САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра математической теории игр и статистических решений

***Хайрутдинова Жасмин Джамильевна***

**Выпускная квалификационная работа бакалавра**

**Эффективное сжатие реляционных баз данных**

Направление: 01.03.02 Прикладная математика и информатика

ООП: Прикладная математика, фундаментальная информатика и программирование

Научный руководитель:

доктор физ.-мат. наук, профессор

Олемской Игорь Владимирович

Рецензент:

кандидат физ.-мат. наук

Коврижных Николай Александрович

Санкт-Петербург

2022

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc103712421)

[ВВЕДЕНИЕ 3](#_Toc103712422)

[ПОСТАНОВКА ЗАДАЧИ 5](#_Toc103712423)

[ОБЗОР ЛИТЕРАТУРЫ 6](#_Toc103712424)

[ГЛАВА 1 7](#_Toc103712425)

[ФИЗИЧЕСКОЕ ХРАНЕНИЕ ИНФОРМАЦИИ В СУБД POSTGRESQL 7](#_Toc103712426)

[ФИЗИЧЕСКОЕ ХРАНЕНИЕ ИНФОРМАЦИИ В СУБД MYSQL 9](#_Toc103712427)

[ГЛАВА 2 10](#_Toc103712428)

[ТЕСТОВЫЙ НАБОР ДАННЫХ 10](#_Toc103712429)

[БИБЛИОТЕКА PYTHON-ZSTANDARD 11](#_Toc103712430)

[СЖАТИЕ ФАЙЛОВ С РАЗНЫМИ УРОВНЯМИ СЖАТИЯ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 12](#_Toc103712431)

[ГЛАВА 3 15](#_Toc103712432)

[АЛГОРИТМ ПЕРЕСТАНОВКИ БАЙТОВ ДЛЯ ЭЛЕМЕНТОВ ОДИНАКОВОЙ ДЛИНЫ НА ПРИМЕРЕ POSTGRESQL 15](#_Toc103712433)

[АЛГОРИТМ ПЕРЕСТАНОВКИ БАЙТОВ ДЛЯ ЛЮБЫХ ДАННЫХ НА ПРИМЕРЕ POSTGRESQL 17](#_Toc103712434)

[АЛГОРИТМ ПЕРЕСТАНОВКИ БАЙТОВ ДЛЯ ЛЮБЫХ ДАННЫХ НА ПРИМЕРЕ MYSQL 20](#_Toc103712435)

[ИССЛЕДОВАНИЕ ЭНТРОПИИ ТАБЛИЦ 22](#_Toc103712436)

[ВЫВОДЫ 23](#_Toc103712437)

[ЗАКЛЮЧЕНИЕ 24](#_Toc103712438)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc103712439)

[ПРИЛОЖЕНИЯ 27](#_Toc103712440)

# ВВЕДЕНИЕ

В связи с развитием информационных технологий тема хранения данных становится все более актуальной.

**Рисунок 1. Рост глобальной датасферы по годам [14]**



 Согласно оценкам International Data Corporation (IDC) количество цифровой информации в мире растет в геометрической прогрессии, и чтобы размеры всей этой информации не стали в скором времени проблемой, нужно постоянно совершенствовать эффективность сжатия данных.

В общем случае могут использоваться стандартные алгоритмы сжатия данных без анализа их оптимальности. Но одни и те же алгоритмы могут давать абсолютно разный результат на разных типах данных. Анализ структуры данных поможет подобрать оптимальный алгоритм для каждого случая. В данной работе предложен алгоритм, преобразующий формат хранения таблиц реляционных баз данных к виду, который будет эффективнее сжиматься стандартными алгоритмами.

Для выполнения поставленной задачи взята идея сжатия табличных данных по столбцам [5], на основе которой реализован алгоритм преобразования данных для эффективного сжатия на примере СУБД MySQL и PostgreSQL

Большинство алгоритмов сжатия основывается на кодировании одинаковых последовательности байтов. Физическое хранение данных в колоночных БД реализовано по столбцам, вследствие чего данные одного типа оказываются рядом, что повышает эффективность их сжатия.

Эффективность алгоритма сжатия определяется с помощью коэффициента сжатия CR (compression ratio):

 .

# ПОСТАНОВКА ЗАДАЧИ

Целью данной научно-исследовательской работы является разработка алгоритма перестановки байтов в файлах реляционных БД для повышения коэффициента сжатия.

Для выполнения поставленных целей необходимо решить ряд задач:

* Изучить принципы физического хранения информации в СУБД PostgreSQL и MySQL
* Создать тестовый набор файлов из различных наборов данных
* Изучить принципы работы алгоритма сжатия zstandard
* Реализовать алгоритм перестановки байт в файлах памяти PostgreSQL и MySQL и провести теоретический анализ его влияния на коэффициент сжатия

# ОБЗОР ЛИТЕРАТУРЫ

В связи с развитием технологий, связанных с хранением и обработкой больших данных, колоночные БД набирают всё большую популярность.

Было неоднократно продемонстрировано, что хранение табличных данных в колоночном формате повышает эффективность их сжатия по сравнению со строковым форматом, традиционно используемым в реляционных СУБД [1, 2].

В своём теоретическом исследовании Д. Абади показал, что алгоритмы сжатия данных лучше работают с данными с низкой информационной энтропией [3].

В статье [4] исследовано хранение данных в распределённой файловой системе HDFS. Авторы пришли к выводу, что хотя применяемые для этого форматы данных Avro (строковый) и Parquet (колоночный) используют одинаковый алгоритм сжатия, Parquet примерно в 1,5 раза компактнее, чем Avro.

Авторы статьи [5] представляют идею транспонирования таблицы текстовых данных CSV, которая помогает сделать данные более однородными и тем самым улучшить коэффициент сжатия.

#  ГЛАВА 1

## ФИЗИЧЕСКОЕ ХРАНЕНИЕ ИНФОРМАЦИИ В СУБД POSTGRESQL

Файлы конфигурации и файлы данных, используемые кластером базы данных, традиционно хранятся вместе в каталоге данных кластера, который по умолчанию называется $PGDATA (по имени переменной среды, которую можно использовать для его определения). В ОС Linux обычно $PGDATA находится в /var/lib/pgsql/data.

В каталоге $PGDATA содержатся подкаталоги и управляющие файлы. Подкаталог base содержит подкаталоги для каждой базы данных. Они именуются в соответствии с OID (идентификатором базы данных в каталоге pg\_database). Каждая таблица и индекс хранятся в отдельном файле. Файл таблицы также именуется по OID этой таблицы в базе данных.

Каждая таблица хранится как массив страниц фиксированного размера (по умолчанию 8 KB, но при необходимости можно выбрать другой размер страницы при компиляции сервера). В таблице все страницы логически эквивалентны. Под элементом в странице подразумевается строка таблицы.

**Таблица 1. Общая компоновка страницы PostgreSQL [6]**

| **Элемент** | **Описание** |
| --- | --- |
| Данные заголовка страницы (PageHeaderData) | Длина — 24 байта. Содержит общую информацию о странице, включая указатели свободного пространства. |
| Данные идентификаторов элементов (ItemId) | Массив идентификаторов, указывающих на фактические элементы. Каждый идентификатор представляет собой пару «смещение, длина» и занимает 4 байта. |
| Свободное пространство (FreeSpace) | Незанятое пространство. Новые идентификаторы элементов размещаются с начала этой области, сами новые элементы — с конца. |
| Элементы (Item) | Сами элементы данных как таковые. |
| Специальное пространство (Special) | Специфические данные метода доступа. Для различных методов хранятся различные данные. Для обычных таблиц таких данных нет. |

**Рисунок 2. Компоновка страницы**



## ФИЗИЧЕСКОЕ ХРАНЕНИЕ ИНФОРМАЦИИ В СУБД MYSQL

Будем рассматривать механизм хранения InnoDB. Узнать расположение каталогов баз данных можно с помощью команды:

mysql> SHOW VARIABLES LIKE 'datadir';

В ОС Windows это, как правило, MySQL\MySQL Server 8.0\Data. Каждая база данных хранится в одноименной папке в этой директории. Каждая таблица хранится в отдельном файле с расширением .ibd.

Каждая таблица хранится как массив страниц фиксированного размера (по умолчанию 16 КB). Под элементом в странице подразумевается строка таблицы.

Каждая страница состоит из 7 частей:

**Таблица 2. Общая компоновка страницы MySQL [7]**

| **Элемент** | **Описание** |
| --- | --- |
| Fil Header [:38] | Файловый заголовок |
| Page Header [38:94] | Заголовок страницы |
| Infimum + Supremum Records | Системное поле, для навигации |
| User Records | Данные |
| Free Space | Незанятое пространство |
| Page Directory | Системное поле, количество указателей записей |
| Fil Trailer | Контрольная сумма |

# ГЛАВА 2

## ТЕСТОВЫЙ НАБОР ДАННЫХ

 Для изучения алгоритма сжатия был подготовлен следующий набор данных:

**Таблица 3. Тестовый набор данных**

|  |  |
| --- | --- |
| **Файл** | **Описание** |
| Данные 0 | Данные о террористических актах в разных странах [13]. Данные, загруженные из .csv файла. 7 столбцов, 3 целочисленных, 1 дата, 3 строковых непостоянной длины, один из которых содержит длинные строки. Размер 4,21 MB |
| Данные 2 | Логи NBA за сезон 2014-2015 года [12]. Данные, загруженные из .csv файла. 21 столбец, из которых: 8 целочисленных, 2 целочисленных фиксированной длины, 4 вещественных, 1 время, 2 строковых длины 1, 4 строковых непостоянной длины. Размер 19,9 MB |
| Данные 3 | Сгенерированные синтетические данные. 2 столбца, один заполненный одинаковыми строками, второй – счетчик. Размер 60,2 MB |

## БИБЛИОТЕКА PYTHON-ZSTANDARD

Zstandard — алгоритм сжатия данных без потерь, разрабатываемый с 2015 года Яном Колле. Относится к семейству LZ алгоритмов и работает на идее поиска одинаковых подстрок из 4 байтов. Алгоритм реализует 22 уровня сжатия, различающихся скоростью и эффективностью (уровень «1» — самый быстрый, уровень «22» — наиболее эффективный) [8]. Алгоритм характеризуется большой скоростью распаковки сжатых данных.

Для начала создадим объект класса ZstdCompressor:

cctx = zstd.ZstdCompressor(level=levels[i])

Класс имеет много параметров, по-своему влияющих на сжатие, но нас интересует только параметр level, отвечающий за уровень сжатия.

Теперь сожмем содержимое файла:

compressed = cctx.compress(data)

Для нашего случая этого будет достаточно, так как мы сжимаем последовательно по одному файлу, размер которых не очень велик. В случае если приходится работать с очень большими объемами данных, следует использовать сжатие в потоке ввода-вывода.

## СЖАТИЕ ФАЙЛОВ С РАЗНЫМИ УРОВНЯМИ СЖАТИЯ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Был реализован алгоритм, который считывает файлы, последовательно сжимает их, засекает время сжатия, считает степень сжатия и так для каждого уровня сжатия от 1 до 22. Полученные результаты выводятся в виде графиков (код программы в приложениях). Для построения графиков используется библиотека Matplotlib.

Будем обозначать:

CR (compression rate) – степень сжатия

Level – уровень сжатия

Compression speed – скорость сжатия

**График 1. Зависимость степени сжатия от уровня**

****

|  |
| --- |
| **График 2. Зависимость скорости сжатия от степени сжатия** |
|  |

Уровень 3 устанавливается по умолчанию, если не указывать параметр level. Как видно по графикам, он и в самом деле в большинстве случаев будет оптимальным, сочетая высокую скорость сжатия и хорошую степень сжатия. После этого уровня CR растет не так сильно, но скорость начинает падать резко. Поэтому, если нам надо быстро сжать данные, level=3 будет оптимальным.

В том случае, если нам нужно как можно сильнее сжать данные, то по графикам можно сделать вывод, что оптимальным будет level=18. После этого уровня степень сжатия увеличивается очень незначительно, а на данных 3 (синтетических) и вовсе падает. При этом скорость все еще сильно падает. Таким образом, нецелесообразно в большинстве случаев использовать уровни сжатия выше 18.

Такие скачки на графиках обусловлены тем, что уровень управляет набором параметров алгоритма:

**Рисунок 3. Параметры уровней [10]**



Самый главный параметр – стратегия (последний). Видим, что смена стратегии происходит на 3, 5, 7, 13, 16, 18 уровнях. Это соотносится со скачками на полученных графиках. После 18 уровня стратегия не меняется существенно (смена с ZSTD\_btultra на ZSTD\_btultra2 влияет на результат в меньшинстве случаев), что обосновывает предположение об оптимальности 18 уровня для большинства файлов.

# ГЛАВА 3

## АЛГОРИТМ ПЕРЕСТАНОВКИ БАЙТОВ ДЛЯ ЭЛЕМЕНТОВ ОДИНАКОВОЙ ДЛИНЫ НА ПРИМЕРЕ POSTGRESQL

Структура файла таблицы может быть использована для повышения эффективности сжатия. Как описано выше, файлы имеют строгую и удобную структуру, а потому с ними легко производить манипуляции. Получив и декодировав заголовок каждой страницы, мы можем легко узнать количество и размер элементов на каждой странице (строк таблицы). Если мы знаем, что на странице эти значения равны, то мы можем сделать перестановку байтов в элементах, чтобы сделать данные более однородными. Суть перестановки: сначала записываем первые байты всех элементов по порядку, затем вторые байты всех элементов и так далее, в конце будут последние байты всех элементов.

Эта идея была проверена на файле с равномерными сгенерированными данными. Для обнаружения элементов понадобятся следующие поля заголовка:

**Таблица 4. Некоторые поля заголовка**

|  |  |  |
| --- | --- | --- |
| **Поле** | **Длина** | **Описание** |
| pd\_lower | 2 байта | Смещение до начала свободного пространства |
| pd\_upper | 2 байта | Смещение до конца свободного пространства |
| pd\_special | 2 байта | Смещение до начала специального пространства |

Индексы элементов записываются с начала страницы после заголовка, а соответствующие им элементы – с конца после специального пространства.

Оцениваем количество и длину элементов для каждой страницы (предварительно декодировав поля из заголовка):

n\_items = (pd\_lower - 24) // 4

item\_size = (pd\_special - pd\_upper) // n\_items

В нашем случае на каждой странице получилось одинаковое количество и размер элементов. Значит, можем делать перестановку и сжимать файл (код алгоритма в приложениях).

Результат:

* CR после преобразования: 114,59
* CR до преобразования: 12,09

Видим, что CR был улучшен примерно в 9,5 раза.

Несмотря на то, что такой алгоритм можно применять лишь в ограниченных случаях, он все-таки может быть полезен, если нам надо сжать очень большие таблицы, хранящие строки одинаковой длины, например файлы с ключами, номерами, идентификаторами.

Следует подробнее изучить, на каких файлах его можно успешно применять. Дальнейшая цель – адаптировать алгоритм таким образом, чтобы мы могли подавать на вход файлы с любыми данными, и тогда мы сможем эффективно сжимать любые файлы.

Для подтверждения результатов производилась проверка: декомпрессия данных и последующее их преобразование к исходному виду. Данные после обратных преобразований совпали с исходными данными, что говорит о том, что алгоритм корректен.

## АЛГОРИТМ ПЕРЕСТАНОВКИ БАЙТОВ ДЛЯ ЛЮБЫХ ДАННЫХ НА ПРИМЕРЕ POSTGRESQL

Чтобы алгоритм можно было применять на любых данных, добавим проверку: если все элементы на одной странице (файла памяти таблицы) одинаковы, то делаем описанную выше перестановку, иначе оставляем страницу как есть. Таким образом, даже если малая часть страниц в таблице будет преобразована, в сумме в базе данных мы получим хороший выигрыш в сжатии.

Размер элементов можно получить как разность смещений соседних элементов. Смещение считается от начала страницы до начала элемента, хранится в битовом флаге lp\_off поля ItemID, соответствующего элементу.

Результаты для PostgreSQL:

**Таблица 5. Описание тестовых БД**

|  |  |
| --- | --- |
| Название базы данных | Описание |
| custom | БД, собранная из отдельных таблиц, взятых с сайта <https://www.kaggle.com>. Подбирались наиболее подходящие данные – минимум строковых данных, таблицы преимущественно заполнены числовыми значениями |
| postgres | База с информацией о фильмах и их прокате [11] |
| sportdb | База с информацией о проведении спортивных мероприятий[11] |

**Таблица 6. Результаты для БД «custom»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| name | cr\_new | cr\_old | pct | n\_shuffled | n\_pages |
|  foe  |  4.11  |  3.12  |  31.46  | 78 | 78 |
|  height  |  3.03  |  3.03  |  0.00  | 0 | 2 |
|  fitbitsandgradesdata  |  3.97  |  2.99  |  32.90  | 6 | 6 |
|  multi\_attribution\_sample  |  6.12  |  3.74  |  63.69  | 19 | 19 |
| health\_heart\_experimental  |  8.52  |  4.24  |  100.95  | 598 | 598 |
|  **---Total---**  |  **7.44**  |  **4.05**  |  **83.96**  | **701** | **703** |

 **Таблица 7. Результаты для БД «postgres»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| name | cr\_new | cr\_old | pct | n\_shuffled | n\_pages |
|  film  |  3.82  |  3.82  |  0.00  | 0 | 55 |
|  actor  |  4.17  |  4.17  |  0.00  | 0 | 2 |
|  address  |  2.46  |  2.46  |  0.00  | 0 | 8 |
|  category  |  21.17  |  21.17  |  0.00  | 0 | 1 |
|  city  |  3.64  |  3.64  |  0.00  | 0 | 5 |
|  country  |  4.15  |  4.15  |  0.00  | 0 | 1 |
|  customer  |  4.21  | 4.21  |  0.00  | 1 | 9 |
|  film\_actor  |  25.07  | 5.19  |  383.38  | 30 | 30 |
|  film\_category  |  24.02  |  7.55  |  218.28  | 6 | 6 |
|  inventory  |  35.44  |  7.52  |  371.55  | 25 | 25 |
|  language  |  42.67  |  42.67  |  0.00  | 1 | 1 |
|  rental  |  4.73  |  3.04  |  55.88  | 108 | 150 |
|  staff  |  34.71  |  34.71  |  0.00  | 0 | 1 |
|  store  |  84.45  |  83.59  |  1.03  | 1 | 1 |
| payment\_p2020\_01  |  5.74  |  3.64  |  57.96  | 9 | 9 |
| payment\_p2020\_02  |  5.81  |  3.45  |  68.53  | 17 | 17 |
| payment\_p2020\_03  |  6.08  |  3.58  |  70.03  | 42 | 42 |
| payment\_p2020\_04  |  6.03  |  3.52  |  71.42  | 50 | 50 |
| payment\_p2020\_05  |  5.70  |  5.70  |  0.00  | 0 | 2 |
|  **---Total---**  |  **5.47**  |  **3.61**  |  **51.58**  | **290** | **415** |

**Таблица 8. Результаты для БД «sportdb»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| name | cr\_new | cr\_old | pct | n\_shuffled | n\_pages |
|  american\_football\_defensive\_stats  |  6.64  |  6.28  |  5.77  | 1 | 5 |
| american\_football\_down\_progress\_stats  |  13.72  |  12.94  |  6.03  | 1 | 1 |
|  american\_football\_fumbles\_stats  |  6.39  |  6.39  |  0.00  | 0 | 1 |
|  american\_football\_offensive\_stats  |  20.74  |  16.86  |  23.04  | 1 | 1 |
|  american\_football\_passing\_stats  |  3.55  |  3.55  |  0.00  | 0 | 2 |
|  american\_football\_penalties\_stats  |  22.38  |  19.37  |  15.57  | 1 | 1 |
|  american\_football\_rushing\_stats  |  3.69  |  3.69  |  0.00  | 0 | 1 |
|  american\_football\_sacks\_against\_stats  |  12.90  |  12.90  |  0.00  | 0 | 1 |
|  american\_football\_scoring\_stats  |  5.52  |  5.52  |  0.00  | 0 | 4 |
|  american\_football\_special\_teams\_stats  |  3.68  |  3.68  |  0.00  | 0 | 1 |
|  baseball\_defensive\_stats  |  5.95  |  5.95  |  0.00  | 0 | 9 |
|  baseball\_offensive\_stats  |  4.78  |  4.78  |  0.00  | 0 | 21 |
|  baseball\_pitching\_stats  |  3.71  |  3.71  |  0.00  | 0 | 11 |
|  basketball\_defensive\_stats  |  6.78  |  6.78  |  0.00  | 0 | 1 |
|  basketball\_offensive\_stats  |  5.95  |  5.93  |  0.36  | 1 | 2 |
|  basketball\_rebounding\_stats  |  6.27  |  5.17  |  21.36  | 1 | 1 |
|  display\_names  |  3.26  |  3.26  |  0.00  | 0 | 44 |
|  ice\_hockey\_action\_participants  |  7.28  |  5.13  |  41.96  | 1 | 1 |
|  ice\_hockey\_action\_plays  |  2.88  |  2.88  |  0.00  | 0 | 1 |
|  ice\_hockey\_defensive\_stats  |  15.91  |  13.52  |  17.67  | 1 | 1 |
|  ice\_hockey\_offensive\_stats  |  5.36  |  5.36  |  0.00  | 0 | 6 |
|  ice\_hockey\_player\_stats  |  12.60  |  5.38  |  134.46  | 4 | 4 |
|  stats  |  9.10  |  9.10  |  0.00  | 0 | 120 |
|  team\_american\_football\_stats  |  25.60  |  20.90  |  22.50  | 1 | 1 |
|  affiliations  |  14.50  |  14.50  |  0.00  | 0 | 1 |
|  affiliations\_documents  |  18.35  |  6.78  |  170.87  | 6 | 6 |
|  documents  |  4.51  |  4.51  |  0.00  | 0 | 9 |
|  affiliations\_events  |  33.54  |  7.28  |  360.68  | 58 | 58 |
|  events  |  9.78  |  6.84  |  43.13  | 54 | 54 |
|  publishers  |  80.31  |  80.31  |  0.00  | 1 | 1 |
|  affiliation\_phases  |  16.09  |  12.28  |  31.04  | 1 | 1 |
|  american\_football\_action\_plays  |  5.40  |  5.40  |  0.00  | 0 | 3 |
|  american\_football\_action\_participants  |  4.45  |  4.45  |  0.00  | 0 | 2 |
|  persons  |  5.64  |  4.79  |  17.79  | 11 | 33 |
|  american\_football\_event\_states  |  5.52  |  5.58  |  -1.04  | 1 | 2 |
|  teams  |  8.48  |  5.96  |  42.34  | 1 | 1 |
|  baseball\_event\_states  |  28.44  |  21.06  |  35.07  | 1 | 1 |
|  baseball\_action\_plays  |  26.68  |  26.68  |  0.00  | 0 | 1 |
|  positions  |  7.24  |  7.24  |  0.00  | 0 | 1 |
|  core\_person\_stats  |  3.88  |  3.88  |  0.00  | 0 | 8 |
|  document\_contents  |  4.90  |  4.90  |  0.00  | 0 | 5 |
|  document\_fixtures  |  17.10  |  17.10  |  0.00  | 0 | 1 |
|  document\_classes  |  39.57  |  39.57  |  0.00  | 0 | 1 |
|  document\_fixtures\_events  |  19.46  |  7.57  |  157.24  | 29 | 29 |
|  events\_documents  |  27.69  |  7.88  |  251.27  | 36 | 36 |
|  events\_sub\_seasons  |  20.99  |  6.63  |  216.73  | 20 | 20 |
|  sub\_seasons  |  42.67  |  55.73  |  -23.44  | 1 | 1 |
|  ice\_hockey\_event\_states  |  8.33  |  8.33  |  0.00  | 0 | 1 |
|  injury\_phases  |  4.55  |  4.55  |  0.00  | 0 | 8 |
|  seasons  |  65.02  |  70.02  |  -7.14  | 1 | 1 |
|  latest\_revisions  |  4.50  |  4.50  |  0.00  | 0 | 3 |
|  participants\_events  |  5.77  |  5.60  |  3.08  | 6 | 70 |
|  person\_event\_metadata  |  15.77  |  5.73  |  175.03  | 30 | 30 |
|  periods  |  10.94  |  4.61  |  137.22  | 3 | 3 |
|  team\_phases  |  16.31  |  8.22  |  98.41  | 3 | 3 |
|  teams\_documents  |  8.78  |  5.05  |  74.10  | 2 | 2 |
|  **---Total---**  |  **7.71**  |  **6.03**  |  **27.82**  | **278** | **637** |

## АЛГОРИТМ ПЕРЕСТАНОВКИ БАЙТОВ ДЛЯ ЛЮБЫХ ДАННЫХ НА ПРИМЕРЕ MYSQL

Аналогичный алгоритм реализуем и для MySQL.

В MySQL берем только страницы, в которых поле fil\_page\_type=0x45bf. Длина элемента хранится в соответствующем поле в самом элементе.

Результаты для MySQL:

**Таблица 9. Описание тестовых БД**

|  |  |
| --- | --- |
| **Название базы данных** | **Описание** |
| sakila | Демонстрационная БД, поставляющаяся вместе с MySQL. Содержит данные о фильмах и их прокате |
| world | Демонстрационная БД, поставляющаяся вместе с MySQL. Содержит данные о странах |

**Таблица 10. Результаты для БД «sakila»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| name | cr\_new | cr\_old | pct | n\_shuffled | n\_pages |
|  actor.ibd  |  17.53  |  17.53  |  0.00  | 0 | 8 |
|  address.ibd  |  4.91  |  4.91  |  -0.03  | 1 | 16 |
|  category.ibd  |  54.00  |  54.00  |  0.00  | 0 | 7 |
|  city.ibd  |  8.03  |  8.03  |  0.00  | 1 | 9 |
|  country.ibd  |  30.67  |  30.67  |  0.00  | 0 | 7 |
|  customer.ibd  |  7.03  |  6.44  |  9.25  | 2 | 13 |
|  film.ibd  |  5.72  |  5.33  |  7.30  | 4 | 22 |
|  film\_actor.ibd  |  6.36  |  3.99  |  59.42  | 14 | 22 |
| ilm\_category.ibd  |  12.49  |  10.27  |  21.57  | 4 | 10 |
|  film\_text.ibd  |  6.30  |  5.59  |  12.60  | 2 | 17 |
|  inventory.ibd  |  13.85  |  5.39  |  156.84  | 21 | 28 |
|  language.ibd  |  56.66  |  58.75  |  -3.56  | 1 | 7 |
|  payment.ibd  |  31.65  |  18.85  |  67.91  | 64 | 640 |
|  rental.ibd  |  17.97  |  12.76  |  40.77  | 71 | 640 |
|  staff.ibd  |  4.59  |  4.59  |  0.00  | 3 | 11 |
|  store.ibd  |  67.12  |  67.12  |  0.00  | 3 | 9 |
|  **---Total---**  |  **18.71**  |  **13.00**  |  **43.95**  | **191** | **1466** |

**Таблица 11. Результаты для БД «world»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| name | cr\_new | cr\_old | pct | n\_shuffled | n\_pages |
|  city.ibd  |  5.62  |  5.08  |  10.62  | 27 | 38 |
|  country.ibd  |  9.43  |  9.44  |  -0.10  | 1 | 11 |
| countrylanguage.ibd  |  11.90  |  10.30  |  15.59  | 10 | 15 |
|  **---Total---**  |  **6.96**  |  **6.33**  |  **9.94**  | **38** | **64** |

## ИССЛЕДОВАНИЕ ЭНТРОПИИ ТАБЛИЦ

Поскольку алгоритмы семейства LZ основываются на поиске одинаковых подстрок длиной 4 байта, для теоретического обоснования была исследована энтропия таких подстрок на примере некоторых таблиц.

Энтропия вычисляется по формуле:

**Таблица 12. Энтропия для таблиц БД «custom»**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| name | cr\_new | cr\_old | pct | n\_shffld | n\_pages | e\_old | e\_new |
|  foe  |  4.11  |  3.12  |  31.46  | 78 | 78 |  8.53  |  6.03  |
|  height  |  3.03  |  3.03  |  0.00  | 0 | 2 |  5.71  |  5.71  |
|  fitbitsandgradesdata  |  3.97  |  2.99  |  32.90  | 6 | 6 |  6.52  |  4.42  |
|  multi\_attribution\_sample  |  6.12  |  3.74  |  63.69  | 19 | 19 |  5.83  |  3.33  |
| health\_heart\_experimental |  8.52  |  4.24  | 100.95  | 598 | 598 |  6.78  |  3.69  |

Последний столбец отображает энтропию таблиц после перестановки байтов, предпоследний – до. Видим, что энтропия падает там, где производилась перестановка, что говорит о большей однородности данных и косвенно способствует улучшению сжатия алгоритмом zstandard.

# ВЫВОДЫ

**Таблица 13. Сводная таблица результатов по всем СУБД**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **СУБД** | **Имя БД** | **CR до перестановки** | **CR после перестановки** | **Улучшение** |
| PostgreSQL |  |  |  |  |
|  | custom | 4,05 | 7,44 | 83,96% |
|  | postgres | 3,61 | 5,47 | 51,58% |
|  | sportdb | 6,03 | 7,71 | 27,82% |
| MySQL |  |  |  |  |
|  | sakila | 13 | 18,71 | 43,95% |
|  | world | 6,33 | 6,96 | 9,94% |

По результатам анализа выяснено, что данное преобразование повышает эффективность сжатия только для страниц, все элементы которых имеют одинаковый размер. Поэтому в алгоритм была добавлена проверка, и преобразование производится только при выполнении условия равенства всех длин элементов. Несмотря на то, что этому условию удовлетворяют не все страницы таблиц, а преимущественно состоящие из числовых или иных равномерных данных, в рамках полной базы данных степень сжатия увеличивается. Алгоритм был проверен на разных данных, в том числе на базах, наиболее приближенных к реальным.

Была исследована энтропия некоторых таблиц. После перестановки байтов данные становятся более однородными, поэтому энтропия уменьшается.

# ЗАКЛЮЧЕНИЕ

Был изучен общий принцип работы алгоритмов сжатия, на его основе была выбрана стратегия сжатия табличных данных по столбцам.

Было изучено физическое хранение данных в СУБД PostgreSQL и MySQL.

Была изучена работа библиотеки python-zstandard.

На основании полученных знаний был реализован алгоритм преобразования данных страниц для повышения коэффициента сжатия.

В больших промышленных БД (например, OLTP) предложенный алгоритм можно успешно применять, потому что в них, как правило, хранятся однообразные данные, и можно с уверенностью предполагать, что бо́льшая часть из них пройдет проверку.

В данной работе алгоритм реализован для PostgreSQL и MySQL, но его можно адаптировать для любых других реляционных СУБД, изучив необходимые сведения о физическом хранении таблиц БД из документации.

Также было проведено теоретическое обоснование предложенного алгоритма на основе исследования энтропии.

.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. 2005. C-store: a column-oriented DBMS. In Proceedings of the 31st international conference on Very large data bases (VLDB '05). VLDB Endowment, 553–564.
2. Tapiador, Daniel & O’Mullane, William & Brown, A.G.A. & Luri, X. & Huedo, Eduardo & Osuna, Pedro. (2014). A framework for building hypercubes using MapReduce. Computer Physics Communications. 185. 1429–1438. 10.1016/j.cpc.2014.02.010.
3. Daniel J. Abadi, Peter A. Boncz, and Stavros Harizopoulos. 2009. Column-oriented database systems. Proc. VLDB Endow. 2, 2 (August 2009), 1664–1665. https://doi.org/10.14778/1687553.1687625
4. D. Plase, L. Niedrite, and R. Taranovs, “A comparison of HDFS compact data formats: Avro versus Parquet”, MLA, vol. 9, no. 3, pp. 267-276, Jul. 2017.
5. Sebastian Wandelt, Xiaoqian Sun, Ulf Leser, Column-wise compression of open relational data, Information Sciences, Volumes 457–458, 2018, Pages 48-61, ISSN 0020-0255
6. <https://www.postgresql.org/docs/current/storage-page-layout.html>
7. <https://dev.mysql.com/doc/internals/en/innodb.html>
8. <https://python-zstandard.readthedocs.io/en/latest/compressor.html#zstdcompressionwriter>
9. [https://www.kaggle.com](https://www.kaggle.com/dansbecker/nba-shot-logs?select=shot_logs.csv)
10. [https://github.com/facebook/zstd](https://github.com/facebook/zstd%20)
11. <https://www.postgresql.org/ftp/projects/pgFoundry/dbsamples/>
12. [https://www.kaggle.com/datasets/dansbecker/nba-shot-logs?select=shot\_logs.csv](https://www.kaggle.com/datasets/dansbecker/nba-shot-logs?select=shot_logs.csv%20)
13. [https://www.kaggle.com/datasets/argolof/predicting-terrorism](https://www.kaggle.com/datasets/argolof/predicting-terrorism%20)
14. <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>

# ПРИЛОЖЕНИЯ

 **Приложение 1. Ссылка на репозиторий с программным кодом**

<https://github.com/yamero76/NIR>