Авторы: Егор Рогов, Павел Лузанов

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

Темы

Соединение хешированием

Использование оперативной памяти и временных файлов

Группировка с помощью хеширования

Соединение хешированием

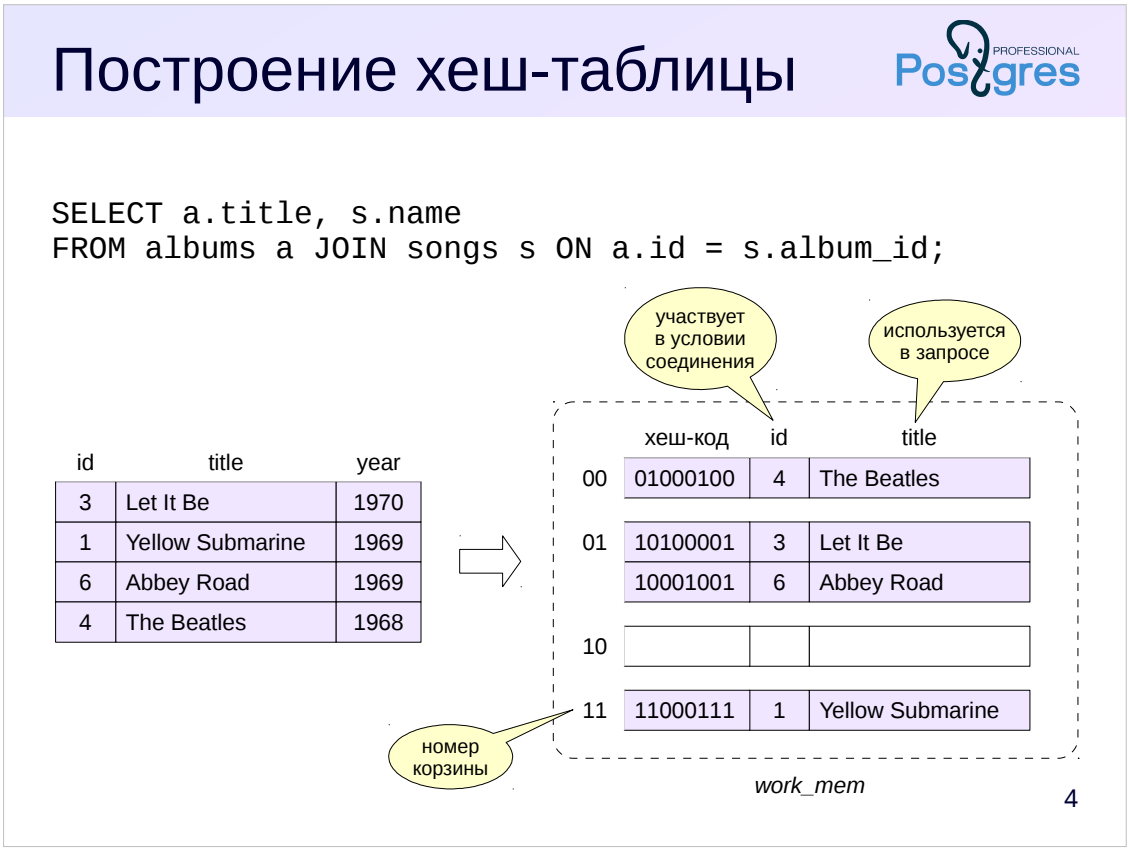
Построение хеш-таблицы

Выполнение соединения

Вычислительная сложность

Параллельное соединение

Построение хеш-таблицы



Первым этапом в памяти строится хеш-таблица.

Идея хеширования состоит в том, что функция хеширования *равномерно* распределяет значения по ограниченному числу корзин хеш-таблицы. В таком случае разные значения как правило будут попадать в разные корзины. Если равномерности не будет, в одну корзину может попасть много значений. В таком случае они выстраиваются в список, и по мере увеличения длины списка эффективность поиска по хеш-таблице будет падать.

Итак, строки первого набора читаются последовательно, и для каждой из них вычисляется хеш-функция от значения полей, входящих в условие соединения (в нашем примере — числовые идентификаторы).

По значению хеш-функции определяется номер корзины. Например, если используется 4 корзины, то в качестве номера корзины можно взять два младших бита.

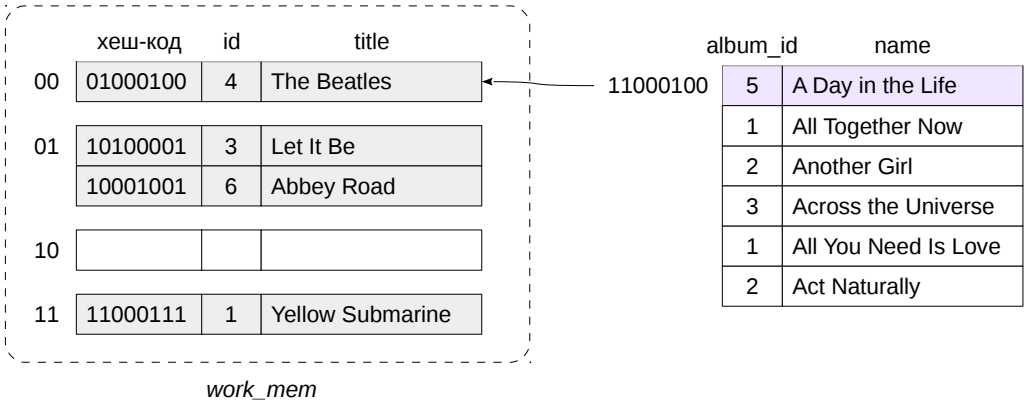
В корзину хеш-таблицы помещаются вычисленный хеш-код и все поля, которые входят в условие соединения или используются в запросе.

Размер хеш-таблицы в памяти ограничен значением параметра *work_mem*. Наилучшая эффективность достигается, если вся хеш-таблица помещается в этот объем памяти целиком. (Это еще одна причина не использовать в запросе лишние поля, в т. ч. «звездочку».)

Исходный код алгоритма можно найти в файле src/backend/executor/nodeHashjoin.c.

Выполнение соединения

```
SELECT a.title, s.name
FROM albums a JOIN songs s ON a.id = s.album_id;
```



На втором этапе мы последовательно читаем второй набор строк. По мере чтения мы вычисляем хеш-функцию от значения полей, участвующих в условии соединения. Если в соответствующей корзине хеш-таблицы обнаруживается строка

- с таким же хеш-кодом,

- и со значениями полей, подходящими под условие соединения,

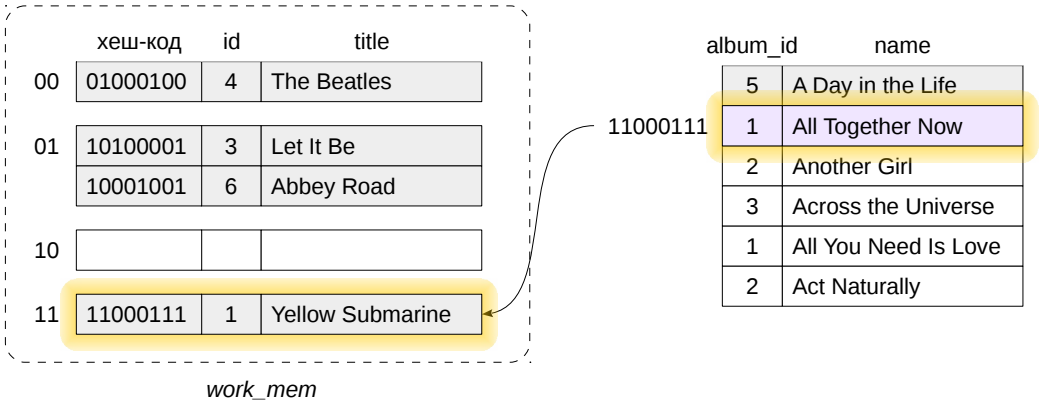
то мы нашли пару.

Проверки одного только хеш-кода недостаточно. Во-первых, не все условия соединения, перечисленные в запросе, могут быть учтены при выполнении соединения хешированием (поддерживаются только эквисоединения). Во-вторых, возможны коллизии, при которых разные значения получат одинаковые хеш-коды (вероятность этого мала, но тем не менее она есть).

В нашем примере для первой строки соответствия нет.

Выполнение соединения

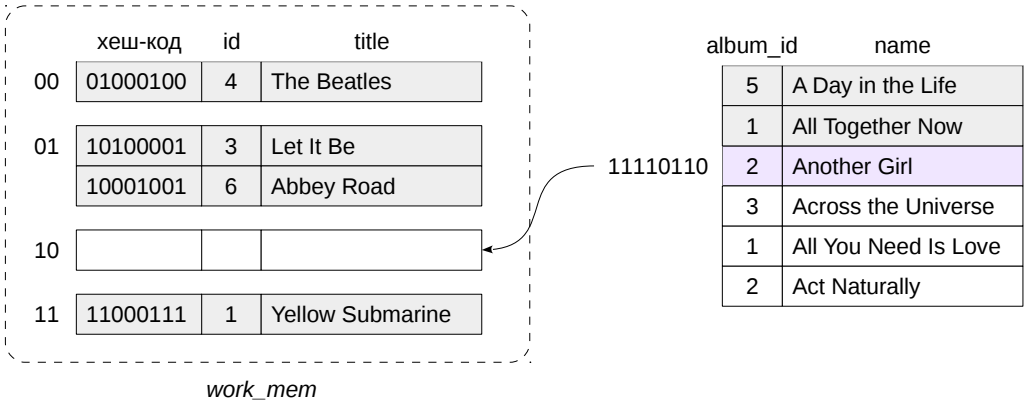
```
SELECT a.title, s.name
FROM albums a JOIN songs s ON a.id = s.album_id;
```



Вторая строка второго набора дает соответствие, которое уже можно вернуть вышестоящему узлу плана: («Yellow Submarine», «All Together Now»).

Выполнение соединения

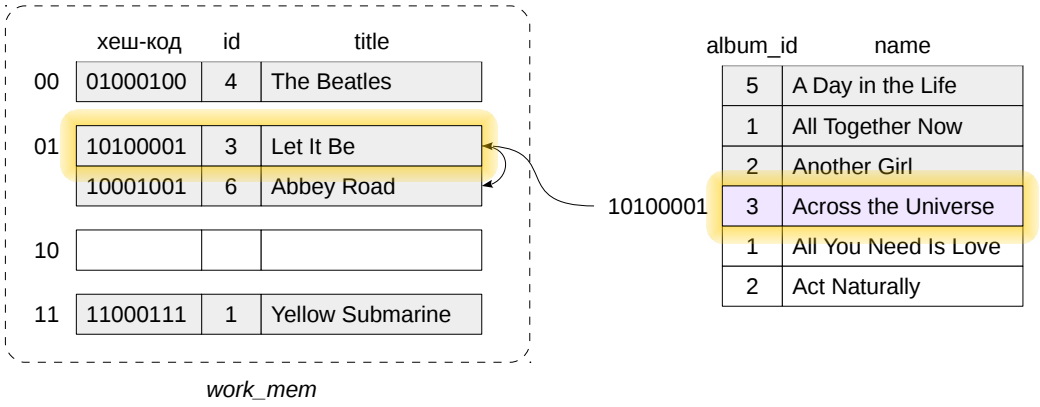
```
SELECT a.title, s.name
FROM albums a JOIN songs s ON a.id = s.album_id;
```



Для третьей строки соответствия нет (соответствующая корзина хеш-таблицы пуста).

Выполнение соединения

```
SELECT a.title, s.name
FROM albums a JOIN songs s ON a.id = s.album_id;
```

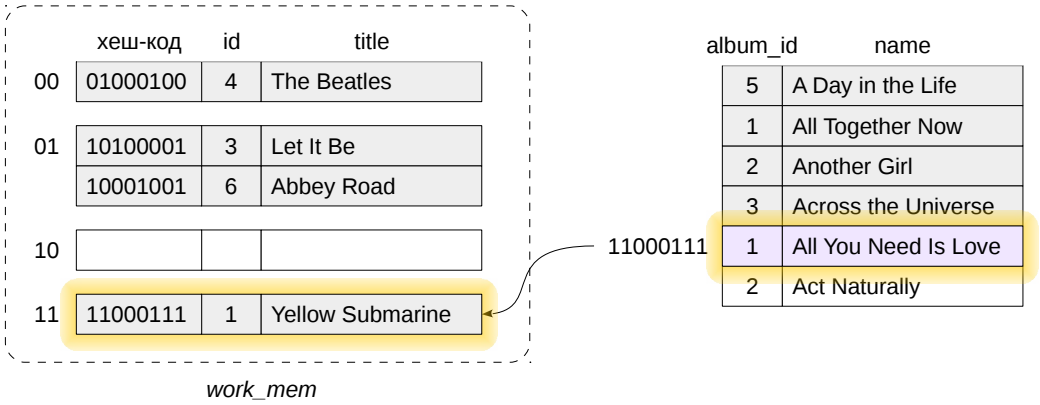


Для четверной получаем соответствие («Let It Be», «Across the Universe»).

Заметим, что в корзине хеш-таблицы оказалось две строки первого набора, и в общем случае их придется просмотреть обе.

Выполнение соединения

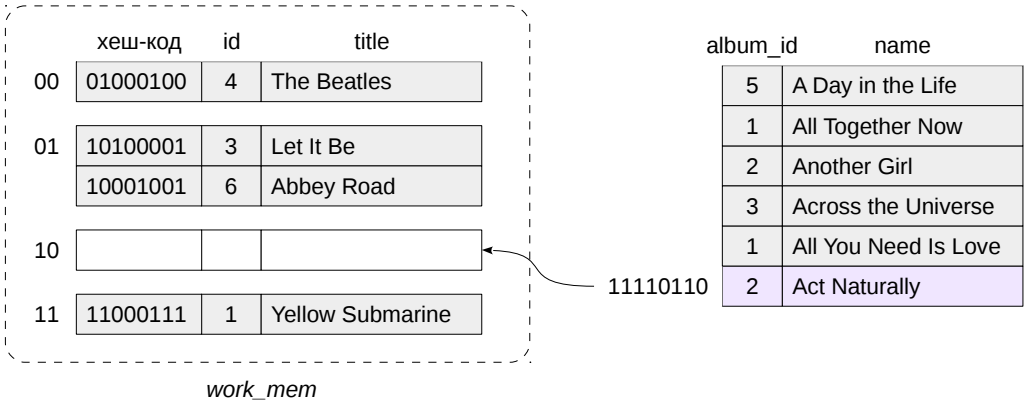
```
SELECT a.title, s.name
FROM albums a JOIN songs s ON a.id = s.album_id;
```



Для пятой строки получаем соответствие («Yellow Submarine», «All You Need Is Love»).

Выполнение соединения

```
SELECT a.title, s.name
FROM albums a JOIN songs s ON a.id = s.album_id;
```



Для шестой строки соответствия нет. На этом работа соединения завершена.

Вычислительная сложность

$\sim N + M$,

где N и M — число строк в первом и втором наборах данных

Начальные затраты на построение хеш-таблицы

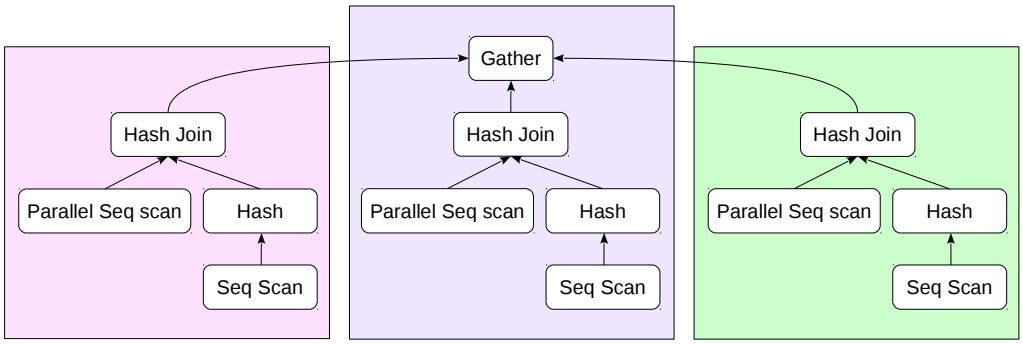
Эффективно для большого числа строк

Общая сложность соединения хешированием пропорциональна сумме числа строк в одном и другом наборах данных. Поэтому метод соединения хешированием гораздо эффективнее вложенного цикла при большом числе строк.

Однако, чтобы начать соединение, требуется заплатить накладные расходы на построение хеш-таблицы: из-за этого при небольшом числе строк эффективнее вложенный цикл.

Соединение хешированием (в сочетании с полным сканированием таблиц) характерно для OLAP-запросов, в которых надо обработать большое число строк, причем общая пропускная способность важнее времени отклика.

В параллельном режиме



каждый процесс строит собственную хеш-таблицу в локальной памяти

сканирование второго набора строк выполняется параллельно

полностью параллельное выполнение в версии 11

Начиная с версии PostgreSQL 9.6, соединение хешированием может выполняться в «почти параллельном» режиме.

Сначала каждый из рабочих процессов читает первый набор строк и строит в своей локальной памяти свою собственную хеш-таблицу — естественно, у всех процессов она получается одинаковая. (Заметим, что с диска данные будут фактически читаться только один раз: первый процесс начинает последовательное сканирование, а остальные подключаются к буферному кольцу и получают доступ к уже прочитанным в кэш страницам — так работает последовательное сканирование.)

Чтение второго набора строк выполняется уже с помощью параллельного последовательного сканирования. Таким образом, кажый из рабочих процессов проверяет по хеш-таблице только часть данных.

В версии 11 реализовано полноценное параллельное соединение хешированием: сначала все рабочие процессы строят одну общую хеш-таблицу в разделяемой памяти, а затем — читают и проверяют второй набор строк. Подробнее см. статью автора патча Томаса Мунро.

Использование памяти

Общий подход

Использование памяти при соединении хешированием

Общий подход

Оперативная память

- память каждой операции ограничена параметром *work_mem*
- при выполнении запроса несколько операций могут использовать память одновременно
- если выделенной памяти не хватает, используется диск (иногда операция может и превысить ограничение)
- больше памяти — быстрее выполнение

Дисковая память

- общая дисковая память сеанса ограничена параметром *temp_file_limit* (без учета временных таблиц)

В рассмотренном примере хеш-таблица полностью поместилась в локальной памяти процесса, ограниченной значением *work_mem*.

Если хеш-таблица оказывается больше, то используется алгоритм, использующий как имеющуюся оперативную память, так и обращающийся по мере необходимости ко временным файлам на диске.

Это справедливо и для других операций, использующих оперативную память (например, для сортировки, с которой мы познакомимся позже).

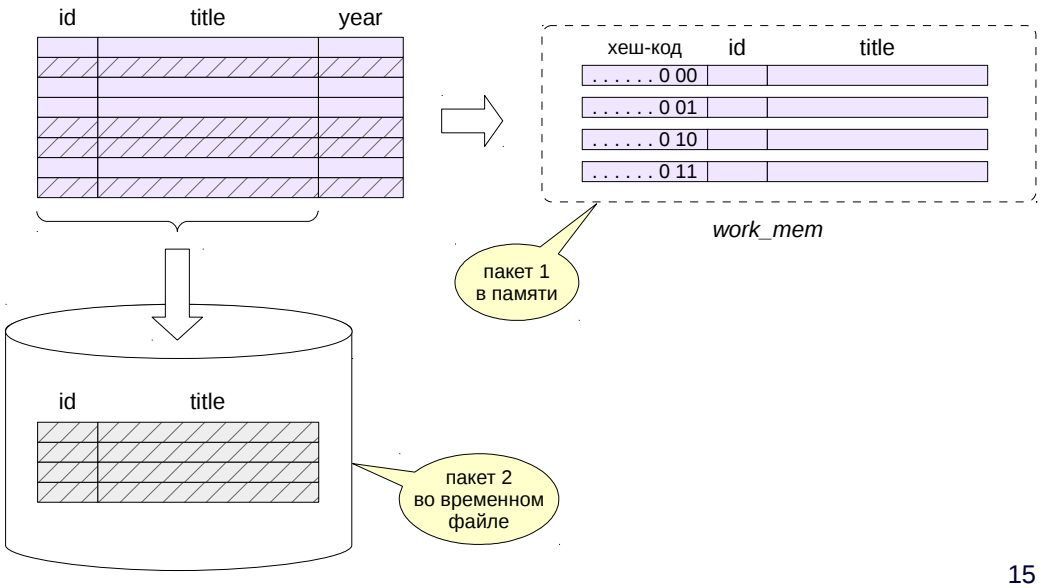
Ограничение задается для каждой отдельной операции. Следует учитывать, что при выполнении запроса может одновременно выполняться несколько таких операций, и каждая из них будет использовать память в размере *work_mem*. Кроме того, само ограничение не является жестким: некоторые операции в случае безысходности могут выходить за ограничение. Поэтому сложно предугадать реальный размер памяти, который может быть задействован процессом.

Использование временных файлов на диске также можно ограничить. Здесь параметром *temp_file_limit* определяется общее ограничение дисковой памяти для сеанса. Временные таблицы в это ограничение не входят.

В целом, чем больше памяти разрешено использовать параметром *work_mem,* тем быстрее будет идти обработка.

https://postgrespro.ru/docs/postgresql/10/runtime-config-resource#GUC-WORK-MEM

Построение хеш-таблицы



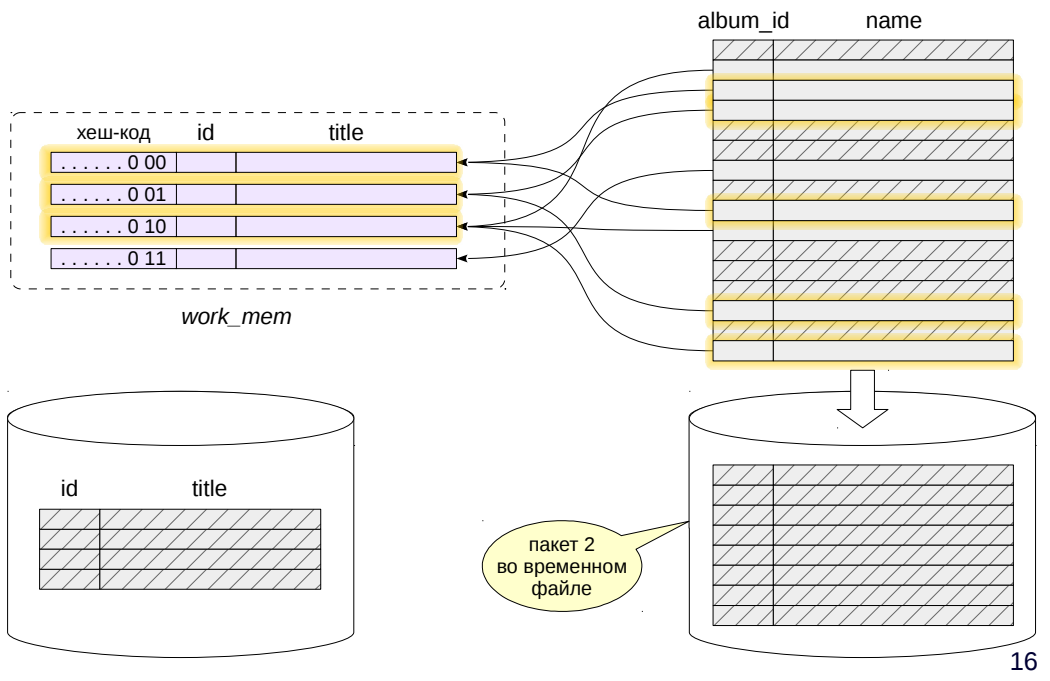
Если хеш-таблица не помещается в *work_mem,* первый набор строк разбивается на отдельные пакеты — так, чтобы в каждый пакет попало примерно одинаковое число строк. На рисунке показано два пакета — строки, относящиеся ко второму, выделены штриховкой.

При планировании запроса заранее вычисляется минимально необходимое число пакетов так, чтобы хеш-таблица для каждого пакета помещалась в памяти. Это число не уменьшается, даже если оптимизатор ошибся с оценками, но при необходимости может динамически увеличиваться.

В принципе, может так не повезти, что увеличение числа пакетов не приведет к уменьшению хеш-таблицы. В этом случае хеш-таблица будет занимать больше памяти, чем *work_mem* (в надежде на то, что общей памяти сервера хватит).

Так или иначе, но хеш-таблица для первого пакета остается в памяти, а строки, принадлежащие другим пакетам, сбрасываются на диск во временные файлы — каждый пакет в свой файл.

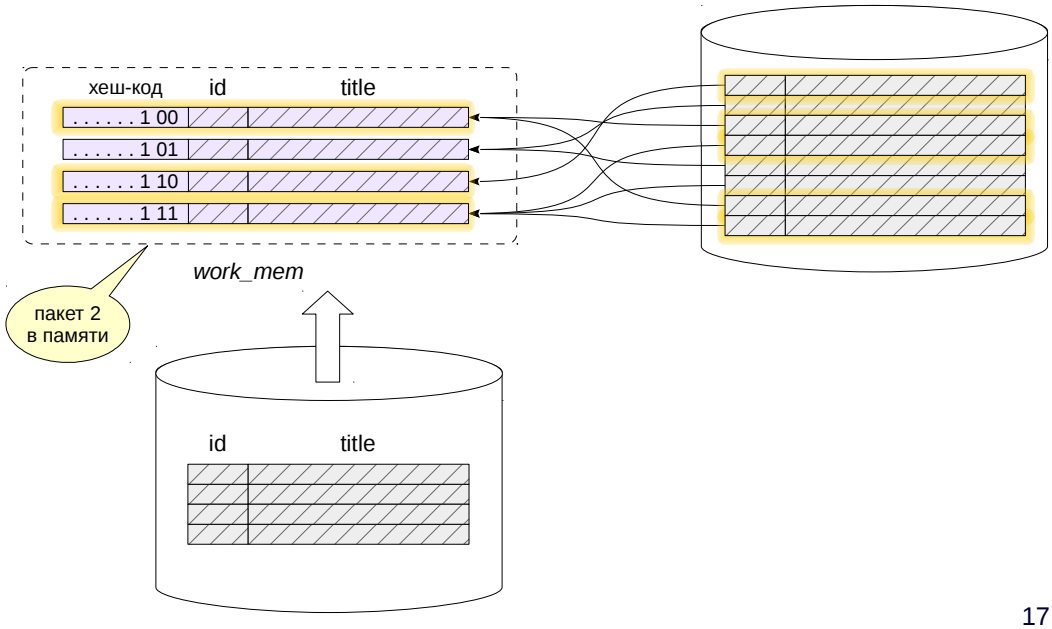
Соединение – пакет 1



Далее мы читаем второй набор строк. Если строка принадлежит первому пакету, сопоставляем ее с хеш-таблицей так же, как и в простом варианте. Если же строка принадлежит другому пакету, сбрасываем ее на диск во временный файл — опять же, каждый пакет в свой файл.

Таким образом, при *N* пакетах обычно будет использоваться 2(*N*–1) файлов (возможно меньше, если часть пакетов окажется пустыми).

Соединение – пакет 2



Далее по очереди обрабатываются все пакеты, начиная со второго. Из временного файла в хеш-таблицу считываются строки первого набора, затем из другого временного файла считываются и сопоставляются строки второго набора.

Процедура повторяется для всех оставшихся *N*–1 пакетов.

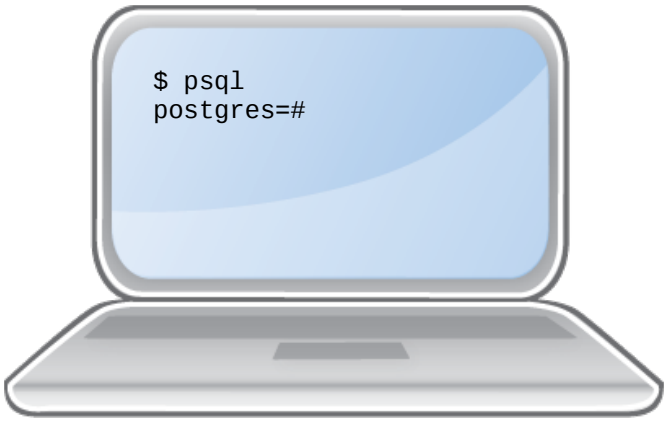
На этом соединение завершено и временные файлы освобождаются.

Заметим, что при нехватке оперативной памяти алгоритм соединения становится двухпроходным: каждый пакет (кроме первого) требуется записать на диск и затем прочитать повторно. Разумеется, это сказывается на эффективности соединения. Поэтому важно, чтобы:

- в хеш-таблицу попадали только действительно нужные поля (обязанность автора запроса),

- хеш-таблица строилась по меньшему набору строк (обязанность планировщка).

Демонстрация



Итоги

Соединение хешированием требует подготовки

- надо построить хеш-таблицу

Эффективно для больших выборок

Зависит от порядка соединения

- внутренний набор должен быть меньше внешнего, чтобы минимизировать хеш-таблицу

Поддерживает только эквисоединения

- для хеш-кодов операторы «больше» и «меньше» не имеют смысла

В отличие от соединения вложенным циклом, хеш-соединение требует подготовки: построения хеш-таблицы. Пока таблица не построена, ни одна результирующая строка не может быть получена.

Зато соединение хешированием эффективно работает на больших объемах данных. Оба набора строк читаются последовательно и только один раз (два раза в случае нехватки оперативной памяти).

Ограничением соединения хеширования является поддержка только эквисоединений. Дело в том, что хеш-значения можно сравнивать только на равенство, операции «больше» и «меньше» просто не имеют смысла.

Практика

1. Напишите запрос, показывающий занятые места в салоне для всех рейсов.
   Какой способ соединения выбрал планировщик? Проверьте, хватило ли оперативной памяти для размещения хеш-таблиц.
2. Измените запрос, чтобы он выводил только общее количество занятых мест.
   Как изменился план запроса? Почему планировщик не использовал аналогичный план для предыдущего запроса?

1. Для этого достаточно соединить рейсы (flights) с посадочными талонами (boarding_passes).