Оглавление

1. Постановка задачи	2
2. Загрузка данных	2
2.1. Объем загруженных данных	2
2.2. Время загрузки данных из WB	2
2.3. Описание структуры загруженных данных	3
2.4. Алгоритм загрузки данных из WB	5
3. Проведение экспериментов	6
3.1. Алгоритм проведения экспериментов	6
3.2. Проведение экспериментов в PostgreSQL с описанием запросов	7
3.3. Проведение экспериментов в SQLite с описанием запросов	14
3.4. Сравнение времени выполнения запросов в PostgreSQL и SQLite	18
4. Заключение	22
5. Список используемой литературы	22
Приложения	23
Приложение 1. DDL сценарий создания таблицы в PostgreSQL и SQLit	e 23
Приложение 2. Программный код загрузки данных	24
Приложение 3. Программный код проведения экспериментов	30

1. Постановка задачи

Создать программу для выгрузки данных из внешнего источника, разобрать и структурировать исходные данные [1], загрузить их в СУБД PostgreSQL [2] и SQLite для проведения исследований [3]. Проанализировать возможности двух СУБД [4] посредством сравнения времени выполнения СRUD запросов к большому объему загруженных данных [5].

2. Загрузка данных

2.1. Объем загруженных данных

В данной работе мы используем 4 базы данных разного размера. Численные значения объема баз данных приведены в Таблице 2.1.

Таблица 2.1 – Объем загруженных данных в СУБД PostgreSQL и SQLite

Наименовани	е базы данных	Количество строк	Объем, МБ
postgres_1	SQLite_1	483 439	134
postgres_2	SQLite_2	966 899	268
postgres_3	SQLite_3	1 933 819	537
postgres_4	SQLite_4	3 867 659	1076

2.2. Время загрузки данных из WB

Время загрузки данных из WB составило 1 час 40 минут. За это время удалось получить 451 147 строк с параметрами товаров маркетплейса, в каждой строке 36 параметров, описывающих конкретный товар (раздел 2.3). Для получения данных использовалось специальное API, которое позволяет получать данные пакетами по 10 000 строк. Это связано с ограничениями, установленными API по количеству данных, которые можно запросить за один раз. Получение, обработка и загрузка каждого блока данных в две СУБД занимает примерно 2 минуты.

2.3. Описание структуры загруженных данных

В результате выгрузки данных из внешнего источника посредством специального API была получена таблица «wb_products», содержащая 36 столбцов. DDL сценарий создания таблицы в СУБД продемонстрирован в Приложении 1. Фрагмент JSON файла, полученного от API, показан на Рисунке 2.1. Краткое описание каждого столбца приведено в Таблице 2.2.

```
"data": {
         "products": [
3
4
             "__sort": 18974,
 5
            "ksort": 0,
 6
             "time1": 5,
7
             "time2": 56,
             "dist": 1443,
9
             "id": 75985699,
10
             "root": 60021157,
11
             "kindId": 0,
12
             "subjectId": 4512,
13
             "subjectParentId": 239,
14
15
             "name": "Фонарь для велосипеда аккумуляторный",
16
             "brand": "Bicycle lights",
             "brandId": 879640,
17
             "siteBrandId": 889640,
18
             "supplierId": 602290,
```

Рисунок 2.1 – Фрагмент полученного JSON файла

Таблица 2.2 – Описание	е столбцов	таблицы «wt	_products»
------------------------	------------	-------------	------------

№	Наименование	Тип	Описание
1	sort	INT	Категория товара
2	ksort	INT	Дополнительная категория товара
3	time1	INT	Минимальное время доставки товара
4	time2	INT	Максимальное время доставки товара
5	dist	INT	Расстояние до города товара
6	id	INT	Уникальный идентификатор товара
7	root	INT	Идентификатор корневого товара
8	kindId	INT	Вид идентификатора товара
9	subjectId	INT	Идентификатор типа товара
10	subjectidParentId	INT	Идентификатор корневого типа товара

Продолжение таблицы 2.2.

№	Наименование	Тип	Описание
11	name	TEXT	Наименование товара
12	brand	TEXT	Бренд товара
13	brandId	INT	Идентификатор бренда товара
14	siteBrandId	INT	Идентификатор сайта бренда товара
15	supplierId	INT	Идентификатор поставщика товара
16	sale	INT	Скидка на товар в процентах
17	priceU	INT	Стоимость товара без скидки в копейках
18	salePriceU	INT	Стоимость товара со скидкой в копейках
19	logisticsCost	INT	Затраты на логистику
20	saleConditions	INT	Условия продажи
21	pics	INT	Количество фотографий товара
22	rating	INT	Рейтинг товара на сайте
23	feedbacks	INT	Количество отзывов на сайте
24	panelPromoId	INT	Идентификатор категории рекламы
25	promoTextCat	TEXT	Текст категории рекламы
26	volume	INT	Количество товара в наличии
27	diffPrice	TEXT	Наличие разных цен на товар
28	colorsName	TEXT	Название цвета товара
29	colorsId	INT	Идентификатор цвета товара
30	sizeName	TEXT	Название размера товара
31	sizesOrigName	TEXT	Оригинальное имя размера товара
32	sizesRank	INT	Ранг размера товара
33	sizesOptionId	INT	Идентификатор опции размера товара
34	sizesWh	INT	Габариты товара
35	sizesSign	TEXT	Обозначение размера товара
36	link	TEXT	Ссылка на страницу товара

2.4. Алгоритм загрузки данных из WB

Последовательность действий при выгрузке данных из внешнего источника посредством специального API приведена на Рисунке 2.2. Программный код загрузки данных показан в Приложении 2.

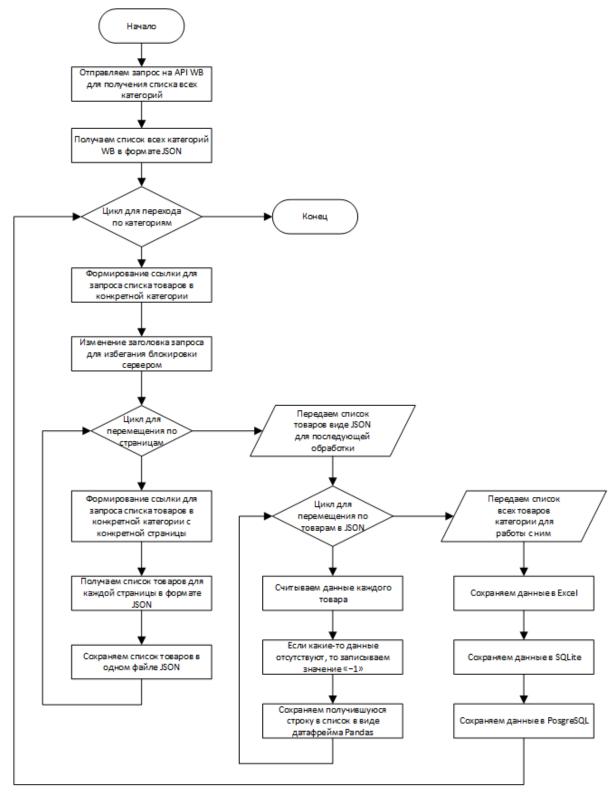


Рисунок 2.2 – Алгоритм загрузки данных из WB

3. Проведение экспериментов

3.1. Алгоритм проведения экспериментов

Последовательность шагов при анализе возможностей двух СУБД посредством выполнения CRUD запросов показана на Рисунке 3.1. Программный код проведения экспериментов представлен в Приложении 3.

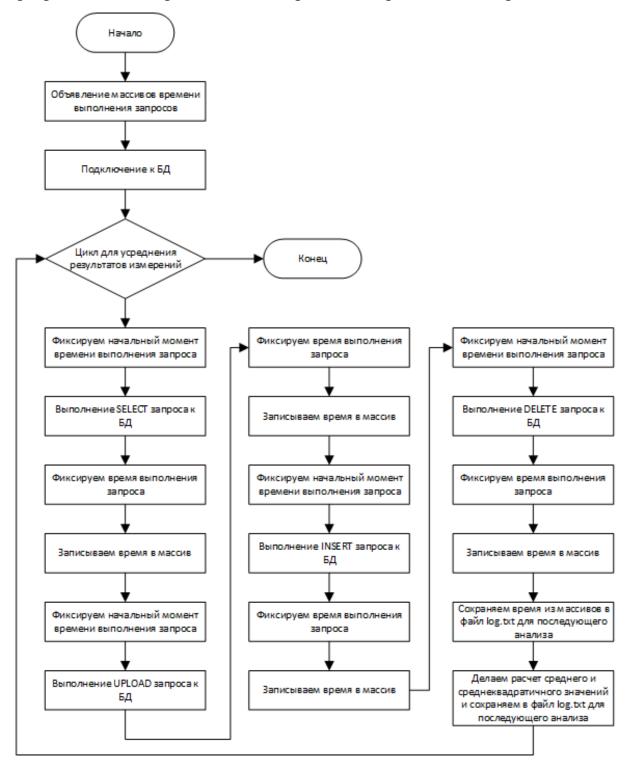


Рисунок 3.1 – Алгоритм проведения экспериментов

3.2. Проведение экспериментов в PostgreSQL с описанием запросов

В рамках эксперимента требуется установить зависимость времени выполнения запросов CRUD к БД разных размеров, для каждого запроса рассчитать среднее и среднеквадратичное отклонение. Используются БД posgres1, posqres2, posqres3, posqres4 (раздел 2.1).

SELECT * FROM wb products

SELECT. Выполняется запрос к базе данных, и выгружаются все строки из таблицы wb_product. Используя встроенную в pgAdmin функцию Explain, получаем полную информацию по запросу Select. Данная функция позволяем получить графическую визуализацию запроса, аналитику физического плана и некоторые статистические параметры. Результаты выполнения Explain для запроса представлены на Рисунке 3.2.

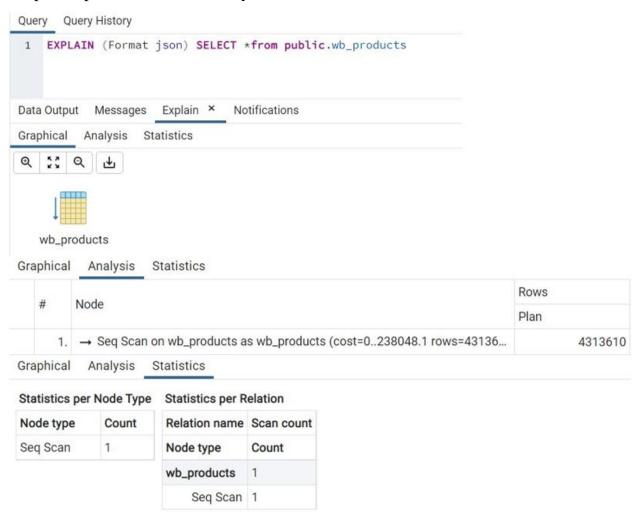


Рисунок 3.2 – Выполнение Explain для запроса Select

UPDATE wb products SET dist = 100 WHERE dist = 99

UPDATE. Используя встроенную в pgAdmin Explain функцию, получаем полную информацию по запросу Update. Результаты запроса представлены на Рисунке 3.3.

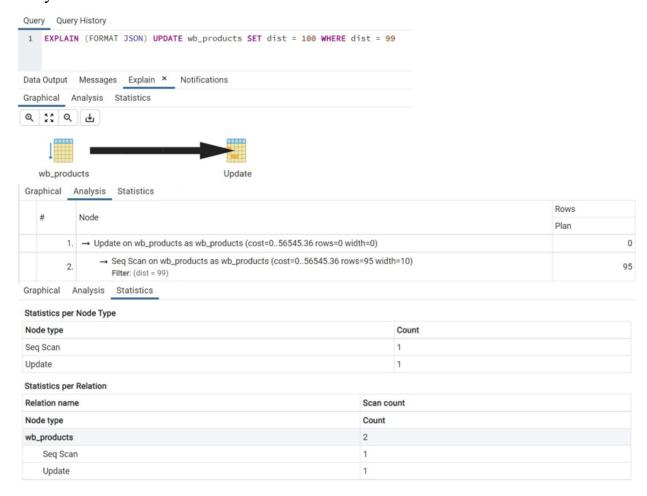


Рисунок 3.3 – Выполнение Explain для запроса Update

INSERT into wb_products values (110099, 525755, 43, 31, 5566, 51446, 175520, 332, 41, 12, 'Блузка офисная для девочки школьная', 'Camicia', 822269, 832269, 373633, 68, 265000, 84800, 0, 0, 14, 5, 1276, 159147, 'ХИТЫВЕСНЫ', 32, '0', 'белый', 16777215, '42', '42', 166651, 20080, 120762, 'jf7LEXB15Ze71c5hNa1Z2KnZyc4=',

^{&#}x27;https://www.wildberries.ru/catalog/110665146/detail.aspx?targetUrl=BP')

INSERT. Используя встроенную в pgAdmin функцию Explain, получаем полную информацию по запросу Insert. В данном запросе требуется указать данные для всех 36 столбцов таблицы, описанных в Разделе 2.3. Результаты запроса представлены на Рисунке 3.4.

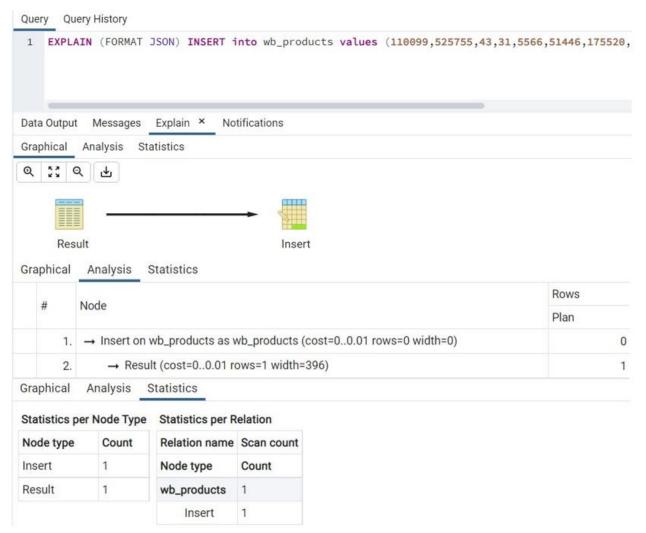


Рисунок 3.4 – Выполнение Explain для запроса Insert

DELETE FROM wb_products WHERE id=80012708

DELETE. Используя встроенную в pgAdmin функцию Explain, получаем полную информацию по запросу Delete. Результаты запроса представлены на Рисунке 3.5.

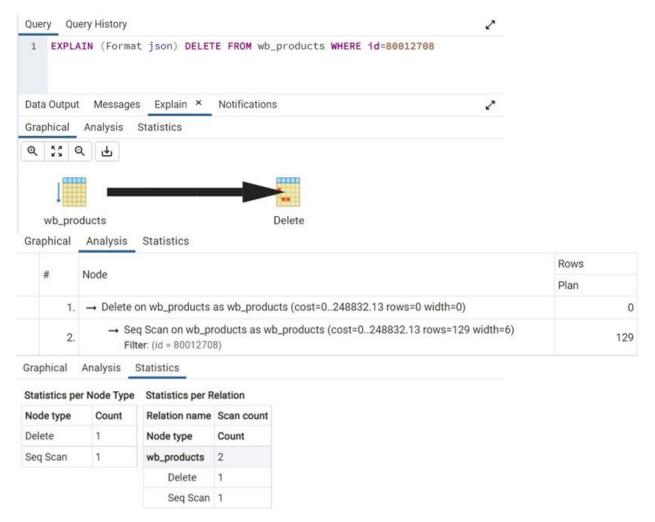


Рисунок 3.5 – Выполнение Explain для запроса Delete

Далее в соответствии с алгоритмом проведения экспериментов, описанном в Разделе 3.1, выполняем 20 CRUD запросов к каждой из двух СУБД для расчета среднего значения и среднеквадратичного отклонения.

Результат выполнения 20 запросов Select к СУБД PostgreSQL показан в Таблице 3.1 и визуализирован на Рисунке 3.6.

Таблица 3.1 – Результат выполнения 20 запросов Select в PostgreSQL

№ Select	postgres1	postgres2	postgres3	postgres4
1	1,28805542	2,596428871	4,628715992	8,907176495
2	1,188027143	2,519071817	4,633093357	11,77524018
3	1,135818481	2,469276905	4,685551167	9,022065878
4	1,16257906	2,667906046	4,546559095	8,970579863
5	1,358062506	2,415505409	4,782265186	8,91914773

Продолжение таблицы 3.1.

№ Select	postgres1	postgres2	postgres3	postgres4
6	1,150325537	2,400585651	4,807919264	8,809303284
7	1,127218008	2,415766478	4,590075731	9,069332123
8	1,145205021	2,460457325	4,522244215	9,291076422
9	1,173143148	2,605240345	4,535361528	8,969914436
10	1,138268471	2,493346453	4,524473906	9,092027187
11	1,19404006	2,418659687	4,509439945	9,087626219
12	1,133038521	2,404831409	4,529818535	8,938804626
13	1,139374495	2,404574871	4,506833315	9,220055342
14	1,161930323	2,449455738	4,473920107	9,351839542
15	1,178367376	2,406604052	4,481939793	9,035997391
16	1,178851366	2,656934977	4,478961229	8,801584482
17	1,170704842	2,412128448	4,470255613	9,122145176
18	1,238681555	2,452447414	4,429638624	8,950104952
19	1,177023649	2,42353344	4,492684126	8,88197422
20	1,158051729	2,381591797	4,447512388	8,918031216

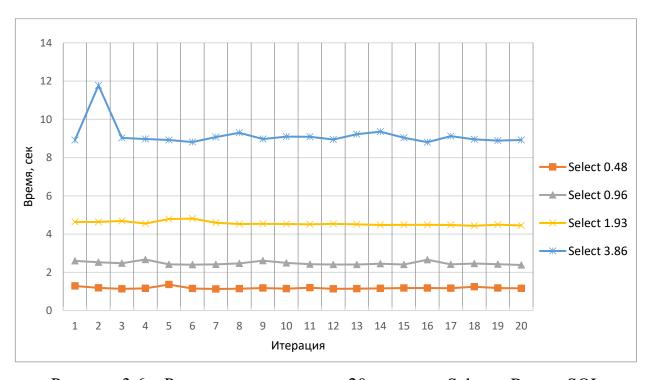


Рисунок 3.6 – Результат выполнения 20 запросов Select в PostgreSQL

С увеличением количества строк в СУБД PostgreSQL увеличивается и время выполнения запроса Select. Расчеты среднего времени выполнения СRUD запросов показаны в Таблице 3.2 и на Рисунке 3.7, а расчеты среднеквадратичного отклонения времени выполнения запросов продемонстрированы в Таблице 3.3 и на Рисунке 3.8.

Как описано ранее в Разделе 2.1:

- количество строк в БД postgres_1 483 439;
- количество строк в БД postgres_2 966 899;
- количество строк в БД postgres_3 1 933 819;
- количество строк в БД postgres_4 3 867 659.

Таблица 3.2 – Среднее время выполнения CRUD запросов в PostgreSQL

	postgres1	postgres2	postgres3	postgres4
SELECT	1,179838336	2,472717357	4,553863156	9,156701338
UPDATE	0,250711727	0,575242329	0,896314692	3,074826372
DELETE	0,183049989	0,405863261	0,503250325	0,959520447
INSERT	0,010333824	0,002795243	0,007140672	0,004440761

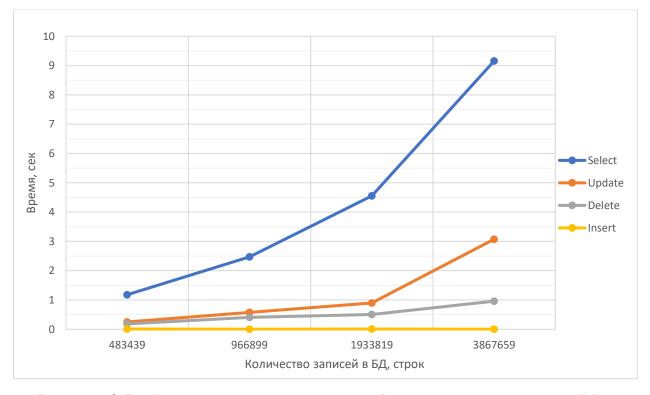


Рисунок 3.7 – Среднее время выполнения CRUD запросов в PostgreSQL

Таблица 3.3 – Среднеквадратичное отклонение времени выполнения CRUD запросов для СУБД PostgreSQL

	postgres1	postgres2	postgres3	postgres4
SELECT	0,00499819	0,020905637	0,024398539	0,054754681
UPDATE	0,001152968	0,017804095	0,028019635	0,201439186
DELETE	0,002124666	0,005974271	0,005247237	0,003881541
INSERT	0,002062634	0,001912584	0,00084331	0,000451901

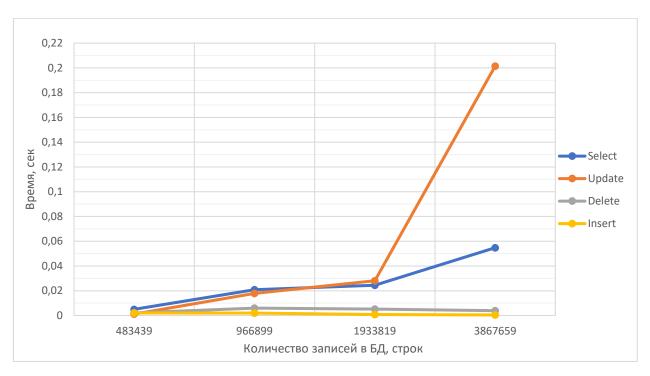


Рисунок 3.8 — Среднеквадратичное отклонение времени выполнения CRUD запросов для СУБД PostgreSQL

Исходя из роста функции среднего значения времени выполнения запросов Select и Update (рисунок 3.7) и среднеквадратичного отклонения (рисунок 3.8), мы можем сделать вывод, что чем больше количество строк в БД, тем больше времени требуется для выполнения запросов Select и Update.

Delete и Insert выполняются очень быстро, и изменения, связанные с увеличением базы данных, не существенно влияют на скорость выполнения запросов (рисунок 3.8).

3.3. Проведение экспериментов в SQLite с описанием запросов

В результате экспериментов в СУБД SQLite с требуется установить зависимость времени выполнения запросов при работе с SQLite CRUD к БД разных размеров, для каждого запроса рассчитать среднее и среднеквадратичное отклонение. Для эксперимента используются БД SQLite_1, SQLite_2, SQLite_3, SQLite_4 (раздел 2.1). Используемые запросы точно такие же, как в Разделе 3.2.

Далее в соответствии с алгоритмом проведения экспериментов, описанном в Разделе 3.1, выполняем 20 CRUD запросов к каждой из двух СУБД для расчета среднего значения и среднеквадратичного отклонения.

Как описано ранее в Разделе 2.1:

- количество строк в БД SQLite_1 483 439;
- количество строк в БД SQLite_2 966 899;
- количество строк в БД SQLite_3 1 933 819;
- количество строк в БД SQLite_4 3 867 659.

Результат выполнения 20 запросов Update к СУБД SQLite приведен в Таблице 3.4 и продемонстрирован на Рисунке 3.9. Используется запрос UPDATE wb_products SET dist = 100 WHERE dist = 99.

Таблица 3.4 – Результат выполнения 20 запросов UPDATE в СУБД SQLite

№ Update	SQLite_1	SQLite_2	SQLite_3	SQLite_4
1	0,144148588	0,563112974	1,089136628	1,147135258
2	0,183686495	0,646323204	1,094427347	2,963939667
3	0,136494398	0,301933527	0,582471371	3,009844303
4	0,154922485	0,339317799	0,607671022	2,854712248
5	0,173482656	0,340688229	1,067941904	2,674207211
6	0,165002346	0,318427563	1,123900175	2,344498634
7	0,157465935	0,326195955	1,17198348	2,504748344

Продолжение таблицы 3.4.

№ Update	SQLite_1	SQLite_2	SQLite_3	SQLite_4
8	0,153924227	0,316440582	1,107788801	2,270922422
9	0,15418601	0,315835953	0,664513111	1,159860849
10	0,206473589	0,324757576	0,644913673	1,129983425
11	0,187879086	0,339012146	0,614657879	1,18343401
12	0,13750267	0,303218842	0,614030838	1,212903738
13	0,171632528	0,323848248	0,600276947	1,240617275
14	0,187623501	0,313157797	0,592596054	1,30702734
15	0,148228168	0,312974453	0,605332136	1,160255909
16	0,139810801	0,310529232	0,637218237	1,212408066
17	0,142521143	0,621321678	0,611548901	1,145684481
18	0,188092947	0,569136381	1,148563862	1,217784166
19	0,141818047	0,55821228	1,028506041	1,154778719
20	0,143033504	0,669199705	1,120984554	1,158035517

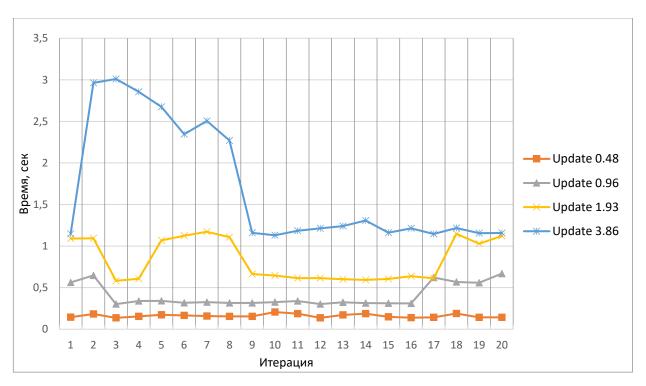


Рисунок 3.9 – Результат выполнения 20 запросов UPDATE в СУБД SQLite

С увеличением количества строк в СУБД SQLite увеличивается и время выполнения запроса UPDATE. Расчеты среднего времени выполнения CRUD запросов показаны в Таблице 3.5 и на Рисунке 3.10, а расчеты среднеквадратичного отклонения времени выполнения запросов продемонстрированы в Таблице 3.6 и на Рисунке 3.11.

Таблица 3.5 – Среднее время выполнения CRUD запросов в СУБД SQLite

	SQLite_1	SQLite_2	SQLite_3	SQLite_4
SELECT	0,000053138	0,000156999	0,000101864	0,000098162
UPDATE	0,160896456	0,405682206	0,836423147	1,702639079
DELETE	0,167783177	0,408836341	0,891940571	1,567275655
INSERT	0,017631091	0,015506065	0,016887665	0,012220359

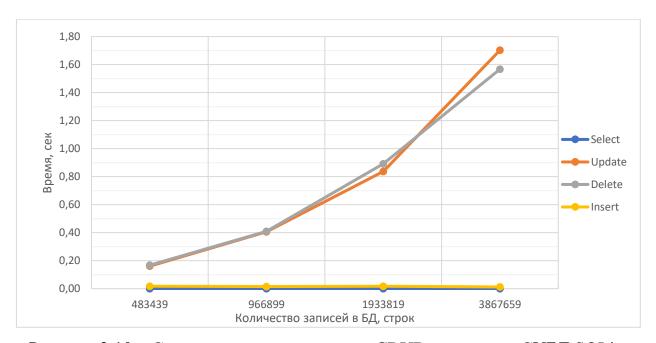


Рисунок 3.10 – Среднее время выполнения CRUD запросов в СУБД SQLite

Таблица 3.6 – Среднеквадратичное отклонение времени для СУБД SQLite

	SQLite_1	SQLite_2	SQLite_3	SQLite_4
SELECT	0,000012236	0,000036154	0,000023412	0,000022518
UPDATE	0,004098042	0,060455062	0,065282864	0,124940626
DELETE	0,006559406	0,038146823	0,050134814	0,067431881
INSERT	0,002358979	0,000855617	0,000988429	0,000482925

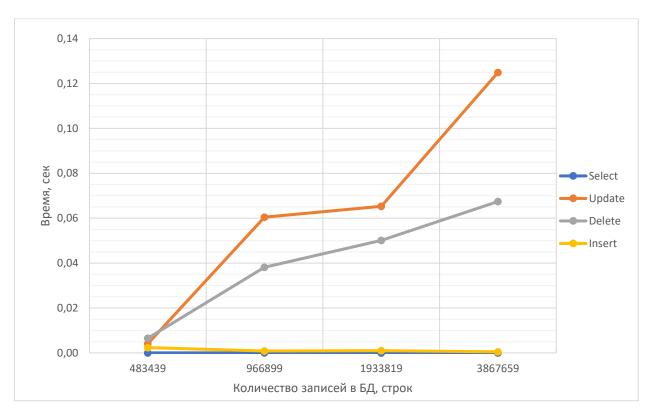


Рисунок 3.11 — Среднеквадратичное отклонение времени выполнения CRUD запросов для СУБД SQLite

Исходя из роста функции среднего значения времени выполнения запросов Delete и Update (рисунок 3.10) и среднеквадратичного отклонения (рисунок 3.11), мы можем сделать вывод, что чем больше количество строк в БД, тем больше времени требуется для выполнения запросов Delete и Update.

Select и Insert выполняются очень быстро, и изменения, связанные с увеличением базы данных, не существенно влияют на скорость выполнения запросов (рисунок 3.11).

3.4. Сравнение времени выполнения запросов в PostgreSQL и SQLite

Сравним скорость выполнения запросов в PostgesSQL и SQLite по значению среднеквадратичного отклонения запросов CRUD для четырех баз данных с разным количеством строк (раздел 2.1). Для сравнения скорости запросов сделаем 4 таблицы (таблицы 3.7-3.10) и визуализируем графики среднеквадратичного отклонения для 4 запросов (рисунки 3.12-3.15).

Таблица 3.7 — Среднее время выполнения и среднеквадратичное отклонение времени выполнения в СУБД PostgreSQL и SQLite запроса Select

SELECT	483 439 966 899 1 933 819 3 867								
	Средн	ее время выпо.	лнения						
PostgreSQL	1,179838336	2,472717357	4,553863156	9,156701338					
SQLite	0,000053138	0,000156999	0,000101864	0,000098162					
Сред	неквадратично	е отклонение в	времени выпол	нения					
PostgreSQL	0,00499819	0,020905637	0,024398539	0,054754681					
SQLite	0,000012236	0,000036154	0,000023412	0,000022518					



Рисунок 3.12 — Среднеквадратичное отклонение времени выполнения в СУБД PostgreSQL и SQLite запроса Select

Из Рисунка 3.12 видно, что SELECT запрос в SQLite выполняется существенно быстрее. И разница будет еще сильнее заметна на большем количестве строк, так как с увеличением базы данных растет время запроса.

Таблица 3.8 — Среднее время выполнения и среднеквадратичное отклонение времени выполнения в СУБД PostgreSQL и SQLite запроса Update

UPDATE	483 439 966 899 1 933 819 3 867							
	Средн	ее время выпо	лнения					
PostgreSQL	0,250711727	0,575242329	0,896314692	3,074826372				
SQLite	0,160896456	0,405682206	0,836423147	1,702639079				
Сред	неквадратично	ое отклонение і	времени выпол	нения				
PostgreSQL	0,001152968	0,017804095	0,028019635	0,201439186				
SQLite	0,004098042	0,060455062	0,065282864	0,124940626				

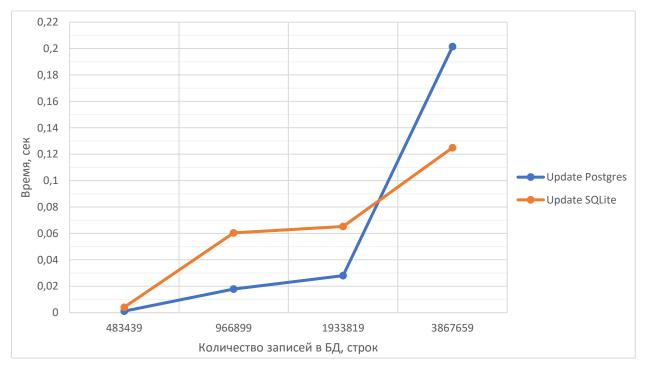


Рисунок 3.13 — Среднеквадратичное отклонение времени выполнения в СУБД PostgreSQL и SQLite запроса Update

Из Рисунка 3.13 видно, что Update запрос в SQLite и PostgreSQL выполняется, примерно одинаково. И с увеличением БД растет время запроса.

Таблица 3.9 — Среднее время выполнения и среднеквадратичное отклонение времени выполнения в СУБД PostgreSQL и SQLite запроса Delete

DELETE	483 439 966 899 1 933 819 3 86								
	Средн	ее время выпо	лнения						
PostgreSQL	0,183049989	0,405863261	0,503250325	0,959520447					
SQLite	0,167783177	0,408836341	0,891940571	1,567275655					
Сред	неквадратично	ое отклонение і	времени выпол	нения					
PostgreSQL	0,002124666	0,005974271	0,005247237	0,003881541					
SQLite	0,006559406	0,038146823	0,050134814	0,067431881					

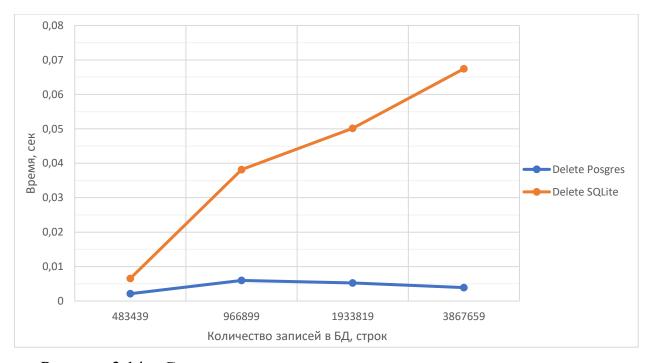


Рисунок 3.14 — Среднеквадратичное отклонение времени выполнения в СУБД PostgreSQL и SQLite запроса Delete

Из Рисунка 3.14 видно, что с увеличением БД время запроса растет в SQLite, а в PostgresSQL уменьшается или остаётся почти неизменным. Такая необычная зависимость может быть объяснена наличием оптимизирующего алгоритма в PostgresSQL для запроса Delete.

Таблица 3.10 — Среднее время выполнения и среднеквадратичное отклонение времени выполнения в СУБД PostgreSQL и SQLite запроса Insert

INSERT	483 439 966 899 1 933 819 3 867									
	Средн	ее время выпо	лнения							
PostgreSQL	0,010333824	0,002795243	0,007140672	0,004440761						
SQLite	0,017631091	0,015506065	0,016887665	0,012220359						
Сред	неквадратично	ое отклонение і	времени выпол	нения						
PostgreSQL	0,002062634	0,001912584	0,00084331	0,000451901						
SQLite	0,002358979	0,000855617	0,000988429	0,000482925						

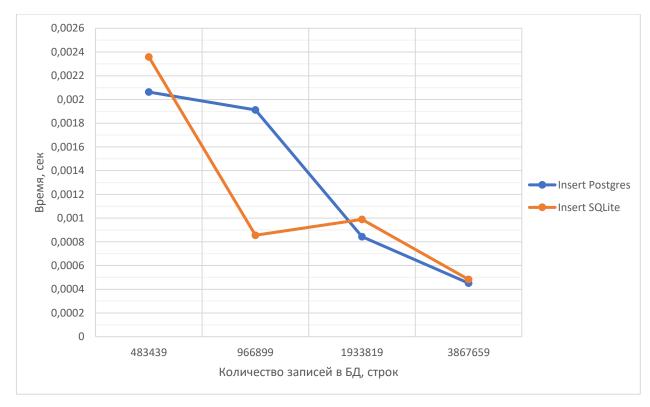


Рисунок 3.15 — Среднеквадратичное отклонение времени выполнения в СУБД PostgreSQL и SQLite запроса Insert

Из Рисунка 3.15 видно, что Insert запрос в SQLite и PostgreSQL выполняется, примерно одинаково. И с увеличением БД уменьшается время запроса благодаря оптимизации на большом количестве записываемых в СУБД данных.

4. Заключение

Создали программу для выгрузки данных из внешнего источника, разобрали и структурировали исходные данные, загрузили их в СУБД PostgreSQL и SQLite для проведения исследований. Проанализировали возможности двух СУБД посредством сравнения времени выполнения CRUD запросов к большому объему загруженных данных.

5. Список используемой литературы

- 1. Виноградов В., Виноградова М. В. Постреляционные модели данных и языки запросов / Москва: Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет), 2017. 100 с. ISBN 978-5-7038-4283-6. EDN XFIMAW.
- 2. Мангушева А.Р. Базы данных на СУБД PostgreSQL / Казань: Общество с ограниченной ответственностью "Редакционно-издательский центр "Школа", 2020. 93 с. ISBN 978-5-00162-258-1. EDN ACMIUL.
- 3. Голдовский Я.М. Введение в постреляционные базы данных: учебное пособие для студентов, обучающихся по направлению "Информатика и вычислительная техника" по дисциплине "Постреляционные базы данных" / Московский гос. ун-т путей сообщ. (МИИТ), Ин-т упр. и информ. технологий, Каф. "Вычислительные системы и сети". Москва: МИИТ, 2008. 92 с. EDN QMTPUH.
- 4. Парфенов Ю.П. Постреляционные хранилища данных: учебное пособие / Министерство образования и науки Российской Федерации, Уральский федеральный университет. Екатеринбург: Издательство Уральского университета, 2016. 120 с. ISBN 978-5-7996-1827-8. EDN WYZTQX.
- 5. Григорьев Ю.А., Плутенко А.Д., Плужникова А.Ю. Реляционные базы данных и системы NoSQL / Благовещенск: Амурский государственный университет, 2018. 424 с. ISBN 978-5-93493-308-2. EDN XTSEKL.

Приложения

Приложение 1. DDL сценарий создания таблицы в PostgreSQL и SQLite

Таблица товаров маркетплейса «wb_products» состоит из 36 столбцов, описанных в Разделе 2.3. Для создания таблицы в СУБД PostgreSQL и SQLite выполняется одинаковый DDL сценарий:

CREATE TABLE IF NOT EXISTS wb_products (sort INT NOT NULL, ksort INT NOT NULL, time1 INT NOT NULL, time2 INT NOT NULL, dist INT NOT NULL, id INT NOT NULL, root INT NOT NULL, kindId INT NOT NULL, subjectId INT NOT NULL, subjectParentId INT NOT NULL, name TEXT NOT NULL, brand TEXT NOT NULL, brandId INT NOT NULL, siteBrandId INT NOT NULL, supplierId INT NOT NULL, sale INT NOT NULL, priceU INT NOT NULL, salePriceU INT NOT NULL, logisticsCost INT NOT NULL, saleConditions INT NOT NULL, pics INT NOT NULL, rating INT NOT NULL, feedbacks INT NOT NULL, panelPromoId INT NOT NULL, promoTextCat TEXT NOT NULL, volume INT NOT NULL, diffPrice TEXT NOT NULL, colorsName TEXT NOT NULL, colorsId INT NOT NULL, sizeName TEXT NOT NULL, sizesOrigName TEXT NOT NULL, sizesRank INT NOT NULL, sizesOptionId INT NOT NULL, sizesWh INT NOT NULL, sizesSign TEXT NOT NULL, link TEXT NOT NULL);

Приложение 2. Программный код загрузки данных

Для реализации программы загрузки данных из внешнего источника в СУБД PostgreSQL и SQLite был использован язык программирования Python. В программе использовалось 7 библиотек: random, tqdm, pandas, psycopg2, requests, sqlite3, json

Функция get_catalogs_wb получает все каталоги WB с помощью запроса к специальному API. После получения информации функция сохраняет данные в JSON файл для дальнейшей обработки.

```
url = 'https://www.wildberries.ru/webapi/menu/main-menu-ru-ru.json'
headers = {'Accept': "*/*", 'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"}
  response = requests.get(url, headers=headers).json()
  with open('wb_catalogs_data.json', 'w', encoding='UTF-8') as file:
    json.dump(response, file, indent=2, ensure ascii=False)
  data list = []
  for d in response:
    try:
       for child in d['childs']:
         if target == child['url']:
            data_list.append({
               'category_name': child['name'],
               'category_url': child['url'],
               'shard': child['shard'],
               'query': child['query']})
          else:
            for sub_child in child['childs']:
               data_list.append({
                 'category_name': sub_child['name'],
                 'category url': sub child['url'],
                 'shard': sub_child['shard'],
                 'query': sub_child['query']})
    except:
       # print(f'не имеет дочерних каталогов *{d["name"]}*')
       continue
  return data_list
```

Функция search_caterogory_in_catalog ищет совпадение категории со списком из файла с каталогами (рисунок 4). Это необходимо, чтобы отправлять серверу запросы только к существующим спискам товаров и избегать блокировки сервером из-за некорректного запроса:

```
try:
for catalog in catalog_list:
```

```
if catalog['category_url'] == target:
    print(f'найдено совпадение: {catalog["category_name"]}')
    name_category = catalog['category_name']
    shard = catalog['shard']
    query = catalog['query']
    return name_category, shard, query
    else:
        # print('нет совпадения')
        pass
except:
    print('Данный раздел не найден!')
```

Функция get_data_from_json забирает данные из JSON файла с данными о товарах и сохраняет их в список data_list для последующей работы с ними в программе. Также в этой функции реализован поиск исключений, когда отсутствует какой-либо параметр товара.

```
data list = []
  for data in json_file['data']['products']:
        price = int(data["priceU"] / 100)
     # except:
     # price = 0
     # print (data)
     try:
       datatime2 = data['time2']
     except KeyError:
       datatime2 = f'-1'
     try:
       datatime1 = data['time1']
     except KeyError:
       datatime1 = f'-1'
     try:
       datadist = data['dist']
     except KeyError:
       datadist = f'-1'
     try:
       datapanelPromoId = data['panelPromoId']
       datapromoTextCat = data['promoTextCat']
     except KeyError:
       datapanelPromoId = f'-1'
       datapromoTextCat = f'-1'
     try:
       datacolorsName = data['colors'][0]['name']
       datacolorsid = data['colors'][0]['id']
     except IndexError:
       datacolorsName = f'-1'
       datacolorsid = f'-1'
     data_list.append({
          '__sort': data['__sort'],
```

```
'time1': datatime1,
        'time2': datatime2,
        'dist': datadist,
        'id': data['id'],
        'root': data['root'],
        'kindId': data['kindId'],
        'subjectId': data['subjectId'],
        'subjectParentId': data['subjectParentId'],
        'Наименование': data['name'],
        'Бренд': data['brand'],
        'id бренда': data['brandId'],
        'siteBrandId': data['siteBrandId'],
        'supplierId': data['supplierId'],
        'Скидка': data['sale'],
        'priceU': data['priceU'],
        'salePriceU': data['salePriceU'],
        'logisticsCost': data['logisticsCost'],
        'saleConditions': data['saleConditions'],
        'pics': data['pics'],
        'rating': data['rating'],
        'feedbacks': data['feedbacks'],
        'panelPromoId': datapanelPromoId,
        'promoTextCat': datapromoTextCat,
        'volume': data['volume'],
        'diffPrice': data['diffPrice'],
        'colors_name': datacolorsName,
        'colors id': datacolorsid,
        'sizes_name': data["sizes"][0]["name"],
        'sizes_origName': data['sizes'][0]['origName'],
        'sizes_rank': data['sizes'][0]['rank'],
        'sizes_optionId': data['sizes'][0]['optionId'],
        'sizes_wh': data['sizes'][0]['wh'],
        'sizes_sign': data['sizes'][0]['sign'],
        'Ссылка': f'https://www.wildberries.ru/catalog/{data["id"]}/detail.aspx?targetUrl=BP'
   })
return data_list
```

Функция get_content отправляет запрос к серверу, получает список товаров с параметрами в конкретной категории и сохраняет их в JSON файл для последующей работы с ними. Также в этой функции реализован обход ограничений сервера на огромное количество однотипных запросов с помощью изменения заголовка запроса на заголовки самых популярных браузеров.

```
'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36',
```

'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/602.2.14 (KHTML, like Gecko) Version/10.0.1 Safari/602.2.14',

'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36',

'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36',

'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36',

'Mozilla/5.0 (Windows NT 10.0; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0'] headers = {'User-Agent': choice(desktop_agents), 'Accept':

'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'} data_list = []

for page in tqdm(range(1, 101)):

```
# url = f'https://wbxcatalog-ru.wildberries.ru/{shard}' \
```

- # f/catalog?appType=1&curr=rub&dest=-1029256,-102269,-1278703,-1255563'\
- # f'&{query}&lang=ru&locale=ru&sort=sale&page={page}'\
- # f'&priceU={low_price * 100};{top_price * 100}'

f'&locale=ru&page={page}&priceU={1 * 100};{1000000 * 100}'\

f'®=0®ions=64,83,4,38,80,33,70,82,86,30,69,1,48,22,66,31,40&sort=popular&spp=0&{q uery}'

```
data = requests.get(url, headers=headers).json()
if len(get_data_from_json(data)) > 0:
    data_list.extend(get_data_from_json(data))
else:
    print(f'Сбор данных завершен.')
    break
# print (data_list)
return data_list
```

Функция save_database сохраняет данные из списка товаров в постоянное хранилище СУБД SQLite и СУБД PostgreSQL для последующей работы с ними

Функция parser_all выполняет сбор данных и сохранение товаров из всего списка каталогов, используя функции, описанные выше.

```
target = url
  catalog_list = get_catalogs_wb(target)
  try:
    # поиск введенной категории в общем каталоге
    name_category, shard, query = search_category_in_catalog(target, catalog_list=catalog_list)
    # сбор данных в найденном каталоге
                       get_content(shard=shard,
    data list
                                                                      low_price=low_price,
                                                    query=query,
top_price=top_price)
    # сохранение найденных данных
    #save_excel(data_list, f'{name_category}_from_{low_price}_to_{top_price}')
    save_database(data_list, conSQLite, conPostgreSQL)
  except TypeError:
    print('Ошибка! Возможно не верно указан раздел. Удалите все доп фильтры с ссылки')
  except PermissionError:
    print('Ошибка! Вы забыли закрыть созданный ранее excel файл. Закройте и повторите
попытку')
```

Код основной функции, в ней происходит подключение к базам данных PostgreSQL и SQLite и создание таблицы с помощью DDL сценария из Приложения 1 (рисунок 10).

```
conSQLite = sqlite3.connect("PBD_DZ_SQLite.db")
  conPostgreSQL = psycopg2.connect(user="postgres",
                     password="1234",
                     host="127.0.0.1",
                     port="5432",
                     database="PBD_DZ_PostgreSQL")
  Create wb products = "'CREATE TABLE IF NOT EXISTS wb products (...);"
  conSQLite.cursor().execute(Create_wb_products)
  conSQLite.commit()
  conPostgreSQL.cursor().execute(Create_wb_products)
  conPostgreSQL.commit()
  low_price = 1
  top\_price = 500000
  with open('wb_catalogs_data.json', 'r', encoding='UTF-8') as json_file:
    data = json.load(json_file)
    data list = []
    listID = [128313, 128604, 9156, 62057]
    for d in data:
```

```
try:
        if d['id'] not in listID:
          for child in d['childs']:
             try:
                for child1 in child['childs']:
                  try:
                     if child1['id'] not in listID:
                        for child2 in child1['childs']:
                             for child3 in child2['childs']:
                                data_list.append(
                                  child3['url'])
                          except:
                             data_list.append(child2['url'])
                  except:
                     data_list.append(child1['url'])
             except:
                data_list.append(child['url'])
     except:
        continue
for item in data_list:
  print(item)
  parser all(item, low price, top price, conSQLite, conPostgreSQL)
conSQLite.close()
conPostgreSQL.close()
```

Также в основной функции запускается сбор данных из всех каталогов, которые удалось получить от Wildberries на прошлых этапах. При получении данных из каталогов требуется погружаться на несколько уровней по иерархии в связи с глубокой вложенностью структуры JSON файла.

В результате работы программного кода загрузки данных в обе СУБД было записано 451 147 строк с 36 параметрами в каждой строке (рисунок 12).

(P)	d police	b products/	190 07 199	tarcou. q	eostarcser Pt	95901050	. 8																			
p40.15	m 8 -	A T	v Seint	× 1	m • v	F3 F5	· 55	75 III-v	0																	
	Outry Out																			∠ soutch	OM V					
																				2. 90.5101	100					
* Cwtwlege	1 SELECT	- FROM pu	bine.wb_pr	whets																						
Credittippes	Data Output	Messages	Notification	5																						
Estendora Foreign Data Weappers	5 G v	n a s	2 "																							
Languages			. tere!	. 271	e2 - de		м .	not .	kendel	. March	adjopment	race.	bred	a books a six	drontid a d	unded . re		eices .	saleptow _A legistourse _A	plecodrine	. 60	. neira	. Sectoris .	prespondi a promovers	. volume . Offers	e a colorar
Pakkantany	Med	a my		. 100	die No		Mayor	parks.	lands.						- 1	riager of	202	Kega	rioge rioge	nioger	100,50	rioger	Hope	HOUSE NO.	11093 110	
Schwings (1)			98/7	4	21	506	157752800	11/04/599		2 4		Signal progression ranges obvious forecast on	CASH UH THE VOCN	7325155	1305456	119084	- 4	40,000	105/03 0		0	12	5 759			00131
· public	2		5297	4	21	504	22195000	16661745		2 13		Pytean megals	UN In Style	72353	60050	2500	70	6668330	130000 0		0	9	5 1759		12 504	
> N Aggregates	8		1817/8		91	006	38793676			2 18		Pyrouso sevo sor securalis	039339900	755934	285004	83420	90	1293790	108403 0		0		5 1550			OCF 8
3 Spilators	4		9215	4	21	500	60012706			2 10		Pythania sosposis facusar	CharChagason	15563	25563	100605	- 00	609330	121603 0		0		5 567			Outu
3 % Borrain	1		900	4	21	126	100701392			2 4		Барага экинова пракушеная с даленны руковом офизный сака-	MOMETY	474485	DICCES	471261	76	455000	196723 0		0	19	5 (3)			1954
> D MEConfiguration	9		1906	*	21	50.6	42772000			2 19		Pytauro xerocol existiny asoptails former edisorer granical	Middle Tee RESTRATOR	466271	900073	340003	81	102100	142303 0			23	4 1218			9033
> N FIEUchingies	7		7960	4	21	1987	90005546	75465195		2 13		Pyterac coeposits remove coming greener States observed and are consumed.	Garrier	91110	800064	930670 829669	- 69	626000	195303 0 95603 0		0	12	5 1593			1901
3 ALP REPAREN	8		2504		6	1997				, ,							- 10	52 (22)	9201 0 18001 0		0	12	5 750			
3 R FEE tomplates	7		17070	*	21	554	121511146	100423119		2 112		E CONTRACTOR TO MECANISME	AND DEVID	1022317	515042	379603 1193727	- 20	204300	190403 0			14	5 790			1691
3 @ Foreign rables	10			4	21							боди неносии с дляным рукавам переси прадустное будах		200				26 4830			0	13				1901
> W Functions > D Materialized virus	12		1900		81	556	35791308			2 18		Prouse deserve consequente renal	PARAMETER RESONATION	60027	960871	940001 400001	81	(55.00	14(81) 0		0	18	4 1010		60 1986	900
3 % German	12		16300	4	21	558	100971972	500011187		2 4		 В близ жиможи пригодителя с Ятинетел эфомом офиситу с игр. 	Manager	620335	640008	401001 630003	70	165000	108703 0		0	12	5 193			(25)
) © Properties	13		990	4	21							Digital services (and objected religiods (1971) read symbols				41900		244930	19601 0		0	15				
	15		3456		31	506	7/31/36/04			2 16		Python metals strong	Dr. cart	912/4	150041	41900	- 4	659300	19703 0 9703 0			14	5 340			Gr.
w Tobles (2)	15		8975	- 1	- 11	500	16163985	29706232		2 19				002158	217358	2200	70	00000	11763 0		0	11	5 711			(41)
> El salumente			2770			100	133342415	11/041220		, ,		Рускию некорог опросей в обиская будках менера продолжное будках разделения народня ублокия болог ческог	LACY OF THE MOON	1323133	170400	119004	70	42200	100/22 0				2 222			7921
> E ub_switchcopy	17		2007	-1	- 2	900	157792315	7011149				Employees response a property of the conference	Row.	2646930	1309696	217466	40	465300	19662 0				5 222			900
> % Trigger Functions			200		21	168	CHEMIN	MD2194				PARKET PRIORS CONT. ORCCOME.	V200720080	281471	75401	277008	70	N4231	19403 0			11	4 10		N 1850	000
y 🛅 Types			2470			500	110000146			: :		Equa opione po per ou acustos	Series	622289	000000	273622	- 60	24.5330	04602 0				5 1279			Out
) El Viene	All .		1070	- 1	- 21	974	110005147	100775530				State of the property of the party of the pa	GMICE	822769	800004	32966	- 4	263300	8000 0			14	5 1229			900
5 Subscriptions			T-40	-	-	100	4012225			2 10		Disease annual transport of the second river	Prince Perinded Bands	9997/7	960077	20007	10	192787	14000 0			14	4 1016		W later	rand
oxtgree	23		1076	- 1	21	554	62279546					Signer agreement dryne gydnau or oddolaen	LACY OF THE MOON	1327150	1305856	115304	72	000000	11202 0		0	14	1 113			7804
h/Croop Toles	20	25	1900	-	E1	996	14049374	11996591		2 4		STAGE OF STREET LINE CO.	Carrier	201799	80016	11909	36	900000	12000 0		0	11	5 45			901
;*(74		19675	- 1	-	100	145117920			2 4		Equalização establicas espapas	Service	922299	100001	1189122	61	ME E 200	9553 0			16	1 79			7921
	26		19001	4	21	006	114580006	05778977		2 4		Sayara area care capa nyon on other on	LACY OF THE MOON	1305154	1305854	115004	70	000000	117003 0		0	2	5 133			911
	97	10.	Here		10	100	SERVICE			7 19		Prisario recepio samonar	Litters	1001111	107950	1000	- 10	149.731	11/01 0			15	5 390			000
	29		1900	4	21	555	35327600	13062423		2 10		рубщия виновин бизи сокрайть стогое с дугиным дуквам	helel (B)	94177	10/177	125919	42	255300	190400 0		0	12	4 529		45 504	Gera
	79	26 1	HOUSE	4	21	556	25639079	26739852		2 4		Same process (syn price or other on	LACK DE THE WOOM	1805154	1885856	119864	- 66	600000	179000 0		0	16	5 838	154599 Bycanomi		(PP)
	22	25	1047	4	21	100	97775305	40077214		1 4		Simple residue residentes de compressor presentación del con e	NOVMETT	401722	940059	421001	100	4/1000	141503 0		0	17	3 267	110140 Nebari se	11 July	011
	31	20	19465	4	21	524	1010914	5011101		2 4		Secret dyna	A-6 Assessme Assess by Harris Aurigan	(710	16760	11121	41	265000	70123 0		0		5 1229			Ger
	87	83 6	1500	4	91	506	142723000	120701020		2 4		STRUTES ARE STRUCKED LONG OF	carrico	802299	800316	1198198	29	165500	10,001 0		0	15	5 29	menter makonifica	48 1000	9001
	33	52.	120-40	4	21	555	111029094	29761752		2 13		Рубедин женого комплекторного Ступе мексен фиделени	Sig Line	207153	217753	25203	79	565530	117603 0		0		5 200	114193 Ownerson	003 30 Talay	14.00
	As	33. 1	RUS	4	21	556	71763961	aumn		2 18		Рускию в свято инновательности местя	Parason Parasona res	466321	46971	315571	81	753190	107623 0		0	18	4 1010	ESTAN Mahasalan	60 files	0000
	39	34 4	LARGE	4	27	508	75525709	35451343		2 16		Potagous como escoprosopció terras	Edearance interfered	955277	999077	349923	17	02100	14,803 0		0	11	4 1010	155146 Nakorésa	60 late	20010
	35	25 1	17176	4	21	504	17109900	13003379		2 13	4 1	Pythana cuaporawa pjanimo centrala	ANNEN	134690	166690	114785	69	555000	170503 0		0	3	5 509	150140 Nebustra	20 5000	Gera
	87	86 1	17549	4	31	006	63934934	85759479		2 19	4 1	Pyrituria se cada degas la tratan especer agraga ta tera rapria	Katyino	107455	117466	400456	40	(61300)	414(0) 0		0		5 340		90 1966	6000
	33	37 4	17307	4	21	555	128547951	112007299		2 10	4 1	Рубево бурка соврейт велисок неродик с васлу сноги	THEIR	1258774	1260074	1055275	64	40,5000	145000 0		0	15	5 70	15650 Departure	023 21 laber	1921
	39	26	1964	4	21	526	1203-076	10045755		2 19	4 1	Руйкан миноски (или сверов) офисил в воличе приложник	Nicrond	31003540		45401	40	22490	85000 0		0	13	5 509	1000 Assessm	003 15 felies	(PER
	41		No. Cl		93	144	1000000	1005455				NAMES AND CORPORATION OF A STATE OF STA	Concessions	23491	10411	26%	101	£1111	294411 0			17	5 200	THANK MALE CON	101 10 1000	OCC 1

Рисунок 4 – Полученная база данных в СУБД PostgreSQL

Приложение 3. Программный код проведения экспериментов

Для реализации программного кода для проведения экспериментов был использован язык программирования Python. В программе использовалось 5 библиотек: datetime, psycopg2, sqlite3, math, time.

Функция arrayStrToFile записывает в текстовый файл log.txt строку или массив строк, которые подаются на вход функции.

```
def arrayStrToFile(array):
    with open('logs.txt', 'a', encoding='utf-8') as file:
        date = datetime.now().strftime('%Y-%m-%d_%H:%M:%S')
        if(isinstance(array, str)):
        file.write(f'{array}\n')
        else:
        for elem in array:
            file.write(f'{elem}\n')
```

Функция sred_sum_sv считает среднее значение и среднеквадратичное отклонение времени запроса и записывает их в файл.

```
def sred_sum_cv(time_array, name):
  n = len(time\_array)
  sred sum select = sum(time array)/len(time array)
  sum otckl = 0
  for i in range(n):
    sum_otckl = sum_otckl+(time_array[i]-sred_sum_select)**2
  sred cv otckl = math.sqrt(sum otckl/(n-1))
  print(f"\nСреднее значение времении запроса {name}\n{round(sred_sum_select,5)}")
  print(f"\nСреднеквадратическое отклонение времени запросы
{name}\n{round(sred_cv_otckl,5)}")
  arrayStrToFile(name)
  arrayStrToFile(time array)
  arrayStrToFile(f"Среднее значение времении запроса {name}")
  arrayStrToFile([sred_sum_select])
  arrayStrToFile(f"Среднеквадратическое отклонение времени запросы {name}")
  arrayStrToFile([sred cv otckl])
```

В начале основной функции программы выполняется подключение к СУБД PostgreSQL с помощью библиотеки psycopg2 или к СУБД SQLite с помощью библиотеки sqlite3 (рисунок 16).

```
def main():
    time_select = [] # Мссив времени выполнения запросов Select
    time_update = [] # Массив времени выполнения запросов Update
    time_delete = [] # Массив времени выполнения запросов Delete
```

После требуется п раз провести эксперимент для дальнейшего получения среднего значения и среднеквадратичного отклонения для СУБД PostgreSQL и SQLite.

```
for i in range(n):
     # Запрос в базу данных
     PostgreSQL_Select_1 = time.time()
     cursor.execute(postgreSQL_select_Query)
     PostgreSQL Select = time.time()-PostgreSQL Select 1
     selectData = cursor.fetchall()
     time select.append(PostgreSQL Select)
     print(PostgreSQL_Select, PostgreSQL_Select_1)
     print(len(selectData))
     # Обновление таблицы wb_products в PostgreSQL
     PostgreSQL_update_1 = time.time()
     cursor.execute(
       "UPDATE wb_products SET dist = 100 WHERE dist = 99")
     PostgreSQL_update = time.time()-PostgreSQL_update_1
     time_update.append(PostgreSQL_update)
     # Добавление selectData[0] в таблицу wb_products в PostgreSQL
     PostgreSQL Insert1 = time.time()
     cursor.execute("INSERT into wb_products values (
       # cursor.execute("'INSERT into wb_products values (
        ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", selectData[3])
     connection.commit()
     PostgreSQL_Insert = time.time()-PostgreSQL_Insert1
     time_insert.append(PostgreSQL_Insert)
```

В конце основной функции программы выполняется запись результатов эксперимента в файл log.txt и происходит закрытие соединения с подключенной СУБД.

[#] Записываем результаты в log

```
arrayStrToFile("")
  arrayStrToFile(f"======= База данных {len(selectData)} ========")
  arrayStrToFile("")
  sred_sum_cv(time_select, "Select")
  sred_sum_cv(time_update, "Update")
  sred_sum_cv(time_delete, "Delete")
  sred_sum_cv(time_insert, "Insert")
# Отслеживаем ошибку psycopg2. Operational Error
except psycopg2. Operational Error as error:
  print("Ошибка при работе с PostgreSQL", error)
# Независимо от результата закрываем cursor
finally:
  if connection:
    cursor.close()
    connection.close()
    print("\nСоединение с PostgreSQL закрыто")
```

Фрагмент текстового файла log.txt, в котором записаны все экспериментов для обеих СУБД, продемонстрирован на Рисунке 19.

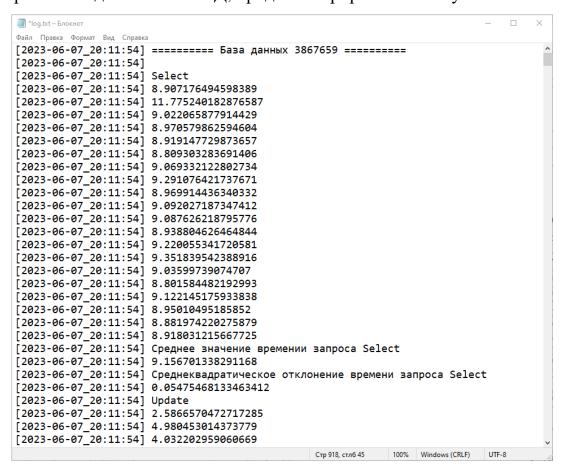


Рисунок 5 – Эксперименты; фрагмент текстового файла log.txt