Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЁТ О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Сравнение скорости CRUD операций (Clickhouse с PostgreSQL)
тема

Руководитель _____ А. Н. Пупков
подпись, дата инициалы, фамилия

Студент КИ18/17-1б 031831229 _____ В. А. Прекель
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2020

СОДЕРЖАНИЕ

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Необходимо провести сравнение скорости и анализ CRUD операций для систем управления баз данных Clickhouse и PostgreSQL.

Теоретическая часть:

1. Описать что такое CRUD операции;

2. Описать как работают хранилища данных, ссылаясь на соответствующую документацию (например если вы пишете про ClickHouse, нужно сослаться на документ https://clickhouse.yandex/docs/ru/);

3. Найти информацию о том, как и почему скорость CRUD операций хранилищ отличается, провести сравнительный анализ для каждой операции с детальным и обоснованным объяснением (со ссылками на источники);

4. Сделать выводы о том, почему в данных хранилищах имеются различия в выполнении CRUD операций, чем это вызвано и как дизайн системы влияет на данный параметр.

Экспериментальная часть:

- Установить docker toolbox (или более свежее решение);

- Скачать контейнеры с соответствующими базами данных;

- Написать два простых скрипта выполняющих CRUD операции для каждой из пары баз данных и измеряющих время выполнения;

- Каждый эксперимент провести несколько раз, при этом:

- Нужно указать параметры (виртуальной) машины, на которой проводились исследования (кол-во RAM, CPU, потоков);

- Указать количество итераций для каждого эксперимента;

- Привести значения математического ожидания и дисперсии для каждого результата;

- Сделать графики с пояснениями;

- Сделать выводы о том, почему в данных хранилищах имеются различия в выполнении CRUD операций, чем это вызвано и как дизайн системы влияет на данный параметр.

ВВЕДЕНИЕ

Система управления базами данных – набор программного обеспечения, позволяющий определять, обрабатывать, получать и управлять данными в базе данных. [1] Соответственно, основные операции – создания, чтения, изменения и удаления называются CRUD-операциями. В случае с SQL-СУБД, за CRUD операции отвечают соответственно конструкции INSERT, SELECT, UPDATE, DELETE соответственно. [2] В разных системах, с разным дизайном, на идентичной схеме данных, время выполнения CRUD-операций может различаться.

1 Теоретическая часть

1.1 Описание СУБД

Существуют строковые и столбцовые (колоночные, column-based) СУБД. Они отличаются тем, что данные, принадлежащие к одной строке в строковых СУБД, хранятся рядом, а в столбцовых СУБД хранятся рядом данные, принадлежащие к одному столбцу. Это позволяет экономить время в запросах на чтение, которые не затрагивают все столбцы таблицы и проводить прочие оптимизации, в том числе по сжатию данных, которые невозможны в столбцовых СУБД. [3] Примеры строковых СУБД: MySQL, PostgreSQL, и MS SQL Server. Примеры столбцовых СУБД: ClickHouse, Vertica, Paraccel, Sybase IQ, Exasol, Infobright, InfiniDB, MonetDB, LucidDB, SAP HANA, Google Dremel, Google PowerDrill, Druid, and kdb+. [4]

ClickHouse – строковая СУБД, предназначенная для анализа данных и OLAP-запросов. Поддерживает SQL, хоть с во многих случаях не совпадающий со стандартом. Представляется как по-настоящему столбцовая СУБД, поддерживающая сжатие данных, хранящая данные на диске, параллельно использующая процессорные ядра, и позволяющая работать на нескольких серверах в кластере. [5] Была разработана в Яндексе и выпущена под открытой лицензией в 2016 году. [6]

PostgreSQL – строковая объектно-реляционная СУБД. Поддерживает большую часть SQL стандарта, а также множество современных функций, такие как сложные запросы, внешние ключи, триггеры, изменяемые представления, транзакционная целостность, многоверсионность. [7] Начинает свою историю из 80-тых годов с проекта POSTGES в Беркли. PostgreSQL основано на последней версии POSTGRES 4.2, выпущенной в 1994 году. [8]

1.2 Анализ работы СУБД

Исходя из того, что PostgreSQL строковая СУБД, то запросы в узких таблицах должны работать быстрее. Так же, ClickHouse предназначен для

извлечения и вставки большого количества данных одним запросом, поэтому выполнение множества аналогичных запросов должно быть медленнее, чем в PostgreSQL. [9]

Так же ClickHouse поддерживает операции изменения и удаления через механизм мутаций нестандартным синтаксисом ALTER TABLE … UPDATE и ALTER TABLE … DELETE соответственно. [10] Механизм мутаций не был реализован на релизе Clickhouse и логично предположить, изменение/удаление данных будет работать хуже, чем в строковых СУБД. [11]

2 Экспериментальная часть

2.1 Подготовка к эксперименту

Для развёртывания тестируемых СУБД использовался Docker – программный пакет для контейнеризации. [12] Для работы Docker на Windows, использовался Docker Desktop [13] с бэкэндом WSL 2 [14]. Были скачаны соответствующие контейнеры yandex/clickhouse-server [15] и postgres [16].

Написана программа на языке C#, которая подключается к СУБД, выполняет нужные запросы определённое количество раз и сохраняет результаты в CSV-файл. Так как время выполнения запросов может зависеть не только от СУБД, а от программного обеспечения, обеспечивающего подключение клиентского приложения к СУБД, было решено использовать несколько драйверов, использующие технологию ADO.NET. Для ClickHouse: ClickHouse.Ado [17], Octonica.ClickHouseClient [18]. Для PostgreSQL: Npgsql [19], dotConnect Express [20] [21].

Было решено тестировать на таблице с примитивными данными. В таблице 2 столбца, первый – первичный ключ, второй – значение. Оба столбца – целое 32-ух битное число, оба столбца не могут принимать значение null.

Листинг 1 – Создание таблицы в PostgreSQL и ClickHouse

```
CREATE TABLE Benchmark
(
    key   integer PRIMARY KEY,
```

```
    value integer NOT NULL
);

CREATE TABLE Benchmark
(
    key   Int32,
    value Int32
) ENGINE = MergeTree()
      ORDER BY key
      PRIMARY KEY key;
```

Для create-операции использован следующий INSERT-запрос, вставляющий запись с ключом i и значением i. Этот запрос повторяется N раз, значение i последовательно растёт от 0 до N-1. Этот запрос идентичен для ClickHouse и PostgreSQL.

Листинг 2 – INSERT-запрос (Create)

```
INSERT INTO Benchmark VALUES ({key}, {value});
```

Для read-операции написан SELECT-запрос, который так же N раз вызывается и запрашивает значение с ключём i. Этот запрос идентичен для ClickHouse и PostgreSQL.

Листинг 3 – SELECT-запрос (Read)

```
SELECT value FROM Benchmark WHERE key = {key};
```

Для update-операции значение записи с ключём i заменяется на N-i следующим UPDATE-запросом. Для ClickHouse используется нестандартная конструкция ALTER TABLE … UPDATE.

Листинг 4 – UPDATE-запрос (Update)

```
UPDATE Benchmark SET value = {newValue} WHERE key = {key};

ALTER TABLE Benchmark UPDATE value = {newValue} WHERE key = {key};
```

Для delete-операции удаляются записи с 0 по N-1 следующим DELETE-запросом. Для ClickHouse используется нестандартная конструкция ALTER TABLE … DELETE.

Листинг 5 – DELETE-запрос (Delete)

```
DELETE FROM Benchmark WHERE key = {key};
```

```
ALTER TABLE Benchmark DELETE WHERE key = {key};
```

Полный код программы указан в приложении А.

Параметры машины: процессор Ryzen 3 1200 3.59 GHz, 4 потока, 16 Gb RAM, SSD.

2.2 Результаты эксперимента

Были проведены замеры для 100, 1000 и 5000 записей. Для каждой пары СУБД-драйвер было выполнено порядка 1400, 60 и 20 итераций соответственно.

Таблица 1 – Кол-во итераций

СУБД-драйвер	**Число записей**	**Число итераций**
ClickHouse (ClickHouse.Ado)	100	1412
ClickHouse (ClickHouse.Ado)	1000	58
ClickHouse (ClickHouse.Ado)	5000	18
ClickHouse (Octonica.ClickHouseClient)	100	419
ClickHouse (Octonica.ClickHouseClient)	1000	12
ClickHouse (Octonica.ClickHouseClient)	5000	4
Postgres (Npgsql)	100	1436
Postgres (Npgsql)	1000	59
Postgres (Npgsql)	5000	19
Postgres (dotConnect.Express.for.PostgreSQL)	100	1409
Postgres (dotConnect.Express.for.PostgreSQL)	1000	66
Postgres (dotConnect.Express.for.PostgreSQL)	5000	21

Таблица 2 – Средние значения для 100 записей

СУБД-драйвер	**Среднее Create, мс**	**Среднее Read, мс**	**Среднее Update, мс**	**Среднее Delete, мс**
ClickHouse (ClickHouse.Ado)	1218,75	821,34	1402,14	1475,33
ClickHouse (Octonica.ClickHouseClient)	515,61	530,12	1176,07	1236,83
Postgres (Npgsql)	316,23	123,70	313,01	310,82

Postgres (dotConnect.Express.for.PostgreSQL)	440,13	234,82	416,92	422,21

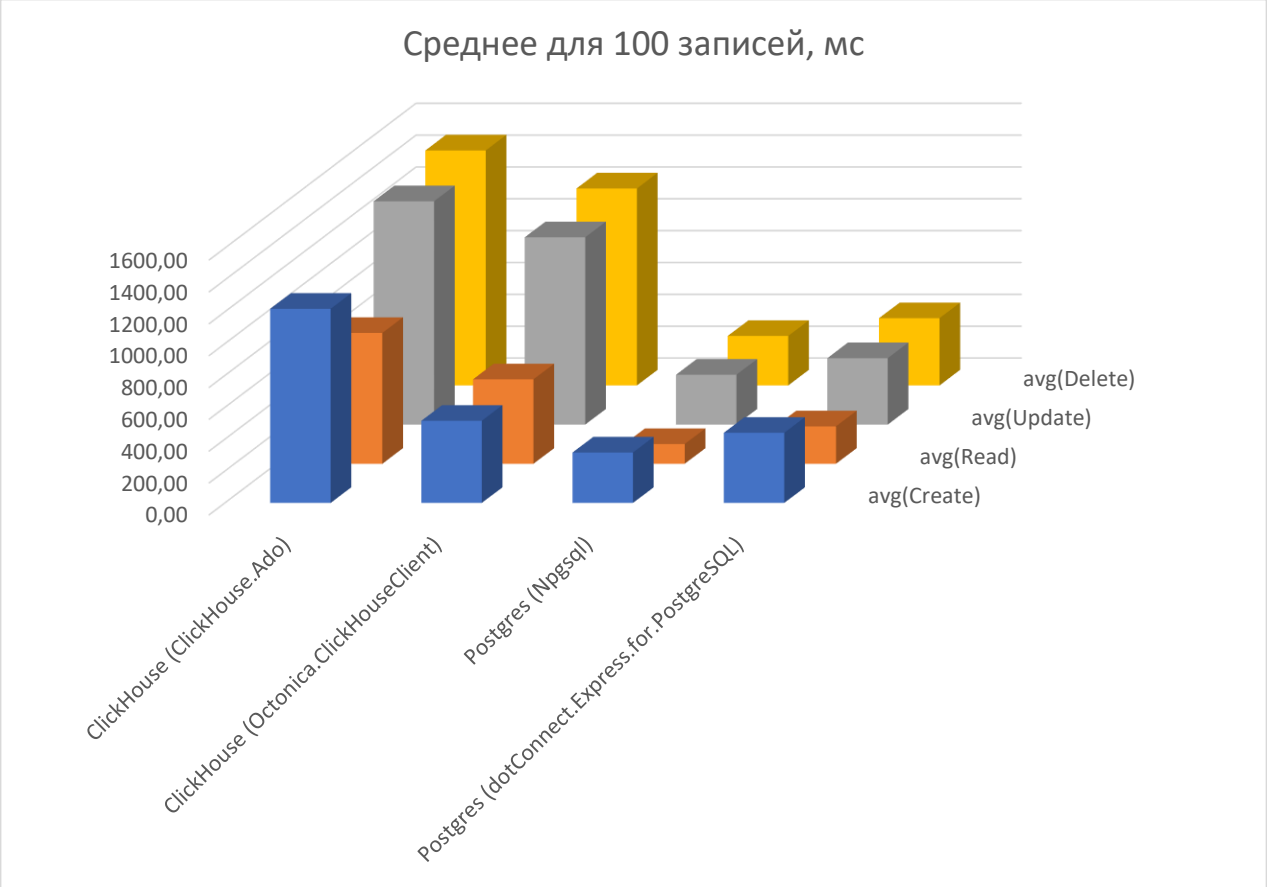

Рисунок 1 – Диаграмма времени для 100 записей

Таблица 3 – Средние значения для 1000 записей

СУБД-драйвер	**Среднее Create, мс**	**Среднее Read, мс**	**Среднее Update, мс**	**Среднее Delete, мс**
ClickHouse (ClickHouse.Ado)	12466,90	8617,71	14845,54	17636,25
ClickHouse (Octonica.ClickHouseClient)	5164,08	5825,96	13580,42	15463,70
Postgres (Npgsql)	2944,41	1249,75	2868,14	3067,58
Postgres (dotConnect.Express.for.PostgreSQL)	4141,51	2357,73	4154,59	4180,64

Рисунок 2 – Диаграмма времени для 1000 записей

Таблица 4 – Средние значения для 5000 записей

СУБД-драйвер	**Среднее Create, мс**	**Среднее Read, мс**	**Среднее Update, мс**	**Среднее Delete, мс**
ClickHouse (ClickHouse.Ado)	59829,90	43998,25	87143,45	96201,52
ClickHouse (Octonica.ClickHouseClient)	26856,12	25999,40	79564,14	85883,76
Postgres (Npgsql)	15949,27	6240,25	14199,52	15217,90
Postgres (dotConnect.Express.for.PostgreSQL)	21423,44	12294,57	20163,62	20084,61

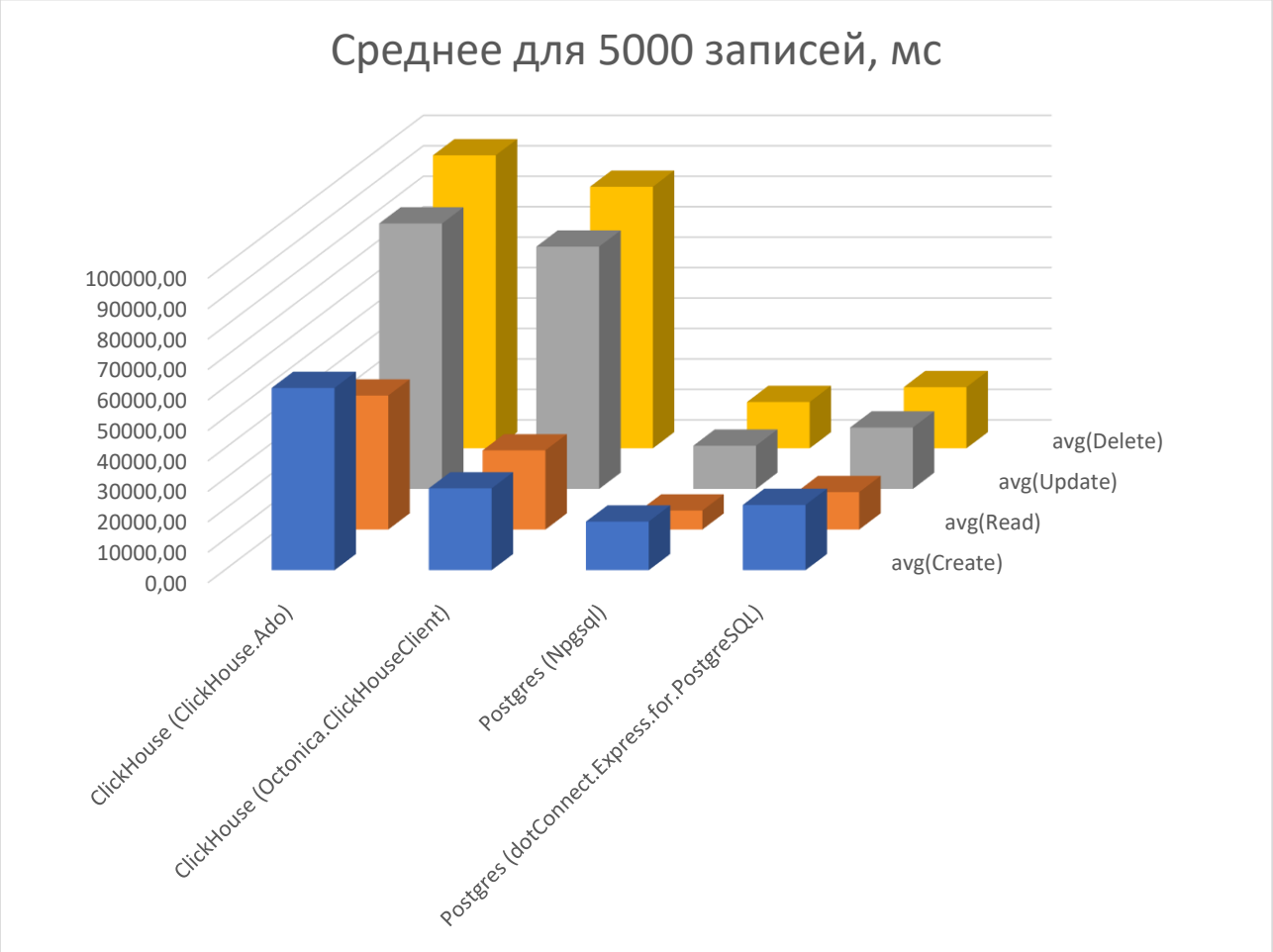

Рисунок 3 – Диаграмма времени для 5000 записей

Таблица 5 – Среднеквадратическое отклонение

СУБД-драйвер	**Число записей**	**Среднеквадратическое отклонение Create, мс**	**Среднеквадратическое отклонение Read, мс**	**Среднеквадратическое отклонение Update, мс**	**Среднеквадратическое отклонение Delete, мс**
ClickHouse (ClickHouse.Ado)	100	436,65	426,62	778,57	752,08
ClickHouse (ClickHouse.Ado)	1000	1845,48	1334,42	3601,66	5672,27
ClickHouse (ClickHouse.Ado)	5000	4684,36	4486,35	14106,16	15475,30

ClickHouse (Octonica.ClickHouseClient)	100	281,57	142,02	717,51	780,70
ClickHouse (Octonica.ClickHouseClient)	1000	1484,75	1721,69	5942,46	6245,25
ClickHouse (Octonica.ClickHouseClient)	5000	3133,14	3051,76	5524,54	5182,94
Postgres (Npgsql)	100	259,76	20,05	235,76	256,03
Postgres (Npgsql)	1000	1280,43	198,89	1170,70	1681,38
Postgres (Npgsql)	5000	4433,23	776,54	3148,96	4561,54
Postgres (dotConnect.Express.for.PostgreSQL)	100	524,68	44,31	223,69	300,89
Postgres (dotConnect.Express.for.PostgreSQL)	1000	1266,77	243,26	1129,42	1183,94
Postgres (dotConnect.Express.for.PostgreSQL)	5000	4124,73	2758,63	4591,16	4345,34

Таблица 6 – Дисперсия

СУБД-драйвер	**Число записей**	**Дисперсия Create, мс^2**	**Дисперсия Read, мс^2**	**Дисперсия Update, мс^2**	**Дисперсия Delete, мс^2**
ClickHouse (ClickHouse.Ado)	100	190663,47	182007,28	606167,06	565617,16
ClickHouse (ClickHouse.Ado)	1000	3405813,86	1780677,27	12971936,85	32174603,80
ClickHouse (ClickHouse.Ado)	5000	21943247,58	20127380,21	198983724,94	239484965,83
ClickHouse (Octonica.ClickHouseClient)	100	79282,95	20170,40	514821,26	609492,88
ClickHouse (Octonica.ClickHouseClient)	1000	2204468,98	2964213,95	35312827,37	39003140,99
ClickHouse (Octonica.ClickHouseClient)	5000	9816589,63	9313224,15	30520517,71	26862823,78
Postgres (Npgsql)	100	67474,23	402,15	55582,05	65551,42
Postgres (Npgsql)	1000	1639496,44	39557,04	1370547,54	2827024,48
Postgres (Npgsql)	5000	19653503,22	603016,45	9915934,56	20807675,77
Postgres (dotConnect.Express.for.PostgreSQL)	100	275285,86	1963,63	50035,57	90535,59
Postgres (dotConnect.Express.for.PostgreSQL)	1000	1604718,05	59173,97	1275583,73	1401713,74
Postgres (dotConnect.	5000	17013380,89	7610043,46	21078783,44	18881943,99

ЗАКЛЮЧЕНИЕ

В заключение можно отметить, что, как и ожидалось, на большом количестве нересурсозатратных запросов в узкой таблице, колоночная СУБД ClickHouse покажет себя хуже, чем строковая PostgreSQL. Особенно плохо обстоит ситуация с update и delete операциями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. What is a Database Management System (DBMS)? - Definition from Techopedia [Электронный ресурс] – Режим доступа: https://www.techopedia.com/definition/24361/database-management-systems-dbms (Дата обращения: 21.07.2020).

2. Create, read, update and delete – Wikipedia [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Create,_read,_update_and_delete(Дата обращения: 21.07.2020).

3. Column-oriented DBMS – Wikipedia [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Column-oriented_DBMS (Дата обращения: 21.07.2020).

4. Overview | ClickHouse Documentation [Электронный ресурс] – Режим доступа: https://clickhouse.tech/docs/en/ (Дата обращения: 21.07.2020).

5. Distinctive Features | ClickHouse Documentation [Электронный ресурс] – Режим доступа: https://clickhouse.tech/docs/en/introduction/distinctive-features/ (Дата обращения: 21.07.2020).

6. Яндекс открывает ClickHouse / Яндекс corporate blog / Habr [Электронный ресурс] – Режим доступа: https://habr.com/en/company/yandex/blog/303282/ (Дата обращения: 21.07.2020).

7. PostgreSQL: Documentation: 12: 1. What Is PostgreSQL? [Электронный ресурс] – Режим доступа: https://www.postgresql.org/docs/12/intro-whatis.html (Дата обращения: 21.07.2020).

8. PostgreSQL: Documentation: 12: 2. A Brief History of PostgreSQL [Электронный ресурс] – Режим доступа: https://www.postgresql.org/docs/12/history.html (Дата обращения: 21.07.2020).

9. Performance | ClickHouse Documentation [Электронный ресурс] – Режим доступа: https://clickhouse.tech/docs/en/introduction/performance/ (Дата обращения: 21.07.2020).

10. ALTER | ClickHouse Documentation [Электронный ресурс] – Режим доступа: https://clickhouse.tech/docs/en/sql-reference/statements/alter/ (Дата обращения: 21.07.2020).

11. How to Update Data in ClickHouse [Электронный ресурс] – Режим доступа: https://clickhouse.tech/blog/en/2016/how-to-update-data-in-clickhouse/ (Дата обращения: 21.07.2020).

12. Docker Engine overview | Docker Documentation [Электронный ресурс] – Режим доступа: https://docs.docker.com/engine/ (Дата обращения: 21.07.2020).

13. Docker Desktop overview | Docker Documentation [Электронный ресурс] – Режим доступа: https://docs.docker.com/desktop/ (Дата обращения: 21.07.2020).

14. Docker Desktop WSL 2 backend | Docker Documentation [Электронный ресурс] – Режим доступа: https://docs.docker.com/docker-for-windows/wsl/ (Дата обращения: 21.07.2020).

15. yandex/clickhouse-server - Docker Hub [Электронный ресурс] – Режим доступа: https://hub.docker.com/r/yandex/clickhouse-server/ (Дата обращения: 21.07.2020).

16. postgres - Docker Hub [Электронный ресурс] – Режим доступа: https://hub.docker.com/_/postgres (Дата обращения: 21.07.2020).

17. killwort/ClickHouse-Net: Yandex ClickHouse fully managed .NET client [Электронный ресурс] – Режим доступа: https://github.com/killwort/ClickHouse-Net (Дата обращения: 21.07.2020).

18. Octonica/ClickHouseClient: ClickHouse .NET Core driver [Электронный ресурс] – Режим доступа: https://github.com/Octonica/ClickHouseClient (Дата обращения: 21.07.2020).

19. npgsql/npgsql: Npgsql is the .NET data provider for PostgreSQL. [Электронный ресурс] – Режим доступа: https://github.com/npgsql/npgsql (Дата обращения: 21.07.2020).

20. ADO.NET Provider for PostgreSQL with Entity Framework Support [Электронный ресурс] – Режим доступа: https://www.devart.com/dotconnect/postgresql/ (Дата обращения: 21.07.2020).

21. NuGet Gallery | dotConnect.Express.for.PostgreSQL 7.17.1696 [Электронный ресурс] – Режим доступа: https://www.nuget.org/packages/dotConnect.Express.for.PostgreSQL/ (Дата обращения: 21.07.2020).

ПРИЛОЖЕНИЕ А

Исходный код и результаты доступны на GitHub по URL: https://github.com/prekel/PostgresClickHouseCRUD.

Листинг 1 – PostgresClickHouseCRUD.Abstract/Queries.cs

```
namespace PostgresClickHouseCRUD.Abstract
{
    public static class Queries
    {
        public static string CreateTableQuery(string table) =>
            $"CREATE TABLE {table} (key integer PRIMARY KEY, value integer NOT
NULL)";

        public static string CreateTableClickHouseQuery(string table) =>
            $"CREATE TABLE {table} (key Int32, value Int32) ENGINE = MergeTree()
ORDER BY key PRIMARY KEY key";

        public static string CreateOneQuery(string table, int key, int value) =>
            $"INSERT INTO {table} VALUES ({key}, {value})";

        public static string ReadOneQuery(string table, int key) =>
            $"SELECT value FROM {table} WHERE key = {key}";

        public static string UpdateOneQuery(string table, int key, int newValue)
=>
            $"UPDATE {table} SET value = {newValue} WHERE key = {key}";

        public static string UpdateOneClickHouseQuery(string table, int key, int
newValue) =>
            $"ALTER TABLE {table} UPDATE value = {newValue} WHERE key = {key}";

        public static string DeleteOneQuery(string table, int key) => $"DELETE
FROM {table} WHERE key = {key}";

        public static string DeleteOneClickHouseQuery(string table, int key) =>
            $"ALTER TABLE {table} DELETE WHERE key = {key}";

        public static string DropTableQuery(string table) => $"DROP TABLE IF
EXISTS {table}";
    }
}
```

Листинг 2 – PostgresClickHouseCRUD.Abstract/IDb.cs

```
using System;

namespace PostgresClickHouseCRUD.Abstract
{
    public interface IDb : IDisposable
    {
        public string TableName { get; set; }

        public void Connect();
```

```
        public void Disconnect();

        public void CreateTable();

        public void CreateOne(int key, int value);

        public void ReadOne(int key);

        public void UpdateOne(int key, int newValue);

        public void DeleteOne(int key);

        public void DropTableIfExists();
    }
}
```

Листинг 3 – PostgresClickHouseCRUD.Abstract/AbstractDb.cs

```
using System.Data;

namespace PostgresClickHouseCRUD.Abstract
{
    public abstract class AbstractDb<TConnection, TCommand> : IDb
        where TConnection : class, IDbConnection, new()
        where TCommand : IDbCommand, new()
    {
        protected AbstractDb(string connectionString, string tableName)
        {
            TableName = tableName;
            Connection.ConnectionString = connectionString;
        }

        private TConnection Connection { get; } = new TConnection();

        public string TableName { get; set; }

        public void Connect()
        {
            Connection.Open();
        }

        public void Disconnect()
        {
            Connection.Close();
        }

        public void CreateTable()
        {
            using var cmd = new TCommand {CommandText = CreateTableQuery(),
Connection = Connection};
            cmd.ExecuteNonQuery();
        }

        public void CreateOne(int key, int value)
        {
            using var cmd = new TCommand {CommandText = CreateOneQuery(key,
value), Connection = Connection};
            cmd.ExecuteNonQuery();
```

```
        }

        public void ReadOne(int key)
        {
            using var cmd = new TCommand {CommandText = ReadOneQuery(key),
Connection = Connection};
            cmd.ExecuteNonQuery();
        }

        public void UpdateOne(int key, int newValue)
        {
            using var cmd = new TCommand {CommandText = UpdateOneQuery(key,
newValue), Connection = Connection};
            cmd.ExecuteNonQuery();
        }

        public void DeleteOne(int key)
        {
            using var cmd = new TCommand {CommandText = DeleteOneQuery(key),
Connection = Connection};
            cmd.ExecuteNonQuery();
        }

        public void DropTableIfExists()
        {
            using var cmd = new TCommand {CommandText = DropTableQuery(),
Connection = Connection};
            cmd.ExecuteNonQuery();
        }

        public void Dispose()
        {
            Connection.Dispose();
        }

        protected abstract string CreateTableQuery();

        protected abstract string CreateOneQuery(int key, int value);

        protected abstract string ReadOneQuery(int key);

        protected abstract string UpdateOneQuery(int key, int newValue);

        protected abstract string DeleteOneQuery(int key);
        protected abstract string DropTableQuery();
    }
}
```

Листинг 4 – PostgresClickHouseCRUD.ClickHouse/ClickHouseAdoDb.cs

```
using ClickHouse.Ado;

using PostgresClickHouseCRUD.Abstract;

namespace PostgresClickHouseCRUD.ClickHouse
{
    public class ClickHouseAdoDb : AbstractDb<ClickHouseConnection,
ClickHouseCommand>
```

```
    {
        public ClickHouseAdoDb(string connectionString, string tableName) :
base(connectionString, tableName)
        {
        }

        protected override string CreateTableQuery() =>
Queries.CreateTableClickHouseQuery(TableName);

        protected override string CreateOneQuery(int key, int value) =>
Queries.CreateOneQuery(TableName, key, value);

        protected override string ReadOneQuery(int key) =>
Queries.ReadOneQuery(TableName, key);

        protected override string UpdateOneQuery(int key, int newValue) =>
            Queries.UpdateOneClickHouseQuery(TableName, key, newValue);

        protected override string DeleteOneQuery(int key) =>
Queries.DeleteOneClickHouseQuery(TableName, key);

        protected override string DropTableQuery() =>
Queries.DropTableQuery(TableName);

        public override string ToString() => "ClickHouse (ClickHouse.Ado)";
    }
}
```

Листинг 5 – PostgresClickHouseCRUD.ClickHouse/ClickHouseOctonicaClientDb.cs

```
using Octonica.ClickHouseClient;

using PostgresClickHouseCRUD.Abstract;

namespace PostgresClickHouseCRUD.ClickHouse
{
    public class ClickHouseOctonicaClientDb : IDb
    {
        public ClickHouseOctonicaClientDb(string connectionString, string
tableName)
        {
            TableName = tableName;
            Connection = new ClickHouseConnection(connectionString);
        }

        protected ClickHouseConnection Connection { get; }

        public string TableName { get; set; }

        public void Dispose()
        {
            Connection.Dispose();
        }

        public void Connect()
        {
```

```
            Connection.Open();
        }

        public void Disconnect()
        {
            Connection.Close();
        }

        public void CreateTable()
        {
            using var cmd = Connection.CreateCommand();
            cmd.CommandText = Queries.CreateTableClickHouseQuery(TableName);
            cmd.ExecuteNonQuery();
        }

        public void CreateOne(int key, int value)
        {
            using var cmd = Connection.CreateCommand();
            cmd.CommandText = Queries.CreateOneQuery(TableName, key, value);
            cmd.ExecuteNonQuery();
        }

        public void ReadOne(int key)
        {
            using var cmd = Connection.CreateCommand();
            cmd.CommandText = Queries.ReadOneQuery(TableName, key);
            cmd.ExecuteNonQuery();
        }

        public void UpdateOne(int key, int newValue)
        {
            using var cmd = Connection.CreateCommand();
            cmd.CommandText = Queries.UpdateOneClickHouseQuery(TableName, key,
newValue);
            cmd.ExecuteNonQuery();
        }

        public void DeleteOne(int key)
        {
            using var cmd = Connection.CreateCommand();
            cmd.CommandText = Queries.DeleteOneClickHouseQuery(TableName, key);
            cmd.ExecuteNonQuery();
        }

        public void DropTableIfExists()
        {
            using var cmd = Connection.CreateCommand();
            cmd.CommandText = Queries.DropTableQuery(TableName);
            cmd.ExecuteNonQuery();
        }

        public override string ToString() => "ClickHouse
(Octonica.ClickHouseClient)";
    }
}
```

Листинг 6 – PostgresClickHouseCRUD.Postgres/PostgresNpgsqlDb.cs

```
using Npgsql;

using PostgresClickHouseCRUD.Abstract;

namespace PostgresClickHouseCRUD.Postgres
{
    public class PostgresNpgsqlDb : AbstractDb<NpgsqlConnection, NpgsqlCommand>
    {
        public PostgresNpgsqlDb(string connectionString, string tableName) :
base(connectionString, tableName)
        {
        }

        protected override string CreateTableQuery() =>
Queries.CreateTableQuery(TableName);

        protected override string CreateOneQuery(int key, int value) =>
Queries.CreateOneQuery(TableName, key, value);

        protected override string ReadOneQuery(int key) =>
Queries.ReadOneQuery(TableName, key);

        protected override string UpdateOneQuery(int key, int newValue) =>
            Queries.UpdateOneQuery(TableName, key, newValue);

        protected override string DeleteOneQuery(int key) =>
Queries.DeleteOneQuery(TableName, key);

        protected override string DropTableQuery() =>
Queries.DropTableQuery(TableName);

        public override string ToString() => "Postgres (Npgsql)";
    }
}
```

Листинг 7 – PostgresClickHouseCRUD.Postgres/PostgresDotConnectExpress Db.cs

```
using Devart.Data.PostgreSql;

using PostgresClickHouseCRUD.Abstract;

namespace PostgresClickHouseCRUD.Postgres
{
    public class PostgresDotConnectExpressDb : AbstractDb<PgSqlConnection,
PgSqlCommand>
    {
        public PostgresDotConnectExpressDb(string connectionString, string
tableName) : base(connectionString,
            tableName)
        {
        }

        protected override string CreateTableQuery() =>
Queries.CreateTableQuery(TableName);

        protected override string CreateOneQuery(int key, int value) =>
```

```
Queries.CreateOneQuery(TableName, key, value);

        protected override string ReadOneQuery(int key) =>
Queries.ReadOneQuery(TableName, key);

        protected override string UpdateOneQuery(int key, int newValue) =>
            Queries.UpdateOneQuery(TableName, key, newValue);

        protected override string DeleteOneQuery(int key) =>
Queries.DeleteOneQuery(TableName, key);

        protected override string DropTableQuery() =>
Queries.DropTableQuery(TableName);

        public override string ToString() => "Postgres
(dotConnect.Express.for.PostgreSQL)";
    }
}
```

Листинг 8 – PostgresClickHouseCRUD.Benchmark/CRUDBenchmark.cs

```
using System;
using System.Diagnostics;
using System.Text.Json;

using PostgresClickHouseCRUD.Abstract;

namespace PostgresClickHouseCRUD.Benchmark
{
    public class CRUDBenchmark
    {
        public CRUDBenchmark(IDb db, int recordCount)
        {
            Db = db;
            RecordCount = recordCount;
        }

        public IDb Db { get; }

        public int RecordCount { get; }

        public RunResult Run()
        {
            var er = 0;
            var r = new RunResult(this);
            while (true)
            {
                try
                {
                    var guid = Guid.NewGuid();
                    Console.WriteLine($"{DateTime.Now} Started {Db}
{RecordCount} {guid} {er}");
                    Db.TableName = $"CRUDBenchmark_{guid.ToString().Replace("-",
"_")}";

                    Db.Connect();
                    CreateTable();
                    r.Create = Create().TotalMilliseconds;
```

```
                r.Read = Read().TotalMilliseconds;
                r.Update = Update().TotalMilliseconds;
                r.Delete = Delete().TotalMilliseconds;
                DropTable();
                Db.Disconnect();

                break;
            }
            catch (Exception e)
            {
                Console.Error.WriteLine(e);
                Console.Error.WriteLine(e.StackTrace);
                if (er++ > 5)
                {
                    break;
                    //throw;
                }
            }
        }

        Console.WriteLine($"{DateTime.Now} {JsonSerializer.Serialize(r)}");

        return r;
    }

    public TimeSpan CreateTable()
    {
        var sw = new Stopwatch();
        sw.Start();
        Db.DropTableIfExists();
        Db.CreateTable();
        sw.Stop();

        return sw.Elapsed;
    }

    public TimeSpan Create()
    {
        var sw = new Stopwatch();
        sw.Start();

        for (var i = 0; i < RecordCount; i++)
        {
            Db.CreateOne(i, i);
        }

        sw.Stop();

        return sw.Elapsed;
    }


    public TimeSpan Read()
    {
        var sw = new Stopwatch();
        sw.Start();
```

```
        for (var i = 0; i < RecordCount; i++)
        {
            Db.ReadOne(i);
        }

        sw.Stop();

        return sw.Elapsed;
    }

    public TimeSpan Update()
    {
        var sw = new Stopwatch();
        sw.Start();
        for (var i = 0; i < RecordCount; i++)
        {
            Db.UpdateOne(i, RecordCount - i);
        }

        sw.Stop();

        return sw.Elapsed;
    }

    public TimeSpan Delete()
    {
        var sw = new Stopwatch();
        sw.Start();
        for (var i = 0; i < RecordCount; i++)
        {
            Db.DeleteOne(i);
        }

        sw.Stop();

        return sw.Elapsed;
    }

    public TimeSpan DropTable()
    {
        var sw = new Stopwatch();
        sw.Start();

        Db.DropTableIfExists();

        sw.Stop();

        return sw.Elapsed;
    }

    public class RunResult
    {
        public RunResult(CRUDBenchmark bench)
        {
            DbmsAndDriverName = bench.Db.ToString();
            RecordCount = bench.RecordCount;
```

```
            }

            public string DbmsAndDriverName { get; }
            public int RecordCount { get; }

            public double Create { get; protected internal set; }

            public double Read { get; protected internal set; }

            public double Update { get; protected internal set; }

            public double Delete { get; protected internal set; }
        }
    }
}
```

Листинг 9 – PostgresClickHouseCRUD.Benchmark/Program.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;

using CsvHelper;

using PostgresClickHouseCRUD.Abstract;
using PostgresClickHouseCRUD.ClickHouse;
using PostgresClickHouseCRUD.Postgres;

namespace PostgresClickHouseCRUD.Benchmark
{
    internal class Program
    {
        private static IEnumerable<CRUDBenchmark> GetBenchmarks(IEnumerable<IDb>
dbs, int launchCount,
            IEnumerable<int> recordsCounts) =>
            Enumerable.Range(0, launchCount)
                .Join(dbs, i => true, db => true, (i, db) => (i, db))
                .Join(recordsCounts, tuple => true, n => true, (tuple, n) => new
{tuple.db, tuple.i, n})
                .Select(t => new CRUDBenchmark(t.db, t.n));

        private static void Main(string[] args)
        {
            using var npgsql =
                new
PostgresNpgsqlDb("Host=localhost;Username=postgres;Password=qwerty;Database=post
gres;Port=15432",
                    "CRUDBenchmark");
            using var dotConnect = new PostgresDotConnectExpressDb(

"Host=localhost;User=postgres;Password=qwerty;Database=postgres;Port=15432",
"CRUDBenchmark");
            using var chAdo = new
ClickHouseAdoDb("Host=127.0.0.1;Port=9000;User=default", "CRUDBenchmark");
            using var chClient = new
ClickHouseClientDb("Host=127.0.0.1;Port=8123;User=default", "CRUDBenchmark");
```

```
            using var chOctonica =
                new
ClickHouseOctonicaClientDb("Host=127.0.0.1;Port=9000;User=default",
"CRUDBenchmark");

            var dblist = new List<IDb> {npgsql, dotConnect, chAdo, chOctonica};
            foreach (var i in dblist)
            {
                i.Connect();
                i.Disconnect();
            }

            var r = new Random();
            var benchlist = GetBenchmarks(dblist, 50, new List<int> {100})
                .Concat(GetBenchmarks(dblist, 10, new List<int> {1000}))
                .Concat(GetBenchmarks(dblist, 3, new List<int> {5000}))
                .Concat(GetBenchmarks(dblist, 10, new List<int> {1000}))
                .Concat(GetBenchmarks(dblist, 3, new List<int> {5000}))
                .Concat(GetBenchmarks(dblist, 500, new List<int> {100}))
                .OrderBy(o => r.Next());

            using var stream = File.Open($"{DateTime.Now:yyyy-MM-dd HH-mm-ss-
ffff}.csv", FileMode.Append);
            using var writer = new StreamWriter(stream);
            using var csv = new CsvWriter(writer,
CultureInfo.GetCultureInfo("ru-ru"));

            csv.Configuration.HasHeaderRecord = false;

            foreach (var b in benchlist)
            {
                var result = b.Run();
                csv.WriteRecords(new List<CRUDBenchmark.RunResult> {result});
            }
        }
    }
}
```