**Авторские права**
© Postgres Professional, 2019 год.
Авторы: Егор Рогов, Павел Лузанов

**Использование материалов курса**
Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

**Обратная связь**
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

**Отказ от ответственности**
Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или непрямым, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

СКАЧАНО С WWW.SW.HELP - ПРИСОЕДИНЯЙСЯ!

# Темы

В-деревья

Индексное сканирование

Исключительно индексное сканирование

# B-деревья

Структура

Использование

# B-дерево

Postgres PROFESSIONAL

## Индекс

- вспомогательная структура во внешней памяти
- сопоставляет ключи и идентификаторы строк таблицы

## Устройство: дерево поиска

- сбалансированное
- сильно ветвистое
- только сортируемые типы данных (операции «больше», «меньше»)
- результаты автоматически отсортированы

## Использование

- ускорение доступа
- поддержка ограничений целостности



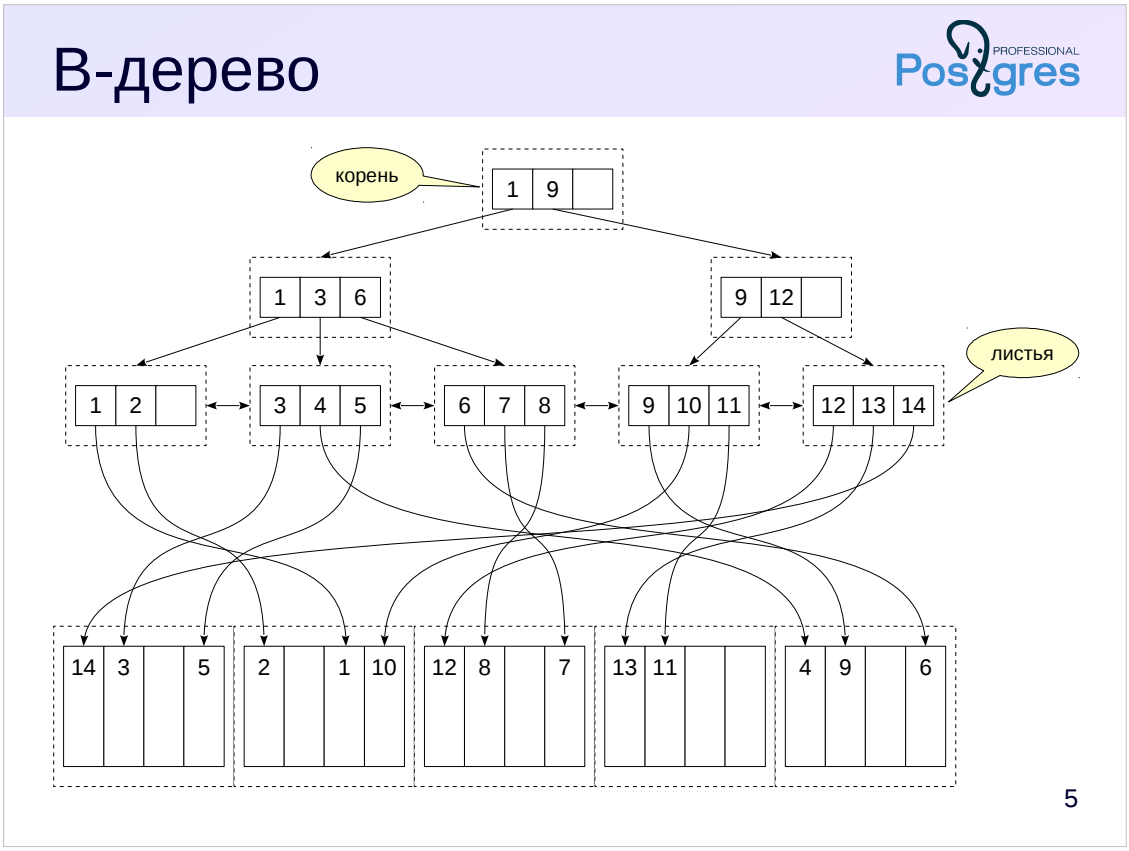
В этом модуле мы будем рассматривать только один из доступных в PostgreSQL типов индексов: B-дерево (B-tree). Это самый часто применяющийся на практике тип индекса.

Как и все индексы в PostgreSQL, B-дерево является вторичной структурой — индекс не содержит в себе никакой информации, которую нельзя было бы получить из самой таблицы. Индекс можно удалить и пересоздать. Если бы не поддержка ограничений целостности (первичные и уникальные ключи), можно было бы сказать, что индексы влияют только на производительность, но не на логику работы.

Любой индекс сопоставляет значения проиндексированных полей (ключи поиска) и идентификаторы строк таблицы. Для этого в индексе B-tree строится упорядоченное дерево ключей, в котором можно быстро найти нужный ключ, а вместе с ним — и ссылку на табличную строку. Но проиндексировать можно только данные, допускающие сортировку (должны быть определены операции «больше», «меньше»). Например, числа, строки и даты могут быть проиндексированы, а точки на плоскости — нет (для них существуют другие типы индексов).

Особенностями B-дерева является его сбалансированность (постоянная глубина) и сильная ветвистость. Хотя размер дерева зависит от проиндексированных столбцов, на практике деревья обычно имеют глубину не больше 4–5.

Кроме поддержки ограничений целостности, задача B-деревьев (как и всех индексов) состоит в ускорении доступ к данным.




На слайде представлен пример B-дерева (верхняя часть рисунка). У дерева есть корневая, внутренние и листовые страницы. Страницы состоят из индексных строк (или элементов), в которых содержатся

- значения столбцов, по которым построен индекс (ключи);

- ссылки — либо на страницы более низкого уровня (во внутренних страницах индекса), либо на строки таблицы (в листовых страницах).

Внутри страниц все ключи упорядочены.

Индексная страница может быть заполнена не полностью. Свободное место используется для вставки в индекс новых значений. Если же на странице не хватает места — она разделяется на две новых страницы. Разделившиеся страницы никогда не объединяются, и это в некоторых случаях может приводить к разрастанию индекса.

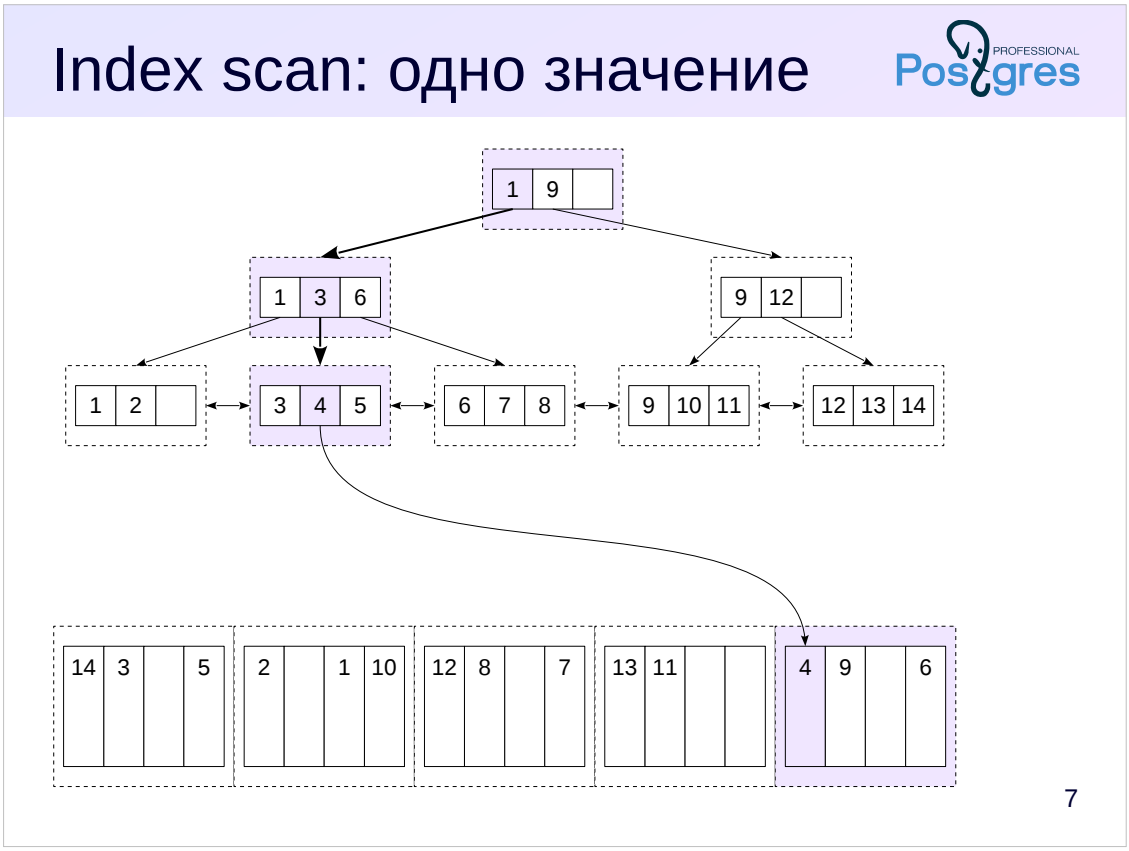
Листовые страницы дерева объединены двунаправленным списком. Это позволяет быстро переходить от одного значения к следующему за ним (или предыдущему).

# Индексное сканирование

Индексное сканирование (Index Scan)

- чтение одного значения
- чтение диапазона значений

Параллельное индексное сканирование (Parallel Index Scan)




Рассмотрим поиск одного значения с помощью индекса. Например, нам нужно найти в таблице строку, где значение проиндексированного столбца равно четырем.

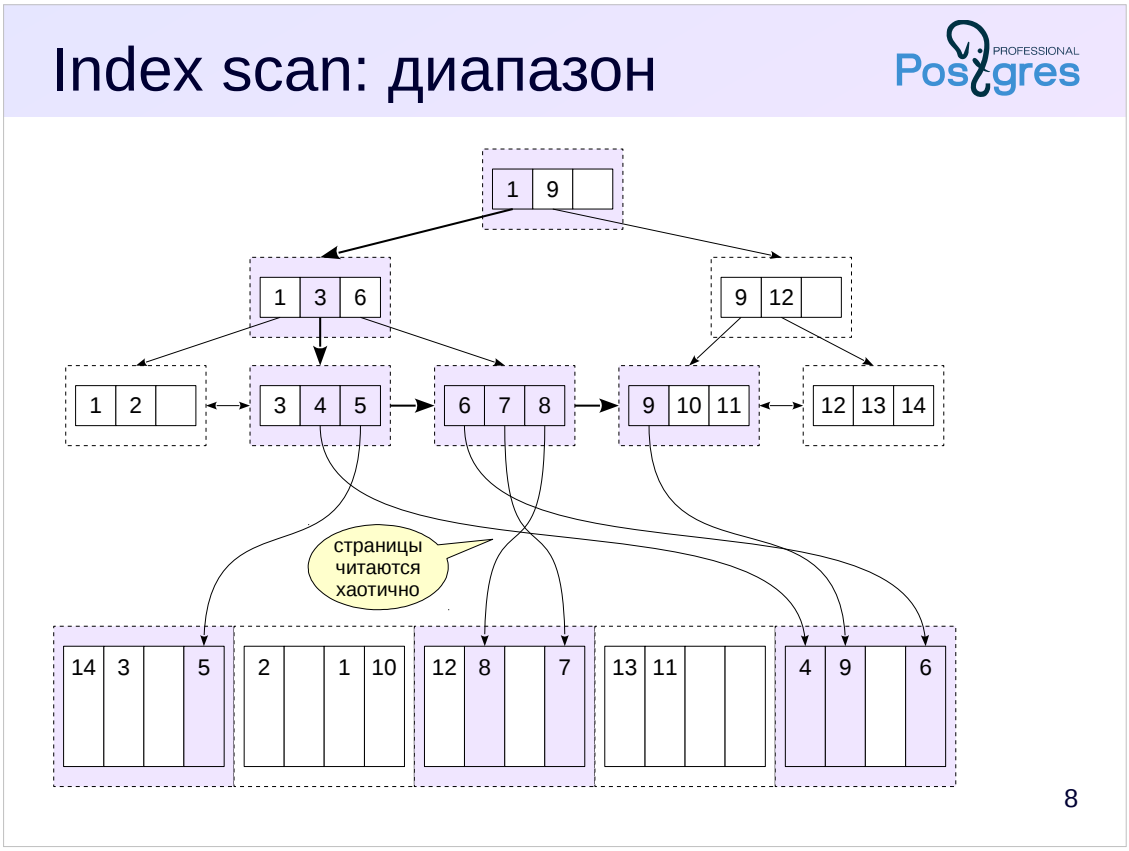
Начинаем с корня дерева. Строки в корневой странице определяют диапазоны значений ключей в нижележащих страницах: «от 1 до 9» и «9 и больше». Нам подходит диапазон «от 1 до 9», что соответствует строке с ключом 1. Заметим, что ключи хранятся упорядоченно, следовательно, поиск по странице выполняется очень эффективно.

Ссылка из найденной строки приводит нас к странице второго уровня. В ней мы находим диапазон «от 3 до 6» (ключ 3) и переходим к странице третьего уровня.

Эта страница является листовой. В ней мы находим ключ, равный 4, и переходим на страницу таблицы.

В действительности процесс более сложен: мы не говорили про неуникальные ключи, про проблемы одновременного доступа и так далее. Но не будем углубляться в детали реализации.

Обратите внимание: на этой и следующих иллюстрациях цветом выделены те строки и те страницы, которые потребовалось прочитать.



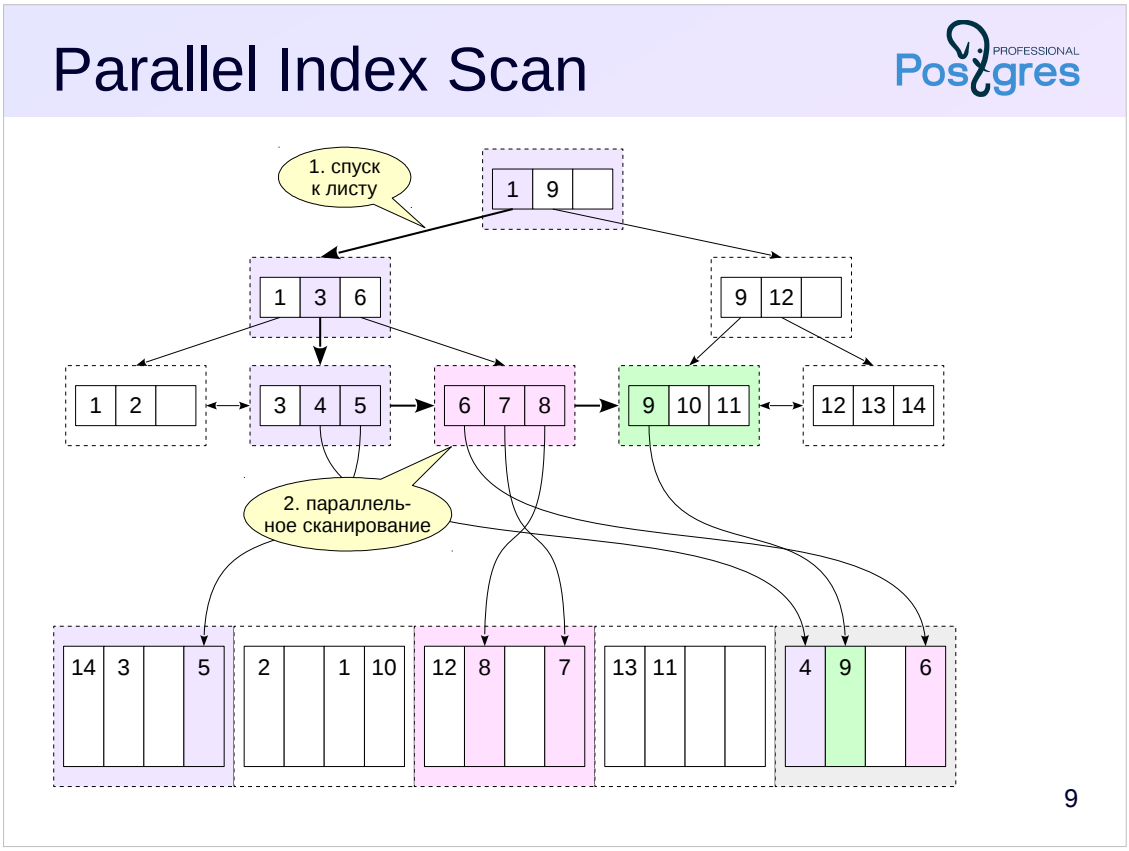

В-дерево позволяет эффективно искать не только отдельные значения, но и диапазоны значений (по условиям «меньше/больше», «меньше/больше или равно», «between»).

Вот как это происходит. Сначала мы ищем крайний ключ условия. Например, для условия «от 4 до 9 (включительно)» мы можем выбрать значение 4 или 9, а для условия «меньше 9» надо взять 9. Затем спускаемся до листовой страницы индекса так, как мы рассматривали в предыдущем примере, и получаем первое значение из таблицы.

Дальше остается двигаться по листовым страницам индекса вправо (или влево, в зависимости от условия), перебирая строки этих страниц до тех пор, пока мы не встретим ключ, выпадающий из диапазона. В нашем случае мы перебираем ключи 5, 6, и так далее до 9. Встретив ключ 10, прекращаем поиск.

Нам помогают два свойства: упорядоченность ключей на всех страницах и связанность листовых страниц двунаправленным списком.

Обратите внимание, что к одной и той же табличной странице нам пришлось обращаться несколько раз. Мы прочитали последнюю страницу таблицы (значение 4), затем первую (5), затем опять последнюю (6) и так далее.




Индексный доступ может выполняться в параллельном режиме. Это происходит в два этапа. Сначала ведущий процесс спускается от корня дерева к листовой странице. Затем рабочие процессы выполняют параллельное чтение листовых страниц индекса, двигаясь по указателям.

Процесс, прочитавший индексную страницу, выполняет и чтение необходимых табличных страниц. При этом может получиться так, что одну и ту же табличную страницу прочитают несколько процессов (этот пример показан на иллюстрации: последняя табличная страница содержит строки, ссылки на которые ведут от нескольких индексных страниц, прочитанных разными процессами). Конечно, сама страница будет находиться в буферном кэше в одном экземпляре.

# Число рабочих процессов

Равно нулю (параллельный план не строится)

если *размер выборки* < *min_parallel_index_scan_size* = 512kB

Фиксировано

если для таблицы указан параметр хранения *parallel_workers*

Вычисляется по формуле

$1 + \lfloor \log_3(\textit{размер выборки} / \textit{min_parallel_index_scan_size}) \rfloor$

но не больше, чем *max_parallel_workers_per_gather*



Число рабочих процессов выбирается примерно так же, как и в случае последовательного сканирования. Сравнивается объем данных, который предполагается прочитать из индекса (определяемый числом индексных страниц) со значением параметра *min_parallel_index_scan_size* (по умолчанию 512kB).

Если в случае последовательного сканирования таблицы объем данных определялся размером всей таблицы, то при индексном доступе планировщик должен спрогнозировать, сколько индексных страниц будет прочитано. Как именно это происходит, рассматривается позже в теме «Статистика».

Если размер выборки меньше, параллельный план не рассматривается оптимизатором. Например, никогда не будет выполняться параллельно доступ к одному значению — просто нечего распараллеливать.

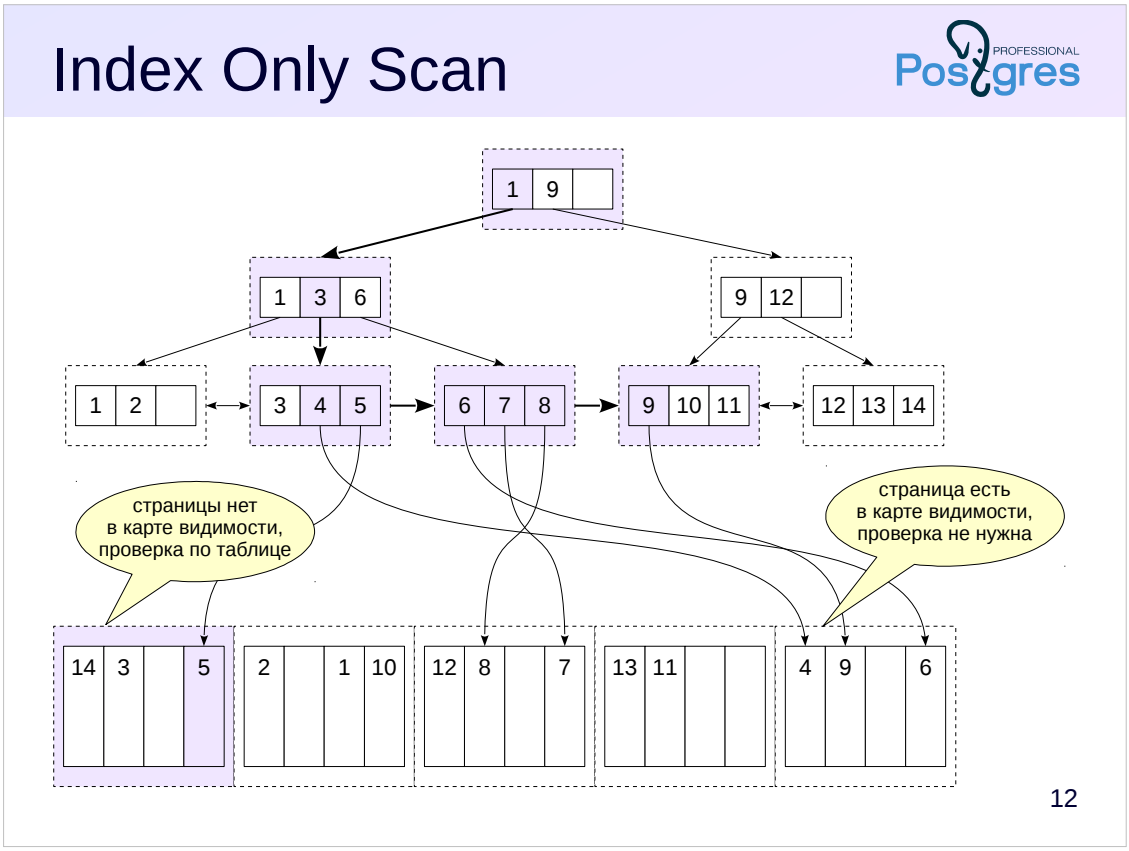
Если размер выборки достаточно велик, число рабочих процессов определяется по формуле, если только оно не указано явно в параметре хранения *parallel_workers* таблицы (не индекса!).

Число процессов в любом случае не будет превышать значения параметра *max_parallel_workers_per_gather*.

# Только индекс

Сканирование только индекса (Index Only Scan)
и карта видимости

Параллельное сканирование (Parallel Index Only Scan)




Если в запросе требуются только проиндексированные данные, то они уже есть в самом индексе — к таблице в этом случае обращаться не надо. Такой индекс называется *покрывающим* для запроса.

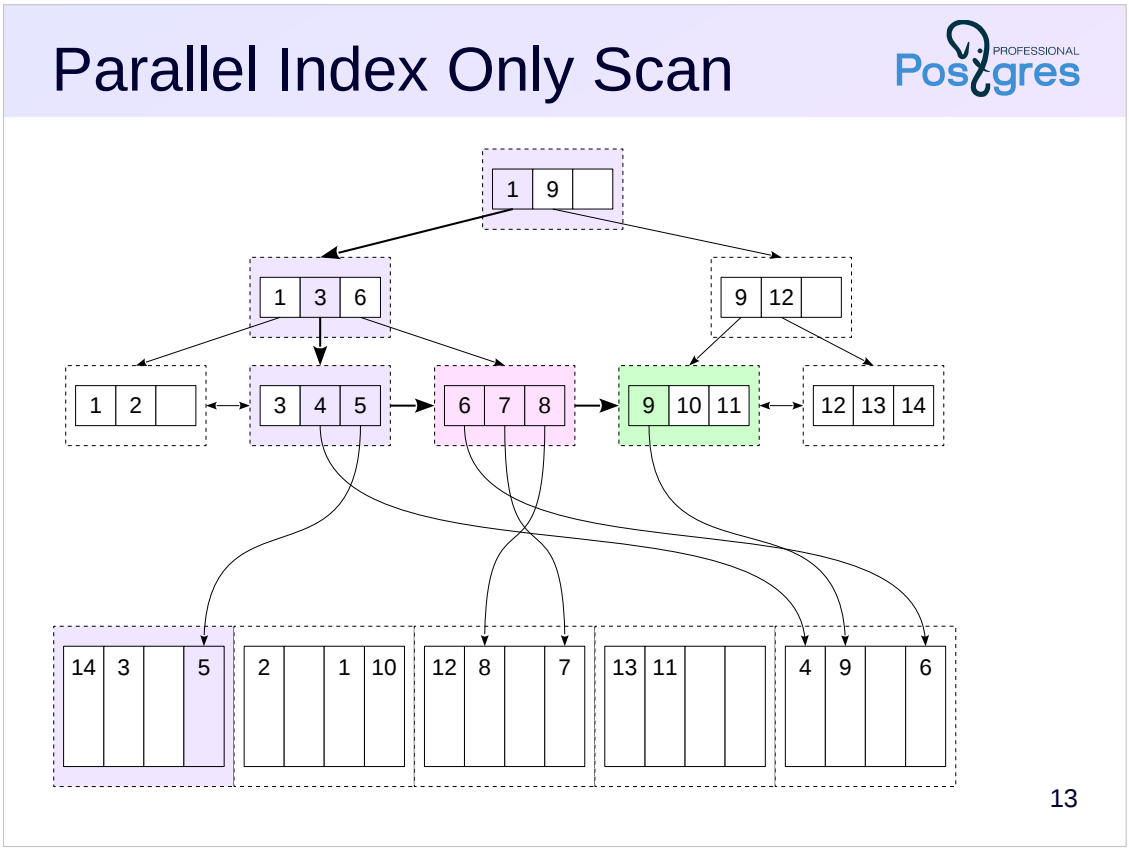
Это хорошая оптимизация, исключающая обращения к табличным страницам. Но, к сожалению, индексные страницы не содержат информацию о видимости строк — чтобы проверить, надо ли показывать найденную в индексе строку, мы вынуждены заглянуть и в табличную страницу, что сводит оптимизацию на нет.

Поэтому критическую роль в эффективности исключительно индексного сканирования играет карта видимости. Если табличная страница содержит гарантированно видимые данные, и это отражено в карте видимости, то к такой табличной странице не надо обращаться. Но страницы, не отмеченные в карте видимости, посетить все-таки придется.

Это одна из причин, по которой стоит выполнять очистку (vacuum) достаточно часто — именно этот процесс обновляет карту видимости.

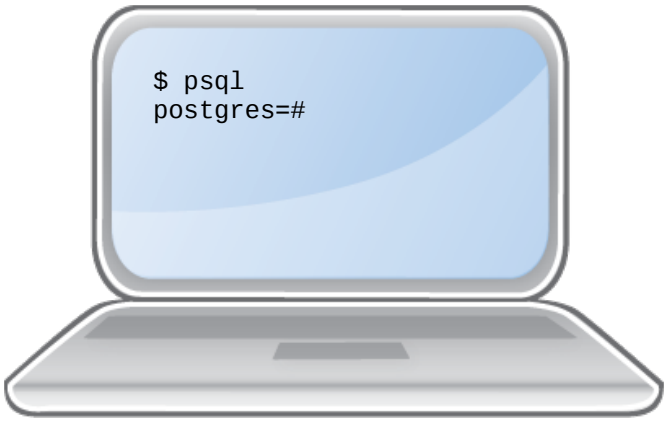
Планировщик не знает, сколько табличных страниц потребуют проверки, но учитывает приблизительную оценку этого числа. При плохом прогнозе планировщик может отказаться от использования исключительно индексного сканирования.

Карта видимости непосредственно просматривается уже при выполнении запроса, и для каждой строки принимается решение, надо ли заглядывать в таблицу или нет.




Исключительно индексное сканирование может выполняться параллельно. Это происходит точно так же, как и при обычном индексном сканировании: ведущий процесс спускается от корня к листовой страницы, а затем рабочие процессы параллельно сканируют листовые страницы индекса.

# Демонстрация



# Итоги

Индексы

- ускоряют доступ при высокой селективности
- поддерживают ограничения целостности
- позволяют получить отсортированные данные

Покрывающие индексы

- позволяют экономить на обращении к таблице

Возможен параллельный индексный доступ

# Практика

1. Напишите запрос, выбирающий максимальную сумму бронирования.
   Проверьте план выполнения. Какой метод доступа выбрал планировщик? Эффективен ли такой доступ?
2. Создайте индекс по столбцу bookings.total_amount.
   Снова проверьте план выполнения запроса. Какой метод доступа выбрал планировщик теперь?
3. При создании индекса можно указать порядок сортировки столбца. Зачем, если индекс можно просматривать в любом направлении?



1. Этот запрос можно написать как минимум двумя разными способами. Во-первых, можно воспользоваться предложениями ORDER BY и LIMIT 1. Во-вторых, можно вывести максимальное число с помощью агрегатной функции max.

Попробуйте оба варианта.

3. Подсказка: это важно для индексов, построенных по нескольким столбцам.