



Systemy baz danych

Kompletny podręcznik

PIERWSZE TAK SZCZEGÓŁOWE, KOMPLEKSOWE WPROWADZENIE!
TRZEJ ZNANI NAUKOWCY Z OZIECZINY IT NAUCZĄ CIĘ:

- profesjonalnego projektowania i korzystania z baz danych
- tworzenia i wdrażania złożonych aplikacji bazodanowych
- sprawniej implementacji systemów zarządzania bazami danych

Hector Garcia-Molina,
Jeffrey D. Ullman, Jennifer Widom

Prentice
Hall

Wydanie II



» Idź do

- Spis treści
- Przykładowy rozdział
- Skorowidz

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 32 230 98 63
e-mail: helion@helion.pl
© Helion 1991–2011

Systemy baz danych. Kompletny podręcznik. Wydanie II

Autorzy: [Hector Garcia-Molina](#), [Jeffrey D. Ullman](#), [Jennifer Widom](#)

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-246-3303-6

Tytuł oryginału: [Database Systems: The Complete Book \(2nd Edition\)](#)

Format: 172×245, stron: 1048



Pierwsze tak szczegółowe, kompleksowe wprowadzenie!

Trzej znani naukowcy z dziedziny IT nauczą Cię:

- profesjonalnego projektowania i korzystania z baz danych
- tworzenia i wdrażania złożonych aplikacji bazodanowych
- sprawnej implementacji systemów zarządzania bazami danych

Z kluczowej roli, jaką bazy danych odgrywają w codziennym życiu milionów ludzi, zdajemy sobie sprawę za każdym razem, gdy wpisujemy hasło w wyszukiwarce Google, robimy zakupy w internetowej księgarni czy logujemy się do swojego konta w banku. Szybkie, bezpieczne i niezawodne przetwarzanie oraz przechowywanie ogromnych ilości informacji stało się dziś strategicznym czynnikiem funkcjonowania większości firm, organizacji i instytucji państwowych. Ten ogromny potencjał współczesnych baz danych jest dziś sumą wiedzy i technologii rozwijanych przez kilka ostatnich dziesięcioleci. Owocem tych prac jest przede wszystkim wyspecjalizowane oprogramowanie – systemy zarządzania bazami danych DBMS, czyli rozbudowane narzędzia do wydajnego tworzenia dużych zbiorów informacji i zarządzania nimi. Niestety, mają one jedną zasadniczą wadę – należą do najbardziej złożonych rodzajów oprogramowania!

W związku z tym trzech znanych naukowców w dziedzinie IT z Uniwersytetu Stanforda. Hector Garcia-Molina, Jeffrey D. Ullman i Jennifer Widom – postanowiło stworzyć pierwszy tak kompletny podręcznik, wprowadzający do systemów baz danych. Zawiera on opis najnowszych standardów bazy danych SQL 1999, SQL/PSM, SQL/CLI, JDBC, ODL oraz XML – i to w znacznie szerszym zakresie niż w większości publikacji. Podręcznik został przygotowany w taki sposób, aby po jego przeczytaniu użytkownik czy projektowanie baz danych, pisanie programów w różnych językach związanych z systemami DBMS oraz ich sprawna implementacja nie stanowiły dla Czytelnika najmniejszego problemu!

Kompleksowe podejście do systemów baz danych z punktu widzenia projektanta, użytkownika i programisty

Spis treści

Wstęp	25
O autorach	29
1. Świat systemów baz danych	31
1.1. Ewolucja systemów baz danych	31
1.1.1. Wczesne systemy zarządzania bazami danych	32
1.1.2. Relacyjne systemy baz danych	32
1.1.3. Coraz mniejsze systemy	33
1.1.4. Coraz większe systemy	33
1.1.5. Integrowanie informacji	34
1.2. Elementy systemu zarządzania bazami danych	34
1.2.1. Polecenia w języku definicji danych	34
1.2.2. Omówienie przetwarzania zapytań	35
1.2.3. Zarządzanie pamięcią i buforem	36
1.2.4. Przetwarzanie transakcji	37
1.2.5. Procesor zapytań	38
1.3. Zarys dziedziny systemów baz danych	38
1.4. Literatura do rozdziału 1.	40
I Modelowanie relacyjnych baz danych	41
2. Relacyjny model danych	43
2.1. Przegląd modeli danych	43
2.1.1. Czym jest model danych?	43
2.1.2. Ważne modele danych	44
2.1.3. Krótki opis modelu relacyjnego	44
2.1.4. Krótki opis modelu semistrukturalnego	45
2.1.5. Inne modele danych	46
2.1.6. Porównanie podejść do modelowania	46
2.2. Podstawy modelu relacyjnego	47
2.2.1. Atrybuty	47
2.2.2. Schematy	47
2.2.3. Krotki	48
2.2.4. Dziedziny	48
2.2.5. Równoważne reprezentacje relacji	49

2.2.6. Egzemplarze relacji	49
2.2.7. Klucze relacji	49
2.2.8. Przykładowy schemat bazy danych	50
2.2.9. Ćwiczenia do podrozdziału 2.2	52
2.3. Definiowanie schematu relacji w języku SQL	53
2.3.1. Relacje w SQL-u	53
2.3.2. Typy danych	54
2.3.3. Proste deklaracje tabel	55
2.3.4. Modyfikowanie schematów relacji	56
2.3.5. Wartości domyślne	57
2.3.6. Deklarowanie kluczy	57
2.3.7. Ćwiczenia do podrozdziału 2.3	59
2.4. Algebraiczny język zapytań	60
2.4.1. Dlaczego potrzebujemy specjalnego języka zapytań?	61
2.4.2. Czym jest algebra?	61
2.4.3. Przegląd algebry relacji	61
2.4.4. Operacje specyficzne dla zbiorów wykonywane na relacjach	62
2.4.5. Projekcja	63
2.4.6. Selekcja	64
2.4.7. Iloczyn kartezjański	65
2.4.8. Złączenia naturalne	65
2.4.9. Złączenia warunkowe	68
2.4.10. Łączenie operacji w celu tworzenia zapytań	69
2.4.11. Nadawanie i zmienianie nazw	70
2.4.12. Zależności między operacjami	71
2.4.13. Notacja liniowa wyrażeń algebraicznych	72
2.4.14. Ćwiczenia do podrozdziału 2.4	73
2.5. Więzy relacji	79
2.5.1. Algebra relacji jako język więzów	79
2.5.2. Więzy integralności referencyjnej	80
2.5.3. Więzy klucza	81
2.5.4. Inne przykłady dotyczące więzów	82
2.5.5. Ćwiczenia do podrozdziału 2.5	82
2.6. Podsumowanie rozdziału 2.	83
2.7. Literatura do rozdziału 2.	84
3. Teoria projektowania relacyjnych baz danych	85
3.1. Zależności funkcyjne	85
3.1.1. Definicja zależności funkcyjnej	86
3.1.2. Klucze relacji	87
3.1.3. Nadklucze	88
3.1.4. Ćwiczenia do podrozdziału 3.1	89
3.2. Reguły dotyczące zależności funkcyjnych	89
3.2.1. Wnioskowanie na temat zależności funkcyjnych	90
3.2.2. Reguła podziału i łączenia	90
3.2.3. Trywialne zależności funkcyjne	91
3.2.4. Obliczanie domknięcia atrybutów	92
3.2.5. Dlaczego algorytm obliczania domknięć działa?	94
3.2.6. Reguła przechodniości	96
3.2.7. Domykanie zbiorów zależności funkcyjnych	97
3.2.8. Projekcje zależności funkcyjnych	98
3.2.9. Ćwiczenia do podrozdziału 3.2	99

3.3. Projektowanie schematów relacyjnych baz danych	101
3.3.1. Anomalie	101
3.3.2. Dekompozycja relacji	102
3.3.3. Postać normalna Boyce'a-Codda	103
3.3.4. Dekompozycja do BCNF	104
3.3.5. Ćwiczenia do podrozdziału 3.3	107
3.4. Dekompozycja — dobre, złe i okropne skutki	108
3.4.1. Odzyskiwanie informacji po dekompozycji	108
3.4.2. Test dla złączeń bezstratnych oparty na algorytmie chase	111
3.4.3. Dlaczego algorytm chase działa?	113
3.4.4. Zachowywanie zależności	114
3.4.5. Ćwiczenia do podrozdziału 3.4	115
3.5. Trzecia postać normalna	116
3.5.1. Definicja trzeciej postaci normalnej	116
3.5.2. Algorytm syntezy do schematów o trzeciej postaci normalnej	117
3.5.3. Dlaczego algorytm syntezy do 3NF działa?	118
3.5.4. Ćwiczenia do podrozdziału 3.5	118
3.6. Zależności wielowartościowe	119
3.6.1. Niezależność atrybutów i wynikająca z tego nadmiarowość	119
3.6.2. Definicja zależności wielowartościowej	120
3.6.3. Wnioskowanie na temat ZW	121
3.6.4. Czwarta postać normalna	123
3.6.5. Dekompozycja do czwartej postaci normalnej	124
3.6.6. Związki między postaciami normalnymi	125
3.6.7. Ćwiczenia do podrozdziału 3.6	126
3.7. Algorytm wyprowadzania ZW	127
3.7.1. Domknięcie i algorytm chase	127
3.7.2. Rozszerzanie algorytmu chase na ZW	128
3.7.3. Dlaczego algorytm chase działa dla ZW?	130
3.7.4. Projekcje ZW	131
3.7.5. Ćwiczenia do podrozdziału 3.7	132
3.8. Podsumowanie rozdziału 3.	132
3.9. Literatura do rozdziału 3.	134
4. Wysokopoziomowe modele baz danych	135
4.1. Model związków encji	136
4.1.1. Zbiory encji	136
4.1.2. Atrybuty	136
4.1.3. Związki	137
4.1.4. Diagramy ER	137
4.1.5. Egzemplarze diagramów ER	138
4.1.6. Krotność w związkach binarnych w modelu ER	139
4.1.7. Związki wieloargumentowe	139
4.1.8. Role w związkach	140
4.1.9. Atrybuty związków	142
4.1.10. Przekształcanie związków wieloargumentowych na binarne	143
4.1.11. Podklasy w modelu ER	144
4.1.12. Ćwiczenia do podrozdziału 4.1	146
4.2. Zasady projektowania	148
4.2.1. Wierność	148
4.2.2. Unikanie nadmiarowości	149
4.2.3. Prostota ma znaczenie	149

4.2.4. Wybór odpowiednich związków	149
4.2.5. Wybór elementów odpowiedniego rodzaju	151
4.2.6. Ćwiczenia do podrozdziału 4.2	153
4.3. Więzy w modelu ER	155
4.3.1. Klucze w modelu ER	155
4.3.2. Reprezentowanie kluczy w modelu ER	155
4.3.3. Integralność referencyjna	156
4.3.4. Więzy stopni	157
4.3.5. Ćwiczenia do podrozdziału 4.3	157
4.4. Słabe zbiory encji	158
4.4.1. Przyczyny tworzenia słabych zbiorów encji	158
4.4.2. Wymogi związane ze słabymi zbiorami encji	160
4.4.3. Notacja do opisu słabych zbiorów encji	161
4.4.4. Ćwiczenia do podrozdziału 4.4	161
4.5. Z diagramów ER na projekty relacyjne	162
4.5.1. Ze zbiorów encji na relacje	162
4.5.2. Ze związków ER na relacje	163
4.5.3. Łączenie relacji	165
4.5.4. Przekształcanie słabych zbiorów encji	166
4.5.5. Ćwiczenia do podrozdziału 4.5	168
4.6. Przekształcanie struktur podklas na relacje	169
4.6.1. Przekształcanie w stylu modelu ER	170
4.6.2. Podejście obiektowe	171
4.6.3. Stosowanie wartości null do łączenia relacji	171
4.6.4. Porównanie podejść	172
4.6.5. Ćwiczenia do podrozdziału 4.6	173
4.7. Język UML	174
4.7.1. Klasy w języku UML	174
4.7.2. Klucze klas w UML-u	175
4.7.3. Asocjacje	175
4.7.4. Asocjacje zwrotne	177
4.7.5. Klasy asocjacji	177
4.7.6. Podklasy w UML-u	178
4.7.7. Agregacje i kompozycje	179
4.7.8. Ćwiczenia do podrozdziału 4.7	180
4.8. Z diagramów UML na relacje	181
4.8.1. Podstawy przekształcania diagramów UML na relacje	181
4.8.2. Z podklas w UML-u na relacje	181
4.8.3. Z agregacji i kompozycji na relacje	182
4.8.4. Odpowiednik słabych zbiorów encji w UML-u	183
4.8.5. Ćwiczenia do podrozdziału 4.8	184
4.9. Język ODL	184
4.9.1. Deklaracje klas	184
4.9.2. Atrybuty w ODL-u	185
4.9.3. Związki w ODL-u	186
4.9.4. Związki zwrotne	186
4.9.5. Krotność związków	187
4.9.6. Typy w ODL-u	188
4.9.7. Podklasy w ODL-u	190
4.9.8. Deklarowanie kluczy w ODL-u	190
4.9.9. Ćwiczenia do podrozdziału 4.9	192

4.10. Z projektów w ODL-u na projekty relacyjne	192
4.10.1. Z klas w ODL-u na relacje	193
4.10.2. Atrybuty złożone w klasach	193
4.10.3. Przedstawianie atrybutów wartości w formie zbioru	194
4.10.4. Reprezentowanie innych konstruktorów typów	195
4.10.5. Przedstawianie związków w ODL-u	197
4.10.6. Ćwiczenia do podrozdziału 4.10	197
4.11. Podsumowanie rozdziału 4.	198
4.12. Literatura do rozdziału 4.	200

II Programowanie relacyjnych baz danych201

5. Algebraiczne i logiczne języki zapytań203

5.1. Operacje relacyjne na wielozbiorach	203
5.1.1. Dlaczego wielozbiory?	204
5.1.2. Suma, część wspólna i różnica dla wielozbiorów	205
5.1.3. Projekcje wielozbiorów	206
5.1.4. Selekcja na wielozbiorach	207
5.1.5. Iloczyn kartezjański wielozbiorów	207
5.1.6. Złączenia wielozbiorów	208
5.1.7. Ćwiczenia do podrozdziału 5.1	209
5.2. Dodatkowe operatory algebry relacji	210
5.2.1. Eliminowanie powtórzeń	211
5.2.2. Operatory agregacji	211
5.2.3. Grupowanie	212
5.2.4. Operator grupowania	213
5.2.5. Rozszerzanie operatora projekcji	214
5.2.6. Operator sortowania	215
5.2.7. Złączenia zewnętrzne	216
5.2.8. Ćwiczenia do podrozdziału 5.2	218
5.3. Logika relacji	219
5.3.1. Predykaty i atomy	219
5.3.2. Atomy arytmetyczne	220
5.3.3. Reguły i zapytania w Datalogu	220
5.3.4. Znaczenie reguł Datalogu	221
5.3.5. Predykaty ekstensjonalne i intensjonalne	223
5.3.6. Reguły Datalogu stosowane do wielozbiorów	224
5.3.7. Ćwiczenia do podrozdziału 5.3	225
5.4. Algebra relacji i Datalog	226
5.4.1. Operacje logiczne	226
5.4.2. Projekcja	227
5.4.3. Selekcja	228
5.4.4. Iloczyn kartezjański	230
5.4.5. Złączenia	230
5.4.6. Symulowanie w Datalogu operacji złożonych	231
5.4.7. Porównanie Datalogu i algebry relacji	232
5.4.8. Ćwiczenia do podrozdziału 5.4	233
5.5. Podsumowanie rozdziału 5.	234
5.6. Literatura do rozdziału 5.	235

6. SQL — język baz danych	237
6.1. Proste zapytania w SQL-u	238
6.1.1. Projekcje w SQL-u	240
6.1.2. Selekcja w SQL-u	242
6.1.3. Porównywanie łańcuchów znaków	243
6.1.4. Dopasowywanie wzorców w SQL-u	243
6.1.5. Data i czas	244
6.1.6. Wartości NULL i porównania z takimi wartościami	245
6.1.7. Wartość logiczna UNKNOWN	246
6.1.8. Porządkowanie danych wyjściowych	248
6.1.9. Ćwiczenia do podrozdziału 6.1	249
6.2. Zapytania obejmujące więcej niż jedną relację	251
6.2.1. Iloczyn kartezjański i złączenia w SQL-u	251
6.2.2. Jednoznaczne określanie atrybutów	252
6.2.3. Zmienne krotkowe	253
6.2.4. Przetwarzanie zapytań obejmujących wiele relacji	254
6.2.5. Suma, część wspólna i różnica zapytań	256
6.2.6. Ćwiczenia do podrozdziału 6.2	258
6.3. Podzapytania	259
6.3.1. Podzapytania zwracające wartości skalarne	260
6.3.2. Warunki dotyczące relacji	261
6.3.3. Warunki obejmujące krotki	262
6.3.4. Podzapytania skorelowane	263
6.3.5. Podzapytania w klauzulach FROM	264
6.3.6. Wyrażenia ze złączeniami w SQL-u	265
6.3.7. Złączenia naturalne	266
6.3.8. Złączenia zewnętrzne	267
6.3.9. Ćwiczenia do podrozdziału 6.3	269
6.4. Operacje na całych relacjach	271
6.4.1. Eliminowanie powtórzeń	271
6.4.2. Powtórzenia w sumach, częściach wspólnych i różnicach	272
6.4.3. Grupowanie i agregacja w SQL-u	273
6.4.4. Operatory agregacji	273
6.4.5. Grupowanie	274
6.4.6. Grupowanie, agregacja i wartości null	276
6.4.7. Klauzule HAVING	277
6.4.8. Ćwiczenia do podrozdziału 6.4	278
6.5. Modyfikowanie bazy danych	279
6.5.1. Wstawianie	279
6.5.2. Usuwanie	281
6.5.3. Aktualizowanie	282
6.5.4. Ćwiczenia do podrozdziału 6.5	283
6.6. Transakcje w SQL-u	284
6.6.1. Możliwość szeregowania operacji	284
6.6.2. Atomowość	286
6.6.3. Transakcje	286
6.6.4. Transakcje tylko do odczytu	287
6.6.5. Odczyt „brudnych danych”	288
6.6.6. Inne poziomy izolacji	291
6.6.7. Ćwiczenia do podrozdziału 6.6	292
6.7. Podsumowanie rozdziału 6.	293
6.8. Literatura do rozdziału 6.	294

7. Więzy i wyzwalacze	295
7.1. Klucze zwykłe i klucze obce	295
7.1.1. Deklarowanie więzów klucza obcego	296
7.1.2. Zachowywanie integralności referencyjnej	297
7.1.3. Odroczone sprawdzanie więzów	299
7.1.4. Ćwiczenia do podrozdziału 7.1	301
7.2. Więzy atrybutów i krotek	302
7.2.1. Więzy NOT NULL	302
7.2.2. Więzy CHECK atrybutów	303
7.2.3. Więzy CHECK krotek	304
7.2.4. Porównanie więzów krotek i atrybutów	305
7.2.5. Ćwiczenia do podrozdziału 7.2	306
7.3. Modyfikowanie więzów	307
7.3.1. Przypisywanie nazw więzom	307
7.3.2. Modyfikowanie więzów tabel	308
7.3.3. Ćwiczenia do podrozdziału 7.3	309
7.4. Asercje	309
7.4.1. Tworzenie asercji	310
7.4.2. Stosowanie asercji	310
7.4.3. Ćwiczenia do podrozdziału 7.4	311
7.5. Wyzwalacze	312
7.5.1. Wyzwalacze w SQL-u	313
7.5.2. Możliwości w zakresie projektowania wyzwalaczy	314
7.5.3. Ćwiczenia do podrozdziału 7.5	317
7.6. Podsumowanie rozdziału 7.	319
7.7. Literatura do rozdziału 7.	319
8. Widoki i indeksy	321
8.1. Widoki wirtualne	321
8.1.1. Deklarowanie widoków	321
8.1.2. Zapytania o widoki	323
8.1.3. Zmienianie nazw atrybutów	323
8.1.4. Ćwiczenia do podrozdziału 8.1	324
8.2. Modyfikowanie widoków	324
8.2.1. Usuwanie widoku	324
8.2.2. Widoki modyfikowalne	325
8.2.3. Wyzwalacze INSTEAD OF dla widoków	327
8.2.4. Ćwiczenia do podrozdziału 8.2	328
8.3. Indeksy w SQL-u	329
8.3.1. Cel stosowania indeksów	329
8.3.2. Deklarowanie indeksów	330
8.3.3. Ćwiczenia do podrozdziału 8.3	331
8.4. Wybieranie indeksów	331
8.4.1. Prosty model kosztów	331
8.4.2. Wybrane przydatne indeksy	332
8.4.3. Obliczanie najlepszych indeksów	333
8.4.4. Automatyczne wybieranie tworzonych indeksów	336
8.4.5. Ćwiczenia do podrozdziału 8.4	337
8.5. Widoki zmaterializowane	337
8.5.1. Przechowywanie widoku zmaterializowanego	338
8.5.2. Okresowa konserwacja widoków zmaterializowanych	339

8.5.3. Modyfikowanie zapytań w celu zastosowania widoków zmaterializowanych	340
8.5.4. Automatyczne tworzenie widoków zmaterializowanych	342
8.5.5. Ćwiczenia do podrozdziału 8.5	342
8.6. Podsumowanie rozdziału 8.	343
8.7. Literatura do rozdziału 8.	344
9. SQL w środowisku serwerowym	345
9.1. Architektura trójwarstwowa	345
9.1.1. Warstwa serwerów WWW	346
9.1.2. Warstwa aplikacji	347
9.1.3. Warstwa bazy danych	347
9.2. Środowisko SQL-a	348
9.2.1. Środowiska	348
9.2.2. Schematy	349
9.2.3. Katalogi	350
9.2.4. Klienci i serwery w środowisku SQL-a	350
9.2.5. Połączenia	351
9.2.6. Sesje	352
9.2.7. Moduły	352
9.3. Interfejs łączący SQL z językiem macierzystym	353
9.3.1. Problem niezgodności impedancji	354
9.3.2. Łączenie SQL-a z językiem macierzystym	355
9.3.3. Sekcja DECLARE	355
9.3.4. Używanie zmiennych wspólnych	356
9.3.5. Jednowierszowe instrukcje SELECT	357
9.3.6. Kursory	357
9.3.7. Modyfikowanie danych za pomocą kursora	359
9.3.8. Zabezpieczanie się przed jednoczesnymi aktualizacjami	360
9.3.9. Dynamiczny SQL	361
9.3.10. Ćwiczenia do podrozdziału 9.3	362
9.4. Procedury składowane	363
9.4.1. Tworzenie funkcji i procedur w PSM-ie	364
9.4.2. Wybrane proste formy instrukcji w PSM-ie	365
9.4.3. Instrukcje rozgałęziające	366
9.4.4. Zapytania w PSM-ie	367
9.4.5. Pętle w PSM-ie	368
9.4.6. Pętle FOR	370
9.4.7. Wyjątki w PSM-ie	371
9.4.8. Stosowanie funkcji i procedur PSM-a	373
9.4.9. Ćwiczenia do podrozdziału 9.4	373
9.5. Używanie interfejsu poziomego wywołań	375
9.5.1. Wprowadzenie do SQL/CLI	375
9.5.2. Przetwarzanie instrukcji	377
9.5.3. Pobieranie danych z wyników zapytania	378
9.5.4. Przekazywanie parametrów do zapytań	380
9.5.5. Ćwiczenia do podrozdziału 9.5	381
9.6. Interfejs JDBC	381
9.6.1. Wprowadzenie do JDBC	381
9.6.2. Tworzenie instrukcji w JDBC	382
9.6.3. Operacje na kursorach w JDBC	384
9.6.4. Przekazywanie parametrów	385
9.6.5. Ćwiczenia do podrozdziału 9.6	385

9.7. PHP	385
9.7.1. Podstawy PHP	386
9.7.2. Tablice	387
9.7.3. Biblioteka DB z repozytorium PEAR	387
9.7.4. Tworzenie połączenia z bazą danych za pomocą biblioteki DB	388
9.7.5. Wykonywanie instrukcji SQL-a	388
9.7.6. Operacje oparte na kursorze w PHP	389
9.7.7. Dynamiczny SQL w PHP	389
9.7.8. Ćwiczenia do podrozdziału 9.7	390
9.8. Podsumowanie rozdziału 9.	390
9.9. Literatura do rozdziału 9.	391
10. Zaawansowane zagadnienia z obszaru relacyjnych baz danych	393
10.1. Bezpieczeństwo i uwierzytelnianie użytkowników w SQL-u	393
10.1.1. Uprawnienia	394
10.1.2. Tworzenie uprawnień	395
10.1.3. Proces sprawdzania uprawnień	396
10.1.4. Przyznawanie uprawnień	397
10.1.5. Diagramy przyznanych uprawnień	398
10.1.6. Odbieranie uprawnień	400
10.1.7. Ćwiczenia do podrozdziału 10.1	403
10.2. Rekurencja w SQL-u	404
10.2.1. Definiowanie relacji rekurencyjnych w SQL-u	404
10.2.2. Problematiczne wyrażenia w rekurencyjnym SQL-u	407
10.2.3. Ćwiczenia do podrozdziału 10.2	409
10.3. Model obiektowo-relacyjny	410
10.3.1. Od relacji do relacji obiektowych	411
10.3.2. Relacje zagnieżdżone	411
10.3.3. Referencje	413
10.3.4. Podejście obiektowe a obiektowo-relacyjne	414
10.3.5. Ćwiczenia do podrozdziału 10.3	415
10.4. Typy definiowane przez użytkownika w SQL-u	415
10.4.1. Definiowanie typów w SQL-u	415
10.4.2. Deklaracje metod w typach UDT	417
10.4.3. Definicje metod	417
10.4.4. Deklarowanie relacji za pomocą typów UDT	418
10.4.5. Referencje	418
10.4.6. Tworzenie identyfikatorów obiektów dla tabel	419
10.4.7. Ćwiczenia do podrozdziału 10.4	421
10.5. Operacje na danych obiektowo-relacyjnych	421
10.5.1. Podążanie za referencjami	421
10.5.2. Dostęp do komponentów krotek o typie UDT	422
10.5.3. Generatory i modyfikatory	423
10.5.4. Sortowanie elementów o typie UDT	425
10.5.5. Ćwiczenia do podrozdziału 10.5	426
10.6. Techniki OLAP	427
10.6.1. OLAP i hurtownie danych	428
10.6.2. Aplikacje OLAP	428
10.6.3. Wielowymiarowe ujęcie danych w aplikacjach OLAP	429
10.6.4. Schemat gwiazdy	430
10.6.5. Podział i wycinki	432
10.6.6. Ćwiczenia do podrozdziału 10.6	434

10.7. Kostki danych	435
10.7.1. Operator kostki (CUBE)	435
10.7.2. Operator kostki w SQL-u	437
10.7.3. Ćwiczenia do podrozdziału 10.7	438
10.8. Podsumowanie rozdziału 10.	439
10.9. Literatura do rozdziału 10.	441

III Modelowanie i programowanie danych semistrukturalnych443

11. Semistrukturalny model danych445

11.1. Dane semistrukturalne	445
11.1.1. Uzasadnienie powstania modelu danych semistrukturalnych	445
11.1.2. Reprezentowanie danych semistrukturalnych	446
11.1.3. Integrowanie informacji za pomocą danych semistrukturalnych	447
11.1.4. Ćwiczenia do podrozdziału 11.1	449
11.2. XML	449
11.2.1. Znaczniki semantyczne	449
11.2.2. XML ze schematem i bez niego	450
11.2.3. Poprawny składniowo XML	450
11.2.4. Atrybuty	452
11.2.5. Atrybuty łączące elementy	452
11.2.6. Przestrzenie nazw	453
11.2.7. XML i bazy danych	454
11.2.8. Ćwiczenia do podrozdziału 11.2	455
11.3. Definicje typów dokumentu	455
11.3.1. Forma definicji DTD	456
11.3.2. Korzystanie z definicji DTD	458
11.3.3. Listy atrybutów	459
11.3.4. Identyfikatory i referencje	460
11.3.5. Ćwiczenia do podrozdziału 11.3	461
11.4. XML Schema	462
11.4.1. Struktura dokumentów XML Schema	462
11.4.2. Elementy	462
11.4.3. Typy złożone	463
11.4.4. Atrybuty	465
11.4.5. Typy proste z ograniczeniami	466
11.4.6. Klucze w XML Schema	467
11.4.7. Klucze obce w dokumentach XML Schema	469
11.4.8. Ćwiczenia do podrozdziału 11.4	471
11.5. Podsumowanie rozdziału 11.	471
11.6. Literatura do rozdziału 11.	472

12. Języki programowania dla formatu XML473

12.1. XPath	473
12.1.1. Model danych w języku XPath	473
12.1.2. Węzły dokumentu	474
12.1.3. Wyrażenia XPath	475
12.1.4. Względne wyrażenia XPath	476
12.1.5. Atrybuty w wyrażeniach XPath	476
12.1.6. Osie	477
12.1.7. Kontekst wyrażień	478
12.1.8. Symbole wieloznaczne	478

12.1.9. Warunki w wyrażeniach XPath	479
12.1.10. Ćwiczenia do podrozdziału 12.1	481
12.2. Język XQuery	483
12.2.1. Podstawy języka XQuery	484
12.2.2. Wyrażenia FLWR	484
12.2.3. Zastępowanie zmiennych ich wartościami	488
12.2.4. Złączenia w XQuery	489
12.2.5. Operatory porównywania w XQuery	490
12.2.6. Eliminowanie powtórzeń	491
12.2.7. Kwantyfikatory w XQuery	492
12.2.8. Agregacje	492
12.2.9. Rozgałęzianie w wyrażeniach XQuery	493
12.2.10. Sortowanie wyników zapytania	494
12.2.11. Ćwiczenia do podrozdziału 12.2	495
12.3. Język XSLT	496
12.3.1. Podstawy języka XSLT	496
12.3.2. Szablony	496
12.3.3. Pobieranie wartości z danych w formacie XML	497
12.3.4. Rekurencyjne stosowanie szablonów	498
12.3.5. Iteracje w XSLT	500
12.3.6. Instrukcje warunkowe w XSLT	501
12.3.7. Ćwiczenia do podrozdziału 12.3	502
12.4. Podsumowanie rozdziału 12.	503
12.5. Literatura do rozdziału 12.	504

IV Implementowanie systemów baz danych 505

13. Zarządzanie pamięcią drugiego stopnia 507

13.1. Hierarchia pamięci	507
13.1.1. Hierarchia pamięci	507
13.1.2. Transfer danych między poziomami	509
13.1.3. Pamięć nietrwała i trwała	509
13.1.4. Pamięć wirtualna	510
13.1.5. Ćwiczenia do podrozdziału 13.1	511
13.2. Dyski	511
13.2.1. Mechanika dysków	511
13.2.2. Kontroler dysku	513
13.2.3. Cechy operacji dostępu do dysku	513
13.2.4. Ćwiczenia do podrozdziału 13.2	515
13.3. Przyspieszanie dostępu do pamięci drugiego stopnia	516
13.3.1. Model przetwarzania oparty na wejściu-wyjściu	516
13.3.2. Porządkowanie danych według cylindrów	517
13.3.3. Używanie wielu dysków	517
13.3.4. Tworzenie dysków lustrzanych	518
13.3.5. Szeregowanie operacji dyskowych i algorytm windy	518
13.3.6. Wstępne pobieranie i buforowanie na dużą skalę	520
13.3.7. Ćwiczenia do podrozdziału 13.3	521
13.4. Problemy z dyskami	522
13.4.1. Nieregularne błędy	522
13.4.2. Sumy kontrolne	523
13.4.3. Pamięć stabilna	524

13.4.4. Możliwości obsługi błędów w pamięci stabilnej	524
13.4.5. Przywracanie danych po awarii dysku	525
13.4.6. Tworzenie kopii lustrzanych jako technika zapewniania nadmiarowości	525
13.4.7. Bloki parzystości	526
13.4.8. Usprawnienie — RAID 5	529
13.4.9. Obsługa awarii kilku dysków	529
13.4.10. Ćwiczenia do podrozdziału 13.4	532
13.5. Porządkowanie danych na dysku	534
13.5.1. Rekordy o stałej długości	535
13.5.2. Umieszczanie rekordów o stałej długości w blokach	536
13.5.3. Ćwiczenia do podrozdziału 13.5	537
13.6. Przedstawianie adresów bloków i rekordów	537
13.6.1. Adresy w systemach klient-serwer	537
13.6.2. Adresy logiczne i ustrukturyzowane	538
13.6.3. Przemiana wskaźników	539
13.6.4. Zapisywanie bloków z powrotem na dysku	542
13.6.5. Rekordy i bloki przyklejone	543
13.6.6. Ćwiczenia do podrozdziału 13.6	544
13.7. Dane i rekordy o zmiennej długości	545
13.7.1. Rekordy o polach o zmiennej długości	546
13.7.2. Rekordy z powtarzającymi się polami	546
13.7.3. Rekordy o zmiennym formacie	548
13.7.4. Rekordy, które nie mieszczą się w bloku	549
13.7.5. Obiekty BLOB	549
13.7.6. Zapisywanie kolumn	550
13.7.7. Ćwiczenia do podrozdziału 13.7	551
13.8. Modyfikowanie rekordów	552
13.8.1. Wstawianie	552
13.8.2. Usuwanie	553
13.8.3. Aktualizacje	554
13.8.4. Ćwiczenia do podrozdziału 13.8	555
13.9. Podsumowanie rozdziału 13.	555
13.10. Literatura do rozdziału 13.	557
14. Struktury indeksów	559
14.1. Podstawy struktur indeksów	560
14.1.1. Pliki sekwencyjne	560
14.1.2. Indeksy gęste	561
14.1.3. Indeksy rzadkie	562
14.1.4. Wiele poziomów indeksu	562
14.1.5. Indeksy pomocnicze	563
14.1.6. Zastosowania indeksów pomocniczych	564
14.1.7. Poziom pośredni w indeksach pomocniczych	565
14.1.8. Pobieranie dokumentów i indeksy odwrócone	567
14.1.9. Ćwiczenia do podrozdziału 14.1	570
14.2. Drzewa zbalansowane	571
14.2.1. Struktura drzew zbalansowanych	572
14.2.2. Zastosowania drzew zbalansowanych	574
14.2.3. Wyszukiwanie w drzewach zbalansowanych	576
14.2.4. Zapytania zakresowe	576
14.2.5. Wstawianie elementów do drzew zbalansowanych	577
14.2.6. Usuwanie elementów z drzew zbalansowanych	579

14.2.7. Wydajność drzew zbalansowanych	582
14.2.8. Ćwiczenia do podrozdziału 14.2	582
14.3. Tablice z haszowaniem	584
14.3.1. Tablice z haszowaniem w pamięci drugiego stopnia	584
14.3.2. Wstawianie elementów do tablicy z haszowaniem	585
14.3.3. Usuwanie elementów z tablicy z haszowaniem	586
14.3.4. Wydajność indeksów opartych na tablicy z haszowaniem	587
14.3.5. Tablice z haszowaniem rozszerzalnym	587
14.3.6. Wstawianie do tablic z haszowaniem rozszerzalnym	588
14.3.7. Tablice z haszowaniem liniowym	590
14.3.8. Wstawianie do tablic z haszowaniem liniowym	591
14.3.9. Ćwiczenia do podrozdziału 14.3	593
14.4. Indeksy wielowymiarowe	595
14.4.1. Zastosowania indeksów wielowymiarowych	595
14.4.2. Wykonywanie zapytań zakresowych za pomocą tradycyjnych indeksów	596
14.4.3. Wykonywanie zapytań o najbliższego sąsiada z wykorzystaniem tradycyjnych indeksów	597
14.4.4. Przegląd struktur wielowymiarowych indeksów	598
14.5. Struktury haszujące na wielowymiarowe dane	598
14.5.1. Pliki siatki	598
14.5.2. Wyszukiwanie w pliku siatki	599
14.5.3. Wstawianie danych do plików siatki	600
14.5.4. Wydajność plików siatki	602
14.5.5. Podzielone funkcje haszujące	603
14.5.6. Porównanie plików siatki i haszowania podzielonego	604
14.5.7. Ćwiczenia do podrozdziału 14.5	605
14.6. Struktury drzewiaste dla danych wielowymiarowych	606
14.6.1. Indeksy na wielu kluczach	607
14.6.2. Wydajność indeksów na wielu kluczach	607
14.6.3. Drzewa kd	608
14.6.4. Operacje na drzewach kd	610
14.6.5. Przystosowywanie drzew kd do pamięci drugiego stopnia	611
14.6.6. Drzewa czwórkowe	612
14.6.7. R-drzewa	613
14.6.8. Operacje na r-drzewach	614
14.6.9. Ćwiczenia do podrozdziału 14.6	616
14.7. Indeksy bitmapowe	617
14.7.1. Uzasadnienie stosowania indeksów bitmapowych	618
14.7.2. Bitmapy skompresowane	620
14.7.3. Operacje na wektorach bitowych w postaci kodów długości serii	621
14.7.4. Zarządzanie indeksami bitmapowymi	622
14.7.5. Ćwiczenia do podrozdziału 14.7	623
14.8. Podsumowanie rozdziału 14.	623
14.9. Literatura do rozdziału 14.	625
15. Wykonywanie zapytań	627
15.1. Wprowadzenie do operatorów z fizycznego planu zapytania	629
15.1.1. Skanowanie tabel	629
15.1.2. Sortowanie w trakcie skanowania tabel	629
15.1.3. Model obliczeń dla operatorów fizycznych	630
15.1.4. Parametry do pomiaru kosztów	630
15.1.5. Koszty operacji wejścia-wyjścia dla operatorów skanowania	631
15.1.6. Iteratory do implementowania operatorów fizycznych	631

15.2. Algorytmy jednoprzebiegowe	634
15.2.1. Algorytmy jednoprzebiegowe dla operacji krotkowych	635
15.2.2. Algorytmy jednoprzebiegowe dla jednoargumentowych operacji na całych relacjach	636
15.2.3. Algorytmy jednoprzebiegowe dla operacji dwuargumentowych	638
15.2.4. Ćwiczenia do podrozdziału 15.2	640
15.3. Złączenia zagnieżdżone	641
15.3.1. Krotkowe złączenia zagnieżdżone	641
15.3.2. Iterator dla krotkowych złączeń zagnieżdżonych	642
15.3.3. Algorytm złączenia zagnieżdżonego opartego na blokach	643
15.3.4. Analiza złączeń zagnieżdżonych	644
15.3.5. Przegląd opisanych wcześniej algorytmów	644
15.3.6. Ćwiczenia do podrozdziału 15.3	645
15.4. Algorytmy dwuprzebiegowe oparte na sortowaniu	645
15.4.1. Dwuetapowe wielośćściżkowe sortowanie przez scalanie	646
15.4.2. Eliminowanie powtórzeń za pomocą sortowania	647
15.4.3. Grupowanie i agregacja z wykorzystaniem sortowania	647
15.4.4. Algorytm obliczania sumy oparty na sortowaniu	648
15.4.5. Obliczanie za pomocą sortowania części wspólnej i różnicy	649
15.4.6. Prosty algorytm złączenia oparty na sortowaniu	649
15.4.7. Analiza prostego złączenia przez sortowanie	650
15.4.8. Wydajniejsze złączenie przez sortowanie	651
15.4.9. Omówienie algorytmów opartych na sortowaniu	652
15.4.10. Ćwiczenia do podrozdziału 15.4	652
15.5. Dwuprzebiegowe algorytmy oparte na haszowaniu	653
15.5.1. Podział relacji przez haszowanie	653
15.5.2. Algorytm usuwania powtórzeń oparty na haszowaniu	654
15.5.3. Grupowanie i agregacja oparte na haszowaniu	654
15.5.4. Suma, część wspólna i różnica oparte na haszowaniu	655
15.5.5. Algorytm złączenia przez haszowanie	655
15.5.6. Zmniejszanie liczby dyskowych operacji wejścia-wyjścia	656
15.5.7. Przegląd algorytmów opartych na haszowaniu	658
15.5.8. Ćwiczenia do podrozdziału 15.5	659
15.6. Algorytmy oparte na indeksach	659
15.6.1. Indeksy klastrujące i nieklastrujące	659
15.6.2. Selekcja oparta na indeksie	660
15.6.3. Złączenie za pomocą indeksu	662
15.6.4. Złączenia z wykorzystaniem posortowanego indeksu	663
15.6.5. Ćwiczenia do podrozdziału 15.6	664
15.7. Zarządzanie buforem	665
15.7.1. Architektura menedżera buforów	665
15.7.2. Strategie zarządzania buforami	667
15.7.3. Związki między fizycznym operatorem selekcji a zarządzaniem buforami	668
15.7.4. Ćwiczenia do podrozdziału 15.7	669
15.8. Algorytmy o więcej niż dwóch przebiegach	670
15.8.1. Wieloprzebiegowe algorytmy oparte na sortowaniu	670
15.8.2. Wydajność wieloprzebiegowych algorytmów opartych na sortowaniu	671
15.8.3. Wieloprzebiegowe algorytmy oparte na haszowaniu	672
15.8.4. Wydajność wieloprzebiegowych algorytmów opartych na haszowaniu	672
15.8.5. Ćwiczenia do podrozdziału 15.8	673
15.9. Podsumowanie rozdziału 15.	673
15.10. Literatura do rozdziału 15.	675

16. Kompilator zapytań	677
16.1. Parsowanie i przetwarzanie wstępne	678
16.1.1. Analiza składni i drzewa wyprowadzenia	678
16.1.2. Gramatyka prostego podzbioru SQL-a	679
16.1.3. Preprocesor	681
16.1.4. Przetwarzanie wstępne zapytań obejmujących widoki	682
16.1.5. Ćwiczenia do podrozdziału 16.1	685
16.2. Prawa algebraiczne pomocne przy ulepszaniu planów zapytań	685
16.2.1. Prawa przechodniości i łączności	685
16.2.2. Prawa obejmujące selekcję	687
16.2.3. Przenoszenie selekcji	689
16.2.4. Prawa obejmujące projekcję	691
16.2.5. Prawa dotyczące złączeń i iloczynów	692
16.2.6. Prawa obejmujące eliminowanie powtórzeń	693
16.2.7. Prawa dotyczące grupowania i agregacji	693
16.2.8. Ćwiczenia do podrozdziału 16.2	695
16.3. Od drzewa wyprowadzenia do logicznych planów zapytań	697
16.3.1. Przekształcanie na algebrę relacji	697
16.3.2. Eliminowanie podzapytań z warunków	699
16.3.3. Usprawnianie logicznego planu zapytania	703
16.3.4. Grupowanie operatorów łącznych i przechodnich	704
16.3.5. Ćwiczenia do podrozdziału 16.3	705
16.4. Szacowanie kosztów operacji	706
16.4.1. Szacowanie wielkości relacji pośrednich	707
16.4.2. Szacowanie rozmiaru po projekcji	707
16.4.3. Szacowanie rozmiaru relacji po selekcji	708
16.4.4. Szacowanie rozmiaru wyniku złączenia	710
16.4.5. Złączenia naturalne oparte na wielu atrybutach	711
16.4.6. Złączenia wielu relacji	712
16.4.7. Szacowanie rozmiaru wyników innych operacji	713
16.4.8. Ćwiczenia do podrozdziału 16.4	714
16.5. Wprowadzenie do wyboru planu na podstawie kosztów	715
16.5.1. Szacowanie parametrów związanych z rozmiarem	716
16.5.2. Obliczanie statystyk	718
16.5.3. Heurystyki zmniejszania kosztów logicznych planów zapytań	719
16.5.4. Sposoby wyliczania planów fizycznych	721
16.5.5. Ćwiczenia do podrozdziału 16.5	723
16.6. Określanie kolejności złączeń	724
16.6.1. Znaczenie lewego i prawego argumentu złączenia	724
16.6.2. Drzewa złączeń	725
16.6.3. Lewostronnie zagłębione drzewa złączeń	726
16.6.4. Programowanie dynamiczne przy określaniu kolejności złączeń i grupowaniu	728
16.6.5. Programowanie dynamiczne i bardziej szczegółowe funkcje obliczania kosztów	732
16.6.6. Algorytm zachłanny wyboru kolejności złączenia	733
16.6.7. Ćwiczenia do podrozdziału 16.6	734
16.7. Uzupełnianie fizycznego planu zapytania	734
16.7.1. Wybór metody selekcji	735
16.7.2. Wybieranie metody złączenia	737
16.7.3. Przekazywanie potokowe a materializacja	738
16.7.4. Przekazywanie potokowe w operacjach jednoargumentowych	738
16.7.5. Przekazywanie potokowe w operacjach dwuargumentowych	739

16.7.6. Notacja dla fizycznych planów zapytań	741
16.7.7. Porządkowanie operacji fizycznych	744
16.7.8. Ćwiczenia do podrozdziału 16.7	745
16.8. Podsumowanie rozdziału 16.	746
16.9. Literatura do rozdziału 16.	747
17. Radzenie sobie z awariami systemu	749
17.1. Problemy i modele z obszaru odpornych operacji	749
17.1.1. Rodzaje awarii	750
17.1.2. Więcej o transakcjach	751
17.1.3. Prawidłowe wykonywanie transakcji	752
17.1.4. Operacje podstawowe w transakcjach	753
17.1.5. Ćwiczenia do podrozdziału 17.1	755
17.2. Rejestrowanie z możliwością wycofywania	756
17.2.1. Rekordy dziennika	756
17.2.2. Reguły rejestrowania z możliwością wycofywania	757
17.2.3. Przywracanie stanu z wykorzystaniem rejestrowania z możliwością wycofywania	759
17.2.4. Tworzenie punktów kontrolnych	761
17.2.5. Nieblokujące punkty kontrolne	762
17.2.6. Ćwiczenia do podrozdziału 17.2	765
17.3. Rejestrowanie z możliwością powtarzania	766
17.3.1. Reguła rejestrowania z możliwością powtarzania	766
17.3.2. Przywracanie stanu przy stosowaniu rejestrowania z możliwością powtarzania	767
17.3.3. Tworzenie punktów kontrolnych w dzienniku z możliwością powtarzania	768
17.3.4. Przywracanie stanu za pomocą dziennika z możliwością powtarzania z punktami kontrolnymi	769
17.3.5. Ćwiczenia do podrozdziału 17.3	771
17.4. Rejestrowanie z możliwością wycofywania i powtarzania	771
17.4.1. Reguły wycofywania i powtarzania	772
17.4.2. Przywracanie stanu przy rejestrowaniu z możliwością wycofywania i powtarzania	772
17.4.3. Tworzenie punktów kontrolnych w dziennikach z możliwością wycofywania i powtarzania	773
17.4.4. Ćwiczenia do podrozdziału 17.4	775
17.5. Zabezpieczanie się przed uszkodzeniem nośnika	776
17.5.1. Archiwum	776
17.5.2. Archiwizowanie nieblokujące	777
17.5.3. Przywracanie stanu za pomocą archiwum i dziennika	779
17.5.4. Ćwiczenia do podrozdziału 17.5	780
17.6. Podsumowanie rozdziału 17.	780
17.7. Literatura do rozdziału 17.	782
18. Sterowanie współbieżnością	783
18.1. Harmonogramy szeregowo i szeregowalne	784
18.1.1. Harmonogramy	784
18.1.2. Harmonogramy szeregowo	784
18.1.3. Harmonogramy szeregowalne	786
18.1.4. Skutki semantyki działania transakcji	787
18.1.5. Notacja do opisu transakcji i harmonogramów	788
18.1.6. Ćwiczenia do podrozdziału 18.1	789
18.2. Szeregowalność konfliktowa	789
18.2.1. Konflikty	789
18.2.2. Grafy poprzedzania i sprawdzanie szeregowalności konfliktowej	791

18.2.3. Dlaczego test z wykorzystaniem grafu poprzedzania działa?	793
18.2.4. Ćwiczenia do podrozdziału 18.2	794
18.3. Wymuszanie szeregowalności za pomocą blokad	795
18.3.1. Blokad	796
18.3.2. Program szeregujący z funkcją blokowania	798
18.3.3. Blokowanie dwufazowe	799
18.3.4. Dlaczego blokowanie dwufazowe działa?	799
18.3.5. Ćwiczenia do podrozdziału 18.3	800
18.4. Systemy blokowania z kilkoma rodzajami blokad	802
18.4.1. Blokad dzielone i na wyłączność	802
18.4.2. Macierze zgodności	804
18.4.3. „Rozwijanie” blokad	805
18.4.4. Blokad z aktualizacją	806
18.4.5. Blokad inkrementacji	807
18.4.6. Ćwiczenia do podrozdziału 18.4	809
18.5. Architektura programu szeregującego z funkcją blokowania	811
18.5.1. Program szeregujący wstawiający operacje związane z blokadami	811
18.5.2. Tablica blokad	813
18.5.3. Ćwiczenia do podrozdziału 18.5	816
18.6. Hierarchie elementów bazy danych	816
18.6.1. Blokad o różnej szczegółowości	816
18.6.2. Blokad ostrzegawcze	817
18.6.3. Fantomy i poprawna obsługa wstawiania	820
18.6.4. Ćwiczenia do podrozdziału 18.6	820
18.7. Protokół drzewa	821
18.7.1. Uzasadnienie blokowania opartego na drzewach	821
18.7.2. Reguły dostępu do danych w strukturach drzewiastych	822
18.7.3. Dlaczego protokół drzewa działa?	823
18.7.4. Ćwiczenia do podrozdziału 18.7	826
18.8. Sterowanie współbieżnością za pomocą znaczników czasu	826
18.8.1. Znaczniki czasu	827
18.8.2. Fizycznie niewykonalne działania	828
18.8.3. Problemy z „brudnymi” danymi	829
18.8.4. Reguły szeregowania na podstawie znaczników czasu	830
18.8.5. Wielowersyjne znaczniki czasu	831
18.8.6. Znaczniki czasu a blokowanie	833
18.8.7. Ćwiczenia do podrozdziału 18.8	833
18.9. Sterowanie współbieżnością przez sprawdzanie poprawności	834
18.9.1. Architektura programu szeregującego opartego na sprawdzaniu poprawności	834
18.9.2. Reguły sprawdzania poprawności	835
18.9.3. Porównanie trzech mechanizmów sterowania współbieżnością	838
18.9.4. Ćwiczenia do podrozdziału 18.9	839
18.10. Podsumowanie rozdziału 18.	839
18.11. Literatura do rozdziału 18.	841
19. Więcej o zarządzaniu transakcjami	843
19.1. Szeregowalność a możliwość przywracania stanu	843
19.1.1. Problem „brudnych” danych	844
19.1.2. Kaskadowe ponawianie	845
19.1.3. Harmonogramy odtwarzalne	845
19.1.4. Harmonogramy zapobiegające kaskadowemu ponawianiu	846

19.1.5. Zarządzanie ponawianiem przy użyciu blokad	847
19.1.6. Zatwierdzanie grupowe	849
19.1.7. Rejestrowanie logiczne	849
19.1.8. Przywracanie stanu na podstawie dzienników logicznych	852
19.1.9. Ćwiczenia do podrozdziału 19.1	853
19.2. Zakleszczenie	854
19.2.1. Wykrywanie zakleszczenia za pomocą limitów czasu	854
19.2.2. Graf oczekiwania	855
19.2.3. Zapobieganie zakleszczeniu przez porządkowanie elementów	857
19.2.4. Wykrywanie zakleszczeń za pomocą znaczników czasu	858
19.2.5. Porównanie metod zarządzania zakleszczeniem	860
19.2.6. Ćwiczenia do podrozdziału 19.2	861
19.3. Długie transakcje	862
19.3.1. Problemy z długimi transakcjami	862
19.3.2. Sagi	864
19.3.3. Transakcje kompensujące	865
19.3.4. Dlaczego transakcje kompensujące działają?	866
19.3.5. Ćwiczenia do podrozdziału 19.3	867
19.4. Podsumowanie rozdziału 19.	867
19.5. Literatura do rozdziału 19.	868
20. Równoległe i rozproszone bazy danych	869
20.1. Równoległe algorytmy działające na relacjach	869
20.1.1. Modele równoległości	870
20.1.2. Równoległe wykonywanie operacji działających krotka po krotce	872
20.1.3. Algorytmy równoległe dla operacji działających na całych relacjach	873
20.1.4. Wydajność algorytmów równoległych	873
20.1.5. Ćwiczenia do podrozdziału 20.1	875
20.2. System map-reduce do obsługi równoległości	876
20.2.1. Model pamięci	876
20.2.2. Funkcja odwzorowująca	876
20.2.3. Funkcja redukująca	877
20.2.4. Ćwiczenia do podrozdziału 20.2	878
20.3. Rozproszone bazy danych	879
20.3.1. Podział danych	879
20.3.2. Transakcje rozproszone	880
20.3.3. Replikacja danych	880
20.3.4. Ćwiczenia do podrozdziału 20.3	881
20.4. Przetwarzanie zapytań rozproszonych	881
20.4.1. Problem złączenia w środowisku rozproszonym	881
20.4.2. Redukcje za pomocą złączeń częściowych	882
20.4.3. Złączenia wielu relacji	883
20.4.4. Hipergrafy acykliczne	884
20.4.5. Ciągi kompletnej redukcji dla hipergrafów acyklicznych	886
20.4.6. Dlaczego algorytm pełnej redukcji działa?	887
20.4.7. Ćwiczenia do podrozdziału 20.4	887
20.5. Zatwierdzanie w środowisku rozproszonym	888
20.5.1. Zapewnianie atomowości w środowisku rozproszonym	888
20.5.2. Dwufazowe zatwierdzanie	889
20.5.3. Przywracanie stanu przy transakcjach rozproszonych	891
20.5.4. Ćwiczenia do podrozdziału 20.5	893

20.6. Blokowanie w środowisku rozproszonym	893
20.6.1. Scentralizowane systemy blokad	894
20.6.2. Model kosztów dla algorytmów blokowania rozproszonego	894
20.6.3. Blokowanie zreplikowanych elementów	895
20.6.4. Blokowanie kopii podstawowej	896
20.6.5. Blokady globalne oparte na blokadach lokalnych	896
20.6.6. Ćwiczenia do podrozdziału 20.6	898
20.7. Wyszukiwanie rozproszone w systemach P2P	898
20.7.1. Sieci P2P	899
20.7.2. Problem z haszowaniem rozproszonym	899
20.7.3. Scentralizowane rozwiązania z obszaru haszowania rozproszonego	900
20.7.4. Okrąg z cięciwami	900
20.7.5. Odnośniki w okręgach z cięciwami	901
20.7.6. Wyszukiwanie z wykorzystaniem tablicy podręcznej	902
20.7.7. Dodawanie nowych węzłów	904
20.7.8. Opuszczanie sieci przez węzły	907
20.7.9. Awaria węzła	907
20.7.10. Ćwiczenia do podrozdziału 20.7	907
20.8. Podsumowanie rozdziału 20.	908
20.9. Literatura do rozdziału 20.	909

V Inne zagadnienia z obszaru zarządzania dużymi zbiorami danych ... 911

21. Integrowanie informacji 913

21.1. Wprowadzenie do integrowania informacji	913
21.1.1. Po co stosować integrowanie informacji?	914
21.1.2. Problem heterogeniczności	915
21.2. Tryby integrowania informacji	917
21.2.1. Federacyjne systemy baz danych	917
21.2.2. Hurtownie danych	919
21.2.3. Mediator	920
21.2.4. Ćwiczenia do podrozdziału 21.2	922
21.3. Nakładki w systemach opartych na mediatorze	923
21.3.1. Szablony wzorców zapytań	924
21.3.2. Generatory nakładek	925
21.3.3. Filtry	926
21.3.4. Inne operacje nakładki	927
21.3.5. Ćwiczenia do podrozdziału 21.3	928
21.4. Optymalizacja oparta na możliwościach	929
21.4.1. Problem ograniczonych możliwości źródeł	929
21.4.2. Notacja do opisywania możliwości źródeł	930
21.4.3. Wybór planu zapytania na podstawie możliwości	931
21.4.4. Dołączanie optymalizacji na podstawie kosztów	932
21.4.5. Ćwiczenia do podrozdziału 21.4	933
21.5. Optymalizowanie zapytań mediatora	933
21.5.1. Uproszczona notacja dla ozdorników	934
21.5.2. Uzyskiwanie wyników podzadań	934
21.5.3. Algorytm łańcuchowy	935
21.5.4. Sumowanie widoków w mediatorze	938
21.5.5. Ćwiczenia do podrozdziału 21.5	939

21.6. Mediator lokalny w formie widoku	940
21.6.1. Uzasadnienie stosowania mediatorów LAV	940
21.6.2. Terminologia dotycząca mediatorów LAV	941
21.6.3. Rozszerzanie rozwiązań	942
21.6.4. Zawieranie się zapytań koniunkcyjnych	944
21.6.5. Dlaczego test na istnienie odwzorowania zawierającego działa?	945
21.6.6. Wyszukiwanie rozwiązań dla zapytań mediatora	946
21.6.7. Dlaczego twierdzenie LMSS jest prawdziwe?	947
21.6.8. Ćwiczenia do podrozdziału 21.6	948
21.7. Określanie encji	948
21.7.1. Określanie, czy rekordy reprezentują tę samą encję	948
21.7.2. Scalanie podobnych rekordów	950
21.7.3. Przydatne cechy funkcji określania podobieństwa i scalania	951
21.7.4. Algorytm R-Swoosh dla rekordów ICAR	952
21.7.5. Dlaczego algorytm R-Swoosh działa?	954
21.7.6. Inne sposoby określania encji	954
21.7.7. Ćwiczenia do podrozdziału 21.7	955
21.8. Podsumowanie rozdziału 21.	957
21.9. Literatura do rozdziału 21.	958
22. Drażnienie danych	961
22.1. Wyszukiwanie często występujących zbiorów elementów	961
22.1.1. Model koszyka zakupów	962
22.1.2. Podstawowe definicje	963
22.1.3. Reguły asocjacji	964
22.1.4. Model obliczeń dla często występujących zbiorów elementów	965
22.1.5. Ćwiczenia do podrozdziału 22.1	966
22.2. Algorytmy do wyszukiwania często występujących zbiorów elementów	967
22.2.1. Rozkład często występujących zbiorów elementów	967
22.2.2. Naiwny algorytm wyszukiwania częstych zbiorów elementów	968
22.2.3. Algorytm a-priori	969
22.2.4. Implementacja algorytmu a-priori	970
22.2.5. Lepsze wykorzystanie pamięci głównej	971
22.2.6. Kiedy warto stosować algorytm PCY?	972
22.2.7. Algorytm wieloetapowy	973
22.2.8. Ćwiczenia do podrozdziału 22.2	974
22.3. Wyszukiwanie podobnych elementów	975
22.3.1. Miara podobieństwa Jaccarda	975
22.3.2. Zastosowania podobieństwa Jaccarda	976
22.3.3. Minhashing	977
22.3.4. Minhashing i odległość Jaccarda	978
22.3.5. Dlaczego minhashing działa?	978
22.3.6. Implementowanie minhashingu	979
22.3.7. Ćwiczenia do podrozdziału 22.3	980
22.4. LSH	981
22.4.1. Określanie encji i przykład zastosowania LSH	981
22.4.2. Zastosowanie LSH do sygnatur	982
22.4.3. Łączenie minhashingu i LSH	984
22.4.4. Ćwiczenia do podrozdziału 22.4	985

22.5. Grupowanie dużych zbiorów danych	986
22.5.1. Zastosowania grupowania	986
22.5.2. Miary odległości	988
22.5.3. Grupowanie aglomeracyjne	990
22.5.4. Algorytmy oparte na k-średnich	992
22.5.5. Algorytm k-średnich dla dużych zbiorów danych	993
22.5.6. Przetwarzanie porcji punktów z pamięci	995
22.5.7. Ćwiczenia do podrozdziału 22.5	997
22.6. Podsumowanie rozdziału 22.	998
22.7. Literatura do rozdziału 22.	999
23. Systemy baz danych i internet	1001
23.1. Architektura wyszukiwarek	1001
23.1.1. Komponenty wyszukiwarek	1001
23.1.2. Roboty internetowe	1002
23.1.3. Przetwarzanie zapytań w wyszukiwarkach	1005
23.1.4. Tworzenie rankingu stron	1005
23.2. Określanie ważnych stron za pomocą wskaźnika PageRank	1006
23.2.1. Intuicyjne podstawy algorytmu PageRank	1006
23.2.2. Rekurencyjne ujęcie wskaźnika PageRank — pierwsze podejście	1007
23.2.3. Pułapki i ślepe uliczki	1009
23.2.4. Wskaźnik PageRank z uwzględnieniem pułapek i ślepych uliczek	1011
23.2.5. Ćwiczenia do podrozdziału 23.2	1013
23.3. Wskaźnik PageRank specyficzny dla tematu	1014
23.3.1. Zbiór teleportacji	1014
23.3.2. Obliczanie wskaźnika PageRank specyficznego dla tematu	1015
23.3.3. Spam odnośnikami	1016
23.3.4. Wskaźnik PageRank specyficzny dla tematu i spam odnośnikami	1017
23.3.5. Ćwiczenia do podrozdziału 23.3	1018
23.4. Strumienie danych	1019
23.4.1. Systemy zarządzania strumieniami danych	1019
23.4.2. Zastosowania strumieni	1020
23.4.3. Model danych oparty na strumieniach	1021
23.4.4. Przekształcanie strumieni na relacje	1022
23.4.5. Przekształcanie relacji na strumienie	1023
23.4.6. Ćwiczenia do podrozdziału 23.4	1025
23.5. Drażnienie danych w strumieniach	1025
23.5.1. Uzasadnienie	1026
23.5.2. Określanie liczby bitów	1027
23.5.3. Określanie liczby różnych elementów	1030
23.5.4. Ćwiczenia do podrozdziału 23.5	1031
23.6. Podsumowanie rozdziału 23.	1032
23.7. Literatura do rozdziału 23.	1034
Skorowidz	1037

Wykonywanie zapytań

Obszerne zagadnienie przetwarzania zapytań opisano w tym i następnym (16.) rozdziale. *Procesor zapytań* to grupa komponentów systemu DBMS, które przekształcają zapytania użytkownika i polecenia modyfikacji danych na ciąg operacji na bazie danych oraz wykonują je. Ponieważ SQL umożliwia wyrażanie zapytań na bardzo wysokim poziomie, procesor zapytań musi obsługiwać wiele szczegółów dotyczących wykonywania zapytań. Ponadto naiwna strategia wykonywania zapytań może spowodować nieoptymalne wykorzystanie czasu.

Na rysunku 15.1 przedstawiono podział zagadnień z rozdziałów 15. i 16. Ten rozdział dotyczy głównie wykonywania zapytań — algorytmów manipulujących danymi z bazy. Skoncentrowano się na operacjach rozszerzonej algebry relacji opisanych w podrozdziale 5.2. Ponieważ SQL działa zgodnie z modelem opartym na wielozbiorach, założono, że relacje to wielozbiory, dlatego użyto przeznaczonych dla nich operatorów z podrozdziału 5.1.



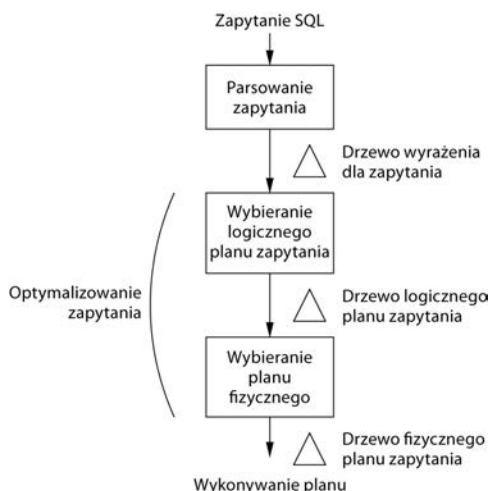
15.1. Główne elementy procesora zapytań

Omówiono podstawowe metody wykonywania operacji algebry relacji. Metody te różnią się podstawową strategią. Główne podejścia to skanowanie, haszowanie, sortowanie i indeksowanie. Techniki różnią się też założeniami dotyczącymi ilości dostępnej pamięci głównej. Niektóre algorytmy wymagają, aby pamięć główna umożliwiała zapisanie przynajmniej jednej z relacji uwzględnianych w operacji. W innych przyjęto, że argumenty operacji są za duże, aby mieściły się w pamięci. Poszczególne algorytmy różnią się kosztem i używanymi strukturami.

Wstęp do kompilowania zapytań

Aby wykonywaniu zapytania nadać kontekst, przedstawiono bardzo krótki zarys następnego rozdziału. Kompilacja zapytania dzieli się na trzy główne kroki pokazane na rysunku 15.2.

- Parsowanie*. Tworzone jest *drzewo wyprowadzenia* dla zapytania.
- Przepisywanie zapytania*. Drzewo wyprowadzenia jest przekształcane na wyjściowy plan zapytania, który zwykle stanowi algebraiczną reprezentację zapytania. Wyjściowy plan jest następnie przekształcany na równoważny, którego wykonanie powinno zająć mniej czasu.
- Generowanie planu fizycznego*. Abstrakcyjny plan zapytania z punktu b), często nazywany *logicznym planem zapytania*, jest przekształcany na *fizyczny plan zapytania* przez wybór algorytmów będących implementacją operatorów z planu logicznego, a także przez wybór kolejności stosowania tych operatorów. Plan fizyczny, podobnie jak efekt parsowania i plan logiczny, jest reprezentowany jako drzewo wyrażenia. Plan fizyczny obejmuje też takie szczegóły jak sposób dostępu do relacji z zapytania, a także informacje o tym kiedy (i czy w ogóle) relację należy sortować.



15.2. Zarys procesu kompilowania zapytań

Kroki b) i c) związane są z *optymalizatorem zapytań* i stanowią trudne aspekty kompilowania zapytania. Aby wybrać najlepszy plan zapytania, trzeba ustalić pewne rzeczy.

- (1) Która z algebraicznie równoważnych postaci zapytania prowadzi do powstania najwydajniejszego algorytmu ustalania odpowiedzi?
- (2) Których algorytmów należy użyć jako implementacji każdej operacji z wybranej postaci zapytania?
- (3) Jak operacje powinny przekazywać dane między sobą — potokowo, przez bufor pamięci głównej czy przez dysk?

Każdy wybór oparty jest na metadanych dotyczących bazy. Typowe metadane dostępne dla optymalizatora zapytań obejmują rozmiar każdej relacji, statystyki w rodzaju przybliżonej liczby i częstotliwości występowania różnych wartości atrybutu, informacje o istnieniu określonych indeksów i informacje o układzie danych na dysku.

15.1. Wprowadzenie do operatorów z fizycznego planu zapytania

Fizyczne plany zapytania bazują na operatorach, z których każdy stanowi implementację jednego kroku planu. Operatory fizyczne zwykle są konkretnymi implementacjami jednej z operacji algebry relacji. Potrzebne są też operatory fizyczne do wykonywania zadań, które nie obejmują operacji algebry relacji. Przykładowo często trzeba „przeskanować” tabelę, czyli przenieść do pamięci głównej każdą krotkę relacji. Relacja jest zwykle operandem pewnej innej operacji.

W tym podrozdziale wprowadzono podstawowe cegiełki fizycznych planów zapytań. W dalszych podrozdziałach omówiono bardziej skomplikowane algorytmy, które są wydajną implementacją operatorów algebry relacji. Algorytmy te stanowią kluczową część fizycznych planów zapytań. Wprowadzono też iteratory — jest to ważne narzędzie, które operatorom z fizycznego planu zapytania umożliwia przekazywanie między sobą zadań krotek i odpowiedzi.

15.1.1. Skanowanie tabel

Prawdopodobnie najbardziej podstawowym zadaniem w fizycznym planie zapytania jest wczytanie całej zawartości relacji R . Odmiana tej operacji obejmuje prosty predykat, co umożliwia wczytanie tylko tych krotek relacji R , które spełniają predykat. Są dwa podstawowe podejścia do znajdowania krotek relacji R .

- (1) Często relacja R jest przechowywana w pamięci drugiego stopnia, a krotki są uporządkowane w blokach. System „wie”, w których blokach znajdują się krotki relacji R , i może pobierać bloki jeden po drugim. Operacja ta to *skanowanie tabeli*.
- (2) Jeśli istnieje indeks na dowolnym atrybucie relacji R , można go użyć, aby uzyskać dostęp do wszystkich krotek z R . Przykładowo indeks rzadki relacji R można, co opisano w punkcie 14.1.3, wykorzystać do przejścia do wszystkich bloków przechowujących R , nawet jeśli nie wiadomo, które to bloki. Ta operacja to *skanowanie indeksu*.

Skanowanie indeksu opisano ponownie w punkcie 15.6.2, w ramach omówienia implementowania selekcji. Na razie warto zaobserwować, że indeks można wykorzystać nie tylko do pobrania *wszystkich* krotek indeksowanej relacji, ale też krotek o określonej wartości (lub przedziale wartości) atrybutu lub atrybutów stanowiących klucz wyszukiwania dla indeksu.

15.1.2. Sortowanie w trakcie skanowania tabel

Jest wiele powodów, dla których przydatne jest sortowanie relacji w czasie wczytywania krotek. Zapytanie może obejmować klauzulę `ORDER BY`, wymagającą posortowania relacji. Ponadto niektóre techniki implementowania operacji algebry relacji wymagają, aby jeden lub oba argumenty były posortowanymi relacjami. Algorytmy te występują w podrozdziale 15.4 i w innych miejscach.

Operator *sortowania w czasie skanowania* w fizycznym planie zapytania przyjmuje relację R oraz wykaz atrybutów, według których należy posortować dane, i zwraca posortowaną relację R . Sortowanie w czasie skanowania można zaimplementować na kilka sposobów. Jeśli relację R trzeba posortować według atrybutu a i istnieje indeks w postaci drzewa zbalansowanego na a , skanowanie indeksu umożliwia uporządkowanie R w pożądanej kolejności. Jeżeli relacja R jest na tyle mała, że mieści się w pamięci głównej, można pobrać krotki przez skanowanie tabeli lub indeksu, a następnie wykorzystać algorytm sortowania w pamięci głównej. Jeśli R jest zbyt duża, aby zapisać ją w pamięci głównej, można wykorzystać wielościeżkowe sortowanie przez scalanie, co omówiono w punkcie 15.4.1.

15.1.3. Model obliczeń dla operatorów fizycznych

Zwykle zapytanie obejmuje kilka operacji algebry relacji, a powiązany fizyczny plan zapytania zawiera kilka operatorów fizycznych. Ponieważ rozsądny wybór operatorów do planu fizycznego jest w procesorze zapytań kluczowy, możliwe musi być oszacowanie kosztu każdego używanego operatora. Jako miary kosztu używamy tu liczby dyskowych operacji wejścia-wyjścia. Miara ta jest spójna z założeniem (punkt 13.3.1), że dłużej trwa pobieranie danych z dysku niż wykonywanie przydatnych operacji na nich po przeniesieniu ich do pamięci głównej.

Przy porównywaniu algorytmów dla tych samych operacji obowiązuje założenie, które początkowo może wydawać się zaskakujące.

- Przyjmujemy, że argumenty operatora znajdują się na dysku, natomiast wynik zostaje zapisany pamięci głównej.

Jeśli operator tworzy ostateczną odpowiedź na zapytanie, a wynik jest zapisywany na dysku, koszt wykonania tej operacji zależy tylko od wielkości odpowiedzi, a nie od sposobu jej obliczenia. Wystarczy dodać koszt ostatecznego zapisu do łącznego kosztu zapytania. Jednak w wielu zastosowaniach odpowiedź w ogóle nie jest zapisywana na dysku, tylko wyświetlana lub przekazywana do programu formatującego. Koszt dyskowych operacji wejścia-wyjścia wynosi wtedy zero lub zależy od tego, co pewna nieznana aplikacja robi z danymi. W obu sytuacjach koszt zapisania odpowiedzi nie wpływa na wybór algorytmu wykonywania operacji.

Podobnie wynik operacji stanowiącej część zapytania (w odróżnieniu od całego zapytania) często nie jest zapisywany na dysku. W punkcie 15.1.6 omówiono iteratory. Działają one tak — wynik uruchomienia jednego operatora O_1 jest tworzony w pamięci głównej, często po małym fragmencie, i przekazywany jako argument do innego operatora O_2 . Wtedy nigdy nie trzeba zapisywać wyniku działania operatora O_1 na dysku. Ponadto można uniknąć ponoszenia kosztu wczytywania z dysku argumentu operatora O_2 .

15.1.4. Parametry do pomiaru kosztów

Wprowadźmy teraz parametry (nazywane czasem statystykami) służące do wyrażania kosztów operatorów. Oszacowanie kosztów jest niezbędne, jeśli optymalizator ma określać, który z wielu planów zapytania prawdopodobnie będzie działał najszybciej. W podrozdziale 16.5 pokazano, jak wykorzystać szacunki kosztów.

Potrzebny jest parametr reprezentujący część pamięci głównej wykorzystywaną przez operator, a także inne parametry określające rozmiar jego argumentów. Załóżmy, że pamięć główna jest podzielona na bufor o rozmiarze równym wielkości bloków dysku. M oznacza liczbę buforów w pamięci głównej dostępnych przy działaniu konkretnego operatora.

Czasem można traktować M jak całą pamięć główną lub większość tej pamięci. Zdarzają się jednak sytuacje, w których kilka operacji współużytkuje pamięć główną, dlatego M może być znacznie mniejsza niż łączna wielkość pamięci głównej. W praktyce, jak opisano to w podrozdziale 15.7, liczba buforów dostępnych operacji może nie być przewidywalną stałą. Nieraz jest ustalana w trakcie wykonywania programu na podstawie tego, jakie inne procesy działają w tym samym czasie. Wtedy M to szacunkowa liczba buforów dostępnych operacji.

Rozważmy teraz parametry do pomiaru kosztu dostępu do argumentów (czyli do relacji). Parametry te, mierzące rozmiar i rozkład danych w relacji, często są obliczane okresowo, aby optymalizatorowi zapytań ułatwić wybór operatorów fizycznych.

Przyjęto tu upraszczające założenie, zgodnie z którym dane są pobierane z dysku po jednym bloku. W praktyce czasem można zastosować jedną z technik omówionych w podrozdziale 13.3

do przyspieszenia pracy algorytmu, jeśli możliwe jest jednoczesne wczytanie wielu bloków z daną relacją, przylegających do siebie na ścieżce. Istnieją trzy rodziny parametrów — B , T i V .

- Przy opisywaniu rozmiaru relacji R najczęściej ważna jest liczba bloków potrzebnych do przechowywania wszystkich krotek relacji. Liczbę bloków oznacza się jako $B(R)$ lub — jeśli wiadomo, że chodzi o relację R — samo B . Zwykle zakłada się, że R jest *sklastrowana* (ang. *clustered*; jest zapisana w B lub około B bloków).
- Czasem trzeba ustalić liczbę krotek w R . Wartość tę oznacza się jako $T(R)$ lub — gdy wiadomo, że chodzi o R — samo T . Jeśli potrzebna jest liczba krotek relacji R mieszczących się w jednym bloku, można wykorzystać stosunek T/B .
- Czasem warto też określić liczbę różnych wartości pojawiających się w kolumnie relacji. Jeśli R to relacja, a jednym z jej atrybutów jest a , wyrażenie $V(R, a)$ oznacza liczbę różnych wartości kolumny a w R . Bardziej ogólnie — jeśli $[a_1, a_2, \dots, a_n]$ to lista atrybutów, $V(R, [a_1, a_2, \dots, a_n])$ to liczba różnych n krotek w kolumnach relacji R odpowiadających atrybutom $a_1, a_2, \dots, a_n$. Formalnie można stwierdzić, że jest to liczba krotek w $\delta(\pi_{a_1, a_2, \dots, a_n}(R))$.

15.1.5. Koszty operacji wejścia-wyjścia dla operatorów skanowania

W ramach prostego zastosowania przedstawionych parametrów można zapisać liczbę dyskowych operacji wejścia-wyjścia potrzebnych dla każdego z omówionych operatorów skanowania tabeli. Jeśli relacja R jest sklastrowana, liczba dyskowych operacji wejścia-wyjścia dla takich operatorów to w przybliżeniu B . Jeżeli R mieści się w pamięci głównej, można zaimplementować sortowanie w czasie skanowania, wczytując R do pamięci i wykonując sortowanie w pamięci, co także wymaga tylko B dyskowych operacji wejścia-wyjścia.

Jeśli jednak R nie jest sklastrowana, liczba potrzebnych dyskowych operacji wejścia-wyjścia jest zwykle dużo większa. Jeżeli R jest rozproszona między krotkami innych relacji, przeglądanie tabeli dla relacji R może wymagać wczytania tylu bloków, ile jest krotek w R (koszt operacji wejścia-wyjścia wynosi T). Podobnie przy sortowaniu relacji R mieszczącej się w pamięci potrzeba T dyskowych operacji wejścia-wyjścia do przeniesienia całej R do pamięci.

Na koniec rozważmy koszt skanowania indeksu. Ogólnie indeks na relacji R zajmuje znacznie mniej niż $B(R)$ bloków. Dlatego skanowanie całej R , na co potrzeba przynajmniej B dyskowych operacji wejścia-wyjścia, wymaga znacznie więcej operacji niż sprawdzenie całego indeksu. Dlatego, choć skanowanie indeksu wymaga sprawdzenia zarówno relacji, jak i indeksu, to:

- nadal można używać B (lub T) do szacowania kosztu dostępu do całych sklastrowanych (lub niesklastrowanych) relacji z wykorzystaniem indeksu.

Jeśli jednak potrzebna jest tylko część R , często można uniknąć sprawdzania całego indeksu i całej R . Analizę takich zastosowań indeksu odkładamy do punktu 15.6.2.

15.1.6. Iteratory do implementowania operatorów fizycznych

Wiele operatorów fizycznych można zaimplementować jako *iterator*. Iterator łączy trzy metody umożliwiające odbiorcy wyniku działania fizycznego operatora pobieranie danych krotka po krotce. Oto te metody.

- (1) `Open()`. Metoda rozpoczyna proces pobierania krotek, jednak ich nie wczytuje. Inicjuje struktury danych potrzebne do wykonania operacji i wywołuje metodę `Open()` dla jej argumentów.
- (2) `GetNext()`. Zwraca następną krotkę z wyniku i — jeśli trzeba — dostosowuje struktury danych, aby umożliwić pobranie kolejnych krotek. Przy pobieraniu następnej krotki z wyniku zwykle wywołuje raz lub kilka razy metodę `GetNext()` dla argumentów. Jeżeli nie można zwrócić dalszych krotek, `GetNext()` zwraca specjalną wartość `NotFound` (zakładamy, że nie można jej pomylić z krotką).
- (3) `Close()`. Ta metoda kończy iterowanie po pobraniu wszystkich krotek (lub krotek żądanych przez odbiorcę). Zwykle wywołuje metodę `Close()` dla argumentów operatora.

W opisie iteratorów i ich metod zakładamy, że dla każdego typu iteratora (na przykład dla każdego typu operatora fizycznego zaimplementowanego jako iterator) istnieje klasa, w której zdefiniowano metody `Open()`, `GetNext()` i `Close()` działające na jej egzemplarzach.

Przykład 15.1. Prawdopodobnie najprostszy jest iterator będący implementacją operatora skanowania tabeli. Iterator jest zaimplementowany jako klasa `TableScan`, a operator skanowania tabeli w planie zapytania to egzemplarz klasy mający parametr w postaci skanowanej relacji R . Założmy, że R to relacja sklastrowana w liście bloków, do której można w wygodny sposób uzyskać dostęp (przyjmujemy, iż operacja „pobierz następny blok R ” jest zaimplementowana w systemie pamięci i nie trzeba jej szczegółowo opisywać). Ponadto zakładamy, że w bloku znajduje się katalog rekordów (krotek), dlatego można łatwo pobrać następną krotkę z bloku lub stwierdzić, iż osiągnięto ostatnią krotkę.

Na listingu 15.3 znajduje się zarys trzech metod iteratora. Zakładamy, że istnieje wskaźnik b do bloku i wskaźnik t do krotki w bloku wskazywanym przez b . Oba wskaźniki mogą prowadzić „poza” ostatni blok lub ostatnią krotkę w bloku. Można wykryć wystąpienie tych warunków. Zauważmy, że metoda `Close()` w przykładzie nie wykonuje żadnych działań. W praktyce może ona na różne sposoby porządkować wewnętrzną strukturę systemu DBMS. Może informować menedżer bufora, że niektóre bufora nie są już potrzebne, lub menedżer współbieżności o zakończeniu wczytywania relacji. ■

```
Open() {
    b := pierwszy blok R;
    t := pierwsza krotka bloku b;
}

GetNext() {
    IF (t prowadzi poza ostatnią krotkę bloku b) {
        zwiększ b, aby prowadził do następnego bloku;
        IF (nie ma następnego bloku)
            RETURN NotFound;
        ELSE /* b to nowy blok. */
            t := pierwsza krotka bloku b;
    } /* Teraz można zwrócić i zwiększyć t. */
    oldt := t;
    zwiększanie t, aby prowadził do następnej krotki b;
    RETURN oldt;
}

Close() {
}
```

15.3. Metody iteratora dla operatora skanowania tabeli dla relacji R

Przykład 15.2. Teraz rozważmy przykład, w którym iterator wykonuje większość zadań w metodzie `Open()`. Używamy operatora sortowania przy skanowaniu. Wczytuje on krotki relacji R i zwraca je w posortowanej kolejności. Do czasu sprawdzenia wszystkich krotek w R nie można zwrócić nawet pierwszej krotki. Dla uproszczenia założmy, iż relacja R jest na tyle mała, że mieści się w pamięci głównej.

Metoda `Open()` musi wczytywać całą relację R do pamięci głównej. Może też sortować krotki z R . Wtedy `GetNext()` musi tylko zwrócić po kolei każdą krotkę w posortowanej kolejności. Metoda `Open()` może też pozostawić R nieposortowaną. Wtedy w `GetNext()` należy pobierać pierwszą z pozostałych krotek, co w efekcie oznacza wykonanie jednego przebiegu sortowania przez wybieranie. ■

Przykład 15.3. Na koniec rozważmy prosty przykład łączenia iteratorów przez wywoływanie innych iteratorów. Operacja to suma wielozbiorów $R \cup S$, w której najpierw należy zwrócić wszystkie krotki R , a następnie wszystkie krotki S bez eliminowania powtórzeń. Niech R i S oznaczają iteratory zwracające relacje R i S (tym samym są dziećmi operatora sumy w planie zapytania dla operacji $R \cup S$). Iteratory R i S mogą odpowiadać skanowaniu tabeli relacji składowanych R i S . Ponadto mogą być iteratorami, które wywołują zestaw innych iteratorów w celu obliczenia R i S . Niezależnie od tego ważne jest, aby dostępne były metody `R.Open()`, `R.GetNext()` i `R.Close()` oraz odpowiadające im metody iteratora S .

Zarys metod iteratora dla sumy przedstawiono na listingu 15.4. Ciekawe jest to, że metody korzystają ze zmiennej współużytkowanej `ObecRel` przechowującej R lub S (w zależności od tego, która relacja jest obecnie wczytywana). ■

```
Open() {
    R.Open();
    ObecRel := R;
}

GetNext() {
    IF (ObecRel = R) {
        t := R.GetNext();
        IF (t <> NotFound) /* R nie została pobrana w całości. */
            RETURN t;
        ELSE /* R została pobrana w całości. */ {
            S.Open();
            ObecRel := S;
        }
    }
    /* Tu trzeba wczytać dane z S. */
    RETURN S.GetNext();
    /* Zauważmy, że jeśli pobrano całą S, S.GetNext()
       zwróci NotFound, co jest prawidłowym działaniem
       metody GetNext. */
}

Close() {
    R.Close();
    S.Close();
}
```

15.4. Tworzenie iteratorów dla sumy na podstawie iteratorów R i S

Dlaczego iteratory?

W podrozdziale 16.7 pokazano, że połączenie iteratorów w planie zapytania pozwala uzyskać wysoką wydajność. Inaczej działa *materializacja*, kiedy to wynik działania każdego operatora jest zwracany w całości — zostaje zapisany albo na dysku, albo w pamięci głównej. Przy stosowaniu iteratorów jednocześnie aktywnych jest wiele operacji. Krotki są przekazywane między operatorami na żądanie, co zmniejsza zapotrzebowanie na pamięć. Oczywiście, co opisano dalej, nie wszystkie operatory fizyczne umożliwiają sensowną obsługę iteracji (potokowania). Czasem prawie wszystkie zadania trzeba wykonać w metodzie `Open()`, co przypomina materializację.

15.2. Algorytmy jednoprzebiegowe

W tym miejscu rozpoczyna się omówienie bardzo ważnego zagadnienia w obszarze optymalizowania zapytań: „Jak należy wykonywać każdy z poszczególnych kroków — na przykład złączenie lub selekcję — logicznego planu zapytania?”. Wybór algorytmu dla każdego operatora to kluczowa część procesu przekształcania logicznego planu zapytania na plan fizyczny. Choć zaproponowano wiele algorytmów dla operatorów, należą one głównie do trzech klas.

- (1) Metody oparte na sortowaniu (podrozdział 15.4).
- (2) Metody oparte na haszowaniu (podrozdziały 15.5 i 20.1).
- (3) Metody oparte na indeksach (podrozdział 15.6).

Ponadto algorytmy działania operatorów można podzielić na trzy „stopnie” na podstawie trudności i kosztów.

- a) Niektóre metody wymagają tylko jednokrotnego wczytania danych z dysku. Są to algorytmy *jednoprzebiegowe*. Omówiono je w tym podrozdziale. Zwykle przynajmniej jeden z argumentów musi mieścić się w pamięci głównej, choć zdarzają się wyjątki, zwłaszcza w zakresie selekcji i projekcji, co opisano w podrozdziale 15.2.1.
- b) Niektóre metody działają dla danych, które nie mieszczą się w dostępnej pamięci głównej, ale nie działają dla największych wyobrażalnych zbiorów danych. Są to algorytmy *dwuprzebiegowe*, cechujące się początkowym odczytem danych z dysku, przetwarzaniem ich w pewien sposób, zapisem wszystkich (lub prawie wszystkich) na dysku i późniejszym wczytywaniem ich po raz wtóry w celu dalszego przetwarzania w ramach drugiego przebiegu. Algorytmy tego typu pojawiają się w podrozdziałach 15.4 i 15.5.
- c) Niektóre metody działają dla dowolnie dużych danych. Metody te wymagają trzech lub więcej przebiegów i są naturalnym, rekurencyjnym uogólnieniem algorytmów dwuprzebiegowych. Metody wieloprzebiegowe omówiono w podrozdziale 15.8.

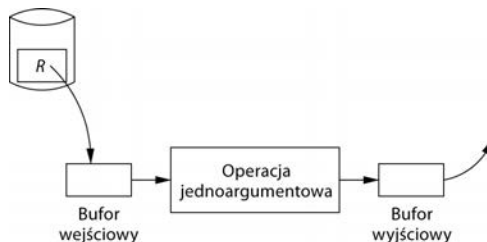
W tym podrozdziale skoncentrowano się na metodach jednoprzebiegowych. Tu i w dalszych punktach operatory dzielone są na trzy ogólne grupy.

- (1) *Krotkowe operacje jednoargumentowe (działające na pojedynczych krotkach)*. Te operacje — selekcja i projekcja — nie wymagają jednoczesnego zapisania całej relacji (a nawet dużej jej części) w pamięci. Dlatego można wczytywać dane blok po bloku, używać jednego bufora w pamięci głównej i generować dane wyjściowe.

- (2) *Operacje jednoargumentowe na całych relacjach.* Jednoargumentowe operacje tego typu wymagają jednoczesnego umieszczenia w pamięci wszystkich lub większości krotek, dlatego algorytmy jednoprzebiegowe działają tylko dla relacji o rozmiarze M (liczba dostępnych buforów pamięci głównej) lub mniejszym. Operacje tej klasy to γ (grupowanie) i δ (usuwanie powtórzeń).
- (3) *Operacje dwuargumentowe na całych relacjach.* Do tej klasy należą wszystkie pozostałe operacje — suma, część wspólna, różnica, złączenia i iloczyny w wersjach dla zbiorów oraz wielozbiorów. Aby zastosować algorytm jednoprzebiegowy, w każdej operacji, z wyjątkiem sumy wielozbiorów, przynajmniej jeden argument musi mieć rozmiar nie większy niż M .

15.2.1. Algorytmy jednoprzebiegowe dla operacji krotkowych

Operacjom krotkowym, $\sigma(R)$ i $\pi(R)$, odpowiadają oczywiste algorytmy niezależnie od tego, czy relacja mieści się w pamięci głównej. Należy czytywać bloki relacji R jeden po drugim do bufora wejściowego, wykonywać operacje na każdej krotce i przenosić pobrane lub zrzucone krotki do bufora wyjściowego, co pokazano na rysunku 15.5. Ponieważ bufor wyjściowy może być buforem wejściowym innego operatora lub udostępniać dane użytkownikowi albo aplikacji, nie uwzględniono bufora wyjściowego przy obliczaniu potrzebnej pamięci. Dlatego konieczne jest tylko, aby — niezależnie od B — $M \geq 1$ dla bufora wejściowego.



15.5. Przeprowadzanie selekcji lub projekcji na relacji R

Liczba dyskowych operacji wejścia-wyjścia zależy tylko od sposobu udostępniania relacji R będącej argumentem. Jeśli R początkowo znajduje się na dysku, koszt odpowiada skanowaniu tabeli lub indeksu dla R . Koszt tego typu omówiono w punkcie 15.1.5 — zwykle wynosi B dla sklastrowanej R i T dla niesklastrowanej R . Warto jednak pamiętać o ważnym wyjątku, kiedy w warunku selekcji stała porównywana jest z atrybutem, na którym oparty jest indeks. Wtedy można użyć indeksu do pobrania podzbioru bloków przechowujących R i poprawienia w ten sposób wydajności — często w znacznym stopniu.

Dodatkowe bufony pozwalają przyspieszyć operacje

Choć operacje działające krotka po krotce można wykonać z jednym buforem wejściowym i jednym buforem wyjściowym, co pokazano na rysunku 15.5, często można przyspieszyć przetwarzanie przez przydział dodatkowych buforów wejściowych. Pomysł ten po raz pierwszy pojawił się w punkcie 13.3.2. Jeśli R jest zapisana w cylindrach w przyległych blokach, można do buforów wczytać cały cylinder. Koszt wyszukiwania i opóźnienia obrotowego ponoszony jest tylko dla jednego bloku na cylinder. Podobnie jeśli dane wyjściowe operacji można zapisać w pełnych cylindrach, zapis nie wiąże się z prawie żadną stratą czasu.

15.2.2. Algorytmy jednoprzebiegowe dla jednoargumentowych operacji na całych relacjach

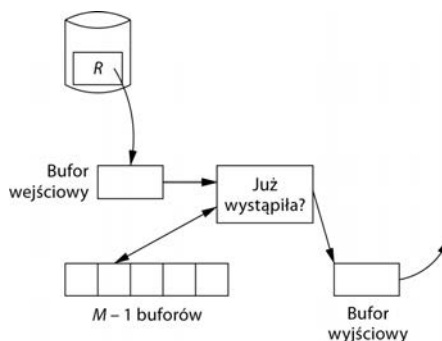
Rozważmy teraz operacje jednoargumentowe stosowane do całych relacji, a nie do pojedynczych krotek, czyli eliminowanie powtórzeń (δ) i grupowanie (γ).

Eliminowanie powtórzeń

Aby wyeliminować powtórzenia, można wczytać jeden po drugim każdy blok z R , przy czym dla każdej krotki trzeba ustalić, czy:

- (1) jest to pierwsze wystąpienie danej krotki (wtedy należy skopiować ją do danych wyjściowych);
- (2) krotka pojawiła się już wcześniej (wtedy nie należy jej zwracać).

Aby to ustalić, trzeba przechowywać w pamięci jedną kopię każdej napotkanej krotki, co pokazano na rysunku 15.6. Jeden bufor przechowuje w pamięci jeden blok krotek z R , a pozostałe $M - 1$ buforów można wykorzystać do przechowywania pojedynczych egzemplarzy każdej pobranej krotki.



15.6. Zarządzanie pamięcią przy jednoprzebiegowym eliminowaniu powtórzeń

Do zapisywania napotkanych krotek trzeba odpowiednio dobrać strukturę danych w pamięci głównej. Naiwne podejście to użycie listy krotek. Przy sprawdzaniu nowej krotki z R należy wtedy porównać ją ze wszystkimi napotkanymi krotkami. Jeśli nie jest równa żadnej z nich, trzeba skopiować ją do danych wyjściowych i dodać do przechowywanej w pamięci listy pobranych krotek.

Jeśli jednak w pamięci głównej znajduje się n krotek, każda nowa krotka zajmuje czas procesora proporcjonalnie do n , dlatego cała operacja potrzebuje czasu proporcjonalnego do n^2 . Ponieważ n może być bardzo duże, ilość potrzebnego czasu podaje w wątpliwość założenie, że ważny jest tylko czas wykonywania dyskowych operacji wejścia-wyjścia. Dlatego w pamięci głównej potrzebna jest struktura umożliwiająca dodanie nowej krotki i określenie w czasie rozsądnym powoli względem n , czy dana krotka już się pojawiła.

Można na przykład użyć tablicy z haszowaniem z dużą liczbą kubeków lub jednej z wersji zbalansowanego binarnego drzewa wyszukiwań¹. Każda z tych struktur wymaga dodatkowego

¹ Omówienie odpowiednich struktur pamięci głównej można znaleźć w: A. V. Aho, J. E. Hopcroft i J. D. Ullman, *Data Structures and Algorithms*, 1983 (wydanie polskie: *Algorytmy i struktury danych*, Helion, 2003). Przetwarzanie n elementów za pomocą haszowania zajmuje średnio $O(n)$, a z wykorzystaniem drzew zbalansowanych — $O(n \log n)$. W obu przypadkach wzrost długości czasu przetwarzania jest wystarczająco bliski liniowemu, jak na nasze potrzeby.

miejsca obok pamięci potrzebnej na przechowywanie krotek. Przykładowo tablica z haszowaniem w pamięci głównej wymaga tablicy kubelków i miejsca na wskaźniki prowadzące do krotek w kubelku. Jednak dodatkowe koszty są niskie w porównaniu z pamięcią potrzebną na zapisanie krotek, dlatego w tym rozdziale je pominięto.

Zgodnie z tym założeniem w $M - 1$ dostępnych buforach pamięci głównej można zapisać tyle krotek, ile mieści się w $M - 1$ blokach relacji R . Jeśli w pamięci głównej ma zmieścić się jedna kopia każdej niepowtarzalnej krotki z R , $B(\delta(R))$ musi być nie większe niż $M - 1$. Ponieważ zakładamy, że M jest znacznie większe niż 1, zwykle stosowane jest uproszczone przybliżenie tej reguły:

- $B(\delta(R)) \leq M$

Zauważmy, że zwykle nie można określić rozmiaru $\delta(R)$, nie obliczając samego $\delta(R)$. Jeśli rzeczywisty rozmiar będzie większy od oszacowanego ($B(\delta(R))$ jest większe od M), trzeba ponieść istotne koszty związane z przeładowywaniem, ponieważ bloki przechowujące różne krotki z R trzeba często przenosić do pamięci i z niej usuwać.

Grupowanie

Operacja grupowania, γ_L , zwraca zero lub więcej atrybutów grupujących i zwykle jeden lub więcej atrybutów zagregowanych. Po utworzeniu w pamięci głównej jednego wpisu dla każdej grupy — dla każdej wartości atrybutów grupujących — można przejrzeć krotki z R po jednym bloku na raz. *Wpis* dla grupy składa się z wartości atrybutów grupujących i zakumulowane wartości każdej agregacji.

- Dla agregacji $\text{MIN}(a)$ lub $\text{MAX}(a)$ należy zapisać minimalną lub maksymalną wartość atrybutu a dla wszystkich sprawdzonych już krotek z grupy. Jeśli to konieczne, należy zmienić minimum lub maksimum przy pobraniu każdej krotki z grupy.
- Dla agregacji COUNT należy dodać 1 dla każdej pobranej krotki z danej grupy.
- Dla agregacji $\text{SUM}(a)$ należy dodać wartość atrybutu a do zakumulowanej sumy dla danej grupy (pod warunkiem, że a jest różne od NULL).
- $\text{AVG}(a)$ to trudny przypadek. Trzeba przechowywać dwie zakumulowane wartości — liczbę krotek w grupie i sumę wartości atrybutu a w tych krotkach. Obie wartości należy obliczać tak, jak dla agregacji COUNT i SUM . Po sprawdzeniu wszystkich krotek z R trzeba podzielić sumę przez liczbę krotek, aby uzyskać średnią.

Po wyczytaniu wszystkich krotek z R do bufora wejściowego i użyciu ich do obliczenia agregacji dla odpowiednich grup można utworzyć dane wyjściowe, zapisując krotkę dla każdej grupy. Warto zauważyć, że do czasu sprawdzenia ostatniej krotki nie można rozpocząć tworzenia danych wyjściowych dla operacji γ . Dlatego algorytm ten nie pasuje do modelu działania iteratorów. W metodzie *Open* trzeba zakończyć grupowanie, zanim za pomocą metody *GetNext* będzie można pobrać pierwszą krotkę.

Aby przetwarzanie każdej krotki w pamięci było wydajne, trzeba w pamięci głównej użyć struktury danych umożliwiającej znalezienie wpisu dla każdej grupy na podstawie wartości atrybutów grupujących. Jak opisano to w kontekście operacji δ , dobrze nadają się do tego typowe struktury danych stosowane w pamięci głównej, takie jak tablice z haszowaniem i drzewa zbalansowane. Należy jednak pamiętać, że kluczem wyszukiwania dla tych struktur są tylko atrybuty grupujące.

Operacje na niesklastrowanych danych

Wszystkie obliczenia dotyczące liczby dyskowych operacji wejścia-wyjścia wymaganych dla omawianych operacji są oparte na założeniu, że relacje podane jako operandy są sklastrowane. W — zwykle rzadkich — sytuacjach, kiedy operand R nie jest sklastrowany, wczytanie wszystkich krotek z R może wymagać $T(R)$, a nie $B(R)$ dyskowych operacji wejścia-wyjścia. Zauważmy jednak, że można przyjąć, iż każda relacja zwracana przez operator jest sklastrowana — nie ma powodu, aby zapisywać tymczasową relację w niesklastrowany sposób.

Liczba dyskowych operacji wejścia-wyjścia potrzebnych dla tego jednoprzebiegowego algorytmu wynosi B (jest tak w każdym jednoprzebiegowym algorytmie dla operatora jednoargumentowego). Liczba niezbędnych buforów pamięci M nie jest powiązana z B w żaden prosty sposób, choć zwykle M ma wartość mniejszą niż B . Problem polega na tym, że wpisy dla grup mogą być dłuższe lub krótsze niż krotki z R , a liczba grup może być równa liczbie krotek z R lub mniejsza. Jednak zwykle wpisy dla grup nie są dłuższe od krotek z R , a grup jest dużo mniej niż krotek.

15.2.3. Algorytmy jednoprzebiegowe dla operacji dwuargumentowych

Dalej opisano operacje dwuargumentowe — sumę, część wspólną, różnicę, iloczyn i złączenie. Ponieważ w niektórych przypadkach trzeba odróżnić wersje operatorów oparte na zbiorach i wielozbiorach, występują przy nich indeksy dolne Z (dla zbiorów) i W (dla wielozbiorów). Przykładowo $\cup_W$ oznacza sumę wielozbiorów, a $-_Z$ — różnicę zbiorów. Aby uprościć omawianie złączeń, uwzględniono tylko złączenia naturalne. Złączenie równościowe można zaimplementować w ten sam sposób (po odpowiednim przemianowaniu atrybutów), a złączenia warunkowe można potraktować jak iloczyn lub złączenie równościowe, po którym następuje selekcja dla warunków niewyrażalnych w złączeniu równościowym.

Sumę wielozbiorów można obliczyć za pomocą bardzo prostego algorytmu jednoprzebiegowego. Aby obliczyć $R \cup_W S$, należy skopiować do danych wyjściowych każdą krotkę z R , a następnie z S , tak jak w przykładzie 15.3. Liczba dyskowych operacji wejścia-wyjścia to $B(R) + B(S)$, jak zawsze w jednoprzebiegowych algorytmach na operandach R i S , natomiast $M = 1$ jest wystarczającą wartością niezależnie od wielkości R i S .

Inne operacje dwuargumentowe wymagają wczytania mniejszego z operandów R i S do pamięci głównej oraz utworzenia odpowiedniej struktury danych, tak aby krotki można było szybko wstawiać i znajdować, co opisano w punkcie 15.2.2. Tak jak wcześniej, wystarczająca jest tablica z haszowaniem lub drzewo zbalansowane. Dlatego w przybliżeniu można określić wymóg dla operacji dwuargumentowych na relacjach R i S , jeśli mają być wykonywane w jednym przebiegu:

- $\min(B(R), B(S)) \leq M$

Ujmijmy to dokładniej — jeden bufor służy do wczytywania bloków większej relacji, natomiast około M buforów trzeba na całą mniejszą relację i jej strukturę danych przechowywaną w pamięci głównej.

Dalej przedstawiono szczegóły różnych operacji. W każdym przypadku założono, że R jest większą z relacji, dlatego S jest przechowywana w pamięci głównej.

Suma zbiorów

Należy wczytać S do $M - 1$ buforów w pamięci głównej i zbudować ułatwiającą wyszukiwanie strukturę, której kluczem wyszukiwania jest cała krotka. Wszystkie krotki są też kopiowane do danych wyjściowych. Następnie trzeba wczytać jeden po drugim każdy blok z R do M -tego bufora. Dla każdej krotki t z R trzeba sprawdzić, czy t znajduje się w S . Jeśli nie, należy skopiować t do danych wyjściowych. Jeżeli t występuje w S , t można pominąć.

Część wspólna zbiorów

Należy wczytać S do $M - 1$ buforów i zbudować ułatwiającą wyszukiwanie strukturę, której kluczem wyszukiwania są całe krotki. Trzeba wczytać każdy blok R i dla każdej krotki z R sprawdzić, czy t znajduje się także w S . Jeśli tak, należy skopiować t do danych wyjściowych; w przeciwnym razie t można pominąć.

Różnica zbiorów

Ponieważ różnica nie jest przemienne, trzeba rozróżnić wyrażenie $R - S$ od $S - R$ (nadal obowiązuje założenie, że R to większa relacja). W obu przypadkach należy wczytać S do $M - 1$ buforów i zbudować ułatwiającą wyszukiwanie strukturę, której kluczem wyszukiwania są całe krotki.

Aby obliczyć $R - S$, trzeba wczytać każdy blok z R i sprawdzić każdą krotkę t z tego bloku. Jeśli t znajduje się w S , należy pominąć t ; w przeciwnym razie trzeba skopiować t do danych wyjściowych.

Aby obliczyć $S - R$, także trzeba wczytać wszystkie bloki R i po kolei sprawdzić każdą krotkę t . Jeśli t znajduje się w S , należy usunąć t z przechowywanej w pamięci głównej kopii S . Jeżeli t nie występuje w S , nie trzeba nic robić. Po sprawdzeniu wszystkich krotek z R można skopiować do danych wyjściowych wszystkie pozostałe krotki z S .

Część wspólna wielozbiorów

Należy wczytać S do $M - 1$ buforów i powiązać każdą wartość krotki z *licznikiem*, który początkowo mierzy liczbę wystąpień krotki w S . Poszczególne kopie krotki t nie są przechowywane osobno. Zamiast tego należy zapisać jedną kopię t i powiązać ją z licznikiem określającym liczbę wystąpień t .

Jeśli powtórzeń jest niewiele, struktura ta może zająć nieco więcej niż $B(S)$ bloków, choć często pozwala zmniejszyć S . Dlatego nadal obowiązuje założenie, że $B(S) \leq M$ wystarczy do działania algorytmu jednoprzebiegowego, choć ten warunek to tylko przybliżenie.

Następnie trzeba wczytać każdy blok R i dla każdej krotki t z R sprawdzić, czy t występuje w S . Jeśli nie, można pominąć t — krotka ta nie należy do części wspólnej. Jeżeli jednak t pojawia się w S , a licznik powiązany z t ma wartość dodatnią, trzeba dodać t do danych wyjściowych i zmniejszyć licznik o 1. Jeśli t występuje w S , jednak licznik doszedł do 0, nie należy zwracać t — w danych wyjściowych znajduje się już tyle kopii t , ile istniało ich w S .

Różnica wielozbiorów

Aby obliczyć $S -_W R$, należy wczytać krotki z S do pamięci głównej i ustalić liczbę wystąpień krotek o różnych wartościach (tak jak dla części wspólnej wielozbiorów). Przy wczytywaniu każdej krotki t z R należy sprawdzić, czy występuje ona w S . Jeśli tak, trzeba zmniejszyć wartość powiązanego licznika. Na koniec trzeba skopiować z pamięci głównej do danych wyjściowych każdą krotkę o pozytywnej wartości licznika (wartość ta wyznacza liczbę kopii).

Aby obliczyć $R -_W S$, także trzeba wczytać krotki z S do pamięci głównej i ustalić liczbę wystąpień różnych krotek. Krotkę t z licznikiem o wartości c można traktować jak c powodów,

aby nie kopiować t do danych wyjściowych przy wczytywaniu krotek z R . Przy wczytywaniu krotki t z R należy sprawdzić, czy t występuje w S . Jeśli nie, t kopiowana jest do danych wyjściowych. Jeżeli t pojawia się w S , trzeba sprawdzić wartość c powiązaną z t . Jeśli $c = 0$, należy skopiować t do danych wyjściowych. Przy $c > 0$ t nie jest kopiowana, natomiast trzeba zmniejszyć wartość c o 1.

Iloczyn

Należy wczytać S do $M - 1$ buforów pamięci głównej. Nie jest potrzebna żadna specjalna struktura danych. Następnie wczytywany jest każdy blok z R . Każdą krotkę t z R należy złączyć t z każdą krotką S z pamięci głównej. Złączone krotki trzeba dodać do danych wyjściowych.

Algorytm ten wymaga dużej ilości czasu procesora na krotkę z R , ponieważ każdą z nich trzeba dopasować do $M - 1$ bloków pełnych krotek. Jednak dane wyjściowe także są duże, a czas w przeliczeniu na krotkę z tych danych jest krótki.

Złączenie naturalne

W tym i innych algorytmach złączenia przyjęto, że relacja $R(X, Y)$ jest złączana z $S(Y, Z)$, gdzie Y reprezentuje wszystkie atrybuty wspólne dla R i S , X to wszystkie atrybuty z R , które nie występują w S , a Z to atrybuty z S nieobecne w R . Nadal zakładamy, że S to mniejsza relacja. Aby obliczyć złączenie naturalne, należy wykonać następujące operacje.

- (1) Wczytać wszystkie krotki z S i umieścić je w pamięci głównej we wspomagającej wyszukiwanie strukturze (z atrybutami Y jako kluczem wyszukiwania). Można wykorzystać na to $M - 1$ bloków pamięci.
- (2) Wczytać każdy blok z R do jednego pozostałego bufora pamięci głównej. Używając struktury ułatwiającej wyszukiwanie, należy dla każdej krotki t z R znaleźć krotki z S o zgodnych atrybutach Y . Dla każdej pasującej krotki z S trzeba utworzyć krotkę przez złączenie jej z t . Wynikowa krotka jest umieszczana w danych wyjściowych.

Podobnie jak wszystkie jednoprzebiegowe algorytmy dwuargumentowe, ten wymaga $B(R) + B(S)$ dyskowych operacji wejścia-wyjścia do wczytania operandów. Działa dla $B(S) \leq M - 1$ (w przybliżeniu dla $B(S) \leq M$).

Nie omawiamy tu złączeń innych niż naturalne. Warto pamiętać, że złączenie równościowe jest obliczane w zasadzie tak samo, jednak trzeba uwzględnić fakt, iż równe atrybuty z obu relacji mogą mieć różne nazwy. Złączenie warunkowe, które nie jest złączeniem równościowym, można zastąpić złączeniem równościowym lub iloczynem, po którym następuje selekcja.

15.2.4. Ćwiczenia do podrozdziału 15.2

Ćwiczenie 15.2.1. Dla każdej z podanych operacji napisz iterator z wykorzystaniem algorytmu opisanego w tej sekcji: a) projekcja, b) usuwanie powtórzeń (δ), c) grupowanie (γ_L), d) suma zbiorów, e) część wspólna zbiorów, f) różnica zbiorów, g) część wspólna wielozbiorów, h) różnica wielozbiorów, i) iloczyn, j) złączenie naturalne.

Ćwiczenie 15.2.2. Dla każdego z operatorów z ćwiczenia 15.2.1 określ, czy operator jest *blokujący* (czyli czy pierwsze dane wyjściowe można utworzyć dopiero po wczytaniu wszystkich danych wejściowych). Ujmijmy to inaczej — operator blokujący to taki, dla którego jedyne możliwe iteratory wykonują wszystkie ważne zadania w funkcji Open.

Ćwiczenie 15.2.3. W tabeli 15.9 pokazano wymogi dotyczące pamięci i dyskowych operacji wejścia-wyjścia dla algorytmów opisanych w tym i następnym podrozdziale. Założono jednak, że wszystkie argumenty są składowane. Jak zmieniają się wartości, jeśli jeden lub oba argumenty nie są składowane?

! Ćwiczenie 15.2.4. Przedstaw algorytmy jednoprzebiegowe dla każdego z poniższych operatorów z rodziny złączeń:

- $R \bowtie S$ przy założeniu, że R mieści się w pamięci (definicję złączenia częściowego przedstawiono w ćwiczeniu 2.4.8).
- $R \bowtie S$ przy założeniu, że S mieści się w pamięci.
- $R \bowtie S$ przy założeniu, że R mieści się w pamięci (definicję nierównozłączenia częściowego przedstawiono w ćwiczeniu 2.4.9).
- $R \bowtie S$ przy założeniu, że S mieści się w pamięci.
- $R \bowtie_L S$ przy założeniu, że R mieści się w pamięci (definicję złączeń zewnętrznych zawiera punkt 5.2.7).
- $R \bowtie_L S$ przy założeniu, że S mieści się w pamięci.
- $R \bowtie_R S$ przy założeniu, że R mieści się w pamięci.
- $R \bowtie_R S$ przy założeniu, że S mieści się w pamięci.
- $R \bowtie S$ przy założeniu, że R mieści się w pamięci.

15.3. Złączenia zagnieżdżone

Przed przejściem do bardziej złożonych algorytmów przedstawionych w następnych podrozdziałach warto przyrzeć się rodzinie algorytmów dla operatora złączenia zagnieżdżonego (z pętłą). Algorytmy te wykonują w pewnym sensie półtora przebiegu, ponieważ w każdej wersji dla jednego z dwóch argumentów krotki wczytywane są tylko raz, natomiast dla drugiego — wielokrotnie. Złączeń zagnieżdżonych można używać dla relacji dowolnej wielkości. Nie jest konieczne, aby jedna z relacji mieściła się w pamięci głównej.

15.3.1. Krotkowe złączenia zagnieżdżone

Najprostsza odmiana złączeń zagnieżdżonych obejmuje pętle przechodzące po poszczególnych krotkach używanych relacji. W tym algorytmie, *krotkowym złączeniu zagnieżdżonym*, złączenie $R(X, Y) \bowtie S(Y, Z)$ obliczane jest tak:

```

FOR każda krotka s z S DO
  FOR każda krotka r z R DO
    IF r i s można złączyć w krotkę t THEN
      zwróć t;
```

Przy nieostrożnym buforowaniu bloków relacji R i S algorytm może wymagać aż $T(R)T(S)$ dyskowych operacji wejścia-wyjścia. Jednak w wielu sytuacjach algorytm można zmodyfikować, tak aby znacznie obniżyć koszty. Jedną z możliwości jest użycie indeksu na atrybucie lub atrybutach z R uwzględnianych w złączeniu, aby znaleźć krotki z R pasujące do danej krotki z S bez konieczności wczytywania całej relacji R . Złączenia oparte na indeksach omówiono w punkcie 15.6.3. Drugie usprawnienie wymaga starannego określenia podziału krotek z R i S na bloki. Należy wykorzystać maksymalnie dużo pamięci, aby zmniejszyć liczbę dyskowych operacji wejścia-wyjścia przy przechodzeniu przez pętlę wewnętrzną. W punkcie 15.3.3 omówiono wersję złączeń zagnieżdżonych opartą na blokach.

15.3.2. Iterator dla krotkowych złączeń zagnieżdżonych

Zaletą złączeń zagnieżdżonych jest to, że dobrze wpasowują się w model iteratorów, co — jak opisano w punkcie 16.7.3 — w niektórych sytuacjach pozwala uniknąć zapisywania relacji pośrednich na dysku. Iterator dla operacji $R \bowtie S$ można łatwo utworzyć na podstawie iteratorów dla R i S , które obsługują metody `R.Open()` i inne opisane w punkcie 15.1.6. Kod trzech metod iteratora dla złączeń zagnieżdżonych pokazano na listingu 15.7 (przyjęto założenie, że R i S nie są puste).

```
Open() {
    R.Open();
    S.Open();
    s := S.GetNext();
}

GetNext() {
    REPEAT {
        r := R.GetNext();
        IF (r = NotFound) { /* Pobrano całą relację R dla
                           obecnego s. */
            R.Close();
            s := S.GetNext();
            IF (s = NotFound) RETURN NotFound;
            /* Pobrano całą relację R i S. */
            R.Open();
            r := R.GetNext();
        }
    }
    UNTIL (r i s zostaną złączone);
    RETURN złączenie r i s;
}

Close() {
    R.Close();
    S.Close();
}
```

15.7. Metody iteratora dla krotkowych złączeń zagnieżdżonych dla relacji R i S

15.3.3. Algorytm złączenia zagnieżdżonego opartego na blokach

Można usprawnić przedstawione w punkcie 15.3.1 krotkowe złączenie zagnieżdżone. Przy obliczaniu $R \bowtie S$ warto zadbać o:

- (1) zapewnienie dostępu do obu argumentów (relacji) według bloków,
- (2) wykorzystanie maksymalnej ilości pamięci głównej do zapisania krotek należących do relacji S (relacji z pętli zewnętrznej).

Punkt 1. gwarantuje, że przy przechodzeniu w wewnętrznej pętli po krotkach R do czytania R potrzebna będzie jak najmniejsza liczba dyskowych operacji wejścia-wyjścia. Punkt 2. pozwala złączyć każdą wczytaną krotkę z R z tyloma krotkami z S , ile mieści się w pamięci.

Tak jak w punkcie 15.2.3, założmy, że $B(S) \leq B(R)$. Przyjmijmy jednak, że $B(S) > M$ (żadna relacja nie mieści się w całości w pamięci głównej). Trzeba wielokrotnie wczytać $M - 1$ bloków z S do buforów pamięci głównej. Dla krotek S w pamięci głównej należy utworzyć wspomagającą wyszukiwanie strukturę, której klucze wyszukiwania to wspólne atrybuty R i S . Następnie trzeba przejść przez wszystkie bloki relacji R , wczytując po kolei każdy z nich do ostatniego bloku pamięci. Wtedy można porównać wszystkie krotki bloku R ze wszystkimi krotkami z bloków S wczytanych do pamięci głównej. Pasujące krotki należy zwrócić jako złączoną krotkę. Struktura pętli zagnieżdżonej z tego algorytmu jest widoczna w bardziej formalnym jego ujęciu na listingu 15.8. Algorytm z listingu jest czasem nazywany blokowym złączeniem zagnieżdżonym. Dalej nazywamy go po prostu *złączeniem zagnieżdżonym*, ponieważ jest to wersja rozwiązania z pętlą zagnieżdżoną najczęściej stosowana w praktyce.

```

FOR każdej porcji M-1 bloków S DO BEGIN
    wczytaj bloki do buforów pamięci głównej;
    uporządkuj krotki we wspomagającej wyszukiwanie strukturze,
    której kluczem wyszukiwania są wspólne atrybuty R i S;
    FOR każdego bloku b z R DO BEGIN
        wczytaj b do pamięci głównej;
        FOR każdej krotki t z b DO BEGIN
            znajdź w pamięci głównej krotki z S, które
            można złączyć z t;
            zwróć złączenie t z każdą z takich krotek;
        END;
    END;
END;

```

15.8. Algorytm złączenia zagnieżdżonego

Program z listingu 15.8 wygląda tak, jakby miał trzy pętle zagnieżdżone. Jeśli jednak spojrzeć na kod na odpowiednim poziomie abstrakcji, w rzeczywistości istnieją tylko dwie pętle. Pierwsza, zewnętrzna, przechodzi po krotkach z S . Dwie pozostałe przechodzą po krotkach z R — użyto tu dwóch pętli, aby podkreślić fakt, że kolejność przechodzenia po krotkach z R nie jest dowolna. Trzeba sprawdzać krotki blok po bloku (służy do tego druga pętla), a w bloku należy sprawdzić wszystkie krotki przed przejściem do następnego bloku (odpowiada za to trzecia pętla).

Przykład 15.4. Niech $B(R) = 1000$, $B(S) = 500$, a $M = 101$. 100 bloków pamięci posłuży do buforowania S w 100-blokowych częściach, dlatego zewnętrzna pętla z listingu 15.8 zostanie wykonana pięć razy. W każdej iteracji trzeba wykonać 100 dyskowych operacji wejścia-wyjścia do wczytania części relacji S , a ponieważ w drugiej pętli trzeba wczytać całą relację R , zajmuje to 1000 dyskowych operacji wejścia-wyjścia. Tak więc łączna liczba takich operacji to 5500.

Zauważmy, że po zamianie rolami relacji R i S algorytm będzie wymagał nieco więcej dyskowych operacji wejścia-wyjścia. Trzeba 10 razy wykonać zewnętrzną pętlę, a w każdym powtórzeniu wykonać 600 operacji, co w sumie daje ich 6000. Przetwarzanie mniejszej relacji w zewnętrznej pętli zwykle daje pewne korzyści. ■

15.3.4. Analiza złączeń zagnieżdżonych

Analizy z przykładu 15.4 można powtórzyć dla dowolnego $B(R)$, $B(S)$ i M . Przy założeniu, że S to mniejsza relacja, liczba jej części (iteracji zewnętrznej pętli) wynosi $B(S)/(M - 1)$. W każdej iteracji należy wczytać $M - 1$ bloków z S i $B(R)$ bloków z R . Liczba dyskowych operacji wejścia-wyjścia wynosi więc $B(S)(M - 1 + B(R))/(M - 1)$ lub $B(S) + (B(S)B(R))/(M - 1)$.

Przy założeniu, że M , $B(S)$ i $B(R)$ to duże wartości, jednak M jest najmniejszą z nich, przybliżeniem powyższego wzoru jest $B(S)B(R)/M$. Oznacza to, że koszt jest proporcjonalny do iloczynu rozmiarów obu relacji podzielonego przez ilość dostępnej pamięci głównej. Jeśli obie relacje są duże, można uzyskać znacznie wyższą wydajność niż dla złączeń zagnieżdżonych. Jednak dla małych relacji, takich jak w przykładzie 15.4, koszt dla takich złączeń nie jest dużo wyższy niż dla złączenia jednorazowego, które tu wymagałoby 1500 operacji. W rzeczywistości dla $B(S) \leq M - 1$ złączenie zagnieżdżone działa tak samo jak jednorazowy algorytm złączenia opisany w punkcie 15.2.3.

Choć złączenie zagnieżdżone zwykle nie jest najwydajniejszym algorytmem złączenia, warto zauważyć, że w niektórych wczesnych relacyjnych systemach DBMS była to jedyna dostępna metoda. Nawet obecnie jest ona czasem potrzebna jako podprocedura w wydajniejszych algorytmach złączenia, na przykład wtedy, kiedy duża liczba krotek z każdej relacji ma tę samą wartość atrybutów używanych przy złączeniu. Przykład, w którym złączenie zagnieżdżone jest niezbędne, przedstawiono w punkcie 15.4.6.

15.3.5. Przegląd opisanych wcześniej algorytmów

W tabeli 15.9 pokazano wymogi związane z pamięcią główną i dyskowymi operacjami wejścia-wyjścia dla algorytmów omówionych w podrozdziałach 15.2 i 15.3. Wymogi dotyczące pamięci dla operacji γ i δ są bardziej złożone — $M = B$ to tylko luźne przybliżenie. Dla γ M zależy od liczby grup, a dla δ — od liczby różnych krotek.

Operatory	Potrzebne M (w przybliżeniu)	Operacje dyskowe	Punkt
σ, π	1	B	15.2.1
γ, δ	B	B	15.2.2
$\cup, \cap, -, \times, \bowtie$	$\min(B(R), B(S))$	$B(R) + B(S)$	15.2.3
$\bowtie$	Dowolne $M \geq 2$	$B(R)B(S)/M$	15.3.3

15.9. Wymogi związane z pamięcią główną i dyskowymi operacjami wejścia-wyjścia dla algorytmów jednorazowych oraz zagnieżdżonych

15.3.6. Ćwiczenia do podrozdziału 15.3

Ćwiczenie 15.3.1. Przedstaw trzy metody iteratora dla wersji złączeń zagnieżdżonych opartej na blokach.

Ćwiczenie 15.3.2. Załóżmy, że $B(R) = B(S) = 10\,000$, a $M = 1000$. Oblicz koszt dyskowych operacji wejścia-wyjścia dla złączenia zagnieżdżonego.

Ćwiczenie 15.3.3. Używane są relacje z ćwiczenia 15.3.2. Jaka wartość M jest potrzebna do obliczenia $R \bowtie S$ za pomocą algorytmu zagnieżdżonego w nie więcej niż: a) 100 000, !b) 25 000, !c) 15 000 dyskowych operacji wejścia-wyjścia?

! Ćwiczenie 15.3.4. Może się wydawać, że jeśli R i S są niesklastrowane, złączenie zagnieżdżone wymaga około $T(R)T(S)/M$ dyskowych operacji wejścia-wyjścia.

- a) Jak można w istotny sposób zmniejszyć ten koszt?
- b) Jak można przeprowadzić złączenie zagnieżdżone, jeśli tylko jedna relacja (R lub S) jest niesklastrowana? Rozważ zarówno przypadek, kiedy niesklastrowana jest większa relacja, jak i sytuację, kiedy niesklastrowana jest mniejsza relacja.

! Ćwiczenie 15.3.5. Iterator z listingu 15.7 nie działa poprawnie, jeśli R lub S jest pusta. Zmodyfikuj metody, tak aby działały nawet wtedy, kiedy jedna lub obie relacje są puste.

15.4. Algorytmy dwuprzebiegowe oparte na sortowaniu

Od tego miejsca rozpoczyna się omówienie algorytmów wieloprzebiegowych. Służą one do wykonywania operacji algebry relacji na relacjach zbyt dużych dla jednoprzebiegowych algorytmów z podrozdziału 15.2. Skoncentrowano się tu na *algorytmach dwuprzebiegowych*, w których dane z operandów (relacji) są wczytywane do pamięci głównej, przetwarzane w pewien sposób, ponownie zapisywane na dysku, a następnie powtórnie wczytywane z dysku w celu dokończenia operacji. Naturalnie można rozwinąć tę technikę do dowolnej liczby przebiegów, w których dane są wielokrotnie wczytywane do pamięci głównej. Skoncentrowano się jednak na algorytmach dwuprzebiegowych, ponieważ:

- a) dwa przebiegi to zwykle wystarczająca liczba nawet dla bardzo dużych relacji,
- b) uogólnienie na więcej niż dwa przebiegi nie jest trudne. Omówiono je w punkcie 15.4.1 i — bardziej ogólnie — w podrozdziale 15.8.

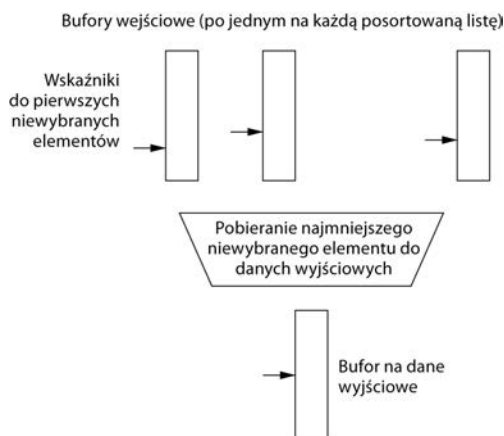
Na początku opisano implementację operatora sortowania, τ , aby zilustrować ogólne podejście — podział relacji R , dla której $B(R) > M$, na porcje o wielkości M , sortowanie ich, a następnie przetwarzanie posortowanych podlist, tak że w danym momencie w pamięci głównej potrzebny jest tylko jeden blok z posortowaną podlistą.

15.4.1. Dwuetapowe wielościeżkowe sortowanie przez scalanie

Bardzo duże relacje można sortować w dwóch przebiegach, używając algorytmu *dwuetapowego wielościeżkowego sortowania przez scalanie* (DWSPS; ang. *two-phase, multiway merge sort*). Załóżmy, że przy sortowaniu można użyć M buforów w pamięci głównej. DWSPS sortuje relację R w następujący sposób.

- *Etap 1.* Należy wielokrotnie zapełnić M buforów nowymi krotkami z R i posortować je, używając algorytmu sortowania w pamięci głównej. Każdą *posortowaną podlistę* trzeba zapisać w pamięci drugiego stopnia.
- *Etap 2.* Należy scalić posortowane podlisty. Ten etap wymaga, aby istniało co najwyżej $M - 1$ posortowanych podlist, co ogranicza rozmiar R . Potrzebny jest jeden blok wyjściowy na każdą posortowaną podlistę i jeden blok na dane wyjściowe. Wykorzystanie buforów pokazano na rysunku 15.10. Wskaźnik do każdego bloku wyjściowego określa pierwszy z posortowanych elementów, którego nie przeniesiono jeszcze do danych wyjściowych. Posortowane podlisty scalane są w jedną posortowaną listę wszystkich rekordów w następujący sposób.

- (1) Znajdowanie najmniejszego klucza wśród pierwszych niewybranych elementów na wszystkich listach. Ponieważ porównywanie odbywa się w pamięci głównej, odpowiednio jest wyszukiwanie liniowe. Liczba instrukcji maszynowych jest proporcjonalna do liczby podlist. Jeśli jednak trzeba, można zastosować metodę opartą na kolejkach priorytetowych². Czas znajdowania najmniejszego elementu jest wtedy proporcjonalny do logarytmu liczby podlist.
- (2) Przenoszenie najmniejszego elementu na pierwszą dostępną pozycję w bloku danych wyjściowych.
- (3) Jeśli blok danych wyjściowych jest pełny, należy zapisać go na dysku i ponownie zainicjować ten sam bufor w pamięci głównej dla następnego bloku wyjściowego.
- (4) Jeżeli blok, z którego pobrano najmniejszy element, jest już pusty, należy wczytać kolejny blok tej samej posortowanej podlisty do bufora zajmowanego przez opróżniony blok. Jeśli nie ma więcej bloków, bufor pozostaje pusty i przy dalszym wyszukiwaniu najmniejszego z pozostałych elementów nie należy uwzględniać elementów z tej listy.



15.10. Uporządkowanie pamięci głównej dla wielościeżkowego scalania

² Zobacz: A. V. Aho i J. D. Ullman, *Foundations of Computer Science*, Computer Science Press, 1992.

Aby DWSPS działał, nie może istnieć więcej niż $M - 1$ podlist. Załóżmy, że R mieści się w B blokach. Ponieważ każda podlista składa się z M bloków, liczba podlist wynosi B/M . Dlatego konieczne jest, aby: $B/M \leq M - 1$ lub $B \leq M(M - 1)$ (około $B \leq M^2$).

Algorytm wymaga wczytania w pierwszym przebiegu B bloków i kolejnych B dyskowych operacji wejścia-wyjścia na zapis posortowanych podlist. Każda posortowana podlista jest ponownie wczytywana w drugim przebiegu, co daje w sumie $3B$ operacji. Jeśli — jak zwykle — pomijamy koszty zapisu wyniku na dysku (ponieważ wynik można przekazywać w potoku bez zapisywania na dysku), $3B$ to wszystkie operacje potrzebne dla operatora sortowania τ . Jeżeli jednak trzeba zapisać wynik na dysku, niezbędnych jest $4B$ operacji.

Przykład 15.5. Załóżmy, że bloki mają po 64 000 bajtów, a pamięć główna ma jeden gigabajt pojemności. M ma więc wartość 16 000. Dlatego relację mieszczącą się w B blokach można posortować, o ile B nie wynosi więcej niż $(16\,000)^2 = 2^{28}$. Ponieważ bloki mają rozmiar $64\,000 = 2^{16}$, relację można posortować, o ile ma nie więcej niż 2^{44} bajtów, czyli 16 terabajtów. ■

W przykładzie 15.5 pokazano, że nawet w przeciętnym komputerze DWSPS wystarcza do posortowania w dwóch przebiegach wszystkich relacji, oprócz niezwykle dużych. Jeśli jednak nawet relacja jest większa, ten sam pomysł można zastosować rekurencyjnie. Należy podzielić relację na porcje o wielkości $M(M - 1)$, wykorzystać DWSPS do posortowania każdej z nich, a następnie potraktować uzyskane posortowane listy jako podlisty w trzecim przebiegu. W podobny sposób rozwiązanie można rozwinąć do dowolnej liczby przebiegów.

15.4.2. Eliminowanie powtórzeń za pomocą sortowania

Aby wykonać operację $\delta(R)$ w dwóch przebiegach, należy posortować krotki z R w podlistach, tak jak w DWSPS. W drugim przebiegu dostępną pamięć główną trzeba wykorzystać na przechowywanie jednego bloku z każdej posortowanej podlisty i jednego bloku wyjściowego, tak jak w DWSPS. Jednak zamiast sortować dane w drugim przebiegu, należy wielokrotnie pobierać pierwszą (w kolejności sortowania) niesprawdzoną krotkę t z wszystkich posortowanych podlist. W danych wyjściowych trzeba zapisać jedną kopię t , a z bloków wejściowych usunąć wszystkie wystąpienia t . W ten sposób dane wyjściowe będą zawierać dokładnie jedną kopię każdej krotki z R (krotki te będą posortowane). Kiedy blok wyjściowy jest pełen (lub blok wejściowy jest pusty), zarządzanie buforami odbywa się tak jak w DWSPS.

Liczba dyskowych operacji wejścia-wyjścia w tym algorytmie (jak zawsze, z pominięciem obsługi danych wyjściowych) jest taka sama jak przy sortowaniu — $3B(R)$. Warto porównać tę liczbę z wartością $B(R)$ dla algorytmu jednoprzebiegowego z punktu 15.2.2. Jednak algorytm dwuprzebiegowy działa dla dużo większych plików niż algorytm jednoprzebiegowy. W DWSPS dla algorytmu dwuprzebiegowego obowiązuje wymóg $B \leq M^2$, natomiast dla algorytmu jednoprzebiegowego — $B \leq M$. Ujmijmy to inaczej — wyeliminowanie powtórzeń za pomocą algorytmu dwuprzebiegowego wymaga tylko $\sqrt{B(R)}$ bloków pamięci głównej w porównaniu z $B(R)$ bloków niezbędnych w algorytmie jednoprzebiegowym.

15.4.3. Grupowanie i agregacja z wykorzystaniem sortowania

Algorytm dwuprzebiegowy dla operacji $\gamma_L(R)$ przypomina algorytm dla operacji $\delta(R)$ lub DWSPS. Oto jego krótki opis.

- (1) Należy wczytać do pamięci krotki z R po M bloków na raz. Krotki trzeba posortować w każdym zbiorze M bloków, używając jako klucza sortowania atrybutów grupujących z L . Każda posortowana podlista zapisywana jest na dysku.
- (2) Trzeba użyć jednego bufora pamięci głównej na każdą podlistę i początkowo wczytać do bufora pierwszy blok każdej podlisty.
- (3) Należy wielokrotnie znajdować najmniejszą wartość klucza sortowania (atrybutów grupujących) wśród pierwszych dostępnych krotek z buforów. Wartość ta, v , staje się następną grupą, dla której:
 - (a) następują przygotowania do obliczenia wszystkich agregacji z listy L tej grupy; tak jak w punkcie 15.2.2, można użyć liczby elementów i sumy zamiast średniej,
 - (b) sprawdzana jest każda krotka o kluczu sortowania v i następuje akumulacja wartości dla potrzebnych agregacji,
 - (c) jeśli bufor staje się pusty, należy zastąpić go następnym blokiem z danej podlisty.

Kiedy nie ma już więcej krotek o kluczu sortowania v , należy zwrócić krotkę składającą się z atrybutów grupujących z L i powiązanych wartości agregacji obliczonych dla grupy.

Dwuprzebiegowy algorytm dla δ , podobnie jak dla γ , wymaga $3B(R)$ dyskowych operacji wejścia-wyjścia i działa dla $B(R) \leq M^2$.

15.4.4. Algorytm obliczania sumy oparty na sortowaniu

Kiedy potrzebna jest suma wielozbiorów, jednoprzebiegowy algorytm z punktu 15.2.3 (w którym po prostu kopiowane są obie relacji) działa niezależnie od wielkości argumentów, dlatego nie trzeba rozważać dwuprzebiegowego algorytmu dla $\cup_W$. Jednak jednoprzebiegowy algorytm dla $\cup_Z$ funkcjonuje tylko wtedy, kiedy przynajmniej jedna relacja mieści się w pamięci głównej, dlatego trzeba poznać algorytm dwuprzebiegowy dla sumy zbiorów. Opisana metoda działa dla części wspólnej i różnicy zbiorów oraz wielozbiorów, co opisano w punkcie 15.4.5. Aby obliczyć $R \cup_Z S$, należy zmodyfikować DWSPS w następujący sposób.

- (1) W pierwszym kroku na podstawie R i S trzeba utworzyć posortowane podlisty.
- (2) Należy użyć po jednym buforze pamięci głównej na każdą podlistę z R i S oraz zainicjować bufor pierwszy blokiem odpowiedniej podlisty.
- (3) Trzeba wielokrotnie znajdować pierwszą krotkę t z wszystkich buforów, kopiować t do danych wyjściowych i usuwać z buforów wszystkie jej kopie (jeśli R i S to zbiory, istnieją najwyżej dwie kopie). Zarządzanie pustymi buforami wejściowymi i pełnym buforem wyjściowym odbywa się tak jak w DWSPS.

Zauważmy, że każda krotka z R i S jest wczytywana do pamięci głównej dwukrotnie — przy tworzeniu podlist i jako element jednej z nich. Krotka jest ponadto raz zapisywana na dysku jako część nowo utworzonej podlisty. Dlatego koszt dyskowych operacji wejścia-wyjścia wynosi $3(B(R) + B(S))$.

Algorytm działa, o ile łączna liczba podlist dla dwóch relacji nie przekracza $M - 1$, ponieważ potrzebny jest jeden bufor na każdą podlistę i jeden na dane wyjściowe. Dlatego suma rozmiarów obu relacji nie może przekraczać $M^2 - B(R) + B(S) \leq M^2$.

15.4.5. Obliczanie za pomocą sortowania części wspólnej i różnicy

Niezależnie od wersji (dla zbiorów lub wielozbiorów) algorytmy działają w zasadzie tak samo jak w punkcie 15.4.4, natomiast inaczej traktowane są kopie krotki t na początku posortowanych podlist. W każdym algorytmie używana jest krotka t najmniejsza (w kolejności sortowania) spośród wszystkich pozostałych w buforach wejściowych. Należy utworzyć dane wyjściowe w opisany sposób, a następnie usunąć wszystkie kopie t z buforów wejściowych.

- Dla części wspólnej zbiorów należy zwrócić t , jeśli występuje zarówno w R , jak i w S .
- Dla części wspólnej wielozbiorów należy zwrócić t tyle razy, ile razy minimalnie występuje w R i S . Zauważmy, że t nie jest zwracana, jeśli liczba ta wynosi 0 (czyli t nie występuje w jednej lub obu relacjach).
- Dla różnicy zbiorów ($R -_Z S$) należy zwrócić t wtedy i tylko wtedy, jeśli występuje w R , ale nie w S .
- Dla różnicy wielozbiorów ($R -_W S$) należy zwrócić t tyle razy, ile występuje w R , minus liczbę jej wystąpień w S . Oczywiście, jeśli t pojawia się w S przynajmniej tyle razy, co w R , w ogóle nie należy jej zwracać.

W operacjach na wielozbiorach trzeba pamiętać o pewnym drobiazgu. Przy liczeniu wystąpień t może się okazać, że wszystkie pozostałe krotki w buforze wejściowym to t . Wtedy w następnym bloku podlisty mogą pojawiać się dalsze t . Dlatego jeśli w buforze pozostają same t , trzeba wczytać kolejny blok podlisty i kontynuować liczenie t . Proces ten może obejmować kilka bloków i podlist.

Analizy dla tej rodziny algorytmów są takie same jak dla algorytmu obliczania sumy zbiorów opisanego w punkcie 15.4.4. Potrzeba:

- $3(B(R) + B(S))$ dyskowych operacji wejścia-wyjścia,
- w przybliżeniu $B(R) + B(S) \leq M2$, aby algorytm mógł działać.

15.4.6. Prosty algorytm złączenia oparty na sortowaniu

Istnieje kilka sposobów na wykorzystanie sortowania do złączenia dużych relacji. Przed analizą algorytmów zwróćmy uwagę na problem, który może wystąpić przy złączeniu, choć nie dotyczył wcześniejszych operacji dwuargumentowych. Przy złączeniu liczba krotek relacji, które mają tę samą wartość porównywanych atrybutów, a tym samym muszą jednocześnie znajdować się w pamięci głównej, może przekroczyć pojemność pamięci. Skrajny przypadek występuje, kiedy porównywane atrybuty mają tylko jedną wartość, dlatego każdą krotkę jednej relacji trzeba złączyć z wszystkimi krotkami drugiej. Wtedy nie ma innego wyboru, jak zastosowanie złączenia zagnieżdżonego dla dwóch zbiorów krotek o identycznych wartościach porównywanych atrybutów.

Aby tego uniknąć, można spróbować zmniejszyć wykorzystanie pamięci głównej przez inne elementy algorytmu i udostępnić dużą liczbę buforów na krotki o danej wartości porównywanych atrybutów. W tym punkcie opisano algorytm, który zapewnia największą liczbę buforów na takie krotki. W punkcie 15.4.8 omówiono inny algorytm oparty na sortowaniu, który wymaga mniej dyskowych operacji wejścia-wyjścia, ale może prowadzić do problemów przy dużej liczbie krotek o tej samej wartości porównywanych atrybutów.

Przy złączeniu relacji $R(X, Y)$ i $S(Y, Z)$ oraz M blokach pamięci głównej na buforory należy wykonać następujące kroki.

- (1) Posortować R algorytmem DWSPS z Y jako kluczem sortującym.
- (2) W podobny sposób posortować S .
- (3) Scaić posortowane R i S . Wystarczą do tego dwa buforory — jeden na aktualny blok z R i drugi na aktualny blok z S . Poniższe kroki są wykonywane wielokrotnie.
 - (a) Wyszukiwanie najmniejszej wartości y porównywanych atrybutów Y na początku bloków R i S .
 - (b) Jeśli y nie pojawia się na początku drugiej relacji, należy usunąć krotki o kluczu sortowania y .
 - (c) W przeciwnym razie w obu relacjach trzeba znaleźć wszystkie krotki o kluczu sortowania y . Jeśli trzeba, należy wczytywać bloki z posortowaną relacją R i (lub) S do momentu, kiedy wiadomo, że w żadnej relacji nie ma już y . Można do tego wykorzystać M buforów.
 - (d) Zwracanie wszystkich krotek utworzonych przez złączenie krotek z R i S o identycznej wartości y atrybutów Y .
 - (e) Jeśli w pamięci głównej nie ma niesprawdzonych krotek z którejś z relacji, należy dla tej relacji ponownie wczytać dane do bufora.

Przykład 15.6. Rozważmy relacje R i S z przykładu 15.4. Przypominamy, że relacje zajmują 1000 i 500 bloków, a liczba buforów w pamięci głównej to $M = 101$. Przy stosowaniu dla relacji algorytmu DWSPS i zapisywaniu wyniku na dysku potrzeba czterech dyskowych operacji wejścia-wyjścia na blok (po dwie na każdy etap). Dlatego sortowanie R i S wymaga $4(B(R) + B(S))$ (6000) dyskowych operacji wejścia-wyjścia.

Przy scalaniu posortowanych R i S w celu znalezienia złączonych krotek każdy blok z R i S wczytywany jest po raz piąty, co daje kolejnych 1500 operacji. Przy scalaniu zwykle potrzebne są tylko dwa spośród 101 bloków pamięci. Jeśli jednak trzeba, można wykorzystać wszystkie 101 bloków na krotki R i S o wspólnej wartości y atrybutów Y . Dlatego wystarczy, jeśli dla żadnego y krotki z R i S o tej wartości atrybutów Y nie zajmują razem więcej niż 101 bloków.

Zauważmy, że łączna liczba dyskowych operacji wejścia-wyjścia w tym algorytmie to 7500 (złączenie zagnieżdżone z przykładu 15.4 wymagało ich 5500). Jednak złączenie zagnieżdżone to z natury algorytm o złożoności kwadratowej, działający w czasie proporcjonalnym do $B(R)B(S)$, natomiast koszt operacji wejścia-wyjścia w złączeniu przez sortowanie rośnie liniowo, proporcjonalnie do $B(R) + B(S)$. Jedynie stałe czynniki i mały rozmiar przykładowych relacji (każda jest tylko 5 do 10 razy większa od pojemności przydzielonych buforów) sprawiają, że tu złączenie zagnieżdżone jest wydajniejsze. ■

15.4.7. Analiza prostego złączenia przez sortowanie

Jak wspomniano w przykładzie 15.6, algorytm z punktu 15.4.6 wykonuje pięć dyskowych operacji wejścia-wyjścia na każdy blok relacji podanych jako argumenty. Trzeba też ustalić, jak duże musi być M , aby proste złączenie przez sortowanie działało. Podstawowe ograniczenie dotyczy tego, że możliwe musi być przeprowadzenie dla relacji R i S dwuetapowego wielościeżkowego sortowania przez scalanie. W punkcie 15.4.1 stwierdzono, że wymaga to, aby $B(R) \leq M^2$ i $B(S) \leq M^2$. Ponadto konieczne jest, aby wszystkie krotki o tej samej wartości atrybutów Y mieściły się w M buforach. Podsumujmy.

- Proste złączenie przez sortowanie wymaga $5(B(R) + B(S))$ dyskowych operacji wejścia-wyjścia.
- Do działania konieczne jest, aby: $B(R) \leq M^2$ i $B(S) \leq M^2$.
- Krotki o tej samej wartości porównywanych atrybutów muszą mieścić się w M blokach.

15.4.8. Wydajniejsze złączenie przez sortowanie

Jeśli bardzo duża liczba krotek o identycznej wartości złączanych atrybutów nie stanowi zagrożenia, można zaoszczędzić dwie dyskowe operacje wejścia-wyjścia na blok przez połączenie drugiego etapu sortowania z samym złączeniem. Ten algorytm to *złączenie przez sortowanie* (ang. *sort join*; inne nazwy to złączenie przez scalanie lub złączenie przez sortowanie ze scalaniem). Aby obliczyć $R(X, Y) \bowtie S(Y, Z)$ z wykorzystaniem M buforów w pamięci głównej, należy wykonać poniższe kroki.

- (1) Utworzyć posortowane podlisty o wielkości M przy użyciu Y jako klucza sortowania dla R i S .
- (2) Przenieść pierwszy blok każdej podlisty do bufora. Zakładamy, że liczba podlist nie przekracza M .
- (3) Wielokrotnie znaleźć najmniejszą wartość y atrybutów Y wśród pierwszych dostępnych krotek wszystkich podlist. Trzeba znaleźć wszystkie krotki z obu relacji o wartości y , na przykład używając niektórych z M dostępnych buforów na ich zapisanie, jeśli istnieje mniej niż M podlist. Należy zwrócić złączenie wszystkich krotek z R z wszystkimi krotkami z S mającymi tę samą wartość atrybutów Y . Jeśli bufor jednej z podlist zostanie opróżniony, trzeba usunąć go z dysku.

Przykład 15.7. Ponownie wróćmy do problemu z przykładu 15.4 — złączenia relacji R i S o rozmiarach 1000 oraz 500 bloków przy dostępnych 101 buforach. Należy podzielić R na 10, a S — na 5 podlist (każda o długości 100) i je posortować³. 15 buforów można wykorzystać na analizowane bloki każdej z podlist. Jeśli zdarzy się sytuacja, w której wiele krotek ma tę samą wartość atrybutów Y , można zapisać je w pozostałych 86 buforach.

Na każdy blok danych wykonywane są trzy dyskowe operacje wejścia-wyjścia. Dwie z nich służą do tworzenia posortowanych podlist. Następnie każdy blok każdej takiej podlisty należy ponownie wczytać do pamięci głównej w procesie wielościeżkowego scalania. Dlatego łączna liczba dyskowych operacji wejścia-wyjścia to 4500. ■

Ten algorytm złączenia przez sortowanie jest wydajniejszy od algorytmu z punktu 15.4.6, choć nie zawsze można z niego skorzystać. W przykładzie 15.7 stwierdzono, że liczba dyskowych operacji wejścia-wyjścia wynosi tu $3(B(R) + B(S))$. Algorytm można zastosować do danych niemal tak dużych, jak w poprzednim algorytmie. Rozmiar posortowanych podlist to M bloków, a na dwóch listach może być najwyżej M podlist. Dlatego wystarczający jest wymóg: $B(R) + B(S) \leq M^2$.

³ Technicznie można uporządkować podlisty, aby miały długość po 101 bloków. Wtedy ostatnia podlista z R ma 91 bloków, a ostatnia podlista z S — 96 bloków. Jednak wtedy koszty są dokładnie takie same.

15.4.9. Omówienie algorytmów opartych na sortowaniu

W tabeli 15.11 zamieszczono analizy algorytmów omówionych w podrozdziale 15.4. W punktach 15.4.6 i 15.4.8 napisano, że w algorytmach złączenia obowiązują ograniczenia dotyczące liczby krotek o tej samej wartości porównywanych atrybutów. Przy naruszeniu ograniczenia konieczne może być zastosowanie złączenia zagnieżdżonego.

Operatory	Potrzebne M (w przybliżeniu)	Dyskowe operacje wejścia-wyjścia	Punkt
τ, γ, δ	$\sqrt{B}$	$3B$	15.4.1, 15.4.2, 15.4.3
$\cup, \cap, -$	$\sqrt{B(R) + B(S)}$	$3(B(R) + B(S))$	15.4.4, 15.4.5
$\bowtie$	$\sqrt{\max(B(R), B(S))}$	$5(B(R) + B(S))$	15.4.6
$\Join$	$\sqrt{B(R) + B(S)}$	$3(B(R) + B(S))$	15.4.8

15.11. Wymogi związane z pamięcią główną i dyskowymi operacjami wejścia-wyjścia dla algorytmów opartych na sortowaniu

15.4.10. Ćwiczenia do podrozdziału 15.4

Ćwiczenie 15.4.1. Dla każdej z podanych operacji napisz iterator, który wykorzystuje algorytm opisany w podrozdziale: a) eliminowanie powtórzeń (δ), b) grupowanie (γ_L), c) część wspólna zbiorów, d) różnica wielozbiorów, e) złączenie naturalne.

Ćwiczenie 15.4.2. Jeśli $B(R) = B(S) = 10\,000$ i $M = 1000$, jakie są wymogi dotyczące dyskowych operacji wejścia-wyjścia dla: a) sumy zbiorów, b) prostego złączenia przez sortowanie, c) wydajniejszego złączenia przez sortowanie opisanego w punkcie 15.4.8.

! Ćwiczenie 15.4.3. Załóżmy, że drugi przebieg algorytmu opisanego w podrozdziale nie wymaga wszystkich M buforów, ponieważ istnieje mniej niż M podlist. Jak można zmniejszyć liczbę dyskowych operacji wejścia-wyjścia, wykorzystując dodatkowe bufory?

! Ćwiczenie 15.4.4. W przykładzie 15.6 omówiono złączenie dwóch relacji R i S o 1000 bloków oraz 500 blokach dla $M = 101$. Jednak potrzebne są dodatkowe dyskowe operacje wejścia-wyjścia, jeśli istnieje tyle krotek o danej wartości, że krotki z żadnej z relacji nie mieszczą się w pamięci głównej. Określ łączną liczbę operacji potrzebnych, jeśli:

- istnieją tylko dwie wartości atrybutów Y , a każda pojawia się w połowie krotek z R i w połowie krotek z S (Y to atrybut lub atrybuty, na których oparte jest złączenie),
- istnieją pięć wartości atrybutów Y , a każda występuje w obu relacjach równie często,
- istnieje 10 wartości atrybutów Y , a każda występuje w obu relacjach równie często.

! Ćwiczenie 15.4.5. Wykonaj ćwiczenie 15.4.4 dla wydajniejszego złączenia przez sortowanie, opisanego w punkcie 15.4.8.

Ćwiczenie 15.4.6. Ile pamięci potrzeba, aby można zastosować dwuprzebiegowy oparty na sortowaniu algorytm dla relacji obejmujących po 10 000 bloków, jeśli operacją jest: a) δ , b) γ , c) operacja dwuargumentowa, na przykład złączenie lub suma.

Ćwiczenie 15.4.7. Opisz dwuprzebiegowy oparty na sortowaniu algorytm dla każdego z operatorów z rodziny złączeń wymienionego w ćwiczeniu 15.2.4.

! Ćwiczenie 15.4.8. Załóżmy, że rekordy mogą być większe niż bloki (występują rekordy dzielone). Jak zmienia to wymogi dotyczące pamięci dla dwuprzebiegowych algorytmów opartych na sortowaniu?

!! Ćwiczenie 15.4.9. Czasem można zaoszczędzić kilka dyskowych operacji wejścia-wyjścia, zostawiając ostatnią podlistę w pamięci. W celu wykorzystania tego podejścia sensowne może być nawet zastosowanie podlist o mniej niż M blokach. Ile dyskowych operacji wejścia-wyjścia można zaoszczędzić w ten sposób?

15.5. Dwuprzebiegowe algorytmy oparte na haszowaniu

Istnieje rodzina opartych na haszowaniu algorytmów, które rozwiązują problemy omówione w podrozdziale 15.4. Wszystkie te algorytmy oparte są na podstawowym pomysłe — jeśli dane są za duże, aby zmieścić je w buforach pamięci głównej, należy utworzyć skrót każdej krotki z argumentu (lub argumentów), używając odpowiedniego klucza haszującego. We wszystkich standardowych operacjach istnieje sposób na taki dobór klucza haszującego, aby wszystkie krotki, które trzeba przetwarzać wspólnie, trafiały do tego samego kubeczka.

Następnie operacje wykonywane są kubeczek po kubeczku (lub na parach kubeczków o tej samej wartości skrótu w operacjach dwuargumentowych). W efekcie wielkość operandów zmniejsza się tyle razy, ile jest kubeczków (czyli mniej więcej M razy). Zauważmy, że w opisanych w podrozdziale 15.4 algorytmach opartych na sortowaniu można osiągnąć podobne korzyści przez wstępne przetwarzanie, choć w podejściach związanych z sortowaniem i haszowaniem służą do tego różne środki.

15.5.1. Podział relacji przez haszowanie

Zacznijmy od omówienia podziału (z wykorzystaniem M buforów) relacji R na $M - 1$ kubeczków o mniej więcej równej wielkości. Funkcja h to funkcja haszująca, przyjmująca jako argument kompletne krotki z R (wszystkie atrybuty z R są częścią klucza haszującego). Każdy kubeczek wiązany jest z jednym buforem. Ostatni bufor przechowuje kolejne bloki z R . Należy obliczyć skrót $h(t)$ dla każdej krotki t z tego bloku i skopiować ją do odpowiedniego bufora. Jeśli bufor jest pełny, należy zapisać go na dysku i dla tego samego kubeczka zainicjować inny blok. Ostatecznie zapisywany jest ostatni blok każdego kubeczka (jeśli nie jest pusty). Bardziej szczegółowy opis algorytmu znajduje się na listingu 15.12.

```

zainicjuj M-1 kubełków za pomocą M-1 pustych buforów;
FOR każdego bloku b z relacji R DO BEGIN
    wczytaj blok b do M-tego bufora;
    FOR każdej krotki t z b DO BEGIN
        IF w buforze dla kubełka h(t) nie ma miejsca na t THEN
            BEGIN
                skopiuj bufor na dysk;
                zainicjuj nowy pusty blok w danym buforze;
            END;
        skopiuj t do bufora dla kubełka h(t);
    END;
END;
FOR każdego kubełka DO
    IF bufor dla kubełka nie jest pusty THEN
        zapisz bufor na dysku;

```

15.12. Podział relacji R na $M - 1$ kubełków

15.5.2. Algorytm usuwania powtórzeń oparty na haszowaniu

Rozważmy teraz szczegóły opartych na haszowaniu algorytmów dla różnych operacji algebry relacji, które mogą wymagać algorytmów dwuprzebiegowych. Po pierwsze, rozważmy usuwanie powtórzeń — $\delta(R)$. Za pomocą haszowania należy podzielić R na $M - 1$ kubełków, tak jak na listingu 15.12. Zauważmy, że dwie kopie tej samej krotki t trafiają do tego samego kubełka. Dlatego można przetwarzać kubełek po kubełku, wykonywać δ na każdym kubełku osobno i obliczyć odpowiedź jako sumę $\delta(R_i)$, gdzie R_i to część relacji R z i -tego kubełka. Jednoprzebiegowy algorytm z punktu 15.2.2 można zastosować do usunięcia powtórzeń kolejno z każdego R_i i zapisania uzyskanych niepowtarzalnych krotek.

Ta metoda działa, o ile poszczególne R_i są na tyle małe, że mieszczą się w pamięci głównej, co pozwala zastosować algorytm jednoprzebiegowy. Ponieważ można przyjąć, że funkcja haszująca h dzieli R na kubełki o równym rozmiarze, każda R_i zajmuje około $B(R)/(M - 1)$ bloków. Jeśli liczba bloków jest nie większa niż M ($B(R) \leq M(M - 1)$), można użyć dwuprzebiegowego algorytmu opartego na haszowaniu. Jak opisano to w punkcie 15.2.2, konieczne jest tylko, aby liczba różnych krotek w jednym kubełku mieściła się w M buforach. Dlatego według ostrożnych szacunków (przy założeniu, że M i $M - 1$ to prawie to samo) wymagane jest, aby $B(R) \leq M^2$, dokładnie tak, jak w opartym na sortowaniu dwuprzebiegowym algorytmie dla δ .

Liczba dyskowych operacji wejścia-wyjścia jest także podobna do tej w algorytmie opartym na sortowaniu. Każdy blok z R wczytywany jest raz przy haszowaniu krotek, a ponadto każdy blok wszystkich kubełków zapisywany jest na dysku. Następnie należy ponownie wczytać każdy blok wszystkich kubełków w jednoprzebiegowym algorytmie działającym dla danego kubełka. Dlatego łączna liczba dyskowych operacji wejścia-wyjścia to $3B(R)$.

15.5.3. Grupowanie i agregacja oparte na haszowaniu

Aby wykonać operację $\gamma_L(R)$, należy ponownie zacząć od haszowania wszystkich krotek z R w celu zapisania ich w $M - 1$ kubełkach. Jednak żeby się upewnić, że wszystkie krotki z tej samej grupy znajdują się w jednym kubełku, trzeba wybrać funkcję haszującą opartą wyłącznie na atrybutach grupujących z listy L .

Po podzieleniu R na kubełki można wykorzystać jednoprzebiegowy algorytm dla γ (opisany w punkcie 15.2.2) do przetworzenia po kolei każdego kubełka. Jak wyjaśniono w kontekście δ w punkcie 15.5.2, każdy kubełek można przetworzyć w pamięci głównej, pod warunkiem, że $B(R) \leq M^2$.

Jednak w drugim przebiegu przy przetwarzaniu każdego kubełka potrzebny jest tylko jeden rekord na grupę. Dlatego jeśli nawet rozmiar kubełka jest większy niż M , można obsłużyć kubełek w jednym przebiegu, pod warunkiem że rekordy dla wszystkich grup z kubełka zajmują nie więcej niż M buforów. Tak więc dla dużych grup można obsłużyć znacznie większe relacje R , niż określa to reguła $B(R) \leq M^2$. Jeśli jednak M jest większe niż liczba grup, nie można zapełnić wszystkich kubełków. Dlatego ograniczenie rozmiaru R za pomocą funkcji od M jest skomplikowane. Według ostrożnych szacunków ograniczenie to $B(R) \leq M^2$. Zauważmy też, że liczba dyskowych operacji wejścia-wyjścia dla γ , podobnie jak dla δ , to $3B(R)$.

15.5.4. Suma, część wspólna i różnica oparte na haszowaniu

W operacjach dwuargumentowych trzeba zastosować tę samą funkcję haszującą dla krotek z obu argumentów. Aby na przykład obliczyć $R \cup_Z S$, należy umieścić każdą z relacji R i S w $M - 1$ kubełkach (na przykład w $R_1, R_2, \dots, R_{M-1}$ i $S_1, S_2, \dots, S_{M-1}$). Następnie można obliczyć sumę zbiorów R_i i S_j dla wszystkich i oraz zwrócić wynik. Zauważmy, że jeśli krotka t występuje w R i S , dla pewnego i znajduje się zarówno w R_i , jak i w S_i . Dlatego przy obliczaniu sumy dwóch kubełków należy zwrócić tylko jedną kopię t i nie ma możliwości, że w wyniku pojawią się powtórzenia. Dla operacji $\cup_W$ zalecany jest prosty algorytm obliczania sumy wielozbiorów (punkt 15.2.3).

Aby obliczyć część wspólną lub różnicę R i S , należy — tak jak dla sumy zbiorów — utworzyć $2(M - 1)$ kubełków i zastosować odpowiedni algorytm jednoprzebiegowy do każdej pary powiązanych kubełków. Zauważmy, że wszystkie algorytmy jednoprzebiegowe wymagają $B(R) + B(S)$ dyskowych operacji wejścia-wyjścia. Do tej liczby trzeba dodać dwie dyskowe operacje wejścia-wyjścia na blok niezbędne do podziału przez haszowanie krotek z dwóch relacji i zapisania kubełków na dysku, co w sumie daje $3(B(R) + B(S))$ operacji.

Algorytmy wymagają do działania, aby w jednym przebiegu można było obliczyć sumę, część wspólną lub różnicę R_i i S_i , których rozmiary to w przybliżeniu $B(R)/(M - 1)$ oraz $B(S)/(M - 1)$. Przypominamy, że jednoprzebiegowe algorytmy dla tych operacji wymagają, żeby mniejszy operand mieścił się w $M - 1$ blokach. Dlatego dwuprzebiegowe algorytmy oparte na haszowaniu wymagają, aby — w przybliżeniu — $\min(B(R), B(S)) \leq M^2$.

15.5.5. Algorytm złączenia przez haszowanie

Aby obliczyć $R(X, Y) \bowtie S(Y, Z)$ za pomocą dwuprzebiegowego algorytmu opartego na haszowaniu, można postępować niemal tak samo jak przy innych operacjach dwuargumentowych, opisanych w punkcie 15.5.4. Jedyna różnica polega na tym, że jako klucza haszującego trzeba użyć tylko atrybutów uwzględnianych przy złączeniu (Y). Wiadomo wtedy, że złączane krotki R i S znajdują się w odpowiadających sobie kubełkach R_i i S_i dla pewnego i . Jednoprzebiegowe złączenie wszystkich par z powiązanych kubełków kończy algorytm (nazywamy go *złączeniem przez haszowanie*⁴).

⁴ Czasem nazwa złączenie przez haszowanie jest zarezerwowana dla odmiany jednoprzebiegowego algorytmu złączenia z punktu 15.2.3, w którym wspomagającą wyszukiwanie strukturą w pamięci głównej jest tablica z haszowaniem. Wtedy opisany tu algorytm dwuprzebiegowy nazywany jest złączeniem przez haszowanie z podziałem.

Przykład 15.8. Przypominamy dwie relacje R i S z przykładu 15.4, których wielkość to 1000 i 500 bloków. W pamięci głównej dostępnych jest 101 buforów. Za pomocą haszowania można zapisać każdą relację w 100 kubekach, tak że średnia wielkość kubeków to 10 (dla R) lub 5 bloków (dla S). Ponieważ mniejsza liczba, 5, jest znacznie niższa niż liczba dostępnych buforów, można oczekiwać, że jednoprzebiegowe złączenie każdej pary kubeków nie sprawi problemów.

Liczba dyskowych operacji wejścia-wyjścia wynosi 1500 przy wczytywaniu R i S w czasie haszowania, kolejne 1500 operacji potrzeba na zapis wszystkich kubeków na dysku, a trzecie 1500 operacji zajmuje ponowne wczytanie każdej pary kubeków do pamięci głównej w czasie jednoprzebiegowego złączenia powiązanych kubeków. Tak więc liczba potrzebnych operacji to 4500, tak jak dla wydajnego złączenia przez sortowanie z punktu 15.4.8. ■

Przykład 15.8 można uogólnić, stwierdzając, że:

- złączenie przez haszowanie wymaga $3(B(R) + B(S))$ dyskowych operacji wejścia-wyjścia,
- dwuprzebiegowy algorytm złączenia przez haszowanie działa, jeśli (w przybliżeniu) $\min(B(R), B(S)) \leq M^2$.

Ostatni punkt wynika z tego, co w innych operacjach dwuargumentowych — jeden kubek z każdej pary musi mieścić się w $M - 1$ buforach.

15.5.6. Zmniejszanie liczby dyskowych operacji wejścia-wyjścia

Jeśli w pierwszym przebiegu mamy dostęp do większej ilości pamięci niż potrzeba na przechowywanie jednego bloku na kubek, można zmniejszyć liczbę dyskowych operacji wejścia-wyjścia. Jedną z możliwości jest zastosowanie kilku bloków dla każdego kubka i zapisanie ich jako grupy w przyległych blokach na dysku. Ujmijmy to ściśle — technika ta nie zmniejsza liczby operacji, ale je przyspiesza, ponieważ przy zapisie można pominąć czas wyszukiwania i opóźnienie obrotowe.

Istnieje jednak kilka sztuczek stosowanych w celu uniknięcia zapisu niektórych kubeków na dysku i ponownego ich odczytu. Tu opisano najwydajniejszą technikę, *hybrydowe złączenie przez haszowanie*. Przyjmijmy, że w celu obliczenia złączenia $R \bowtie S$ (S to mniejsza relacja) trzeba utworzyć k kubeków, co wymaga znacznie mniej niż M pamięci (jest to ilość dostępnej pamięci). W czasie haszowania S można zdecydować się na przechowywanie m spośród k kubeków w całości w pamięci głównej i zachować tylko jeden blok na każdy z pozostałych $k - m$ kubeków. Jest to możliwe, pod warunkiem że oczekiwany rozmiar kubeków w pamięci plus jeden blok na każdy z pozostałych kubeków nie przekracza M , czyli:

$$mB(S)/k + k - m \leq M \quad (15.1)$$

Oto wyjaśnienie — oczekiwany rozmiar kubka to $B(S)/k$, a w pamięci znajduje się m kubeków.

Teraz, przy wczytywaniu krotek z drugiej relacji (R) w celu jej podziału na kubki w pamięci przechowywane są:

- (1) m niezapisanych na dysku kubeków z S

oraz

- (2) jeden blok dla każdego z $k - m$ kubeków z R , dla których powiązane kubki z S zapisano na dysku.

Jeśli krótka t z R trafia w wyniku haszowania do jednego z pierwszych m kubeków, można natychmiast złączyć ją z wszystkimi krotkami z powiązanego kubka dla S , tak jak przy jednoprzebiegowym złączeniu przez haszowanie. Do usprawnienia złączenia niezbędne jest uporządkowanie każdego z dostępnych w pamięci kubków z S w wydajną strukturę wspomagającą wyszukiwanie, tak jak przy złączeniu jednoprzebiegowym. Jeśli t trafia do jednego z kubków, dla których powiązany kubek z S znajduje się na dysku, t należy zapisać w bloku kubka w pamięci głównej, a następnie na dysku, tak jak dla dwuprzebiegowego złączenia opartego na haszowaniu.

W drugim przebiegu powiązane kubki z R i S można złączyć w standardowy sposób. Nie trzeba jednak łączyć par kubków, dla których kubek z S znajdował się w pamięci. Algorytm już złączył te kubki i zwrócił wyniki.

Oszczędność dyskowych operacji wejścia-wyjścia to dwa na każdy pozostawiony w pamięci blok kubków z S i powiązanych kubków z R . Ponieważ w pamięci znajduje się m/k kubków, oszczędności wynoszą $2(m/k)(B(R) + B(S))$. Trzeba więc ustalić, jak zmaksymalizować wartość m/k z uwzględnieniem ograniczeń z równania (15.1). Zaskakująca odpowiedź jest taka: należy wybrać $m = 1$, a następnie możliwie najmniejsze k .

Oto intuicyjne uzasadnienie — wszystkie oprócz $k - m$ buforów z pamięci głównej można wykorzystać do przechowywania krotek z S w tej pamięci. Im więcej jest tych krotek, tym mniej trzeba dyskowych operacji wejścia-wyjścia. Dlatego warto zminimalizować k , czyli łączną liczbę kubków. W tym celu należy utworzyć kubki tak duże, że ledwo mieszczą się w pamięci głównej. Kubki mają mieć rozmiar M , a tym samym $k = B(S)/M$. Wtedy w dodatkowej pamięci głównej znajduje się miejsce na tylko jeden kubek ($m = 1$).

W praktyce kubki muszą być nieco mniejsze niż M . W przeciwnym razie zabraknie miejsca na jeden kompletny kubek i jeden blok dla pozostałych $k - 1$ kubków przechowywanych w danym czasie w pamięci. Jeśli dla uproszczenia założymy, że k to około $B(S)/M$, a $m = 1$, wtedy liczba dyskowych operacji wejścia-wyjścia będzie mniejsza o:

$$2M(B(R) + B(S))/B(S)$$

Łączny koszt wynosi $(3 - 2M/B(S))(B(R) + B(S))$.

Przykład 15.9. Rozważmy problem z przykładu 15.4. Trzeba złączyć relacje R i S o 1000 bloków oraz 500 blokach, używając $M = 101$. Przy stosowaniu hybrydowego złączenia przez haszowanie k (liczba kubków) powinna wynosić około $500/101$. Założmy, że $k = 5$. Wtedy kubek ma średnio 100 bloków z krotkami z S . Przy próbie umieszczenia jednego z takich kubków i czterech dodatkowych bloków na cztery pozostałe kubki potrzeba 104 bloków pamięci głównej, a nie można ryzykować, że przechowywany w pamięci kubek się w niej nie zmieści.

Dlatego lepiej wybrać $k = 6$. Teraz przy haszowaniu S w pierwszym przebiegu dostępnych jest pięć buforów na pięć kubków i do 96 buforów na kubek przechowywany w pamięci, którego oczekiwany rozmiar to $500/6$, czyli 83. Liczba dyskowych operacji wejścia-wyjścia używanych dla S w pierwszym przebiegu to 500 na wczytanie całej S i $500 - 83 = 417$ na zapis pięciu kubków na dysku. Przy przetwarzaniu R w pierwszym przebiegu trzeba wczytać całą R (1000 dyskowych operacji wejścia-wyjścia) i zapisać 5 z 6 kubków (833 operacje).

W drugim przebiegu wczytywane są wszystkie kubki zapisane na dysku, co daje $417 + 833 = 1250$ dodatkowych operacji. Łączna liczba dyskowych operacji wejścia-wyjścia wynosi więc 1500 na wczytanie R i S , 1250 na wczytanie 5 z 6 relacji i kolejne 1250 na ponowne wczytanie krotek, co daje 4000 operacji. Można porównać tę liczbę z 4500 dyskowych operacji wejścia-wyjścia potrzebnych przy prostym złączeniu przez haszowanie lub złączeniu przez sortowanie. ■

15.5.7. Przegląd algorytmów opartych na haszowaniu

W tabeli 15.13 przedstawiono wymogi dotyczące pamięci i dyskowych operacji wejścia-wyjścia dla każdego z algorytmów omówionych w podrozdziale. Podobnie jak w innych rodzajach algorytmów, szacunki dla γ i δ są ostrożne, ponieważ operacje te zależą od liczby powtórzeń lub grup, a nie od liczby krotek w relacji podanej jako argument.

Operatory	Potrzebne M (w przybliżeniu)	Dyskowe operacje wejścia-wyjścia	Punkt
γ, δ	$\sqrt{B}$	$3B$	15.5.2, 15.5.3
$\cup, \cap, -$	$\sqrt{B(S)}$	$3(B(R) + B(S))$	15.5.4
$\bowtie$	$\sqrt{B(S)}$	$3(B(R) + B(S))$	15.5.5
$\bowtie$	$\sqrt{B(S)}$	$(3 - 2M/B(S))(B(R) + B(S))$	15.5.6

15.13. Wymogi dotyczące pamięci głównej i dyskowych operacji wejścia-wyjścia w algorytmach opartych na haszowaniu; dla operacji binarnych zakładamy, że $B(S) \leq B(R)$

Zauważmy, że wymogi dla algorytmów opartych na sortowaniu i odpowiadających im algorytmów opartych na haszowaniu są niemal identyczne. Oto istotne różnice między dwoma podejściami.

- (1) Wymogi dotyczące rozmiaru w opartych na haszowaniu algorytmach dla operacji dwuarumentowych zależą tylko od mniejszego z dwóch argumentów, a nie — tak jak w algorytmach opartych na sortowaniu — od sumy ich rozmiarów.
- (2) Algorytmy oparte na sortowaniu czasem umożliwiają utworzenie wyniku w posortowanej postaci i późniejsze wykorzystanie tego faktu. Wyniku można użyć w innym opartym na sortowaniu algorytmie dla dalszego operatora lub podać go jako odpowiedź na zapytanie, jeśli dane mają być posortowane.
- (3) Algorytmy oparte na haszowaniu wymagają, aby kubelki miały równy rozmiar. Ponieważ zwykle występują przynajmniej niewielkie różnice w wielkości, nie można wykorzystać kubelków, które średnio zajmują M bloków. Trzeba ograniczyć rozmiar do nieco mniejszej wartości. Jest to szczególnie odczuwalne, jeśli liczba różnych kluczy haszujących jest niewielka (przy małej liczbie grup w czasie grupowania lub niewielkiej liczbie wartości atrybutów uwzględnianych przy złączeniu).
- (4) W algorytmach opartych na sortowaniu posortowane podlisty można zapisać w przyległych blokach dysku (jeśli dysk jest odpowiednio uporządkowany). Dlatego jedna na trzy dyskowe operacje wejścia-wyjścia na blok może wiązać się z małym opóźnieniem obrotowym lub czasem wyszukiwania i przebiegać dużo szybciej niż takie operacje w algorytmach opartych na haszowaniu.
- (5) Ponadto jeśli wartość M jest znacznie większa niż liczba posortowanych podlist, można jednocześnie wczytać kilka przyległych bloków posortowanej podlisty, co także pozwala skrócić opóźnienie i czas wyszukiwania.
- (6) Jeśli jednak w algorytmie opartym na haszowaniu wystarczy utworzyć mniej niż M kubelków, można jednocześnie zapisać kilka bloków z kubelka. Zapewnia to na etapie zapisywania przy haszowaniu te same korzyści, które uzyskujemy przy drugim odczycie w algorytmach opartych na sortowaniu, co opisano w punkcie 5. Także tu można uporządkować dysk w taki sposób, aby zapisać kubelki w kolejnych blokach ścieżki. Wtedy kubelki można wczytywać z niewielkim opóźnieniem i czasem wyszukiwania — tak jak w punkcie 4. możliwy był wydajny zapis posortowanych podlist.

15.5.8. Ćwiczenia do podrozdziału 15.5

Ćwiczenie 15.5.1. Pomysł wykorzystany w hybrydowym złączeniu przez haszowanie, zapisywanie jednego kubeczka w pamięci głównej, można zastosować także do innych operacji. Pokaż, jak zmniejszyć koszt o zapisywanie i wczytywanie jednego kubeczka z każdej relacji w opartym na haszowaniu dwuprzebiegowym algorytmie dla operacji: a) δ , b) γ , c) $\cap_H$, d) $-_Z$.

Ćwiczenie 15.5.2. Jeśli $B(S) = B(R) = 10\,000$, a $M = 1000$, ile dyskowych operacji wejścia-wyjścia trzeba na hybrydowe złączenie przez haszowanie?

Ćwiczenie 15.5.3. Napisz iteratory będące implementacją opartych na haszowaniu dwuprzebiegowych algorytmów dla operacji: a) δ , b) γ , c) $\cap_H$, d) $-_Z$, e) $\bowtie$.

! Ćwiczenie 15.5.4. Wykonywana jest oparta na haszowaniu dwuprzebiegowa operacja grupowania na relacji R o odpowiednim rozmiarze — $B(R) \leq M^2$. Istnieje jednak tak mało grup, że niektóre z nich są większe niż M (nie można zapisać ich w całości w pamięci głównej). Jakie modyfikacje, jeśli w ogóle, są potrzebne w przedstawionym algorytmie?

! Ćwiczenie 15.5.5. Załóżmy, że zastosowano dysk, w którym czas do przeniesienia głowicy nad blok wynosi 100 milisekund, a odczyt jednego bloku trwa $\frac{1}{2}$ milisekundy. Tak więc odczyt k kolejnych bloków po umiejscowieniu głowicy zajmuje $k/2$ milisekund. Wykonujemy dwuprzebiegowe złączenie przez haszowanie $R \bowtie S$, gdzie $B(R) = 1000$, $B(S) = 500$, a $M = 101$. Aby przyspieszyć złączenie, należy użyć jak najmniejszej liczby kubeczków (zakładamy, że rozkład krotek w kubeczkach jest równomierny) oraz wczytywać i zapisywać możliwie dużo bloków na kolejnych pozycjach dysku. Liczymy 100,5 milisekundy na losową dyskową operację wejścia-wyjścia i $100 + k/2$ milisekundy na odczyt (lub zapis) k kolejnych bloków z dysku (albo na dysk).

- Ile czasu zajmuje dyskowa operacja wejścia-wyjścia?
- Ile czasu zajmuje taka operacja w hybrydowym złączeniu przez haszowanie w wersji z przykładu 15.9?
- Ile czasu w tych samych warunkach zajmuje złączenie oparte na sortowaniu przy założeniu, że posortowane podlisty zapisywane są w kolejnych blokach dysku?

15.6. Algorytmy oparte na indeksach

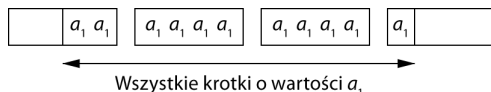
Istnienie indeksu na jednym lub kilku atrybutach relacji pozwala zastosować pewne algorytmy niedostępne bez indeksu. Algorytmy oparte na indeksach są szczególnie przydatne dla operatora selekcji, jednak można je z bardzo dobrymi efektami stosować także przy złączeniu i innych operacjach dwuargumentowych. W tym podrozdziale przedstawiono te algorytmy. Kontynuowane jest też rozpoczęte w punkcie 15.1.1 omówienie operatora skanowania indeksu, służącego do dostępu do tabel składowanych z indeksem. Aby zrozumieć wiele zagadnień, najpierw trzeba zapoznać się z dygresją i indeksami klastrującymi (ang. *clustering index*).

15.6.1. Indeksy klastrujące i nieklastrujące

W punkcie 15.1.3 stwierdzono, że relacja jest sklastrowana, jeśli krotki są upakowane w prawie minimalnej liczbie bloków potrzebnych do ich zapisania. We wszystkich dotychczasowych analizach zakładano, że relacje są sklastrowane.

Istnieją też *indeksy klastrujące*. Są to indeksy na atrybucie lub atrybutach, dzięki którym wszystkie krotki o danej wartości klucza wyszukiwania dla tego indeksu pojawiają się w prawie minimalnej liczbie bloków. Warto pamiętać, że relacja niesklastrowana nie może mieć indeksu klastrującego⁵, jednak nawet relacja sklastrowana może mieć indeksy nieklastrujące.

Przykład 15.10. Relacja $R(a, b)$ posortowana według atrybutu a i upakowana w blokach w tej kolejności jest sklastrowana. Indeks na a jest klastrujący, ponieważ dla danej wartości a , a_1 , wszystkie krotki o tej wartości znajdują się obok siebie. Dlatego występują upakowane w bloki, czasem z wyjątkiem pierwszego i ostatniego bloku zawierającego wartość a_1 atrybutu a , co pokazano na rysunku 15.14. Jednak indeks na b zwykle nie jest klastrujący, ponieważ krotki o danej wartości b są rozproszone po całym pliku (chyba że wartości a i b są bardzo ściśle skorelowane). ■



15.14. Indeks klastrujący powoduje zapisanie wszystkich krotek o danej wartości w (prawie) minimalnej możliwej liczbie bloków

15.6.2. Selekcja oparta na indeksie

W punkcie 15.1.1 omówiono implementację selekcji, $\sigma_C(R)$, przez wczytanie wszystkich krotek relacji R , sprawdzenie warunku C i zwrócenie tych krotek, które go spełniają. Jeśli nie istnieją indeksy dla R , jest to najlepsze rozwiązanie. Liczba dyskowych operacji wejścia-wyjścia wynosi $B(R)$ lub — jeśli R nie jest sklastrowana⁶ — nawet $T(R)$ (jest to liczba krotek z R). Załóżmy jednak, że warunek C ma postać $a = v$, gdzie a to atrybut, na którym oparty jest indeks, a v to wartość. Można wtedy poszukać w indeksie wartości v i otrzymać wskaźniki do krotek z R o wartości v atrybutu a . Krotki te stanowią wynik operacji $\sigma_{a=v}(R)$, dlatego wystarczy je pobrać.

Jeśli indeks na $R.a$ jest klastrujący, liczba dyskowych operacji wejścia-wyjścia przy pobieraniu zbioru $\sigma_{a=v}(R)$ wynosi średnio $B(R)/V(R, a)$. Rzeczywista liczba z kilku przyczyn może być nieco wyższa.

- (1) Często indeks nie jest w całości przechowywany w pamięci głównej, dlatego potrzebne są dyskowe operacje wejścia-wyjścia na wyszukiwanie indeksu.
- (2) Choć wszystkie krotki o $a = v$ można zmieścić w b blokach, krotki mogą być zapisane w $b + 1$ blokach, ponieważ nie zaczynają się od początku bloku.

Jeśli nawet krotki z R są sklastrowane, czasem nie są upakowane w blokach możliwie ściśle. Dostępne może być na przykład dodatkowe miejsce na krotki później wstawiane do R . Relacja R może też znajdować się w sklastrowanym pliku, co omówiono w punkcie 14.1.6.

⁵ Technicznie jeśli indeks jest oparty na kluczu relacji, istnieje tylko jedna krotka o danej wartości klucza indeksowania, dlatego indeks zawsze jest klastrujący, nawet wtedy, kiedy relacja nie jest sklastrowana. Jeżeli jednak istnieje tylko jedna krotka na wartość klucza indeksowania, klastrowanie nie daje korzyści, a wydajność indeksu jest taka sama jak indeksu nieklastrującego.

⁶ Przypominamy notację opisaną w punkcie 15.1.3: $T(R)$ oznacza liczbę krotek z R , $B(R)$ to liczba bloków, w których mieści się R , a $V(R, L)$ określa liczbę różnych krotek w $L(R)$.

Ponadto wynik trzeba zaokrąglić w górę, jeśli iloraz $B(R)/V(R, a)$ nie jest liczbą całkowitą. Najważniejsze jest to, że jeśli a to klucz R , wtedy $V(R, a) = T(R)$, co jest zwykle dużo większą wartością niż $B(R)$. Potrzebne są wtedy jedna dyskowa operacja wejścia-wyjścia na pobranie krotki o kluczu o wartości v plus inne takie operacje przy dostępie do indeksu.

Rozważmy teraz, co się dzieje, kiedy indeks na $R.a$ jest nieklastrujący. Oto pierwsze przybliżenie. Każda pobierana krotka znajduje się w innym bloku, dlatego trzeba uzyskać dostęp do $T(R)/V(R, a)$ krotek. Dlatego szacunkowo potrzeba $T(R)/V(R, a)$ dyskowych operacji wejścia-wyjścia. Wartość ta może być wyższa, jeśli trzeba wczytać z dysku bloki indeksu. Może też być niższa, jeżeli szczęśliwie niektóre pobrane krotki znajdują się w tym samym bloku, a sam blok pozostaje zbuforowany w pamięci.

Przykład 15.11. Załóżmy, że $B(R) = 1000$, a $T(R) = 20\,000$. Relacja R ma więc 20 000 krotek upakowanych maksymalnie po 20 na blok. Niech a będzie jednym z atrybutów relacji R . Przyjmijmy, że istnieje indeks na a i należy wykonać operację $\sigma_{a=0}(R)$. Oto kilka możliwych sytuacji i liczba dyskowych operacji wejścia-wyjścia potrzebnych w najgorszym przypadku. Koszt dostępu do bloków indeksu jest pomijany.

- (1) Jeśli R jest sklastrowana, natomiast nie użyto indeksu, koszt wynosi 1000 dyskowych operacji wejścia-wyjścia. Trzeba pobrać każdy blok R .
- (2) Jeżeli R nie jest sklastrowana, koszt to 20 000 dyskowych operacji wejścia-wyjścia.
- (3) Jeśli $V(R, a) = 100$, a indeks jest klastrujący, algorytm oparty na indeksie wymaga $1000/100 = 10$ dyskowych operacji wejścia-wyjścia plus operacje potrzebne na dostęp do indeksu.
- (4) Jeśli $V(R, a) = 10$, a indeks jest nieklastrujący, algorytm oparty na indeksie wymaga $20\,000/10 = 2000$ dyskowych operacji wejścia-wyjścia. Jest to koszt wyższy niż przy przeglądaniu całej relacji R , jeśli R jest sklastrowana, natomiast indeks nie istnieje.
- (5) Jeśli $V(R, a) = 20\,000$, czyli a jest kluczem, algorytm oparty na indeksie wymaga 1 dyskowej operacji wejścia-wyjścia plus operacje potrzebne na dostęp do indeksu. Nie ma znaczenia, czy indeks jest klastrujący.

Przeglądanie indeksu jako metoda dostępu jest pomocne także w kilku innych rodzajach operacji selekcji.

- a) Indeks w rodzaju drzewa zbalansowanego umożliwia wydajny dostęp do wartości klucza wyszukiwania z określonego przedziału. Jeśli istnieje indeks na atrybucie a relacji R , można wykorzystać go do pobrania krotek z odpowiedniego przedziału z R w operacjach selekcji podobnych do $\sigma_{a \geq 10}(R)$, a nawet $\sigma_{a \geq 10 \text{ AND } a \leq 20}(R)$.
- b) Selekcję ze złożonym warunkiem C można czasem zaimplementować jako skanowanie indeksu, po którym następuje kolejna selekcja dotycząca tylko krotek zwróconych przy przeglądaniu indeksu. Jeśli C ma postać $a = v \text{ AND } C'$, gdzie C' to dowolny warunek, można rozbić selekcję na dwie takie operacje. Pierwsza powinna sprawdzać tylko równość $a = v$, natomiast druga — warunek C' . W pierwszej można zastosować operator skanowania indeksu. Rozbicie operacji selekcji to jedno z wielu usprawnień, jakie optymalizator zapytań może wprowadzić w logicznym planie zapytania. Zagadnienie to opisano przede wszystkim w punkcie 16.7.1.

15.6.3. Złączenie za pomocą indeksu

We wszystkich rozważanych operacjach dwuargumentowych, a także w jednoargumentowych operacjach γ i δ na pełnych relacjach, można z powodzeniem korzystać z pewnych indeksów. Opracowanie większości algorytmów pozostawiamy jako ćwiczenia, natomiast tu koncentrujemy się na złączeniach. Szczególnie analizowane jest złączenie naturalne $R(X, Y) \bowtie S(Y, Z)$. Przypominamy, że X , Y i Z mogą oznaczać zbiory atrybutów, choć można też traktować je jak pojedyncze atrybuty.

W pierwszym algorytmie złączenia opartym na indeksie założmy, że w S istnieje indeks na atrybutach Y . Jednym ze sposobów na obliczenie złączenia jest pobranie każdego bloku z R i w ramach wszystkich bloków sprawdzenie każdej krotki t . Niech t_Y oznacza komponent lub komponenty t odpowiadające atrybutom Y . Za pomocą indeksu można znaleźć w S wszystkie krotki, w których t_Y to komponenty atrybutów Y . Te krotki S można złączyć z krotką t z R , dlatego należy zwrócić takie złączenia.

Liczba dyskowych operacji wejścia-wyjścia zależy od wielu czynników. Po pierwsze, jeśli R jest składowana, należy wczytać $B(R)$ bloków w celu pobrania wszystkich krotek z R . Jeżeli R nie jest składowana, potrzebnych może być do $T(R)$ dyskowych operacji wejścia-wyjścia.

Dla każdej krotki t z R trzeba wczytać średnio $T(S)/V(S, Y)$ krotek z S . Jeśli dla S istnieje nieskładowany indeks na Y , liczba dyskowych operacji wejścia-wyjścia potrzebnych do wczytania S wynosi $T(R)T(S)/V(S, Y)$. Jeżeli jednak indeks jest składowany, wystarczy $T(R)B(S)/V(S, Y)$ operacji⁷. W obu sytuacjach konieczne może być dodanie kilku dyskowych operacji wejścia-wyjścia na wartość Y , aby uwzględnić czytanie samego indeksu.

Niezależnie od tego, czy R jest składowana, najistotniejszy jest koszt dostępu do krotek z S . Po pominięciu kosztu wczytywania R należy przyjąć $T(R)T(S)/V(S, Y)$ lub $T(R)(\max(1, B(S)/V(S, Y)))$ jako koszt dla tej metody złączenia dla przypadków z nieskładowanym i składowanym indeksem dla S .

Przykład 15.12. Wykorzystajmy podstawowy przykład — relacje $R(X, Y)$ i $S(Y, Z)$ obejmujące 1000 i 500 bloków. Założmy, że w bloku mieści się po 10 krotek z każdej relacji, zatem $T(R) = 10\,000$, a $T(S) = 5000$. Przyjmijmy też, iż $V(S, Y) = 100$ (w krotkach z S istnieje 100 różnych wartości Y).

Założmy, że R jest składowana i dla S istnieje indeks klastrowy na Y . Wtedy przybliżona liczba dyskowych operacji wejścia-wyjścia, z pominięciem dostępu do samego indeksu, wynosi 1000 na odczyt bloków z R plus $10\,000 \times 500 / 100 = 50\,000$. Liczba ta jest wyraźnie wyższa niż dla opisanych wcześniej metod działających na tych samych danych. Jeśli R lub indeks dla S nie są składowane, koszt jest jeszcze wyższy. ■

Choć na podstawie przykładu 15.12 może się wydawać, że złączenie oparte na indeksie to bardzo zły pomysł, istnieją inne sytuacje, w których złączenie $R \bowtie S$ ma większy sens. Najczęściej dzieje się tak, kiedy R jest bardzo mała w porównaniu z S , a wartość $V(S, Y)$ jest duża. W ćwiczeniu 15.6.5 opisano typowe zapytanie, w którym selekcja przed złączeniem sprawia, że R jest mała. Wtedy większa część S nie jest sprawdzana przez algorytm, ponieważ większość wartości Y w ogóle nie występuje w R . Z kolei w złączeniu opartym na sortowaniu i haszowaniu każda krotka z S sprawdzana jest przynajmniej raz.

⁷ Trzeba jednak pamiętać, że jeśli $B(S)/V(S, Y)$ ma wartość poniżej 1, trzeba zastąpić ją liczbą 1, co opisano w punkcie 15.6.2.

15.6.4. Złączenia z wykorzystaniem posortowanego indeksu

Jeśli indeks jest drzewem zbalansowanym (lub inną strukturą, z której można łatwo pobrać posortowane krotki relacji), istnieje wiele innych możliwości zastosowania go. Prawdopodobnie najprostszą jest obliczanie $R(X, Y) \bowtie S(Y, Z)$, kiedy istnieje odpowiedni indeks na Y dla R lub S . Można wtedy zastosować zwykłe złączenie przez sortowanie, jednak nie trzeba wykonywać pośredniego kroku, polegającego na sortowaniu jednej z relacji według Y .

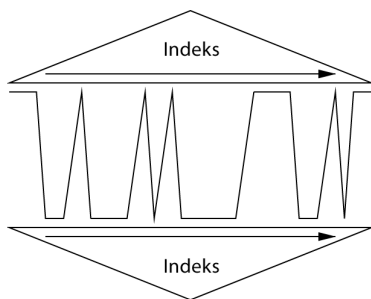
Skrajnym przypadkiem jest sytuacja, kiedy indeksy na Y istnieją zarówno dla R , jak i dla S . Wtedy trzeba wykonać tylko ostatni krok prostego złączenia opartego na sortowaniu (punkt 15.4.6). Metoda ta jest czasem nazywana *złączeniem zig-zag*, ponieważ polega na przechodzeniu między indeksami tam i z powrotem w celu znalezienia wspólnych wartości Y . Zauważmy, że nie trzeba pobierać krotek z R o wartości Y , która nie występuje w S (i na odwrot).

Przykład 15.13. Przyjmijmy, że istnieją relacje $R(X, Y)$ i $S(Y, Z)$ o indeksach na Y w obu relacjach. Niech klucze wyszukiwania (wartości Y) dla krotek z R mają wartości 1, 3, 4, 4, 4, 5, 6, a dla krotek z S — 2, 2, 4, 4, 6, 7. Zaczynamy od pierwszych kluczy z R i S , czyli 1 i 2. Ponieważ $1 < 2$, należy pominąć pierwszy klucz z R i sprawdzić drugi klucz, 3. Teraz obecny klucz z S jest mniejszy niż aktualny klucz z R , dlatego można pominąć dwie 2 z S i dojść do 4.

Na tym etapie klucz 3 z R ma wartość mniejszą niż klucz z S , dlatego można pominąć klucz z R . Teraz oba klucze mają wartość 4. Należy podążyć za wskaźnikami powiązanymi z wszystkimi kluczami o wartości 4 z obu relacji, pobrać odpowiednie krotki i złączyć je. Zauważmy, że do czasu natrafienia na wspólny klucz, 4, nie pobrano z relacji żadnych krotek.

Po wyczerpaniu wartości 4 należy przejść do klucza 5 w R i 6 w S . Ponieważ $5 < 6$, przechodzimy do następnego klucza w R . Teraz oba klucze mają wartość 6, dlatego należy pobrać powiązane krotki i je złączyć. Ponieważ pobrano całą R , wiadomo, że w relacjach nie ma więcej par krotek, które można by złączyć. ■

Jeśli indeksy to drzewa zbalansowane, można sprawdzić liście obu drzew w kolejności od lewej do prawej, używając wbudowanych w strukturę wskaźników do liści, co pokazano na rysunku 15.15. Jeśli R i S są sklastrowane, pobranie wszystkich krotek o danym kluczu wymaga bardzo małej liczby dyskowych operacji wejścia-wyjścia. Zauważmy, że w skrajnych sytuacjach, kiedy istnieje tyle krotek z R i S , iż nie mieszczą się one w pamięci głównej, trzeba zastosować poprawkę, taką jak omówiona w punkcie 15.4.6. Jednak w typowych przypadkach etap złączenia wszystkich krotek o identycznej wartości Y można wykonać za pomocą tylu dyskowych operacji wejścia-wyjścia, ile potrzeba na wczytanie krotek.



15.15. Złączenie zig-zag z wykorzystaniem dwóch indeksów

Przykład 15.14. Kontynuujemy przykład 15.12, aby zobaczyć, jaką wydajność dla przykładowych danych ma złączenie z wykorzystaniem sortowania i indeksów. Po pierwsze, założmy, że dla S istnieje indeks na Y umożliwiający pobranie krotek z S posortowanych według Y . W przykładzie przyjmujemy też, że obie relacje i indeks są sklastrowane. Na razie założmy, że nie ma indeksu dla R .

Jeśli dostępnych jest 101 bloków pamięci głównej, można wykorzystać je do utworzenia 10 posortowanych podlist dla 1000-blokowej relacji R . Liczba dyskowych operacji wejścia-wyjścia przy odczycie i zapisie całej R wynosi 2000. Następnie można użyć 11 bloków pamięci — 10 na podlisty z R i 1 na blok z krotkami z S pobranymi za pomocą indeksu. Pomijamy operacje i bufory pamięci potrzebne do manipulowania indeksem. Jeśli jednak indeks to drzewo zbalansowane, koszty i tak są niskie. W drugim przebiegu należy wczytać wszystkie krotki z R i S , co zajmuje w sumie 1500 dyskowych operacji wejścia-wyjścia plus mała liczba potrzebna na jednokrotne wczytanie bloków indeksu. Dlatego łączna liczba dyskowych operacji wejścia-wyjścia to 3500, czyli mniej niż w omawianych wcześniej metodach.

Przyjmijmy teraz, że w R i S istnieją indeksy na Y . Nie trzeba wtedy sortować żadnej relacji. Wystarczy 1500 dyskowych operacji wejścia-wyjścia na wczytanie bloków z R i S za pomocą indeksów. W praktyce ustalenie na podstawie samych indeksów, że dużej części krotek z R lub S nie można złączyć z krotkami drugiej relacji, pozwala zmniejszyć łączny koszt znacznie poniżej 1500 dyskowych operacji wejścia-wyjścia. Jednak zawsze należy dodać małą liczbę operacji potrzebnych na wczytanie samych indeksów. ■

15.6.5. Ćwiczenia do podrozdziału 15.6

Ćwiczenie 15.6.1. Załóżmy, że istnieje indeks na atrybucie $R.a$. Opisz, jak można wykorzystać ten indeks, aby usprawnić wykonywanie poniższych operacji. W jakich warunkach algorytm oparty na indeksie jest wydajniejszy od algorytmów opartych na sortowaniu lub haszowaniu?

- $R \cup_z S$ (przyjmij, że R i S nie zawierają powtórzeń, choć w obu może znajdować się identyczna krotka).
- $R \cap_z S$ (także tu R i S to zbiory).
- $\delta(R)$.

Ćwiczenie 15.6.2. Załóżmy, że $B(R) = 10\,000$, $T(R) = 500\,000$, istnieje indeks na $R.a$, a $V(R, a) = k$ dla pewnego k . Określ koszt operacji $\sigma_{a=0}(R)$ jako funkcję od k w podanych dalej warunkach. Możesz pominąć dyskowe operacje wejścia-wyjścia potrzebne do uzyskania dostępu do samego indeksu.

- Indeks jest klastrowy.
- Indeks nie jest klastrowy.
- R jest sklastrowane, natomiast indeks nie jest używany.

Ćwiczenie 15.6.3. Wykonaj ćwiczenie 15.6.2 dla zapytania zakresowego — $\sigma_{C \leq a \text{ AND } a \leq D}(R)$. Można przyjąć, że C i D to stałe, takie że w przedziale znajduje się $k/10$ wartości.

! **Ćwiczenie 15.6.4.** Jeśli R jest sklastrowana, natomiast indeks na $R.a$ nie jest klastrowy, w zależności od k korzystniejsze może być zaimplementowanie zapytania za pomocą skanowania tabeli R lub indeksu. Dla jakich wartości k lepszy będzie indeks, jeśli relacja i zapytanie są takie jak w: a) ćwiczeniu 15.6.2, b) ćwiczeniu 15.6.3.

Ćwiczenie 15.6.5. Rozważmy zapytanie SQL:

```
SELECT dataUrodzenia FROM GwiazdyWFilmie, GwiazdaFilmowa
WHERE tytulFilmu = 'King Kong' AND nazwiskoGwiazdy = nazwisko;
```

W zapytaniu wykorzystano „filmowe” relacje:

```
GwiazdyWFilmie(tytulFilmu, rokFilmu, nazwiskoGwiazdy)
GwiazdaFilmowa(nazwisko, adres, plec, dataUrodzenia)
```

Po przekształceniu zapytania na algebrę relacji kluczowe będzie złączenie równościowe pomiędzy:

$$\sigma_{\text{tytulFilmu} = \text{'King Kong'}}(\text{GwiazdyWFilmie})$$

i relacją GwiazdaFilmowa, które można zaimplementować jako złączenie naturalne $R \bowtie S$. Ponieważ istnieją tylko trzy filmy o tytule *King Kong*, wartość $T(R)$ jest bardzo mała. Załóżmy, że dla S (relacja GwiazdaFilmowa) dostępny jest indeks na atrybucie nazwisko. Porównaj dla operacji $R \bowtie S$ koszt złączenia opartego na indeksie z kosztem złączenia opartego na sortowaniu lub haszowaniu.

! Ćwiczenie 15.6.6. W przykładzie 15.14 określono liczbę dyskowych operacji wejścia-wyjścia dla złączenia $R \bowtie S$, w którym dla R i (lub) S istnieją indeksy sortowania na atrybutach uwzględnianych w złączeniu. Jednak metody opisane w owym przykładzie nie zadziałają, jeśli istnieje zbyt wiele krotek o identycznych wartościach tych atrybutów. Poniżej jakiej liczby bloków zajmowanych przez krotki o tej samej wartości w opisanych metodach nie potrzeba dodatkowych dyskowych operacji wejścia-wyjścia?

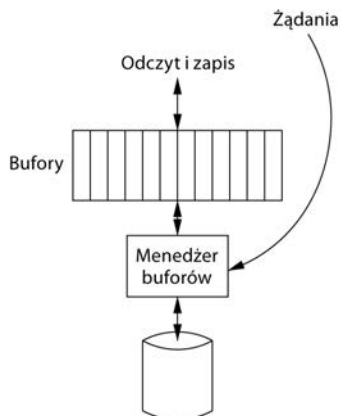
15.7. Zarządzanie buforem

Przyjęto, że dla operatorów działających na relacjach dostępna jest w pamięci głównej pewna liczba M buforów, które można wykorzystać na potrzebne dane. W praktyce buforów rzadko są z góry przydzielane operatorowi, a wartość M może się zmieniać w zależności od warunków w systemie. Za główne zadanie, czyli udostępnianie buforów pamięci głównej procesom (na przykład zapytaniom) działającym na bazie danych, odpowiada *menedżer buforów*. Menedżer odpowiada za udostępnianie procesom potrzebnej pamięci i minimalizowanie przy tym opóźnień oraz liczby żądań niemożliwych do zrealizowania. Zadania menedżera buforów przedstawiono na rysunku 15.16.

15.7.1. Architektura menedżera buforów

Istnieją dwie ogólne architektury menedżerów buforów.

- (1) Menedżer buforów bezpośrednio kontroluje pamięć główną (tak działa wiele relacyjnych systemów DBMS).
- (2) Menedżer buforów przydziela buforów w pamięci wirtualnej i umożliwia systemowi operacyjnemu określenie, które buforów znajdują się w danym momencie w pamięci głównej, a które pozostają w zarządzanej przez system przestrzeni wymiany na dysku. W ten sposób działa wiele systemów DBMS opartych na pamięci głównej i obiektowych.



15.16. Menedżer buforów odpowiada na żądania dostępu do bloków dysku w pamięci głównej

Niezależnie od podejścia zastosowanego w systemie DBMS powstaje ten sam problem: menedżer buforów powinien ograniczać liczbę używanych buforów, tak aby mieściły się w dostępnej pamięci głównej. Kiedy menedżer buforów bezpośrednio kontroluje pamięć główną, a zażądano zbyt dużej ilości pamięci, menedżer musi określić zwalniany bufor i zwrócić jego zawartość na dysk. Jeżeli blok w buforze się nie zmienił, wystarczy usunąć go z pamięci głównej, jednak po wprowadzeniu zmian blok trzeba z powrotem zapisać na jego miejscu na dysku. Jeśli menedżer buforów przedziela pamięć w pamięci wirtualnej, może przydzielić więcej buforów, niż mieści się w pamięci głównej. Jeżeli jednak rzeczywiście używane są wszystkie bufory, nastąpi przeładowanie — standardowy problem w systemie operacyjnym, kiedy to wiele bloków jest przenoszonych do i z przestrzeni wymiany dysku. W tej sytuacji system poświęca większość czasu na wymianę bloków i wykonuje bardzo niewiele użytecznych operacji.

Standardowo liczba buforów to parametr ustawiany w czasie inicjowania systemu DBMS. Można oczekiwać, że zgodnie z ustawieniami bufory będą zajmować dostępną pamięć główną niezależnie od tego, czy są alokowane w pamięci głównej czy wirtualnej. W dalszych punktach tryb buforowania nie ma znaczenia. Przyjęto, że istnieje *pula buforów* o stałym rozmiarze, czyli zestaw buforów dostępnych przy obsłudze zapytań i innych operacji na bazie danych.

Zarządzanie pamięcią przy przetwarzaniu zapytań

Zakładamy, że menedżer buforów przydziela operatorowi M buforów w pamięci głównej, przy czym wartość M zależy od warunków w systemie (w tym innych przetwarzanych operatorów i zapytań) oraz może zmieniać się dynamicznie. Kiedy operator ma dostęp do M buforów, może używać niektórych z nich do wczytywania stron dysku, innych na strony indeksu, a jeszcze innych — na etapy sortowania lub tablice haszujące. W niektórych systemach DBMS pamięć nie jest przydzielana z jednej puli, ale raczej z odrębnych pul pamięci (z różnymi menedżerami buforów) o odmiennym przeznaczeniu. Operator może otrzymać D buforów z puli na strony wczytane z dysku i H buforów na tablicę haszującą. To podejście zapewnia większe możliwości w zakresie konfigurowania i dostrajania systemu, jednak nie zawsze prowadzi do optymalnego globalnego wykorzystania pamięci.

15.7.2. Strategie zarządzania buforami

Krytyczny wybór dokonywany w menedżerze buforów dotyczy tego, który blok należy usunąć z puli buforów, kiedy potrzebny jest bufor na właśnie zażądany blok. Popularne *strategie wymiany buforów* mogą być znane z zastosowań reguł szeregowania w innym kontekście (na przykład w systemach operacyjnych). Oto niektóre strategie.

Algorytm LRU

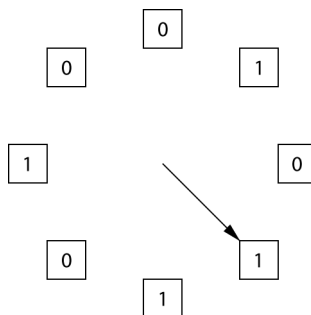
Reguła LRU (ang. *least-recently used*, czyli najdłużej nieużywany) polega na usuwaniu bloku, który nie był wczytywany ani zapisywany przez najdłuższy czas. Metoda ta wymaga, aby menedżer buforów przechowywał tabelę z czasem ostatniego dostępu do bloku w każdym buforze. Przy każdym dostępie do bazy danych trzeba też dodać wpis do tabeli, dlatego przechowywanie informacji jest pracochłonne. Jednak LRU to efektywna strategia. Intuicyjnie można stwierdzić, że długo nieużywane bufory prawdopodobnie będą potrzebne później od tych używanych niedawno.

Pierwszy na wejściu, pierwszy na wyjściu (FIFO)

Kiedy potrzebny jest bufor, a zastosowano strategię FIFO, bufor najdłużej zajmowany przez ten sam blok jest opróżniany, a następnie wykorzystywany na nowy blok. W tym podejściu menedżer buforów musi znać tylko czas wczytania do bufora obecnie zajmującego go bloku. Wpis w tabeli można więc dodać w czasie wczytywania bloku z dysku. Nie trzeba modyfikować tabeli przy dostępie do bloku. Strategia FIFO wymaga mniej pracy niż LRU, ale powoduje więcej błędów. Często używany blok, na przykład blok z korzeniem indeksu w postaci drzewa zbalansowanego, ostatecznie staje się najstarszym blokiem w buforze. Jest wtedy zapisywany z powrotem na dysku, po czym wkrótce trzeba czytać go do innego bufora.

Algorytm „zegarowy” (drugiej szansy)

Algorytm ten to często stosowane, wydajne przybliżenie strategii LRU. Można wyobrazić sobie, że bufory uporządkowane są w okrąg, co pokazano na rysunku 15.17. „Wskazówka” pokazuje jeden z buforów i porusza się zgodnie z ruchem wskazówek zegara, kiedy trzeba znaleźć bufor, w którym należy umieścić blok z dysku. Każdy bufor jest powiązany z „flagą” o wartości 0 lub 1. Bufory z wartością 0 są narażone na przesłanie zawartości z powrotem na dysk; bufory z wartością 1 są na to odporne. Przy wczytywaniu bloku do bufora flagę należy ustawić na 1. Także dostęp do zawartości bufora powoduje ustawienie flagi na 1.



15.17. Algorytm „zegarowy” przechodzi po kolejnych buforach i zastępuje flagi 01...1 flagami 10...0

Inne sztuczki dotyczące algorytmu „zegarowego”

Algorytm „zegarowy” wyboru zwalnianych buforów nie ogranicza się do schematu opisanego w punkcie 15.7.2, gdzie flagi przyjmowały wartości 0 i 1. Do ważnej strony można początkowo przypisać flagę o wartości większej niż 1 i zmniejszać ją o 1 przy każdym przejściu wskazówki. W praktyce można zastosować przyklejanie bloków, przypisując do bloku flagę o nieskończonej wartości. W odpowiednim momencie system zwalnia blok, ustawiając flagę na 0.

Kiedy menedżer buforów potrzebuje bufor na nowy blok, szuka pierwszej wartości 0, poruszając się zgodnie z ruchem wskazówek zegara. Jeśli natrafia na flagę 1, ustawia ją na 0. Dlatego blok jest usuwany z bufora tylko wtedy, jeśli nie używano go przez czas potrzebny wskazówce na wykonanie pełnego obrotu i ustawienie flagi na 0, a następnie wykonanie kolejnego pełnego obrotu oraz znalezienie bufora z niezmodyfikowaną flagą 0. Przykładowo na rysunku 15.17 wskazówka spowoduje ustawienie flagi 1 na 0 w następnym buforze, a następnie przejdzie do bufora z flagą 0, zastąpi jego blok i ustawi flagę na 1.

Kontrola ze strony systemu

Procesor zapytań lub inne komponenty systemu DBMS mogą dawać menedżerowi buforów wskazówki pozwalające uniknąć pewnych błędów, które mogą wystąpić przy ścisłym przestrzeganiu algorytmów LRU, FIFO lub „zegarowego”. W punkcie 13.6.5 wspomniano, że czasem z przyczyn technicznych *nie można przenieść* bloku z pamięci głównej na dysk bez wcześniejszego zmodyfikowania pewnych innych prowadzących do niego bloków. Bloki te są przyklejone, a menedżer buforów musi zmodyfikować strategię wymiany buforów, aby uniknąć usunięcia takich bloków. Daje to możliwość wymuszenia pozostawienia pewnych bloków w pamięci głównej przez oznaczenie ich jako przyklejonych, nawet jeśli nie ma technicznych przyczyn, dla których nie można zapisać ich na dysku. Przykładowo rozwiązanie problemu z algorytmem FIFO (związanego z korzeniem drzewa zbalansowanego) polega na przyklejeniu korzenia, co wymusza stałe utrzymywanie go w pamięci. Podobnie w algorytmach w rodzaju jednoprzebiegowego złączenia przez haszowanie procesor zapytań może przykleić bloki mniejszej relacji, aby zagwarantować, że na stałe pozostaną w pamięci głównej.

15.7.3. Związki między fizycznym operatorem selekcji a zarządzaniem buforami

Optymalizator zapytań ostatecznie wybiera zbiór operatorów fizycznych używanych do wykonania danego zapytania. Wybór może być oparty na założeniu, że przy przetwarzaniu każdego operatora dostępnych jest M buforów. Jednak, jak pokazano, w menedżerze buforów nie zawsze pożądane lub możliwe jest zagwarantowanie dostępności M buforów na czas wykonywania zapytania. Dlatego trzeba zadać dwa powiązane pytania na temat operatorów fizycznych.

- (1) Czy algorytm potrafi dostosować się do zmian w wartości M (liczbie buforów dostępnych w pamięci głównej)?
- (2) Jeśli oczekiwana liczba M buforów nie jest dostępna, a niektóre bloki, które powinny znajdować się w pamięci, menedżer buforów przeniósł na dysk, w jaki sposób strategia wymiany buforów stosowana przez menedżera wpływa na liczbę dodatkowych potrzebnych operacji wejścia-wyjścia?

Przykład 15.15. Jako ilustracja tych problemów posłuży oparte na blokach złączenie zagnieżdżone z rysunku 15.8. Podstawowy algorytm nie zależy od wartości M , choć wpływa ona na jego wydajność. Dlatego M można ustalić tuż przed rozpoczęciem działania algorytmu.

Możliwe jest nawet to, że M zmieni się między iteracjami pętli zewnętrznej. Przy każdym wczytywaniu do pamięci głównej części relacji S (relacji z pętli zewnętrznej) można wykorzystać wszystkie dostępne w tym czasie bufory oprócz jednego. Ostatni bufor zarezerwowany jest na blok z R (relacji z pętli wewnętrznej). Tak więc liczba powtórzeń pętli zewnętrznej zależy od średniej liczby buforów dostępnych w każdej iteracji. Jednak o ile *średnio* dostępnych jest M buforów, analizy kosztów z punktu 15.3.4 są poprawne. W skrajnym przypadku można mieć szczęście i odkryć, że w pierwszej iteracji dostępne są bufory na zapisanie całej S . Wtedy złączenie zagnieżdżone działa tak jak złączenie jednoprzebiegowe, opisane w punkcie 15.2.3.

Oto inny przykład zależności buforowania i złączenia zagnieżdżonego. Załóżmy, że do wymiany buforów zastosowano strategię LRU, a na bloki z R dostępnych jest k buforów. Przy wczytywaniu kolejno każdego bloku z R pod koniec iteracji pętli zewnętrznej w buforach będzie znajdować się k ostatnich bloków z R . Następnie można wczytać do $M - 1$ buforów przeznaczonych na S nowe bloki z S i ponownie rozpocząć wczytywanie bloków z R (w następnej iteracji w pętli zewnętrznej). Jeśli jednak wczytywanie ponownie zacznie się od początku, trzeba wymienić k buforów przeznaczonych dla R . Samo $k > 1$ nie zapewnia wtedy zmniejszenia liczby dyskowych operacji wejścia-wyjścia.

Lepsza implementacja złączenia zagnieżdżonego (przy stosowaniu strategii LRU do wymiany buforów) polega na pobieraniu bloków z R w naprzemiennej kolejności: od początku do końca i od końca do początku (tak zwane *odbijanie*). W ten sposób przy k buforach dostępnych dla R w każdej (oprócz pierwszej) iteracji pętli zewnętrznej można zaoszczędzić k dyskowych operacji wejścia-wyjścia. Oznacza to, że druga i kolejne iteracje wymagają tylko $B(R) - k$ operacji dla R . Zauważmy, że nawet dla $k = 1$ (czyli bez *dotatkowych* buforów dostępnych dla R) oszczędność wynosi jedną operację na iterację. ■

Zmienna wartość M i strategia wymiany buforów używana w menedżerze buforów mają znaczenie także w innych algorytmach. Oto kilka przydatnych spostrzeżeń.

- Algorytm oparty na sortowaniu można dostosować do zmiany wartości M . Jeśli M maleje, można zmienić rozmiar podlist, ponieważ opisane algorytmy oparte na sortowaniu nie wymagają, aby podlisty miały stałą wielkość. Głównym ograniczeniem przy zmniejszeniu M jest to, że czasem trzeba utworzyć tyle podlist, że nie można przydzielić bufora na każdą z nich w procesie scalania.
- W algorytmach opartych na haszowaniu po zmniejszeniu M można ograniczyć liczbę kubeków, o ile nie staną się one tak duże, że przestaną mieścić się w przydzielonej pamięci głównej. Jednak — inaczej niż w algorytmach opartych na sortowaniu — nie można reagować na zmiany w M w czasie działania algorytmu. Po określeniu liczba kubeków pozostaje stała w czasie pierwszego przebiegu. Jeśli bufory staną się niedostępne, bloki należące do niektórych kubeków trzeba będzie usunąć na dysk.

15.7.4. Ćwiczenia do podrozdziału 15.7

Ćwiczenie 15.7.1. Załóżmy, że chcemy przeprowadzić złączenie $R \bowtie S$, a dostępna pamięć waha się między M i $M/2$. Za pomocą M , $B(R)$ i $B(S)$ podaj warunki, które gwarantują możliwość wykonania poniższych algorytmów:

- a) jednoprzebiegowe złączenia,
- b) dwuprzebiegowe złączenia opartego na haszowaniu,
- c) dwuprzebiegowe złączenia opartego na sortowaniu.

! Ćwiczenie 15.7.2. Jak zmniejsza się liczba dyskowych operacji wejścia-wyjścia w złączeniu zagnieżdżonym po udostępnieniu dodatkowych buforów, jeśli strategią wymiany buforów jest:

- a) pierwszy na wejściu, pierwszy na wyjściu,
- b) algorytm „zegarowy”.

!! Ćwiczenie 15.7.3. W przykładzie 15.15 zasugerowano, że w czasie złączenia można wykorzystać dodatkowe dostępne bufor, przechowując w nich więcej niż jeden blok z R i przeglądając bloki z R w odwrotnej kolejności w parzystych iteracjach pętli zewnętrznej. Można też jednak udostępnić tylko jeden bufor na R i zwiększyć liczbę buforów dostępnych dla S . Która strategia prowadzi do najmniejszej liczby dyskowych operacji wejścia-wyjścia?

15.8. Algorytmy o więcej niż dwóch przebiegach

Choć dwa przebiegi wystarczają przy operacjach na wszystkich relacjach, oprócz tych największych, należy wiedzieć, że podstawowe techniki omówione w podrozdziałach 15.4 i 15.5 można uogólnić na algorytmy, które — w odpowiedniej liczbie przebiegów — przetwarzają relacje o dowolnym rozmiarze. W tym podrozdziale omówiono uogólnienie podejść opartych na sortowaniu i haszowaniu.

15.8.1. Wieloprzebiegowe algorytmy oparte na sortowaniu

W punkcie 15.4.1 napomknięto, że DWSPS można rozwinąć w algorytm trzyprzebiegowy. Istnieje prosta rekurencyjna technika sortowania, które umożliwia kompletne sortowanie dowolnie dużych relacji lub tworzenie n posortowanych podlist dla dowolnego n .

Założmy, że do sortowania relacji R można wykorzystać M buforów w pamięci głównej (przyjmujemy, że R jest sklastrowana). Oto potrzebne operacje.

PRZYPADK PODSTAWOWY: Jeśli R mieści się w M blokach ($B(R) \leq M$), należy wczytać R do pamięci głównej, posortować za pomocą dowolnego algorytmu sortującego w pamięci głównej i zapisać posortowaną relację na dysku.

INDUKCJA: Jeżeli R nie mieści się w pamięci głównej, trzeba podzielić bloki z R na M grup — $R_1, R_2, \dots, R_M$. Należy rekurencyjnie posortować R_i dla $i = 1, 2, \dots, M$, a następnie scalić M posortowanych podlist, tak jak w punkcie 15.4.1.

Jeśli ma miejsce nie tylko sortowanie R , ale też wykonywanie na R operacji jednoargumentowej, na przykład γ lub δ , należy zmodyfikować powyższy algorytm, tak aby przy ostatecznym scalaniu wykonać działania na krotkach z początków posortowanych podlist. Oto te działania.

- Dla δ należy zwrócić jedną kopię każdej specyficznej krotki i pominąć jej pozostałe wystąpienia.
- Dla γ trzeba posortować dane tylko według atrybutów grupujących i w odpowiedni sposób połączyć krotki o danej wartości tych atrybutów, tak jak opisano to w punkcie 15.4.3.

Przy wykonywaniu operacji dwuargumentowej, takiej jak obliczanie części wspólnej lub złączenia, stosowany jest ten sam pomysł, przy czym dwie pierwsze relacje należy najpierw podzielić na łącznie M podlist. Następnie każdą podlistę trzeba posortować za pomocą opisanego algorytmu rekurencyjnego. Na zakończenie każda z M podlist wczytywana jest do jednego bufora i można wykonać operację w sposób omówiony w odpowiednich punktach podrozdziału 15.4.

M buforów można podzielić między relacje R i S w dowolny sposób. Aby jednak zminimalizować łączną liczbę przebiegów, zwykle należy podzielić buforów proporcjonalnie do liczby bloków zajmowanych przez relacje. Na R należy przeznaczyć $M \times B(R)/(B(R) + B(S))$ buforów, a na S — resztę.

15.8.2. Wydajność wieloprzebiegowych algorytmów opartych na sortowaniu

Rozważmy teraz związek między liczbą potrzebnych dyskowych operacji wejścia-wyjścia, rozmiarem relacji i pojemnością pamięci głównej. Niech $s(M, k)$ będzie maksymalnym rozmiarem relacji, którą można posortować za pomocą M buforów i k przebiegów. Wtedy można obliczyć $s(M, k)$ w następujący sposób.

PRZYPADK PODSTAWOWY: Jeśli $k = 1$ (dopuszczalny jest jeden przebieg), konieczne jest, aby $B(R) \leq M$. Ujmijmy to inaczej — $s(M, 1) = M$.

INDUKCJA: Przyjmijmy, że $k > 1$. Wtedy można podzielić R na M części, z których każdą można posortować w $k - 1$ przebiegach. Jeśli $B(R) = s(M, k)$, to $s(M, k)/M$ (czyli rozmiar każdej z M części R) nie może przekraczać $s(M, k - 1)$. Tak więc $s(M, k) = M s(M, k - 1)$.

Po rozwinięciu rekurencji okazuje się, że:

$$s(M, k) = M s(M, k - 1) = M^2 s(M, k - 2) = \dots = M^{k-1} s(M, 1)$$

Ponieważ $s(M, 1) = M$, okazuje się, że $s(M, k) = M^k$. W k przebiegach można posortować relację R , jeśli $B(R) \leq M^k$. Ujmijmy to inaczej — przy sortowaniu relacji R w k przebiegach minimalną liczbą buforów jest $M = (B(R))^{1/k}$.

Algorytm sortujący w każdym przebiegu wczytuje wszystkie dane z dysku i ponownie je zapisuje. Dlatego k -przebiegowy algorytm sortujący wymaga $2kB(R)$ dyskowych operacji wejścia-wyjścia.

Rozważmy teraz koszt wieloprzebiegowego złączenia $R(X, Y) \bowtie S(Y, Z)$ (jest to reprezentatywna operacja dwuargumentowa na relacjach). Niech $j(M, k)$ to największa liczba bloków, taka że w k przebiegach z wykorzystaniem M buforów można złączyć relacje mające w sumie $j(M, k)$ lub mniej bloków. Oznacza to, że złączenie jest możliwe, jeśli $B(R) + B(S) \leq j(M, k)$.

W ostatnim przebiegu należy scalić M posortowanych podlist z obu relacji. Każda z podlist jest sortowana za pomocą $k - 1$ przebiegów, dlatego nie może być dłuższa niż $s(M, k - 1) = M^{k-1}$, co w sumie daje $M s(M, k - 1) = M^k$. Tak więc $B(R) + B(S) \leq M^k$. Po odwróceniu ról parametrów można ponadto stwierdzić, że do obliczenia złączenia w k przebiegach potrzeba $(B(R) + B(S))^{1/k}$ buforów.

Przy określaniu liczby dyskowych operacji wejścia-wyjścia potrzebnych w algorytmach wieloprzebiegowych należy pamiętać, że — inaczej niż przy sortowaniu — dla złączeń i innych operacji na relacjach nie trzeba uwzględniać zapisu końcowego wyniku na dysk. Dlatego sortowanie podlist wymaga $2(k - 1)(B(R) + B(S))$ operacji, a kolejnych $B(R) + B(S)$ operacji potrzeba na wczytanie posortowanych podlist w ostatnim przebiegu. W sumie oznacza to $(2k - 1)(B(R) + B(S))$ operacji.

15.8.3. Wieloprzebiegowe algorytmy oparte na haszowaniu

W operacjach na dużych relacjach istnieje powiązana rekurencyjna technika haszowania. Relację (lub relacje) należy podzielić przez haszowanie na $M - 1$ kubełków, przy czym M to liczba dostępnych buforów w pamięci. Operacje jednoargumentowe należy wykonać osobno na każdym kubełku. Operacje dwuargumentowe, takie jak złączenie, trzeba przeprowadzić na każdej parze odpowiadających sobie kubełków, tak jakby były całymi relacjami. Technikę tę można opisać rekurencyjnie.

PRZYPADK PODSTAWOWY: Jeśli w operacji jednoargumentowej relacja mieści się w M buforach, należy wczytać ją do pamięci i wykonać operację. Jeżeli w operacji dwuargumentowej jedna z relacji mieści się w $M - 1$ buforach, trzeba wykonać operację, wczytując relację do pamięci głównej, a następnie wczytać drugą relację blok po bloku do M -tego bufora.

INDUKCJA: Jeżeli żadna relacja nie mieści się w pamięci głównej, należy za pomocą haszowania podzielić obie na $M - 1$ kubełków, co opisano w punkcie 15.5.1. Następnie trzeba rekurencyjnie wykonywać operację na każdym kubełku (lub każdej parze) i akumulować dane wyjściowe z wszystkich kubełków (lub par).

15.8.4. Wydajność wieloprzebiegowych algorytmów opartych na haszowaniu

Dalej w tekście obowiązuje założenie, że przy haszowaniu relacji krotki dzielone są możliwie równomiernie między kubełki. W praktyce założenie można częściowo spełnić, wybierając w pełni losową funkcję haszującą, jednak rozkład krotek w kubełkach zawsze będzie nieco nierównomierny.

Najpierw rozważmy operację jednoargumentową, na przykład γ lub δ na relacji R z wykorzystaniem M buforów. Niech $u(M, k)$ będzie liczbą bloków największej relacji, jaką może obsłużyć k -przebiegowy algorytm haszowania. Liczbę u można zdefiniować rekurencyjnie w następujący sposób.

PRZYPADK PODSTAWOWY: $u(M, 1) = M$, ponieważ relacja R musi mieścić się w M buforach (a więc $B(R) \leq M$).

INDUKCJA: Zakładamy, że pierwszy krok to podział relacji R na $M - 1$ kubełków o równej wielkości. Dlatego można obliczyć $u(M, k)$ w opisany tu sposób. Kubełki w następnym przebiegu muszą być na tyle małe, że można je obsłużyć w $k - 1$ przebiegach. Kubełki mają więc rozmiar $u(M, k - 1)$. Ponieważ R dzielona jest na $M - 1$ kubełków, konieczne jest, aby $u(M, k) = (M - 1)u(M, k - 1)$.

Po rozwinięciu rekurencji okazuje się, że $u(M, k) = M(M - 1)^{k-1}$ lub — w przybliżeniu dla dużego M — $u(M, k) = M^k$. Można też wykonać jedną z jednoargumentowych operacji na relacji R w k przebiegach przy M buforach i założeniu, że $M \geq (B(R))^{1/k}$.

Podobną analizę można przeprowadzić dla operacji dwuargumentowych. Rozważmy złączenie (tak jak w punkcie 15.8.2). Niech $j(M, k)$ będzie górnym ograniczeniem rozmiaru mniejszej z dwóch relacji R i S ze złączenia $R(X, Y) \bowtie S(Y, Z)$. Tak jak wcześniej M to liczba dostępnych buforów, a k to liczba przebiegów.

PRZYPADK PODSTAWOWY: $j(M, 1) = M - 1$. Oznacza to, że przy stosowaniu jedno-przebiegowego algorytmu złączenia R lub S musi mieścić się w $M - 1$ blokach, co opisano w punkcie 15.2.3.

INDUKCJA: $j(M, k) = (M - 1)j(M, k - 1)$. Oznacza to, że w pierwszym z k przebiegów można podzielić każdą relację na $M - 1$ kubeków i oczekiwać, iż każdy kubek będzie stanowił $1/(M - 1)$ całej relacji. Możliwe musi być złączenie każdej pary odpowiadających sobie kubków w $M - 1$ przebiegach.

Rozwijając rekurencję dla $j(M, k)$, ustalamy, że $j(M, k) = (M - 1)^k$. Przy założeniu, że M jest duże, można w przybliżeniu określić, iż $j(M, k) = M^k$. Oznacza to, że złączenie $R(X, Y) \bowtie S(Y, Z)$ można przeprowadzić w k przebiegach za pomocą M buforów, jeśli $(B(R) + B(S)) \leq M^k$.

15.8.5. Ćwiczenia do podrozdziału 15.8

Ćwiczenie 15.8.1. Załóżmy, że $B(R) = 20\,000$, $B(S) = 50\,000$ i $M = 101$. Opisz działanie poniższych algorytmów przy obliczaniu $R \bowtie S$.

- Trójpzebiegowy algorytm oparty na sortowaniu.
- Trójpzebiegowy algorytm oparty na haszowaniu.

! Ćwiczenie 15.8.2. Istnieje kilka sztuczek do zwiększania wydajności algorytmów dwupzebiegowych. Określ, czy poniższe sztuczki można wykorzystać w algorytmie wielopzebiegowym, a jeśli tak, to w jaki sposób.

- Sztuczka z hybrydowego złączenia przez haszowanie z punktu 15.5.6.
- Usprawnienie algorytmu opartego na sortowaniu przez zapisywanie bloków obok siebie na dysku (punkt 15.5.7).
- Usprawnienie algorytmu opartego na haszowaniu przez zapisywanie bloków obok siebie na dysku (punkt 15.5.7).

15.9. Podsumowanie rozdziału 15.

- Przetwarzanie zapytania.** Zapytania są kompilowane (co obejmuje rozbudowane optymalizowanie), a następnie wykonywane. Poznawanie dziedziny wykonywania zapytań obejmuje naukę metod wykonywania operacji algebry relacji oraz pewnych rozszerzeń zapewniających możliwości SQL-a.
- Plany zapytań.** Zapytania są kompilowane najpierw do logicznych planów zapytań (które często przypominają wyrażenia algebry relacji), a następnie są przekształcane na fizyczny plan zapytania przez wybranie implementacji każdego operatora, określenie kolejności złączeń i dokonanie innych wyborów, co opisano w rozdziale 16.
- Skanowanie tabeli.** Dostęp do krotek relacji zapewnia kilka operatorów fizycznych. Operator skanowania tabeli wczytuje każdy blok z krotkami relacji. Przeglądanie indeksu polega na zastosowaniu indeksu do znajdowania krotek, a skanowanie z sortowaniem powoduje posortowanie krotek.
- Miary kosztów działania operatorów fizycznych.** Zwykle największy wpływ na czas ma liczba dyskowych operacji wejścia-wyjścia potrzebnych do wykonania danej operacji. W użytym tu modelu ważny jest tylko czas dyskowych operacji wejścia-wyjścia, przy czym uwzględniono wyłącznie czas i pamięć potrzebne na wczytanie argumentów, a nie na zapisanie wyniku.

- *Iteratory*. Kilka operacji związanych z wykonywaniem zapytania można wygodnie połączyć, jeśli przeprowadzi się je za pomocą operatora. Mechanizm ten obejmuje trzy metody — rozpoczynając tworzenie relacji, generującą następną krotkę relacji i kończącą proces.
- *Algorytmy jednoprzebiegowe*. O ile jeden z argumentów operatora algebry relacji mieści się w pamięci głównej, można wykonać operację przez wczytanie mniejszej relacji do pamięci i wczytywanie drugiego argumentu blok po bloku.
- *Złączenie zagnieżdżone*. Ten prosty algorytm złączenia działa nawet wtedy, kiedy żaden argument nie mieści się w pamięci głównej. Wczytuje jak największą część mniejszej relacji do pamięci i porównuje ją z całym drugim argumentem. Proces należy powtarzać do czasu wczytania kolejno wszystkich części mniejszej relacji do pamięci.
- *Algorytmy dwuprzebiegowe*. Oprócz złączenia zagnieżdżonego, większość algorytmów dla argumentów niemieszczących się w pamięci oparta jest na sortowaniu, haszowaniu lub indeksie.
- *Algorytmy oparte na sortowaniu*. Dzielą argumenty na posortowane podlisty o wielkości pamięci głównej. Podlisty te są odpowiednio scalane w pożądany wynik. Przykładowo do scalania krotek wszystkich podlist w posortowanej kolejności służy dwuetapowe wielościeżkowe sortowanie przez scalanie.
- *Algorytmy oparte na haszowaniu*. Tu do podziału argumentów na kubelki służy funkcja haszująca. Następnie operację należy zastosować dla odrębnych kubelków (w operacjach jednoargumentowych) lub dla ich par (dla operacji dwuargumentowych).
- *Haszowanie a sortowanie*. Algorytmy oparte na haszowaniu są często lepsze od opartych na sortowaniu, ponieważ tylko jeden z ich argumentów musi być „mały”. Algorytmy oparte na sortowaniu działają dobrze, jeśli istnieje inna przyczyna do przechowywania danych w posortowanej postaci.
- *Algorytmy oparte na indeksach*. Użycie indeksu to doskonały sposób na przyspieszenie selekcji, w której w warunku atrybut z indeksu porównywany jest ze stałą. Złączenia oparte na indeksie to znakomita technika, jeśli jedna z relacji jest mała, a dla drugiej istnieje indeks oparty na atrybutach używanych w złączeniu.
- *Menedżer buforów*. Dostępność bloków pamięci kontroluje menedżer buforów. Kiedy w pamięci potrzebny jest nowy bufor, menedżer stosuje jedną ze znanych strategii wymiany (na przykład LRU) do określenia, który bufor zostanie zapisany na dysku.
- *Radzenie sobie ze zmienną liczbą buforów*. Często nie można z góry przewidzieć liczby buforów w pamięci głównej dostępnych dla operacji. Wtedy wydajność algorytmu wykonywania operacji powinna łagodnie maleć wraz ze zmniejszaniem się liczby dostępnych buforów.
- *Algorytmy wieloprzebiegowe*. Algorytmy dwuprzebiegowe oparte na sortowaniu lub haszowaniu mają naturalne rekurencyjne odpowiedniki, które wymagają trzech lub więcej przebiegów i działają dla większych ilości danych.

15.10. Literatura do rozdziału 15.

[6] i [2] to dwa przeglądy optymalizacji zapytań. [8] zawiera przegląd optymalizacji zapytań rozproszonych.

W [5] opisano wczesne badania metod złączenia. W [3] znajdują się analizy, przegląd i usprawnienia zarządzania pulą buforów.

Pierwszy opis zastosowania technik opartych na sortowaniu znajduje się w [1]. Zalety algorytmów opartych na haszowaniu przedstawiono w [7] i [4]; ta ostatnia praca to źródło hybrydowego złączenia opartego na haszowaniu.

- (1) M. W. Blasgen i K. P. Eswaran, *Storage access in relational databases*, „IBM Systems J.” 16:4 (1977), s. 363 – 378.
- (2) S. Chaudhuri, *An overview of query optimization in relational systems*, „Proc. Seventeenth Annual ACM Symposium on Principles of Database Systems”, s. 34 – 43, czerwiec, 1998.
- (3) H.-T. Chou i D. J. DeWitt, *An evaluation of buffer management strategies for relational database systems*, „Intl. Conf. on Very Large Databases”, s. 127 – 141, 1985.
- (4) D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. Stonebraker i D. Wood, *Implementation techniques for main-memory database systems*, „Proc. ACM SIGMOD Intl. Conf. on Management of Data” (1984), s. 1 – 8.
- (5) L. R. Gotlieb, *Computing joins of relations*, „Proc. ACM SIGMOD Intl. Conf. on Management of Data” (1975), s. 55 – 63.
- (6) G. Graefe, *Query evaluation techniques for large databases*, „Computing Surveys” 25:2 (czerwiec, 1993), s. 73 – 170.
- (7) M. Kitsuregawa, H. Tanaka i T. Moto-oka, *Application of hash to data base machine and its architecture*, „New Generation Computing” 1:1 (1983), s. 66 – 74.
- (8) D. Kossman, *The state of the art in distributed query processing*, „Computing Surveys” 32:4 (grudzień, 2000), s. 422 – 469.

Skorowidz

3NF, *Patrz* trzecia postać normalna

4NF, *Patrz* czwarta postać normalna

A

adapter, 917, *Patrz* nakładka

adornment, *Patrz* ozdóbik

adres

fizyczny, *Patrz* pamięć adres

fizyczny

logiczny, *Patrz* pamięć adres

logiczny

agent, *Patrz* SQL agent

agglomerative, *Patrz* grupowanie

aglomeracyjne

agregacja, 174, 179, 182, 210, 213,

232, 234, 273, 276, 427, 429,

435, 492, 647, 654, 693, 714

akcja, 313, 319, 395

aksjomaty Armstronga, 97

algebra relacji, 38, 45, 60, 69, 203,

232, 234, 255, 629, 703, 741

algorytm, 722

a-priori, 969, 970, 998, 999

BFR, 994, 999

chase, 111, 113, 127, 128, 130

dwuargumentowy na całych

relacjach, 635, 638

dwuetapowego

wielościęzkowego

sortowania przez scalanie,

Patrz DWSPS

dwuprzebiegowy, 634, 644,

645, 647, 648, 654, 655, 674

FIFO, *Patrz* FIFO

jednoargumentowy

krotkowy, 634, 635

na całych relacjach, 635, 636

jednoprzebiegowy, 634, 636,

638, 644, 648, 674, 875, 969

k-średnich, 992, 993

LRU, *Patrz* LRU

łańcuchowy, 934, 935, 957

naiwny, 968, 969

okrąg z cięciwami, 900, 901,

902, 909

określania domknięcia,

Patrz domknięcie

oparty na haszowaniu, 634,

653, 654, 655, 658, 669,

672, 674

oparty na indeksach, 634, 659,

660, 662, 674

oparty na sortowaniu, 634,

645, 648, 649, 650, 652,

658, 669, 670, 671, 674

PCY, 971, 972, 973, 998

quicksort, 44

równoległy, 872, 873, 875, 908

R-Swoosh, 952, 953, 954,

958, 981

Soundex, 950

syntezy do schematów

o trzeciej postaci

normalnej, 117

trzyprzebiegowy, 670

wieloetapowy, 973, 974, 998

wieloprzebiegowy, 634, 671,

672, 674

windy, 518, 520, 555

wspinaczkowy, 722

zachłanny, 733, 935, 957

zegarowy, 667

złączenia zagnieżdżonego

krotkowego, 641

złączenia zagnieżdżonego

opartego na blokach, 643

Amazon.com, 33, 346, 929, 962

American National Standards

Institute, *Patrz* SQL ANSI

anomalie, 101, 108, 149, 195

aktualizacji, 102, 149

usuwania, 102, 103, 149

ANSI, *Patrz* SQL ANSI

architektura

bez współużytkowania, 869,

870, 871, 872, 876, 879, 908

klient-serwer, 345, 390, 537

trójwarstwowa, 345, 390, 915

archiwizacja, 776, 777, 779, 781

nieblokująca, 777, 781

zrzut pełny, 777

zrzut przyrostowy, 777, 781

arkusz stylów, 496

Armstrong, *Patrz* aksjomaty

Armstronga

array, *Patrz* tablica

asercja, 309, 310, 319, 394

asocjacja, 174, 175, 179, 181, 199

klasa, 177

reguła, 964, 998

zwrotna, 177

atak DoS, 1004

atom, 219, 235, 678

arytmetyczny, 220

relacji, 220

relacyjny, 220

atomicity, *Patrz* atomowość

atomowość, 31, 36, 37, 48, 284,

286, 294, 411, 751, 753, 768,

888, 889

atrybut, 47, 48, 49, 50, 51, 53, 55,

56, 57, 58, 59, 61, 63, 64, 65,

68, 71, 78, 79, 136, 137, 142,

151, 155, 174, 183, 185, 216,

239, 394, 411, 452, 465, 474,

476, 484, 489, 503, 682

atrybut

- dowolny, 478
 - histogram, 716, 718
 - nazwa, 70
 - o wspólnej nazwie, 252
 - podstawowy, 116
 - wejściowy, 691
 - więzy, *Patrz* więzy, więzy atrybutu
 - wyjściowy, 691
 - zależny, 430
- authorization id, *Patrz* identyfikator uwierzytelniania
- awaria, 749, 757, 776, 779, 850, *Patrz* też dysk magnetyczny awaria systemu, 750, 780

B

- bag, *Patrz* wielozbiór
- baza danych, 31, 48, 50, 390, 510
- adres, 540, 541, 542, 543, 556
 - egzemplarz, 138
 - ekstensjonalna, *Patrz* EBD
 - integracja, 347
 - integrowanie, 445, 447
 - intensjonalna, *Patrz* IBD
 - kopia, 776, 777, 781
 - modelowanie, *Patrz* modelowanie
 - modyfikowanie, 39, 279
 - programowanie, *Patrz* programowanie
 - przestrzeń adresowa, 537, 540
 - relacyjna, 33, 445
 - rozproszona, 879, 880, 908
 - schemat, 48, 49, 101, 309
 - stan, 752
 - niespójny, 752
 - spójny, 752
 - system federacyjny, 917
 - wirtualna, 917
- baza minimalna relacji, *Patrz* relacja baza minimalna
- BCNF, 103, 104, 108, 114, 195
- bezpieczeństwo danych, 32
- biblioteka
- DB, 387
 - PEAR, 387
- binary large objects, *Patrz* bitmapa, 972, 973
- skompresowana, 620, 625
- BLOB, 549, 556
- pobieranie, 550
 - przechowywanie, 550

blok

- obsługi wyjątków, 371
 - przepełnienia, *Patrz* dysk magnetyczny blok przepełnienia
- blokada, 37, 783, 795, 796, 798, 799, 800, 802, 803, 804, 807, 808, 811, 812, 813, 814, 815, 816, 817, 818, 820, 821, 822, 826, 833, 838, 840, 864, 867
- dzielona, 802, 803, 814, 840, 897
- inkrementacji, 807, 808, 847
- menedżer, 894
- miejsce obsługi, 894, 909
- na wyłączność, 802, 803, 814, 840, 848, 863, 897
- ostrzegawcza, 817, 818
- rozwijanie, 805, 806
- tablica, 37, 894, 895, 896
- z aktualizacją, 806, 807, 814, 840
- blokowanie, 833, 893, 895, 896, 909
- dostępu, 285, 287
- dwufazowe, 783, 797, 799, 801, 802, 803, 821, 822, 833, 840
- rygorystyczne, 847, 867
- z kilkoma rodzajami blokad, 802
- broadcast, *Patrz* emisja
- brudne dane, *Patrz* dane brudne
- B-tree, *Patrz* drzewo zbalansowane
- bucket, *Patrz* kubek
- bufor, 665, 753, 754, 811
- menedżer, 36, 665, 666, 667, 668, 674, 727, 751, 757, 783
- strategia wymiany, 667

C

- Call-Level Interface, *Patrz* CLI
- chain, *Patrz* algorytm łańcuchowy
- check point, *Patrz* punkt kontrolny
- check-out-check-in, *Patrz* system wypożyczania i zwracania
- chord circle, *Patrz* algorytm okrąg z cięciwami
- ciało, 220, 233
- ciąg
- elementów, 476, 478, 484, 485, 487, 488
 - jednoelementowy, 490
 - opisanych pól, 548
- CLI, 39, 375
- clustered file, *Patrz* plik klastrowy

clustering, *Patrz* grupowanie

- Codd Ted, 32
- collaborative filtering, *Patrz* filtrowanie asocjacyjne
- COMMIT, *Patrz* dziennik rekord COMMIT
- consistency, *Patrz* spójność
- cost-based enumeration, *Patrz* wyliczenie oparte na kosztach
- crawler, *Patrz* robot internetowy
- czas, 244, 245
- GMT, 245
- czwarta postać normalna, 123, 125, 131, 195

D

- dane, 36
- agregacja, *Patrz* agregacja
 - aktualizowanie, 282, 439
 - brudne, 288, 829, 844, 845, 846, 848, 849, 854, 867
 - drażnienie, 40, 961, 976, 998, 999, 1025
 - finansowe, 1021
 - hurtownia, *Patrz* hurtownia danych
 - kompresja, 551, 620, 621
 - kopia zapasowa, 880, 908
 - kostka, *Patrz* kostka danych
 - logiczna jednostka, *Patrz* pamięć jednostka logiczna
 - manipulowanie, 60
 - model, 32, 43, 44, 46
 - drzewiasty, *Patrz* dane
 - model hierarchiczny
 - fizyczny, 43
 - hierarchiczny, 32, 46
 - obiektowo-relacyjny, 46
 - obiektowy, 46
 - relacyjny, 44, 46, 47, 48, 60
 - semistrukturalny, 44, 45, 46
 - sieciowy, 46
- nadmiarowe, *Patrz* nadmiarowość
- operacja, 44
- podział, 879, 880, 908
- przywracanie, 528, 529, 531, 543
- replikacja, 880, 908
- rozproszone, 908
- samoopisowe, 446
- satelitarne, 1020
- semistrukturalne, 39, 445, 446, 448, 450, 452, 468, 471, 477
- modelowanie, 39

struktura, 34, 43
 strumień, 1019, 1021, 1025, 1033
 szyfrowanie, 551
 średni czas do utraty, *Patrz* dysk magnetyczny
 średni czas do utraty danych
 typ, 54
 usuwanie, 281
 wielowymiarowe, 429, 440, 598, 606, 608, 613
 więzy, 44, 45, 46
 wstawianie, 279
 z czujników, 1020
 data, 244
 database management system, *Patrz* DBMS
 Data-Definition Language, *Patrz* DDL
 Datalog, 203, 219, 224, 232, 234, 405, 935, 938
 operacja złożona, 231
 reguła, 220, 221, 235
 Data-Manipulation Language, *Patrz* DML
 data-stream-management system, *Patrz* DSMS
 DBMS, 31, 32, 33, 34, 35, 36, 37, 44, 204, 284, 329, 510, 516, 537, 545, 749, 774, 862
 obiektowo-relacyjny, 175
 DDL, 31, 34
 dead end, *Patrz* ślepa uliczka
 DECLARE, 355
 definicja
 DTD, *Patrz* DTD
 definicja typu dokumentu, *Patrz* DTD
 dekompozycja, *Patrz* relacja dekompozycja
 DELETE, 281
 denial of service, *Patrz* atak DoS
 diagram
 encji-relacji, *Patrz* diagram ER
 ER, 135, 137, 138, 155, 162, 166
 GRANT, *Patrz* uprawnienia diagram
 UML, 181, 199
 związków encji, 198
 dictionary, *Patrz* słownik
 distinct-values, 491
 DML, 35, 36
 Document Type Definitions, *Patrz* DTD
 dokument, 474, 484, 489, 496, 503, 567

domena, 350
 domknięcie, 93, 99, 127
 dostęp, 31, 32, 37, 422, 440, 454, 510, 513, 551
 sekwencyjny, 518
 szybkość, 507, 508, 510, 555
 drażnienie danych, *Patrz* dane drażnienie
 druga postać normalna, 116
 drzewo, 45, 46, 446, 468, 598, 744
 czwórkowe, 606, 612, 613, 625
 kd, 606, 608, 610, 611, 624
 krzaczaste, 744
 operatorów algebraicznych, 38
 protokół, 822, 823, 826
 r, 606, 613, 614, 625
 węzeł wewnętrzny, 72, 231
 wyprowadzenia, 38, 628, 678, 681, 682, 697, 746
 wyszukiwań binarnych, 329
 zbalansowane, 39, 542, 559, 571, 572, 574, 576, 596, 607, 611, 624, 629, 636, 638, 661, 663, 668, 821, 822, 840, 850
 usuwanie elementów, 579
 wstawianie elementów, 577
 wydajność, 582
 zbalansowane bloków dysku, 39
 złączeń, 725
 bieżące, 733
 krzaczaste, 726
 lewostronnie zagłębione, 726, 727, 728, 729, 733, 747
 prawostronnie zagłębione, 726, 728
 DSMS, 1019, 1020, 1033, 1034
 DTD, 39, 450, 455, 456, 469, 471
 durability, *Patrz* transakcja trwałość
 duży obiekt binarny, *Patrz* BLOB
 DWSPS, 646, 647, 650, 670
 dysjunkcyjna postać normalna, 228
 dysk magnetyczny, 508, 510, 511, 555
 adres bloku, 537
 awaria, 522, 525, 528, 529, 531, 555, 750, 876
 blok, 509, 510, 511, 513, 534, 536, 537, 538, 550, 555, 556, 752, 780
 blok przepelnienia, 553, 554
 błędy, 511, 522, 523, 524, 555
 cylinder, 511, 550, 555, 876

czas dostępu, 514, 516, 517, 518, 520, 555
 dostęp, 513, 516
 kontroler, 513, 518, 555
 lustrzany, 516, 518, 525, 526, 534, 555
 nadmiarowy, 525, 526, 528, 529, 533, 534
 opóźnienie, 514, 555
 przepustowość, 516, 518
 rekord, *Patrz* rekord sektor, 511, 513, 524, 555
 strona, 752
 ścieżka, 511, 555
 średni czas do awarii, 525
 średni czas do utraty danych, 525
 talerz, 511
 uszkodzenie nośnika, 522, 524, 555, 750
 współużytkowanie, 871
 zespół dyskowy, 511
 zespół głowicowy, 511, 512, 513
 dziedzina, 48
 dziennik, 37, 751, 756, 757, 761, 769, 776, 780, 849, 880, 890, 891, 892
 logiczny, 852
 menedżer, 751, 756, 843
 numer porządkowy, 852
 opróżnienie, 757
 punkt kontrolny, *Patrz* punkt kontrolny
 rekord, 756
 ABORT, 760
 aktualizacji, 757
 COMMIT, 756, 757, 759, 760, 766, 767, 781
 START, 756, 759
 z możliwością powtarzania, *Patrz* rejestrowanie
 z możliwością powtarzania z możliwością wycofywania, *Patrz* rejestrowanie
 z możliwością wycofywania zatwierdzanie, 756

E

EBD, 224, 235
 egzemplarz, 49, 53, 138
 ekstensja, 223
 ekstraktor, 917, 957
 element, 456, 462, 463, 474, 476, 484, 503
 zagnieżdżony, 45

emisja, 897
 encja, 136, 157, 199, 948, 955,
 958, 981
 atrybut, 136
 zbiór, 136, 137, 143, 151, 162,
 165, 170, 174
 zbiór łączący, 143
 zbiór słaby, 158, 160, 161, 166,
 168, 183, 191, 198
 entity-relationship, *Patrz* model
 związków encji
 ER, *Patrz* model związków encji
 etykieta, 446, 450, 471
 extensible hash table, *Patrz*
 tablica z haszowaniem
 rozszerzalnym
 Extensible Markup Language,
 Patrz XML
 eXtensible Modeling Language,
 Patrz XML
 Extensible Stylesheet Language
 for Transformations, *Patrz*
 XSLT
 Extensible Stylesheet Language
 Transform, *Patrz* XSLT

F

fantom, 820
 FIFO, 667, 668
 filtrowanie, 926
 asocjacyjne, 963, 976, 987
 Flickr, 33
 flush log, *Patrz* dziennik
 opróżnienie
 FLUSH LOG, 757, 759
 FLWOR, 484, 487, 494, 503
 FLWR, *Patrz* FLWOR
 funkcja
 dominacji, 955
 haszująca, 584, 585, 587, 588,
 591, 595, 598, 599, 604,
 624, 653, 872, 873, 874,
 900, 901, 909, 974
 odwzorowująca, 876, 877,
 878, 908
 określania podobieństwa, 951
 redukująca, 876, 877, 908
 scalania, 951

G

GAV, *Patrz* mediator globalny
 generator, 423
 gigabajt, 33
 global-as-view, *Patrz* mediator
 globalny

głębokość wyszukiwania, 1003,
 1032
 Google, 33, 900, 1006
 graf, 45, 46, 137, 446, 452, 471,
 477, 864, 953
 acykliczny, 791, 792, 793,
 824, 840
 hipergraf, *Patrz* hipergraf
 oczekiwania, 855, 858, 861, 868
 ogon, 793
 poprzedzania, 791, 793, 794,
 824, 840
 zależności, 406
 grupowanie, 210, 212, 213, 232,
 234, 273, 274, 276, 432, 637,
 647, 654, 693, 714, 961, 986,
 987, 992, 993
 aglomeracyjne, 986, 990
 hierarchiczne, 990, 992

H

Hamming, *Patrz* odległość
 Hamminga, kod Hamminga
 harmonogram, 783, 784, 786,
 787, 788, 791, 793, 795, 796,
 802, 803, 808, 826, 839, 840
 odtworzalny, 846, 847
 poprawny, 801, 803
 rygorystyczny, 847, 848
 szeregowalny, 786, 787, 788,
 790, 792, 801, 811, 820,
 822, 839, 840, 847
 konfliktowo, 790, 791, 792,
 793, 794, 799, 801, 840
 szeregowy, 784, 786, 793, 794,
 799, 800, 839
 haszowanie, 874, 908, *Patrz też*
 funkcja haszująca, tablica
 z haszowaniem
 heterogeniczność, 915, 916
 heurystyka, 685, 715, 722, 728, 733
 przekształcania zapytań, 719
 hierarchia, 182, 199
 hipergraf, 884, 885, 886, 888, 908
 acykliczny, 885, 886, 887, 908
 redukcja, 885
 redukcja uszu, 885, 886, 887, 908
 histogram, 746, *Patrz* atrybut
 histogram
 hurtownia danych, 34, 428, 429,
 440, 917, 919, 923, 924, 957

I

IBD, 224, 231, 233, 235
 ICAR, 952, 953, 954, 958
 identyfikator, 460, 471, 892
 obiektu, 419, 440
 identyfikator uwierzytelniania, 393
 bieżący, 396
 PUBLIC, 393, 439
 iloczyn kartezyjański, 61, 65, 230,
 251, 255
 increment lock, *Patrz* blokada
 inkrementacji
 indeks, 36, 38, 329, 331, 333, 336,
 342, 343, 387, 550, 559, 567,
 574, 584, 737, 741, 821
 bitmapowy, 617, 618, 620,
 622, 623, 625
 deklarowanie, 330
 gęsty, 560, 561, 563, 574,
 576, 623
 główny, 560
 jednowymiarowy, 595, 596, 598
 klastrujący, 660
 koszt skanowania, 631
 na atrybucie, 660
 na wielu kluczach, 606, 607, 624
 odwrócony, 560, 567, 570, 624,
 1002, 1005, 1032
 pomocniczy, 560, 563, 564,
 565, 567, 596, 622, 624
 rzadki, 560, 562, 574, 623
 skanowanie, 629, 661, 673
 struktura, 560
 wieloatrybutowy, 329, 330
 wielopoziomowy, 562, 571,
 577, 623
 wielowymiarowy, 595, 596, 598
 inkrementacja, 802, 807, 808,
 811, 818, 840
 INSERT, 279
 instrukcja
 ALTER TABLE, 308
 COMMIT, 287, 288, 751
 CREATE INDEX, 330
 CREATE TABLE, 53, 57,
 192, 296, 302
 CREATE VIEW, 321
 DELETE, 281, 294, 359
 DROP INDEX, 331
 INSERT, 279, 294
 przyszanawania uprawnień, 350
 ROLLBACK, 287, 288, 751
 SELECT, 357
 SELECT-FROM-WHERE, 247
 SET CONSTRAINT, 308

SET TRANSACTION
ISOLATION, 291
SET TRANSACTION READ
ONLY, 288
SET TRANSACTION READ
WRITE, 288
UPDATE, 282, 294, 359
integralność referencyjna, 160, 297
integrowanie informacji, 34, 40,
445, 447, 913, 914, 948, 957
interfejs
JDBC, *Patrz* JDBC
ODBC, *Patrz* ODBC
poziomu wywołań, 354,
Patrz CLI
uniwersalny, 287
uniwersalny SQL, *Patrz* SQL
interfejs uniwersalny
zapytań, *Patrz* zapytanie
interfejs uniwersalny
inverted index, *Patrz* indeks
odwrócony
isolation, *Patrz* izolowanie
iterator, 629, 630, 631, 634, 642,
674, 738
izolacja, *Patrz* transakcja izolacja
izolowanie, 31, 36, 37, 288, 784, 839

J

Java Database Interconnectivity,
Patrz JDBC
JDBC, 39, 375, 382, 384, 387, 391
język
Datalog, *Patrz* Datalog
definicji danych, *Patrz* DDL
funkcyjny, 484, 503
macierzysty, 39, 238, 353,
354, 375
manipulowania danymi, *Patrz*
DML, język zapytań
obiektowy, 915
ODL, *Patrz* ODL
PHP, *Patrz* PHP
skryptowy, 385
UML, *Patrz* UML
XML, *Patrz* XML
XML Schema, *Patrz* XML
Schema
XPath, *Patrz* XPath
XPATh, *Patrz* XPATh
XQuery, *Patrz* XQuery
XSLT, *Patrz* XSLT
zapytań, 31, 32, 496, 503, 504
CODASYL, 32
SQL, *Patrz* SQL
wysokopoziomowy, 32

K

katalog, 348, 350, 390
kategoria składniowa, 678, 680
klasa, 174, 177, 178, 181, 183,
184, 193, 467, 915
klucz, 190
nazwa, 188
rozłączna, 178
klaster, 348, 390, 986, 990, 991,
993, 994, 995, 997, 999, *Patrz*
też rekord grupowanie
klastrowanie, 955
klauzula, *Patrz* słowo kluczowe
klient, 346, 350, 351, 390
klucz, 49, 50, 51, 53, 57, 85, 87,
89, 97, 119, 155, 158, 160, 163,
166, 175, 183, 190, 387, 416,
467, 472, 538, 560, 561, 562,
585, 587, 598, 876, 877
główny, 88, 175, 178, 295, 296,
552, 560, 564, 574
haszujący, 584, 653, 655, 813
indeksu, 329
kandydujący, 89
minimalny, 88, 89
obcy, 296, 297, 299, 430,
469, 472
więzy, 79, 81, 308
wyszukiwania, 559, 560, 561,
562, 563, 565, 567, 568,
572, 574, 575, 576, 577,
582, 584, 592, 595, 600,
607, 624, 661
k-means, *Patrz* algorytm
k-średnich
kod
błędu, 288
odległość minimalna, *Patrz*
odległość minimalna kodu
osadzony SQL, *Patrz* SQL
kod osadzony
Hamminga, 529
korekcyjny, *Patrz* teoria
kodów korekcyjnych
wyjątku, 288
kodowanie języka, *Patrz* kolacja
kolacja, 350
kompilator zapytań, *Patrz*
zapytanie kompilator
komponent, 34, 48, 57, 59, 82,
380, 422
kompozycja, 174, 179, 182
kompresja, *Patrz* dane kompresja
konflikt, 789, 790, 800, 818,
819, 822

koordynator, 889, 890, 891,
893, 895
wybór, 892
kopia
przyrostowa, *Patrz*
archiwizacja rzrztu
przyrostowy
zapasowa, 32, 525, 556, 750
korzeń, 145, 169, 178, 198, 446,
450, 476, 496, 572, 577, 683,
821, 825
kostka danych, 429, 432, 440
formalna, 429, 435
koszt, 630, 631, 657, 662, 673,
706, 707, 708, 715, 719, 721,
722, 723, 729, 732, 736, 737,
739, 746, 777, 879, 894, 897,
929, 957
koszyk zakupów, 876
krotka, 35, 48, 49, 53, 56, 57, 58,
61, 64, 65, 66, 67, 68, 71, 78,
79, 82, 86, 111, 128, 138, 165,
167, 172, 194, 203, 206, 224,
234, 238, 276, 378, 389, 414,
435, 439, 473, 489, 534, 536,
539, 550, 560, 629, 632, 636,
698, 752, 780, 813, 872
fantomowa, 820
identyfikator, 411, 440
tożsamość, 418
więzy, *Patrz* więzy krotki
wisząca, 210, 216
kubetek, 565, 569, 584, 585, 587,
590, 591, 592, 593, 594, 598,
599, 600, 601, 602, 605, 624,
636, 653, 669, 872, 873, 874,
908, 971, 972, 1028, 1029,
1033
tablica, 584, 588, 589, 590,
591, 599
kursor, 357, 359, 377, 384, 391

L

LAV, *Patrz* mediator lokalny
least-recently used, *Patrz* LRU
liczba binarna, 619, 620
limit czasu, 855, 867, 896
linia siatki, 598
link spam, *Patrz* spam
odnośnikami
list, *Patrz* lista
lista, 189, 196, 210
FROM, 679, 681
SELECT, 679, 682
liść, 446, 452, 572, 576, 682, 741

local-as-view, *Patrz* mediator lokalny
 log, *Patrz* dziennik
 logika biznesowa, 347
 logika zbiorów, 224
 LRU, 667
 LSH, 981, 982, 984, 987, 999

L

łańcuch
 bitowy, 54
 znaków, 54, 55, 138, 189, 211, 243, 386, 387, 471, 489, 497, 949
 łuk, 446, 450, 471

M

macierz
 przejsć, 876, 1007, 1032
 zgodności, 802, 804, 806, 818, 840
 magistrala, 870
 map-reduce, 869, 876, 878, 908, 909
 market-basket, *Patrz* koszyk zakupów
 materializacja, 634, 735, 738, 739, 741, 744, 747
 mechanizm przekształcający, 917
 mediator, 917, 920, 921, 922, 924, 926, 927, 929, 931, 934, 938, 940, 946, 957, 958
 globalny, 940
 lokalny, 940, 941, 957, 958
 menedżer
 blokady *Patrz* blokada menedżer
 bufora, *Patrz* bufor menedżer dziennika, *Patrz* dziennik menedżer
 kontroli współbieżności, 36, 37, 38, 783, 789, 790, 796, 804, 811, 812, 827, 830, 839, 841, 843, 847
 oparty na sprawdzaniu poprawności, 834, 845
 oparty na znacznikach czasu, 828, 841, 845
 z funkcją blokowania, 798
 przywracania stanu, *Patrz* przywracanie stanu
 rejestrowania, 37, 38
 rejestrowania zdarzeń
 i przywracania stanu, 36

transakcji, *Patrz* transakcja menedżer
 zasobów, 35
 metadane, 34, 36, 38, 628
 metoda, 411, 417, 440
 createStatement, 382
 definiowanie, 417
 executeQuery, 383
 executeUpdate, 383
 getFloat, 384
 getInt, 384
 getString, 384
 modyfikatora, 423, 440
 next, 384
 obserwatora, 422, 423, 440
 prepareStatement, 382
 UDT, 417
 miara odległości, 988, 999
 middleware, *Patrz* warstwa pośrednia
 minhashing, 977, 978, 979, 984, 987, 998, 999
 model
 danych, *Patrz* dane model
 ER, *Patrz* model związków encji
 koszyka zakupów, 962, 998
 obiektowo-relacyjny, 410, 414, 440
 obiektyowy, 414, 440
 pojęciowy, 43
 przetwarzania oparty na wejściu-wyjściu, 516
 relacyjny, 410, 440, 445
 związków encji, 39, 136, 139, 143, 146, 148, 149, 155, 157, 163, 170, 172, 174, 175, 183, 187, 198, 445
 modelowanie, 38
 danych
 semistrukturalne, 39
 moduł, 352, 353, 364, 391
 modyfikacja, 35, 36, 44
 modyfikator, 423
 MOLAP, 430, 440
 monotoniczność, 407, 408, 439, 969
 Moore Gordon, 510

N

nadklasa, 144, 146, 178
 nadklucz, 88, 89, 97, 103, 116
 nadmiarowość, 101, 103, 119, 123, 125, 149, 413, 491, 504, 525, 531, 533, 714

nakładka, 448, 913, 917, 920, 921, 924, 925, 928, 957, 958
 generator, 925
 Napster, 899
 niejednolity dostępem do pamięci, *Patrz* NUMA
 niezgodność impedancji, 354, 390
 nonuniform memory access, *Patrz* NUMA
 normalizacja, 38, 950
 NULL, 57, 58, 171, 245, 246, 276, 293, 546
 NUMA, 870

O

obiekt, 414, 422, 471, 534, 752
 BLOB, *Patrz* BLOB
 Object Definition Language, *Patrz* ODL
 obraz, 111
 ODBC, 375, 391
 ODL, 39, 143, 184, 192, 199, 410, 414, 445
 odległość
 edycyjna, 949, 982, 985, 987, 990
 Hamminga, 534
 Jaccarda, 978, 989, 999
 kosinusowa, 989, 990
 Manhattan, 997, 999
 minimalna kodu, 534
 średnia, 991
 odwzorowanie
 zawierające, 944, 945, 947, 948
 wyznaczone przez metodę, 426
 okno, 1021, 1025, 1027, 1033
 okrąg z cięciami, *Patrz* algorytm okrąg z cięciami
 OLAP, 339, 427, 428, 429, 430, 440
 relacyjny, *Patrz* ROLAP
 wielowymiarowy, *Patrz* MOLAP
 On-Line Analytic Processin, *Patrz* OLAP
 Open Database Connectivity, *Patrz* ODBC
 operacja, 61, 71
 kompensująca, 868
 logiczna, 226
 łącząca krotki z dwóch relacji, 61, 65, 68
 modyfikacji, *Patrz* modyfikacja na zbiorach, 61, 62
 przemianowania, 61
 usuwająca fragmenty relacji, 61

operand, 682, 742
 atomowy, 61
 operator, 61, 210, 226, 261
 agregacji, 210, 211, 234, 273, 310
 ALL, 261, 293
 ANY, 261, 262, 293
 CUBE, 435
 EXCEPT, 272, 273
 EXISTS, 261, 293
 filtrowania, 721
 fizyczny, 629, 631, 668, 673,
 719, 721, 742
 grupowania, 210, 212, 213,
 234, 273
 IN, 261, 293
 INTERSECT, 272, 273
 kostki, *Patrz* operator CUBE
 liścia, 742
 logiczny, 228
 łączny, 704, 746
 NATURAL JOIN, 293
 OUTER JOIN, 294
 porównywania, 490, 491, 504
 projekcji, 63
 przechodni, 704, 746
 przekształcający wielozbiory
 na zbiory, 211
 rozszerzonej projekcji, 210,
 213, 214
 selekcji, 64, 242, 255, 659,
 735, 872
 skanowania, 741
 sortowania, 210, 215, 629, 742
 UNION, 272
 usuwania powtórzeń, 210, 211
 złączenia, 881
 zagnieżdżonego, 641
 zewnętrznego, 210, 216
 γ , 213, 273
 δ , 211, 213
 π , 63, 210, 214
 ρS , 70
 optymalizacja Selinger, 721,
 722, 732, 746
 oś, 477, 478, 503
 atrybutu, 477
 dziecka, 477
 kolejnego brata, 477, 478
 poprzedniego brata, 477
 potomka, 477, 503
 przodka, 477, 478
 rodzica, 477, 478, 503
 węzła kontekstowego, 477
 outerjoin, *Patrz* operator
 złączenia zewnętrznego
 ozdobnik, 930, 931, 934, 957

P

P2P, *Patrz* system wymiany
 plików
 PageRank, *Patrz* wskaźnik
 PageRank
 pamięć
 adres, 539, 540, 542, 543, 556
 fizyczny, 538, 539, 556
 logiczny, 538, 556
 ustrukturyzowany, 539,
 553
 blok przyklejony, 543, 556, 668
 bufor, 509
 dostęp niejednolity, *Patrz*
 NUMA
 drugiego stopnia, 36, 507, 508,
 509, 511, 535, 537, 555,
 584, 629, 811
 dyskowa, *Patrz* dysk
 magnetyczny
 flash, 510
 główna, 36, 37, 508, 535, 536,
 555, 584, 608, 630, 747,
 750, 811, 870, 971
 hierarchia, 507, 509, 510, 555
 jednostka logiczna, 511, 555
 jednostki, 509
 lokalna, 870
 nietrwała, 509
 podręczna, 508, 509, 555, 870
 stabilna, 522, 524, 556
 strona, 509, 510
 trwała, 509
 trzeciego stopnia, 508, 509, 555
 wiersz, 509
 wirtualna, 510
 współużytkowanie, 870
 parsowanie, 628, 678, 679, 706,
 734, 746
 parzystość, 522, 525, 526, 555, 556
 peer-to-peer, *Patrz* system
 wymiany plików
 Persistent Stored Modules,
Patrz PSM
 petabajt, 33
 pętla, 500, 504
 zagnieżdżona, 254
 PHP, 39, 375, 385, 391
 pierwsza postać normalna, 116
 PK, *Patrz* klucz główny
 plan przetwarzania zapytania,
Patrz zapytanie plan
 przetwarzania
 plik, 560
 danych, 35, 560, 565, 574, 622
 indeksu, 36, 560

klastrowy, 564
sekwencyjny, 560, 623
 siatki, 598, 599, 600, 624
 wydajność, 602
 podelement, 451, 455, 456, 471,
 489, 500
 podjęzyk
 definiowania danych, 53
 manipulowania danymi, 53
 podklasa, 144, 146, 174, 178, 181,
 190, 198, 199
 podobieństwo
 elementów, 975
 Jaccarda, 976, 977, 983, 984,
 987, 998, 999
 sygnatur, 977
 podwarstwa, 347
 integracji, 347
 podwójne buforowanie, 520
 podzadanie, 220, 229, 407
 podzapytanie, 259, 261, 264, 293,
 305, 484, 746
 skorelowane, 263, 698
 pole, 467, 469
 polityka
 kaskadowa, 297
 ustawiania wartości null, 298
 połączenie, 351
 uśpione, 351
 ponawianie kaskadowe, 845,
 846, 867
 poprawność, 783, 803, 834, 835, 838
 porcja, 876
 postać normalna
 dysjunkcyjna, 228
 Boyce'a-Codda, *Patrz* BCNF
 pośrednik, *Patrz* warstwa
 pośrednia
 potokowanie, 634, 735, 738,
 739, 744
 półkrate, 952, 955, 956
 prawo
 algebraiczne, 207
 autorskie, 899
 DeMorgana, 305
 dotyczące grupowania i
 agregacji, 693
 łączności, 685, 692, 722
 Moore'a, 510
 obejmujące eliminowanie
 powtórzeń, 693
 podziału, 687
 przechodności, 685, 692
 przemienności, 722
 przenoszenia selekcji, 688
 rozdzielności, 207

predykat, 219, 235, 406
 ekstensjonalny, 223
 intensjonalny, 223, 224
 preprocesor, 681, 682
 primary key, *Patrz* klucz główny
 probe relation, *Patrz* relacja
 dopasowywana
 problem dawnych baz danych, 448
 procedura składowana, 350
 procesor
 poleceń DDL, 34
 transakcji, *Patrz* menedżer
 transakcji
 zapytań, 38, 627, 628, 746
 produkt, *Patrz* iloczyn
 kartezjański
 program szeregujący, *Patrz*
 menedżer kontroli
 współbieżności
 programowanie, 39
 dynamiczne, 721, 722, 729, 732
 projekcja, 210, 215, 227, 240, 634,
 691, 738, 935
 rozszerzona, 707
 protokół HTTP, *Patrz* HTTP
 próg częstotliwości, 963, 967,
 969, 970, 971, 972, 974, 998
 przechodność, 97
 przekazywanie potokowe, *Patrz*
 potokowanie
 przełącznik, 870
 przepłot, 550, 556
 przestrzeń adresowa, 537, 753
 bloków dysku, 753
 pamięci wirtualnej, 753
 transakcji, 753
 przetwarzanie
 równoległe, 869, 908
 siatkowe, 898
 przyklejanie, *Patrz* pamięć blok
 przyklejony
 przypisanie, 484
 równoległe, 255
 spójne, 225
 przywracanie stanu, 751, 760,
 761, 767, 769, 772, 774, 779,
 780, 781, 891
 menedżer, 759
 PSM, 39, 353, 363, 373
 instrukcje, 365
 instrukcje rozgałęziające, 366
 pętla, 368
 pętla FOR, 370
 wyjątki, 371
 zapytanie, 367
 pułapka, 1009, 1011, 1012, 1033

punkt kontrolny, 761, 762, 768,
 781, 769
 nieblokujący, 762, 773, 777, 781

R

RAID, 525, 526, 529, 533, 534,
 556, 750
 range query, *Patrz* zapytanie
 zakresowe
 r-drzewo, *Patrz* drzewo r
 redukcja, 908
 redukcja uszu, *Patrz* hipergraf
 redukcja uszu
 Redundant Arrays of
 Independent Disks, *Patrz*
 RAID
 referencja, 411, 413, 418, 421,
 460, 469, 471
 reguła
 asocjacji, 964, 998
 ECA, *Patrz* wyzwalacz
 gramatyki języka, 678
 łączenia, 90
 podziału, 90, 97
 przechodności, 96
 rejestrowania, 843
 rejestrowania z zapisem z
 wypředzeniem, 766
 scalania, 950
 szeregowania, 830
 wycyfowania i powtarzania, 772
 zależności trywialnej, 92, 96
 zapisu Thomasa, 829
 zmienna, 221, 222
 rejestrowanie
 logiczne, 850, 865, 867, 868
 z możliwością powtarzania,
 757, 766, 767, 768, 771,
 772, 773, 780, 781
 z możliwością wycyfowania,
 756, 757, 766, 769, 771,
 773, 777, 781
 rekord, 35, 534, 538, 550, 552, 556
 COMMIT, 773, 846, 850, 867
 dzielony, 549, 556
 END CKPT, 774
 grupowanie, 955
 instrukcji, 375, 377
 klucz-wartość, 876, 899
 niemieszczący się w bloku,
 549, 556
 o stałej długości, 535, 536, 554
 o zmiennej długości, 545, 546,
 554, 622
 o zmiennym formacie, 548
 opisów, 375
 połączenia, 375
 rejestru, 36
 scalanie, 950, 958, 981
 środowiska, 375
 usuwanie, 553
 wstawianie, 552
 rekurencja, 233, 404, 439, 498,
 576, 577, 670, 672, 674, 845
 liniowa, 407
 relacja, 32, 34, 35, 43, 44, 45, 46,
 47, 48, 49, 51, 52, 60, 61, 65,
 66, 71, 76, 78, 111, 135, 162,
 168, 169, 181, 193, 199, 203,
 204, 210, 234, 418, 534, 629,
 732, 752, 780
 baza minimalna, 97
 dekompozycja, 85, 102, 103,
 104, 108, 114, 117, 124
 dopasowywana, 725
 ekstensjonalna, 224, 231
 generowana za pomocą
 obliczeń, *Patrz* widok
 intensjonalna, 224
 klucz, *Patrz* klucz
 nieskończona, 220
 niezmienna, 220
 obiektowa, 411, 413
 operacje, 271
 podstawowa, 725
 projekcja, 204, 240
 pusta, 689
 rekurencyjna, 404, 439
 schemat, 47, 48, 49, 50, 53,
 59, 60
 sklastrowana, 631, 635, 638,
 659, 660
 składowana, 322, *Patrz* tabela
 skończona, 220
 suma, 204
 synteza do trzeciej postaci
 normalnej, 117
 tymczasowa, 404
 więzy, 49, 79
 wirtualna, *Patrz* widok
 wyniki tymczasowe, *Patrz*
 wyniki tymczasowe
 zagnieżdżona, 411, 413, 440
 relacja podobieństwa, 952
 reprezentatywność, 952
 robot internetowy, 1002, 1004,
 1007, 1011, 1012, 1032
 rola, 140
 ROLAP, 430, 440
 rozkład Zipfa, 708
 rozszerzanie, 97

rozwijanie, 433
 równoległość, 869, 878
 równość wyznaczana przez
 metodę, 426
 równoważność, 207, 866

S

saga, 864, 865, 866, 868
 scalanie, 952, 991, 999
 schemat, 348, 349, 390, 395, 411,
 445, 462, 469, 535
 gwiazdy, 430, 440
 INFORMATION_SCHEMA,
 348
 sekwencyjność, 285
 selekcja, 228, 634, 659, 660, 687,
 689, 708, 713, 721, 735, 738, 874
 dwuargumentowa, 699
 przenoszenie w dół drzewa,
 689, 715, 719
 selektor, 467
 selektywność, 733
 Selinger, *Patrz* optymalizacja
 Selingera
 semantyka, 681, 787
 semilattice, *Patrz* półkrata
 seria, 620
 kodowanie długości, 620,
 621, 625
 serwer
 Apache/Tomcat, 346
 aplikacji, 345, 390
 baz danych, 345, 390
 SQL, *Patrz* SQL serwer
 WWW, 345, 346, 390
 sesja, 352
 set, *Patrz* zbiór
 shared-nothing, *Patrz*
 architektura bez
 współużytkowania
 silnik przetwarzania zapytań,
Patrz zapytanie silnik
 przetwarzania
 silnik wykonawczy, 34, 35, 36, 38
 skanowanie, 706
 SLQ, 439
 słownik, 189, 196
 słowo
 kodowe, 534
 pomijane, 570
 słowo kluczowe, 560, 567, 570
 ADD, 308
 ALTER TABLE, 56
 AS, 253
 ASC, 248

AUTHORIZATION, 351, 395
 CHECK, 303, 304, 305, 319
 CONSTRAINT, 307
 CREATE FUNCTION,
 364, 417
 CREATE TABLE, 55
 DATE, 54, 244
 DEFAULT, 57
 DESC, 248
 DISTINCT, 271, 272, 273,
 294, 440
 EMPTY, 456
 EXCEPT, 256, 293
 EXEC SQL, 355
 for, 485, 494, 503
 FROM, 238, 239, 242, 251,
 253, 263, 264, 265, 271,
 293, 325, 494
 GROUP BY, 248, 274, 275,
 277, 440
 HAVING, 248, 277
 INTERSECT, 256, 293
 INTO, 357
 inverse, 186
 JOIN, 265
 key, 191
 keys, 191
 let, 484, 494, 503
 METHOD, 417
 NATURAL, 268
 NATURAL JOIN, 266
 NOT NULL, 302
 ON, 265, 268
 order, 494
 ORDER BY, 248, 440, 629
 PRIMARY KEY, 58, 295, 296
 relationship, 186
 return, 484, 487, 488, 494, 503
 SELECT, 238, 239, 241, 251,
 253, 254, 271, 293, 325,
 494, 678, 682
 Struct, 186
 TIME, 54, 245
 TIMESTAMP, 245
 UNION, 256, 293
 UNIQUE, 295, 296
 UNIQUE, 58
 where, 487, 491, 503
 WHERE, 238, 239, 242, 251,
 253, 254, 260, 271, 293,
 304, 325, 359, 494, 576, 682
 sortowanie, 210, 215, 234, 425,
 440, 494, 504, 552, 560, 598,
 623, 629, 670, 706, 722, 742
 bąbelkowe, 44
 danych wyjściowych, 248
 w czasie skanowania, 629, 631

spam jednostronnikami, 1008, 1016,
 1018, 1033, 1034
 spanned record, *Patrz* rekord
 dzielony
 specyfikacja możliwości, 930
 spider trap, *Patrz* pułapka
 spójność, 37, 783, 784, 786, 799,
 839, 991, 996
 SQL, 33, 39, 47, 53, 210, 237,
 259, 271, 293, 388, 484, 504, 746
 2003, 237
 92, 237
 99, 237, 330, 404
 agent, 353, 390
 ANSI, 237
 dynamiczny, 361, 389, 391
 interfejs uniwersalny, 287,
 352, 353
 kod bezpośrednio osadzony, 354
 kod osadzony, 353, 391
 PSM, *Patrz* PSM
 serwer, 350, 351, 390
 SQL2, 237
 SQL3, 237
 środowisko, 348, 390
 transakcja, 284, *Patrz*
 transakcja
 wzorzec, 243
 START, *Patrz* dziennik rekord
 START
 statystyki, 36, 38, 718, 719
 przybliżone, 718
 striping, *Patrz* technika
 przeplatania
 Structured Query Language,
Patrz SQL
 struktura, 189
 danych, *Patrz* dane struktura
 drzewiasta, 816
 strumień
 bitowy, 1025
 danych, *Patrz* dane strumień
 kliknięć, 1020
 pakietów, 1020
 styl systemu R, *Patrz*
 optymalizacja Selingera
 suma kontrolna, 523, 524, 555
 support threshold, *Patrz* próg
 częstotliwości
 supporting relationship, *Patrz*
 związek pomocniczy
 system
 bankowy, 32
 informacji geograficznej, 595
 obsługi strumieni danych, 40
 plików, 32

system
 projektowy, 862
 rezerwacji biletów, 32
 sterowania procesem pracy, 863
 wymiany plików, 33, 40, 899, 907, 909
 wypożyczania i zwracania, 862
 zarządzania bazami danych, *Patrz* DBMS
 zarządzania strumieniami danych, *Patrz* DSMS
 system plików, 36
 szablon, 496, 504
 rekurencyjny, 498
 szeregowałość, 783, 787, 789, 790, 792, 795, 796, 801, 802, 822, 826, 838, 839, 840, 843, 880
 konfliktowa, 783, 789, 840

Ś

ścieżka, 477, 503, 572
 ścisła równość obiektów, 426
 ślepa uliczka, 1009, 1011, 1012, 1033
 średnica, 991
 środowisko SQL, *Patrz* SQL
 środowisko

T

tabela, 43, 44, 53, 240, 348, 418, 421, *Patrz też* relacja
 CUBE, 435
 dwuwymiarowa, 47
 faktów, 429, 430, 435, 440
 obsługująca referencje, 419
 skanowanie, 629, 673
 tymczasowa, 53
 wymiarów, 430, 440
 tableau, *Patrz* obraz
 tablica, 189, 196, 729
 asocjacyjna, 387
 blokad, 798, 811, 812, 813, 838, 855
 haszowaniem, 968
 konwersji, 540, 541
 kubeków, *Patrz* kubelek
 tablica
 liczników, 972
 numeryczna, 387, 389
 odwzorowań, 538, 540
 przesunięć, 539, 553
 wskaźników, 584
 z haszowaniem, 559, 584, 598, 599, 607, 624, 637, 638, 813

dynamiczna, 587, 601
 liniowym, 590, 591, 624
 podzielonym, 598, 603, 604, 624
 rozproszonym, 898, 900, 909
 rozszerzalnym, 587, 588, 590, 624
 statyczna, 587, 588
 usuwanie elementów, 586
 wstawianie elementów, 585
 wydajność, 587
 tag, 39, 45, 46
 tagged field, *Patrz* ciąg
 opisanych pól
 technika
 LSH, *Patrz* LSH
 OLAP, *Patrz* OLAP
 przeplatania, 517
 teleportacja, 1012, 1014, 1015, 1016, 1017, 1018, 1033
 teoria kodów korekcyjnych, 529, 534
 terabajt, 33
 tombstone, *Patrz* znacznik
 usunięcia
 tożsamość, 183, 489
 transakcja, 36, 37, 39, 237, 284, 286, 287, 294, 428, 535, 750, 780
 anulowanie, 756, 762
 atomowa, *Patrz* atomowość
 długa, 862
 izolacja, 288, 290, 291, 294, 784, 839
 kompensująca, 852, 864, 865, 866
 komponent, 880
 menedżer, 37, 751, 780
 niekompletna, 759
 niezatwierdzona, 759
 odraczanie, 812
 przetwarzanie, 39
 rozproszona, 888, 889, 893, 908, 909
 sekwencyjna, 428
 spójność, 796, 803
 stan, 750
 trwałość, 36
 tylko do odczytu, 287, 288
 wycofanie, 288, 294, 773
 z blokowaniem dwufazowym, 799, 801, 803
 zatwierdzanie, 889, 909
 zatwierdzanie grupowe, 849, 850, 867
 zatwierdzona, 759, 768
 transition matrix, *Patrz* macierz przejść

trend, 427
 trwałość, *Patrz* transakcja
 trwałość
 tryb
 IN, 364
 INOUT, 364
 OUT, 364
 trzecia postać normalna, 116, 118, 125
 twierdzenie LMSS, 946, 947
 two-phase, multiway merge sort, *Patrz* DWSPS
 typ, 682
 BOOLEAN, 54
 CHAR, 54, 243
 DATE, 55
 DECIMAL, 55
 definiowany przez
 użytkownika, *Patrz* UDT
 DOUBLE PRECISION, 55
 FLOAT, 55
 INT, 54
 INTEGER, *Patrz* typ INT
 kolekcji, 189
 NUMERIC, 55
 podstawowy, 473, 484, 491, 503
 prosty, 188, 189, 193, 471
 z ograniczeniami, 466, 471
 REAL, *Patrz* typ FLOAT
 SHORTINT, 54
 TIME, 55
 TIMESTAMP, 245
 VARCHAR, 54, 243
 wskaźnikowy, 418, 421, 440
 wyliczeniowy, 466
 złożony, 463, 465, 472

U

uchwyt, 376
 instrukcji, 378
 UDT, 415, 416, 421, 422, 423, 425, 440
 UML, 39, 135, 143, 174, 178, 181, 199, 445
 Unified Modeling Language, *Patrz* UML
 UPDATE, 282
 update lock, *Patrz* blokada
 z aktualizacją
 uprawnienia, 393, 394, 439
 diagram, 398, 439
 przyznawanie, 397, 439
 user-defined type, *Patrz* UDT
 usuwanie danych, 281
 UTF-8, 450

V

valid, *Patrz* XML prawidłowy

W

walidacja, *Patrz* poprawność

warning lock, *Patrz* blokada ostrzegawcza

warstwa

abstrakcji, *Patrz* warstwa pośrednia

aplikacji, 346, 347

bazy danych, 347

pośrednia, 34, 914, 915, 916

wartość

domyślna, 57, 463

FALSE, 246, 247

logiczna, 220, 242, 261, 310

minhash, 977, 978, 979, 981, 989, 998, 999, 1032

null, 169, 171, 172, 199, 210, 234, 245, 276, 293

pusta, *Patrz* wartość null

skalarna, 260, 261, 273

TRUE, 247

UNKNOWN, 246, 247

warunek, 313, 319, 395, 479, 503, 678, 689

bezpieczeństwa, 222

wektor bitowy, 621, 622, 625

well-formed, *Patrz* XML

poprawny składniowo

węzeł, 137, 231, 446, 468, 471, 474, 475, 484, 503, 577, 598, 678, 697, 825, 899, 900

awaria, 907

końcowy, 446

początkowy, 446

potomny, 198

równorzędnym, 900, 904, 905

wewnętrzny, 446, 608, 611, 612

widok, 53, 321, 343, 348, 681,

682, 746, 920, 938, 940, 957

atrybut, 323

kandydujący, 342

modyfikacja, 324, 325, 327

modyfikowalny, 324, 325, 343

usuwanie, 324

zapytanie, 323

zmaterializowany, 337, 339, 340, 342, 344

wielozbiór, 188, 189, 195, 203,

204, 224, 232, 234, 271, 294, 414, 473, 638, 687, 693

część wspólna, 205, 206, 639, 648, 649, 713

iloczyn, 640

iloczyn kartezjański, 207

krotek, 473

prawa algebraiczne, 207

projekcja, 206

różnica, 205, 639, 648, 713

selekcja, 207

suma, 205, 635, 648, 713

złączanie, 208

złączenie naturalne, 640

więzy, 39, 79, 82, 91, 176, 237, 416, 750

atrybutu, 302, 303, 305, 319, 394

cykliczne, 299

danych, *Patrz* dane więzy

dziedziny atrybutu, 82

integralności referencyjnej, 79, 80, 155, 156, 295, 297, 319, 394

klucza, *Patrz* klucz więzy

klucza obcego, 295, 296, 297, 299

krotki, 302, 304, 305, 308, 310, 319, 394

modyfikowanie, 307

nazwa, 307, 308

relacji, *Patrz* relacja więzy

sprawdzanie, 308, 319

usuwanie, 307

właściciel, 395

wrapper, *Patrz* nakładka

wskaźnik, 440, 539, 540, 546, 554,

560, 561, 562, 574, 596, 624

PageRank, 1004, 1005, 1006,

1007, 1008, 1010, 1012,

1014, 1015, 1016, 1017,

1032, 1033

przemienianie, 539, 540, 542,

543, 556

automatyczne, 541, 556

na żądanie, 541, 556

TrustRank, 1017, 1018

negatywny, 1018

wyszukiwanie, 541

współbieżność, 783, 816, 821, 827, 834, 864

współczynnik spamu, 1018

współużytkowanie, 869

wstawianie danych, 280

wstępne pobieranie, 520

wycofywanie, 756, 757, 759

wyliczenie oparte na kosztach, 706

wymiar, 430

wynik pośredni, *Patrz*

wyniki tymczasowe, 322

wyrażenia równoważne, 70

wyrażenie

algebraiczne, 685, 746

LIKE, 244, 245

logiczne, 309

warunkowe, 242

względne, 476

XPath, 475, 479

wyszukiwarka, 40, 1001, 1005,

1006, 1008, 1016, 1017,

1018, 1032

wyzwalacz, 39, 237, 309, 312, 313,

314, 319, 324, 348, 395, 750

INSTEAD OF, 327, 343

wzorzec, 243, 427

X

XML, 39, 44, 45, 445, 449, 454,

471, 496, 503, 504, 915

poprawny składniowo, 450

prawidłowy, 450

Schema, 453, 462, 463, 467, 469, 471

XML Schema, 39

XPath, 468, 473, 475, 479, 484, 496, 503

XPATH, 39

XQuery, 39, 473, 483, 484, 489,

490, 491, 503, 915

kwantifikator, 492

rozgałęzienie, 493

XSLT, 39, 496, 500, 504

rozgałęzienie, 501

Y

YouTube, 33

Z

zagłodzenie, 860

zakleszczenie, 37, 756, 801, 806,

854, 855, 856, 857, 860, 861,

867, 868, 896

zależność

funkcyjna, 85, 87, 88, 89, 90,

96, 97, 116, 118, 119, 123, 125, 127, 128, 130

łączenie kaskadowe, 96

nietrywialna, 103, 105, 120

projekcja, 98, 131

trywialna, 91, 94

wielowartościowa, 119, 120,

121, 123, 125, 127, 128, 130

- zapytanie, 35, 36, 39, 44, 53, 238,
251, 281, 323, 340, 344, 378,
380, 388, 389, 427, 503, 565,
567, 679, 703, 946, 1002, 1029
ciągłe, 1023
część wspólna, 256
do wyszukiwarki, 1005, 1032
FLWR, *Patrz* FLWOR
interfejs uniwersalny, 238
kompilator, 35, 38, 685, 706, 746
kompilowanie, 39
koniunkcyjne, 941, 942, 944,
945, 947, 948, 957, 958
o położenie, 596, 624
o sąsiada, 596, 603, 607, 608, 624
OLAP, *Patrz* OLAP
optymalizator, 38, 628, 668
optymalizowanie
na możliwościach, 929,
939, 957
na podstawie kosztów, 929,
932, 957
parser, 38
plan fizyczny, 628, 629, 630,
673, 703, 706, 707, 715,
721, 722, 734, 735, 736,
741, 742, 744, 746, 747
plan logiczny, 628, 634, 673,
697, 703, 704, 706, 715,
719, 721, 734, 741, 746
plan przetwarzania, 35, 38,
39, 931
preprocesor, 38
procesor, *Patrz* procesor
zapytań
rozproszone, 881
różnica, 256
SELECT-FROM-WHERE,
238, 239, 260, 263, 265,
271, 293, 310, 357, 358,
484, 559, 678
semantyka, 38
silnik przetwarzania, 1002
suma, 256
szablon, 924
wspomagające podejmowanie
decyzji, *Patrz* OLAP
z dopasowaniem częściowym,
596, 602, 608, 611, 624
zagnieżdżone, 264
zakresowe, 576, 596, 602, 607,
608, 611, 624
zarządzanie pamięcią, 39
zasada poprawności, 752, 753, 784
zbiór, 161, 163, 169, 174, 188,
189, 204, 414, 638, 687
część wspólna, 639, 648, 649
iloczyn, 640
logika, *Patrz* logika zbiorów
prawa algebraiczne, 207
różnica, 639, 648
słaby, 158
suma, 639
złączenie naturalne, 640
zbiór związku, 138
ZF, *Patrz* zależność funkcyjna
Zipf, *Patrz* rozkład Zipfa
złączenie, 216, 251, 489, 634, 649,
650, 659, 703, 713, 722, 874,
917, 931
argument, 725
bezstratne, 108, 111, 113,
117, 118
częściowe, 882, 883, 884,
886, 908
drzewo, *Patrz* drzewo złączeń
jednoprzebiegowe, 727, 737
kolejność, 724, 728
naturalne, 65, 68, 110, 111,
266, 638, 662, 704, 710,
711, 712, 882, 883, 884
oparte na indeksie, 662, 663
przez haszowanie, 655, 657,
668, 722
hybrydowe, 656
przez sortowanie, 650, 651,
652, 722, 737
równościowe, 638, 710, 883
teta, *Patrz* złączenie
warunkowe
warunkowe, 68, 266, 268, 638,
689, 704, 710
wieloargumentowe, 704
wieloprzebiegowe, 737
wielu relacji, 712
zagnieżdżone, 641, 642, 644,
674, 727
zewnętrzne, 216, 234, 246,
267, 268
lewe, 268
prawe, 268
zmienna, 221, 230, 386, 489
anonimowa, 221
krotkowa, 253, 254, 255, 264,
423, 494
lokalna, 227
wspólna, 355, 356, 390
znacznik, 462, 474, 480, 496
czasu, 783, 826, 827, 829, 830,
831, 833, 834, 838, 839,
841, 858, 868
dowolny, 478
główny, 456, 475
otwierający, 449, 456, 471
semantyczny, 449
usunięcia, 539, 554, 556
zamykający, 449, 456, 471
znak
ucieczki, 245
wielkość, 241, 483
zrzut
pełny, *Patrz* archiwizacja
zrzut pełny
przyrostowy, *Patrz*
archiwizacja zrzut
przyrostowy
ZW, *Patrz* zależność
wielowartościowa
związek, 137, 151, 199, 416
atrybut, 142
binarny, 139, 143, 160, 174,
175, 187, 199
jeden do jednego, 139, 140,
144, 188, 199
jeden do wielu, 186
jest, 144, 146, 155, 169, 170, 198
krotność, 139, 187, 198
pomocniczy, 160, 161, 168
wiele do jednego, 139, 140,
143, 156, 160, 165, 174,
186, 188, 191, 197, 199, 564
wiele do wielu, 139, 186, 188,
199, 545
wieloargumentowy, 140,
174, 175
zwrotny, 186, 187, 199
zwijanie, 433
zwrotność, 97

SYSTEMY BAZ DANYCH

Kompletny podręcznik

KANON INFORMATYKI

Z kluczowej roli, jaką bazy danych odgrywają w codziennym życiu milionów ludzi, zdajemy sobie sprawę za każdym razem, gdy wpisujemy hasło w wyszukiwarce Google, robimy zakupy w internetowej księgarni czy logujemy się do swojego konta w banku. Szybkie, bezpieczne i niezawodne przetwarzanie oraz przechowywanie ogromnych ilości informacji stało się dziś strategicznym czynnikiem funkcjonowania większości firm, organizacji i instytucji państwowych. Ten ogromny potencjał współczesnych baz danych jest sumą wiedzy i technologii rozwijanych przez kilka ostatnich dziesięcioleci. Owocem tych prac jest przede wszystkim wyspecjalizowane oprogramowanie – systemy zarządzania bazami danych DBMS, czyli rozbudowane narzędzia do wydajnego tworzenia dużych zbiorów informacji i zarządzania nimi. Niestety, mają one jedną zasadniczą wadę – należą do najbardziej złożonych rodzajów oprogramowania.

W związku z tym trzech znanych naukowców z dziedziny IT z Uniwersytetu Stanforda – Hector Garcia-Molina, Jeffrey D. Ullman i Jennifer Widom – postanowiło stworzyć pierwszy kompletny podręcznik, wprowadzający do systemów baz danych. Zawiera on opis najnowszych standardów bazy danych SQL 1999, SQL PSM, SQL CLI, JDBC, ODL oraz XML – i to w znacznie szerszym zakresie niż w większości publikacji. Podręcznik został przygotowany w taki sposób, aby po jego przeczytaniu użytkownik czy projektowanie baz danych, pisanie programów w różnych językach związanych z systemami DBMS oraz ich sprawna implementacja nie stanowiły dla Czytelnika najmniejszego problemu!

HECTOR GARCIA-MOLINA jest profesorem nauk komputerowych i inżynierii elektrycznej na Uniwersytecie Stanforda. Do jego zainteresowań naukowych należą biblioteki cyfrowe oraz integrowanie informacji i zastosowania baz danych w internecie. Jest laureatem wyróżnienia SIGMOD Innovations i członkiem prezydenckiej grupy doradczej do spraw technologii informacyjnych. Obecnie zajmuje stanowisko członka zarządu Oracle Corp.

JEFFREY D. ULLMAN jest emerytowanym profesorem nauk komputerowych na Uniwersytecie Stanforda. Jest autorem lub współautorem szesnastu podręczników informatycznych. Jego zainteresowania naukowe obejmują drążenie danych, integrowanie informacji i nauczanie wspomaganie elektronicznie. Jest członkiem National Academy of Engineering i laureatem wielu prestiżowych wyróżnień, m.in. Guggenheim Fellowship i Knuth Prize.

JENNIFER WIDOM jest profesorem nauk komputerowych i inżynierii elektrycznej na Uniwersytecie Stanforda. Jej zainteresowania naukowe obejmują wiele aspektów nietradycyjnego zarządzania danymi. Należy do organizacji ACM i National Academy of Engineering. Jest laureatką wyróżnienia ACM SIGMOD Edgar F. Codd Innovations i Guggenheim Fellowship.

Kompleksowe podejście do systemów baz danych — z punktu widzenia projektanta, użytkownika i programisty.

W tej książce znajdziesz obszernie omówienie między innymi:

- modelowania relacyjnego i wysokopoziomowego oraz języka UML i ODL
- zależności funkcyjnych i wielowartościowych oraz algorytmów umożliwiających manipulowanie zależnościami
- różnych aspektów programowania w języku SQL
- architektury trójwarstwowej, kostek danych, modelu relacji zagnieżdżonych i obiektowo-relacyjnych funkcji SQL-a
- XML-a i systemów opartych na tym języku
- zagadnień dotyczących sprawnego implementowania baz danych
- struktur używanych w indeksach, w tym drzew zbalansowanych, struktur haszujących i struktur dla indeksów wielowymiarowych
- wykonywania i optymalizowania zapytań, rejestrowania zdarzeń, kontrolowania współbieżności
- równoległych i rozproszonych baz danych, platformy Map Reduce, bazy danych P2P
- kluczowych technik, takich jak shingling, MinHash i LSH, służących do wyszukiwania podobnych elementów w dużych bazach danych
- algorytmów przeszukiwania sieci WWW, w tym algorytmu PageRank i jego odmian
- modelu strumieniowania danych i rozszerzenia języka SQL

helion.pl
księgarnia
internetowa

Nr katalogowy: 6656



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:

0 801 339900



0 601 339900



Helion

Sprawdź najnowsze promocje

➤ <http://helion.pl/promocje>

Książki najchętniej czytane

➤ <http://helion.pl/bestsellery>

Zamów informacje o nowościach

➤ <http://helion.pl/novosci>

Helion SA

ul. Koszowska 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po **WIECEJ**



KOD KORZYŚCI

ISBN 978-83-246-3303-6



Cena 99,00 zł

Informatyka w najlepszym wydaniu