Optymalizacja wydajności SZBD

1. Optymalizacja wydajności systemu bazodanowego

Wydajność SZBD określana jest najczęściej za pomocą następujących parametrów:

- liczby operacji przeprowadzanych na sekundę, czyli **przepustowości systemu**;
- czasu potrzebnego na zwrócenie wyniku, czyli **czasu reakcji systemu**.

Przepustowość systemu zależy od:

- ✓ zasobów systemu komputerowego serwera
  - pamięci operacyjnej
  - mocy obliczeniowej
  - wydajności systemu wejścia-wyjścia
  - przepustowości lokalnej sieci

Czas reakcji systemu zależy od:

- ✓ logicznej i fizycznej struktury baz danych
- ✓ aplikacji klienckich

Wpływ na wydajność systemu bazodanowego ma również stosowana przez serwer i klientów metoda pobierania i przetwarzania danych. Mała wydajność jednego z podanych składników może negatywnie wpływać na wyniki monitorowania wydajności pozostałych składników.

Monitorowanie wydajności systemu należy przeprowadzać, wg kolejności:

1. Sprawdzić wydajność komputera
2. Prześledzić prace systemu operacyjnego
3. Sprawdzić wykorzystanie zasobów systemowych przez serwer baz danych
4. Przeanalizować strukturę bazy danych.
5. Monitorowanie wydajności aplikacji klienckiej

2. Optymalizacja bazy danych

Najczęstszymi przyczynami małej efektywności pracy z bazą danych są:

- ✓ Nieefektywne projekty baz danych
- ✓ Źle utworzone indeksy
- ✓ Źle skonstruowane zapytania
- ✓ Nieoptymalna struktura przechowywania danych
- ✓ Fragmentacja bazy danych

Większość baz pracuje na ogromnej ilości danych, dlatego powinny one być projektowane tak, aby jak najmniej obciążały serwer, oraz aby nie zajmowały zbyt dużo miejsca na dysku. Wykonanie optymalizacji bazy danych może pomóc w zwiększeniu jej wydajności oraz zmniejszeniu jej rozmiarów. Podstawowymi działaniami które powinny zostać wykonane są:

- ✓ Normalizacja bazy danych
- ✓ Prawidłowe tworzenie indeksów
- ✓ Optymalizacja zapytań i operacji wykonywanych w bazie danych
- ✓ Defragmentacja bazy danych

Normalizacja

Definiuje zasady poprawnej budowy schematu bazy danych. Chroni przed powielaniem tej samej informacji w tabelach.

W celu zwiększenia wydajności bazy danych należy zadbać , aby schemat bazy danych spełniał wymogi normalizacji.

Indeksy

Indeksy w bazie danych pozwalają na szybsze wyszukiwanie danych, co powoduje znacznie szybsze wykonywanie zapytań do bazy.

Zalety stosowania indeksów:

- ✓ Ograniczenie ilości danych odczytywanych z bazy
- ✓ Przyspieszenie wyszukiwania informacji
- ✓ Sortowanie danych

Wada indeksów:

- ✓ Zajmują na dysku dodatkowe miejsce
- ✓ Muszą być na bieżąco aktualizowane (każde wstawienie, usunięcie lub aktualizacja danych w tabeli wiąże się z aktualizacją wszystkich zdefiniowanych dla niej indeksów)

Zastosowanie indeksów przyśpiesza odczyt ale spowalnia modyfikowanie danych ponieważ wymaga aktualizowania informacji w indeksach.

Przykład:

- ✓ **Utworzone są dwa indeksy.**

```
create index i_imie on czytelnicy(imie);
create index i_nazwisko on czytelnicy(nazwisko);
```

- ✓ **Utworzony został indeks**

```
create index i_imie_nazw on czytelnicy(imie, nazwisko);
```

Po wykonaniu poniższego zapytania:

```
select *from czytelnicy where imie="Jan" and nazwisko= "Kowalski";
```

W pierwszym przypadku wykonane zostaną następujące czynności:

- ✓ Wyszukane zostaną rekordy dla imie=”Jan”
- ✓ Wyszukane zostaną rekordy dla nazwisko=”Kowalski”
- ✓ Zostanie wybrana wspólna część zbiorów rekordów z pierwszego i drugiego wyszukiwania i zwrócona jako wynik zapytania

W drugim przypadku potrzebne dane zostaną wyszukane w jednym kroku, w którym równocześnie sprawdzone zostaną wartości w polach **imie** i **nazwisko**.

Ćwiczenie:

Wykonaj kolejno poniższe polecenia:

a) załóż indeks na kolumnę **rocznik**, a drugi na kolumnę przebieg tabeli **Auta**. Nazwij je odpowiednio **indeks_rocznik** oraz **indeks_przebieg**. Następnie utwórz zapytanie wyszukujące wszystkie samochody z rocznika 2006 o przebiegu poniżej 150 000 km i wyświetl kolumny: **id, rocznik, przebieg, wartość, silnik, skrzynia**.
b) załóż indeks nałożony jednocześnie na dwie kolumny **rocznik** i **przebieg** tabeli **Auta**. Nazwij go **indeks_rocznik_przebieg**. Następnie wykonaj zapytanie, jak w podpunkcie a.
c) usuń indeks założony na kolumnie **rocznik**.

Zapytania wykonywane bez indeksów zmuszają silnik bazy danych do sekwencyjnego przeszukiwania wszystkich wierszy w tabeli.

Reindeksacja

Indeksy po wykonaniu operacji modyfikujących (INSERT, UPDATE, DELETE) muszą być aktualizowane, aby spełnić swoją role.

Reindeksacja (ang. reindex) lub inaczej **odbudowa indeksów** jest procesem porządkującym plik indeksowy przez usunięcie nieaktualnych wpisów. Wykonanie reindeksacji zaleca się również po dodaniu lub usunięciu większej ilości danych.

W bazie MySQL polecenie reindeksacji przyjmuje następującą postać:

```
analyze table nazwa_tabeli;
```

Odbudowa indeksów jest jednym z elementów optymalizacji pracy bazy danych. Zaleca się regularne przeprowadzanie odbudowy indeksów.

3. Optymalizacja zapytań

- ✓ Przez optymalizację zapytań rozumiemy konstruowanie poprawnych zapytań SQL, tak aby ich czas wykonywania był jak najkrótszy.
- ✓ Aby przyśpieszyć wykonywanie zapytań, należy je konstruować tak, aby były wykonywane na jak najmniejszej ilości danych. Zmniejszenie ilości przetwarzanych danych zmniejsza ilość potrzebnych zasobów oraz zwiększa efektywność działania indeksów

Sposoby:

- ✓ ograniczenie liczby kolumn do wyświetlenia w instrukcji **select**
- ✓ filtrowanie danych poprzez zastosowanie klauzuli **WHERE**
- ✓ nie powinno się stosować porządkowania danych, jeżeli nie jest to konieczne (**order by**)
- ✓ trzeba unikać zagnieżdżonych zapytań i klauzuli **GROUP BY**

Do takich samych wyników zapytań możemy dochodzić różnymi drogami. Nie wszystkie z nich wymagają takiej samej liczby operacji.

Przykład:

Dla bazy **KOMIS** rozważmy następujące zadania:

Przykład 1

```
SELECT *FROM samochody where exists (select *from modele, samochody where modele.id=samochody.model and modele.nazwa like 'F%');
```

Przykład 2

```
SELECT  *FROM samochody where model in(select id from modele where nazwa like 'F%');
```

Po wykonaniu tych zadań otrzymamy identyczny rezultat, ale każdy z nich jest osiągany innym sposobem.

1. Serwer wykona pełne skanowanie tabeli **samochody** i dla każdego wiersza będzie szukał **modeli** o **nazwie** zaczynającej się na literę **F**, powiązanych kluczem obcym z analizowanym samochodem.

2. Serwer znajdzie wszystkie **modele** o określonej **nazwie**. Następnie będzie przeglądał tabelę **samochody**, używając kluczy obcych, i będzie próbował dopasować **samochód** do jednego ze znalezionych **modeli**.

Drugie zapytanie wymaga mniejszych zasobów serwera. Będzie również szybciej wykonane ponieważ tylko raz system wyszuka konkretne modele.

Pamiętać należy, że choć podstawy teoretyczne algorytmów optymalizacyjnych są ogólnie znane, jednak szczegóły implementacyjne są zwykle tajemnicą producenta i ich efekty mogę się znacznie różnić w zależności od SZBD.

W celu oszacowania złożoności, a tym samym kosztu wykonania zapytania, możemy wykorzystać polecenie **EXPLAIN**. Jego użycie polega na poprzedzeniu badanego zapytania tym słowem kluczowym.

Przykład 1

```
EXPLAIN SELECT *FROM samochody where exists (select *from modele, samochody where modele.id=samochody.model and modele.nazwa like 'F%');
```

Przykład 2

```
EXPLAIN SELECT  *FROM samochody where model in(select id from modele where nazwa like 'F%');
```

Plan zapytania jest prezentowany w postaci drzewa.

Polecenia na węzłach (ang. node) są wykonywane od korzenia (ang.root) aż do liści (ang. leaf node) czyli ostatnich poleceń. Często do wykonania jednej czynności konieczne jest wykonanie (czasem wielokrotne) jej potomków (ang. child node).

Konstruując plan zapytania baza dokonuje wstępnego oszacowania, ile mogą trwać poszczególne operacje. Jest to możliwe, gdyż każda baza przechowuje rozbudowane statystyki dotyczące m.in. liczby wierszy w tabeli, unikatowości danych w określonej kolumnie itp.

W celu usprawnienia wyszukiwania danych, należy również analizować i porównywać plany zapytań.

Wykrycie długotrwałych czynności powinno zmusić do zastanowienia się nad inną konstrukcją zapytania. W przypadku dużych tabel należy unikać:

- Pełnego przeszukiwania (ang. full table scan) czyli przeszukiwanie całej tabeli wiersz po wierszu
- Błędnej kolejności złączeń tabel, gdy jako pierwsza jest przeglądana większa z nich
- Umieszczania filtrów zbyt nisko w drzewie planu, co powoduje uruchomienie operacji na zbyt dużej liczbie wierszy
- Błędnego stosowania zagnieżdżonej pętli (ang. nested loop), czyli pętli, która musi się wykonać określoną liczbę razy, niezależnie od tego, czy będzie to potrzebne (np. gdy przeszukiwane wiersze znajdziemy w połowie wykonywania pętli)

Ćwiczenie:

Wykonaj oszacowanie zależności zapytania (ile wierszy z poszczególnych tabel serwer musi przejrzeć, aby otrzymać wyniki:

```
select *from samochody inner join modele on samochody.model=modele.id inner join producenci on modele.producent=producenci.id;
```

4**. Defragmentacja bazy danych**

Fragmentacja bazy danych - jest zjawiskiem zachodzącym przy usuwaniu i dodawaniu rekordów do bazy. Polega ona na występowaniu niewykorzystanych (pustych) obszarów wewnątrz plików tworzących bazę.

Problem fragmentacji objawia się:

- brakiem zmian wielkości fizycznych plików przechowujących dane z bazy na dysku podczas usuwania rekordów,
- fragmentacja obniża wydajność operacji wykonywanych przez silnik bazy danych

System MySQL

- ✓ automatycznie przeciwdziała fragmentacji bazy danych decydując samodzielnie, w którym miejscu dodać nowe wiersze np. na końcu tabeli lub w wolnej przestrzeni po usuniętych wierszach.
- ✓ może także co pewien czas wykonywać procedury defragmentacji danych. Dzięki nim sprawniej przeszukiwać bazę oraz zmniejszyć rozmiar danych na dysku twardym.
- ✓ automatyczne procedury defragmentacji są uruchamiane przy każdej operacji tworzenia kopii zapasowej bazy danych.

Mechanizmy związane z defragmentacją bazy danych dostępne dla administratora systemu, to możliwość:

- ✓ wywołania automatycznej defragmentacji poprzez uruchomienie procedur tworzenia kopii zapasowych
- ✓ użycia dedykowanej funkcji **OPTYMIZE TABLE**

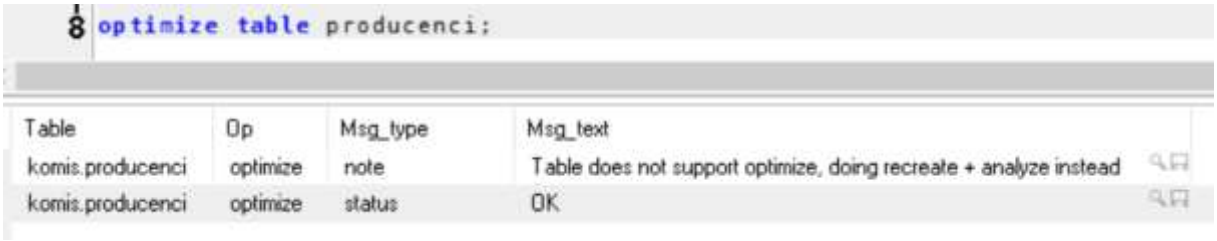
Wykonanie polecenia **OPTYMIZE TABLE** obejmuje defragmentację:

- ✓ reorganizację fizycznego magazynu danych,
- ✓ poprawę wydajności operacji wej-wyj przy dostępie do tabel,
- ✓ przywrócenie nieużywanej przestrzeni dyskowej,
- ✓ reindeksację i reorganizację tabel

Zalecane jest cykliczne wywoływanie operacji **OPTYMIZE TABLE.**

Przykład:

Wykonaj defragmentację tabeli Producenci bazy KOMIS



System wyświetlił komunikat o braku możliwości wykonania optymalizacji dla tej tabeli. Zamiast tego wykona jej ponowne utworzenie (z przepisaniem danych), czyli defragmentację oraz analizę rekordów.

Zadania do samodzielnego wykonania (baza danych KOMIS):

1. Wykonaj reindeksację bazy danych.
2. Sprawdź rozmiary bazy danych na dysku przed oczyszczaniem bazy i porównaj go z rozmiarem bazy po jej oczyszczeniu.
3. Utwórz dwa zapytania łączące dane z dwóch dowolnych tabel z bazy, których wynik jest identyczny, ale plany zapytań są różne
4. W bazie danych utwórz dwa różne zapytania, których plany i wynik są takie same
5. Wykonaj dowolne zapytanie wybierające dane z co najmniej dwóch tabel. Następnie wykonaj 100 000 razy dodanie i usunięcie pojedynczego rekordu z jednej z wybranych tabel. Porównaj czas wykonania identycznego zapytania po wymienionych operacjach. Wykonaj oczyszczanie bazy i ponownie określ czas wykonania zapytania. Wyniki przedstaw w tabeli.
6. Dla każdego z poniższych zapytań w bazie danych przygotuj zrzut ekranu z planem zapytania. Wykonane plany zapytań przedstaw w tabeli i porównaj. Wybierz najlepsze z zapytań i uzasadnij swoją odpowiedź.

   Zapytanie 1

   select ***from** samochody **where exists** (**select** ***from** modele **join** producenci **on** (modele.producent=producenci.id) **where** modele.id=samochody.model **and** producenci.nazwa **like** 'F%');

   Zapytanie 2

   select ***from** samochody **join** modele **on** samochody.model=modele.id **join** producenci **on** (modele.producent=producenci.id) **where** producenci.nazwa **like** 'F%';

   Zapytanie 3

   select *from samochody **where** model b(**select** id **from** modele **where** modele.producent **in**(**select** producenci.id **from** producenci **where** nazwa **like** 'F%'));