Разработанный программный комплекс зарегистрирован в государственном реестре программ для ЭВМ [3], позволяет выполнять дешифрирование графической информации, формируемой сканерными ТПВ-системами непосредственно в процессе выполнения съемочных работ, и может быть включен в состав автоматизированных систем обработки информации бортовой съемочной аппаратуры, что позволит повысить эффективность применения сканерных ТПВ-систем за счет повышения оперативности выполнения задач.

ЛИТЕРАТУРА

1. Saito T., Rehmsmeier M. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets // PLoS One. 2015. 10,3: e0118432

2. Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, Andrew Zisserman. The Pascal Visual Object Classes Challenge – a Retrospective // International Journal of Computer Vision manuscript No. 111, 98-136 (2015). doi:10.1007/s11263-014-07335.

3. Свидетельство о государственной регистрации программы для ЭВМ 2023611283 РФ. Программа автоматизированной обработки графической информации / Мингалев А.В., Белов А. В., Габдуллин И. М., Марданова Д. А, заявитель и правообладатель Акционерное общество «Научно-производственное объединение «Государственный институт прикладной оптики».

УДК 004.658.3

М.В. Ворончихин, асп.,
Н.А. Галанина, проф., д-р техн. наук
(ЧувГУ, г. Чебоксары, Россия)

МЕТОДЫ ОПТИМИЗАЦИИ СТРУКТУРЫ ОТЕЧЕСТВЕННЫХ БАЗ ДАННЫХ НА ОСНОВЕ POSTGRESQL

После выполнения миграции на отечественную СУБД на основе PostgreSQL важной задачей становится оптимизация структуры баз данных с целью повышения производительности и ускорения выполнения существующих бизнес-запросов на новой СУБД. Даже если вся логика приложения корректно работает на мигрированной СУБД, не будет лишним учесть все особенности PostgreSQL, чтобы дополнительно ускорить выполнение запросов и избавиться от «узких мест» в мигрированной базе данных. Применение специфичных приемов оптимизации в PostgreSQL включает в себя не только традиционные методы ин-

дексирования и нормализации, но и более специализированные подходы, предназначенные для увеличения эффективности хранения и обработки данных [1].

Для оценки эффективности предложенных методов оптимизации структуры баз данных было проведено их экспериментальное тестирование на тестовых данных, хранящихся на отечественной СУБД на основе PostgreSQL. Будем смотреть на скорость выполнения запросов и на размер таблиц до и после выполнения данных оптимизаций, а также рассмотрим влияние предложенных методов на производительность системы.

1. Оптимизация типов данных (снижение избыточности). При оптимизации структуры баз данных после миграции на PostgreSQL особое внимание должно уделяться корректному подбору типов данных для каждого столбца таблицы. Важно обеспечить баланс между эффективным использованием дискового пространства и гарантией, что диапазон значений для выбранного типа данных будет полностью покрывать все возможные значения, необходимые для работы системы. Например, вместо использования `bigint`, который занимает больше памяти, целесообразно использовать `smallint`, если значения в столбце ограничены меньшим диапазоном [2]. Также стоит учитывать использование типа `boolean` для хранения логических значений, что позволяет сэкономить место по сравнению с использованием других типов данных, таких как `char (1)`.

Пример:

```
-- До оптимизации:
CREATE TABLE payments_v1 (id serial PRIMARY KEY, number int,
status int);
-- После оптимизации:
CREATE TABLE payments_v2 (id serial PRIMARY KEY, number
smallint, status boolean);
-- Перенос данных
INSERT INTO payments_v1 (number, status)
SELECT number, status
FROM etalon_data;
INSERT INTO payments_v2 (number, status)
SELECT number, (status = 1)::boolean
FROM etalon_data;
```

Результаты:

- уменьшение размера таблицы на 25%: со 121 Мб до 91 Мб;
- ускорение запроса на 31%: с 4 сек. 026 мс. до 2 сек. 769 мс.

Однако стоит отметить, что изменение типа данных столбцов с числового на логический может повлечь за собой необходимость корректировки логики системы. Это связано с тем, что такие изменения

требуют адаптации запросов, хранимых процедур и функций, которые используют эти столбцы.

2. Манипулирование положением столбцов переменной длины. Оптимизация структуры таблицы включает в себя не только выбор типов данных, но и порядок размещения столбцов внутри строки таблицы. Несмотря на то, что с теоретической точки зрения столбцы в реляционных таблицах не упорядочены, на практике каждая СУБД сохраняет строки таблицы и столбцы внутри строк в определённой последовательности, обычно соответствующей порядку их объявления в скрипте создания таблицы.

Если обратиться к особенностям реализации PostgreSQL, можно отметить, что сначала в таблицы рекомендуется размещать столбцы фиксированной длины (например, `integer`, `boolean`), а уже за ними располагать столбцы переменной длины (например, текстовые поля: `text`, `varchar`). Такое расположение позволяет СУБД быстрее рассчитывать смещения столбцов относительно начала строки при чтении данных.

В добавление, если столбцы переменной длины часто содержат значения `NULL` или значения по умолчанию, то их размещение в конце таблицы позволяет системе не хранить эти данные физически, тем самым уменьшая общий размер таблицы на диске.

Пример:

```
-- До оптимизации
CREATE TABLE payments_v1 (id serial PRIMARY KEY, name
varchar(100), address varchar(255), status int);
-- После оптимизации
CREATE TABLE payments_v2 (id serial PRIMARY KEY, status int,
name varchar(100), address varchar(255));
-- Перенос данных
INSERT INTO payments_v1 (name, address, status)
SELECT name, address, status
FROM etalon_data;
INSERT INTO payments_v2 (status, name, address)
SELECT status, name, address
FROM etalon_data;
```

Результаты:

- уменьшение размера таблицы на 14%: со 177 Мб до 153 Мб;
- ускорение запроса на 23%: с 5 сек. 129 мс до 3 сек. 962 мс.

3. Выравнивание столбцов типа integer. Размещение столбцов внутри таблицы влияет на эффективное использование памяти и производительность. Одной из важных особенностей PostgreSQL является выравнивание столбцов определённых типов, таких как `integer`, по ширине 4-байтового слова. Это означает, что данные таких столбцов

располагаются в памяти на границах, кратных 4 байтам, для ускорения доступа и обработки. Однако чередование типов данных, таких как `boolean` и `integer`, приводит к неэффективному использованию памяти.

Для минимизации этих потерь рекомендуется группировать столбцы одного типа, особенно столбцы с выравниванием, такие как `integer`. Это позволяет уменьшить общий размер таблицы, а значит, ускорить доступ к данным.

Пример:

```
-- До оптимизации:
CREATE TABLE payments_v1 (id serial PRIMARY KEY, is_active
boolean, number int);
-- После оптимизации
CREATE TABLE payments_v2 (id serial PRIMARY KEY, number int,
is_active boolean);
-- Перенос данных
INSERT INTO payments_v1 (is_active, number)
SELECT is_active, number
FROM etalon_data;
INSERT INTO payments_v2 (number, is_active)
SELECT number, is_active
FROM etalon_data;
```

Результаты:

- уменьшение размера таблицы на 17%: со 119 Мб до 99 Мб;
- ускорение запроса на 18%: с 3 сек. 821 мс. до 3 сек. 143 мс.

Таким образом, можно сделать вывод, что предложенные методы подтвердили свою эффективность и могут быть рекомендованы для применения в реальных отечественных базах данных, основанных на PostgreSQL, для достижения оптимальных результатов производительности в процессе их эксплуатации.

ЛИТЕРАТУРА

1. Как ускорить работу PostgreSQL с помощью конфигурации базы и оптимизации запросов [Электронный ресурс] // Хабр : [сайт]. [2022]. URL: https://habr.com/ru/companies/slurm/articles/684826/ (дата обращения: 03.01.2025).

2. Оптимизация хранения данных в PostgreSQL [Электронный ресурс] // Хабр : [сайт]. [2024]. URL: https://habr.com/ru/companies/bercut/articles/859700/ (дата обращения: 03.01.2025).