УДК 004.652.4:004.65

Особенности применения партиционирования по дискретным значениям в СУБД PostgreSQL

Забродин Андрей Владимирович — канд. ист. наук, доцент кафедры «Информационные и вычислительные системы». Научные интересы: системы управления базами данных, технологии хранения и обработки больших данных. E-mail: zabrodin@pgups.ru

Сабиров Никита — студент 3-го курса направления 09.03.01 «Информатика и вычислительная техника». Научные интересы: системы управления базами данных, технологии хранения и обработки больших данных, производительность СУБД. E-mail: nikita.sabirov.30@gmail.com

Петербургский государственный университет путей сообщения Императора Александра I, Россия, 190031, Санкт-Петербург, Московский пр., 9

Для цитирования: Забродин А. В., Сабиров Н. Особенности применения партиционирования по дискретным значениям в СУБД PostgreSQL // Интеллектуальные технологии на транспорте. 2025. № 3 (43). С. 49–57. DOI: 10.20295/2413-2527-2025-343-49-57

Аннотация. *С ростом объемов обрабатываемых данных в информационных системах возрастает потребность в использовании методов, обеспечивающих масштабируемость, высокую производительность и эффективное администрирование. Одним из таких методов является партиционирование таблиц — логическое разделение данных на независимые сегменты, способствующее оптимизации выполнения запросов и упрощению управления большими таблицами.* ***Цель:*** *исследование особенностей применения партиционирования по дискретным значениям (list partitioning) в СУБД PostgreSQL и оценка его влияния на производительность SQL-запросов.* ***Результаты:*** *представлены теоретические основы list partitioning, его синтаксическая реализация и ограничения. Проведен сравнительный эксперимент, показавший прирост производительности при использовании партиционирования по сравнению с монолитной таблицей. Описаны типовые сценарии применения метода при работе с категоризованными данными.* ***Практическая значимость:*** *состоит в возможности повышения эффективности обработки запросов и администрирования крупных таблиц в PostgreSQL путем рационального проектирования структуры хранения данных с использованием list partitioning.*

Ключевые слова: *партиционирование данных, PostgreSQL, list partitioning, производительность, оптимизация запросов, индексация, распределение данных, горизонтальное фрагментирование, архитектура баз данных, большие данные*

2.3.5 *— математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей (технические науки)*

Введение

Предметом настоящего исследования является механизм партиционирования данных по дискретным значениям (list partitioning) в системе управления базами данных PostgreSQL. Современные информационные системы оперируют постоянно растущими объемами данных, что требует не только надежного хранения, но и высокой скорости доступа, масштабируемости и эффективности выполнения запросов.

Одним из средств достижения этих целей является партиционирование — логическое разбиение таблиц на части, каждая из которых хранится

и обрабатывается независимо. Среди различных стратегий партиционирования, реализованных в PostgreSQL, особое место занимает партиционирование по списку, позволяющее распределять строки таблицы между партициями на основании точных значений одного или нескольких атрибутов.

Актуальность исследования обусловлена необходимостью повышения производительности операций выборки, обновления и удаления данных в больших таблицах с выраженной категоризованной структурой, что характерно для таких прикладных областей, как системы управления взаимоотношениями с клиентами (Customer Relationship Management, CRM), электронная коммерция и логистика. В статье рассматриваются принципы реализации list partitioning, включая синтаксис, типовые схемы проектирования, преимущества и ограничения технологии.

Теоретическая значимость работы заключается в систематизации представлений знаний о внутреннем устройстве и особенностях функционирования механизма поведения list partitioning в PostgreSQL, а также в формализации критериев его применения. Практическая значимость выражается в проведении экспериментов на выборке данных, демонстрирующих влияние партиционирования на время выполнения запросов. Результаты исследования могут быть применены при проектировании и оптимизации баз данных в системах, работающих с большими объемами категориальных данных, таких как CRM-системы, онлайн-ретейл и логистические платформы.

Теоретические основы партиционирования в PostgreSQL

Партиционирование таблиц — это механизм логического разбиения данных на несколько физических подтаблиц (партиций), управляемых через одну родительскую таблицу [1]. Вариант листового партиционирования (`list partitioning`) позволяет группировать записи по дискретным значениям указанного столбца. Такая схема особенно эффективна при наличии ограниченного множества возможных значений, например типов поездов, категорий товаров, регионов и т. д. [2].

Каждая партиция в данном случае отвечает за один или несколько фиксированных значений поля, указанного в `PARTITION BY LIST()` при создании таблицы [3].

Одним из преимуществ использования list partitioning является возможность исключения неактуальных партиций на этапе планирования выполнения запроса (partition pruning), что позволяет существенно снизить объем обрабатываемых данных и, как следствие, повысить общую производительность системы.

Для демонстрации эффективности листового партиционирования были созданы две идентичные по структуре таблицы (рис. 1).

В таблице `trains_with_partition` реализовано листовое партиционирование по типу поезда (`train_type`). Всего создано 4 партиции (рис. 2).

Каждая партиция отвечает за фиксированный набор значений поля train_type. Такая схема позволяет PostgreSQL эффективно исключать ненужные

```
CREATE TABLE trains_no_partition (
    train_id SERIAL,
    train_number VARCHAR(10) NOT NULL,
    departure_station VARCHAR(50) NOT NULL,
    arrival_station VARCHAR(50) NOT NULL,
    departure_time TIMESTAMP NOT NULL,
    train_type VARCHAR(20) NOT NULL,
    distance_km INT NOT NULL,
    ticket_price DECIMAL(10,2) NOT NULL
);
```

а

```
CREATE TABLE trains_with_partition (
    train_id SERIAL,
    train_number VARCHAR(10) NOT NULL,
    departure_station VARCHAR(50) NOT NULL,
    arrival_station VARCHAR(50) NOT NULL,
    departure_time TIMESTAMP NOT NULL,
    train_type VARCHAR(20) NOT NULL,
    distance_km INT NOT NULL,
    ticket_price DECIMAL(10,2) NOT NULL
) PARTITION BY LIST (train_type);
```

б

Рис. 1. Создание таблиц:
а — без партиционирования; *б* — с партиционированием

```
CREATE TABLE trains_highspeed PARTITION OF trains_with_partition
    FOR VALUES IN ('Сапсан', 'Ласточка', 'Аллегро');

CREATE TABLE trains_long_distance PARTITION OF trains_with_partition
    FOR VALUES IN ('Фирменный', 'Скорый', 'Пассажирский');

CREATE TABLE trains_suburban PARTITION OF trains_with_partition
    FOR VALUES IN ('Электричка', 'Экспресс');

CREATE TABLE trains_other PARTITION OF trains_with_partition
    FOR VALUES IN ('Почтовый', 'Грузопассажирский', 'Ретро');
```

Рис. 2. Создание партиций для разных типов /поездов

партиции при запросах, содержащих фильтрацию по нужному полю. Например, при выполнении запроса `WHERE train_type = 'Сапсан'` будет анализироваться только партиция `trains_highspeed`.

Заполнение таблиц производилось с помощью SQL-скрипта на языке PL/pgSQL, генерирующего 1 000 000 записей в каждую таблицу, равномерно распределенных по станциям и типам поездов (рис. 3). Для повышения производительности поиска были созданы индексы на поля `train_type` и `departure_time` в обеих таблицах (рис. 4).

Индексы в PostgreSQL, как и в большинстве реляционных СУБД, по умолчанию создаются с использованием структуры **B-дерева (B-tree)**, которая обеспечивает логарифмическое время поиска, вставки и удаления. Это делает такие индексы особенно эффективными при фильтрации, сортировке и использовании операторов сравнения (`=`, `<`, `>`, `BETWEEN`, `IN` и др.).

В случае партиционированных таблиц индексы необходимо создавать отдельно для каждой партиции, поскольку каждая партиция — это физически самостоятельная таблица. Если индекс создается на родительской таблице, PostgreSQL автоматически транслирует его создание на все существующие партиции [4]. Однако при добавлении новых партиций индексы должны быть либо созданы вручную, либо настроены с использованием декларативных механизмов, обеспечивающих автоматическое наследование индексной структуры.

Преимущество индексов в контексте партиционирования заключается в том, что они работают в сочетании с механизмом **partition pruning** — когда PostgreSQL при выполнении запроса учитывает ключ партиционирования и обращается только к тем партициям, которые могут содержать подходящие данные. Это позволяет значительно сократить объем данных, подлежащих анализу, и уменьшить общее время выполнения запроса [5].

Например, при выполнении запроса вида `WHERE train_type = 'Сапсан'` СУБД будет использовать только партицию `trains_highspeed` и соответствующий ей локальный индекс, игнорируя остальные. Таким образом, использование индексов на партициях повышает селективность, снижает время выполнения запроса, нагрузку на диск и улучшает общую масштабируемость системы при работе с миллионами записей.

Математическая модель оценки эффективности

Оценка эффективности выполнения запросов проводится на основе времени выполнения (`Execution Time`) и стоимости запроса (`Total Cost`), отображаемой оператором `EXPLAIN ANALYZE`.

Пусть:

- T_0 — время выполнения запроса на непартиционированной таблице;
- T_p — время выполнения запроса на партиционированной таблице;
- $\Delta T = T_0 - T_p$ — выигрыш во времени;
- C_0, C_p — стоимости планов выполнения соответствующих запросов.

```
DO $$
DECLARE
    station_list VARCHAR[] := ARRAY[
        'Москва', 'Санкт-Петербург', 'Новосибирск', 'Екатеринбург', 'Казань',
        'Нижний Новгород', 'Челябинск', 'Самара', 'Омск', 'Ростов-на-Дону'
    ];
    train_types VARCHAR[] := ARRAY[
        'Сапсан', 'Ласточка', 'Аллегро', 'Фирменный', 'Скорый',
        'Пассажирский', 'Электричка', 'Экспресс', 'Почтовый', 'Грузопассажирский', 'Ретро'
    ];
BEGIN
    FOR i IN 1..1000000 LOOP
        INSERT INTO trains_no_partition (
            train_number,
            departure_station,
            arrival_station,
            departure_time,
            train_type,
            distance_km,
            ticket_price
        ) VALUES (
            'T' || (1000 + (random() * 8999))::INT,
            station_list[1 + (random() * (array_length(station_list, 1) - 1))::INT],
            station_list[1 + (random() * (array_length(station_list, 1) - 1))::INT],
            CURRENT_TIMESTAMP - (random() * 365)::INT * INTERVAL '1 day',
            train_types[1 + (random() * (array_length(train_types, 1) - 1))::INT],
            (50 + (random() * 3000))::INT,
            (100 + (random() * 15000))::NUMERIC(10,2)
        );

        INSERT INTO trains_with_partition (
            train_number,
            departure_station,
            arrival_station,
            departure_time,
            train_type,
            distance_km,
            ticket_price
        ) VALUES (
            'T' || (1000 + (random() * 8999))::INT,
            station_list[1 + (random() * (array_length(station_list, 1) - 1))::INT],
            station_list[1 + (random() * (array_length(station_list, 1) - 1))::INT],
            CURRENT_TIMESTAMP - (random() * 365)::INT * INTERVAL '1 day',
            train_types[1 + (random() * (array_length(train_types, 1) - 1))::INT],
            (50 + (random() * 3000))::INT,
            (100 + (random() * 15000))::NUMERIC(10,2)
        );
    END LOOP;
END $$;
```

Рис. 3. Заполнение таблиц данными

```
-- Для обычной таблицы
CREATE INDEX idx_no_partition_train_type ON trains_no_partition(train_type);
CREATE INDEX idx_no_partition_departure_time ON trains_no_partition(departure_time);

-- Для партиционированной таблицы (автоматически создастся на всех партициях)
CREATE INDEX idx_partition_train_type ON trains_with_partition(train_type);
CREATE INDEX idx_partition_departure_time ON trains_with_partition(departure_time);
```

Рис. 4. Создание индексов

Ожидается, что при фильтрации по `train_type`, соответствующей ключу партиционирования, будет наблюдаться значительное снижение как T_p, так и C_p за счет исключения лишних партиций.

Для количественной оценки эффекта от применения партиционирования можно ввести метрику относительного ускорения:

$$\eta = \frac{T_0 - T_p}{T_0} 100, \quad (1)$$

где η — процентное улучшение производительности.

Сравнительный анализ запросов

Для оценки влияния партиционирования на производительность были выполнены три вида SQL-запросов: фильтрация по ключу партиционирования, фильтрация по полю, не участвующему в партиционировании, и агрегатные операции.

Запрос 1. Фильтрация по ключу партиционирования

На рис. 5 представлен пример выполнения запроса с фильтрацией по полю `train_type`, которое является ключом партиционирования. Благодаря механизму отсечения (pruning) PostgreSQL обращается только к релевантной партиции (в данном случае — `trains_highspeed`).

Запрос 2. Фильтрация по дате (не ключ партиционирования)

На рис. 6 представлен план выполнения запроса с фильтрацией по непартиционируемому полю departure_time.

Запрос 3. Агрегация по типу поезда

Рис. 7 демонстрирует план агрегатного запроса с группировкой по `train_type`. Благодаря партиционированию [6] PostgreSQL может обрабатывать каждую партицию независимо, что особенно эффективно при операциях **GROUP BY** по ключу партиционирования. Это позволяет избежать полного сканирования таблицы и сократить время выполнения за счет распределенной обработки данных, что существенно сокращает общее время выполнения запроса.

```
-- Обычная таблица
EXPLAIN ANALYZE
SELECT * FROM trains_no_partition
WHERE train_type = 'Сапсан';

-- Партиционированная таблица
EXPLAIN ANALYZE
SELECT * FROM trains_with_partition
WHERE train_type = 'Сапсан';
```

Рис. 5. План выполнения запроса с фильтрацией по ключу партиционирования (`train_type`)

Результаты выполнения запросов представлены на рис. 8. Исходя из полученных результатов, можно увидеть, что таблица, имеющая партиции, завершила работу гораздо быстрее:

- без партиционирования: `Execution Time: 184,489 ms`;
- с партиционированием: `Execution Time: 36,639 ms`.

Относительное ускорение (1) равно:

$$\eta = \frac{184,489 - 36,639}{184,489} 100 \approx 80,1\ \%.$$

Также можно проверить размеры данных партиций. На рис. 9 представлен запрос, определяющий размер каждой партиции и примерное количество строк.

```
-- Обычная таблица
EXPLAIN ANALYZE
SELECT * FROM trains_no_partition
WHERE departure_time BETWEEN '2023-01-01' AND '2023-01-31';

-- Партиционированная таблица
EXPLAIN ANALYZE
SELECT * FROM trains_with_partition
WHERE departure_time BETWEEN '2023-01-01' AND '2023-01-31';
```

Рис. 6. План выполнения запроса с фильтрацией по непартиционируемому полю (`departure_time`)

```
-- Обычная таблица
EXPLAIN ANALYZE
SELECT train_type, COUNT(*), AVG(ticket_price)
FROM trains_no_partition
GROUP BY train_type;

-- Партиционированная таблица
EXPLAIN ANALYZE
SELECT train_type, COUNT(*), AVG(ticket_price)
FROM trains_with_partition
GROUP BY train_type;
```

Рис. 7. План выполнения агрегатного запроса с группировкой по типу поезда

Результаты, отраженные на рис. 10, указывают на различия в объемах данных по партициям, что объем данных распределен между партициями неравномерно: наиболее загруженной оказалась `trains_long_distance`, а наименьшей — `trains_suburban`. Это свидетельствует о логичности структуры партиционирования [7].

Анализ полученных данных показывает, что каждая из партиций содержит различный объем информации как по размеру данных, так и по оценочному количеству строк. Наибольшая нагрузка приходится на партицию `trains_long_distance`, что,

	QUERY PLAN text
1	Bitmap Heap Scan on trains_no_partition (cost=561.08..16920.49 rows=49633 width=89) (actual time=7.517..182.768 rows=49774 loops=1)
2	Recheck Cond: ((train_type)::text = 'Сапсан'::text)
3	Heap Blocks: exact=15132
4	-> Bitmap Index Scan on idx_no_partition_train_type (cost=0.00..548.67 rows=49633 width=0) (actual time=5.477..5.478 rows=49774 loop...
5	Index Cond: ((train_type)::text = 'Сапсан'::text)
6	Planning Time: 2.994 ms
7	Execution Time: 184.489 ms

	QUERY PLAN text
1	Bitmap Heap Scan on trains_highspeed trains_with_partition (cost=575.91..5038.80 rows=51031 width=85) (actual time=3.549..35.451 rows=50278 loop...
2	Recheck Cond: ((train_type)::text = 'Сапсан'::text)
3	Heap Blocks: exact=3825
4	-> Bitmap Index Scan on trains_highspeed_train_type_idx (cost=0.00..563.15 rows=51031 width=0) (actual time=2.901..2.901 rows=50278 loops=1)
5	Index Cond: ((train_type)::text = 'Сапсан'::text)
6	Planning Time: 5.477 ms
7	Execution Time: 36.639 ms

Рис. 8. Результаты выполнения запросов

```
SELECT
    child.relname AS partition_name,
    pg_get_expr(child.relpartbound, child.oid) AS partition_condition,
    pg_size_pretty(pg_total_relation_size(child.oid)) AS total_size,
    pg_size_pretty(pg_relation_size(child.oid)) AS data_size,
    pg_size_pretty(pg_indexes_size(child.oid)) AS indexes_size,
    pg_class.reltuples::bigint AS estimated_rows
FROM pg_inherits
JOIN pg_class parent ON pg_inherits.inhparent = parent.oid
JOIN pg_class child ON pg_inherits.inhrelid = child.oid
JOIN pg_class ON (pg_class.oid = child.oid)
WHERE parent.relname = 'trains_with_partition'
ORDER BY partition_name;
```

Рис. 9. Запрос просмотра объема данных

	partition_name name	partition_condition text	total_size text	data_size text	indexes_size text	estimated_rows bigint
1	trains_highspeed	FOR VALUES IN ('Сапсан', 'Ласточка', 'Аллегро')	33 MB	30 MB	3544 kB	249742
2	trains_long_distance	FOR VALUES IN ('Фирменный', 'Скорый', 'Пассажирский')	41 MB	37 MB	4264 kB	299178
3	trains_other	FOR VALUES IN ('Почтовый', 'Грузопассажирский', 'Ретр...	35 MB	32 MB	3584 kB	250861
4	trains_suburban	FOR VALUES IN ('Электричка', 'Экспресс')	28 MB	25 MB	2872 kB	200219

Рис. 10. Вывод результата запроса объема данных

вероятно, обусловлено высокой востребованностью дальнемагистральных рейсов, значительным объемом сопутствующей информации, фиксируемой в базе данных, а также частотой их использования.

В то же время самая компактная по объему и количеству строк партиция — `trains_suburban`, что также логично, учитывая более короткие маршруты и, как правило, меньший объем информации по каждому рейсу.

Такое распределение позволяет сделать вывод о корректной логике партиционирования: она обеспечивает эффективное хранение и обработку данных, соответствуя особенностям каждой группы поездов.

Заключение

В результате проведенного исследования была выполнена оценка эффективности использования листового партиционирования (list partitioning) в PostgreSQL на примере таблицы с данными о железнодорожных поездах, классифицированных по типу.

Экспериментальные замеры производительности показали, что при выполнении выборок с фильтрацией по партиционируемому столбцу наблюдается значительное снижение времени выполнения запросов за счет механизма отсечения нерелевантных секций на этапе планирования запросов (partition pruning).

Полученные результаты подтверждают целесообразность применения листового партиционирования в информационных системах с четкой логической структурой и выраженной категоризацией данных, где критически важна скорость выборки по ключевым признакам.

В то же время применение партиционирования требует предварительного анализа предполагаемой нагрузки и характера запросов, поскольку нерациональное проектирование схемы партиций может привести к обратному эффекту — усложнению архитектуры и снижению производительности. Таким образом, использование технологии list partitioning в PostgreSQL должно сопровождаться взвешенным подходом к использованию данной технологии с учетом характера предполагаемых запросов.

Перспективы дальнейших исследований связаны с автоматизацией выбора оптимальных параметров партиционирования, разработкой адаптивных стратегий распределения данных и сравнительным анализом различных типов партиционирования (range, hash, list) с учетом специфики прикладных задач. Особый интерес представляет применение методов машинного обучения для прогнозирования целесообразности применения той или иной схемы партиционирования на этапе проектирования базы данных.

СПИСОК ИСТОЧНИКОВ

1. PostgreSQL 17 Documentation: Table Partitioning. URL: http://www.postgresql.org/docs/17/ddl-partitioning.html (дата обращения: 01.06.2025).

2. List Partitioning // Mastering SQL Using PostgreSQL: Online Course. URL: http://postgresql.itversity.com/05_partitioning_tables_and_indexes/03_list_partitioning.html (дата обращения: 05.06.2025).

3. Raghuwanshi R. How to Use Table Partitioning to Scale PostgreSQL // EDB Postgres Tutorials. 2023. 24 January. URL: http://www.enterprisedb.com/postgres-tutorials/how-use-table-partitioning-scale-postgresql (дата обращения: 05.06.2025).

4. PostgreSQL 17 Documentation: Index Types. URL: https://www.postgresql.org/docs/current/indexes-types.html (дата обращения: 01.06.2025).

5. Soto C. PostgreSQL Table Partitioning: Boosting Performance and Management // HackerNoon. 2023. 04 October. URL: http://hackernoon.com/postgresql-table-partitioning-boosting-performance-and-management (дата обращения: 07.06.2025).

6. Fakirpure M. PostgreSQL Queries with Table Partitions // HostnExtra Learn. Updated 27 January 2024. URL: http://hostnextra.com/learn/paths/postgresql/postgresql-queries-with-table-partitions (дата обращения: 07.06.2025).

7. Blackwood-Sewell J., Soto C. When to Consider Postgres Partitioning // TigerData. 2024. 04 March. URL: https://www.tigerdata.com/learn/when-to-consider-postgres-partitioning (дата обращения: 05.06.2025).

Дата поступления: 23.06.2025
Решение о публикации: 07.08.2025

Features of List Partitioning Application in the PostgreSQL DBMS

Andrey V. Zabrodin — PhD in History, Associate Professor of the Information and Computing Systems Department. Research interests: database management systems, big data storage and processing technologies. E-mail: zabrodin@pgups.ru

Nikita Sabirov — 3rd year Bachelor's Degree Student in 09.03.01 Informatics and Computer Technology specialism. Research interests: database management systems, big data storage and processing technologies, DBMS performance. E-mail: E-mail: nikita.sabirov.30@gmail.com

Emperor Alexander I St. Petersburg State Transport University, 9, Moskovsky ave., Saint Petersburg, 190031, Russia

For citation: Zabrodin A. V., Sabirov N. Features of List Partitioning Application in the PostgreSQL DBMS. *Intellectual Technologies on Transport*, 2025, No. 3 (43), Pp. 49–57. DOI: 10.20295/2413-2527-2025-343-49-57. (In Russian)

Abstract. *As the volume of data processed in information systems increases, there is a growing need to use methods that ensure scalability, high performance, and efficient administration. One such method is table partitioning, which involves the logical division of data into independent segments. This process has been shown to optimize query execution and simplify the management of large tables.* ***Purpose:*** *to examine the particulars of employing list partitioning within the PostgreSQL database management system and to assess its effect on the performance of SQL queries.* ***Results:*** *the theoretical foundations of list partitioning, its syntactic implementation and limitations have been presented. A comparative experiment was conducted that demonstrated an enhancement in productivity when employing partitioning as opposed to a monolithic table. Typical scenarios of using the method when working with categorized data have been described.* ***Practical significance:*** *the efficiency of query processing and administration of large tables in PostgreSQL can be enhanced through the rational design of the data storage structure using list partitioning.*

Keywords: *data partitioning, PostgreSQL, list partitioning, performance, query optimization, indexing, data distribution, horizontal fragmentation, database architecture, big data*

REFERENCES

1. PostgreSQL 17 Documentation: Table Partitioning. Available at: http://www.postgresql.org/docs/17/ddl-partitioning.html (accessed: June 01, 2025).

2. List Partitioning, Mastering SQL Using PostgreSQL: Online Course. Available at: http://postgresql.itversity.com/05_partitioning_tables_and_indexes/03_list_partitioning.html (accessed: June 05, 2025).

3. Raghuwanshi R. How to Use Table Partitioning to Scale PostgreSQL, EDB Postgres Tutorials. Published online at January 24, 2023. Available at: http://www.enterprisedb.com/postgres-tutorials/how-use-table-partitioning-scale-postgresql (accessed: June 05, 2025).

4. PostgreSQL 17 Documentation: Index Types. Available at: http://www.postgresql.org/docs/current/indexes-types.html (accessed: June 01, 2025).

5. Soto C. PostgreSQL Table Partitioning: Boosting Performance and Management, HackerNoon. Published online at October 04, 2023. Available at: http://hackernoon.com/postgresql-table-partitioning-boosting-performance-and-management (accessed: June 07, 2025).

6. Fakirpure M. PostgreSQL Queries with Table Partitions, HostnExtra Learn. Updated January 27, 2024. Available at: http://hostnextra.com/learn/paths/postgresql/postgresql-queries-with-table-partitions (accessed: June 07, 2025).

7. Blackwood-Sewell J., Soto C. When to Consider Postgres Partitioning, TigerData. Published online at March 04, 2024. Available at: https://www.tigerdata.com/learn/when-to-consider-postgres-partitioning (accessed: June 05, 2025).

Received: 23.06.2025
Accepted: 07.08.2025