Politechnika Poznańska

Hurtownie Danych

Kompresja słownikowa

Adrian Glapiński
Paweł Korobczyński

Opiekunowie:
dr hab. inż. Robert Wrembel, prof. nadzw. PP
Michał Bodziony, Senior Architect, IBM Polska

9 czerwca 2021

Spis treści

1 Opis projektu

Celem projektu jest analiza kompresji słownikowej oraz ocena jej jakości. Zebrane zostaną informacje na temat kompresji w dostępnych na rynku bazach danych w celu wybrania jednego silnika bazy danych, który posłuży do dalszych testów. Dodatkowo, przeanalizowane zostaną metody próbkowania, aby znaleźć reprezentatywną próbkę z wygenerowanych danych, która zostanie wykorzystana w eksperymentach. Ponadto, poszukiwane będą takie miary odległości danych tekstowych, które najlepiej pasują do przeprowadzanych badań. Pierwszym eksperymentem będzie zbadanie czy posortowanie danych wejściowych ma wpływ na kompresję w istniejących silnikach baz danych. Nastepnie, przeprowadzone zostanie doświadczenie, które pozwoli na znalezienie miary odległości najlepiej oddającej kompresowalność. W ramach kolejnego eksperymentu, poszukiwane będzie takie posortowanie kolumn tekstowych, które da najwyższy stopień kompresji. Dane będą pochodzić z benchmarku TPC-DS [1]. Projekt jest realizowany dla IBM Software Lab Kraków.

2 Analiza istniejących rozwiązań

2.1 Algorytmy kompresujące w silnikach baz danych

W ramach niniejszego projektu miały zostać przeprowadzone eksperymenty dotyczące kompresji danych w istniejących bazach danych. W tym celu zostały poddane analizie najpopularniejsze systemy baz danych dostępne na rynku. Zebrane informacje zostały zawarte w tabeli 1.

Baza danych	**Wersja**	**Poziom, na którym działa kompresja**	**Algorytm kompresji**
IBM Db2 [2]	12.0	Tabela, blok [3]	Fixed-length/Huffman [4]
PostgreSQL [5]	13.0	Blok [6]	PGLZ (oparty na LZ77) [7]
MongoDB [8]	4.4	Blok [9]	snappy (LZ77), zlib (DEFLATE), zstd [9]
MySQL [10]	8.0	Tabela, blok [11]	LZ77 [12], LZ4 [13]
Oracle [14]	19c	Blok [15]	-
Microsoft SQL Server [16]	2019	Blok [17]	gzip (DEFLATE) [17]

Tabela 1: Kompresja w dostępnych bazach danych

Do dalszych eksperymentów została wybrana baza danych PostgreSQL, po-

nieważ jest udostępniana w ramach licencji Open Source, łatwa w instalacji oraz posiada rozbudowaną dokumentację. Wartym zwrócenia uwagi jest fakt, iż dokumentacje takich silników danych jak Oracle czy Microsoft SQL Server nie zawierają dokładnych informacji na temat implementacji kompresji i uzyskanie niektórych danych nie było możliwe.

2.2 Metody próbkowania

Eksperymenty nie mogły zostać przeprowadzone na pełnych danych ze wzlędu na złożoność obliczeniową algorytmów sortujących oraz algorytmów kompresji danych, dlatego z danych początkowych musiała zostać wybrana reprezentatywna próba. Pod uwagę zostały wzięte nastepujące probabilistyczne metody próbkowania [18]:

- prosta próba losowa - każdy element w populacji ma równą szansę dostania się do próbki,
- próba warstwowa - populacja dzielona jest na warstwy na podstawie określonych cech, a następnie każda z warstw podlega próbkowaniu losowemu,
- próba grupowa - populacja zostaje podzielona na podgrupy (klastry). Próbką zostaje jedna, losowo wybrana podgrupa,
- próba systematyczna - losowy wybór pierwszego elementu, a następnie wybieranie co k-tego elementu, gdzie k jest rozmiarem populacji dzielonym przez rozmiar próbki.

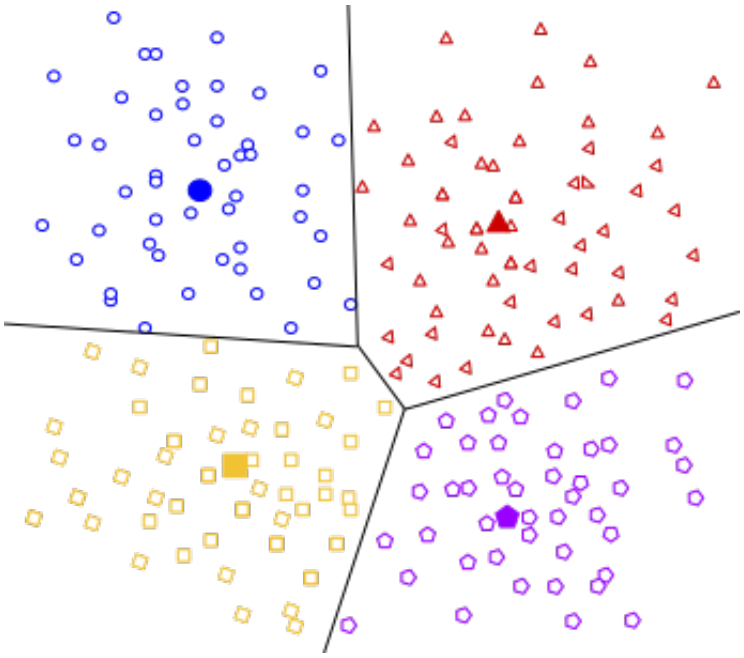
2.3 Algorytmy klastrowania

Jeden z testów zakładał wykorzystanie danych wybranych za pomocą inteligentnej próby ze zbioru poklastrowanego. W tym celu, przeanalizowane zostały następujące typy algorytmów klastrowania [19]:

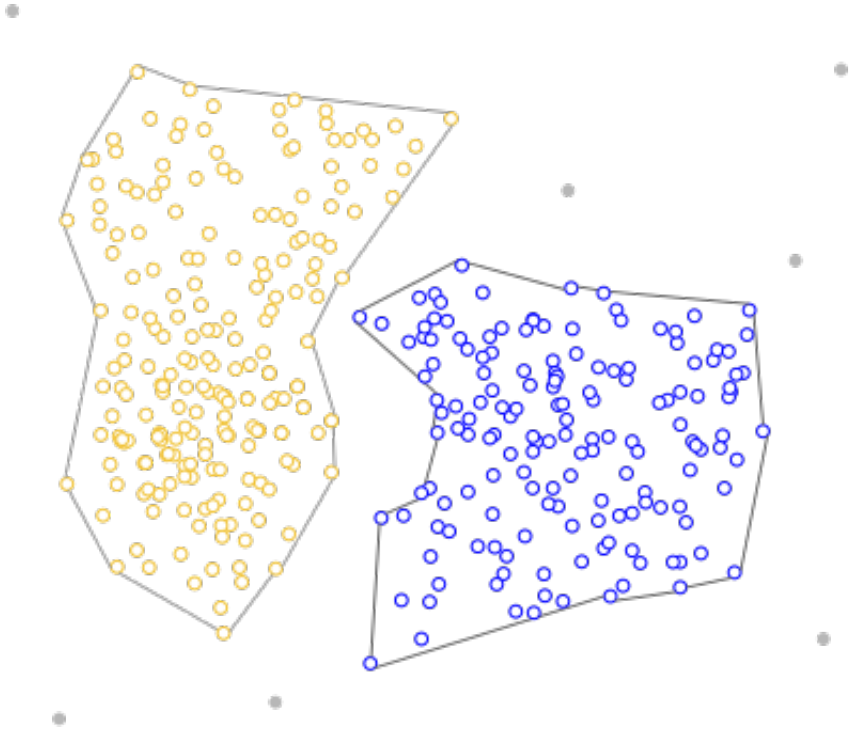
- algorytmy centroidów (eng. centroid - based clustering) - grupowanie danych w sposób niehierarchiczny. Są to algorytmy czułe pod względem wartości odstających i warunków początkowych (rysunek 1). Przykładowe algorytmy: K-Means, Mean-Shift,
- algorytmy oparte na gęstości (eng. density - based clustering) - łączenie obszarów o wysokiej gęstości danych w klastry. Algorytmy te mają trudność z danymi rozmieszczonymi w obszarach o różniej gęstości (rysunek 2). Przykładowy algorytm: DBSCAN,

- algorytmy oparte na rozkładach prawdopodobieństwa (eng. distribution - based clustering) - algorytmy zakładające, że dane złożone są z rozkładów, np. rozkładu normalnego. Wraz z oddalaniem się od środka rozkładu maleje prawdopodobieństwo należenia danych do danego klastra. W razie braku wiedzy na temat rozkładu danych należy stosować inne algorytmy (rysunek 3). Przykładowy algorytm: Expectation-Maximization (EM) using Gaussian Mixture Models (GMM),

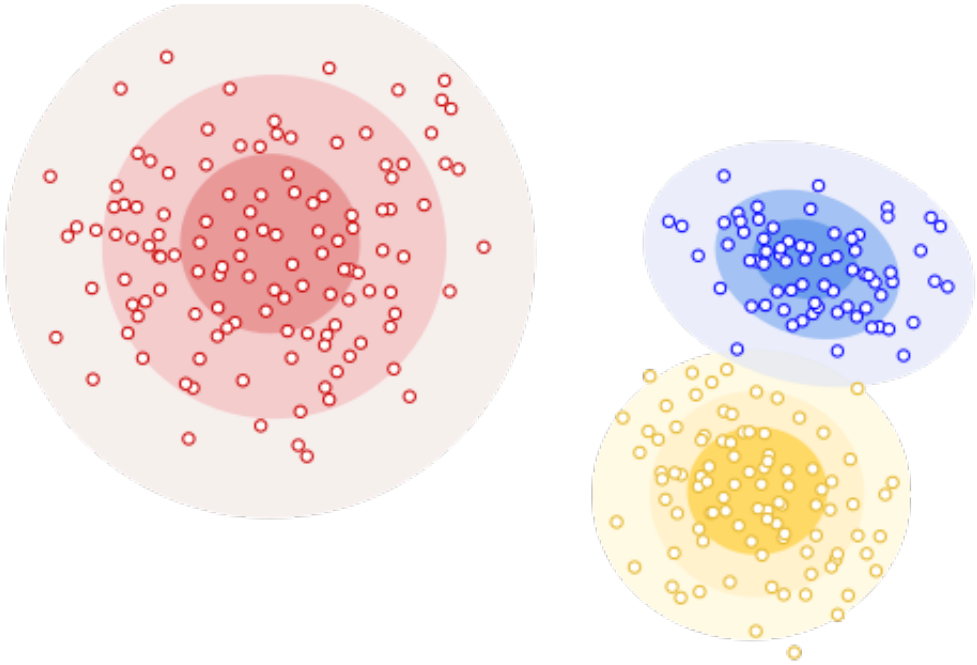
- algorytmy hierarchiczne (eng. hierarchical clustering) - dwa rodzaje algorytmów hierarchicznych: aglomeracyjny (eng. agglomerative) oraz deglomeracyjny (eng. divisive). Pierwszy z nich zaczyna z liczbą klastrów równą liczbie danych, a następnie łączy klastry na podstawie określonej miary odległości, aż do uzyskania klastra zawierającego wszystkie dane. Drugi natomiast, zaczyna z jednym klastrem i dzieli go na kolejne klastry, dokonując podziałów na podstawie wartości atrybutów (rysunek 4).



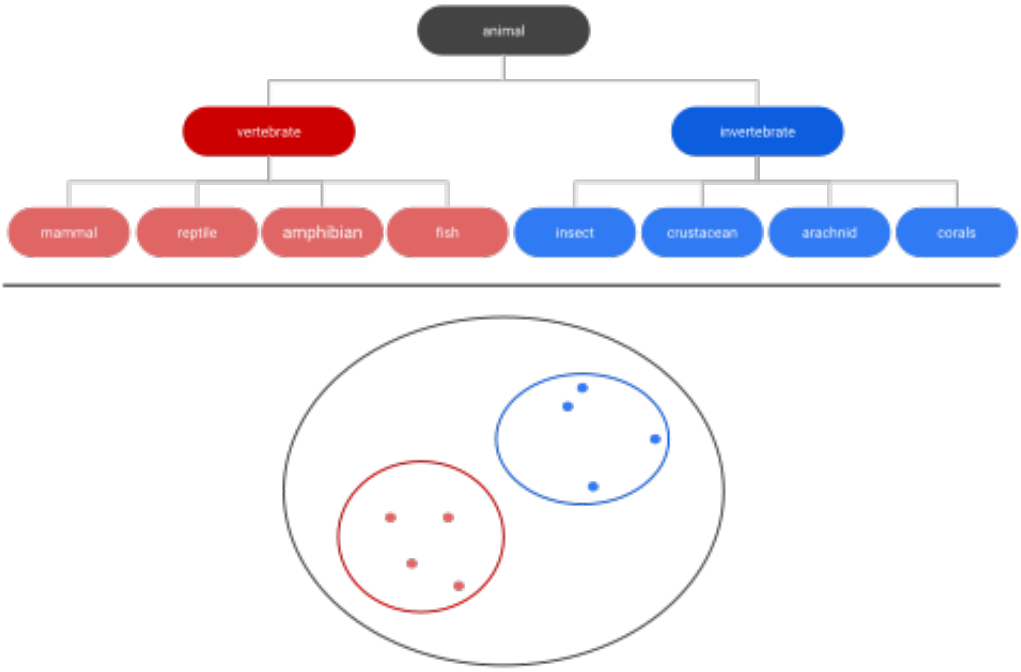
Rysunek 1: Algorytm centroidów [19]



Rysunek 2: Algorytm oparty na gęstości [19]



Rysunek 3: Algorytm oparty na rozkładach prawdopodobieństwa [19]



Rysunek 4: Algorytm hierarchiczny [19]

2.4 Miary odległości danych tekstowych

Jednym z zadań projektu było przeanalizowanie aktualnie najpowszechniej wykorzystywanych miar odległości danych tekstowych, w celu wyboru tych najbardziej odpowiednich do wykonania dalszych badań. Miary te dzielą się na kilka kategorii, do których należą:

- miary edycyjne,
- miary oparte na sekwencjach znaków,
- miary oparte na tokenach.

2.4.1 Miary edycyjne

Miary edycyjne wskazują najmniejszą wymaganą liczbę działań, przekształcających jeden ciąg znaków w drugi. Każda z miar odległości edycyjnej określa odmienny zestaw operacji edycji łańcucha znaków.

Odległość Hamminga
Odległość Hamminga określa najmniejszą liczbę operacji zastępowania elementu innym, jakie pozwalają przeprowadzić jeden ciąg na drugi. W związku z tym, wymagane jest, aby oba badane ciągi znaków były tej samej długości. Czasowa złożoność obliczeniowa tej miary wynosi $\mathcal{O}(n)$, gdzie n to długość ciągów znaków.

Odległość Levenshteina
Odległość Levenshteina jest uogólnieniem odległości Hamminga. Do dozwolonych działań na ciągu znaków w jej przypadku należą:

- wstawienie nowego znaku,
- usunięcie znaku,
- zamiana znaku na inny znak.

Używając algorytmu programowania dynamicznego (algorytm Wagnera-Fischera), odległość pomiędzy dwoma ciągami a i b można obliczyć w czasie $\mathcal{O}(|a| \times |b|)$ [20].

Odległość Damerau-Levenshteina
Odległość Damerau-Levenshteina to uogólnienie odległości Levenshteina. W przypadku tej miary dozwolone operacje edycji ciągu znaków to:

- wstawienie nowego znaku,

- usunięcie znaku,
- zamiana znaku na inny znak,
- zamiana miejscami (transpozycja) dwóch sąsiednich znaków.

Efektywne algorytmy obliczania odległości Damerau-Levenshteina zapewniają taką samą złożoność czasową, jak dla odległości Levenshteina – $\mathcal{O}(|a| \times |b|)$ [20].

Odległość Jaro
Podobieństwo Jaro między ciągami znaków a i b wyraża się następującym wzorem:

$$\mathrm{sim}_j = \begin{cases} 0 & \text{dla } m = 0, \\ \frac{1}{3}\left(\frac{m}{|a|} + \frac{m}{|b|} + \frac{m-t}{m}\right) & \text{dla } m \neq 0 \end{cases}$$

gdzie:

- m to liczba pasujących znaków. Dwa znaki, odpowiednio z a i b, są uważane za pasujące tylko wtedy, gdy są takie same i są nie dalej niż $\left\lfloor \frac{\max(|a|,|b|)}{2} \right\rfloor - 1$ znaków od siebie,
- t to liczba dopasowań po transpozycji drugiego ciągu znaków, podzielona przez 2.

Wartość podobieństwa Jaro to liczba rzeczywista w przedziale $[0, 1]$.
Dystans Jaro d_j jest zdefiniowany jako $d_j = 1 - sim_j$.
Złożoność czasowa algorytmu do obliczania odległości Jaro wynosi $\mathcal{O}(|a|+|b|)$ [20].

Odległość Jaro-Winklera
Algorytm Winklera jest ulepszeniem algorytmu Jaro, powstałym poprzez zastosowanie pomysłów opartych na obserwacjach empirycznych, które wykazały, że mniej błędów występuje zwykle na początku błędnie zapisanych nazwisk osób. Algorytm Winklera zwiększa zatem miarę podobieństwa Jaro dla identycznych znaków początkowych:

$$sim_{jw}(a, b) = sim_j(a, b) + l * p * (1 - sim_j(a, b))$$

gdzie:

- l jest długością wspólnego prefiksu na początku obu ciągów, maksymalnie wynoszącą 4.
- p jest współczynnikiem skalowania. Współczynnik skalowania nie powinien przekraczać 0.25. W przeciwnym razie podobieństwo może stać się większe niż 1, ponieważ maksymalna długość rozpatrywanego prefiksu wynosi 4.

Wartość podobieństwa Jaro-Winklera to liczba rzeczywista w przedziale $[0, 1]$. Podobnie jak w przypadku odległości Jaro, możemy zdefiniować odległość Jaro-Winklera jako $d_{jw} = 1 - sim_{jw}$.
Złożoność czasowa algorytmu do obliczania odległości Jaro-Winklera wynosi $\mathcal{O}(|a| + |b|)$ [20].

2.4.2 Miary oparte na sekwencjach znaków

Popularnymi miarami odległości stosowanymi do oceny podobieństwa sekwencji symboli są:

- Longest Common Subsequence,
- Longest Common Substring.

Longest Common Subsequence
Miara najdłuższej wspólnej podsekwencji definiuje odległość między sekwencjami x i y w następujący sposób:

$$D(x, y) = |x| + |y| - 2|LCS(x, y)|$$

gdzie $LCS(x, y)$ to najdłuższa wspólna podsekwencja sekwencji x i y.
Złożoność czasowa algorytmu do obliczania odległości Longest Common Subsequence wynosi $\mathcal{O}(|x| \times |y|)$ [20].

Longest Common Substring
Miara najdłuższego wspólnego podciągu jest obliczana analogicznie jak dla Longest Common Subsequence, lecz wszystkie znaki muszą następować bezpośrednio jeden po drugim. Na przykład, dla dwóch zdań:

The fox jumped over the high fence

The quick brown fox jumped over the fence

najdłuższą podsekwencją jest:

The fox jumped over the fence

a najdłuższym podciągiem jest:

fox jumped over the

Złożoność czasowa algorytmu do obliczania odległości Longest Common Substring również wynosi $\mathcal{O}(|x| \times |y|)$ [20].

2.4.3 Miary oparte na tokenach

Dla miar odległości opartych na tokenach, oczekiwanym wejściem jest zestaw tokenów, a nie kompletne ciągi znaków. Idea polega na znalezieniu podobnych tokenów w obu zbiorach. Im mniejsza liczba wspólnych tokenów, tym większy dystans między zestawami.

Odległość q-gram
Tokenem, którego używamy w tej metodzie jest q-gram - podciąg znaków o długości q wejściowego ciągu znaków. Przykładowo, ciąg znaków 'peter' zawiera następujące 2-gramy: 'pe', 'et', 'te' oraz 'er'.
Odległość q-gram to suma bezwzględnych różnic między wektorami (q-gramami) o długości q obu ciągów.
Złożoność czasowa algorytmu do obliczania odległości q-gram ciągów a i b wynosi $\mathcal{O}(|a| + |b|)$ [20].

3 Wykorzystanie benchmarku TPC-DS

Do eksperymentów potrzebne były dane, które zostały wygenerowane za pomocą benchmarku TPC-DS [21]. Posiada on dane w formie tabel faktów oraz tabel wymiarów generowanych w ramach sprzedaży/zwrotów różnorodnych przedmiotów. W projekcie zostały wykorzystane następujące tabele:

- Store_sales - tabela faktów (schemat na rysunku 5),
- Customer - tabela wymiarów (schemat na rysunku 6),
- Item - tabela wymiarów (schemat na rysunku 7).

Column	Datatype	NULLs	Primary Key	Foreign Key
ss_sold_date_sk	identifier			d_date_sk
ss_sold_time_sk	identifier			t_time_sk
ss_item_sk (1)	identifier	N	Y	i_item_sk
ss_customer_sk	identifier			c_customer_sk
ss_cdemo_sk	identifier			cd_demo_sk
ss_hdemo_sk	identifier			hd_demo_sk
ss_addr_sk	identifier			ca_address_sk
ss_store_sk	identifier			s_store_sk
ss_promo_sk	identifier			p_promo_sk
ss_ticket_number (2)	identifier	N	Y	
ss_quantity	integer			
ss_wholesale_cost	decimal(7,2)			
ss_list_price	decimal(7,2)			
ss_sales_price	decimal(7,2)			
ss_ext_discount_amt	decimal(7,2)			
ss_ext_sales_price	decimal(7,2)			
ss_ext_wholesale_cost	decimal(7,2)			
ss_ext_list_price	decimal(7,2)			
ss_ext_tax	decimal(7,2)			
ss_coupon_amt	decimal(7,2)			
ss_net_paid	decimal(7,2)			
ss_net_paid_inc_tax	decimal(7,2)			
ss_net_profit	decimal(7,2)			

Rysunek 5: Tabela faktów Store_sales [21]

Column	Datatype	NULLs	Primary Key	Foreign Key
c_customer_sk	identifier	N	Y	
c_customer_id (B)	char(16)	N		
c_current_cdemo_sk	identifier			cd_demo_sk
c_current_hdemo_sk	identifier			hd_demo_sk
c_current_addr_sk	identifier			ca_addres_sk
c_first_shipto_date_sk	identifier			d_date_sk
c_first_sales_date_sk	identifier			d_date_sk
c_salutation	char(10)			
c_first_name	char(20)			
c_last_name	char(30)			
c_preferred_cust_flag	char(1)			
c_birth_day	integer			
c_birth_month	integer			
c_birth_year	integer			
c_birth_country	varchar(20)			
c_login	char(13)			
c_email_address	char(50)			
c_last_review_date_sk	identifier			d_date_sk

Rysunek 6: Tabela wymiarów Customer [21]

Column	Datatype	NULLs	Primary Key	Foreign Key
i_item_sk	identifier	N	Y	
i_item_id (B)	char(16)	N		
i_rec_start_date	date			
i_rec_end_date	date			
i_item_desc	varchar(200)			
i_current_price	decimal(7,2)			
i_wholesale_cost	decimal(7,2)			
i_brand_id	integer			
i_brand	char(50)			
i_class_id	integer			
i_class	char(50)			
i_category_id	integer			
i_category	char(50)			
i_manufact_id	integer			
i_manufact	char(50)			
i_size	char(20)			
i_formulation	char(20)			
i_color	char(20)			
i_units	char(10)			
i_container	char(10)			
i_manager_id	integer			
i_product_name	char(50)			

Rysunek 7: Tabela wymiarów Item [21]

Wymienione wyżej tabele zostały wybrane z tego względu, że posiadają różnorodne typy danych, np. dane tekstowe, daty oraz liczby. Po połączeniu tabeli faktów z tabelami wymiarów dane były gotowe do przeprowadzania na nich eksperymentów.

4 Eksperymenty

4.1 Wpływ sortowania danych na stopień kompresji w PostgreSQL

W niniejszym eksperymencie został zbadany stopień kompresji w bazie danych PostgreSQL w zależności od rodzaju posortowania danych wejściowych. Eksperyment został wykonany w dwóch próbach z różnym przebiegiem oraz z wykorzystaniem innych rozmiarów danych wejściowych.

Eksperyment nr I
Pierwszy eksperyment został przeprowadzony na danych o rozmiarze 373MB (391 569 408B) z posortowaniem wielu różnych kolumn. Wybór takiego rozmiaru próbki wynikał ze złożoności obliczeniowej algorytmu sortowania danych oraz opreracji *INSERT* w bazie danych. Przebieg pierwszego eksperymentu:

1. Wygenerowanie danych za pomocą benchmarku TCP-DS.

2. Załadowanie danych do tabel Store_sales (13GB), Customer (448MB) oraz Item (108MB) do bazy danych PostgreSQL.

3. Utworzenie nowej tabeli *joined* w bazie danych oraz załadowanie do niej danych uzyskanych w wyniku operacji połączenia tabeli faktów z tabelami wymiarów (17GB).

4. Wybór próby losowej (373MB) z tabeli *joined* oraz zapisanie jej do nowo utworzonej tabeli *joined_sample*.

5. Posortowanie tabeli *joined_sample* po wielu kolumnach niezależnie oraz eksport danych bezpośrednio do nowo utworzonych tabel dla każdego posortowania.

6. Sprawdzanie rozmiaru nowo utworzonych tabel i porównanie wyniku z oryginalną tabelą.

Wybrane kolumny wykorzystane do posortowania oraz wyniki eksperymentu prezentuje tabela 2.

Kolumna	**Sortowanie**	**Rozmiar**
brand	rosnąco	373MB 391 061 504B
brand	malejąco	373MB 391 192 576B
category	rosnąco	373MB 391 610 368B
category	malejąco	373MB 391 544 832B
item_desc	rosnąco	372MB 390 553 600B
item_desc	malejąco	373MB 390 889 472B
manufact	rosnąco	373MB 391 151 616B
manufact	malejąco	373MB 390 848 512B
product_name	rosnąco	372MB 390 479 872B
product_name	malejąco	372MB 390 176 768B
brak posortowania		**373MB 391 585 792B**

Tabela 2: Wyniki pierwszego eksperymentu

Eksperyment nr II
Drugi eksperyment został przeprowadzony na danych o rozmiarze 17GB z posortowaniem jednej kolumny *brand*. Wybór tej kolumny został podyktowany tym, że jest ona typu tekstowego i intuicyjnie (z języka angielskiego oznacza *markę*) wydaje się doskonale segregować dane. Przebieg drugiego eksperymentu:

1. Wygenerowanie danych za pomocą benchmarku TPC-DS.
2. Załadowanie danych do tabel Store_sales (13GB), Customer (448MB) oraz Item (108MB) do bazy danych PostgreSQL.
3. Utworzenie nowej tabeli *joined* w bazie danych oraz załadowanie do niej danych uzyskanych w wyniku operacji połączenia tabeli faktów z tabelami wymiarów (17GB).

4. Eksport do pliku danych nieposortowanych z tabeli *joined* oraz posortowanych rosnąco po kolumnie *brand*.

5. Stworzenie tabel o takim samym schemacie: *joined_original* oraz *joined_brand_asc*.

6. Import danych do odpowiednich tabel.

7. Porównanie rozmiaru tabeli *joined_original* oraz *joined_brand_asc*.

Wyniki eksperymentu zostały przedstawione w tabeli 3.

Kolumna	**Sortowanie**	**Rozmiar**
brand	rosnąco	16GB 17 489 657 856B
brak posortowania		**16GB** **17 462 165 504B**

Tabela 3: Wyniki drugiego eksperymentu

4.2 Znalezienie miary odległości najlepiej oddającej kompresowalność

Celem niniejszego eksperymentu było odnalezienie takiej miary odległości, która byłaby najbardziej skorelowana ze stopniem kompresji pliku z danymi w naturalnym porządku, oraz pliku z posortowanymi danymi. Eksperyment został przeprowadzony na danych, które w naturalnym porządku posiadały rozmiar 5 633 496 863B (6GB). Przebieg eksperymentu:

1. Strumieniowe czytanie wierszy pliku danych z wykorzystaniem okna przesuwnego o rozmiarze 2.

2. Obliczanie miar odległości (Levenshtein, Jaro-Winkler, 2-gram) między 2 wierszami w oknie przesuwnym w następujący sposób:

   - obliczenie miar odległości między zawartością i-tej kolumny wiersza 1. i wiersza 2., powtórzone dla każdej wybranej kolumny,
   - obliczenie średniej odległości między wszystkimi kolumnami.

3. Obliczenie średnich miar odległości (Levenshtein, Jaro-Winkler, 2-gram) między wszystkimi wierszami pliku.

4. Skompresowanie pliku z wykorzystaniem algorytmu *gzip*.

5. Obliczenie stopnia kompresji pliku (ilorazu rozmiaru oryginalnego pliku i pliku po kompresji).

Powyższe kroki zostały wykonane 2 razy – dla pliku danych w naturalnym porządku oraz dla pliku danych posortowanych malejąco po kolumnie *category*.

Wyniki eksperymentu zostały przedstawione w tabeli 4.

	Znormalizowana średnia odległość między kolejnymi wierszami	
Algorytm miary odległości	**Naturalny porządek danych**	**Dane posortowane (category malejąco)**
Levenshtein	0.76649	0.71832
Jaro-Winkler	0.49341	0.46041
2-gram	0.83018	0.79381
Stopień kompresji	2.30581	2.32957

Tabela 4: Wyniki eksperymentu

4.3 Posortowanie kolumn tekstowych dające najlepszy stopień kompresji

W eksperymencie zbadano w sposób zachłanny, jakie posortowanie kolumn tekstowych daje najlepszy stopień kompresji. Dodatkowo, podobnie jak w eksperymencie 4.2, obliczono miary odległości i porównano je z uzyskanymi stopniami kompresji. Przebieg eksperymentu:

1. Wybór próbki 20% danych z wykorzystaniem funkcji *TABLESAMPLE* w trybie *BERNOULLI* systemu baz danych PostgreSQL. Próbka 20% zawiera 4 778 775 wierszy – 2 688 867 041B (2.5GB) danych.

2. Obliczenie odległości Jaro-Winklera i Levenshteina próbki.

3. Skompresowanie pliku próbki z wykorzystaniem algorytmu *gzip*.

4. Obliczenie stopnia kompresji pliku próbki (ilorazu rozmiaru oryginalnego pliku i pliku po kompresji).

5. Posortowanie pliku względem jednej z wybranych kolumn z wykorzystaniem konsolowego narzędzia *sort* dostępnego na systemach uniksopodobnych.

6. Obliczenie odległości Jaro-Winklera i Levenshteina posortowanego pliku.

7. Skompresowanie posortowanego pliku z wykorzystaniem algorytmu *gzip*.

8. Obliczenie stopnia kompresji posortowanego pliku.

9. Powtórzenie kroków 5.-8. dla wszystkich kolumn.

10. Wybór najlepszego posortowania (względem stopnia kompresji).

11. Dodanie do kolumny z najlepszym posortowaniem kolejnej, względem której będzie odbywać się sortowanie.

12. Powtórzenie kroków 9.-11. aż do wyczerpania się zbioru pozostałych kolumn, względem których jest możliwość sortowania.

Miary odległości plików były wyznaczane w odmienny sposób względem eksperymentu 4.2:

- obliczenie miar odległości między zawartością i-tej kolumny wiersza 1. i wiersza 2., powtórzone dla każdej kolumny tekstowej,
- obliczenie sumy odległości między wszystkimi kolumnami tekstowymi,
- zsumowanie odległości między wszystkimi wierszami pliku.

Poprzez wykonanie powyższych kroków uzyskano następujące posortowanie, najlepsze względem uzyskanego stopnia kompresji:

$$[25, 47, 28, 32, 34, 36, 38, 39, 40, 41, 42, 43, 45, 53, 54, 55, 56, 60, 61, 62]$$

gdzie poszczególne liczby w tablicy oznaczają numery indeksów (rozpoczynających się od 0) tabeli próbki, względem których odbywało się sortowanie. Informacje o 3 z powyżej wymienionych, najbardziej znaczących kolumnach tabeli próbki przedstawiono w tabeli 5 oraz na rysunku 8.

Nr indeksu kolumny	**Nazwa kolumny**	**Typ kolumny**
25	i_item_id	CHAR(16)
47	c_customer_id	CHAR(16)
28	i_item_desc	VARCHAR(200)

Tabela 5: Trzy najbardziej znaczące kolumny z najlepszego posortowania

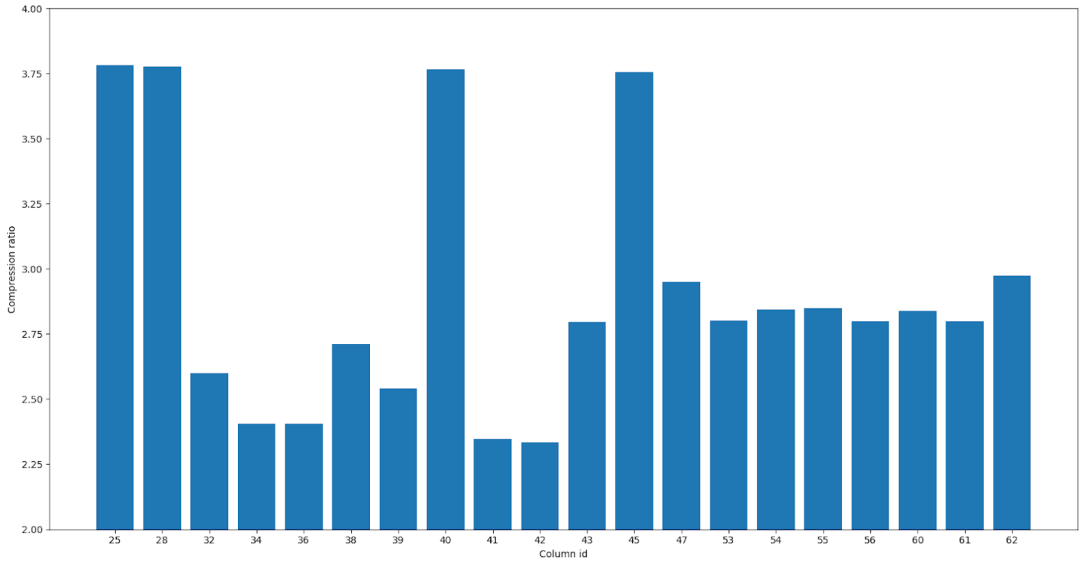
	i_item_id text	c_customer_id text	i_item_desc text
1	AAAAAAAAJALLAAAA	AAAAAAAACMBMAAAA	Capitalist institutions ask thereby difficult, pure classes. On...
2	AAAAAAAAKHEACAAA	AAAAAAAAOAOFMBAA	Great, related sha
3	AAAAAAAANBNLCAAA	AAAAAAAAGPHGFBAA	Red discussions thank fresh voices. Also possible clothes ...
4	AAAAAAAAPCEMBAAA	AAAAAAAAEKDLJAAA	Fixed, specific figures used to want most in a ski
5	AAAAAAAAIFMKCAAA	AAAAAAAAFKCIEAAA	Spanish, equal points should meet so certain sta
6	AAAAAAAACPDLAAAA	AAAAAAAAJAAPMBAA	Long patients may not write also existing notes. Previously...
7	AAAAAAAAKCDMCAAA	AAAAAAAAFINDPAAA	About overseas maps get happily real, willing opinions. Gre...
8	AAAAAAAAOGBABAAA	AAAAAAAABLMKDBAA	Shares should not make however. Planes shall abandon at ...
9	AAAAAAAAAEJADAAA	AAAAAAAABLMKDBAA	Local, small years lead just runs. Prime years
10	AAAAAAAAHOJABAAA	AAAAAAAAGOKBPAAA	Possible films ought to place appropriately in the chairs. S...
11	AAAAAAAABGIDBAAA	AAAAAAAADMEDEAAA	Words go labour cases. Stars discr
12	AAAAAAAAAHCICAAA	AAAAAAAAGOCGCAAA	About effective results would not attempt just traditional, tr...
13	AAAAAAAAJAOACAAA	AAAAAAAAKGFICBAA	Tired, able words might renew right
14	AAAAAAAAFEKGAAAA	AAAAAAAAHGHIHAAA	Trying, particular minutes can try perhaps for a services. R...
15	AAAAAAAAEFMNBAAA	AAAAAAAAEFMCDBAA	Secondary, social rewards stare away to a minutes. Brown, ...
16	AAAAAAAAMPCJAAAA	AAAAAAAAPBNNHBAA	Hours might favour
17	AAAAAAAAILNADAAA	AAAAAAAAJIAGGAAA	Angle
18	AAAAAAAAAHGAAAAA	AAAAAAAAIEBOBAAA	Previously legitimate hearts may intervene right from a exp...
19	AAAAAAAACJGAAAAA	AAAAAAAAAKFBEBAA	Clear types buy years. Companies used to go already. Stabl...
20	AAAAAAAANEJKBAAA	AAAAAAAAIBNLIAAA	Material families should not rally young doors. Values free ...
21	AAAAAAAAMMOFAAAA	AAAAAAAAHKGDHAAA	Current, international daughters pick here on a contacts: lia...

Rysunek 8: Przykładowe zawartości kolumn i_item_id, c_customer_id, i_item_desc

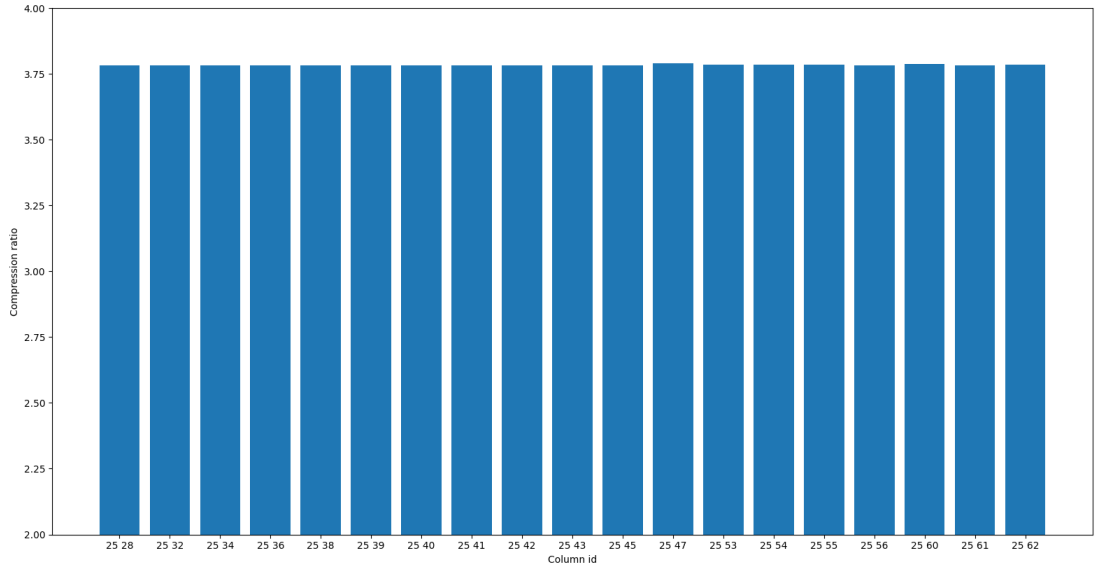
Uzyskane metryki (rozmiar po kompresji, współczynnik kompresji, dystans Levenshteina i Jaro-Winklera) dla danych w naturalnym porządku oraz danych z najlepszym posortowaniem, przedstawiono w tabeli 6. Interesujące zależności odczytane z wyników przedstawiono na rysunkach 9-11.

	Dane w naturalnym porządku	**Najlepsze posortowanie**
Rozmiar po kompresji	1 178 945 259B	709 267 354B
Współczynnik kompresji	2.28074	3.79105
Dystans (Levenshtein)	1 182 163 509	470 956 073
Dystans (Jaro-Winkler)	41 691 867	24 730 511

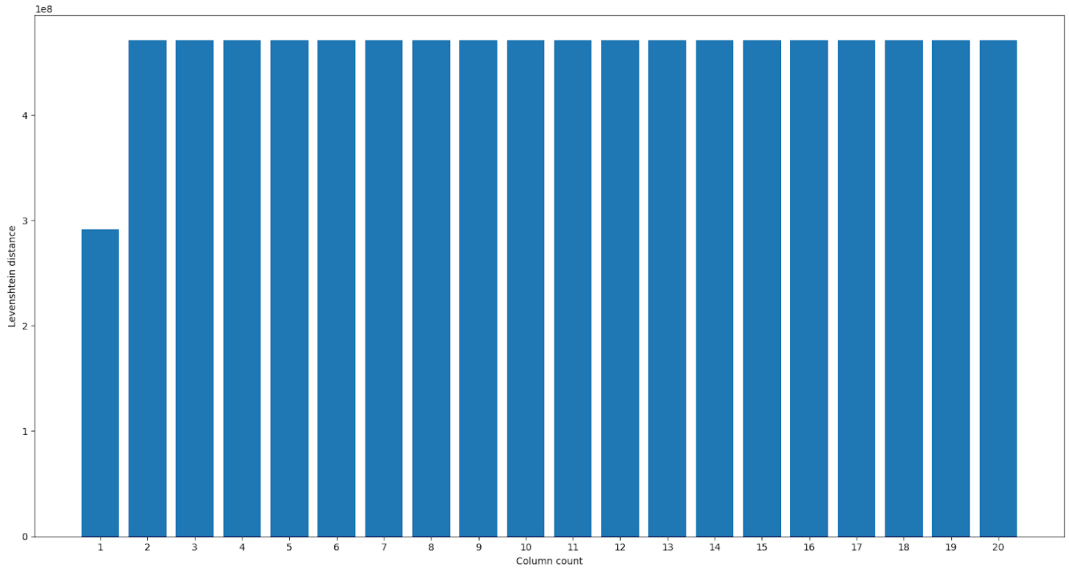
Tabela 6: Wyniki eksperymentu



Rysunek 9: Stopień kompresji dla poszczególnych kolumn (iteracja 1)



Rysunek 10: Stopień kompresji dla poszczególnych kolumn (iteracja 2)



Rysunek 11: Dystans Levenshteina w zależności od liczby posortowanych kolumn

5 Podsumowanie projektu

5.1 Podsumowanie eksperymentu 4.1

Eksperyment miał na celu zweryfikowanie, czy posortowanie danych wejściowych ma wpływ na stopień kompresji w bazie danych PostgreSQL. Został on podzielony na dwie części. Najpierw, zbadana została mniejsza próbka danych (373MB) z większą liczbą posortowań. Wynikowa tabela 2 wskazuje, że posortowanie danych nie ma wpływu na stopień kompresji danych w bazie danych. Najlepszy stopień kompresji został uzyskany przez posortowanie kolumny *product_name* malejąco, który wynosił około 1.00356. Jest to wartość sugerująca, że posortowanie danych wejściowych nie ma wpływu na stopień kompresji, lecz próba o wielkości 373MB mogła być zbyt mała, aby zaobserwować korzyści płynące z posortowania danych, dlatego został przeprowadzony kolejny eksperyment, który przebiegał w podobny sposób, natomiast wykorzystywał dane o rozmiarze 17GB i posortowanie po jednej kolumnie tekstowej *brand*. Tabela 3 przedstawia wynik testu i po raz kolejny stopień kompresji, wynoszący około 0.99842, wskazuje na brak korzyści wynikających z posortowania danych wejściowych.

5.2 Podsumowanie eksperymentu 4.2

Eksperyment miał za zadanie wyłonienie miary odległości, która w sposób najlepszy oddawałaby kompresowalność danych tekstowych. Widoczne różnice w tabeli 6 w miarach odległości, jak i stopniu kompresji, były rzędu wielkości 10^{-2}. Wybór kolumny *category* do posortowania nie miał dużego wpływu na poziom kompresji, co zostało uwidocznione w eksperymencie 4.3. Ze względu na obliczanie średniej odległości między wszystkimi kolumnami, w wynikach miar odległości została utracona informacja o wagach poszczególnych kolumn, będąca następstwem różnych długości zawartości badanych kolumn. Pomimo powyższych faktów, w wynikach ukazanych w tabeli 6 można zaobserwować ujemną korelację między stopniem kompresji a średnią odległością między wierszami tabeli - wraz ze wzrostem stopnia kompresji, maleją wszystkie z badanych miar odległości (Levenshtein, Jaro-Winkler, 2-gram).

5.3 Podsumowanie eksperymentu 4.3

Celem eksperymentu było zbadanie, jaki dobór kolumn do posortowania danych wejściowych okaże się najlepszy w kontekście uzyskania jak największego stopnia kompresji. Proces wyznaczania najlepszego doboru kolumn miał charakter zachłanny i został szczegółowo opisany w podrozdziale 4.3. W tym eksperymencie również dokonano obliczeń miar odległości (Levenshtein oraz Jaro-Winkler), które zostały porównane z uzyskanymi stopniami kompresji. W tym przypadku jednak zmieniono sposób obliczania miar odległości, aby otrzymać bardziej wiarygodne wyniki. Uzyskany rezultat, przedstawiony w tabeli 6, wskazuje że posortowanie odpowiednich kolumn tekstowych tabeli może mieć znaczący wpływ na stopień kompresji - w tym przypadku pozwoliło na uzyskanie 1.66 razy większego stopnia kompresji. Dla najlepszego posortowania otrzymano także 2.51 razy mniejszy dystans Levenshteina oraz 1.69 razy mniejszy dystans Jaro-Winklera. Rysunek 9 przedstawia stopnie kompresji dla danych posortowanych po jednej kolumnie. Wynika z niego, iż posortowanie po jednej z kolumn o indeksach: 25, 28, 40, 45 ma największy wpływ na stopień kompresji. Rysunek 10 pozwala wywnioskować, iż jednoczesne sortowanie dodatkowo po kolejnej kolumnie nie ma znaczącego wpływu na stopień kompresji. Rysunek 11 nasuwa wniosek, iż posortowanie po więcej niż jednej kolumnie jednocześnie nie ma znaczącego wpływu na wynikowy dystans Levenshteina.

6 Kierunki dalszych badań

Kompresja słownikowa jest obszernym zagadnieniem, które pozwala na przeprowadzanie wielu badań i analiz. W niniejszym rozdziale zostanie przedstawiona propozycja kolejnych badań związanych z tym tematem.

Przeprowadzony eksperyment sprawdzający wpływ posortowania danych wejściowych na stopień kompresji w bazie danych PostgreSQL może zostać rozszerzony o sprawdzenie posortowania wszystkich dostępnych kolumn tekstowych w tabeli z wykorzystaniem próby o większym rozmiarze (przynajmniej 17GB). Dodatkowo, testy mogą zostać rozszerzone o inne istniejące na rynku silniki baz danych, np. Oracle czy IBM Db2. Ponadto, może zostać sprawdzone w jaki sposób posortowanie danych wejściowych wpływa na bazy danych NoSQL, np. MongoDB.

Bibliografia

[1] TPC. Oficjalna dokumentacja benchmarków TPC. [on-line] http://tpc.org/, [dostęp: 07.06.2021].

[2] IBM. Oficjalna dokumentacja IBM Db2. [on-line] https://www.ibm.com/docs/en/db2/11.5, [dostęp: 07.06.2021].

[3] IBM. Poziom działania kompresji w bazie danych IBM Db2. [on-line] https://www.ibm.com/docs/en/db2-for-zos/12?topic=performance-compressing-your-data, [dostęp: 09.06.2021].

[4] IBM. Algorytmy kompresji wykorzystane w bazie danych IBM Db2. [on-line] https://www.ibm.com/docs/en/db2-for-zos/12?topic=performance-compressing-your-data, [dostęp: 09.06.2021].

[5] PostgreSQL. Oficjalna dokumentacja PostgreSQL. [on-line] https://www.postgresql.org/docs/13/index.html, [dostęp: 07.06.2021].

[6] PostgreSQL. Dokumentacja mechanizmu kompresji TOAST. [on-line] https://www.postgresql.org/docs/current/storage-toast.html, [dostęp: 07.06.2021].

[7] PostgreSQL. Kod źródłowy implementacji algorytmu PGLZ. [on-line] https://doxygen.postgresql.org/pg__lzcompress_8c_source.html, [dostęp: 07.06.2021].

[8] MongoDB. Oficjalna dokumentacja MongoDB. [on-line] https://docs.mongodb.com/manual/introduction/, [dostęp: 07.06.2021].

[9] MongoDB. Mechanizmy kompresji w WiredTiger Storage Engine. [on-line] https://docs.mongodb.com/manual/core/wiredtiger/#compression, [dostęp: 07.06.2021].

[10] Oracle. Oficjalna dokumentacja MySQL. [on-line] https://dev.mysql.com/doc/refman/8.0/en/, [dostęp: 07.06.2021].

[11] Oracle. Dokumentacja MySQL. Mechanizmy kompresji w InnoDB Storage Engine. [on-line] https://dev.mysql.com/doc/refman/8.0/en/innodb-compression.html, [dostęp: 07.06.2021].

[12] Oracle. Dokumentacja MySQL. Szczegółowy opis mechanizmów kompresji tabeli w InnoDB Storage Engine. [on-line] https://dev.mysql.com/doc/refman/8.0/en/innodb-compression-internals.html, [dostęp: 07.06.2021].

[13] Oracle. Dokumentacja MySQL. Szczegółowy opis mechanizmów kompresji bloku w InnoDB Storage Engine. [on-line] https://dev.mysql.com/doc/refman/8.0/en/innodb-page-compression.html, [dostęp: 07.06.2021].

[14] Oracle. Oficjalna dokumentacja Oracle. [on-line] https://docs.oracle.com/en/database/, [dostęp: 07.06.2021].

[15] Oracle. Poziom kompresji w bazie danych Oracle. [on-line] https://docs.oracle.com/en/database/oracle/oracle-database/19/tgdba/database-performance-tuning-guide.pdf, [dostęp: 09.06.2021].

[16] Microsoft. Oficjalna dokumentacja Microsoft SQL Server. [on-line] https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15, [dostęp: 07.06.2021].

[17] Microsoft. Algorytm i poziom działania kompresji w bazie danych Microsoft SQL Server. [on-line] https://docs.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression?view=sql-server-ver15, [dostęp: 09.06.2021].

[18] Wikipedia. Probabilistyczne metody próbkowania. [on-line] https://en.wikipedia.org/wiki/Sampling_(statistics), [dostęp: 07.06.2021].

[19] Google. Algorytmy klastrowania. [on-line] https://developers.google.com/machine-learning/clustering/clustering-algorithms, [dostęp: 07.06.2021].

[20] Peter Christen. A Comparison of Personal Name Matching: Techniques and Practical Issues. *in 'The Second International Workshop on Mining Complex Data (MCD'06)*, 12 2006.

[21] TPC. Specyfikacja benchmarku TPC-DS. [on-line] http://tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.0.0.pdf, [dostęp: 07.06.2021].