Sprawozdanie

Kamil Urbanowski

Michał Szymacha

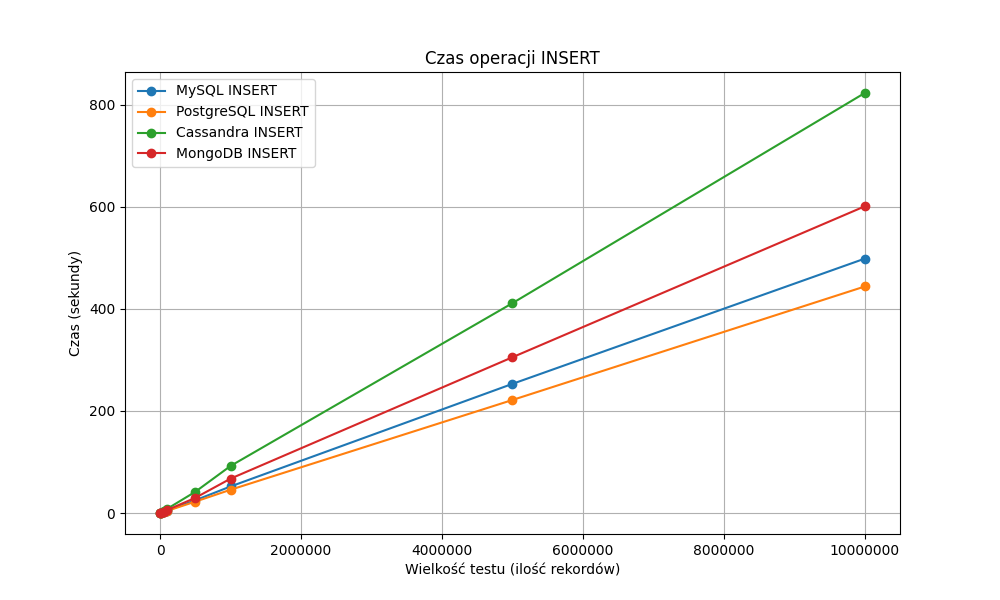
Jakub Święs

# Charakterystyka porównywanych systemów

* **MySQL** – relacyjna baza danych SQL, znana z prostoty i popularności w aplikacjach webowych. Używa silnika InnoDB (domyślnie) opartego na strukturze B-drzewa, co zapewnia szybkie operacje odczytu/zapisu po kluczu głównym. MySQL jest wydajny w prostych transakcjach OLTP, lecz oferuje nieco mniej zaawansowane funkcje SQL niż PostgreSQL.
* **PostgreSQL** – zaawansowana relacyjna baza danych SQL, ceniona za zgodność ze standardami i bogate możliwości (transakcje, procedury, indeksy specjalne). Cechuje się rozbudowanym optymalizatorem zapytań i wsparciem dla złożonych operacji (np. zapytania analityczne, JSON, CTE). W złożonych zapytaniach często przewyższa wydajnością MySQL, dobrze skalując się pionowo (wykorzystanie wielu rdzeni, duże ilości RAM).
* **MongoDB** – nierelacyjna baza typu NoSQL (dokumentowa), przechowująca dane w formacie JSON (BSON). Zapewnia elastyczny schemat (brak sztywnej struktury tabel), co ułatwia rozwój aplikacji ze zmiennymi danymi. MongoDB oferuje indeksy i mechanizm agregacji danych, zapewnia wysoką wydajność przy dużej liczbie prostych zapisów i odczytów, choć brak tradycyjnych transakcji wielowersowych i JOIN może utrudniać złożone zapytania.
* **Cassandra** – rozproszona baza NoSQL oparta o model kolumnowy (inspirowana Google Bigtable). Zaprojektowana do obsługi bardzo dużych wolumenów danych rozproszonych na wielu węzłach, z naciskiem na wysoką dostępność i szybkość zapisu. Wykorzystuje model *eventual consistency* (spójność ostateczna) i zapis oparty o struktury LSM. Cassandra świetnie radzi sobie z szybkim zapisem skalowanym horyzontalnie, jednak nie wspiera dobrze złożonych zapytań *ad-hoc* (brak JOIN, ograniczone agregacje), co może wpływać na wydajność w takich scenariuszach.

# Wyniki testów wydajności

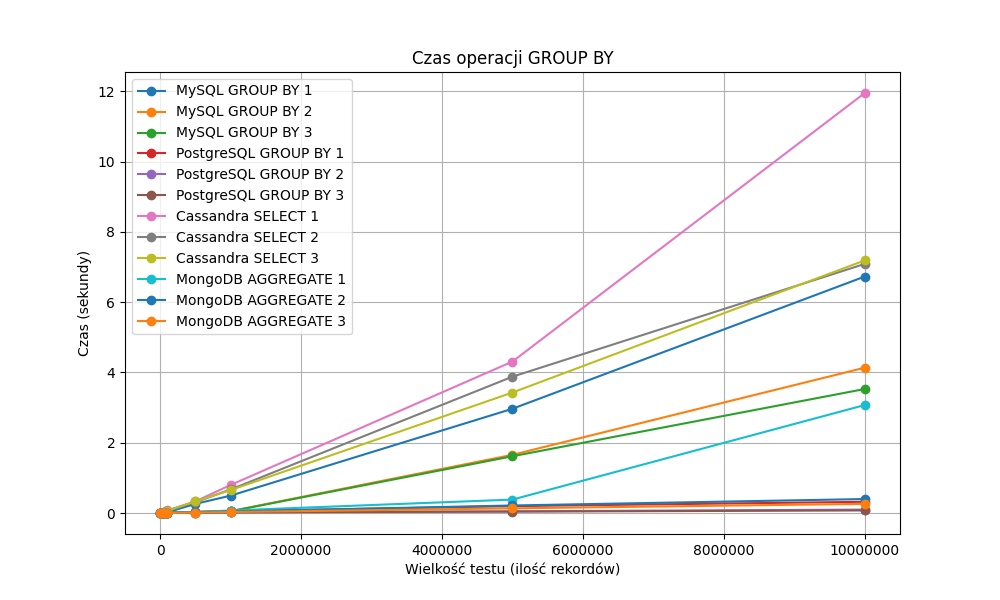
## Wstawianie danych (operacje INSERT)



Rysunek 1 Czas operacji INSERT w funkcji liczby wstawianych rekordów dla MySQL, PostgreSQL, MongoDB i Cassandry.

Badanie wydajności operacji dodawania rekordów ujawniło niemal liniowy wzrost czasu wraz z rozmiarem danych (liczbą wstawionych rekordów) dla wszystkich systemów. Relacyjne bazy danych okazały się nieco szybsze przy insercji od baz NoSQL: PostgreSQL osiągnął najniższe czasy dodawania danych, minimalnie wyprzedzając MySQL na każdym poziomie testu. Na przykład dla 10 milionów wierszy **INSERT** w PostgreSQL trwał ~443 s, podczas gdy w MySQL ~498 s. MongoDB uzyskał czasy pośrednie (ok. 600 s dla 10 mln), a Cassandra była najwolniejsza – ok. 823 s dla 10 mln rekordów. Różnice te uwidaczniają się szczególnie przy większych wolumenach danych; przy małych tabelach (rzędu tysięcy rekordów) wszystkie bazy wykonywały INSERT-y w ułamkach sekund (rozbieżności rzędu milisekund). Ogólny trend wskazuje, że PostgreSQL i MySQL lepiej radzą sobie z masowym importem danych na pojedynczym węźle niż MongoDB i Cassandra.

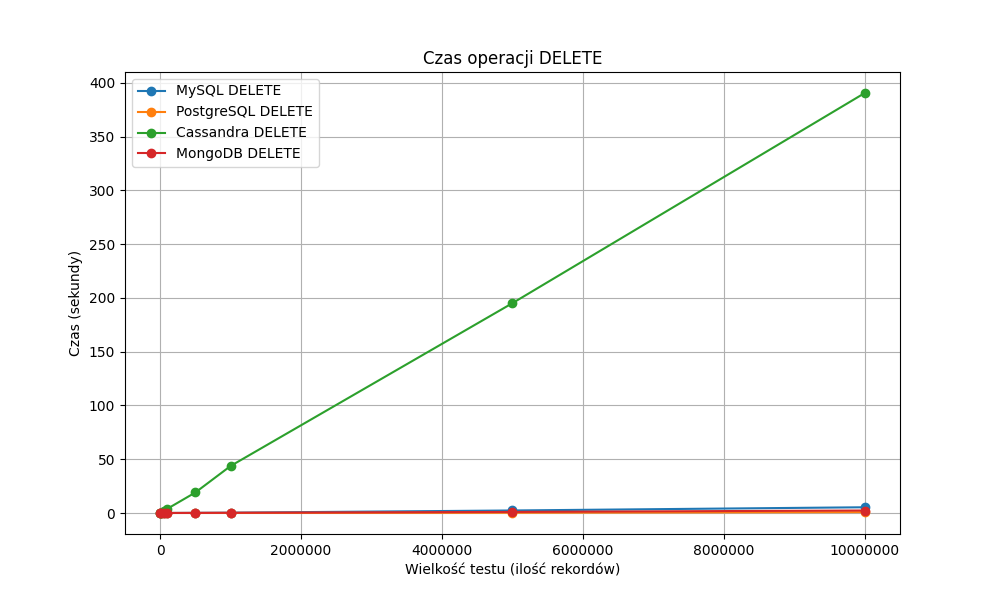
## Zapytania agregujące (SELECT z GROUP BY)



Rysunek 2 Czas wykonywania zapytań SELECT z operacją grupowania (GROUP BY) w funkcji rozmiaru danych.

Test zapytań agregujących (zliczanie rekordów z grupowaniem po jednym lub kilku polach) wykazał znaczące różnice między bazami. **PostgreSQL** zdecydowanie przewyższył pozostałe systemy – nawet dla największego zbioru (10 mln rekordów) odpowiedzi z GROUP BY pojawiały się poniżej 0,4 s, podczas gdy **MySQL** potrzebował od kilku do ~6,7 s (w zależności od zapytania). **MongoDB** również poradził sobie dobrze z agregacją, uzyskując czasy rzędu ułamków sekundy do ~3 s (dla najbardziej wymagającego zapytania na 10 mln danych), wyraźnie lepiej niż MySQL. **Cassandra** wypadła najsłabiej – brak natywnego GROUP BY zmusza ją do pełnego skanowania danych, co skutkowało czasem ~12 s przy 10 mln rekordów w najcięższym zapytaniu. Warto zauważyć, że zarówno w MySQL, jak i PostgreSQL zapytania grupujące po większej liczbie kolumn wykonywały się szybciej niż grupowanie po pojedynczym atrykucie – sugeruje to, że operacje grupowania po kolumnach o mniejszej kardynalności (mniej unikalnych wartości) były efektywniej przetwarzane (mniej grup do utworzenia). Ogólnie, PostgreSQL okazał się najlepszy do zapytań analitycznych z agregacją, MongoDB drugi w kolejności, MySQL trzeci, a Cassandra znacznie wolniejsza w tego typu operacjach.

# Usuwanie danych (operacje DELETE)



Rysunek 3 Czas operacji masowego usunięcia rekordów (DELETE) w zależności od liczby rekordów.

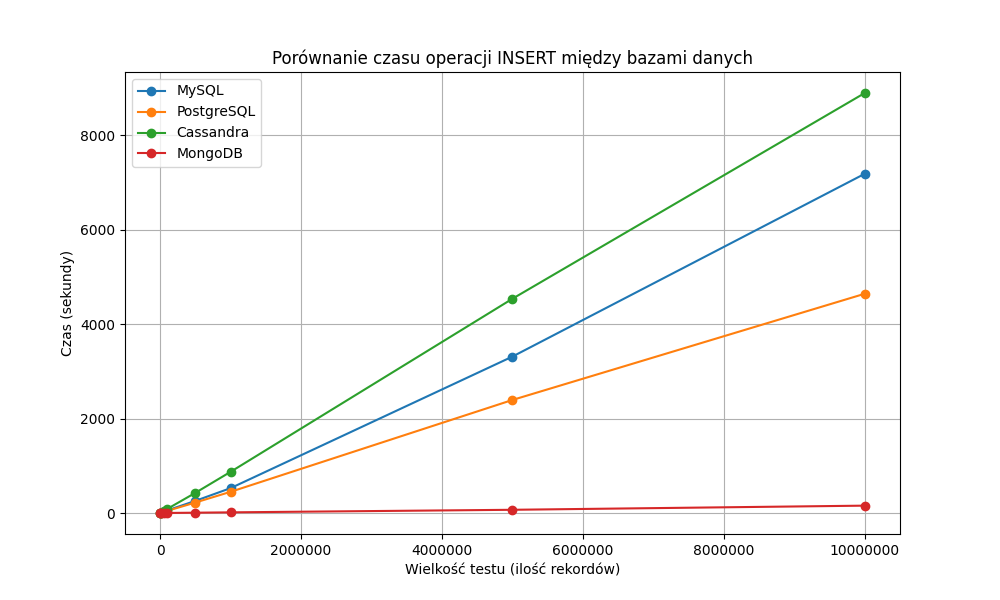
Również w teście usuwania dużej liczby danych zaobserwowano istotne różnice. **PostgreSQL** wykonał operacje DELETE najszybciej – nawet przy usunięciu 10 milionów wierszy czas wyniósł poniżej 1 sekundy. **MySQL** potrzebował już kilku sekund (ok. 5,3 s dla 10 mln), a **MongoDB** plasował się pośrodku (ok. 2,3 s dla 10 mln dokumentów). Najgorzej wypadła **Cassandra**: czas usunięcia 10 milionów wpisów sięgnął ~391 s, a wykres rośnie prawie liniowo wraz z rozmiarem danych. Wskazuje to, że Cassandra usuwa rekordy znacznie mniej efektywnie w trybie masowym niż bazy SQL czy MongoDB. Prawdopodobnie PostgreSQL wykorzystał tu optymalizację (np. **TRUNCATE** lub wydajne kasowanie sekwencyjne), stąd niezwykle niski czas, podczas gdy MySQL i MongoDB usuwały dokumenty bardziej iteracyjnie (z rosnącym kosztem wraz z wielkością tabeli). Skrajnie słaba wydajność Cassandry przy kasowaniu wynika z jej architektury – kasowanie traktowane jest jako modyfikacja danych (zapis znaczników *tombstone*) i nie usuwa fizycznie danych od razu​, co generuje ogromny narzut czasowy przy dużej liczbie rekordów.

# Analiza różnic wydajności

Zróżnicowane wyniki testów można wyjaśnić odmienną architekturą i przeznaczeniem badanych baz danych. Bazy relacyjne (MySQL, PostgreSQL) są projektowane pod kątem spójności ACID i efektywnego przetwarzania zapytań SQL – stąd dobre wyniki w operacjach INSERT (efektywne algorytmy zapisu indeksów, transakcyjność) oraz przewaga PostgreSQL w złożonych zapytaniach agregujących (zaawansowany optymalizator zapytań, możliwość równoległego przetwarzania, wykorzystanie indeksów). MySQL, używający silnika InnoDB, świetnie radzi sobie z prostymi operacjami punktowymi (np. zapytania po kluczu)​, jednak w analizach na dużych zbiorach danych ustępuje PostgreSQL, co potwierdziły nasze testy GROUP BY. PostgreSQL osiąga wysoką wydajność dzięki strategiom takim jak sekwencyjne skanowanie przy dużych tabelach czy JIT dla wyrażeń, co skraca czas przetwarzania agregacji.

MongoDB jako baza NoSQL bez narzutu transakcyjności potrafi osiągać wysoką szybkość w operacjach masowych – w testach INSERT i DELETE wyprzedził MySQL, zbliżając się momentami do PostgreSQL. Elastyczny schemat i brak kosztów związanych z integralnością referencyjną sprzyjają szybkim zapisom. Przy zapytaniach agregujących mechanizm map-reduce/potoków agregacji MongoDB okazał się wydajny, choć nie dorównał wyspecjalizowanemu silnikowi SQL PostgreSQL.

Warto jednak zwrócić uwagę na istotną różnicę w sposobie wykonywania inserta w MongoDB w zależności od użytej metody**:** insert\_many() pozwala na batchowe wstawianie wielu dokumentów naraz, co znacznie zwiększa wydajność przy dużych zbiorach danych – jak wykazano w testach pełnego załadowania danych (insert\_data.py). Z kolei insert\_one(), używany w testach syntetycznych (test.py), wykonywany w pętli dla każdego dokumentu osobno, generuje istotny narzut czasowy z powodu konieczności ustanawiania wielu połączeń i zapisu dziennika (journaling). To spostrzeżenie wyjaśnia, dlaczego MongoDB w zależności od implementacji może prezentować skrajnie różne wyniki przy operacjach INSERT – ten sam silnik, lecz różna metoda wywołania może skutkować ponad dwukrotną różnicą w czasie wykonania.



Rysunek 4 Pierwsz operacja INSERT - uzupełnienie bazy danymi

Cassandra jest wyraźnie zoptymalizowana pod innym kątem – jej architektura faworyzuje scenariusze rozproszone i ciągłe zapisy skalowane horyzontalnie kosztem wydajności pojedynczego węzła w zadaniach analitycznych. Słabe wyniki Cassandry przy odczytach z agregacją i kasowaniu wynikają z braku mechanizmów znanych z SQL (brak globalnych indeksów do grupowania, konieczność skanowania wszystkich partycji) oraz z kosztu mechanizmu *tombstone* przy usuwaniu danych​. Jak zauważają twórcy Cassandry, usunięcie danych generuje tombstones – specjalne rekordy oznaczające skasowanie – które następnie obciążają odczyty i pamięć, dopóki nie zostaną usunięte podczas kompakcji danych​. To tłumaczy niemal liniowy, wysoki czas usuwania – każda usunięta kolumna to dodatkowy zapis i późniejsze czyszczenie podczas konsolidacji danych.

# Wnioski

Przeprowadzone testy pokazują, że wybór bazy danych powinien być dostosowany do charakteru obciążenia:

* **PostgreSQL** – sprawdzi się najlepiej tam, gdzie kluczowa jest złożona analiza danych, raportowanie i skomplikowane zapytania SQL. Dzięki wysokiej wydajności agregacji i solidnym transakcjom jest idealny do zastosowań BI, OLAP oraz systemów wymagających integralności danych.
* **MySQL** – będzie dobrym wyborem do klasycznych zastosowań webowych i OLTP, gdzie dominują proste operacje (wstawianie, odczyt pojedynczych rekordów). Zapewnia solidną wydajność przy umiarkowanych rozmiarach danych i prostszych zapytaniach, choć przy bardzo dużych wolumenach lub potrzebie zaawansowanych agregacji lepszy może okazać się PostgreSQL.
* **MongoDB** – nadaje się do aplikacji potrzebujących elastycznego schematu danych (np. dokumenty JSON) i wysokiej skalowalności zapisu na pojedynczym serwerze lub w klastrze *shardowanym*. Oferuje dobre czasy zapisu i przyzwoite możliwości agregacji, więc sprawdzi się w systemach gromadzących duże ilości danych semi-strukturalnych (np. logi, dane IoT), gdzie relacyjne operacje łączenia nie są wymagane.
* **Cassandra** – jest właściwym rozwiązaniem dla rozproszonych systemów na wielką skalę, zwłaszcza gdy priorytetem jest nieprzerwany zapis i odczyt po kluczach (np. systemy telemetryczne, globalne systemy transakcyjne, systemy monitorowania). Jej wydajność rośnie wraz z dodawaniem węzłów, co pozwala obsłużyć miliardy rekordów, jednak nie nadaje się do *ad-hoc* analiz – brak wsparcia dla join/aggregate i znaczny narzut przy usuwaniu danych oznacza, że Cassandrę warto stosować tam, gdzie wymagania względem złożonych zapytań są minimalne, zaś dominują bardzo częste, proste operacje na wielkich zbiorach danych.