

⚡ TPEK PostgreSQL
👥 КОМАНДА **T**he_**L**ast_**S**iberia



Алёна Куликова

КАПИТАН



ALENA KULIKOVA

ПОДДЕРЖКА
КАПИТАНА



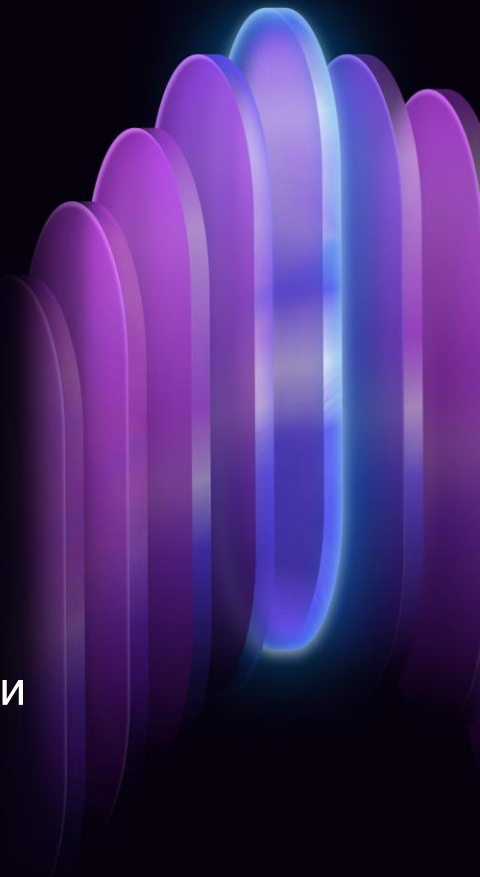
Задача

Разработать решение для мониторинга и анализа SQL-запросов к PostgreSQL, которое:

- оценивает «стоимость» запроса до его выполнения;
- дает рекомендации по оптимизации как самого запроса, так и базы данных;
- помогает избежать проблем с производительностью.

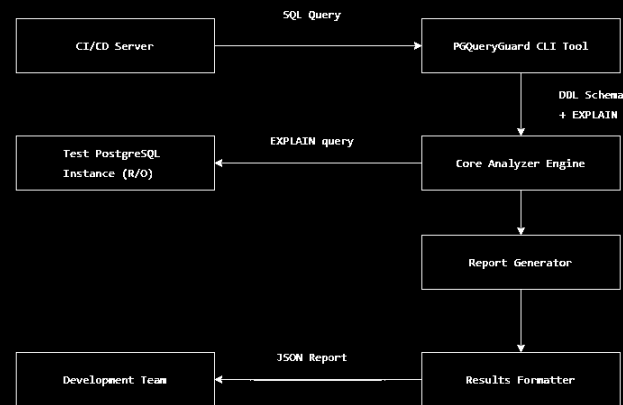
Цель

Умный инструмент для аналитиков, DBA и разработчиков, который предупреждает о рисках и сокращает время ответа БД



Архитектура решения

- **CI/CD Server** - Точка входа в систему
 - Иницирует процесс анализа при изменениях в SQL-файлах
 - Предоставляет тестовое окружение с PostgreSQL
 - Запускает PGQueryGuard CLI Tool
- **PGQueryGuard CLI Tool** - Основной интерфейс взаимодействия с системой
 - Парсинг входных параметров
 - Координация работы между компонентами
 - Форматирование вывода для разных форматов
- **Core Analyzer Engine** - Интеллектуальное ядро системы
 - Генерация и выполнение EXPLAIN запросов
 - Парсинг JSON-ответов от PostgreSQL
 - Классификация рекомендаций по приоритетам
- **Test PostgreSQL Instance** - Контролируемая тестовая среда для анализа
 - Хранение актуальной схемы БД
 - Выполнение EXPLAIN запросов
 - Предоставление статистики и метрик
- **Report Generator** - Преобразование сырых данных в структурированные отчеты
 - Расчет общего score качества запроса
 - Формирование структурированного JSON-отчета
- **Results Formatter** - Адаптация результатов для конечных потребителей
- **Development Team** - Конечные потребители системы



Технологический стек

- Язык
Python 3.10+
- База данных
PostgreSQL 15+
- Основные зависимости
 1. psycopg2-binary
 2. pydantic
 3. rich
 4. typer
 5. pyyaml

Технические трудности

- К сожалению присоединилась буквально за неделю до окончания
- К сожалению, доступ к Т1 Облако, получила за сутки, чтобы что-то настроить и реализовать адекватное

Реализация - Поднятие всех ресурсов

+| TI Облако

База данных

- Managed Service for PostgreSQL
- Managed Service for ClickHouse
- Managed Service for DocumentDB
- Managed Service for Redis
- Managed Service for MySQL
- Managed Service for OpenSearch

The Last Siberia (819,04 Р/сут.) 1 000 092 509 963,22 Р

Managed Service for PostgreSQL > The Last Siberia

819,04 Р/сут. Настроить Подключить Действия

Информация

Имя кластера	The_Last_Siberia
Описание	—
Версия PostgreSQL	15.4
Версия PGBackuper	1.20.0
Количество нод	7
Размер диска на ноду (GB)	20
Количество vCPU на ноду	4
Количество RAM на ноду	8 GB
Тип диска	Average cluster IOPS Read: 10000 IOPS Write: 3000
Совместно / платформа	general purpose / Intel Cascade Lake 2.2 Gt/s
Датум центр	ru-central1-a
Сеть/Подсеть	default/default-ru-central1
Публичный IP-адрес	193.246.150.20
Виртуальный IP-адрес	10.128.0.10
Ограничение скорости, Мбит/сек	100
Порт для подключения PostgreSQL	5000
Время резервного копирования UTC	—
Максимальное кол-во резервных копий	—
Группа безопасности	default
Дни обслуживания	Пн
Время обслуживания	00:00 - 01:00
Тип подключения	TLS и Non-TLS
Статус	Выполнен

+| TI Облако

- Ресурсы
- ИМ и Управление
- Биллинг
- Инструменты
- Центр поддержки
- Cloud Engine
- Наш
- TI Onk
- Объектное хранилище S3
- Ограничения
- Cloud CDN
- Cloud Director
- Контейнеры
- Load Balancer
- Базы данных
- Certificate & Secret Manager
- Managed Service for GitLab

Marketplace

Свернуть меню

The Last Siberia 1 000 092 527 736,22 Р

Текущий баланс счета
1 000 092 527 736,22 Р

Текущий расход

Версия портала
0.116.0

Последние статьи

- Документация по API (Документ)
- Документация
- Клиент с Redis Redis
- Тестирование провайдеров

Cloud Engine

Cloud Compute

- ВМ 0
- Диски 0
- Сетевые 0
- Образы 0
- Резервные копии 0
- Интерфейсы 0
- SSH-ключи 0

Cloud VPC

- Сети 1
- Подсети 1
- IP-адреса 0

Лист событий

События

- В Cloud Engine диск типа High theory, доступный в зоне ru-central1-a и ru-central1-a
- Установка релиза 0.116.0
- Установка релиза 0.115.0
- Установка релиза 0.114.0
- Установка релиза 0.113.0

[Показать еще](#)

Реализация - Поднятие всех ресурсов

TI Облако

The_Last_Siberia (819,04 P/сут.) 1000 092 473 784,22 P

Managed Service for PostgreSQL > The_Last_Siberia

The_Last_Siberia 819,04 P/сут. Настройки Подключиться Действия

Базы данных

Managed Service for PostgreSQL

Managed Service for ClickHouse

Managed Service for DocumentDB

Managed Service for InMemoryDB

Managed Service for MySQL

Managed Service for OpenSearch

Информация Пользователи Базы данных Ноды Резервные копии Настройки СУБД История действий

Базы данных

Название	Владелец	LC collate	LC type
The_Last_Siberia	kagamine01	en_US.UTF-8	UTF8

Подключиться + Добавить

Записей на странице: 10 1-1 записей из 1 1 из 1 страницы

Подключиться

Пример строки подключения:

```
psql "host=193.246.150.20\  
port=5000 \  
sslmode=require \  
dbname={db_name} \  
user={user_name} \  
target_session_attrs=read-write"
```

Реализация – Запуск демо (генерация отчета)

```
(.venv) alyona@alyona-virtualbox:~/PycharmProjects/PythonProjecttha  
caton$ python -m pgqueryguard.main  
PDF-отчет успешно сохранен: output/demo_report_20250909_144930.pdf
```

demo_report_20250909_144930.pdf — PGQueryGuard Report

Файл Изменить Вид Переход Закладки Справка

← Предыдущая 1 (1 из 2) Следующая →

PGQueryGuard — Анализ SQL-запроса

Дата анализа: 2025-09-09 14:49:30
Окружение T1 Cloud: demo

Информация о запросе

Тип запроса:	SELECT
Затронутые таблицы:	customers
Используемые индексы:	customers_idx

Метрики производительности

Общая стоимость:	4693.99
Оценочное время выполнения:	46.94 ms
Оценочное количество строк:	1,665,079
Блоков с диска:	968
Параллельные воркеры:	1
Типы узлов плана:	Merge Join, Index Scan, Hash Join, Sort

Рекомендации по оптимизации

Рекомендация #1: Отсутствует статистика для таблицы customers

Тип: missing_statistics
Приоритет: medium
Влияние: 6/10

Реализация – Описание главного кода

Устанавливает соединение с PostgreSQL и сохраняет информацию о сервере

- Подключается к БД с таймаутом 10 сек.
- Устанавливает `application_name` для мониторинга на стороне сервера.
- Автокоммит включён — так как мы только читаем планы (`EXPLAIN`).
- Выполняет запрос `SELECT version(), current_database()` — чтобы сохранить метаданные для отчёта.

```
33 def connect(self): 1 usage
34     try:
35         self.connection = psycopg2.connect(
36             self.dsn,
37             connect_timeout=10,
38             application_name="pgqueryguard-t1-cloud"
39         )
40         self.connection.autocommit = True
41
42     with self.connection.cursor() as cur:
43         cur.execute("SELECT version(), current_database()")
44         version_info, db_name = cur.fetchone()
45         self.server_version = version_info
46         self.db_name = db_name
47
48     except psycopg2.Error as e:
49         raise
```

Реализация – Описание главного кода

Получает план выполнения запроса в формате JSON

- Сначала пытается выполнить EXPLAIN (FORMAT JSON, VERBOSE, SETTINGS, BUFFERS) — максимально детальный план.
- Если возникает ошибка (например, нет прав на BUFFERS) — fallback на EXPLAIN (FORMAT JSON).
- Парсит JSON и возвращает словарь.

```
51 def get_explain_plan(self, query: str) -> Dict[str, Any]: 1 usage
52     with self.connection.cursor() as cur:
53         try:
54             explain_query = f"EXPLAIN (FORMAT JSON, VERBOSE, SETTINGS, BUFFERS) {query}"
55             cur.execute(explain_query)
56             result = cur.fetchone()
57
58             if result and result[0]:
59                 plan_data = json.loads(result[0])
60                 return plan_data
61             else:
62                 raise Exception("Пустой результат EXPLAIN")
63
64     except psycopg2.Error:
65         try:
66             explain_query = f"EXPLAIN (FORMAT JSON) {query}"
67             cur.execute(explain_query)
68             result = cur.fetchone()
69             if result and result[0]:
70                 plan_data = json.loads(result[0])
71                 return plan_data
72         except psycopg2.Error as e:
73             raise Exception(f"Не удалось получить план выполнения: {e}")
```

Реализация – Описание главного кода

Извлекает числовые метрики производительности из плана выполнения и возвращает объект QueryMetric

total_cost — общая стоимость запроса по оценке планировщика.

planning_time — время планирования (если есть).

max_execution_time — оценка времени выполнения

shared_hit_blocks — блоки, прочитанные из кэша.

shared_read_blocks — блоки, прочитанные с диска.

plan_width — средняя ширина строки результата

total_rows — оценочное количество строк.

node_types — список уникальных типов узлов плана

startup_cost — стоимость до начала возврата первой строки.

total_workers — общее количество воркеров

parallel_workers — количество запущенных параллельных воркеров.

```
75     def extract_metrics(self, plan: Dict[str, Any]) -> QueryMetric: 1usage
76         total_plan = plan[0]['Plan']
77
78         metrics = QueryMetric(
79             total_cost=total_plan['Total Cost'],
80             planning_time=total_plan.get('Planning Time'),
81             max_execution_time=self.estimate_execution_time(total_plan['Total Cost']),
82             shared_hit_blocks=total_plan.get('Shared Hit Blocks', 0),
83             shared_read_blocks=total_plan.get('Shared Read Blocks', 0),
84             plan_width=total_plan['Plan Width'],
85             total_rows=total_plan['Plan Rows'],
86             node_types=self.extract_node_types(total_plan),
87             startup_cost=total_plan.get('Startup Cost', 0),
88             total_workers=total_plan.get('Workers', 0),
89             parallel_workers=total_plan.get('Workers Launched', 0)
90         )
91
92     return metrics
```

Реализация – Описание главного кода

Рекурсивно обходит дерево плана выполнения и собирает уникальные типы узлов

```
94 def extract_node_types(self, plan_node: Dict[str, Any]) -> List[str]: 1 usage
95     node_types = []
96
97     def _extract_nodes(node):
98         if 'Node Type' in node:
99             node_types.append(node['Node Type'])
100         if 'Plans' in node:
101             for child in node['Plans']:
102                 _extract_nodes(child)
103
104     _extract_nodes(plan_node)
105     return list(set(node_types))
```

Заглушка для анализа структуры плана

```
107 def analyze_plan_structure(self, plan: Dict[str, Any]) -> None: 1 usage
108     total_plan = plan[0]['Plan']
109
110     def build_tree(node, parent_tree=None):
111         if 'Plans' in node:
112             for child in node['Plans']:
113                 build_tree(child, parent_tree)
114
115     build_tree(total_plan)
```

Реализация – Описание главного кода

Преобразует "стоимость" запроса в
оценочное время выполнения

```
117 def estimate_execution_time(self, total_cost: float) -> float: 1usage
118     return total_cost * 0.01
119
```

Генерирует список предупреждений на
основе анализа плана и текста запроса

```
120 def extract_indexes_used(self, plan: Dict[str, Any]) -> List[str]: 1usage
121     indexes = []
122
123     def _find_indexes(node):
124         if node.get('Node Type') == 'Index Scan':
125             index_name = node.get('Index Name', 'unknown')
126             relation_name = node.get('Relation Name', 'unknown')
127             indexes.append(f"{relation_name}({index_name})")
128
129         if 'Plans' in node:
130             for child in node['Plans']:
131                 _find_indexes(child)
132
133     _find_indexes(plan[0]['Plan'])
134     return indexes
```

Реализация – Описание главного кода

Генерирует список рекомендаций по оптимизации запроса с учетом особенностей T1 Cloud

Каждая рекомендация — объект Recommendation с описанием, приоритетом, оценкой улучшения, действием и сервисом T1 Cloud

Таблица типов рекомендации

	Standard	Standard	Standard	Standard
1	Тип	Условие	Приоритет	Описание
2	missing_index	Seq Scan + Filter + >10K строк	HIGH	Создать индекс на полях фильтра
3	disk_sort	Sort Method = external	MEDIUM	Увеличить work_mem или оптимизировать ORDER BY
4	inefficient_join	Nested Loop + >1K строк	MEDIUM	Рассмотреть Hash Join"

```
150 def generate_t1_recommendations(self, plan: Dict[str, Any], query: str) -> List[Recommendation]:
151     recommendations = []
152     total_plan = plan[g]['Plan']
153
154     seq_scans = find_plan_nodes(total_plan, 'node_type: Seq Scan')
155     for scan in seq_scans:
156         if scan['Plan Rows'] > 10000 and 'Filter' in scan:
157             table_name = scan.get('Relation Name', 'unknown')
158             rec = Recommendation(
159                 type="missing_index",
160                 description=f"Полное сканирование большой таблицы {table_name} ({scan['Plan Rows']:,} строк)",
161                 priority=Priority.HIGH,
162                 estimated_improvement="Ускорение на 80-95%",
163                 suggested_action=f"Создать индекс на {table_name}({extract_filter_columns(scan)}).",
164                 affected_components=[table_name],
165                 t1_service=T1CloudService.POSTGRESQL,
166                 impact_score=9
167             )
168             recommendations.append(rec)
```

Реализация – Описание главного кода

Рассчитывает общий score запроса на основе метрик и рекомендаций

Стоимость запроса

10000 → -30 5000 → -20 1000 → -10

Дисковые чтения

1000 блоков → -25 500 блоков → -15

Рекомендации

HIGH → -30 MEDIUM → -15

```
202 def calculate_score(self, metrics: QueryMetric, recommendations: List[Recommendation]) -> int:
203     base_score = 100
204
205     if metrics.total_cost > 10000:
206         base_score -= 30
207     elif metrics.total_cost > 5000:
208         base_score -= 20
209     elif metrics.total_cost > 1000:
210         base_score -= 10
211
212     if metrics.shared_read_blocks > 1000:
213         base_score -= 25
214     elif metrics.shared_read_blocks > 500:
215         base_score -= 15
216
217     for rec in recommendations:
218         if rec.priority == Priority.HIGH:
219             base_score -= 30
220         elif rec.priority == Priority.MEDIUM:
221             base_score -= 15
222
223     return max(0, min(100, base_score))
```

Реализация – Описание главного кода

выполняет полный анализ SQL-запроса
и возвращает структурированный
отчёт

```
225     def analyze_query(self, query: str) -> AnalysisReport: 1 usage
226         query_info = extract_query_info(query)
227         plan = self.get_explain_plan(query)
228         if self.verbose:
229             self.analyze_plan_structure(plan)
230         metrics = self.extract_metrics(plan)
231         recommendations = self.generate_t1_recommendations(plan, query)
232         report = AnalysisReport(
233             query=query,
234             metrics=metrics,
235             recommendations=recommendations,
236             is_critical=any(r.priority == Priority.HIGH for r in recommendations),
237             score=self.calculate_score(metrics, recommendations),
238             ti_environment=self.ti_environment,
239             query_type=query_info['type'],
240             tables_affected=query_info['tables'],
241             indexes_used=self.extract_indexes_used(plan),
242             warnings=self.generate_warnings(plan, query)
243         )
244         return report
```


Реализация – Оценка отчета

Как можно заметить,
отчет составляется
информативно и
предоставляет
рекомендацию

PGQueryGuard — Анализ SQL-запроса

Дата анализа: 2025-09-09 15:01:36
Окружение: T1 Cloud: demo

Информация о запросе

Тип запроса:	UPDATE
Загружаемая таблица:	customers, test
Используемые индексы:	test

Метрики производительности

Общая стоимость:	34196.43
Оценочное время выполнения:	341.96 мс
Оценочное количество строк:	310.036
Блоков с диска:	1502
Параллельные воркеры:	3
Типы узлов плана:	Sort, Hash Join, Aggregate

Рекомендации по оптимизации

Рекомендация #1: Полное сканирование большой таблицы customers (1757103 строк)

Тип: missing_index
Приоритет: medium
Влияние: 710

Ожидаемое улучшение: Ускорение на 50%

Рекомендуемое действие: Создать индекс на customers(quantity)

Сервис: T1 Cloud: Managed Service for PostgreSQL

Рекомендация #2: Сортировка выполняется на диске (медианно)

Тип: disk_sort
Приоритет: low
Влияние: 410

Ожидаемое улучшение: Ускорение на 89%

Итоговая оценка

Оценка качества: 62/100

Критичность: Да

Количество рекомендаций: 4

Количество предупреждений: 0

Сгенерировано с помощью PGQueryGuard for T1 Cloud — 2025-09-09



Результаты

- **Разработано** решение для мониторинга и анализа SQL-запросов к PostgreSQL
- **Не смотря на тех. трудности** и отсутствие времени, имеется прототип проекта, который можно развить до автоматизации, что на данный момент является крайне важным:
 - Команды разработки получают проактивные рекомендации до попадания запроса в прод
 - Администраторы БД могут предотвращать инциденты производительности
 - Снижается нагрузка на кластеры PostgreSQL в T1 Cloud

