



Статистика



Оценка кардинальности и селективности

Оценка стоимости

Выбор наилучшего плана

Виды собираемой статистики

Использование статистики для оценки кардинальности

Перебор планов оптимизатором

выбирается план с наименьшей стоимостью

Оценка стоимости

в первую очередь зависит от типа узла и числа строк

Оценка кардинальности

нужны данные о размере таблиц и распределении данных

Статистика

собирается и обновляется ANALYZE, в том числе автоочисткой

Рекурсивная процедура

Кардинальность метода доступа

размер исходной таблицы

селективность предикатов

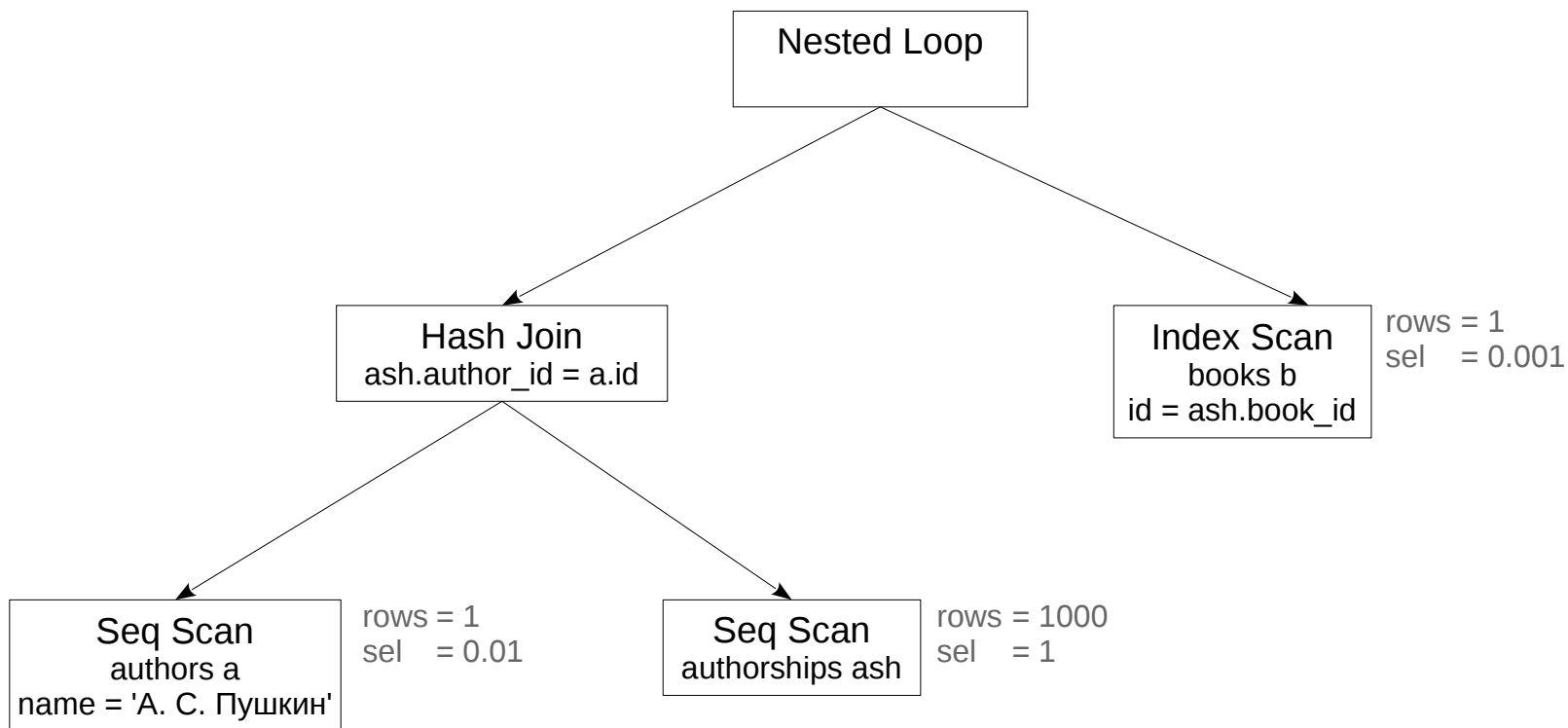
Кардинальность соединения

размеры соединяемых наборов строк

селективность соединения

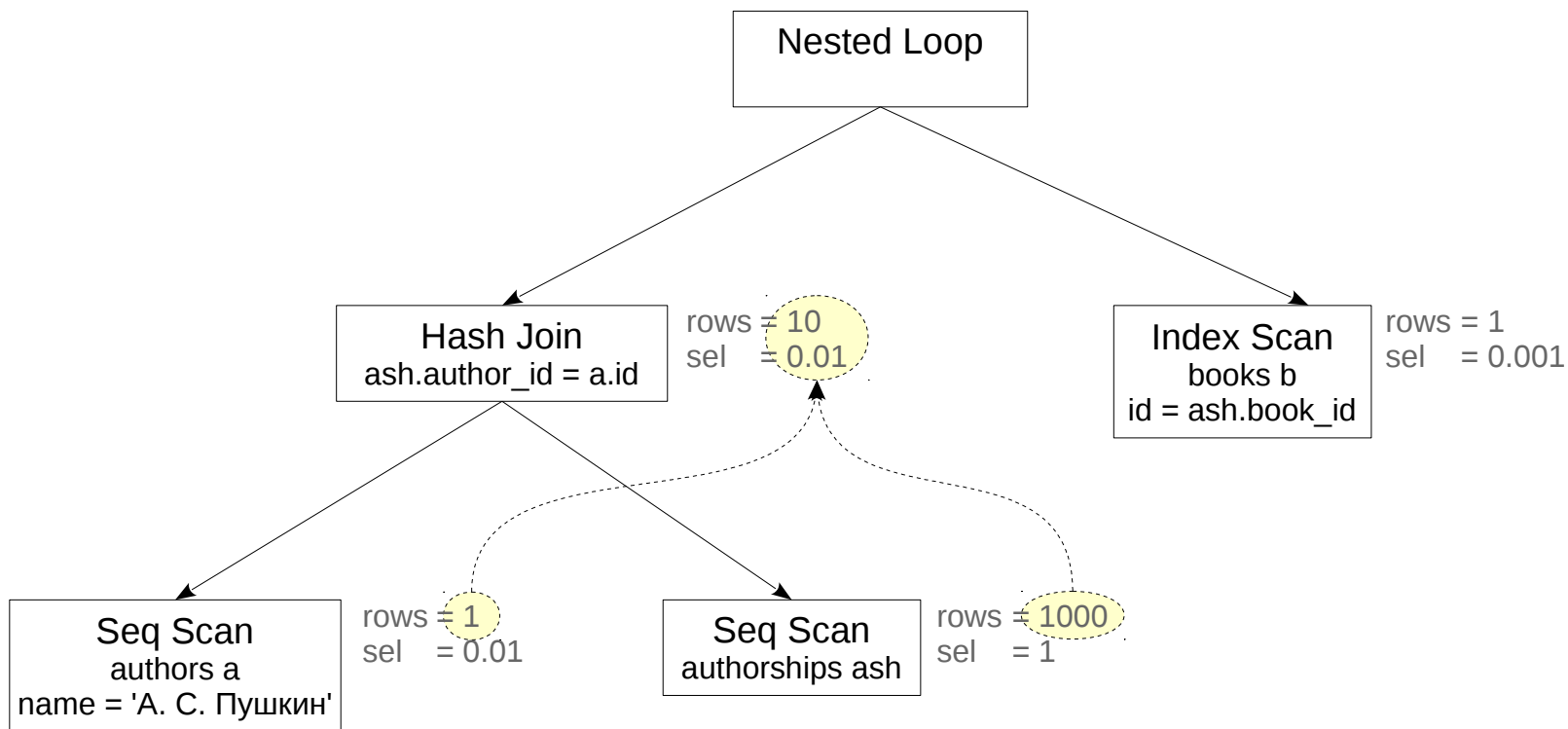
Расчет кардинальности

```
select b.*  
from books b  
join authorships ash on (ash.book_id = b.id)  
join authors a on (a.id = ash.author_id)  
where a.name = 'А. С. Пушкин';
```



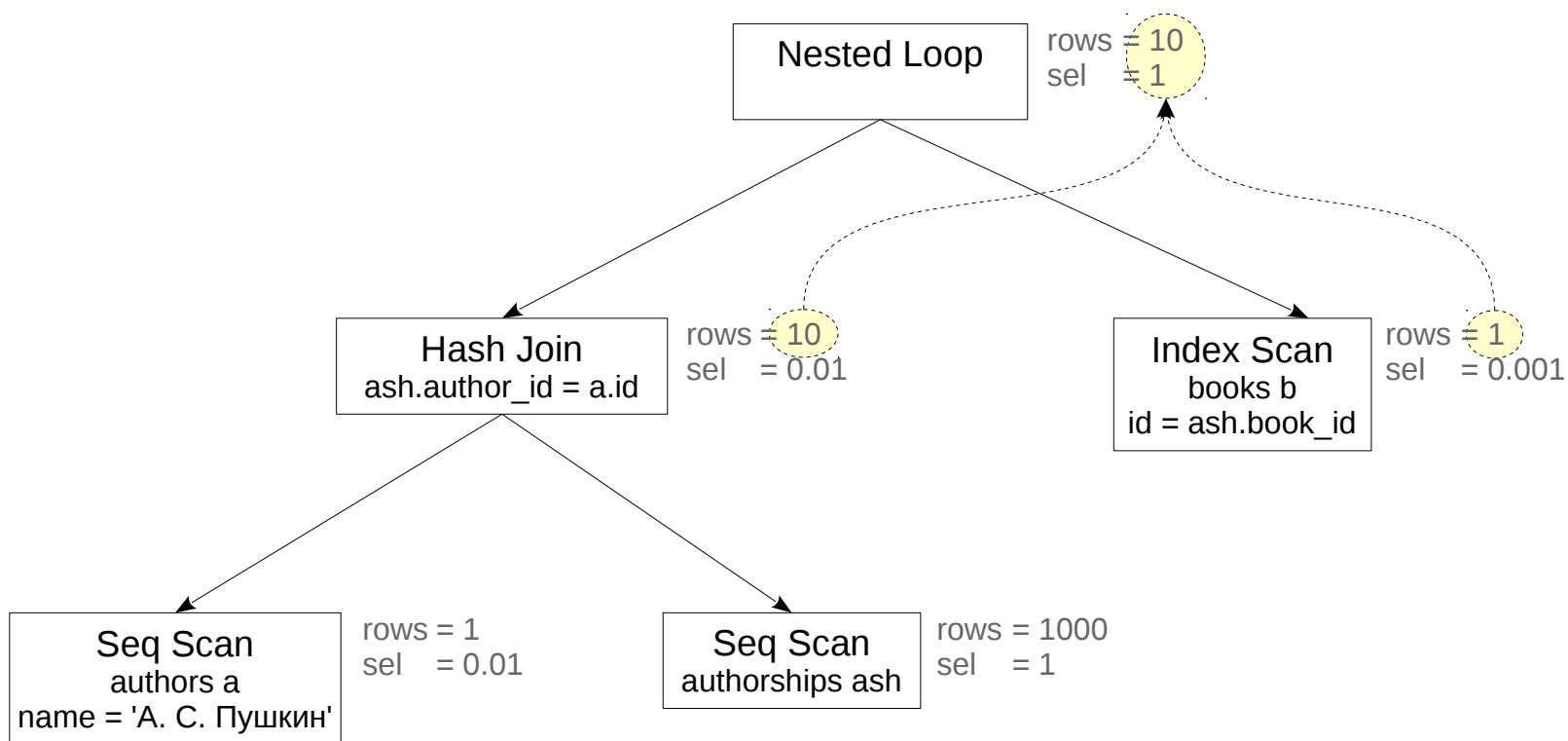
Расчет кардинальности

```
select b.*  
from books b  
join authorships ash on (ash.book_id = b.id)  
join authors a on (a.id = ash.author_id)  
where a.name = 'А. С. Пушкин';
```



Расчет кардинальности

```
select b.*  
from books b  
join authorships ash on (ash.book_id = b.id)  
join authors a on (a.id = ash.author_id)  
where a.name = 'А. С. Пушкин';
```



Рекурсивная процедура

Стоимость поддерева

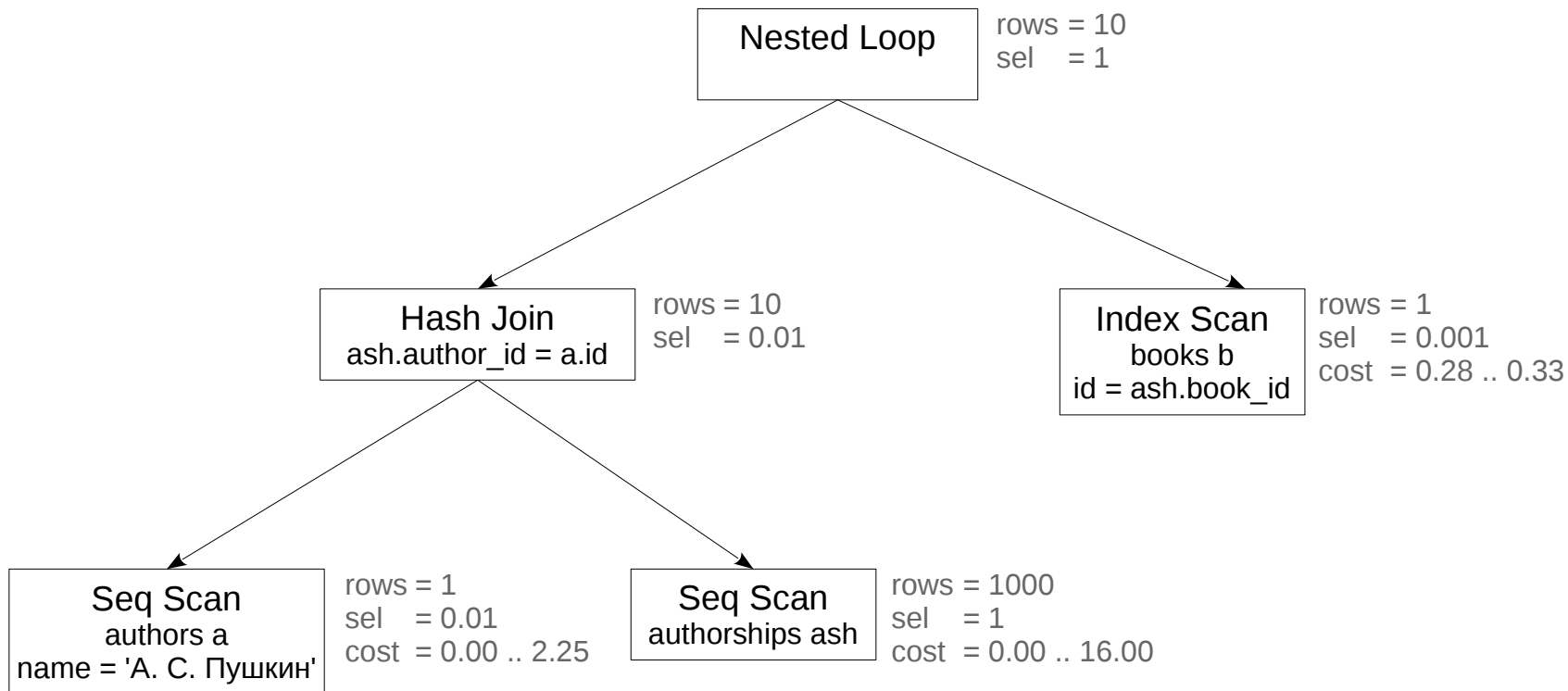
сумма стоимостей дочерних узлов
и стоимости корневого узла поддерева

Стоимость узла

зависит от типа узла и числа обрабатываемых строк (кардинальности)
в меньшей степени зависит от других факторов

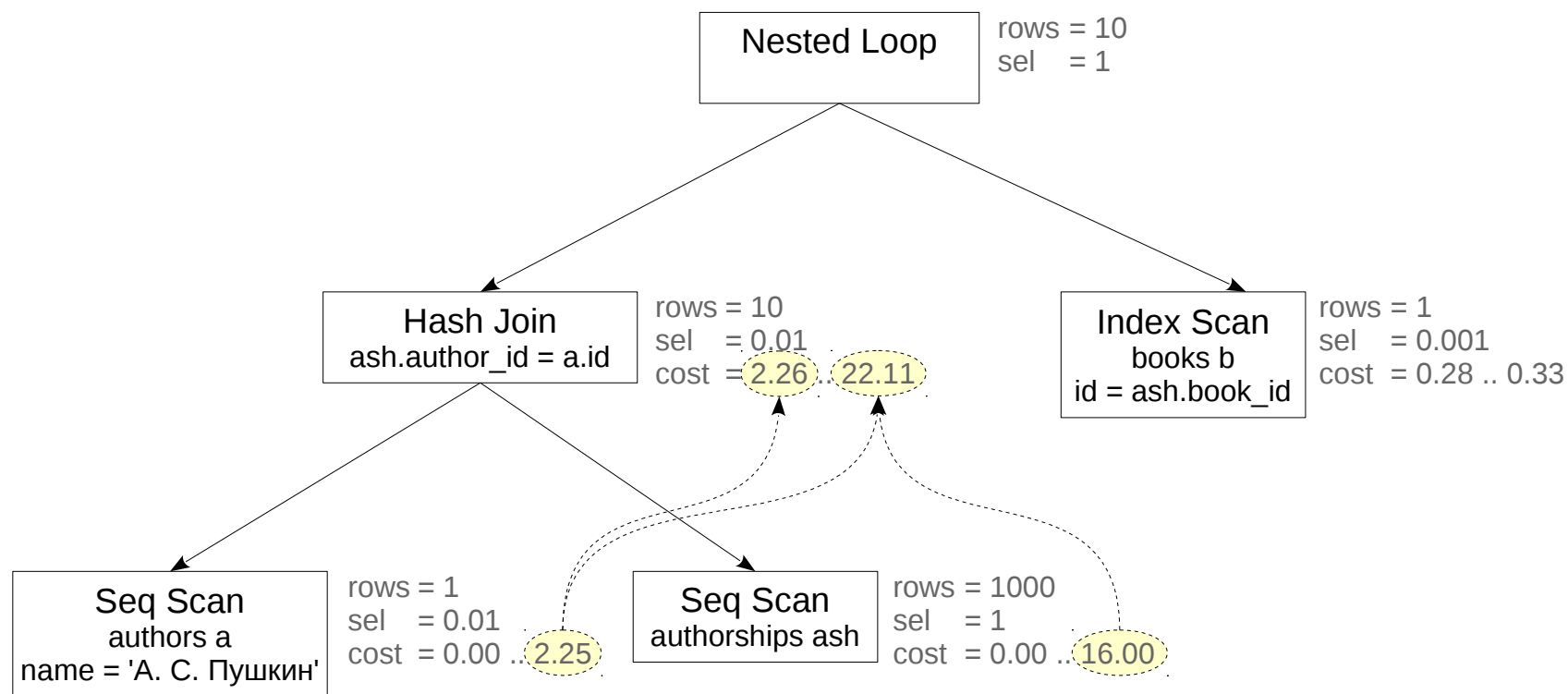
Оценка стоимости

```
select b.*  
from books b  
join authorships ash on (ash.book_id = b.id)  
join authors a on (a.id = ash.author_id)  
where a.name = 'А. С. Пушкин';
```



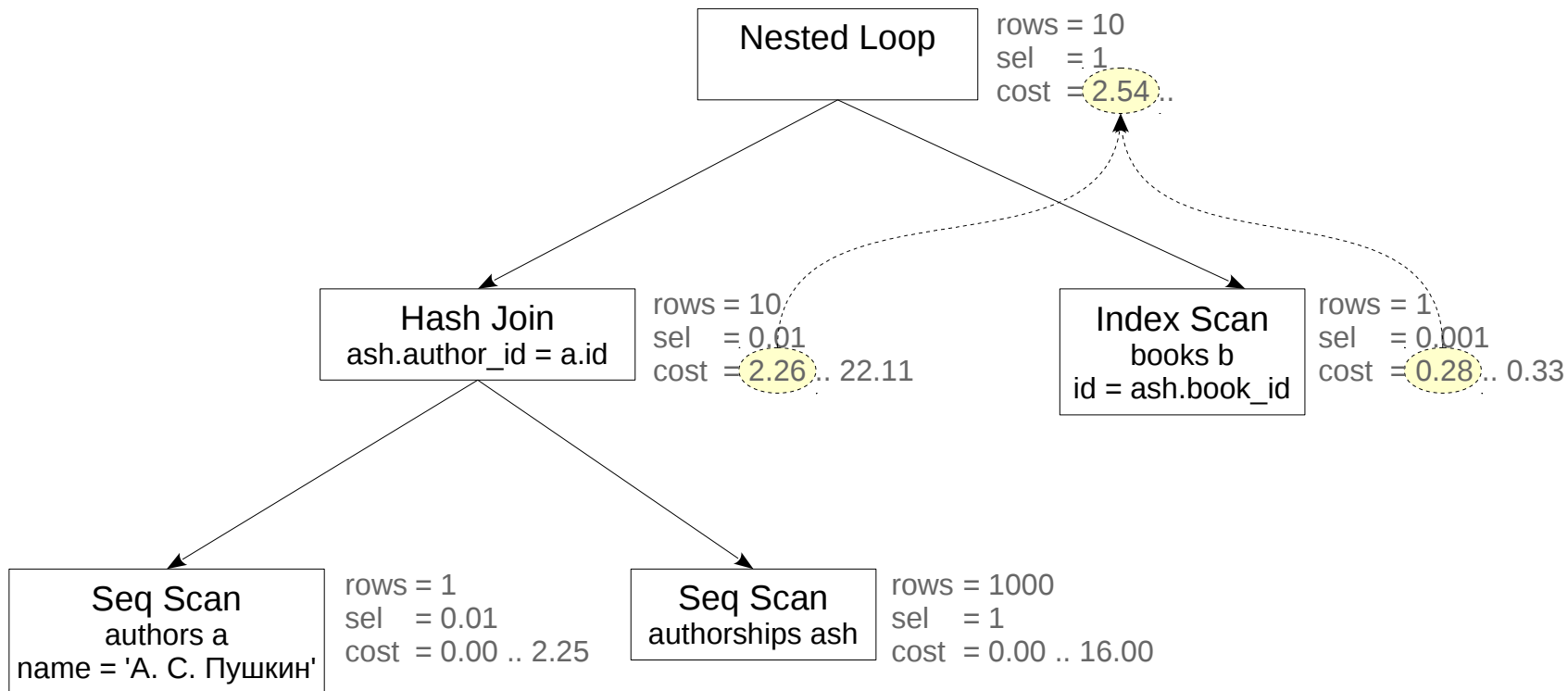
Оценка стоимости

```
select b.*
from   books b
join   authorships ash on (ash.book_id = b.id)
join   authors a on (a.id = ash.author_id)
where  a.name = 'А. С. Пушкин';
```



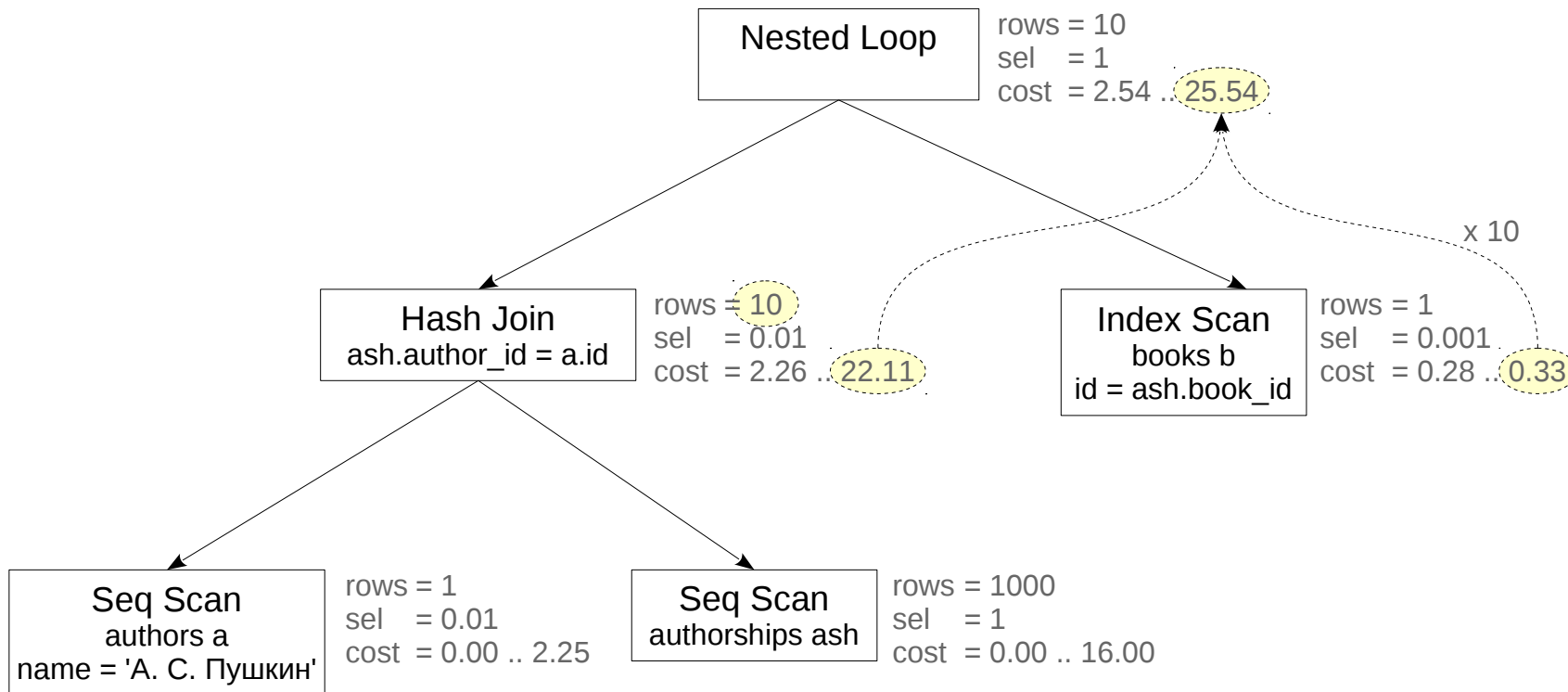
Оценка стоимости

```
select b.*
from   books b
join   authorships ash on (ash.book_id = b.id)
join   authors a on (a.id = ash.author_id)
where  a.name = 'А. С. Пушкин';
```



Оценка стоимости

```
select b.*  
from books b  
join authorships ash on (ash.book_id = b.id)  
join authors a on (a.id = ash.author_id)  
where a.name = 'А. С. Пушкин';
```



ВВОД-ВЫВОД

seq_page_cost = 1.0

random_page_cost = 4.0

CREATE TABLESPACE ... WITH (*параметр = значение*)

Время процессора

cpu_tuple_cost = 0.01

cpu_index_tuple_cost = 0.005

cpu_operator_cost = 0.0025

CREATE FUNCTION ... COST *стоимость*,
(в единицах cpu_operator_cost)

Перебор планов

по возможности полный

при большом числе вариантов — генетический алгоритм (GEQO)

Обычные запросы

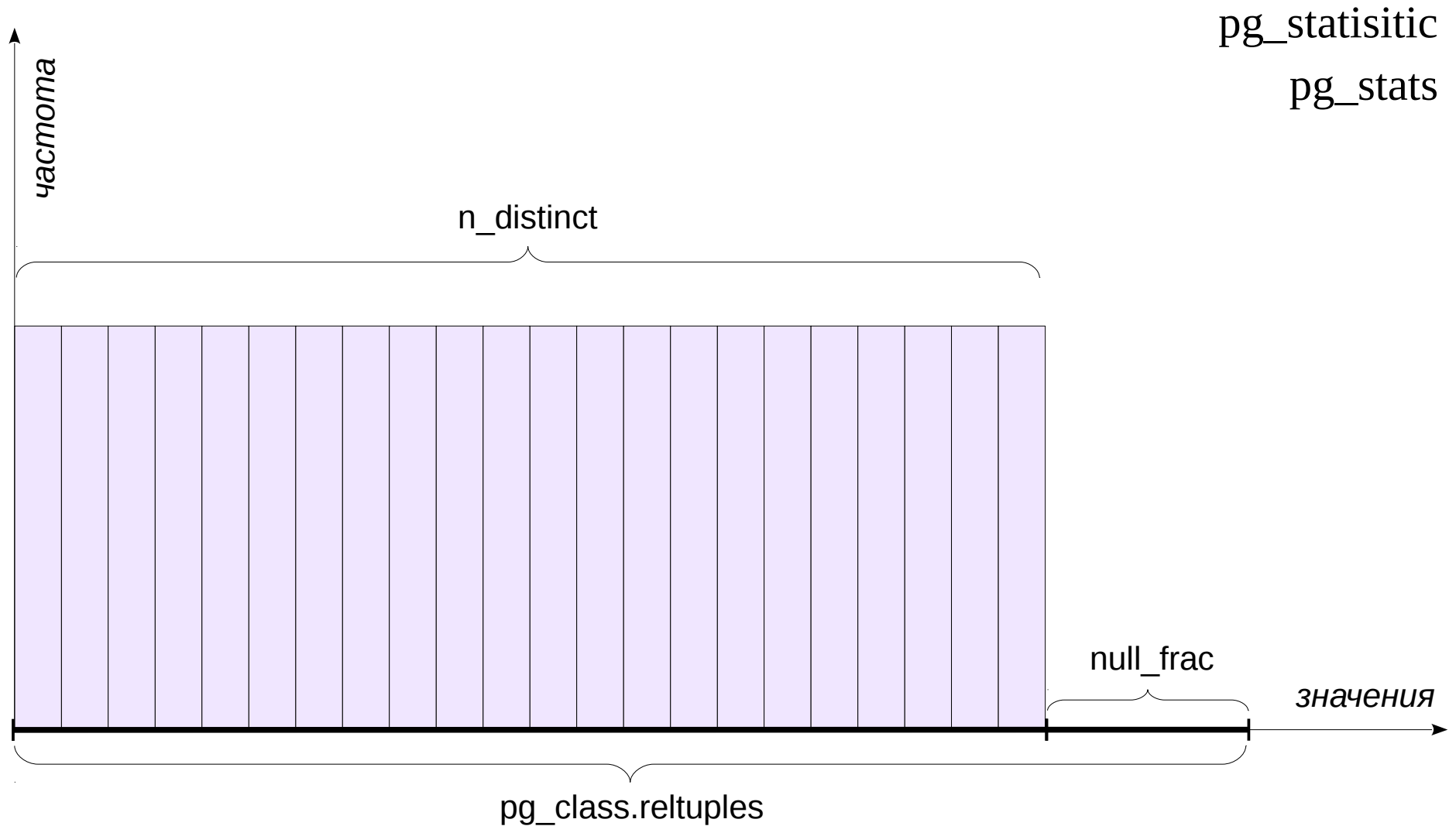
минимальная общая стоимость

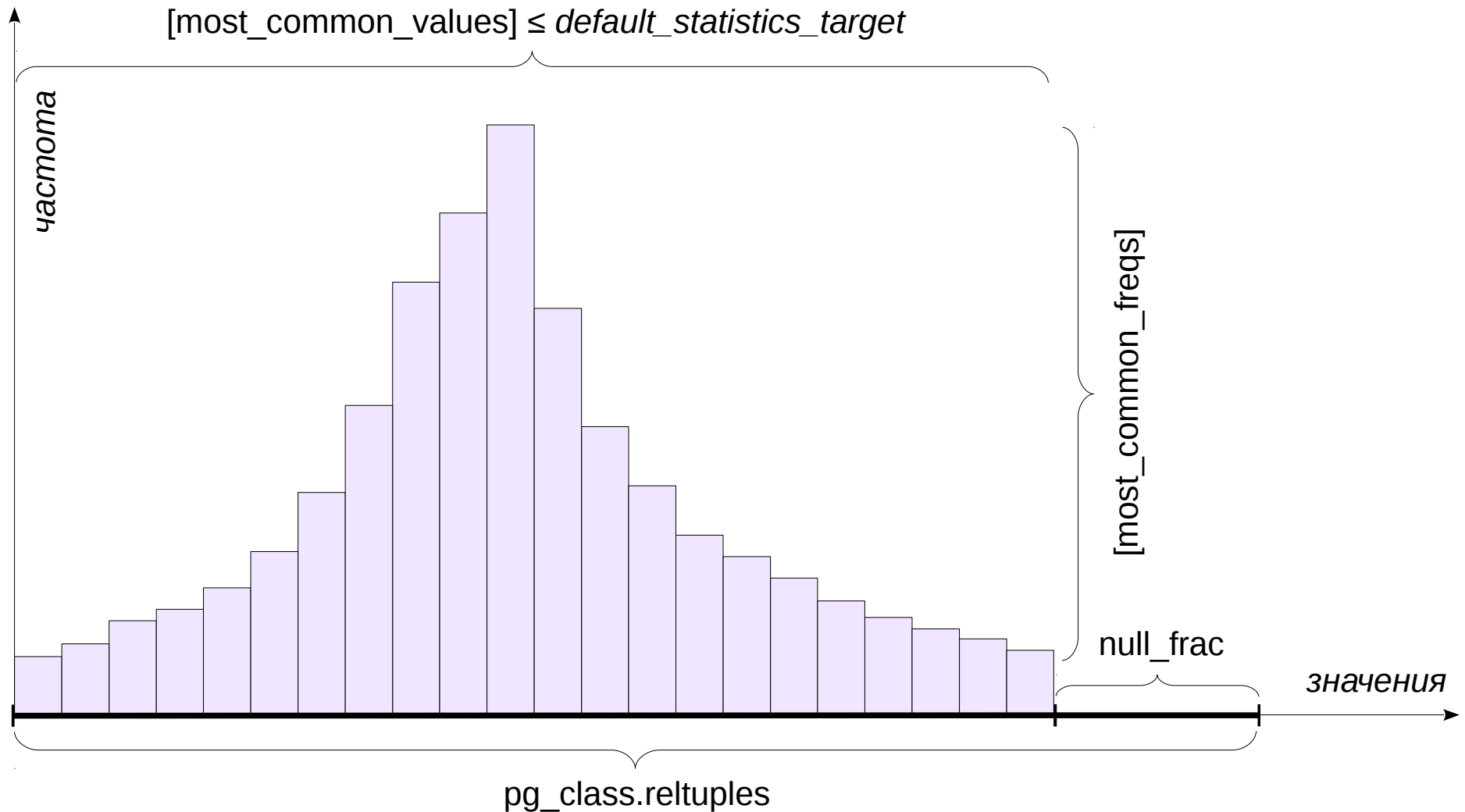
оптимизируется время получения всех строк

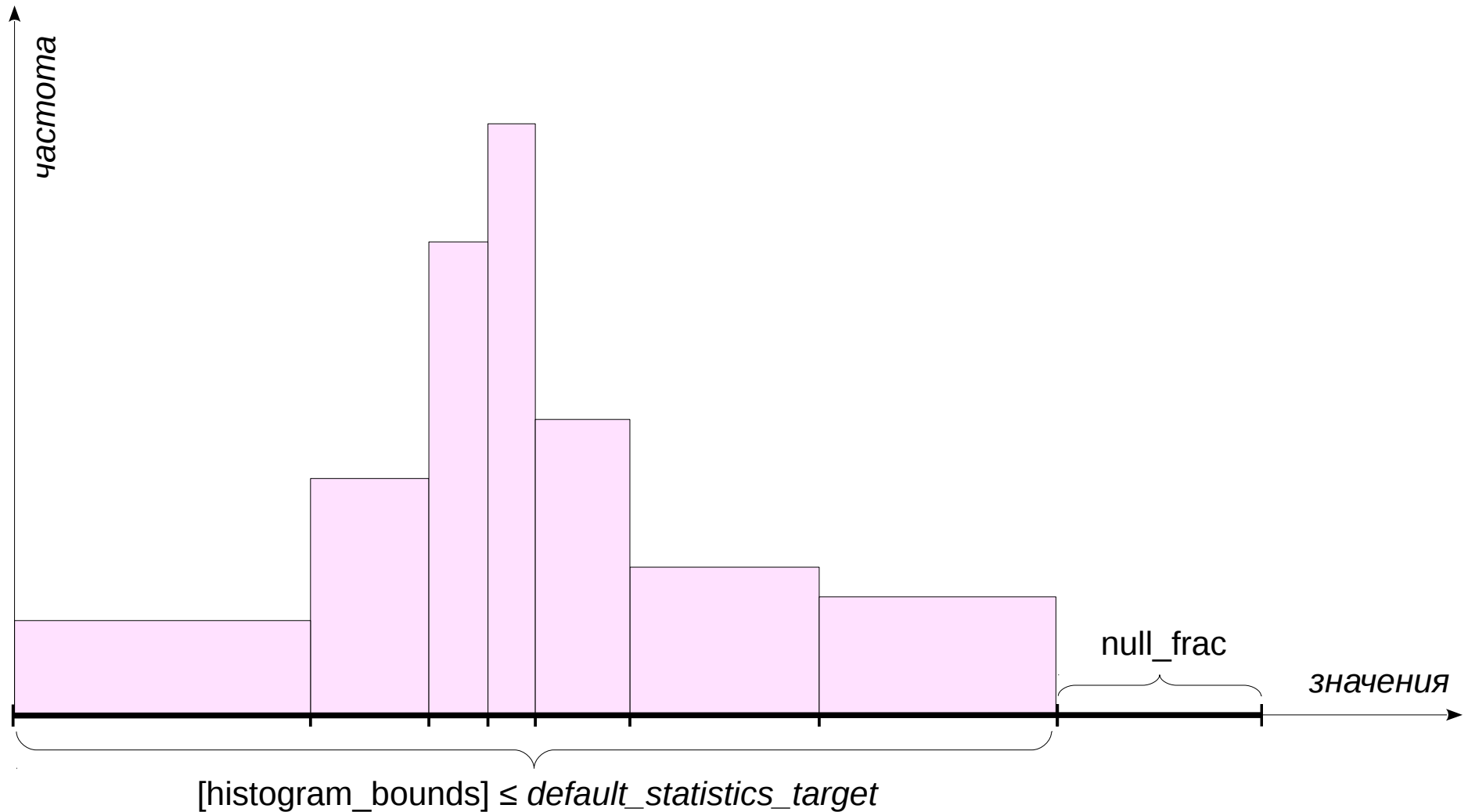
Курсоры

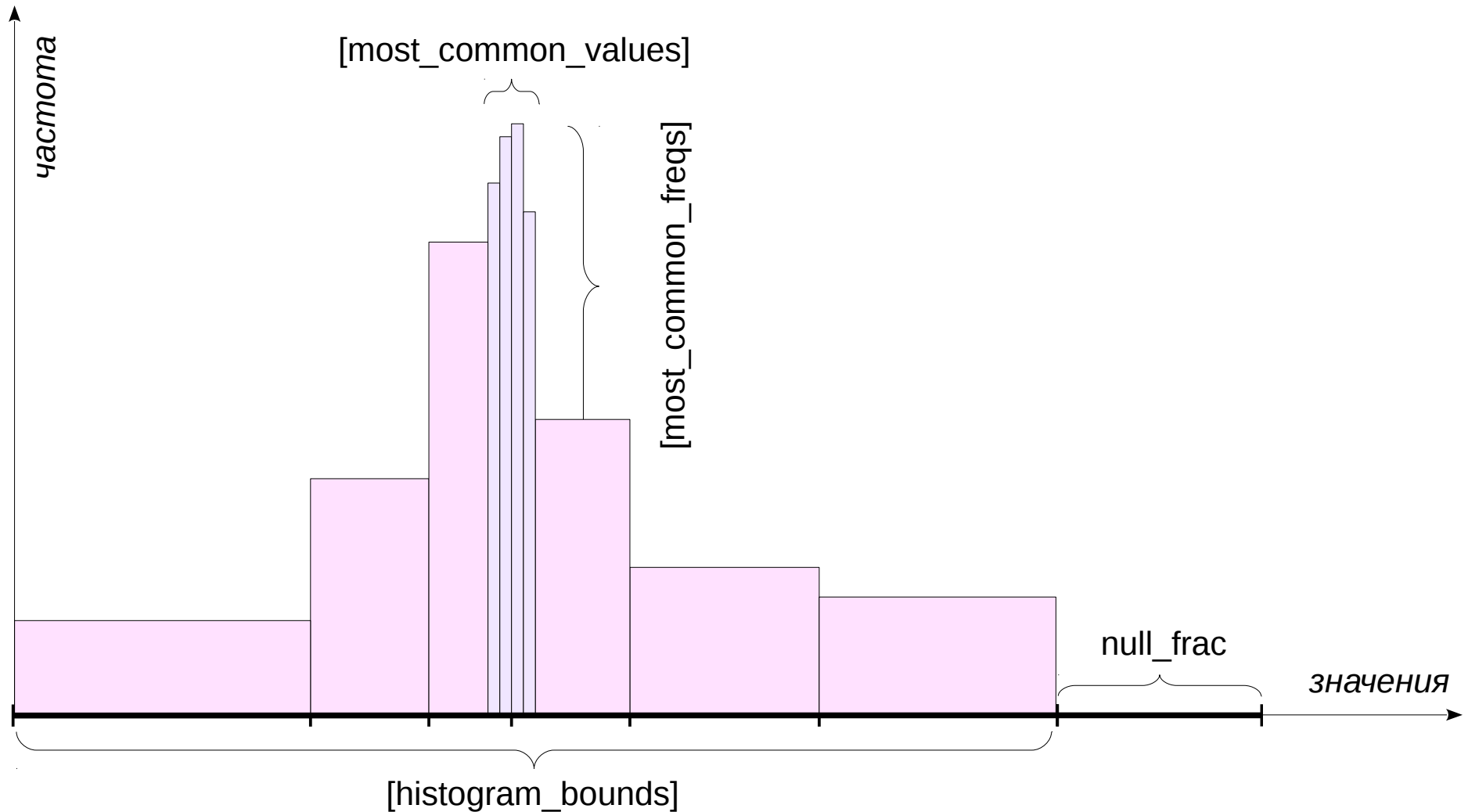
минимальная стоимость для получения *cursor_tuple_fraction* строк

минимизируется время получения части первых строк









Дополнительные поля

avg_width

средний размер строки в байтах

correlation

упорядоченность значений на диске:

1 — по возрастанию

0 — расположены хаотично

-1 — по убыванию

Настройки

default_statistics_target = 100

ALTER TABLE ...

ALTER COLUMN ...

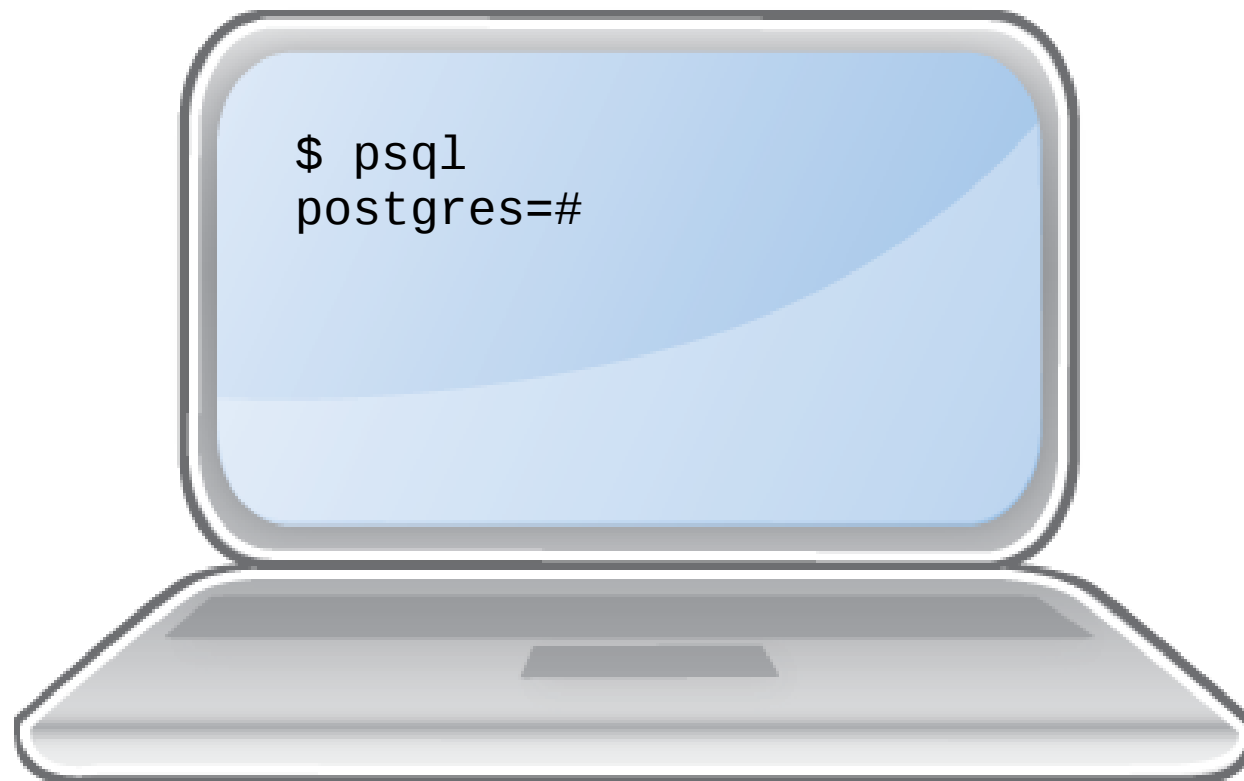
SET STATISTICS

число значений в списке

число корзин в гистограмме

размер выборки при анализе

Демонстрация



Характеристики данных собираются в виде статистики

Статистика нужна для оценки кардинальности

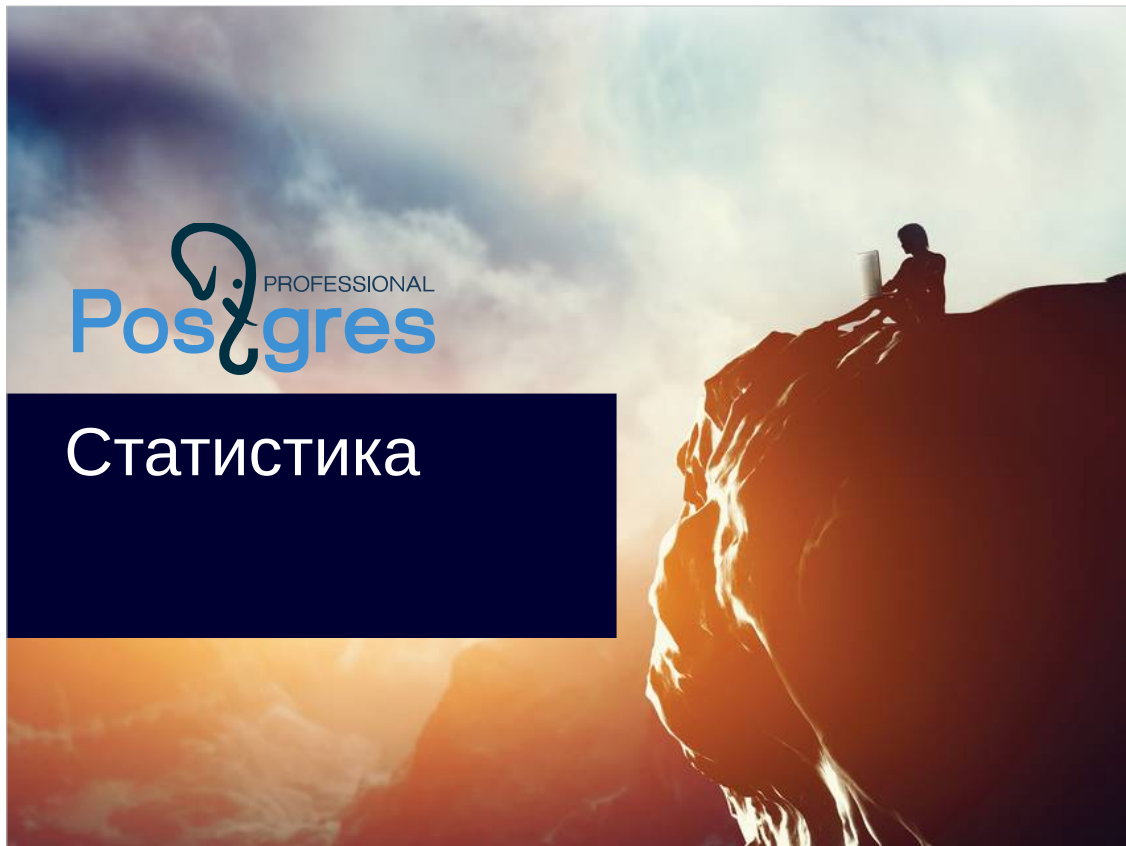
Кардинальность используется для оценки стоимости

Стоимость позволяет выбрать оптимальный план

Залог успеха —

адекватная статистика и корректная кардинальность

1. Создайте базу DB17 и в ней таблицу с двумя полями: идентификатор и логический флаг; отключите автоочистку для этой таблицы.
2. Заполните таблицу: 10 000 строк, флаг установлен для 1% строк.
3. Посмотрите, какая статистика имеется для таблицы и столбцов.
4. Посмотрите планы запросов и объясните результат:
 - а) все строки,
 - б) строки, где флаг установлен,
 - в) строка с указанным идентификатором.
5. Включите анализ таблицы, проверьте наличие статистики.
6. Посмотрите планы тех же запросов. Как изменилась точность оценок? Объясните, как получены оценки кардинальности.



Авторские права

Курс «Администрирование PostgreSQL 9.5. Расширенный курс»

© Postgres Professional, 2016 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 косвенным, 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.

Оценка кардинальности и селективности

Оценка стоимости

Выбор наилучшего плана

Виды собираемой статистики

Использование статистики для оценки кардинальности

Перебор планов оптимизатором

выбирается план с наименьшей стоимостью

Оценка стоимости

в первую очередь зависит от типа узла и числа строк

Оценка кардинальности

нужны данные о размере таблиц и распределении данных

Статистика

собирается и обновляется ANALYZE, в том числе автоочисткой

Мы уже говорили о том, что оптимизатор перебирает всевозможные планы выполнения, оценивает их и выбирает план с наименьшей стоимостью.

Чтобы оценить стоимость каждого узла, в первую очередь надо знать объем обрабатываемых данных, ну и, конечно, тип самого узла. Например, стоимость индексного сканирования напрямую зависит от селективности условия доступа.

Чтобы оценить селективность и кардинальность, надо, в свою очередь, иметь сведения о данных: размер таблиц, распределение данных по столбцам.

Таким образом, в итоге все упирается в *статистику* — информацию, собираемую и обновляемую процессом analyze (который обычно работает в составе autovacuum, но может быть запущен и вручную).

Зная кардинальность, стоимость рассчитывается относительно несложно и обычно довольно точно. Основные же ошибки оптимизатора связаны с неправильной оценкой кардинальности. Это может происходить из-за неадекватной статистики (настройка автоочистки!) или несовершенства моделей, лежащих в основе оптимизатора. Этот вопрос мы будем подробно рассматривать ввиду его важности.

А сначала посмотрим на общий процесс расчета кардинальности и стоимости и выбор лучшего плана.

Рекурсивная процедура

Кардинальность метода доступа

- размер исходной таблицы
- селективность предикатов

Кардинальность соединения

- размеры соединяемых наборов строк
- селективность соединения

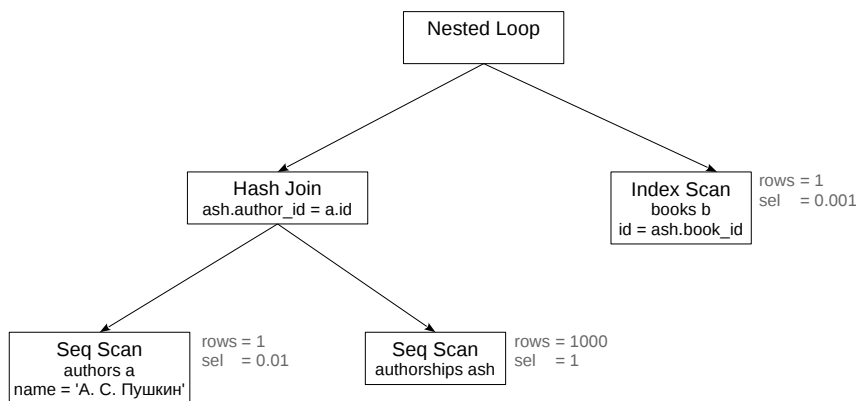
Оценку кардинальности удобно рассматривать как рекурсивный процесс. Чтобы оценить кардинальность узла, надо сначала оценить кардинальности дочерних узлов, а затем — зная тип узла — вычислить на их основе кардинальность самого узла.

Таким образом, сначала можно рассчитать кардинальности листовых узлов, в которых находятся методы доступа к данным. Для этого нам нужно знать размер таблицы и селективность условий, наложенных на нее. Как конкретно это делается, мы рассмотрим позже, а пока будем просто считать, что как-то возможно.

Затем можно рассчитать кардинальности соединений. Кардинальности соединяемых наборов данных нам уже известны, осталось оценить селективность условия соединения. Пока тоже будем считать, что это как-то возможно.

Аналогично можно поступить и с другими узлами, например, с сортировками или агрегациями.

```
select b.*
from   books b
join   authorships ash on (ash.book_id = b.id)
join   authors a on (a.id = ash.author_id)
where  a.name = 'А. С. Пушкин';
```



5

Посмотрим на примере. У нас есть три таблицы: *книги* (books) и *авторы* (authors) связаны отношением многие-ко-многим через таблицу *авторства* (authorships). В запросе мы ищем все книги, которые написал Пушкин.

Сначала оценим кардинальность и селективность листовых узлов — методов доступа.

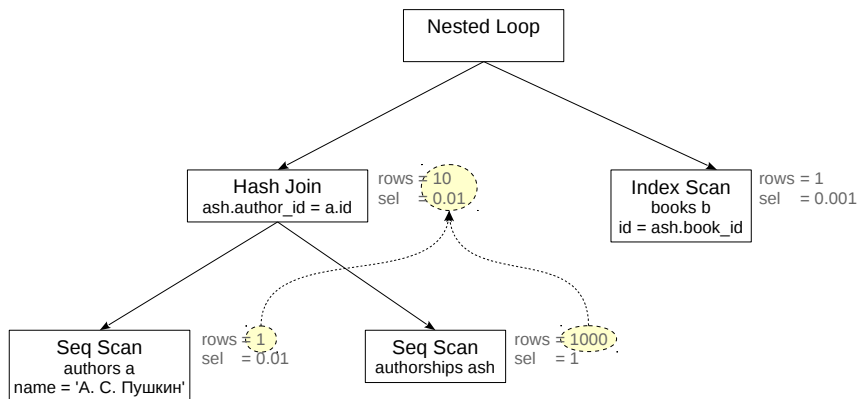
В таблице authors 100 строк, и мы (каким-то образом) оценили селективность условия name = 'А. С. Пушкин' как 0.01. Таким образом, мы ожидаем получить одну строку.

В таблице authorships 1000 строк, дополнительных условий нет, поэтому наша оценка селективности — 1, и мы ожидаем получить все 1000 строк.

В таблице books 1000 строк, селективность условия (каким-то образом) мы оценили как 0.001, и ожидаем получить одну строку.

Расчет кардинальности

```
select b.*
from books b
join authorships ash on (ash.book_id = b.id)
join authors a on (a.id = ash.author_id)
where a.name = 'А. С. Пушкин';
```

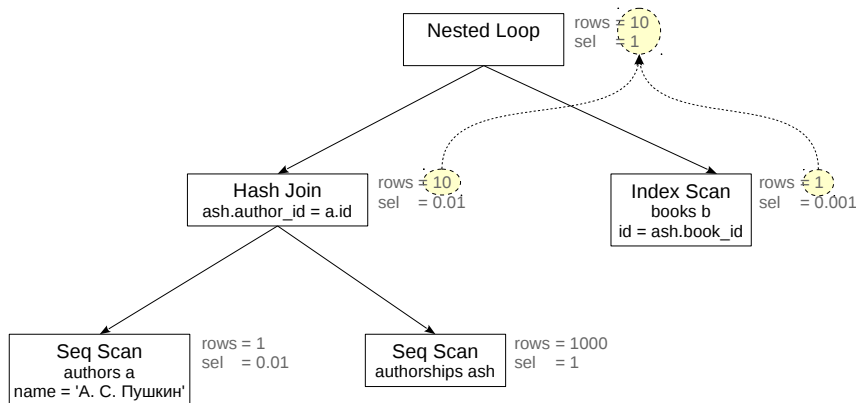


6

Теперь мы можем приступить к оценке соединения authors и authorships. Селективность мы оцениваем (каким-то образом) как 0.01, и после соединения ожидаем получить 10 строк.

Расчет кардинальности

```
select b.*
from books b
join authorships ash on (ash.book_id = b.id)
join authors a on (a.id = ash.author_id)
where a.name = 'А. С. Пушкин';
```



7

Наконец, оценка селективности для соединения результата предыдущего соединения с таблицей books оценивается как 1 (нет никаких условий соединения), и мы ожидаем получить 10 строк.

Тут условие соединения для Nested Loop фактически «переехало» в условие доступа к таблице books.

Важно отметить, что ошибка расчета кардинальности в нижних узлах будет распространяться выше, приводя в итоге к неверной оценке и выбору неудачного плана.

Рекурсивная процедура

Стоимость поддерева

сумма стоимостей дочерних узлов
и стоимости корневого узла поддерева

Стоимость узла

зависит от типа узла и числа обрабатываемых строк (кардинальности)
в меньшей степени зависит от других факторов

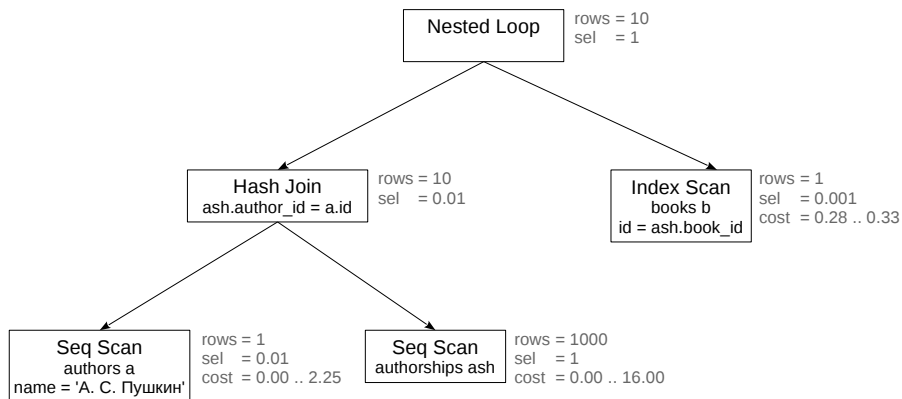
Теперь рассмотрим общий процесс оценки стоимости. Схема та же, что и при оценке кардинальности: чтобы рассчитать стоимость поддерева, сначала надо рассчитать стоимости дочерних узлов, сложить их, и добавить стоимость самого узла.

На самом деле, «сумма» стоимостей дочерних узлов не всегда будет просто суммой. Например, для соединения вложенными циклами стоимость правого поддерева надо еще умножить на число итераций.

Стоимость работы самого узла в первую очередь зависит от числа обрабатываемых строк — которое уже рассчитано.

Оценка стоимости

```
select b.*
from books b
join authorships ash on (ash.book_id = b.id)
join authors a on (a.id = ash.author_id)
where a.name = 'А. С. Пушкин';
```



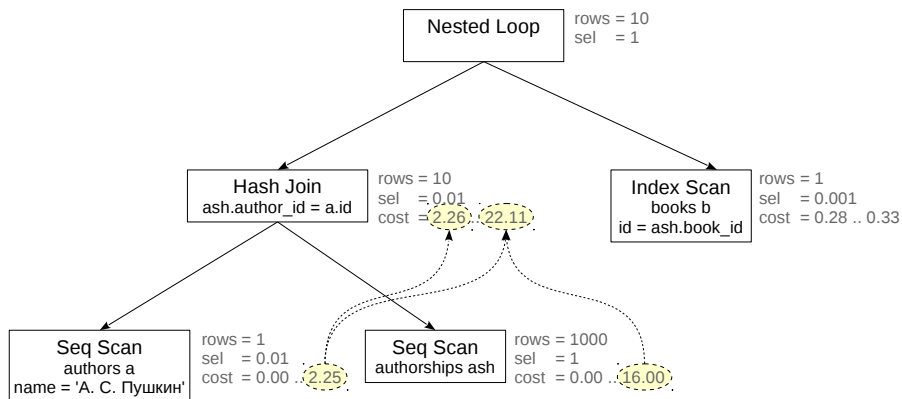
В нашем примере рассчитаем сначала стоимости узлов, соответствующих методам доступа.

Не так уж важно, как получаются конкретные цифры стоимости. Они выражены в неких «условных единицах» и сами по себе не говорят ни о чем. Стоимость нужна лишь для того, чтобы можно было сравнивать разные планы.

Еще раз имеет смысл обратить внимание, что на самом деле вычисляются две стоимости: стоимость подготовительных работ и общая стоимость. Это важно, если нам надо получить только часть результата: первую компоненту стоимости придется «заплатить» в любом случае, плюс некоторую часть второй компоненты.

Оценка стоимости

```
select b.*
from books b
join authorships ash on (ash.book_id = b.id)
join authors a on (a.id = ash.author_id)
where a.name = 'А. С. Пушкин';
```



10

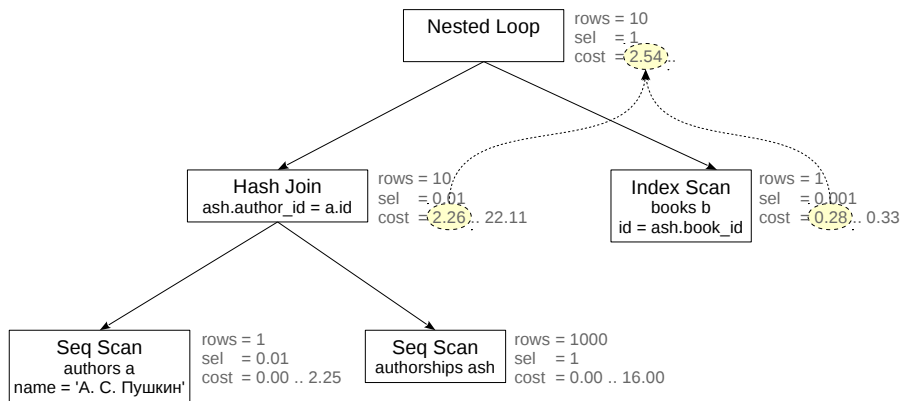
Теперь можно оценить стоимость первого соединения.

Поскольку это соединение хэшированием, для построения хэш-таблицы требуется прочитать таблицу authors полностью. Потому вторая компонента стоимости доступа (2.25) добавляется к стоимости построения таблицы (0.01) и образует первую компоненту стоимости соединения.

Вторая компонента стоимости соединения получается как сумма стоимостей дочерних узлов (2.25+16.00), плюс стоимость самого соединения.

Оценка стоимости

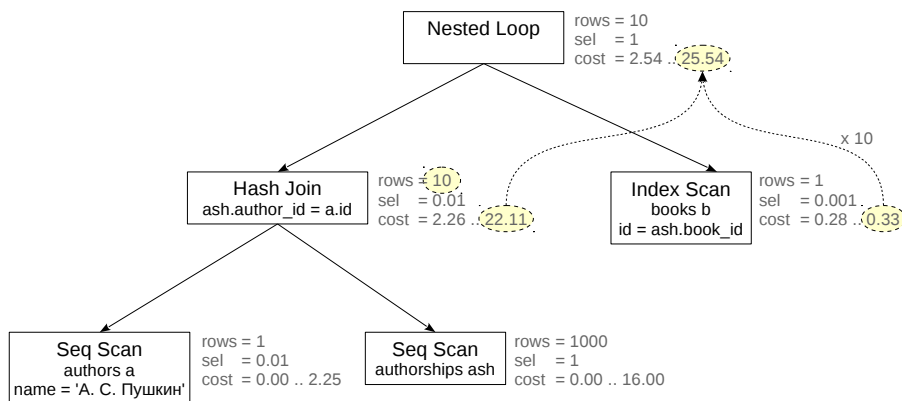
```
select b.*
from books b
join authorships ash on (ash.book_id = b.id)
join authors a on (a.id = ash.author_id)
where a.name = 'А. С. Пушкин';
```



Дальше переходим к оценке второго соединения. Первая компонента получается как сумма первых компонент дочерних узлов.

Оценка стоимости

```
select b.*
from books b
join authorships ash on (ash.book_id = b.id)
join authors a on (a.id = ash.author_id)
where a.name = 'А. С. Пушкин';
```



12

Вторая компонента стоимости получается как сумма вторых компонент дочерних узлов, но стоимость правого узла (соответствующего внутреннему циклу) умножается на число итераций: $22.11 + 10 \cdot 0.33$.

К результату добавляются накладные расходы на собственно соединение.

Ввод-вывод

seq_page_cost = 1.0

random_page_cost = 4.0

CREATE TABLESPACE ... WITH (*параметр = значение*)

Время процессора

cpu_tuple_cost = 0.01

cpu_index_tuple_cost = 0.005

cpu_operator_cost = 0.0025

CREATE FUNCTION ... COST *стоимость*,
(в единицах cpu_operator_cost)

На расчет стоимости влияют несколько параметров, задающих веса.

Во-первых, это параметры `seq_page_cost` и `random_page_cost`, определяющие стоимость чтения одной страницы при последовательном доступе и при произвольном доступе.

Разница в стоимости обусловлена реалиями вращающихся дисков: так как физическое перемещение головки выполняется медленно, то прочитать несколько страниц, идущих подряд, оказывается дешевле, чем то же количество страниц, разбросанных по диску. Для SSD-дисков разница должна быть меньше.

Чтобы можно было учесть свойства разных носителей, параметры можно задавать не только глобально, но и на уровне отдельных табличных пространств.

Во-вторых, параметры для времени процессора. Основное время тратится на ввод-вывод, но тем не менее обработка прочитанных данных тоже требует ресурсов. Можно указать стоимость обработки одной табличной строки (`cpu_tuple_cost`), одной строки индекса (`cpu_index_tuple_cost`) и одной операции (`cpu_operator_cost`; например, операции сравнения).

Можно также задать стоимость пользовательской функции в единицах `cpu_operator_cost`. По умолчанию функции на Си получают оценку 1, а на других языках — 100.

Перебор планов

по возможности полный
при большом числе вариантов — генетический алгоритм (GEQO)

Обычные запросы

минимальная общая стоимость
оптимизируется время получения всех строк

Курсоры

минимальная стоимость для получения *cursor_tuple_fraction* строк
минимизируется время получения части первых строк

Оптимизатор старается перебрать всевозможные планы выполнения запроса, чтобы выбрать из них лучший.

При большом количестве вариантов (в первую очередь на это влияет количество соединяемых таблиц) полный перебор за разумное время становится невозможен. В таком случае PostgreSQL переключается на использование генетического алгоритма (GEQO — Genetic Query Optimization). Это может привести к тому, что оптимизатор выберет не лучший план не из-за ошибок в оценке, а просто потому, что лучший план не рассматривался.

Работой GEQO управляет ряд параметров, которые мы не будем рассматривать подробно.

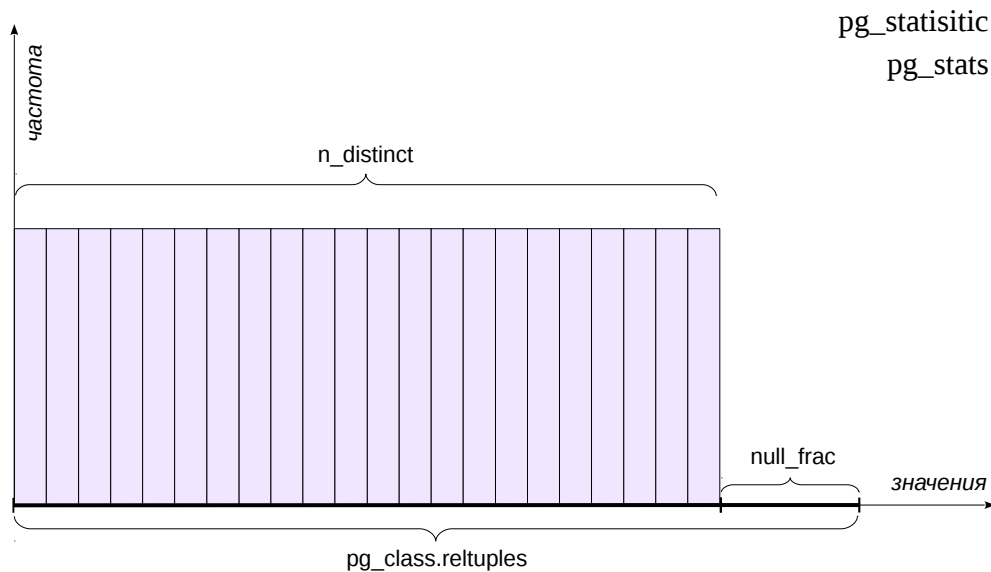
<http://www.postgresql.org/docs/9.5/static/geqo.html>

<http://www.postgresql.org/docs/9.5/static/runtime-config-query.html>

Что же считается «лучшим планом»?

Для обычных запросов это план, минимизирующий время получения всех строк. То есть план с минимальной общей стоимостью (второй компонент).

Однако при использовании курсоров может быть важно как можно быстрее получить первые строки. Поэтому существует параметр *cursor_tuple_fraction* (значение по умолчанию 0.1), задающий долю строк, которую надо получить как можно быстрее. Чем меньше значение этого параметра, тем больше на выбор плана влияет первая компонента стоимости, а не вторая.



Теперь перейдем к вопросу статистики и ее использования для оценки кардинальности и селективности. Какая статистика собирается?

Во-первых, это число строк в таблице (или индексе) и размер объекта в страницах. Эта информация хранится в системном каталоге в таблице `pg_class`, поля `reltuples` и `relpages`.

Информация о размере объекта заполняется некоторыми массовыми операциями (`create index`, `create table as select`), а затем обновляется операциями `vacuum` и `analyze`. Это значение не будет точным, поскольку обычно очистка и анализ просматривают только часть страниц. Оптимизатор масштабирует значение в соответствии с отклонением текущего размера файла данных от `relpages`.

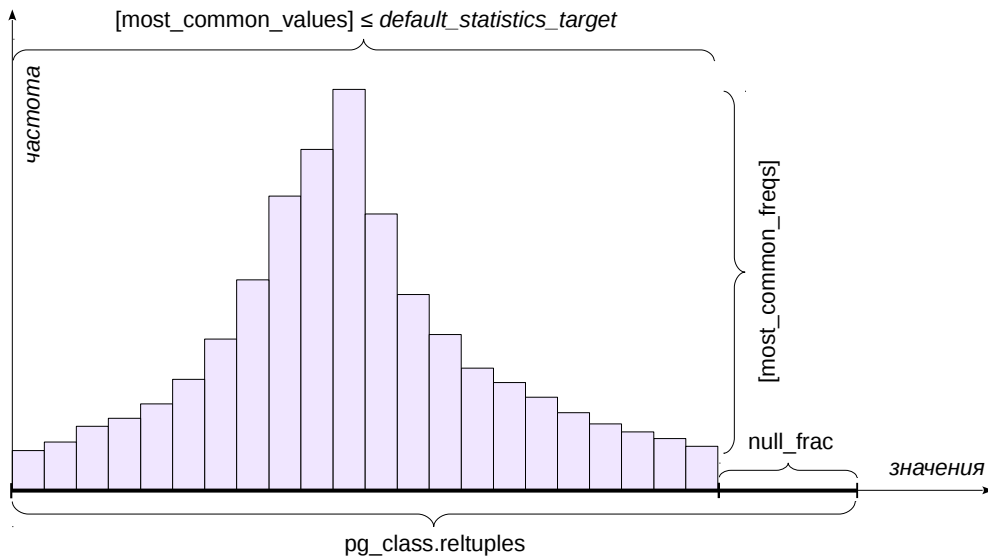
В целом, вся статистика не точна, да и не должна быть точной. Обычно достаточно попадания в порядок, чтобы обеспечить выбор приемлемого плана.

Вся остальная статистика собирается (при анализе) отдельно для каждого столбца таблицы и хранится в таблице `pg_statistic`. Но проще смотреть в представление `pg_stats`, которое показывает информацию в более удобном виде.

Поле `null_frac` содержит долю `null`-значений в таблице (от 0 до 1).

Поле `n_distinct` хранит число уникальных значений в столбце.

Если бы все данные были всегда распределены равномерно, то есть все значения встречались бы с одинаковой частотой, этой информации было бы почти достаточно (нужен еще минимум и максимум).



Но в реальности неравномерные распределения встречаются очень часто. Поэтому собирается следующая информация:

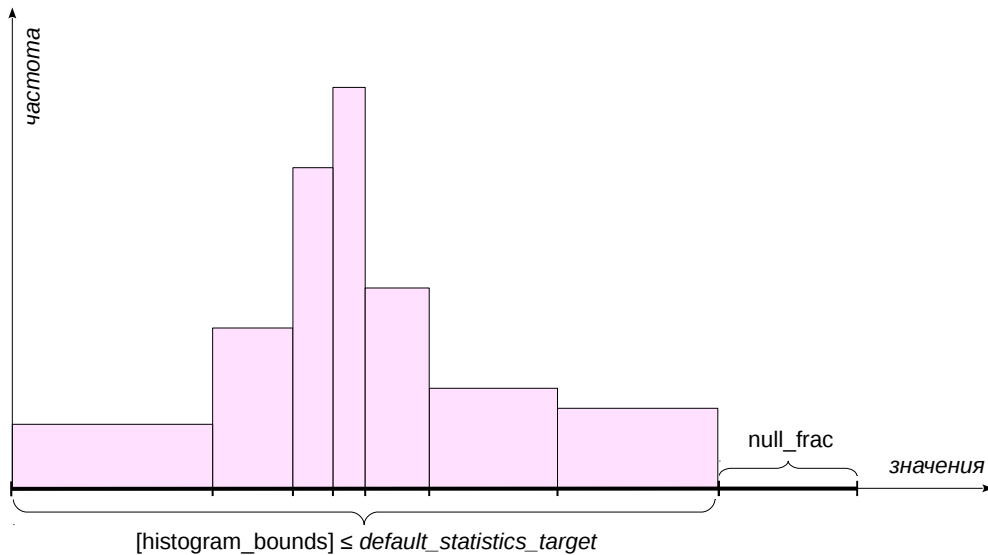
- массив всевозможных значений — поле `most_common_values`,
- массив частот этих значений — поле `most_common_freqs`.

Если нам надо оценить селективность условия *поле = значение*, мы находим *значение* в массиве `most_common_values` и берем частоту из `most_common_freqs` — это и будет селективность.

Если надо оценить селективность условия *поле < значение*, находим в `most_common_values` все числа, меньшие *значения*, и суммируем частоты из `most_common_freqs`.

Все это прекрасно работает, пока число различных значений не очень велико. Максимальный размер задается параметром `default_statistics_target`.

Тонкий момент представляют «большие» значения. Чтобы не увеличивать размер `pg_statistic` и не нагружать планировщик бесполезной работой, значение исключается из статистики и анализа, если его размер превышает 1 КБ. В самом деле, если в поле хранятся такие большие значения, скорее всего они уникальны и не имеют шансов попасть в `most_common_values`.



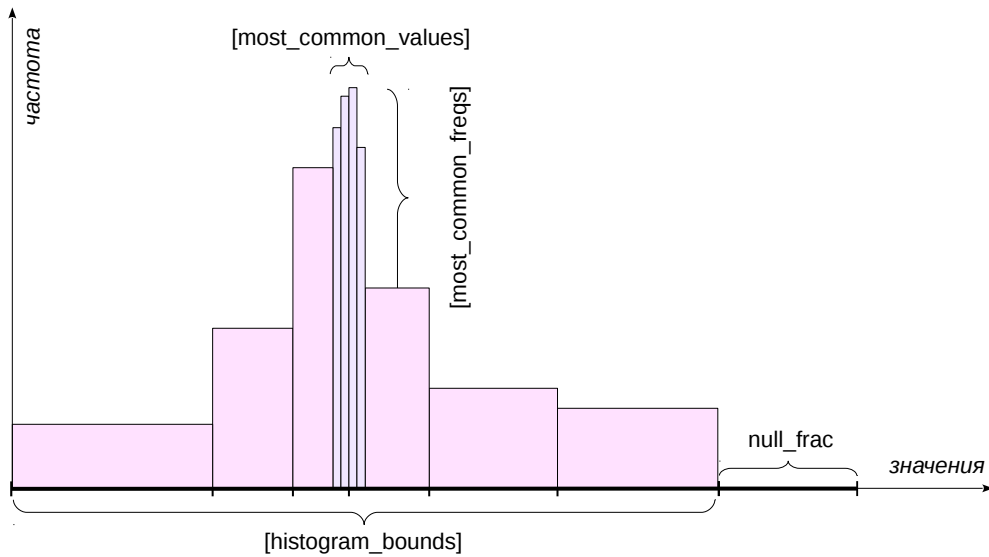
Если число различных значений слишком велико, чтобы записать их в массив, на помощь приходит гистограмма. Гистограмма состоит из нескольких корзин, в которые помещаются значения. Количество корзин ограничено тем же параметром `default_statistics_target`.

Ширина корзин выбирается так, чтобы в каждую попало примерно одинаковое число значений (на рисунке это выражается в одинаковой площади прямоугольников).

При таком построении достаточно хранить только массив крайних значений каждой корзины — поле `histogram_bounds`. Частота одной корзины равна $1/(\text{число корзин})$.

Оценить селективность условия `поле < значение` можно как $N/(\text{общее число корзин})$, где N — число корзин, лежащих слева от `значения`. Оценку можно улучшить, добавив часть корзины, в которую попадает само `значение`.

Если же надо оценить селективность условия `поле = значение`, то гистограмма в этом не может помочь, и приходится довольствоваться предположением о равномерном распределении и брать в качестве оценки $1/n_distinct$.



Но обычно два подхода объединяются: строится список не всех, но наиболее частых значений, а остальные значения покрываются гистограммой.

При этом гистограмма строится так, что в ней не учитываются значения, попавшие в список. Это позволяет улучшить оценки.

Дополнительные поля

avg_width	средний размер строки в байтах
correlation	упорядоченность значений на диске: 1 — по возрастанию 0 — расположены хаотично -1 — по убыванию

Настройки

default_statistics_target = 100	число значений в списке число корзин в гистограмме размер выборки при анализе
ALTER TABLE ...	
ALTER COLUMN ...	
SET STATISTICS	

Еще пара значений статистики. В поле avg_width сохраняется средний размер строки в байтах для расчета необходимого объема памяти.

В поле correlation записывается показатель упорядоченности значений на диске. Если значения хранятся строго по возрастанию, значение будет близко к единице; если по убыванию — к минус единице. Чем более хаотично расположены данные на диске, тем ближе значение к нулю. Именно это поле использует оптимизатор, когда выбираем между сканированием битовой карты и обычным индексным сканированием.

Настройка точности собираемой статистики всего одна — это параметр default_statistics_target (по умолчанию 100). Он определяет и максимальный размер списка most_common_values, и максимальное число корзин в гистограмме. Значение этого параметра можно задавать индивидуально для каждого столбца, чтобы учесть особенности распределения данных. Уменьшение параметра (вплоть до 0) ускоряет работу оптимизатора; увеличение — повышает точность.

Этот же параметр определяет размер случайной выборки, которая используется при анализе. Размер выборки, обеспечивающий хорошую точность оценок, практически не зависит от размера таблицы, поэтому вычисляется максимум из значения глобального параметра и значений для каждого столбца (но не меньше 100) и анализируется 300 раз по столько случайных строк.

<http://www.postgresql.org/docs/9.5/static/planner-stats.html>

<http://www.postgresql.org/docs/9.5/static/row-estimation-examples.html>



Характеристики данных собираются в виде статистики

Статистика нужна для оценки кардинальности

Кардинальность используется для оценки стоимости

Стоимость позволяет выбрать оптимальный план

Залог успеха —

адекватная статистика и корректная кардинальность

1. Создайте базу DB17 и в ней таблицу с двумя полями: идентификатор и логический флаг; отключите автоочистку для этой таблицы.
2. Заполните таблицу: 10 000 строк, флаг установлен для 1% строк.
3. Посмотрите, какая статистика имеется для таблицы и столбцов.
4. Посмотрите планы запросов и объясните результат:
 - а) все строки,
 - б) строки, где флаг установлен,
 - в) строка с указанным идентификатором.
5. Включите анализ таблицы, проверьте наличие статистики.
6. Посмотрите планы тех же запросов. Как изменилась точность оценок? Объясните, как получены оценки кардинальности.

Вставка строк будет работать быстрее, если индексы создать позже. Это относится и к первичным ключам.