



# Bazy Danych

## 4. Strojenie Baz Danych

Opracował: Maciej Penar

## Spis treści

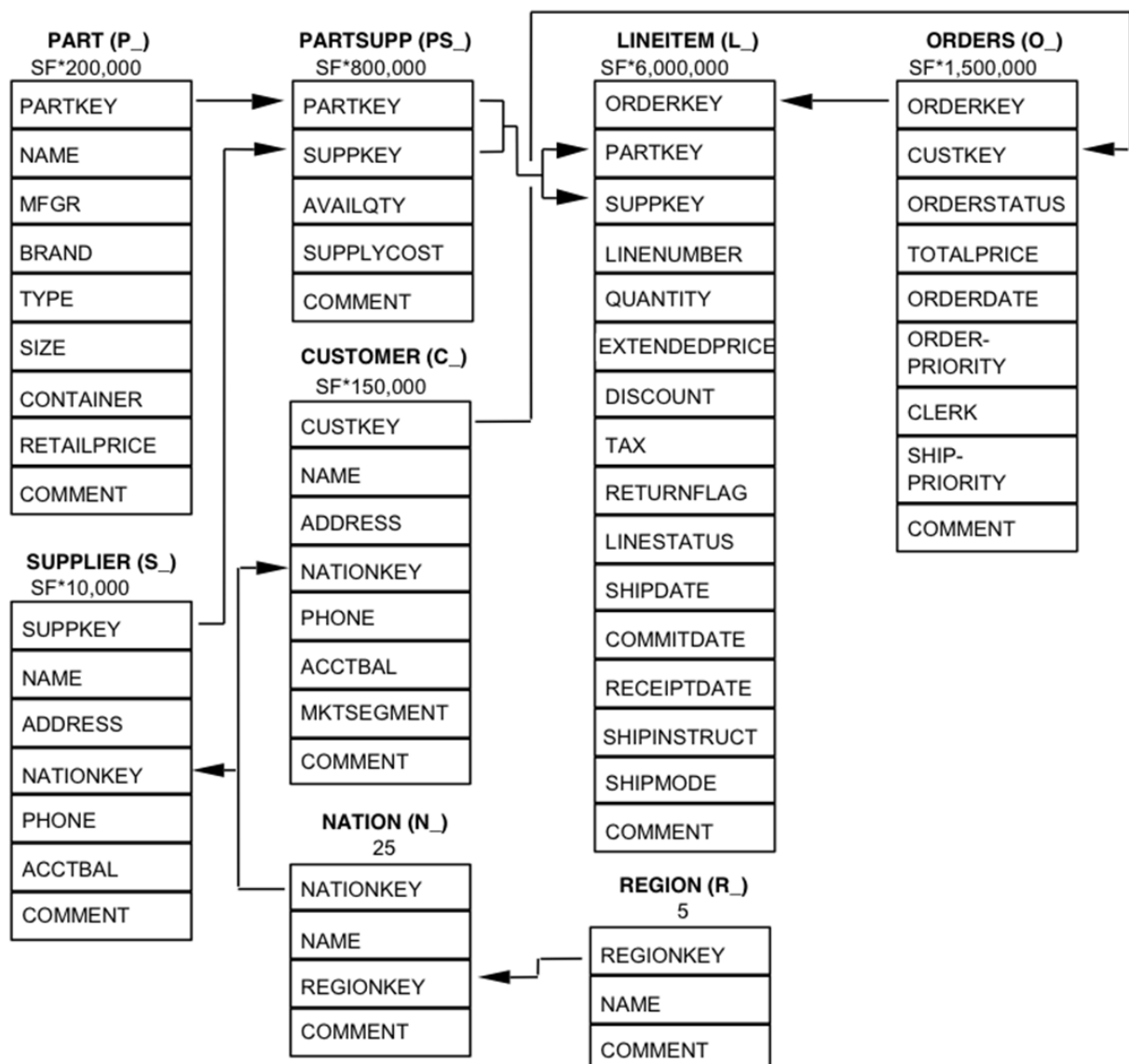
1. Linki .....	3
2. Schemat TCP-H.....	3
3. Krótkie info o Strojeniu Baz Danych.....	4
Kocyk .....	4
Kocyk, a Bazy Danych .....	5
Arsenał.....	5
Struktura tabeli: .....	5
Indeksy – pomocnicze struktury dostępowe: .....	5
Partycjonowanie tabel.....	6
Kompresja danych .....	6
Jak się zabrać do strojenia bazy danych .....	7
Tropy .....	8
4. (6-12 pkt) Strojenie Bazy Danych .....	9
(6-12 pkt) - Sprawozdanie .....	9
(0-6 pkt) - ogłoszenie .....	9

## 1. Linki

Przydatne linki:

- Oracle 12c: [link](#)
- Hammer DB: [link](#)
- Strona Transaction Processing Council: [link](#)
- Strona benchmark TPC-H: [link](#)
- Dokumentacja benchmark TPC-H 2.17.3: [link](#)
- Strojanie Baz Danych (wiki): [link](#)
- Przykładowa konkretyzacja zapytań z benchmarku TPC-H: [link](#)
- Przykładowa konkretyzacja zapytań z bechmarku TPC-H (Hive): [link](#)

## 2. Schemat TCP-H

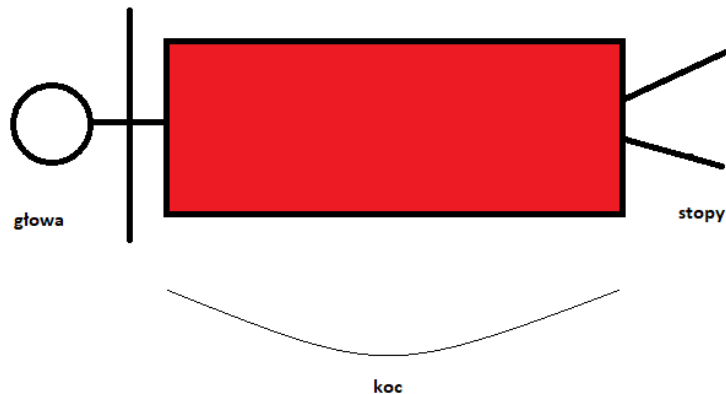


### 3. Krótkie info o Strojeniu Baz Danych

Strojenie baz danych (ang. Database Tuning / Query Tuning) to proces ujednolicenia wydajności bazy danych – najczęściej zapytań oraz komend DML (INSERT/DELETE/UPDATE).

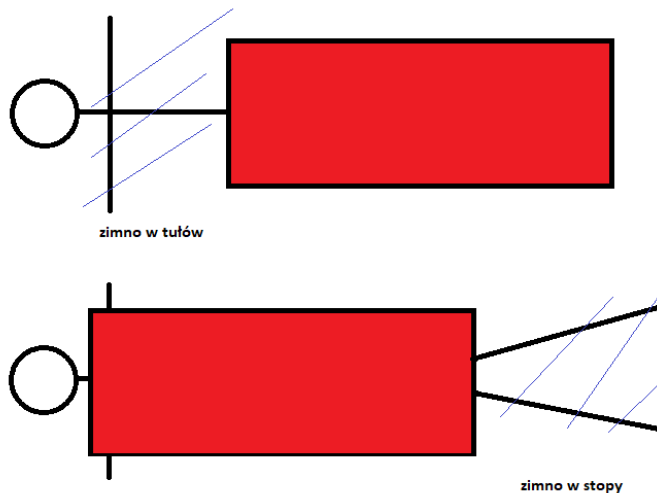
#### KOCYK

Strojenie bazy danych można porównać do problemu zimnej nocy i krótkiego kocyka. Mamy człowieka z krótkim kocem w nocy, tak jak na moim pięknym rysunku:



Mając ograniczony zasób – kocyk, możemy:

- Przesunąć kocyk na stopy – i będzie nam zimno w tułów, ale ciepło w stopy
- Przesunąć kocyk na brzuch – i będzie nam zimno w stopy, ale ciepło w brzuch



W zależności od tego jaki efekt chcemy osiągnąć – należy przesunąć kocyk w odpowiednie miejsce. Nie musi być to pozycja skrajna – być może jesteśmy w stanie stolerować zimne palce u stóp, bo wolimy mieć ciepły brzusek.

Analogicznie do przypadku kocyka – Bazę Danych w radykalnym scenariuszu możemy dostroić:

- Albo do operacji DML:
  - czyli mieć tanie operacje INSERT – skrajnie  $O(1)$
  - ale każdy SELECT/UPDATE/DELETE – kosztem  $O(n)$
- Albo do operacji odczytu:
  - Czyli SELECT/UPDATE/DELETE – kosztem  $O(\log N)$  [punktowy odczyt]
  - INSERT – kosztem  $O(\log N)$  – dla każdego indeksu

Podsumowując: podczas strojenia Bazy Danych **nie można** mieć ciasta i zjeść ciasta.

## ARSENAŁ

Do szukania szeroko rozumianej wydajności w Bazach Danych mogą służyć nam:

### STRUKTURA TABELI:

Struktura	Link	Opis
Pliki stertowe	<a href="#">link</a>	Domyślna struktura w ORACLE DB. Dla SELECT'ów punkt wejścia to wykonanie przeglądu całej przestrzeni danych. Tanie INSERT'y.
Indeks klastrowany	<a href="#">link</a>	Rekordy w tej strukturze są fizycznie posortowane wg. klucza klastrowującego. INSERT'y wymagają znalezienia pozycji na której wsadzamy nowy rekord. SELECTy na warunku klucza klastrowującego (lub jego części) są tanie – tak samo zapytania zakresowe.

### INDEKSY – POMOCNICZE STRUKTURY DOSTĘPWE:

Indeksy służą do przyspieszania niektórych zapytań, kosztem złożoności operacji INSERT. Idealną sytuacją jest gdy tabela bazowa w żaden sposób nie jest angażowana do wykonania zapytania.

Do utworzenia indeksu służy wyrażenie CREATE INDEX:

```
CREATE INDEX [nazwa] ON [tabela] ([kolumna_1], ..., [kolumna_2]);
```

Dobre praktyki związane z indeksami to:

- Indeksy pokrywające (tzw. Covering indexes - [link](#)) – czyli takie które zawierają dodatkowe kolumny – indeksy rosną, wymagają więcej miejsca, ale często nie trzeba dotykać tabeli bazowej (i najczęściej wykonywać swap na buforach bazy danych)

- Nie tworzyć dużo indeksów (książką Bill Karwin – SQL Antipatterns, rozdział „Index Shotgun”) – optymalizator może zgłupieć, bazy danych puchną, wolniejsze inserty. Myśleć o tym że K indeksów to K-razy wolniejszy insert.
- W książce Billa Karwin – SQL Antipatterns, rozdział „Index Shotgun”- znajduje się metodologia dostrajania indeksów tzw. MENTOR (Measure, Explain, Nominate, Test, Optimize, Rebuild), innymi słowy:
  - Najpierw mierzymy wydajność dostrajanych zapytań
  - Potem sprawdzamy plany wykonania zapytania
  - Potem wybieramy atrybuty na których postawimy indeksy
  - Stawiamy i testujemy
  - Optymalizujemy pod kątem tego jak będziemy z tabeli korzystać
  - Na koniec przebudowujemy indeksy ([link](#))

---

#### PARTYCJONOWANIE TABEL

W ramach ciekawostki:

Bardzo skuteczna (i niedoceniana) forma zwiększania wydajności Bazy Danych – zarówno INSERTów jak i SELECT’ów – kosztem małego narzutu na każde wchodzące zapytanie do Bazy Danych. **Jest to tak efektywna forma że jest dostępna zazwyczaj w wersji Enterprise.**

Istnieją dwie formy partycjonowania danych:

- W poziomie – czyli tabelę dzielimy wierszami
- W pionie – czyli dzielimy tabelę kolumnami

---

#### KOMPRESJA DANYCH

W ramach ciekawostki:

Kompresja danych = więcej danych na tej samej przestrzeni.

Więcej danych na tej samej przestrzeni = Więcej danych w buforach

Więcej danych w buforach = mniej swap’owania

Mniej swapowania = mniejsze średnie czasy odpowiedzi

**Kompresja w bazach danych to darmowe przyśpieszenie.... Dostępne w wersji Enterprise.**

## JAK SIĘ ZABRAĆ DO STROJENIA BAZY DANYCH

Założmy że mamy tabelę:

```
CREATE TABLE TEST_INDEX(ID INT, TEXT VARCHAR(100), SAMPL DATE);
```

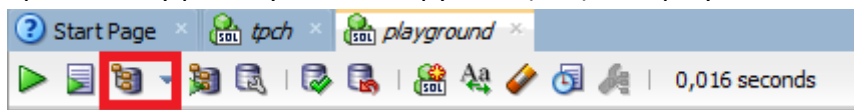
I chcemy dostroić zapytanie:

```
SELECT TEXT FROM TEST_INDEX WHERE ID = ?;
```

1. Przede wszystkim upewniamy się że bufor cache Bazy Danych są puste – to zapytanie wykonujemy **zawsze** przed SELECT'em który chcemy dostroić:

```
ALTER SYSTEM FLUSH BUFFER_CACHE;
```

2. Wykonujemy **skonkretyzowane** zapytanie które dostrajamy i zapisujemy czas
3. Sprawdzamy plan wykonania zapytania (F10) albo przycisk:



4. Plan wygląda mniej więcej:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	3
TABLE ACCESS (FULL)	TEST_INDEX	1	3
Filter Predicate			
ID=1			
Other XML			

Spróbujmy go zinterpretować:

**TABLE ACCESS (FULL)** – czyli zapytanie czyta całą tabelę

**Filter Predicate** – czyli predykat zapytania

Fragment **TABLE ACCESS (FULL)** oznacza kiepską wydajność – inne „czerwone” flagi to algorytmy oparte o **HASH** i tworzenie tymczasowych tabel.

Koszt zapytania został oceniony na 3 (tabela jest bardzo mała).

5. Proponujemy indeks:

```
CREATE INDEX my_index ON TEST_INDEX(ID ASC);
```

6. Sprawdzamy czas i plan zapytania:

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	2
TABLE ACCESS (BY INDEX ROWID BATCHED)	TEST_INDEX	1	2
INDEX (RANGE SCAN)	MY_INDEX	1	1
Filter Predicate			
ID=1			

Dostajemy plan który wykonuje:

**INDEX(RANGE SCAN)** – czyli czyta cały indeks

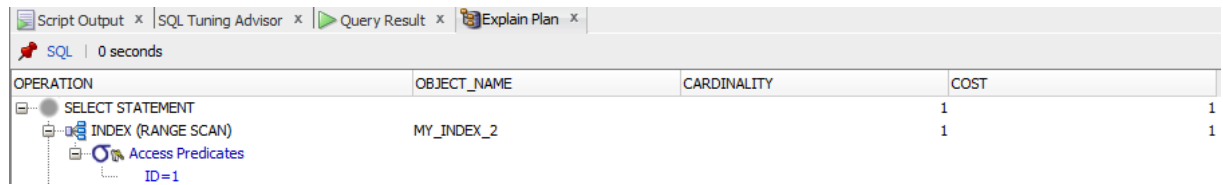
**TABLE ACCESS(BY INDEX ROWID)** – odczytuje zawartość tabeli na podstawie wartości otrzymanych z indeksu

Koszt to 2. Hm...

7. Usuwaamy poprzedni indeks, proponujemy nowy:

**CREATE INDEX my\_index ON TEST\_INDEX(ID ASC, TEXT ASC);**

8. Sprawdźmy czas i plan:



The screenshot shows the SQL Explain Plan interface. The top bar includes tabs for 'Script Output', 'SQL Tuning Advisor', 'Query Result', and 'Explain Plan'. Below the tabs, a table displays the execution plan details. The table has four columns: OPERATION, OBJECT\_NAME, CARDINALITY, and COST. The first row shows 'SELECT STATEMENT' with a cardinality of 1 and a cost of 1. The second row shows 'INDEX (RANGE SCAN)' on 'MY\_INDEX\_2' with a cardinality of 1 and a cost of 1. Below this, 'Access Predicates' are listed, including 'ID=1'.

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	1
INDEX (RANGE SCAN)	MY_INDEX_2	1	1
Access Predicates			
ID=1			

9. Zapytanie wykonuje tylko **INDEX(RANGE SCAN)**, żadna inna operacja nie jest wymagana – ponieważ indeks w pełni pokrywa wymogi zapytania (odczyt TEXT, predykat po ID). Koszt zapytania: 1 (bo mniej się nie da).

#### TROPY

- Indeksy na kluczach obcych
- Przepisanie tabeli na indeks klastrowy (Index-Organized-Table)
- Sprawdzić czy typy danych na indeksie się zgadzają
- Różne typy indeksów:
  - Indeksy bitmapowe
  - Indeksy funkcyjne ([link](#))



## 4. (6-12 pkt) Strojenie Bazy Danych

### (6-12 PKT) - SPRAWOZDANIE

Znaleźć definicję zapytań z dokumentu TPC-H ([link](#)). Definicja jest w sekcji (*Functional Query Definition*). Zwrócić uwagę na adnotacje/komentarze typu: „Return the first X selected rows”. Być może z komentarzy wynika że zapytania **nie** da się dostroić. Zapytania do dostrojenia to:

1. Pricing Summary Report Query (Q1)
2. Minimum Cost Supplier Query (Q2)
3. Shipping Priority Query (Q3)
4. Order Priority Checking Query (Q4)
5. Local Supplier Volume Query (Q5)
6. Forecasting Revenue Change Query (Q6)

Napisać sprawozdanie z zadania (plik 4.Strojenie Baz Danych – Sprawozdanie.docx).

Sprawozdanie zawiera:

1. Skrypty
  - Tworzące indeksy / tabele na bazie TPC-H
  - Usuwające indeksy na bazie TPC-H
2. Konkretyzację strojonych zapytań zgodną z dokumentacją
3. Pomiar wydajności:
  - Przed utworzeniem indeksów
  - Po utworzeniu indeksów
  - \* (2 bonusowe pkt) Opis różnic pomiędzy planami wykonania zapytań

#### Przypomnienie

Pamiętać że każdy pomiar jest bezwartościowy jeśli przed wykonaniem zapytania nie zostały opróżnione cache Bazy Danych, czyli **każdorazowo** powinna być wykonana linijka SQL:

```
alter system flush buffer_cache;
```

#### Przypomnienie 2

Nie strzelać indeksami na ślepo i nie tworzyć ich za dużo

### (0-6 PKT) - OGŁOSZENIE

Jest ryzyko że chciałbym zobaczyć jak działacie pod presją i zadanie powtórzmy na laboratoriach dla innych zapytań np.:

1. Volume Shipping Query (Q7)
2. National Market Share Query (Q8)