BAZY DANYCH

Relacyjne Bazy Danych – ciąg dalszy

kierunek: Informatyka Medyczna

Adam Piórkowski

pioro@agh.edu.pl

http://home.agh.edu.pl/~pioro/dyd/

http://home.agh.edu.pl/~pioro/dyd/BDIM/

Kraków, 2025

- Współbieżność w RBD
- Indeksacja
- Optymalizacja zapytań SQL
- Rozproszone bazy danych
  - partycjonowanie
  - klasteryzacja
  - replikacja
  - hurtownie danych / proces ETL

WSPÓŁBIEŻNOŚĆ W RBD

Współbieżność w RBD

Transakcje w bazach danych.

Punkty bezpieczeństwa transakcji.

Kursory.

Transakcje a współbieżny dostęp wielu użytkowników.

Anomalie w transakcjach przeprowadzanych współbieżnie.

Utracone modyfikacje. Niepowtarzalne odczyty. Błędne odczyty. Pojawianie się fantomów.

Poziomy izolacji transakcji.

Odczyt niezatwierdzonych danych. Odczyt zatwierdzonych danych. Stabilność na poziomie kursora. Odczyt powtarzalny. Transakcje szeregowalne.

Zarządzanie poziomem izolacji transakcji w SQL.

Blokady w bazach danych.

Blokady wierszy. Blokady atrybutów. Blokady tabel. Rodzaje blokad.

Zakleszczenia.

Obsługa blokad w systemie baz danych.

Współbieżność w RBD

Systemy zarządzania bazami danych oferują możliwość wielodostępu. Jednoczesne operacje odczytu/zapisu/modyfikacji przez wielu użytkowników odbywają się współbieżnie. W celu zachowania spójności danych wprowadza się następujące mechanizmy:

- transakcje,
- blokady,
- dzienniki zdarzeń.

Szeregowanie operacji przez s.z.b.d. – rozpoczęcie wykonania każdej operacji ma miejsce po zakończeniu wykonywania poprzedniej.

Transakcje w bazach danych

Transakcja – grupa operacji na danych, z zewnątrz widoczna jako jedna operacja.

Cechy transakcji (ACID):
- *Atomic* – atomowa – widziana z zewnątrz jako jedna operacja atomowa na danych
- *Consistent* – spójna – wykonanie transakcji nie powoduje utraty spójności danych
- *Isolated* – izolowana – działanie transakcji jest izolowane od innych
- *Durable* – trwała – zmiany pomyślnego zakończenia transakcji są stałe

2PC – 2 Phase Commit – dwufazowy protokół zatwierdzania
- faza pierwsza – wysłanie zamówień na realizację składowych transakcji
- faza druga:
  - zatwierdzenie transakcji (*commit*), jeśli wszystkie zamówienia mają potwierdzenie gotowości do realizacji
  - wycofanie transakcji (*rollback*), jeśli co najmniej jedna składowa nie jest dostępna.

Zakończenie transakcji może być:
- jawne – wywołana instrukcjami COMMIT, ROLLBACK,
- niejawne – zakończenie sesji (->COMMIT), awaria systemu (->ROLLBACK), instrukcje definicji danych.

```
=> CREATE TABLE tab(a1 INT);

=> BEGIN;  (BEGIN TRANSACTION;)
=> UPDATE tab Set a1 = a1*2;
=> DELETE FROM tab WHERE a1>3;
=> COMMIT;
```

```
=> CREATE TABLE tab(a1 INT);
=> BEGIN;
=> UPDATE tab Set a1 = a1*2;
=> DELETE FROM tab WHERE a1>3;
=> ROLLBACK;
```

W przypadku awarii system bazy danych odtwarza wartości z logów.

Punkty bezpieczeństwa transakcji

W transakcjach przewidziane jest określenie momentu, do którego wycofywana jest transakcja. Mechanizm ten to punkty bezpieczeństwa transakcji. Zdefiniowane są one następująco (dla standardu SQL99):

- `SAVEPOINT nazwa` – utworzenie punktu ‘nazwa’,
- `RELEASE SAVEPOINT nazwa` – unieważnienie punktu ‘nazwa’,
- `ROLLBACK TO SAVEPOINT nazwa` – wycofanie operacji od punktu ‘nazwa’.

```
CREATE TABLE testowa(tekst VARCHAR(20));

BEGIN;
INSERT INTO testowa VALUES('zostanie');
SAVEPOINT punkts;
INSERT INTO testowa VALUES('zniknie');
ROLLBACK TO SAVEPOINT punkts;
INSERT INTO testowa VALUES('zostanie2');
COMMIT;

=> SELECT * FROM testowa;
    tekst
--------------
 zostanie
 zostanie2
(2 rows)
```

```
BEGIN;
INSERT INTO testowa VALUES('11111');
SAVEPOINT punkts;
INSERT INTO testowa VALUES('22222');
RELEASE SAVEPOINT punkts;
SAVEPOINT punkts;
INSERT INTO testowa VALUES('33333');
ROLLBACK TO SAVEPOINT punkts;
INSERT INTO testowa VALUES('44444');
COMMIT;

=> SELECT * FROM testowa;
 tekst
-------
 11111
 22222
 44444
(3 rows)
```

Kursory

Kursory (podobnie do widoki) są oparte o zadane zapytania, z tym, że przechowują otrzymane dane i umożliwiają do nich stopniowy dostęp (po jednym wierszu). Zalecane są do przetwarzania tabel z dużymi ilościami wierszy.

```
DECLARE nazwa_kursora [BINARY][INSENSITIVE][SCROLL] CURSOR FOR <instrukcja SELECT>
ALLOCATE nazwa [INSENSITIVE] [SCROLL] CURSOR FOR instrukcja      (SQL dynamiczny)
OPEN nazwa_kursora [FOR SELECT * FROM tabela]  /  [USING ... (SQL dynamiczny) ]
FETCH [ # | NEXT | PRIOR | FIRST | LAST | ABSOLUTE # | RELATIVE # ] FROM kursor INTO cel
MOVE [ FORWARD | BACKWARD | RELATIVE ][ # | ALL | NEXT | PRIOR ]{ IN | FROM } kursor

CLOSE nazwa_kursora
```

Kursor binarny – przetwarzanie zapytania do postaci tekstowej może zająć dużo pamięci i może być mniej użyteczne, jeśli na danych wykonywane są operacje binarne.

Kursory statyczne i dynamiczne (SQL92, dynamiczne określanie zapytania)
Kursor przewijalny – nawigacja w obie strony (tylko do odczytu) (inaczej tylko NEXT).
Rodzaje kursorów:

- nieokreślone,
- tylko do odczytu,
- niewrażliwe - ignorują zewnętrzne zmiany w trakcie otwarcia.

Utworzony w obszarze bezpieczeństwa kursor znika; utworzony wcześniej zostaje, a nawigacja w obszarze *nie* przepada.

```
BEGIN;
DECLARE kursor CURSOR FOR
        SELECT * FROM testowa;
FETCH ABSOLUTE 2 FROM kursor ;
CLOSE kursor;
COMMIT;

 tekst
-------
 22222
(1 row)
```

Anomalie w transakcjach przeprowadzanych współbieżnie

O ile pojedyncze operacje mogą być szeregowane, to zestawy operacji dwóch lub więcej użytkowników mogą być wykonywane w sposób przeplatany. Dotyczy to także zestawów operacji, objętych wewnątrz transakcji.

W trakcie współbieżnego wykonywania się dwóch lub więcej transakcji mogą pojawić się następujące anomalie:

- utracone modyfikacje (*lost updates*),
- błędne odczyty (*dirty reads*), (ANSI)
- niepowtarzalne odczyty (*non-repeatable reads, fuzzy reads*), (ANSI)
- pojawianie się fantomów [zjaw] (*phantoms*). (ANSI)

Utracone modyfikacje

Anomalia ma miejsce, gdy dwie transakcje odczytują pewną daną i aktualizują ją, przy czym aktualizacja jednej transakcji jest nadpisywana aktualizacją drugiej transakcji.

```
CREATE TABLE dane(nrindeksu INT,
frekwencja INT);

INSERT INTO dane VALUES (123, 0);

BEGIN;
UPDATE dane
    SET frekwencja = frekwencja + 1
                WHERE nrindeksu = 123);
COMMIT;

SELECT * FROM dane;
 nrindeksu | frekwencja
-----------+------------
       123 |          2
(1 row)
```

```
BEGIN;
UPDATE dane
    SET frekwencja = frekwencja + 2
                WHERE nrindeksu = 123);
COMMIT;
```

Niepowtarzalne odczyty

Anomalia ta ma miejsce, gdy ta sama operacja odczytuje pewną zmienną dwukrotnie, przy czym przy drugim odczycie wartość jest zmieniona (zatwierdzona) przez inną transakcję.

```
CREATE TABLE Dane(a INT);
CREATE TABLE Srednia(sr INT);
CREATE TABLE SrBKP(sr INT);

INSERT INTO Dane VALUES(2);
INSERT INTO Dane VALUES(4);

BEGIN;
INSERT INTO Srednia
VALUES( (SELECT AVG(a) FROM Dane) );

INSERT INTO SrBKP
VALUES( (SELECT AVG(a) FROM Dane) );
COMMIT;

SELECT * FROM Srednia;

sr
----
  3
(1 row)

SELECT * FROM SrBKP;

sr
----
  2
(1 row)
```

```
BEGIN;
DELETE FROM Dane WHERE a = 4;
COMMIT;
```

Błędne odczyty

Anomalia ma miejsce, gdy w jedna transakcja modyfikuje dane, ale ich nie zatwierdza, natomiast druga transakcja odczytuje i wykorzystuje owe niezatwierdzone dane.

```
CREATE TABLE Dane2(nr INT);

INSERT INTO Dane2 VALUES (10);

BEGIN;
UPDATE Dane2 SET nr = nr+1;



ROLLBACK;
```

```
CREATE TABLE Dsr(sr INT);

INSERT INTO Dsr VALUES (1);



BEGIN;
UPDATE Dsr SET sr = (SELECT AVG(nr) FROM Dane2);
COMMIT;
SELECT * FROM Dsr;

sr
----
 11
(1 row)
```

Pojawianie się fantomów

Anomalia ta ma miejsce, gdy podczas wykonywania transakcji przy drugim odczycie zapytania do pewnej tabeli pojawia się rozwiązanie (dane dodane przez inną transakcję), którego wcześniej nie było.

```
CREATE TABLE Dane(a INT);
CREATE TABLE Liczba(c INT, d INT);
INSERT INTO Liczba VALUES(0,0);

BEGIN;

UPDATE Liczba SET c = c + 1
  WHERE EXISTS (SELECT * FROM Dane);



UPDATE Liczba SET d = d + 1
  WHERE EXISTS (SELECT * FROM Dane);

COMMIT;

SELECT * FROM Liczba;
  c | d
---+---
 0 | 1
(1 row)
```

```
BEGIN;
INSERT INTO Dane VALUES(1);
COMMIT;
```

Poziomy izolacji transakcji

W celu uniknięcia opisanych wcześniej anomalii wprowadza się mechanizmy definiujące różne poziomy izolacji transakcji:

- odczyt niezatwierdzonych danych (*uncommitted read*), (ANSI, poziom 0)
- odczyt zatwierdzonych danych (*committed read*), (ANSI, poziom 1)
- odczyt powtarzalny (*repeatable read*), (ANSI, poziom 2)
- transakcje szeregowalne** (serializable). (ANSI, poziom 3)

Wybór poziomu izolacji wiąże się z akceptacją występowania adekwatnych do poziomu izolacji anomalii.

Systemy Z.B.D mogą definiować inne poziomy izolacji transakcji (poza standardem), np:

- IBM DB2: Read stablility, C*ursor stability*,
- MS SQL: Read-Committed-Snapshot (RCSI), Snapshot Isolation (SI),
- ORACLE: Read-Only Isolation Level

Poziomy izolacji transakcji

Odczyt niezatwierdzonych danych

Odczyt niezatwierdzonych danych to podstawowy poziom izolacji transakcji. Dopuszcza odczyty nowych wartości danych przed ich zatwierdzeniem. Najsłabiej chroni przed anomaliami. Może być stosowany do transakcji, w których wspólne dane występują tylko w trybie odczytu.

Odczyt zatwierdzonych danych

Poziom izolacji ‘odczyt zatwierdzonych danych‘ chroni przed anomalią brudnego odczytu. Zmiany wprowadzone przez transakcję są widzialne w bieżącej transakcji, a w przypadku innych transakcji - dopiero po zatwierdzeniu transakcji bieżącej. Takie zatwierdzenie operacji może jednak doprowadzić do niepowtarzalnych odczytów.
Omawiany poziom izolacji blokuje dostęp do wybranych przez siebie rekordów w odczytywanej tabeli. Odwołanie się innej transakcji do zmodyfikowanego i nie zatwierdzonego jeszcze wiersza powoduje zablokowanie (oczekiwanie). Przykładowo: jeśli przy modyfikacji warunek ograniczający wybrał tylko 5 wierszy ze 100, to owe 5 wierszy jest zablokowanych.

Poziomy izolacji transakcji

Stabilność na poziomie kursora

Ten poziom izolacji pozwala na częściową, ale precyzyjną kontrolę transakcji nad efektami współbieżnego wykonania. Dane wytyczane przez kursor bieżącej transakcji nie są dostępne dla innych transakcji w trybie modyfikacji. Możliwe jest natomiast dodawanie nowych wierszy.

Odczyt powtarzalny

Poziom izolacji ‘odczyt powtarzalny‘ chroni przed anomaliami brudnego i niepowtarzalnego odczytu. Dzieje się tak za sprawą mechanizmu, w którym transakcja, która pierwsza zacznie odczytywać daną tabelę, blokuje ją przed dostępem innych transakcji, nie pozwalając na operacje dodawania, usuwania i modyfikacji wszystkich wierszy. Przykładowo: jeśli przy modyfikacji warunek ograniczający wybrał tylko 5 wierszy ze 100, to wszystkie 100 wierszy jest zablokowanych.

Poziomy izolacji transakcji

Transakcje szeregowalne

Poziom izolacji ‘transakcje szeregowalne’ jest mechanizmem redukującym opisane anomalie, jednakże w największym stopniu ograniczającym współbieżność wykonywania transakcji. Wykorzystywany tu jest mechanizm wielowersyjności, tzn. tworzone są wersje stanu tabel dla każdej transakcji z początkiem jej wykonywania. Przez cały czas wykonywania danej transakcji nie są dostępne zmiany (zatwierdzone/niezatwierdzone) przez inne transakcje.

Dwie rozpoczęte transakcje szeregowalne nie mogą modyfikować współbieżnie tych samych danych - jeśli dany zasób został zmodyfikowany i zatwierdzony przez inną transakcję, bieżąca transakcja może zakończyć się niepowodzeniem; jeśli inna transakcja zmodyfikowała zasób, ale została wycofana, bieżąca może dokonać modyfikacji.

W poziomie izolacji ‘transakcje szeregowalne’ liczy się moment rozpoczęcia danej transakcji. Stwarza to wrażenie możliwości poszeregowania kolejnych transakcji – stąd nazwa.

	utracone modyfikacje	Błędne odczyty	niepowtarzalne odczyty	pojawiające się fantomy
odczyt niezatwierdzonych danych	-	możliwe	możliwe	możliwe
odczyt zatwierdzonych danych	-	-	możliwe	możliwe
stabilność na poziomie kursora	-	-	możliwe	możliwe
odczyt powtarzalny	-	-	-	możliwe
transakcje szeregowalne	-	-	-	-

Zarządzanie poziomem izolacji transakcji w SQL

W języku SQL można określić poziom izolacji dla każdej transakcji:

```
SET TRANSACTION ISOLATION LEVEL [poziom];
```

(PostgreSQL – po BEGIN, MySQL – przed)

gdzie poziom określa się literałami:

- `SERIALIZABLE`
- `REPEATABLE READ`
- `READ COMMITTED`
- `READ UNCOMMITTED`

Blokady w bazach danych

Systemy relacyjnych baz danych pozwalają na blokowanie używanych zasobów przez transakcję. Blokowanie takie może być niejawne (poprzez mechanizmy transakcyjne) albo jawne (poprzez komendy użytkownika). Próba dostępu przez transakcję do zablokowanego zasobu kończy się zatrzymaniem wykonywania danej transakcji do momentu zwolnienia pożądanego zasobu.

Blokady można zakładać na:

- wiersze (małe ziarno blokowania, wysoka współbieżność),
- wybrane atrybuty,
- tabele (duże ziarno blokowania, niska współbieżność).
- strony na dysku – od strony systemowej.

Rodzaje blokad

IN (Intent None) – (T) – intencjonalna – właściciel blokady może czytać dane (także niezatwierdzone), ale nie może ich modyfikować
IS (Intent Share) – (T) – intencjonalna współdzielona – blokuje dane w trybie współdzielonym,
IX (Intent Exclusive) (T) – intencjonalna wyłączna – blokuje dane w trybie wyłącznym,,
SIX (Share With Intent Exclusive) – (T) – właściciel blokady może czytać i modyfikować, pozostałe transakcje tylko czytać
X (Exclusive) – (T, R) – wyłączny dostęp dla właściciela blokady
U (Update) – (T,R) - właściciel blokady może czytać i modyfikować, pozostałe transakcje tylko czytać wiersze nieobjęte modyfikacjami
Z (Super Exclusive) – (T) – oprócz wyłącznego dostępu do danych blokowany jest dostęp do schematów relacji.

Blokady wierszy/kolumn

Blokady wierszy

Blokowanie wierszy może odbywać się w trybie wyłączności lub współdzielenia.

(Postgresql)

`SELECT` atrybuty `FROM` tabele `WHERE` warunek_wyboru_wierszy [...] `FOR UPDATE` [`OF` wybrane tabele] [`NOWAIT`]

`SELECT` atrybuty `FROM` tabele `WHERE` warunek_wyboru_wierszy [...] `FOR SHARE` [`OF` wybrane tabele] [`NOWAIT`]

Tryb UPDATE – wyłączny dostęp do wybranych wierszy, blokuje wszelkie operacje innych transakcji

Tryb SHARE – współdzielony dostęp do wybranych wierszy – inne transakcje mogą odczytywać dane bez blokowania

Blokady kolumn (atrybutów)

(Oracle)

`SELECT` atrybuty `FROM` tabele `WHERE` warunek_wyboru_wierszy [...] `FOR UPDATE` [`OF` wybr. kolumny] [`NOWAIT`]

Blokowane są wiersze, które posiadają wybrane atrybuty.

Blokady tabel

Blokady tabel

Dana transakcja może zablokować dostęp do tabeli.

`LOCK [ TABLE ]` lista tabel `[ IN` tryb `MODE ] [ NOWAIT ]`

tryb:

```
ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE |
SHARE UPDATE EXCLUSIVE | SHARE | SHARE ROW EXCLUSIVE |
EXCLUSIVE | ACCESS EXCLUSIVE
```

ACCESS SHARE – pozostałe transakcje mają dostęp do zablokowanej tabeli tylko w trybie odczytu
ROW SHARE – blokowanie modyfikowanych wierszy tylko do odczytu dla innych transakcji
ROW EXCLUSIVE – pełne blokowanie innym transakcjom wierszy przez komendy wykonujące zmiany danych
SHARE – chroni przed współbieżnymi zmianami w bazie,
EXCLUSIVE – dostęp w trybie tylko do odczytu dla transakcji blokujących zasób w trybie ACCESS SHARE
ACCESS EXCLUSIVE – blokuje wszelkie dostępy innych transakcji, nie pozwala zakładać innym transakcjom blokad

Zakleszczenia / obsługa blokad

Możliwe jest wystąpienie zakleszczenia w bazie danych na skutek niewłaściwie założonych blokad.

Przykład zakleszczenia sesji:

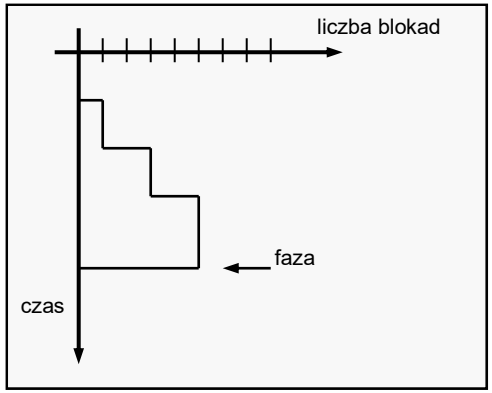
```
BEGIN;
UPDATE tab SET ocena = 5 WHERE index = 1;
UPDATE tab SET ocena = 4 WHERE index = 2;
...
```

```
BEGIN;
UPDATE tab SET ocena = 5 WHERE index = 2;
UPDATE tab SET ocena = 4 WHERE index = 1;
...
```

Obsługa blokad w systemie baz danych

Eskalacja blokad - w przypadku przekroczenia maksymalnej liczby blokad w danym zasobie, system przełącza blokadę na poziom wyżej (np. z wybranych wierszy na tabelę). Chroni to przed nadmiernym wykorzystaniem zasobów.

Czas życia blokad



INDEKSACJA W RBD

Indeksacja atrybutów w RBD

Indeks gęsty – zawiera klucze do wszystkich krotek w tabeli

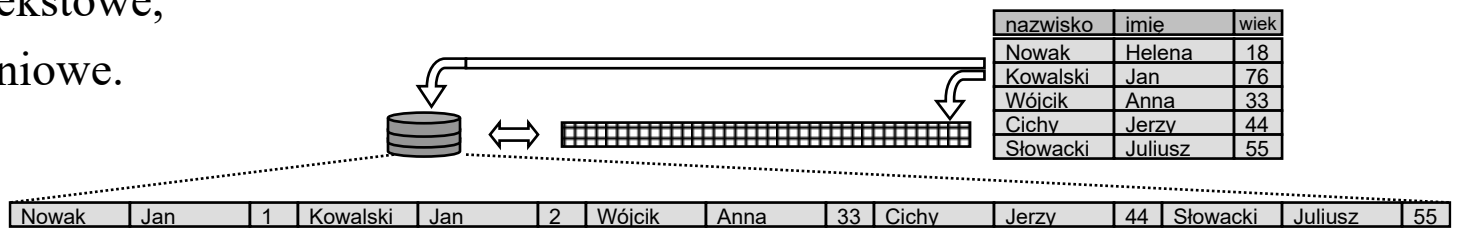
Indeks rzadki - zawiera klucze do wybranych krotek w tabeli

Metody indeksowania:

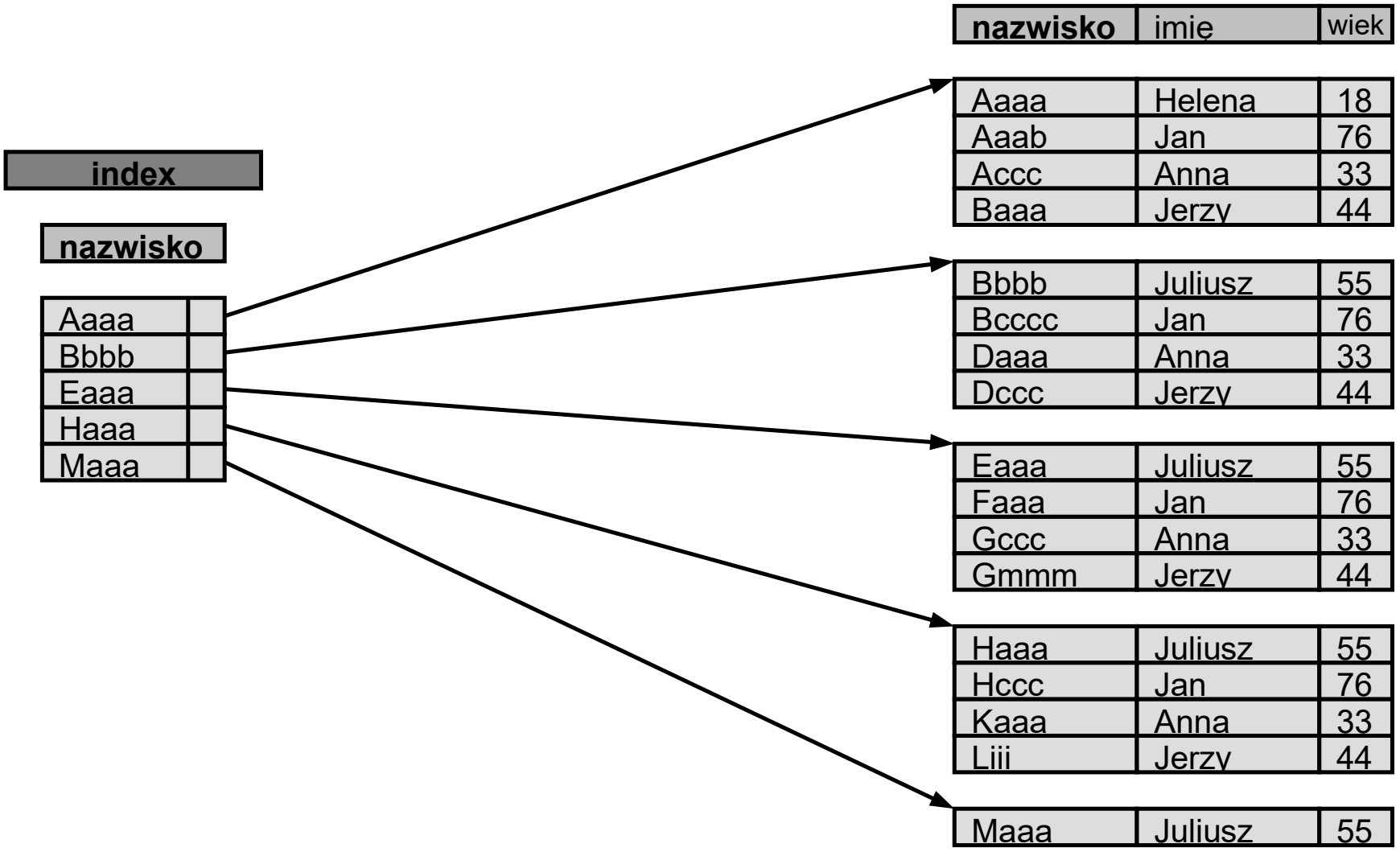
- ISAM,
- B-drzewa - B-TREE/B*-TREE
- R-drzewa – R-TREE/R*-TREE
- indeksy oparte o tablice mieszające (HASH)
- indeksy bitmapowe

Rodzaje indeksów:

- indeksy proste/złożone,
- indeksy unikalne/nieunikalne,
- indeksy klastrowane,
- indeksy pełnotekstowe,
- indeksy złączeniowe.



Metoda ISAM (Indexed Sequential Access Method)



Metoda ISAM (Indexed Sequential Access Method)

Przeszukiwanie indeksu

Liniowe przeszukiwanie indeksu:

1. porównywanie szukanej wartości z kolejnymi wpisami do tablicy indeksów aż do znalezienia pierwszej wartości obejmującej szukaną wartość,
2. przejście do bloku wskazywanego przez indeks
3. przeglądanie wskazanego bloku w poszukiwaniu wpisu

Przeszukiwanie indeksu metodą połowienia:

1. porównanie szukanej wartości ze środkowym wpisem w tabeli indeksów celem wydzielenia połowy, w której znajduje się szukana wartość,
2. rekurencyjna realizacja punktu 1 dla wydzielonej części tablicy wpisów aż do ograniczenia pojedynczego bloku,
3. przeglądanie wskazanego bloku w poszukiwaniu wpisu

Przeszukiwanie indeksu metodą interpolacyjną

- konieczna jest znajomość rozkładu wartości klucza – metoda interpolacyjna działa podobnie, jak metoda połowienia, różni się tym, że wydziela nie połowy, a części w oparciu o podaną funkcję rozkładu.

Metoda ISAM (Indexed Sequential Access Method)

Operacje na krotkach – wpisy swobodne (krotki można przesuwać w pliku, np. sortować)

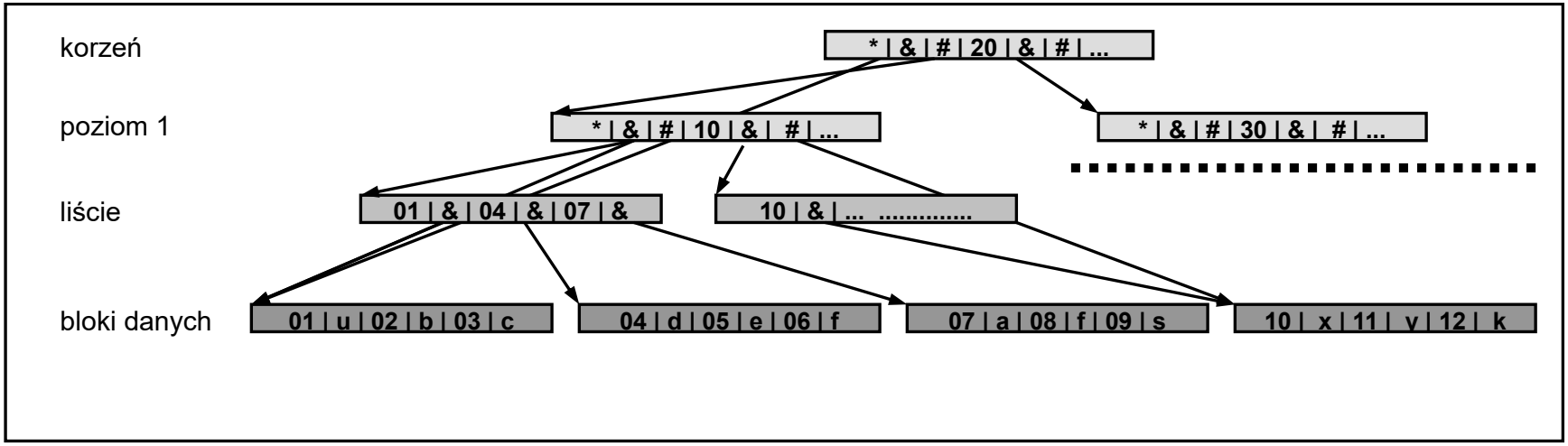
Modyfikacja krotki:
- jeśli zmiana nie dotyczy klucza -> operacja przeszukiwania
- jeśli klucz jest zmieniany -> następuje usunięcie starej krotki i wstawienie nowej.

Usuwanie krotki:
1. wyszukanie krotki,
2. usunięcie krotki z bloku,
- jeśli to była pierwsza krotka, to należy uaktualnić wpis do indeksu,
- jeśli to była jedyna krotka, to należy zwolnić blok.

Wstawienie krotki:
1. wyszukanie bloku B_i, zawierającego wartość klucza nowej krotki lub wartości mniejszą i większą,
2.
- jeśli blok B_i nie jest pełny, nowa krotka jest umieszczona w odpowiednim miejscu, przy 'rozsunięciu' pozostałych w bloku, jeśli to konieczne
  - jeśli to był pierwszy blok i wartość klucza krotki była mniejsza od wartości klucza indeksu tego bloku, to trzeba zmienić wpis indeksu dla tego bloku w tabeli indeksu
- jeśli blok B_i jest zapełniony, to:
  - jeśli następny blok (B_{i+1}) jest niepełny, to należy tam wstawić nadmiarowy rekord z B_i i uaktualnić wpis w tablicy indeksów dla B_{i+1}
    - jeśli B_i jest ostatni, to trzeba dodać nowy blok B_{i+1},

  albo
    - można wstawić nowy blok za B_i i rozdzielić zawartość B_i po połowie miedzy oba bloki.

Metoda B-drzew (drzewa wyważone)

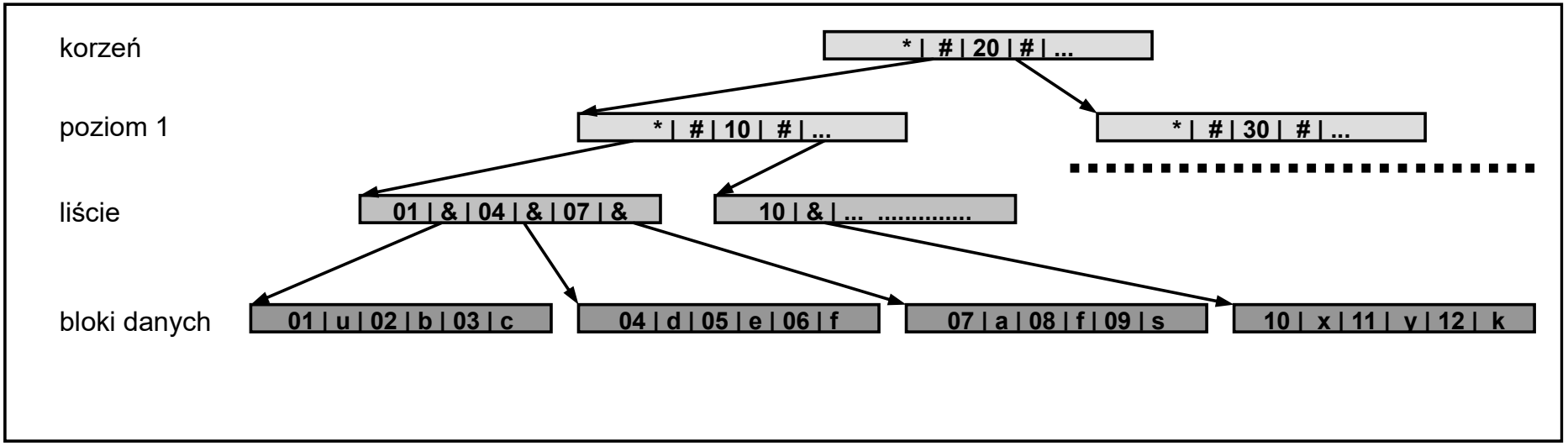


Cechy B-drzewa:

- bloki (strony) danych - zawierają dane posortowane w obrębie jednostki wg klucza, rozmiar zależny od rozmiaru bloku pamięci masowej – przechowywane tam jest *2d-1* krotek,
- bloki indeksu – mogą być węzłami pośrednimi lub węzłami liśćmi:
    - węzły pośrednie listami trójek:
      (wartość klucza, adres bloku węzła potomnego, adres bloku danych dla danego klucza)
    - węzły liście stanowią listę par:
      (wartość klucza, adres bloku danych dla danego klucza)

- każdy wierzchołek zawiera co najmniej *c* elementów (max. liczba elementów 2c+1 - jest zależna od rozmiaru bloku),
- wyszukanie dowolnego bloku danych zajmuje co najwyżej *h* dostępów,
- minimalne wypełnienie drzewa: korzeń ma 1 element, pozostałe wierzchołki *c* elementów,
- maksymalne wypełnienie drzewa: każdy wierzchołek zawiera 2*c* elementów.

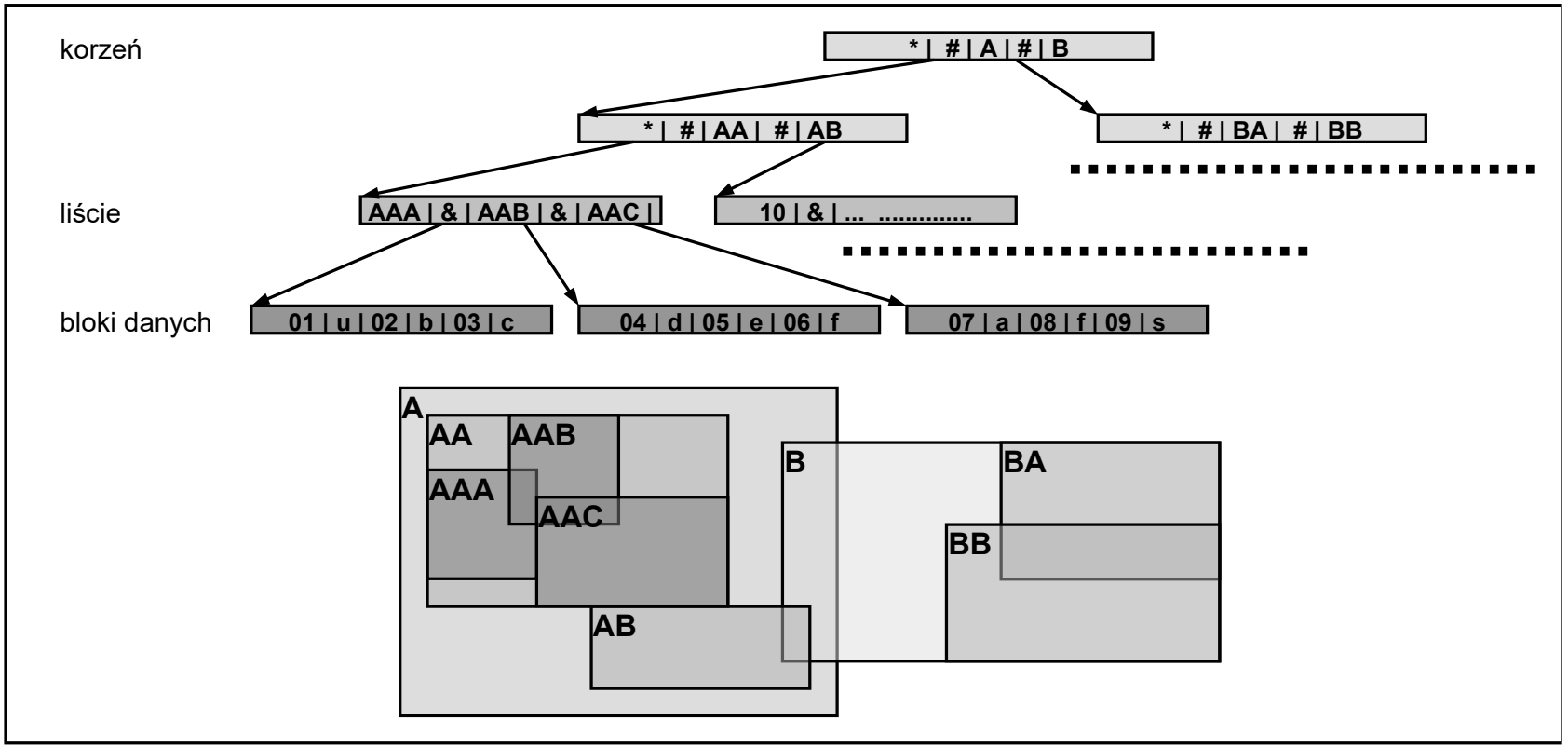
Metoda B*-drzew (drzewa wyważone)



- bloki indeksu – mogą być węzłami pośrednimi lub węzłami liśćmi:
  - węzły pośrednie listami par:
    (wartość klucza, adres bloku węzła potomnego)
  - węzły liście stanowią listę par:
    (wartość klucza, adres bloku danych dla danego klucza)

- bloki węzłów pośrednich zawierają więcej indeksów (+),
- wyszukanie dowolnego bloku danych zajmuje dokładnie *h* dostępów,
- możliwe jest połączenie liści – tworzy się wówczas posortowana lista (+).

Metoda R-drzew

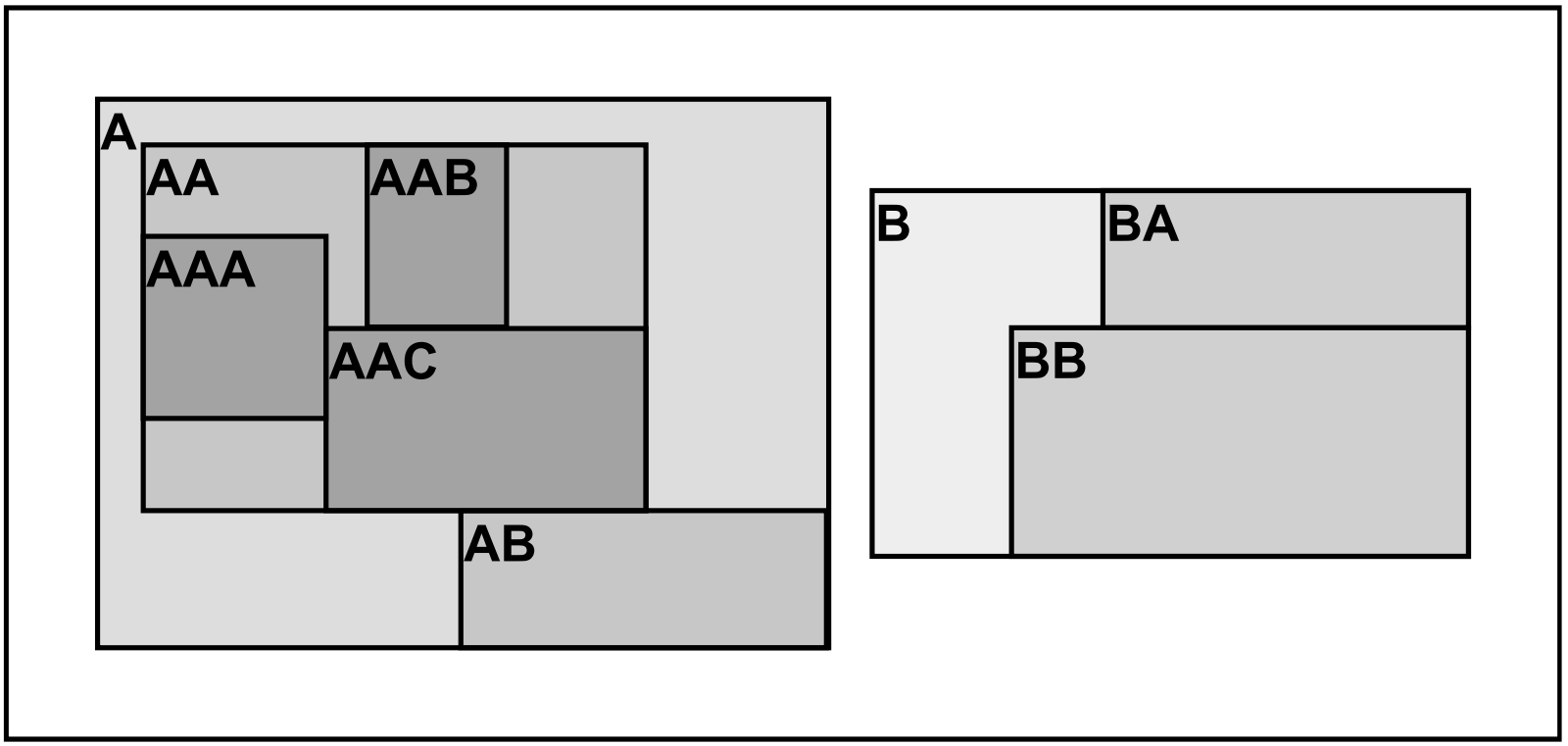


R-drzewa są to indeksy związane z obiektami wielowymiarowymi, posiadającymi pewien obszar dla wybranych atrybutów. Sprawdzają się jako indeksy dla atrybutów przeszukiwanych wg zakresu (ew. poszukiwania sąsiadów), np. ‘zawiera’.

Węzeł pośredni – opisuje pewien obszar, zawiera odnośniki do węzłów potomnych, których zakresy zawierają się wewnątrz tego obszaru. Obszary węzłów potomnych nie muszą tworzyć pełnego pokrycia węzła nadrzędnego, mogą mieć część wspólną.

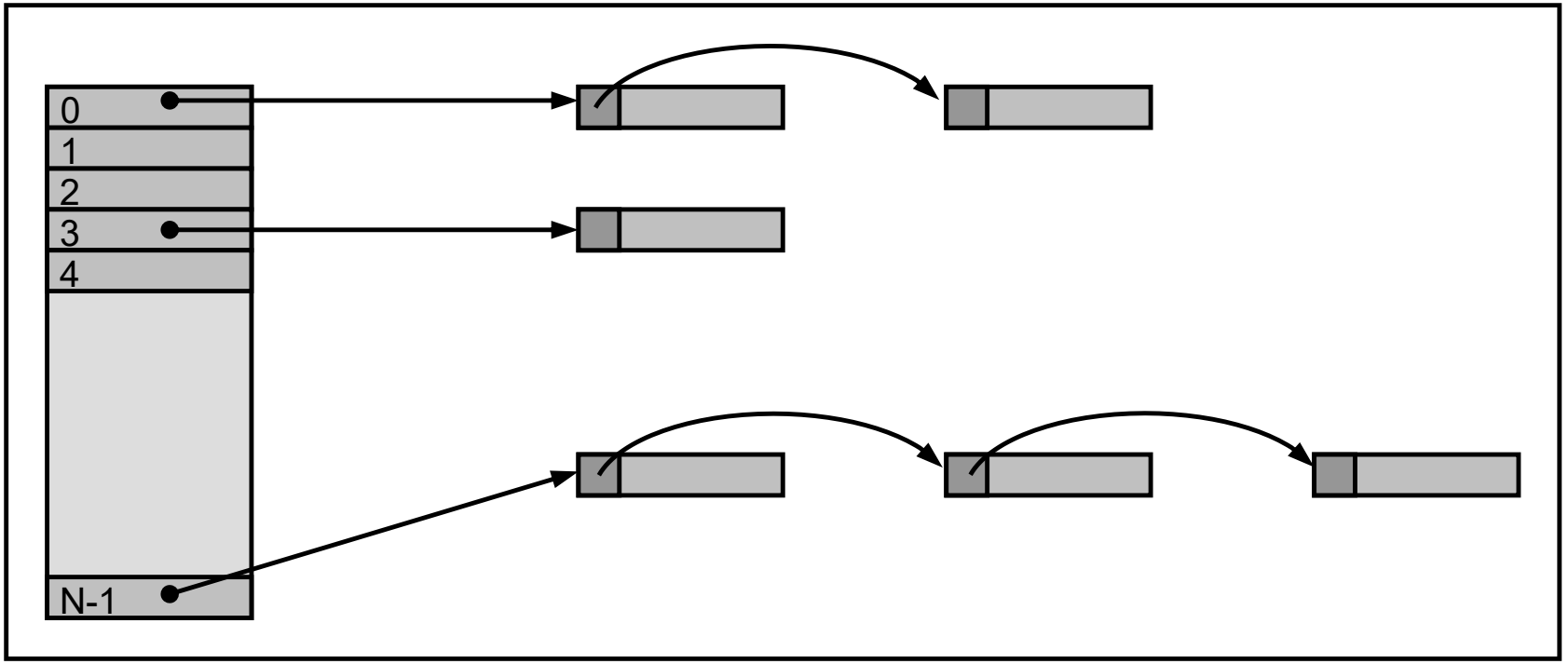
Węzeł liść – odpowiada indeksowanemu obszarowi i wskazuje na dane obiektu.

Metoda R+-drzew



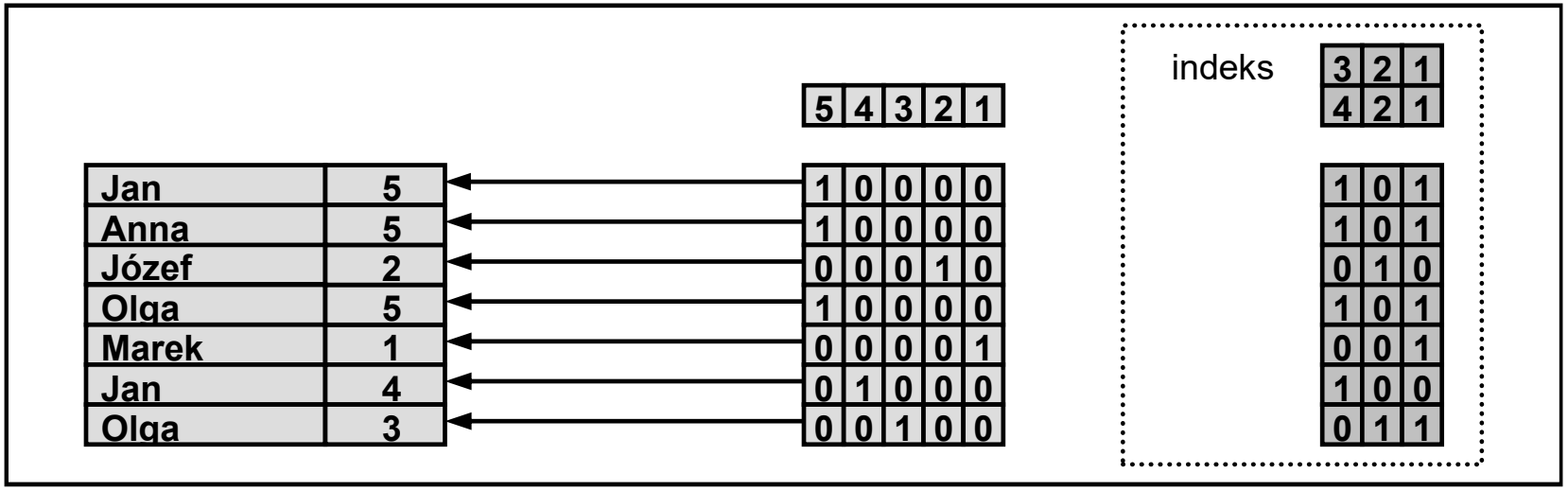
R+-drzewa to struktura danych analogiczna do R-drzew przy założeniu, że zakresy w węzłach potomnych nie mogą mieć części wspólnej.

Metoda tablic mieszających



Funkcja mieszająca – argument to wartość atrybutu, zwraca wskaźnik listy bloków obejmujących dany klucz.

Metoda bitmap



W przypadku niskiego zróżnicowania wartości atrybutu (np. student (rok_st)) stosuje się indeks bitmapowy.
Cechy charakterystyczne:

- operacje bitowe (wyszukiwanie) – szybkie,
- niewielka ilość danych, reprezentujących rekord,
- niewielki rozmiar indeksu (w porównaniu do B-drzewa),
- ułatwiona realizacja operacji logicznych i agregujących – wydajny dla zapytań z porównaniami i zliczeniami,
- jeśli liczba wartości wykroczy poza rozmiar indeksu, dodawany jest nowy,
- możliwość tworzenia zakresów wartości (redukcja rozmiaru indeksu,
- możliwa kompresja indeksu (np. algorytm RLE).

Implementacja: stworzenie indeksu dla wszystkich wierszy tabeli zrzuca konieczność przechowywania adresów wierszy, o ile istnieje funkcja odwzorowująca numer wiersza na jego adres fizyczny.

Rodzaje indeksów

Indeksy unikalne / nieunikalne

Wartości indeksowane mogą się powtarzać (jeśli zdecydowanie często => indeks bitmapowy). Nie są możliwe powtórzenia dla indeksów utworzonych domyślnie z kluczem głównym. Indeks nałożony na kolumnę unikalną również nie przewiduje powtórzeń.

Indeksy złożone

Indeksy można zakładać dla większej niż jedna liczby kolumn (ograniczenia narzucane przez dostawców, modyfikacja przez kompilację) należących do jednej tabeli.

- istotny w przypadku złożonego klucza głównego,
- zaleca się nakładać na kolumny występujące w warunku ograniczającym w zapytaniu,
- kolejność kolumn istotna!!! – zaleca się tworzenie indeksu z kolejnością kolumn wg malejącego zróżnicowania,
- pierwsza kolumna w kluczu związana jest z porządkiem sortowania,
- wiodące części indeksu (kol_1, kol_2, kol_3, kol_4, kol_5) to (kol_1), (kol_1, kol_2), (kol_1, kol_2, kol_3), ...
- indeks będzie wykorzystany w zapytaniach (wybrane s.z.b.d.) w przypadku wystąpienia pierwszej kolumny indeksu w warunku zapytania.

Rodzaje indeksów

Indeksy pełnotekstowe (full search index)

Indeksy pełnotekstowe są stosowane w przypadku przechowywania dużej ilości danych tekstowych. Ich idea jest podobna do skorowidza w książce. Indeksuje się istotne wyrazy, w oparciu o słownik danego języka odrzuca się popularne przyimki, zaimki, można posłużyć się istotnymi statystykami. Wyszukiwanie danego słowa (WHERE ... LIKE -> WHERE CONTAINS) omija skanowanie każdej linijki tekstu, tylko opiera się o index.
Dodatkową opcją jest ranking.

Tworzenie indeksu pełnotekstowego:
- całkowite,
- przyrostowe.

Dodawanie dokumentu:
- przekład tekstu na tokeny,
- zamiana tokenów na leksemy (znaczenie, niezależny od fleksji),
- tworzenie zestawu leksemów reprezentujących dokument (+ ranking)

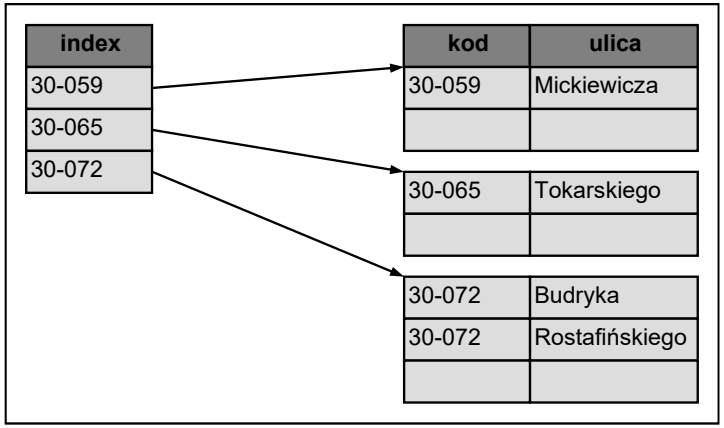
Rodzaje indeksów

Indeksy klastrowane (zgrupowane)

Indeks zgrupowany to indeks rzadki, wskazujący na posortowane na nośniku fizycznym dane wg indeksowanej kolumny. Każda unikalna wartość indeksu wskazuje na pierwszy blok, zwierający tę wartość.
Każda tabela może mieć tylko jeden indeks klastrowany (powód oczywisty).
Tworzenie indeksu wiąże się z uporządkowanym przepisaniem zawartości tabeli. Częste zmiany, dodawanie i usuwanie krotek wymagają reorganizacji zapisanej tabeli, co zmniejsza wydajność.
Współczynnik zapełnienia indeksu zgrupowanego określa procentowo ilość wolnego miejsca, aby dla danej wartości indeksu można było dodawać nowe rekordy bez każdorazowej reorganizacji. Jeśli strona indeksu się zapełni, następuje kosztowne dzielenie stron indeksu.

Indeksy klastrowane opłacalne:
- zapytania zwracają zakresy danych (GROUP BY),
- zapytania związane są z sortowaniem danych,
- dla każdej tabeli – wprowadza porządek w zarządzaniu pamięcią.

Indeksy klastrowane nieopłacalne:
- zapytania o niewielkiej liczbie rekordów wynikowych,
- częsta modyfikacja danych w kolumnie.



Rodzaje indeksów

Indeksy złączeniowe to indeksy stworzone na potrzeby złączeń tabel (optymalizacja).

Sybase

```
CREATE JOIN INDEX join-index-name FOR join-clause

join-clause:
    [ ( ] join-expression join-type join-expression
    [ ON  search-condition ] [ ) ]
join-expression:
    { table-name | join-clause }
join-type:
    [ NATURAL ] FULL [ OUTER ] JOIN
search-condition:
    [ ( ] search-expression [ AND  search-expression ] [ ) ]
```

Oracle - Bitmapowy indeks złączeniowy

```
SELECT  supplier_name FROM
   Parts NATURAL JOIN inventory NATURAL JOIN suppliers
   WHERE part_type = 'piston' AND state = 'nc' AND part_color = 'yellow'

CREATE BITMAP INDEX    part_suppliers_state
   ON          inventory( parts.part_type, supplier.state)
   FROM        inventory i, parts     p, supplier  s
   WHERE       i.part_id = p.part_id  AND   i.supplier_id = p.part_id;
```

Indeksacja w RBD

Indeks może być odczytywany przez s.z.b.d.:
- przegląd unikalnych wartości,
- przegląd zakresu wartości,
- pełny przegląd,
- złączenie oparte o indeksy.

Wyszukiwanie i odczyt wybranych rekordów – przegląd indeksu.
Odczyt wszystkich (większości) rekordów – przegląd sekwencyjny.

Tabela może zawierać zero, jeden lub więcej indeksów. Sterowanie indeksacją w systemach relacyjnych baz danych: pierwotnie objęty indeksacją jest klucz

Indeksacja jest opłacalna gdy:
- zachodzi częste przeszukiwanie tabeli,
- używane jest sortowanie/grupowanie,
- tworzone są złączenia z innymi tabelami,
- dla kolumn często występujących w warunkach złączenia (WHERE, ORDER BY, GROUP BY)

Indeksacja nie jest opłacalna, gdy:
- tabela zawiera niewiele wpisów,
- indeksowane kolumny zawierają dużo powtarzających się wpisów,
- tabele służą do gromadzenia a nie przeglądania wpisów,
- jest dużo utworzonych indeksów, a często występują modyfikacje, dodawania i usuwania krotek.

Indeksacja w RBD

Tworzenie usuwanie indeksu poleceniami SQL:

CREATE [UNIQUE] INDEX index_name
ON table [USING acc_name] (column [ops_name] [, ...]);

PostgreSQL: **acc_name**
- BTREE (Lehman-Yao high-concurrency btrees), (domyślny)
- RTREE (standard rtrees using Guttman's quadratic split algorithm),
- HASH (Litwin's linear hashing).

GiST, GIN – algorytmy specyficzne dla PostgreSQL

- usuwanie indeksu poleceniami SQL:

DROP INDEX index_name ;

Część s.z.b.d. narzuca indeksy unikalne w skali bazy, część w skali tabeli, stąd składnia DROP INDEX może być różna.

MySQL : Uses very fast B-tree disk tables (MyISAM) with index compression.

INDEKSACJA w PostgreSQL

```
bd=> SELECT version();
                           version
-------------------------------------------------------------
 PostgreSQL 10.10, compiled by Visual C++ build 1800, 64-bit
(1 row)

CREATE TABLE dziesiec(cyfra INT);
INSERT INTO dziesiec VALUES (0), (1), (2), (3), (4), (5), (6), (7), (8), (9);

-- metoda 1
CREATE TABLE milion(liczba INT PRIMARY KEY, innaliczba INT);

INSERT INTO milion SELECT setkitysiecy.cyfra * 100000 + dztysiecy.cyfra * 10000 +
  tysiace.cyfra * 1000 + setki.cyfra * 100 + dziesiatki.cyfra * 10 + jednostki.cyfra
                   FROM dziesiec setkitysiecy, dziesiec dztysiecy, dziesiec tysiace,
                            dziesiec setki, dziesiec dziesiatki, dziesiec jednostki;

-- alternatywnie (2)
SELECT setkitysiecy.cyfra * 100000 + dztysiecy.cyfra * 10000 + tysiace.cyfra * 1000
 + setki.cyfra * 100 + dziesiatki.cyfra * 10 + jednostki.cyfra AS liczba INTO milion
                   FROM dziesiec setkitysiecy, dziesiec dztysiecy, dziesiec tysiace,
                            dziesiec setki, dziesiec dziesiatki, dziesiec jednostki;
-- alternatywnie (3)
CREATE TABLE milion AS SELECT setkitysiecy.cyfra * 100000 + dztysiecy.cyfra * 10000
 + tysiace.cyfra * 1000 + setki.cyfra * 100 + dziesiatki.cyfra * 10 +jednostki.cyfra
         AS liczba FROM dziesiec setkitysiecy, dziesiec dztysiecy, dziesiec tysiace,
                            dziesiec setki, dziesiec dziesiatki, dziesiec jednostki;

-- do (2) i (3)
ALTER TABLE milion ADD PRIMARY KEY (liczba);
ALTER TABLE milion ADD COLUMN innaliczba INT;
```

INDEKSACJA w PostgreSQL

```
bd=> SELECT COUNT(*) FROM milion;
  count
---------
 1000000
(1 row)


-- uzupelnienie drugiej kolumny
bd=> UPDATE milion SET innaliczba = liczba+1;
UPDATE 1000000


-- przykladowa zawartość tabeli (12 pierwszych wierszy)
bd=> SELECT * FROM milion LIMIT 12;
 liczba | innaliczba
--------+------------
      0 |          1
      1 |          2
      2 |          3
      3 |          4
      4 |          5
      5 |          6
      6 |          7
      7 |          8
      8 |          9
      9 |         10
     10 |         11
     11 |         12
(12 rows)
```

INDEKSACJA w PostgreSQL

```
-- PRIMARY KEY


bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE liczba = 777;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using milion_pkey on milion  (cost=0.43..8.45 rows=1 width=8) (actual time=0.015..0.018 rows=1 loops=1)
   Index Cond: (liczba = 777)
 Planning time: 0.199 ms
 Execution time: 0.055 ms
(4 rows)


bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE liczba = 777;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using milion_pkey on milion  (cost=0.43..8.45 rows=1 width=8) (actual time=0.014..0.015 rows=1 loops=1)
   Index Cond: (liczba = 777)
 Planning time: 0.087 ms
 Execution time: 0.035 ms
(4 rows)


bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE liczba = 777;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Index Scan using milion_pkey on milion  (cost=0.43..8.45 rows=1 width=8) (actual time=0.025..0.026 rows=1 loops=1)
   Index Cond: (liczba = 777)
 Planning time: 0.131 ms
 Execution time: 0.053 ms
(4 rows)
```

INDEKSACJA w PostgreSQL

```
-- zwykla kolumna

bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE innaliczba = 777;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..17757.76 rows=1 width=8) (actual time=1.051..151.052 rows=1 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on milion  (cost=0.00..16757.66 rows=1 width=8) (actual time=26.723..73.242 rows=0 loops=3)
         Filter: (innaliczba = 777)
         Rows Removed by Filter: 333333
 Planning time: 0.103 ms
 Execution time: 151.085 ms
(8 rows)

bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE innaliczba = 777;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..17757.76 rows=1 width=8) (actual time=0.945..129.720 rows=1 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on milion  (cost=0.00..16757.66 rows=1 width=8) (actual time=19.943..53.857 rows=0 loops=3)
         Filter: (innaliczba = 777)
         Rows Removed by Filter: 333333
 Planning time: 0.106 ms
 Execution time: 129.786 ms
(8 rows)

bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE innaliczba = 777;
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..17757.76 rows=1 width=8) (actual time=0.915..129.833 rows=1 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on milion  (cost=0.00..16757.66 rows=1 width=8) (actual time=19.923..53.878 rows=0 loops=3)
         Filter: (innaliczba = 777)
         Rows Removed by Filter: 333333
 Planning time: 0.114 ms
 Execution time: 129.898 ms
(8 rows)
```

INDEKSACJA w PostgreSQL

```
-- Zalozenie indeksu

bd=> CREATE INDEX milion_innaliczba_indeks ON milion(innaliczba);
CREATE INDEX

bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE innaliczba = 777;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using milion_innaliczba_indeks on milion  (cost=0.42..8.44 rows=1 width=8) (actual time=0.066..0.067 rows=1 loops=1)
   Index Cond: (innaliczba = 777)
 Planning time: 0.330 ms
 Execution time: 0.090 ms
(4 rows)

bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE innaliczba = 777;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using milion_innaliczba_indeks on milion  (cost=0.42..8.44 rows=1 width=8) (actual time=0.018..0.019 rows=1 loops=1)
   Index Cond: (innaliczba = 777)
 Planning time: 0.126 ms
 Execution time: 0.041 ms
(4 rows)

bd=> EXPLAIN ANALYSE SELECT * FROM milion WHERE innaliczba = 777;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Index Scan using milion_innaliczba_indeks on milion  (cost=0.42..8.44 rows=1 width=8) (actual time=0.020..0.021 rows=1 loops=1)
   Index Cond: (innaliczba = 777)
 Planning time: 0.127 ms
 Execution time: 0.044 ms
(4 rows)
```

Optymalizacja zapytań SQL

Dostęp do danych (tabeli):
- pełny przegląd,
- dostęp sekwencyjny,
- indeksowy (indeks unikalny, przegląd indeksu w kierunku rosnącym/malejącym, przegląd zakresowy).

Cel optymalizacji:
- skrócenie czasy wykonywania zapytania - zmniejszenie czasów odpowiedzi,
- zwiększenie przepustowości - liczba transakcji w jednostce czasu,
- zwiększenie bezpieczeństwa systemu (ograniczenie zużycia pamięci),

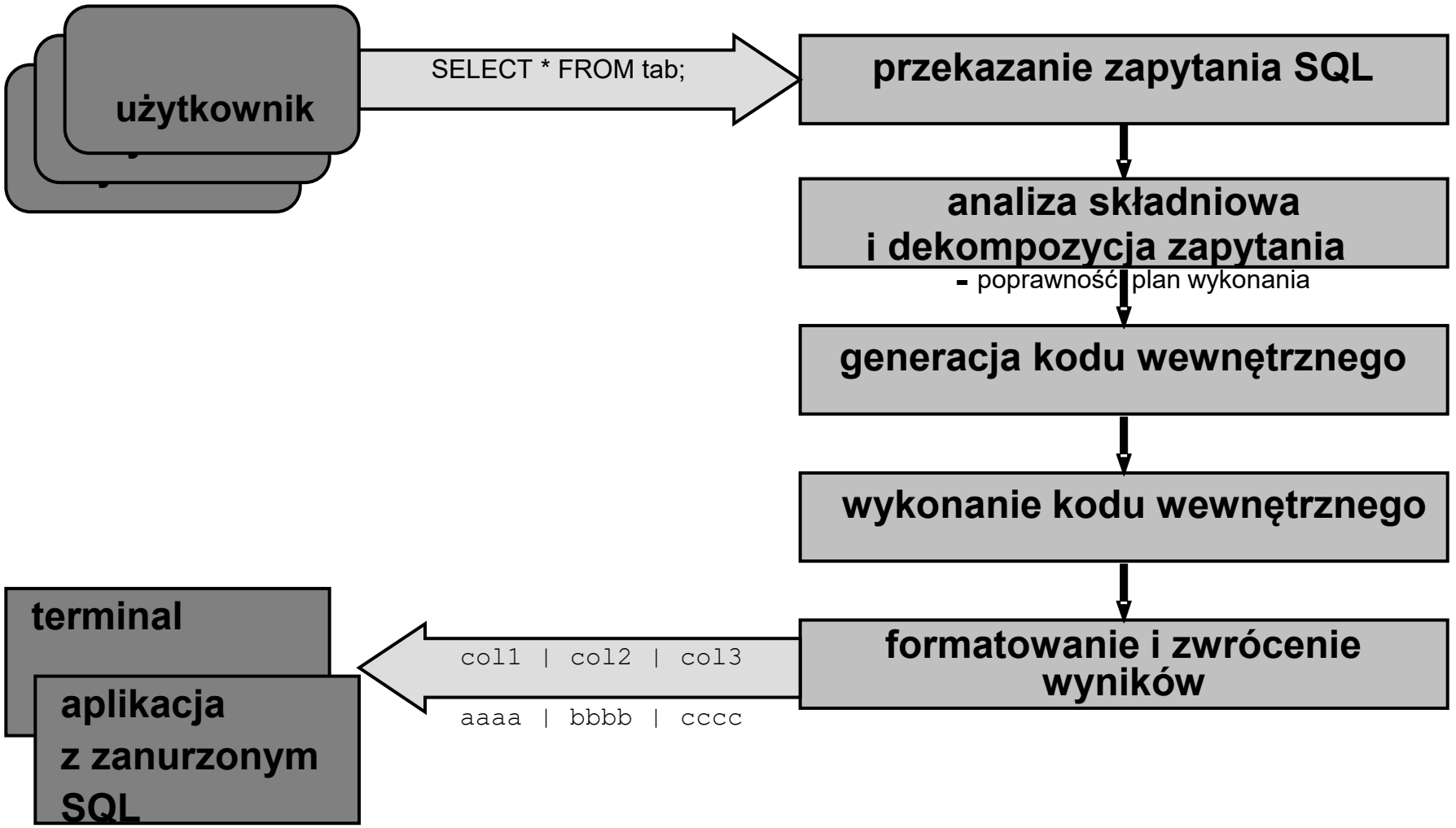
poprzez:
- zmniejszenie liczby dostępów do dysku (indeksacja, klasteryzacja, ..)
- eliminację niepotrzebnych/kosztownych operacji,
- wybór optymalnej kolejności wykonywania zadań.

Techniki optymalizacji zapytań w systemach baz danych

Optymalizacja pojedynczego zapytania SQL, najczęściej przed rozpoczęciem wykonywania.
Metody optymalizacji:
- analiza kodu SQL przez programistę,
- wskazówki – dawane przez programistę - zawierają informacje (przesłanki) jak należy wykonywać polecenie,
- optymalizacja regułowa – statyczna – określenie przez system możliwych planów wykonania i wybór planu o najniższej cenie,
- optymalizacja kosztowa – dynamiczna – określenie przez system możliwych planów wykonania, oszacowanie kosztu w oparciu o wyliczone statystyki i wybór planu o najmniejszym koszcie.

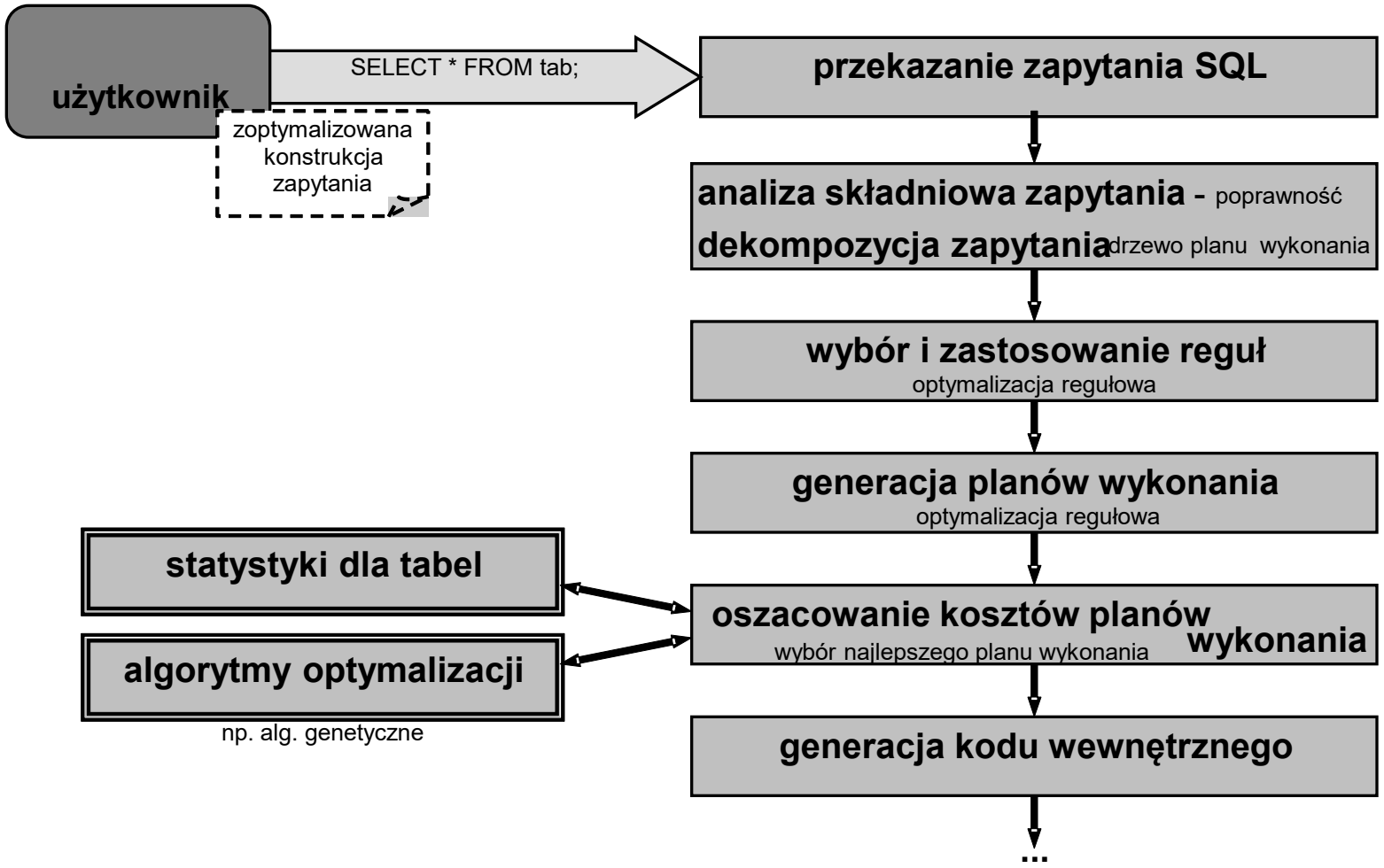
Optymalizacja zapytań SQL

Plan wykonania zapytania



Optymalizacja zapytań SQL

Plan wykonania zapytania z optymalizacją



Optymalizacja zapytań SQL

Przegląd rozwiązań optymalizacyjnych

Każdy z produktów bazodanowych ma własne rozwiązania optymalizacyjne:

- Oracle :
  - optymalizacja regułowa – starsze wersje baz danych – optymalizacja w oparciu o 20 określonych reguł,
  - optymalizacja kosztowa – wybór planu wykonania w oparciu o reguły i statystyki danych w tabelach,
  - możliwość definiowania wskazówek optymalizacyjnych przez programistę,
  - widoki zmaterializowane,
  - tworzenie indeksów na potrzeby zapytania,
  - własne rozszerzenia techniczne, SQL ...

- PostgreSQL :
  - analiza statystyczna tabel,
  - szacowanie kosztów wykonania zapytania,
  - wykorzystanie algorytmów genetycznych do optymalizacji zapytań ☺,
  - ...

- MySQL:
  - szacowanie kosztów wykonania zapytania,
  - zalecane wskazówki dla programistów,

- MS SQL – self-tuning,

- IBM DB2 – self-tuning,

Optymalizacja zapytań SQL

Algorytmy realizacji złączeń tabel

Złączenie dwóch tabel A i B:

- **nested loops - zagnieżdżone pętle** – dla każdego wiersza tabeli A przeglądana jest cała tabela B i wyszukiwany w niej są wiersze odpowiadające wierszowi z relacji A,
  - ⇨ małe rozmiary relacji, niewielka liczba wierszy spełniających warunki, skomplikowane warunki złączenia,

- **block nested loops – zagnieżdżone pętle z buforem** – z tabeli A brany jest pewien blok wierszy i dla tego bloku są sprawdzane wiersze w tabeli B – optymalizacja ma na celu zmniejszyć krotność odczytywania tabeli B z dysku

- **merge join (sort merge) - sortowane scalanie** – złączenie dwufazowe :
  - w pierwszej fazie w obu tabelach sortowane są wartości kolumn złączenia (jeśli nie ma indeksu),
  - w drugiej fazie dla każdego wiersza z tabeli A wyszukiwany jest pierwszy wiersz tabeli B odpowiadający warunkom złączenia – następnie pobierane są kolejne, aż do znalezienia pierwszego wiersza niespełniającego warunku złączenia – dalsza część tabeli B nie jest już przeszukiwana,

  - ⇨ duże rozmiary relacji, wybrane operatory złączenia, indeks w kolumnach złączenia,

- **hash join** - wykorzystanie funkcji mieszających :
  - w pierwszej fazie tworzone są tablice mieszające dla relacji zewnętrznej (wybierana jest ta, która ma mniej wierszy spełniających warunki),
  - łączenie relacji na podstawie grup o tej samej wartości funkcji mieszającej (dokładne sprawdzanie),

  - ⇨ tylko dla równozłączenia, dla dużej liczby wierszy bez indeksu na kolumnach złączenia - kiedy koszt tworzenia indeksu tymczasowego jest większy niż koszt tworzenia tabeli mieszającej,

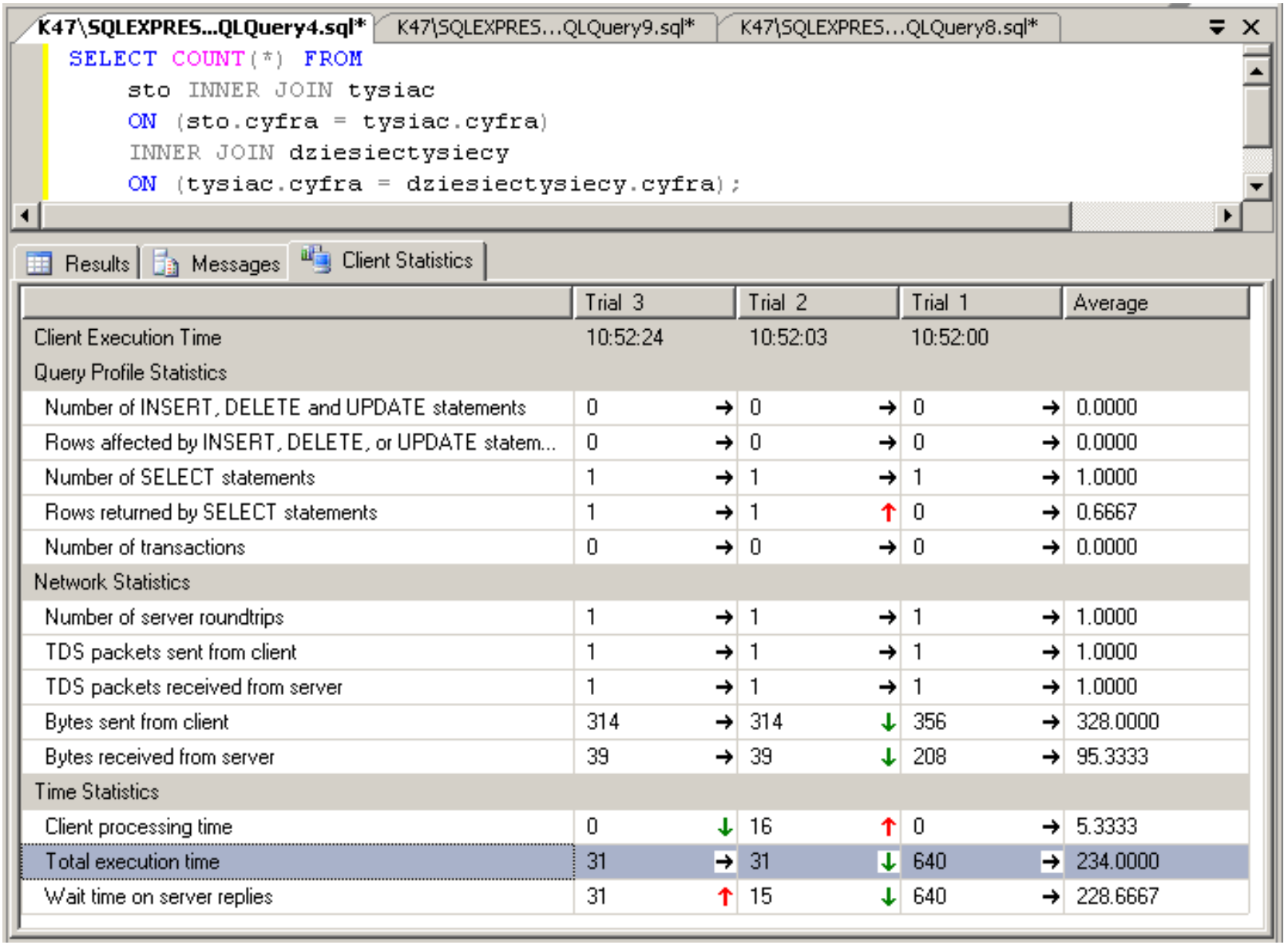
- **semi join** – półzłączenie – sprowadza się do sprawdzenia, czy odpowiednik wiersza tabeli sterującej pojawia się w kolumnie złączenia tabeli wewnętrznej – duplikaty… transformowalne z podzapytania nieskorelowanego

Złączanie większej liczby tabel:
- złączenie pary tabel a następnie złączenie wyniku z następną tabelą,
- wybór tabeli sterującej złączeniem – dająca najmniej rozwiązań, ($\rightarrow$ zmniejszyć liczbę przeszukiwań drugiej tabeli n-l, dla każdego wiersza wynikowego tabeli sterującej odczytywana jest cała tabela wewnętrzna $\rightarrow$ powinna być mniejsza),
- kolejność istotna – na koniec należy umieścić tabelę dającą najmniej rozwiązań,

jeśli jest podany warunek dotyczący jednej tabeli - system odrzuca wstępnie te wiersze, które nie spełniają warunku, aby nie obciążać niepotrzebnie pamięci.

Optymalizacja zapytań SQL

Bazy samostrojące się



	Trial 3	Trial 2	Trial 1	Average
Client Execution Time	10:52:24	10:52:03	10:52:00	
Query Profile Statistics				
Number of INSERT, DELETE and UPDATE statements	0 →	0 →	0 →	0.0000
Rows affected by INSERT, DELETE, or UPDATE statem...	0 →	0 →	0 →	0.0000
Number of SELECT statements	1 →	1 →	1 →	1.0000
Rows returned by SELECT statements	1 →	1 ↑	0 →	0.6667
Number of transactions	0 →	0 →	0 →	0.0000
Network Statistics				
Number of server roundtrips	1 →	1 →	1 →	1.0000
TDS packets sent from client	1 →	1 →	1 →	1.0000
TDS packets received from server	1 →	1 →	1 →	1.0000
Bytes sent from client	314 →	314 ↓	356 →	328.0000
Bytes received from server	39 →	39 ↓	208 →	95.3333
Time Statistics				
Client processing time	0 ↓	16 ↑	0 →	5.3333
Total execution time	31 →	31 ↓	640 →	234.0000
Wait time on server replies	31 ↑	15 ↓	640 →	228.6667

Optymalizacja zapytań SQL

Szacowanie kosztów zapytania

Dostępne statystyki dla zapytania – przykład ORACLE:

- recursive calls – liczba wywołań rekurencyjnych w ramach zapytania,
- db block gets – liczba odczytów aktualnego bloku,
- consistent gets – liczba spójnych odczytów dla komendy SELECT,
- physical reads – liczba bloków odczytanych z dysku,
- redo size – rozmiar generowanego powrotu (B),
- bytes sent via SQL*Net to client – komunikacja z terminalem SQL - wysyłanie,
- bytes received via SQL*Net from client - komunikacja z terminalem SQL - odbieranie,
- SQL*Net roundtrips to/from client – interakcja,
- sorts (memory) – liczba sortowań w pamięci,
- sorts (disk) – liczba sortowań na dysku,
- rows processed – liczba zwróconych wierszy

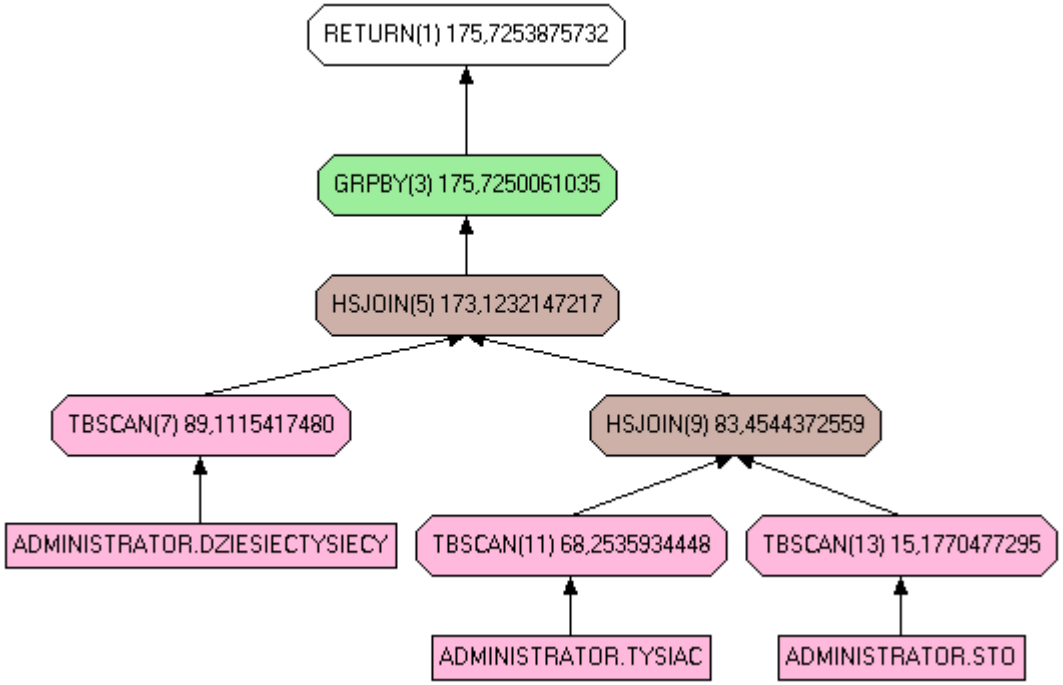
```
SQL> set autotrace on
SQL> SELECT COUNT(*) FROM ccc WHERE kolC IN (SELECT kolbC FROM bbb);

Execution Plan
----------------------------------------------------------
Plan hash value: 1817329480

-------------------------------------------------------------------------------------------
| Id  | Operation           | Name | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |      |     1 |    26 |       |  1922   (4)| 00:00:24 |
|   1 |  SORT AGGREGATE     |      |     1 |    26 |       |            |          |
|*  2 |   HASH JOIN SEMI    |      | 94399 |  2396K|  2312K|  1922   (4)| 00:00:24 |
|   3 |    TABLE ACCESS FULL| CCC  | 94399 |  1198K|       |    64   (5)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| BBB  |   921K|    11M|       |   622   (5)| 00:00:08 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("KOLC"="KOLBC")

Note
-----
   - dynamic sampling used for this statement

Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
       2424  consistent gets
          0  physical reads
          0  redo size
        411  bytes sent via SQL*Net to client
        384  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

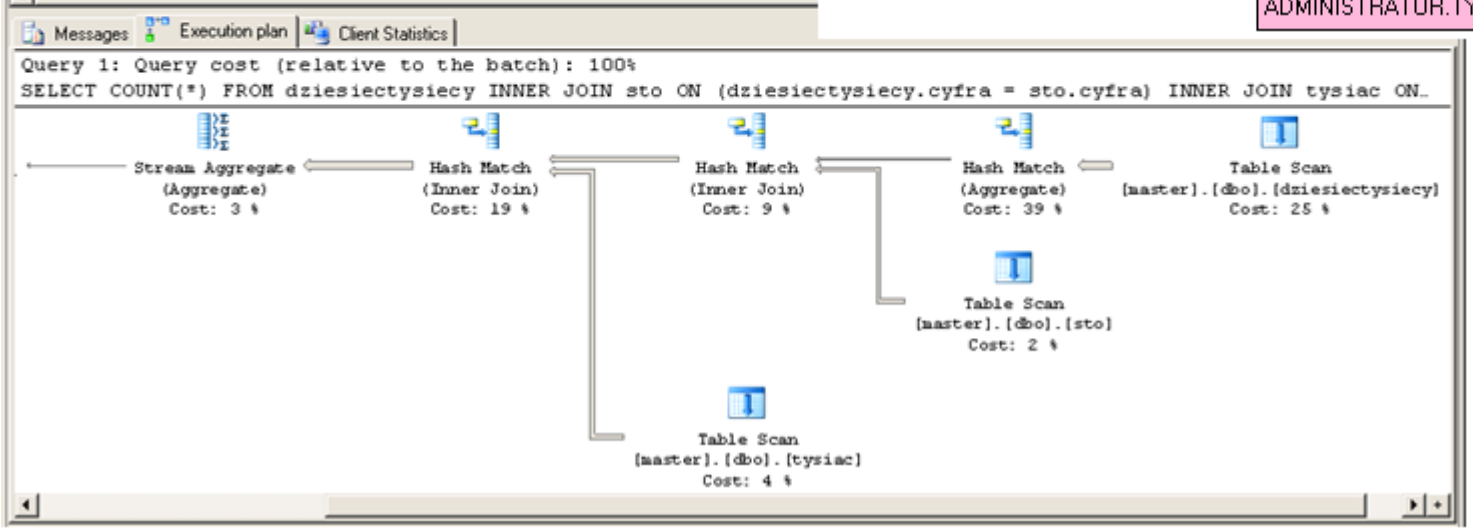
Optymalizacja zapytań SQL

Plany wykonania zapytania

```
SELECT COUNT(*) FROM
    sto INNER JOIN tysiac
    ON (sto.cyfra = tysiac.cyfra)
    INNER JOIN dziesiectysiecy
    ON (tysiac.cyfra = dziesiectysiecy.cyfra);
```

```
SELECT COUNT(*) FROM
    dziesiectysiecy INNER JOIN sto
    ON (dziesiectysiecy.cyfra = sto.cyfra)
    INNER JOIN  tysiac
    ON (sto.cyfra = tysiac.cyfra);
```





Optymalizacja zapytań SQL

Plany wykonania zapytania