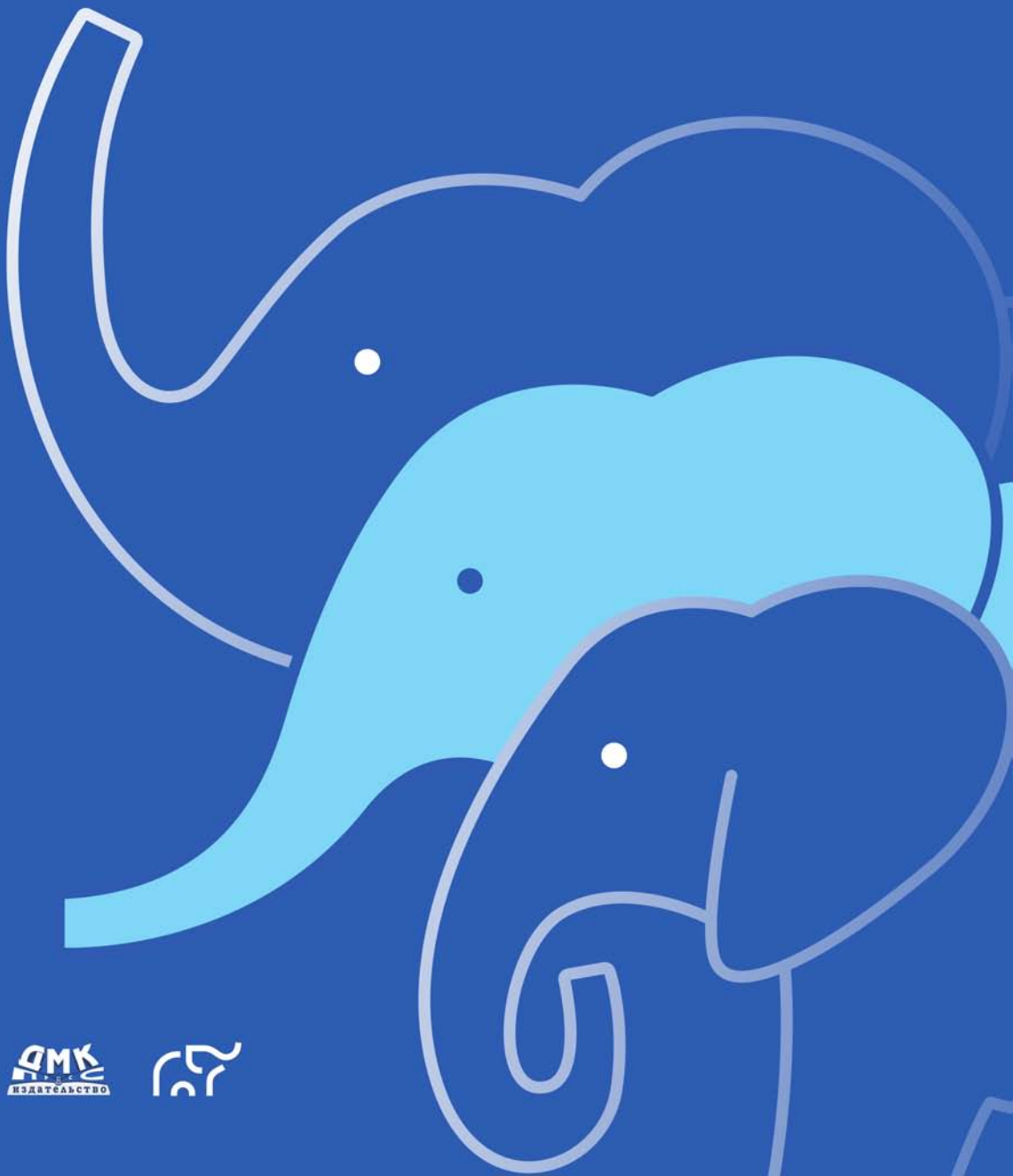
Евгений Моргунов

# PostgreSQL. Профессиональный SQL



ДМК Пресс
Издательство

УДК 004.65
ББК 32.972.134
М79

**Моргунов, Е. П.**

М79 PostgreSQL. Профессиональный SQL : учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова. — М. : ДМК Пресс, 2025. — 444 с.

ISBN 978-5-93700-373-7

Учебное пособие представляет собой вторую часть курса по языку SQL, созданного при поддержке российской компании Postgres Professional. В книге рассмотрены такие расширенные возможности СУБД PostgreSQL, как общие табличные выражения, агрегатные и оконные функции, подзапросы LATERAL, создание пользовательских функций и процедур. Материал сопровождается многочисленными практическими примерами, заданиями и упражнениями, дополняющими основной текст каждой главы. Пособие заинтересует читателей, желающих повысить базовый уровень владения языком SQL. Изучать материал пособия можно как под руководством преподавателя, так и самостоятельно.

Сайт книги: https://postgrespro.ru/education/books/advancedsql.

УДК 004.65
ББК 32.972.134

© Текст, оформление. ООО «ППГ», 2025
© Издание. ДМК Пресс, 2025

ISBN 978-5-93700-373-7

# Оглавление

**Глава 1. Введение** **11**
1.1. О книге . . . . . 11
1.2. Учебная база данных . . . . . 14
1.3. Благодарности автора . . . . . 17

**Глава 2. Общие табличные выражения** **19**
2.1. Запросы с несколькими общими табличными выражениями . . . 19
2.2. Рекурсивные общие табличные выражения . . . . . 28
2.3. Массивы в общих табличных выражениях . . . . . 49
2.4. Модификация данных в общем табличном выражении . . . . . 69
2.5. Контрольные вопросы и задания . . . . . 82

**Глава 3. Аналитические возможности PostgreSQL** **109**
3.1. Агрегатные функции . . . . . 109
3.2. Статистические функции . . . . . 126
3.3. GROUPING SETS, CUBE и ROLLUP . . . . . 137
3.4. Оконные функции . . . . . 154
3.5. Гипотезирующие агрегатные функции . . . . . 186
3.6. Контрольные вопросы и задания . . . . . 187

**Глава 4. Конструкция LATERAL команды SELECT** **225**
4.1. Подзапросы в предложении FROM . . . . . 225
4.2. Вызовы функций в предложении FROM . . . . . 234
4.3. Тип JSON и конструкция LATERAL . . . . . 236
4.4. Контрольные вопросы и задания . . . . . 244

**Глава 5. Подпрограммы** **263**
5.1. Базовые сведения о функциях . . . . . 263
5.2. Функции и зависимости между объектами базы данных . . . . . 288
5.3. Функции, возвращающие множества строк . . . . . 298
5.4. Функции с переменным числом аргументов . . . . . 305
5.5. Конструкция LATERAL и функции . . . . . 309
5.6. Категории изменчивости функций . . . . . 314
5.7. Дополнительные сведения о функциях . . . . . 347

5.8. Элементы теории принятия решений . . . . . 357
5.9. Процедуры . . . . . 367
5.10. Контрольные вопросы и задания . . . . . 372

**Предметный указатель** **435**

# Глава 1
# Введение

## 1.1. О книге

Язык SQL является общепринятым средством взаимодействия с данными. Недавно этому языку исполнилось полвека, но реляционные технологии не только не собираются сдавать своих позиций, но и продолжают развиваться и распространяться все шире. Знание SQL необходимо всем, кто имеет отношение к базам данных: и разработчикам приложений, и аналитикам, и администраторам. Его поддерживают все реляционные СУБД, и PostgreSQL не является исключением. В этой системе весьма полно реализованы требования стандарта к языку SQL и представлен целый ряд «фирменных» расширений.

Несколько лет назад при поддержке компании Postgres Professional была издана книга «PostgreSQL. Основы языка SQL»[1]. В ней рассматривались основы языка и такие вопросы, как транзакции и повышение производительности. Однако современный язык SQL выходит далеко за эти рамки, предлагая множество различных команд и возможностей, зачастую сложных для освоения, но открывающих путь к эффективной работе с данными. Поэтому и возникла идея написать продолжение базового учебника.

Вот такие темы вошли в эту книгу.

**Глава 2** посвящена общим табличным выражениям. Они кратко рассматривались в первой книге. Общие табличные выражения — очень полезное средство, которое позволяет, например, в одном запросе обрабатывать иерархические данные и даже графы общего вида. В качестве иллюстраций рассмотрены структура узлов и деталей самолета, штатное расписание компании, сеть маршрутов между городами.

[1] Моргунов, Е. П. PostgreSQL. Основы языка SQL. — СПб. : БХВ-Петербург, 2018. — 336 с. — ISBN 978-5-9775-4022-3.

В **главе 3** представлены аналитические возможности PostgreSQL. Рассмотрение начинается с агрегатных функций. Наиболее употребительные из них были показаны в базовом учебнике. Продемонстрировано использование основных статистических функций для вычисления дисперсии, среднеквадратического отклонения, моды, медианы. Важное место в этой главе занимают группировки данных. Конструкции `GROUPING SETS`, `CUBE` и `ROLLUP` позволяют свести несколько запросов с обычными группировками к одному. Большое внимание уделено оконным функциям, в частности особенностям формирования оконного кадра в разных режимах. Завершается глава обращением к гипотезирующим агрегатным функциям, то есть функциям, работающим с гипотетическими множествами строк.

Небольшая **глава 4** демонстрирует конструкцию `LATERAL` команды `SELECT`. В отличие от обычного соединения в предложении `FROM`, она позволяет подзапросу обращаться к полям из текущей строки другой таблицы, фактически образуя цикл в рамках одного запроса. Показаны также вызовы стандартных функций в предложении `FROM`.

**Глава 5** посвящена программированию на стороне сервера. Рассмотрение начинается с простых функций, возвращающих одну строку, в том числе перегруженных. Затем вводятся табличные функции, а также функции с переменным числом параметров. Показано применение модификаторов `IN`, `OUT` и `INOUT`. Эта глава — самая большая в книге, поскольку написание и применение функций имеет множество нюансов. Функции могут зависеть от других объектов базы данных, категория изменчивости влияет на решения планировщика при оптимизации запросов и на видимость изменений, при выполнении ряда условий код функции может быть подставлен в запрос, а еще функции могут участвовать в параллельном выполнении. В качестве иллюстрации применения функций для поддержки принятия решений показан выбор наилучших альтернатив с помощью формирования множества Парето. Основной материал главы посвящен функциям, но в завершающем разделе обсуждаются и процедуры, которые во многом схожи с функциями, но имеют целый ряд отличий.

Книгу можно использовать как в процессе изучения технологий баз данных в образовательном учреждении (в бакалавриате и магистратуре), так и для самостоятельной подготовки.

Большое внимание в учебнике уделяется заданиям и упражнениям. Они различаются по объему и сложности, а также имеют разную специфику: многие предполагают написание SQL-запроса, однако есть и такие, которые заключаются в проведении экспериментов и изучении взаимосвязей между объектами базы данных. Упражнения зачастую дополняют и развивают основной текст главы, в них обсуждаются различные тонкости, частные случаи, полезные приемы и поучительные ситуации. Упражнения в конце каждой главы расположены в основном в соответствии с порядком изложения материала. Но есть и комплексные упражнения, выполнение которых требует освоения нескольких разделов или даже всей главы в целом.

Многие положения книги подкрепляются ссылками на соответствующие разделы документации, ведь именно она — первоисточник знаний. Эти разделы стоит просматривать, поскольку книга не является заменой документации и не пытается ее пересказывать.

Для учебных заведений можно предложить следующее распределение времени, исходя из того, что на изучение материала пособия будет отведено 36 академических часов:

| | |
|---|---|
| Глава 2. Общие табличные выражения | 6 часов |
| Глава 3. Аналитические возможности PostgreSQL | 10 часов |
| Глава 4. Конструкция LATERAL команды SELECT | 4 часа |
| Глава 5. Подпрограммы | 16 часов |

Для изучения материала учебника необходимо установить СУБД PostgreSQL (или Postgres Pro Standard) и развернуть учебную базу данных «Авиаперевозки». Скачать PostgreSQL можно с официального сайта postgresql.org/download, на котором приведена и инструкция по установке. Для использования Postgres Pro Standard в учебных заведениях нужно получить академическую лицензию, которую компания Postgres Professional предоставляет бесплатно[1]. Учебная база данных и ее полное описание доступны на сайте компании: postgrespro.ru/education/demodb. Необходима версия от 15.08.2017, вариант small.

На странице книги (postgrespro.ru/education/books/advancedsql) предлагается виртуальная машина, в которой уже установлены СУБД и учебная база данных.

[1] Для этого требуется отправить заявку на academy@postgrespro.ru.

Все SQL-запросы, приведенные в книге, были выполнены в среде PostgreSQL версии 17. Ссылки на разделы документации относятся именно к этой версии.

Большинство запросов, скорее всего, будут корректно выполняться и в более новых версиях PostgreSQL. В старых версиях результаты могут отличаться от приведенных в книге. Например, в книге используется появившийся в версии 17 перегруженный вариант функции `random`, позволяющий задать диапазон случайных значений.

Ряд запросов, представленных в книге, содержат столбцы типа `timestamptz`. Такие запросы могут возвращать результаты, отличающиеся от приведенных в книге для часового пояса Asia/Krasnoyarsk (+07), поскольку зависят от настройки часового пояса на компьютере читателя.

Команды, вводимые пользователем как в среде операционной системы, так и в среде утилиты psql, выделяются в пособии полужирным моноширинным шрифтом, а результаты их выполнения — светлым. Например:

```
postgres$ psql -d demo -U postgres
```

или

```
SELECT min( days_of_week ), max( days_of_week )
FROM routes;
 min | max
-----+-----
 {1} | {7}
(1 строка)
```

## 1.2. Учебная база данных

В качестве учебной базы данных используется база данных «Авиаперевозки», разработанная в компании Postgres Professional. Она представлена в трех вариантах: малом (данные о полетах за месяц), среднем (полеты за квартал) и большом (за год). Эта база данных имеет целый ряд привлекательных качеств:

- данные в ней правдоподобны (например, частоты фамилий и имен пассажиров аналогичны тем, что имеют место в России);

- данных много (даже в малом варианте базы данных есть таблицы, содержащие сотни тысяч записей, а в одной из них — больше миллиона);
- база не слишком простая и не слишком сложная (число сущностей не превышает десятка);
- проиллюстрированы разные приемы проектирования (связь «один к одному», суррогатный ключ, контролируемая избыточность и др.);
- используются разнообразные типы данных (числа, строки, даты и время, интервалы, массивы, JSON).

Легенда такова. Некая авиакомпания совершает пассажирские авиаперевозки по России. Все модели авиапарка компании собраны в представлении «Самолеты» (`aircrafts`), созданном на основе таблицы `aircrafts_data`. Коды и названия моделей соответствуют реальной практике. В таблице «Места» (`seats`) записаны схемы салонов, одинаковые для всех самолетов одной модели.

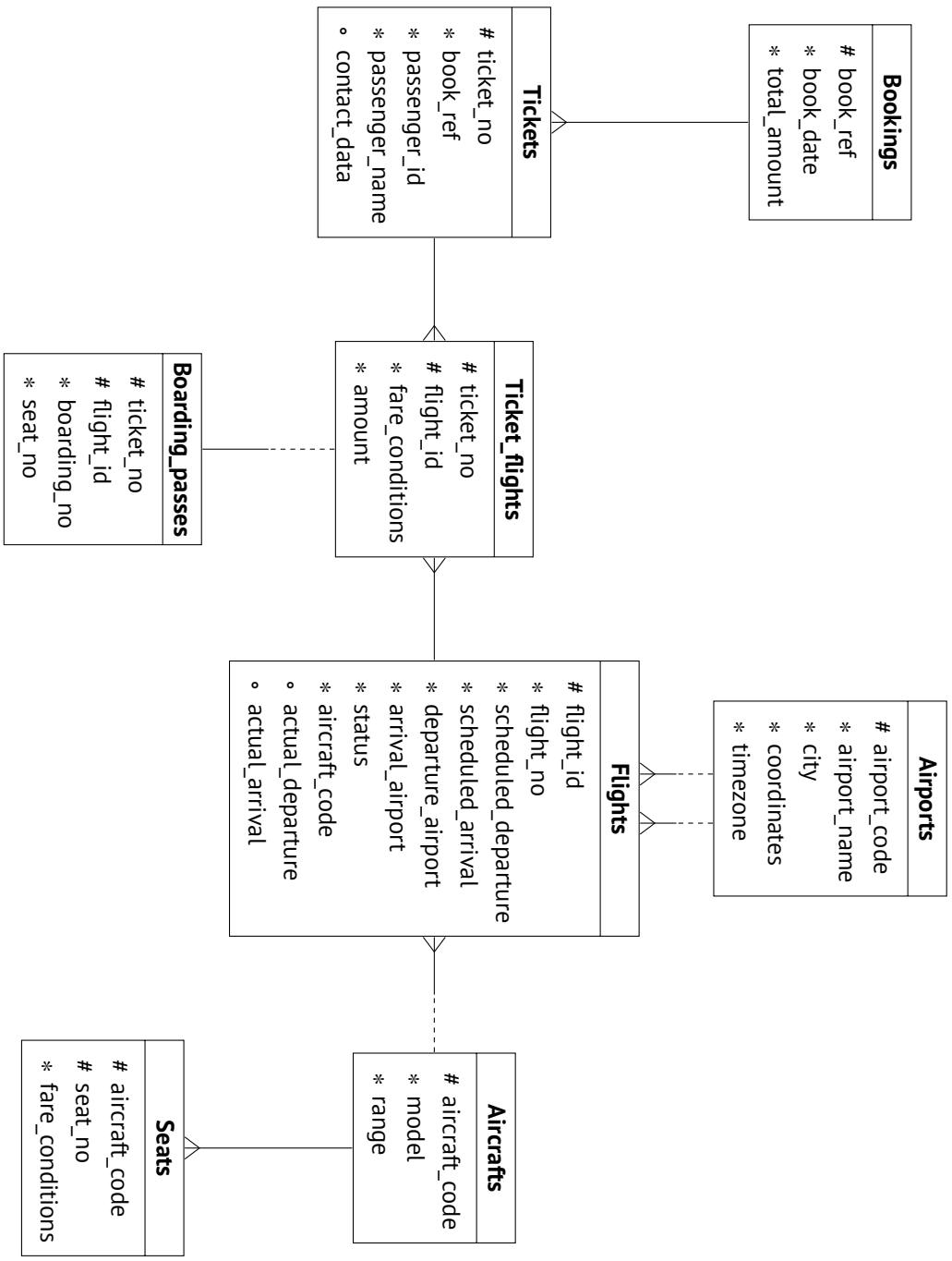
Представление «Аэропорты» (`airports`) на основе таблицы `airports_data` показывает область присутствия авиакомпании. Используются коды аэропортов, присваиваемые международной ассоциацией ИАТА. Расписание рейсов содержится в таблице «Рейсы» (`flights`). Ее первичным ключом является столбец «Идентификатор рейса» (`flight_id`). Это суррогатный ключ. Представление «Рейсы» (`flights_v`), созданное на основе этой и ряда других таблиц, выдает гораздо больше информации, например показывает местное время вылета и прибытия рейсов в соответствующих аэропортах.

Маршруты, то есть пары аэропортов, между которыми авиакомпания выполняет рейсы, можно увидеть в представлении «Маршруты» (`routes`).

Пассажиры бронируют билеты. Эти операции отражаются в таблице «Бронирования» (`bookings`). Столбец «Номер бронирования» (`book_ref`) содержит коды операций, состоящие из шести шестнадцатеричных цифр.

В каждой операции бронирования могут быть оформлены один или несколько билетов (возможно, на разных пассажиров), которые записываются в таблицу «Билеты» (`tickets`). Номера билетов представляют собой тринадцатизначные строки, состоящие из цифр.

Перелеты, входящие в оформленные билеты, записываются в таблицу «Перелеты» (`ticket_flights`). Сумма значений столбца «Стоимость перелета» (`amount`)


Bookings
# book_ref
* book_date
* total_amount
Tickets
# ticket_no
* book_ref
* passenger_id
* passenger_name
◦ contact_data
Ticket_flights
# ticket_no
# flight_id
* fare_conditions
* amount
Boarding_passes
# ticket_no
# flight_id
* boarding_no
* seat_no
Flights
# flight_id
* flight_no
* scheduled_departure
* scheduled_arrival
* departure_airport
* arrival_airport
* status
* aircraft_code
◦ actual_departure
◦ actual_arrival
Airports
# airport_code
* airport_name
* city
* coordinates
* timezone
Aircrafts
# aircraft_code
* model
* range
Seats
# aircraft_code
# seat_no
* fare_conditions

в записях, соответствующих конкретной операции бронирования, должна быть равна значению поля «Полная сумма бронирования» (total_amount) в строке таблицы «Бронирования» (bookings), описывающей эту операцию. Это пример контролируемой избыточности.

Регистрация на рейсы отражается в таблице «Посадочные талоны» (boarding_passes). Она связана с таблицей «Перелеты» (ticket_flights) отношением «один к одному». Столбец «Статус рейса» принимает только ограниченный набор значений, например Departed или Arrived.

Все перечисленные таблицы и представления созданы в схеме bookings. В ней же имеется функция now, представляющая момент выгрузки данных, или *текущий момент*, который можно использовать в запросах вместо стандартной функции now.

## 1.3. Благодарности автора

Хотя на обложке книги указан один человек, такая работа не выполняется единолично. Многие сотрудники компании Postgres Professional помогали автору на разных этапах ее выполнения.

Огромный вклад в успешное завершение этого труда внес Егор Рогов. Он проделал большую работу в качестве редактора: предложил целый ряд структурных изменений текста книги, множество более точных формулировок ключевых понятий, улучшений SQL-запросов и модификаций упражнений, приведенных в книге. Егор выполнил верстку текста, подготовил предметный указатель, нарисовал иллюстрации, помогающие разобраться в сложных понятиях и конструкциях языка SQL.

Павел Толмачев и Павел Лузанов просматривали черновые варианты текста книги и давали полезные комментарии. Включить в книгу главу «Конструкция LATERAL команды SELECT» предложил Павел Лузанов.

В обсуждении названия книги принял участие и высказал полезные идеи Александр Мелешко.

Содержание книги оттачивалось в ходе проведения пробного курса по ее материалу со студентами Алтайского государственного технического университета

и Алтайского государственного университета. Полезные замечания высказывали Михаил Козлов, Олег Целебровский и Павел Гилев, ставшие сотрудниками компании.

И конечно, книга не состоялась бы без всесторонней поддержки и внимания со стороны руководителей компании Postgres Professional Олега Бартунова и Ивана Панченко.

Замечания и предложения можно направлять по адресу edu@postgrespro.ru.

Мы надеемся, что это учебное пособие внесет вклад в повышение уровня квалификации читателя и расширение его профессионального кругозора.

# Глава 2
# Общие табличные выражения

Общие табличные выражения — важное выразительное средство языка SQL, которое позволяет упростить сложный SQL-запрос, разбивая его на обозримые фрагменты, выполняющие отдельные подзадачи. В результате запрос становится более понятным, а зачастую и более эффективным. Не менее важно, что с помощью общих табличных выражений SQL-запрос может обрабатывать данные в цикле, что повышает вычислительные возможности языка SQL.

## 2.1. Запросы с несколькими общими табличными выражениями

В сложных SQL-запросах могут присутствовать подзапросы, в том числе вложенные. Они не только затрудняют понимание работы запроса, но и могут приводить к замедлению его выполнения. К счастью, в современном языке SQL есть средство, позволяющее успешно преодолевать эту проблему.

Для иллюстрации сказанного предположим, что отдел развития нашей авиакомпании хочет определить пассажиропоток (количество пассажиров в единицу времени), проходящий через аэропорты, в которые авиакомпания совершает полеты. От пассажиропотока зависит необходимая численность персонала и размеры платежей за использование наземных служб аэропортов.

Пассажиропоток идет в двух направлениях — вылет пассажиров и их прибытие, — поэтому для каждого аэропорта нам нужно получить общее количество как вылетевших пассажиров, так и прибывших. Для нас не имеет значения, каким рейсом вылетает пассажир; важно лишь то, откуда и куда он летит. Установить это можно, соединив таблицы «Рейсы» (`flights`) и «Посадочные талоны» (`boarding_passes`) по столбцу `flight_id`. Первая таблица содержит сведения об аэропортах вылета и прибытия каждого рейса, а результат соединения будет содержать сведения об аэропортах вылета и прибытия для каждого пассажира. Если теперь сгруппировать полученные строки по коду аэропорта отправления

или прибытия и подсчитать их, то мы решим поставленную задачу. Поскольку это две независимые группировки, придется выполнить два отдельных подзапроса, а в главном запросе соединить их результаты по условию совпадения аэропортов отправления и прибытия, поскольку для каждого аэропорта требуется получить два показателя: число пассажиров, вылетевших из него, и число пассажиров, прибывших в него.

Будем учитывать только прибывшие и вылетевшие рейсы, то есть рейсы со статусом Arrived и Departed. В качестве отчетного периода возьмем один месяц от текущего момента, который в нашей базе данных определяется функцией now из схемы bookings. Попадание рейса в заданный интервал будем определять по плановому времени вылета.

Кроме того, чтобы вывести не только код аэропорта, но и его название, воспользуемся таблицей «Аэропорты» (airports).

Получим такой запрос:

```
SELECT td.departure_airport AS "Код а/п",
       a.airport_name AS "Аэропорт",
       td.dep_pass_count AS "Вылетели",
       ta.arr_pass_count AS "Прибыли",
       td.dep_pass_count + ta.arr_pass_count AS "Всего"
FROM (
       SELECT f.departure_airport, count( * ) AS dep_pass_count
       FROM boarding_passes AS bp
         JOIN flights AS f ON f.flight_id = bp.flight_id
       WHERE f.status IN ( 'Arrived', 'Departed' )
         AND f.scheduled_departure >= bookings.now() - '1 mon'::interval
       GROUP BY f.departure_airport
     ) AS td -- total_departed_pass_counts
     JOIN
     (
       SELECT f.arrival_airport, count( * ) AS arr_pass_count
       FROM boarding_passes AS bp
         JOIN flights AS f ON f.flight_id = bp.flight_id
       WHERE f.status IN ( 'Arrived', 'Departed' )
         AND f.scheduled_departure >= bookings.now() - '1 mon'::interval
       GROUP BY f.arrival_airport
     ) AS ta -- total_arrived_pass_counts
     ON td.departure_airport = ta.arrival_airport
     JOIN airports AS a ON a.airport_code = td.departure_airport
ORDER BY "Всего" DESC;
```

Включив секундомер в утилите psql, определим время выполнения запроса. Для получения стабильного времени лучше выполнить запрос несколько раз.

```
\timing on
 Код а/п |        Аэропорт        | Вылетели | Прибыли | Всего
---------+-------------------------+----------+---------+--------
 SVO     | Шереметьево             |    93205 |   77347 | 170552
 DME     | Домодедово              |    82775 |   75767 | 158542
 LED     | Пулково                 |    35505 |   35575 |  71080
 VKO     | Внуково                 |    36694 |   29335 |  66029
 OVB     | Толмачёво               |    25137 |   27477 |  52614
 SVX     | Кольцово                |    19725 |   20628 |  40353
 AER     | Сочи                    |    16520 |   19182 |  35702
 PEE     | Пермь                   |    17006 |   18122 |  35128
 KHV     | Хабаровск-Новый         |    12130 |   12855 |  24985
 KZN     | Казань                  |     9656 |   10473 |  20129
 ...
 GDX     | Магадан                 |      154 |     253 |    407
 KXK     | Хурба                   |      237 |     158 |    395
 KLF     | Калуга                  |      186 |     168 |    354
 UCT     | Ухта                    |      122 |     169 |    291
 USK     | Усинск                  |       76 |     131 |    207
 CEE     | Череповец               |       78 |      80 |    158
 NYA     | Нягань                  |       51 |      82 |    133
 RGK     | Горно-Алтайск           |       35 |      51 |     86
 SWT     | Стрежевой               |        8 |       8 |     16
(94 строки)
Время: 360,589 мс
```

Попробуем ускорить выполнение этого запроса. В подзапросах сначала выполняется соединение таблиц «Посадочные талоны» (boarding_passes) и «Рейсы» (flights), а затем группировка. Число строк в этих таблицах различается на порядок, поэтому для уменьшения количества соединяемых строк можно заранее сгруппировать строки таблицы «Посадочные талоны» (boarding_passes) по идентификаторам рейсов (flight_id) во вложенном подзапросе. Для каждого рейса эта группировка будет содержать ровно одну строку с количеством пассажиров. После этого каждая строка таблицы «Рейсы» (flights), отобранная условием в предложении WHERE, будет соединяться только с одной строкой полученной группировки. Конечно, в подзапросах первого уровня мы должны будем заменить функцию count на sum, чтобы просуммировать количество пассажиров отдельных рейсов и получить итоговую численность вылетевших и прибывших.

```
 SELECT td.departure_airport AS "Код а/п",
       a.airport_name AS "Аэропорт",
       td.dep_pass_count AS "Вылетели",
       ta.arr_pass_count AS "Прибыли",
       td.dep_pass_count + ta.arr_pass_count AS "Всего"
FROM (
       SELECT f.departure_airport, sum( pass_count ) AS dep_pass_count
       FROM (
              SELECT flight_id, count( * ) AS pass_count
              FROM boarding_passes
              GROUP BY flight_id
            ) AS bp
           JOIN flights AS f ON f.flight_id = bp.flight_id
       WHERE f.status IN ( 'Arrived', 'Departed' )
         AND f.scheduled_departure >= bookings.now() - '1 mon'::interval
       GROUP BY f.departure_airport
     ) AS td -- total_departed_pass_counts
     JOIN
     (
       SELECT f.arrival_airport, sum( pass_count ) AS arr_pass_count
       FROM (
              SELECT flight_id, count( * ) AS pass_count
              FROM boarding_passes
              GROUP BY flight_id
            ) AS bp
           JOIN flights AS f ON f.flight_id = bp.flight_id
       WHERE f.status IN ( 'Arrived', 'Departed' )
         AND f.scheduled_departure >= bookings.now() - '1 mon'::interval
       GROUP BY f.arrival_airport
     ) AS ta -- total_arrived_pass_counts
     ON td.departure_airport = ta.arrival_airport
     JOIN airports AS a ON a.airport_code = td.departure_airport
ORDER BY "Всего" DESC;
```

Время выполнения заметно уменьшилось по сравнению с исходным запросом:

```
 ...
Время: 211,355 мс
```

Ускорение достигнуто, но за счет усложнения запроса. Однако можно и упростить его, не потеряв в скорости выполнения. Вложенный подзапрос, агрегирующий таблицу «Посадочные талоны» (boarding_passes), выполняется дважды. Если бы мы смогли сохранить результат первого выполнения этого подзапроса во временную таблицу, нам не пришлось бы выполнять то же самое действие повторно.

Как реализовать эту идею? Нам поможет предложение WITH, которое вводит в запрос *общее табличное выражение* — Common Table Expression (CTE). Как сказано в разделе документации 7.8 «Запросы WITH (Общие табличные выражения)», результат вычисления этого выражения можно рассматривать в качестве временной таблицы, существующей только во время выполнения всего запроса. Сразу добавим, что это не то же самое, что временная таблица, которая создается командой CREATE TEMP TABLE.

Перенесем вложенный подзапрос в конструкцию WITH, где он получит имя bp. В главном запросе можно ссылаться на это имя, как на имя обычной таблицы.

Вот как теперь будет выглядеть наш запрос:

```
WITH bp AS
( SELECT flight_id, count( * ) AS pass_count                    ⎫
  FROM boarding_passes                                          ⎬ общее табличное выражение
  GROUP BY flight_id                                            ⎭
)
SELECT td.departure_airport AS "Код а/п",
       a.airport_name AS "Аэропорт",
       td.dep_pass_count AS "Вылетели",
       ta.arr_pass_count AS "Прибыли",
       td.dep_pass_count + ta.arr_pass_count AS "Всего"
FROM (
       SELECT f.departure_airport, sum( pass_count ) AS dep_pass_count
       FROM bp
         JOIN flights AS f ON f.flight_id = bp.flight_id
       WHERE f.status IN ( 'Arrived', 'Departed' )
         AND f.scheduled_departure >= bookings.now() - '1 mon'::interval
       GROUP BY f.departure_airport
     ) AS td -- total_departed_pass_counts
     JOIN
     (
       SELECT f.arrival_airport, sum( pass_count ) AS arr_pass_count
       FROM bp
         JOIN flights AS f ON f.flight_id = bp.flight_id
       WHERE f.status IN ( 'Arrived', 'Departed' )
         AND f.scheduled_departure >= bookings.now() - '1 mon'::interval
       GROUP BY f.arrival_airport
     ) AS ta -- total_arrived_pass_counts
     ON td.departure_airport = ta.arrival_airport
     JOIN airports AS a ON a.airport_code = td.departure_airport
ORDER BY "Всего" DESC;
```

Мы избежали вложенных подзапросов, в результате наш запрос стало легче читать и, что очень важно, время его выполнения еще уменьшилось:

```
 ...
Время: 125,967 мс
```

А где можно увидеть подзапрос, помещенный в конструкцию WITH, в плане запроса? Он отображается под строкой CTE bp, относящейся к самому верхнему узлу плана:

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Sort
   Sort Key: (((sum(bp.pass_count)) + ta.arr_pass_count)) DESC
   CTE bp
     -> HashAggregate
          Group Key: boarding_passes.flight_id
          -> Seq Scan on boarding_passes
   -> Hash Join
        Hash Cond: (f.departure_airport = ml.airport_code)
        -> Merge Join
            Merge Cond: (f.departure_airport = ta.arrival_airport)
            -> Sort
                Sort Key: f.departure_airport
                -> HashAggregate
                    Group Key: f.departure_airport
                    -> Hash Join
                        Hash Cond: (bp.flight_id = f.flight_id)
                        -> CTE Scan on bp
                        -> Hash
                            -> Seq Scan on flights f
                                Filter: (((status)::text = ANY ('{Arrived,Departed}'::...
            -> Sort
                Sort Key: ta.arrival_airport
                -> Subquery Scan on ta
                    -> HashAggregate
                        Group Key: f_1.arrival_airport
                        -> Hash Join
                            Hash Cond: (bp_1.flight_id = f_1.flight_id)
                            -> CTE Scan on bp bp_1
                            -> Hash
                                -> Seq Scan on flights f_1
                                    Filter: (((status)::text = ANY ('{Arrived,Departed...
        -> Hash
            -> Seq Scan on airports_data ml
(33 строки)
```

Временная таблица bp используется в запросе дважды (узлы CTE Scan on bp) и поэтому по умолчанию материализуется. Это означает, что результат первого — и единственного — выполнения подзапроса bp сохраняется, а при повторном обращении к таблице bp берутся уже готовые материализованные данные.

Если же результат вычисления общего табличного выражения используется в запросе только один раз, то по умолчанию материализации не происходит. В таком случае в плане не будет узлов CTE Scan: подзапрос встраивается непосредственно в запрос и оптимизируется вместе с ним.

PostgreSQL позволяет потребовать материализации, указав предложение MATERIALIZED, а для ее отмены служит предложение NOT MATERIALIZED. В ряде ситуаций «ручное» управление материализацией позволяет ускорить выполнение запроса. Конкретные примеры показаны в упражнении 1 (с. 82). А в запросах, рассматриваемых сейчас, мы будем полагаться на решения, принимаемые по умолчанию.

Нельзя ли еще упростить наш запрос? Да, это возможно.

В конструкции WITH можно написать несколько подзапросов, при этом один подзапрос может ссылаться на другие, то есть обращаться к результатам их выполнения. Конечно, циклические ссылки не допускаются.

Давайте перенесем оба подзапроса из главного запроса в конструкцию WITH. Теперь все подготовительные действия по группировке данных выполняются в конструкции WITH, а в главном запросе полученные результаты используются так, как будто они хранятся в таблицах.

```
WITH bp AS
( SELECT flight_id, count( * ) AS pass_count
  FROM boarding_passes
  GROUP BY flight_id
),
total_departed_pass_counts AS
( SELECT f.departure_airport, sum( pass_count ) AS dep_pass_count
  FROM bp
    JOIN flights AS f ON f.flight_id = bp.flight_id
  WHERE f.status IN ( 'Arrived', 'Departed' )
    AND f.scheduled_departure >= bookings.now() - '1 mon'::interval
  GROUP BY f.departure_airport
),
```

```
total_arrived_pass_counts AS
( SELECT f.arrival_airport, sum( pass_count ) AS arr_pass_count
  FROM bp
    JOIN flights AS f ON f.flight_id = bp.flight_id
  WHERE f.status IN ( 'Arrived', 'Departed' )
    AND f.scheduled_departure >= bookings.now() - '1 mon'::interval
  GROUP BY f.arrival_airport
)
SELECT td.departure_airport AS "Код а/п",
       a.airport_name AS "Аэропорт",
       td.dep_pass_count AS "Вылетели",
       ta.arr_pass_count AS "Прибыли",
       td.dep_pass_count + ta.arr_pass_count AS "Всего"
FROM total_departed_pass_counts AS td
  JOIN total_arrived_pass_counts AS ta ON td.departure_airport = ta.arrival_airport
  JOIN airports AS a ON a.airport_code = td.departure_airport
ORDER BY "Всего" DESC;

 ...
Время: 128,671 мс
```

Эта модификация не ускоряет запрос, но делает его текст более понятным.

План новой версии запроса будет точно таким же, как и предыдущей версии. Но почему это так? Почему в плане не упоминаются таблицы total_departed_pass_counts и total_arrived_pass_counts? Когда результат подзапроса, помещенного в конструкцию WITH, используется в главном запросе (или в другом подзапросе внутри WITH) только один раз, этот подзапрос по умолчанию не материализуется, а просто встраивается в общий план и оптимизируется вместе с ним. А вот подзапрос с именем bp по-прежнему используется более одного раза, пусть и не в главном запросе, а в подзапросах, находящихся, как и он сам, в конструкции WITH. Поэтому он и в этой версии запроса материализуется и отражается в плане под строкой CTE bp.

В рассмотренной версии запроса остался потенциал для ускорения: соединение временной таблицы bp и таблицы «Рейсы» (flights) выполняется дважды (в подзапросах total_departed_pass_counts и total_arrived_pass_counts). Давайте вынесем эту операцию в отдельный подзапрос flights_pass_counts в конструкции WITH.

Вот что получится: