznaczanie wartości średniej	185
7.2. Identyfikacja minimalnej i maksymalnej wartości w kolumnie	187
7.3. Sumowanie wartości składowanych w kolumnie	189
7.4. Zliczanie wierszy tabeli	191
7.5. Zliczanie różnych wartości w kolumnie	193
7.6. Generowanie sum bieżących	194
7.7. Generowanie iloczynów bieżących	195
7.8. Wygładzanie serii wartości	196
7.9. Wyznaczanie wartości modalnej (dominanty)	198
7.10. Wyznaczanie mediany	200
7.11. Określanie procentowego udziału w wartości łącznej	203
7.12. Agregowanie kolumn zawierających wartości NULL	205
7.13. Wyznaczanie wartości średnich z wyłączeniem wartości spoza określonego przedziału	207
7.14. Wyodrębnianie liczb z łańcuchów alfanumerycznych	208
7.15. Modyfikowanie wartości uwzględnianych w sumach bieżących	211
7.16. Znajdowanie wartości odstających metodą mediany odchylenia bezwzględnego	213
7.17. Wyszukiwanie anomalii przy użyciu prawa Benforda	217
7.18. Podsumowanie	218

<b>8. Działania na datach .....</b>	<b>219</b>
8.1. Dodawanie i odejmowanie dni, miesięcy i lat	219
8.2. Określanie liczby dni pomiędzy dwiema datami	222
8.3. Określanie liczby dni roboczych pomiędzy dwiema datami	224
8.4. Określanie liczby miesięcy lub lat dzielących dwie daty	229
8.5. Określanie liczby sekund, minut lub godzin dzielących dwie daty	231
8.6. Zliczanie wystąpień poszczególnych dni tygodnia w roku	233
8.7. Określanie różnicy dat między rekordem bieżącym a następnym	244
8.8. Podsumowanie	250
<b>9. Przetwarzanie dat .....</b>	<b>251</b>
9.1. Określanie, czy dany rok jest rokiem przestępnym	252
9.2. Określanie liczby dni w roku	258
9.3. Wyodrębnianie jednostek czasu z dat wejściowych	261
9.4. Określanie pierwszego i ostatniego dnia miesiąca	263
9.5. Określanie wszystkich dat występowania konkretnego dnia tygodnia w ciągu danego roku	266
9.6. Określanie dat pierwszego i ostatniego wystąpienia określonego dnia tygodnia w danym miesiącu	272
9.7. Tworzenie kalendarza	279
9.8. Generowanie dat rozpoczynających i kończących poszczególne kwartały danego roku	292
9.9. Określanie daty początkowej i końcowej dla danego kwartału	296
9.10. Uzupełnianie brakujących dat	304
9.11. Przeszukiwanie według określonych jednostek czasu	311
9.12. Porównywanie rekordów według określonych fragmentów dat	313
9.13. Identyfikacja wzajemnie pokrywających się przedziałów czasowych	316
9.14. Podsumowanie	321
<b>10. Praca z przedziałami .....</b>	<b>323</b>
10.1. Lokalizowanie przedziałów w ramach ciągów wartości	323
10.2. Odnajdywanie różnic pomiędzy wierszami należącymi do tej samej grupy lub partycji danych	326
10.3. Lokalizowanie początków i końców przedziałów wartości następujących bezpośrednio po sobie	332
10.4. Uzupełnianie brakujących wartości w przedziale	335
10.5. Generowanie kolejnych wartości liczbowych	339
10.6. Podsumowanie	342

<b>11. Zaawansowane przeszukiwanie .....</b>	<b>343</b>
11.1. Podział zbioru wynikowego na strony	343
11.2. Pomijanie n wierszy tabeli	345
11.3. Stosowanie logiki alternatywy w zapytaniach wykorzystujących złączenia zewnętrzne	347
11.4. Identyfikacja par odwrotnych w przetwarzanym zbiorze wierszy	349
11.5. Wybieranie n pierwszych rekordów	351
11.6. Odnajdywanie rekordów z największymi i najmniejszymi wartościami	352
11.7. Badanie „przyszłych” wierszy	353
11.8. Przenoszenie wartości wierszy	356
11.9. Tworzenie rankingu rezultatów	358
11.10. Eliminowanie powtórzeń	359
11.11. Odnajdywanie wartości skoczka	361
11.12. Generowanie prostych prognoz	367
11.13. Podsumowanie	374
<b>12. Raportowanie i przekształcanie danych .....</b>	<b>375</b>
12.1. Konwertowanie zbioru wynikowego do postaci pojedynczego wiersza	375
12.2. Konwertowanie zbioru wynikowego do postaci zbioru wielowierszowego	378
12.3. Odwrotna transpozycja zbioru wynikowego	382
12.4. Odwrotna transpozycja zbioru danych do postaci zbioru jednokolumnowego	385
12.5. Eliminowanie powtórzeń ze zbioru wynikowego	388
12.6. Przekształcanie zbioru wynikowego w celu ułatwienia obliczeń na wierszach	390
12.7. Tworzenie bloków danych tej samej wielkości	392
12.8. Tworzenie predefiniowanej liczby bloków danych	394
12.9. Tworzenie histogramów poziomych	395
12.10. Tworzenie histogramów pionowych	397
12.11. Zwracanie zbiorów wynikowych bez kolumn wykorzystywanych w procesie grupowania	399
12.12. Wyznaczanie prostych sum częściowych	402
12.13. Wyznaczanie sum częściowych dla wszystkich możliwych kombinacji wyrażeń	405
12.14. Identyfikowanie wierszy niebędących sumami częściowymi	415
12.15. Konwertowanie wierszy na wersję bitową za pomocą wyrażeń CASE	417
12.16. Tworzenie tzw. macierzy rzadkich	419
12.17. Grupowanie wierszy według określonych jednostek czasu	420
12.18. Jednoczesne agregowanie danych według różnych grup i bloków	424
12.19. Agregowanie zmiennych (ruchomych) przedziałów wartości	426
12.20. Obracanie zbioru wynikowego zawierającego sumy częściowe	434
12.21. Podsumowanie	438

<b>13. Zapytania hierarchiczne .....</b>	<b>439</b>
13.1. Wyrażanie relacji rodzic – potomek	440
13.2. Wyrażanie relacji potomek – rodzic – dziadek	443
13.3. Tworzenie hierarchicznego widoku tabeli	448
13.4. Odnajdywanie wszystkich wierszy potomnych dla danego wiersza rodzica	452
13.5. Określanie wierszy występujących w rolach liści, gałęzi i korzeni	454
13.6. Podsumowanie	461
<b>14. Rozmaitości .....</b>	<b>463</b>
14.1. Tworzenie raportów krzyżowych za pomocą operatora PIVOT systemu SQL Server	463
14.2. Odtwarzanie układu raportów krzyżowych za pomocą operatora UNPIVOT systemu SQL Server	465
14.3. Transponowanie zbiorów wynikowych za pomocą klauzuli MODEL systemu Oracle	467
14.4. Wyodrębnianie z łańcucha elementów o nieustalonym położeniu	471
14.5. Znajdowanie liczby dni w roku (rozwiazanie alternatywne tylko dla systemu Oracle)	474
14.6. Przeszukiwanie danych wejściowych pod kątem zawierania łańcuchów alfanumerycznych	475
14.7. Konwertowanie liczb całkowitych na system dwójkowy w systemie Oracle	477
14.8. Obracanie zbioru wynikowego z wartościami rankingowymi	481
14.9. Wstawianie nagłówek kolumn w dwukrotnie obróconych zbiorach wynikowych	485
14.10. Konwertowanie podzapytań skalarnych na podzapytania złożone w systemie Oracle	496
14.11. Przenoszenie uszeregowanych danych do osobnych wierszy	498
14.12. Wyznaczanie procentowych stosunków poszczególnych wartości względem sumy wszystkich wartości	502
14.13. Testowanie występowania wartości w grupie	504
14.14. Podsumowanie	508
<b>A Przypomnienie funkcji okna .....</b>	<b>509</b>
<b>B Wyrażenia tablicowe (CTE) .....</b>	<b>535</b>



# Odczytywanie rekordów

W niniejszym rozdziale skupimy się na podstawowych wyrażeniach SELECT. Opanowanie podstaw jest ważne, gdyż wiele omówionych w tym rozdziale zagadnień przewija się nie tylko w trudniejszych recepturach, ale też w codziennej pracy z językiem SQL.

## 1.1. Odczytywanie wszystkich wierszy i kolumn tabeli

### Problem

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

### Rozwiązanie

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

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

### Omówienie

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

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

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

zapytaniem `SELECT *` mogą się też pojawić w przypadku odwołania się programu do innego zbioru kolumn, niż oczekiwaliśmy. Jeśli wymienimy wszystkie kolumny i okaże się, że któregoś (lub którychś) brakuje, otrzymany komunikat błędu ułatwi namierzenie owych braków.

## 1.2. Odczytywanie podzbioru wierszy tabeli

### Problem

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

### Rozwiązanie

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

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

### Omówienie

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

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

## 1.3. Odnajdywanie wierszy spełniających wiele warunków

### Problem

Chcemy uzyskać wiersze, które spełniają wiele warunków.

### Rozwiązanie

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

```

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

```

## Omówienie

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

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

Warunki ujęte w nawiasy są analizowane łącznie.

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

```

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

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-2005	800		20
7876	ADAMS	CLERK	7788	12-JAN-2008	1100		20

## 1.4. Odczytywanie podzbioru kolumn tabeli

### Problem

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

### Rozwiązanie

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

```

1 select ename, deptno, sal
2   from emp

```

## Omówienie

Wymieniając poszczególne kolumny w klauzuli SELECT, możemy zagwarantować, że zbiór wyników nie będzie zawierał żadnych dodatkowych, zbędnych danych. Tego rodzaju pewność jest szczególnie istotna w sytuacji, gdy zapytania są kierowane do bazy danych za pośrednictwem sieci, ponieważ eliminuje to opóźnienia wynikające z wyszukiwania i przesyłania niepotrzebnych danych.

## 1.5. Definiowanie sensownych nazw kolumn

### Problem

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

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

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

### Rozwiązanie

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

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

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

## Omówienie

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

# 1.6. Odwołania do aliasów kolumn w klauzuli WHERE

## Problem

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

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

## Rozwiązanie

Aby zyskać możliwość odwoływania się do kolumn reprezentowanych przez aliasy, trzeba zastosować tzw. widok wewnętrzny (ang. *inline view*), określanej niekiedy mianem perspektywy wewnętrznej lub wbudowanej:

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

## Omówienie

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

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

Wróćmy do naszego przykładowego zapytania, w którym użyliśmy aliasu. Okazuje się, że aliasy przypisane kolumnom w podzapytaniu mogą być przedmiotem odwołań w zapytaniu zewnętrznym. Dlaczego należy postąpić w taki sposób? Otóż klauzula WHERE jest interpretowana przed klauzulą SELECT, zatem w chwili interpretowania klauzuli WHERE w oryginalnym zapytaniu („Problem”) aliasy SALARY i COMMISSION jeszcze nie istnieją. Powstają one dopiero po zakończeniu przetwarzania klauzuli WHERE. Z kolei klauzula FROM jest przetwarzana przed klauzulą WHERE. Umieszczając oryginalne zapytanie w klauzuli FROM, spowodujemy, że niezbędne aliasy zostaną zdefiniowane przed osiągnięciem skrajnie zewnętrznej klauzuli WHERE, zatem będą widoczne dla tej klauzuli. Opisaną techniką jest szczególnie przydatna w sytuacji, gdy nazwy kolumny tabeli nie zostały zbyt szczęśliwie dobrane.



Widok wewnętrzny w prezentowanym rozwiązaniu jest reprezentowany przez alias X. Nie wszystkie bazy danych wymagają reprezentowania zapytań wewnętrznych za pomocą definiowanych wprost aliasów, ale wszystkie bazy danych akceptują tego rodzaju zabiegi składniowe.

## 1.7. Konkatenacja wartości kolumn

### Problem

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

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

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

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

ENAME	JOB
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

### Rozwiązanie

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

### DB2, Oracle, PostgreSQL

W tych systemach baz danych operatorem konkatenacji jest para pionowych kresek (||):

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

### MySQL

Baza danych MySQL oferuje funkcję nazwaną CONCAT:

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

## SQL Server

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

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

## Omówienie

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

# 1.8. Stosowanie logiki warunkowej w wyrażeniu SELECT

## Problem

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

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

## Rozwiązanie

Musimy użyć wyrażenia CASE, za pośrednictwem którego możemy zdefiniować instrukcje warunkowe bezpośrednio w zapytaniu SELECT:

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

## Omówienie

Wyrażenie CASE umożliwia definiowanie rozmaitych instrukcji warunkowych, wykorzystujących wartości zwracane przez zapytanie. Istnieje możliwość przypisania do wyrażenia CASE aliasu, który zapewni większą czytelność generowanego zbioru wynikowego. W prezentowanym przykładzie nadano wynikowi wyrażenia CASE alias STATUS. Klauzula ELSE ma charakter opcjonalny. Jeśli zrezygnujemy z tej klauzuli i dany wiersz nie zostanie dopasowany do żadnego z badanych kryteriów, wyrażenie CASE zwróci wartość NULL.

## 1.9. Ograniczanie liczby zwracanych wierszy

### Problem

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

### Rozwiązanie

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

#### DB2

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

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

#### MySQL i PostgreSQL

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

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

#### Oracle

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

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

#### SQL Server

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

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



## Omówienie

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

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

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

Z powyższego opisu wynika, że funkcja `ROWNUM` systemu Oracle przypisuje wartość odpowiedniemu licznikowi *po* odczytaniu kolejnego wiersza. Sekwencja działań jest w tym przypadku niezwykle istotna. Wielu programistów baz danych Oracle próbuje zwracać np. tylko piąty wiersz za pomocą wyrażenia `ROWNUM = 5`.

Stosowanie znaku równości w połączeniu z `ROWNUM` nie jest jednak dobrym pomysłem. Oto co się stanie, gdy spróbujemy zwrócić piąty wiersz za pomocą wyrażenia `ROWNUM = 5`:

1. System Oracle zaczyna wykonywać zapytanie.
2. System Oracle odczytuje pierwszy wiersz i przypisuje licznikowi wartość 1.
3. Czy dotarliśmy już do piątego wiersza? Jeśli nie, system Oracle pomija dany wiersz, ponieważ zdefiniowane w klauzuli `WHERE` kryterium liczby wierszy równej 5 nie jest spełnione. Jeśli tak, system Oracle zwróci dany wiersz. Ale odpowiedź nigdy nie brzmi „tak”!
4. System Oracle odczytuje kolejny wiersz ze wskazanej tabeli i przypisuje mu numer 1. Wynika to z faktu, że pierwszy wiersz zwracany przez zapytanie zawsze musi być oznaczany numerem 1.
5. Powrót do kroku trzeciego.

Wystarczy uważnie przeanalizować opisaną powyżej procedurę, aby się przekonać, dlaczego stosowanie warunku `ROWNUM = 5` nie przyniesie spodziewanych rezultatów. Nie otrzymamy przecież piątego wiersza, jeśli najpierw nie zostaną zwrócone wiersze od pierwszego do czwartego!

Ponadto warto zauważyć, że warunek `ROWNUM = 1` spowoduje zwrócenie pierwszego wiersza, co pozornie jest sprzeczne z powyższymi wyjaśnieniami. Otóż w przypadku pierwszego wiersza warunek `ROWNUM = 1` zadziała, bo w celu sprawdzenia, czy wskazana tabela zawiera jakieś wiersze, Oracle musi podjąć przynajmniej jedną próbę pobrania ich. Zachęcamy do ponownego przejrzenia powyższych punktów i zastąpienia wartości 5 jednością; to dobry sposób na przekonanie się, dlaczego warunek `ROWNUM = 1` jest poprawny (w przypadku chęci zwrócenia jednego wiersza).

# 1.10. Zwracanie n losowych rekordów tabeli

## Problem

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

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

## Rozwiązanie

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

### DB2

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

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

### MySQL

Należy użyć wbudowanej funkcji `RAND` łącznie z klauzulami `LIMIT` i `ORDER BY`:

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

### PostgreSQL

Powinniśmy użyć wbudowanej funkcji `RANDOM` łącznie z klauzulami `LIMIT` i `ORDER BY`:

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

### Oracle

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

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

## SQL Server

Aby uzyskać losowy pięcioelementowy zbiór wynikowy, należy użyć wbudowanej funkcji NEWID w połączeniu z klauzulami TOP oraz ORDER BY:

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

## Omówienie

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

W żadnym razie nie należy mylić stosowania funkcji w klauzuli ORDER BY ze stosowaniem stałych numerycznych. Podanie takiej stałej (numera) w klauzuli ORDER BY oznacza, że chcemy posortować dane zgodnie z kolumną o tym numerze na liście klauzuli SELECT. Umieszczenie w klauzuli ORDER BY wywołania jakiejś funkcji sprawi, że operacja sortowania zostanie wykonana zgodnie z rezultatami zwróconymi przez tę funkcję dla każdego kolejnego wiersza.

# 1.11. Odnajdywanie wartości pustych (NULL)

## Problem

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

## Rozwiązanie

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

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

## Omówienie

Ponieważ wartość NULL nigdy nie jest ani równa czemukolwiek, ani różna od czegokolwiek (nawet od samej siebie), nie możemy za pomocą standardowych operatorów = lub != sprawdzić, czy dana kolumna zawiera NULL. Aby się o tym przekonać, należy użyć wyrażenia IS NULL. Odnajdywanie wierszy, które nie zawierają wartości pustych w określonej kolumnie, wymaga zastosowania w klauzuli WHERE wyrażenia IS NOT NULL.

## 1.12. Przekształcanie wartości pustych w rzeczywiste

### Problem

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

### Rozwiązanie

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

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

### Omówienie

Funkcja COALESCE przyjmuje co najmniej jeden argument i zwraca z listy tych argumentów pierwszą wartość różną od NULL. Oznacza to, że w prezentowanym przykładzie wartość kolumny COMM jest zwracana pod warunkiem, że nie jest wartością pustą. W przeciwnym razie funkcja zwraca zero.

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

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

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

## 1.13. Poszukiwanie wzorców

### Problem

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

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

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

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

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

## Rozwiązanie

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

```

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

## Omówienie

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

## 1.14. Podsumowanie

Przedstawione receptury są proste, lecz mają fundamentalne znaczenie. Pobieranie informacji jest podstawą tworzenia zapytań do baz danych, a to oznacza, że podane rozwiązania stanowią serce wszystkich innych zagadnień omówionych w dalszej części tej książki.



# 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** 

# SQL: dokładnie to, czego potrzebujesz!

SQL jest *lingua franca* dla profesjonalistów zajmujących się przetwarzaniem danych. To wyjątkowo dojrzałe narzędzie, z którego korzysta już kilka pokoleń inżynierów i programistów. Wydaje się jednak, że zbyt często ten język nie jest należycie wykorzystywany: wielu użytkowników posługuje się nim na co dzień, ale mało kto wychodzi poza stosunkowo proste zapytania. Tymczasem z możliwości nowoczesnego SQL mogą skorzystać nie tylko osoby zajmujące się bazami danych, ale także analitycy danych, analitycy biznesowi, statystycy czy osoby zajmujące się wizualizacją danych.

To drugie, gruntownie zaktualizowane wydanie nieoczonego zbioru receptur, podanych tak, aby maksymalnie ułatwić rozwiązywanie codziennych problemów związanych z budową zapytań SQL. Uwzględniono tu kilka wariantów języka SQL, zaimplementowanych w systemach: Oracle, DB2, SQL Server, MySQL i PostgreSQL. W książce znalazły się propozycje zastosowania takich nowoczesnych rozwiązań jak funkcje okna, wspólne wyrażenia tablicowe i rekurencyjne zapytania hierarchiczne. Nie zabrakło receptur opracowanych specjalnie dla analityków danych, umożliwiających zastosowanie mediany odchylenia bezwzględne i prawa Benforda, a także wyszukiwanie danych tekstowych na podstawie brzmienia słów.

**W książce znajdziesz receptury, dzięki którym:**

- łatwiej opanujesz podstawy SQL
- dobrze wykorzystasz funkcje okna
- zastosujesz wspólne wyrażenia tablicowe (CTE) do tworzenia czytelniejszych rozwiązań
- zwiększysz użyteczność SQL w analizie danych
- skuteczniej obsłużysz dane liczbowe i ciągi znaków

**Anthony Molinaro** jest analitykiem danych w Johnson & Johnson. Zajmuje się metodami nieparametrycznymi, analizą szeregów czasowych i wielkoskalowymi bazami danych. Jest członkiem otwartej społeczności naukowej OHDSI.

**Robert de Graaf** jest starszym analitykiem danych w firmie RightShip. Wcześniej pracował w przemyśle wytwórczym. Jest zafascynowany potęgą statystyki w rozwiązywaniu praktycznych problemów.

**Helion** 

 [helion.pl](http://helion.pl)

 **HELION SA**  
ul. Kościuszki 1c  
44-100 Gliwice  
tel.: 32 230 98 63  
[helion@helion.pl](mailto:helion@helion.pl)

Sprawdź nasze szkolenia!

**SZKOLENIA**



**AKADEMIA IT & BUSINESS**

[HELIONSZKOLENIA.PL](http://HELIONSZKOLENIA.PL)

**KOD KORZYŚCI**  
Sięgnij po więcej! ▶



ISBN 978-83-283-7847-6



9 788328 378476

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 129,00 zł