Postgres PROFESSIONAL

Способы соединения

Темы

Соединение вложенными циклами

Соединение хэшированием

Соединение слиянием

Модификации:
левые, правые, полные, полу- и анти-соединения

Соединения

Способы соединения — не соединения SQL

- inner/left/right/full/cross join — логические операции
- способы соединения — механизм реализации

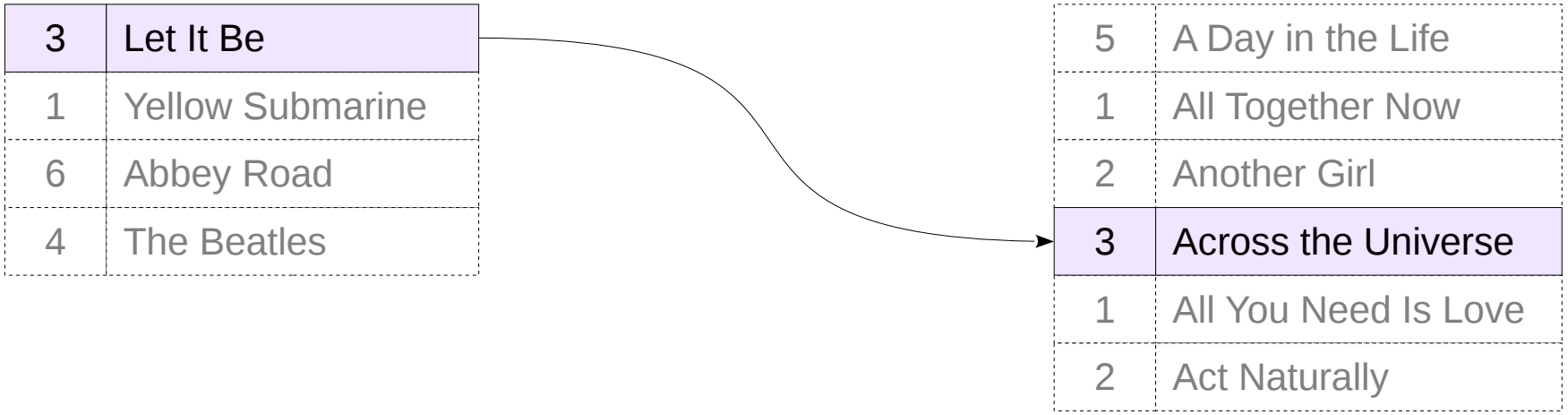
Соединяются не таблицы, а наборы строк

- могут быть получены от любого узла дерева плана

Наборы строк соединяются попарно

- порядок соединений важен с точки зрения производительности
- обычно важен и порядок внутри пары

Nested Loop



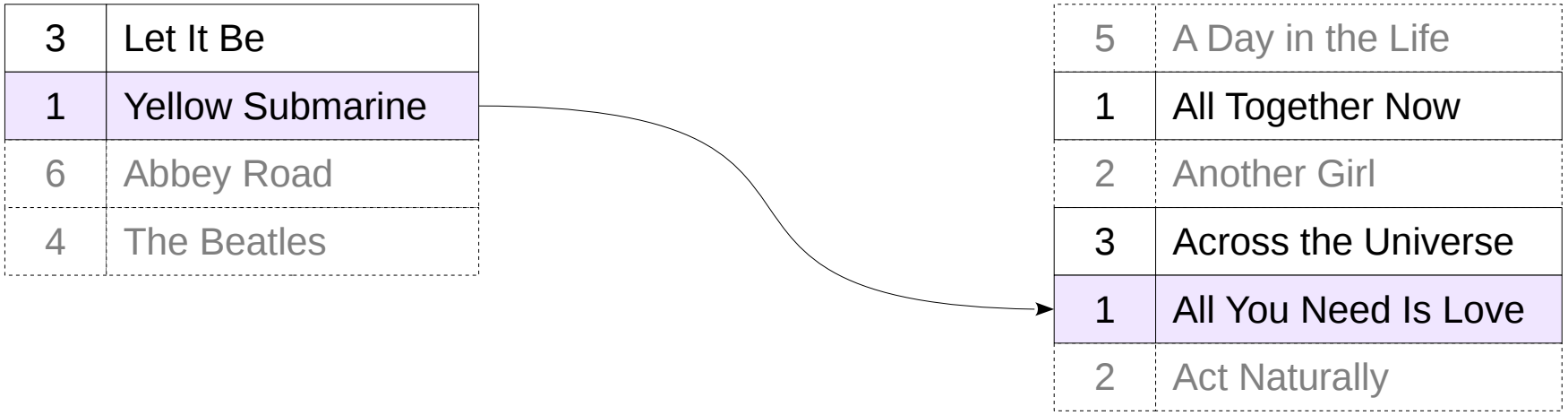
Вложенный цикл

для каждой строки одного набора
перебираем подходящие строки другого набора

Nested Loop


3 Let It Be
1 Yellow Submarine
6 Abbey Road
4 The Beatles
5 A Day in the Life
1 All Together Now
2 Another Girl
3 Across the Universe
1 All You Need Is Love
2 Act Naturally

Nested Loop


3 Let It Be
1 Yellow Submarine
6 Abbey Road
4 The Beatles
5 A Day in the Life
1 All Together Now
2 Another Girl
3 Across the Universe
1 All You Need Is Love
2 Act Naturally

Nested Loop

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

Nested Loop

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

Nested Loop

Не требует подготовительных действий

- может отдавать результат соединения без задержек

Эффективен для небольших выборок

- внешний набор строк не очень велик
- к внутреннему набору есть эффективный доступ (обычно по индексу)

Зависит от порядка соединения

- обычно лучше, если внешний набор меньше внутреннего

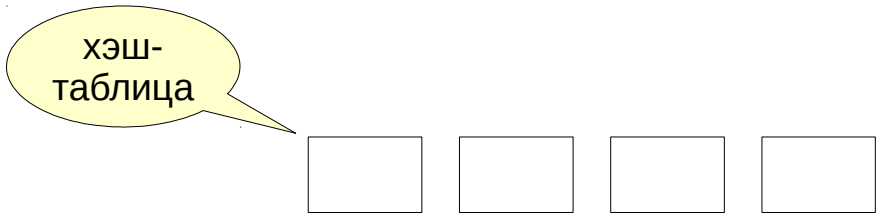
Поддерживает соединение по любому условию

- как эквисоединения, так и любые другие

Hash Join

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

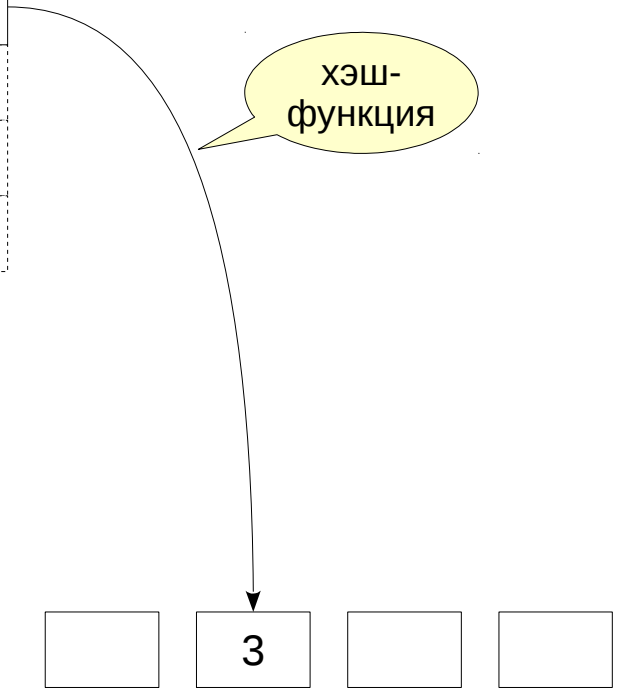
5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally



Соединение с использованием хэш-таблицы

Hash Join

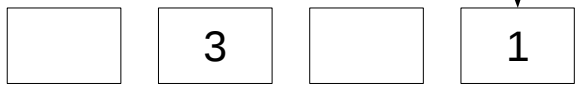
3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles



5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

Hash Join

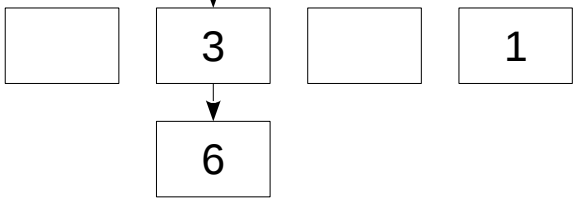
3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles



5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

Hash Join

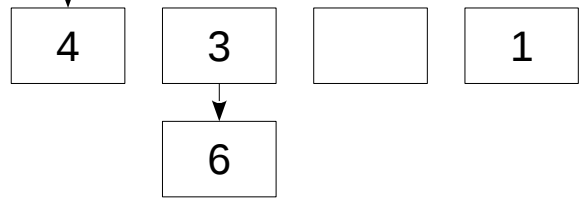
3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles



5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

Hash Join

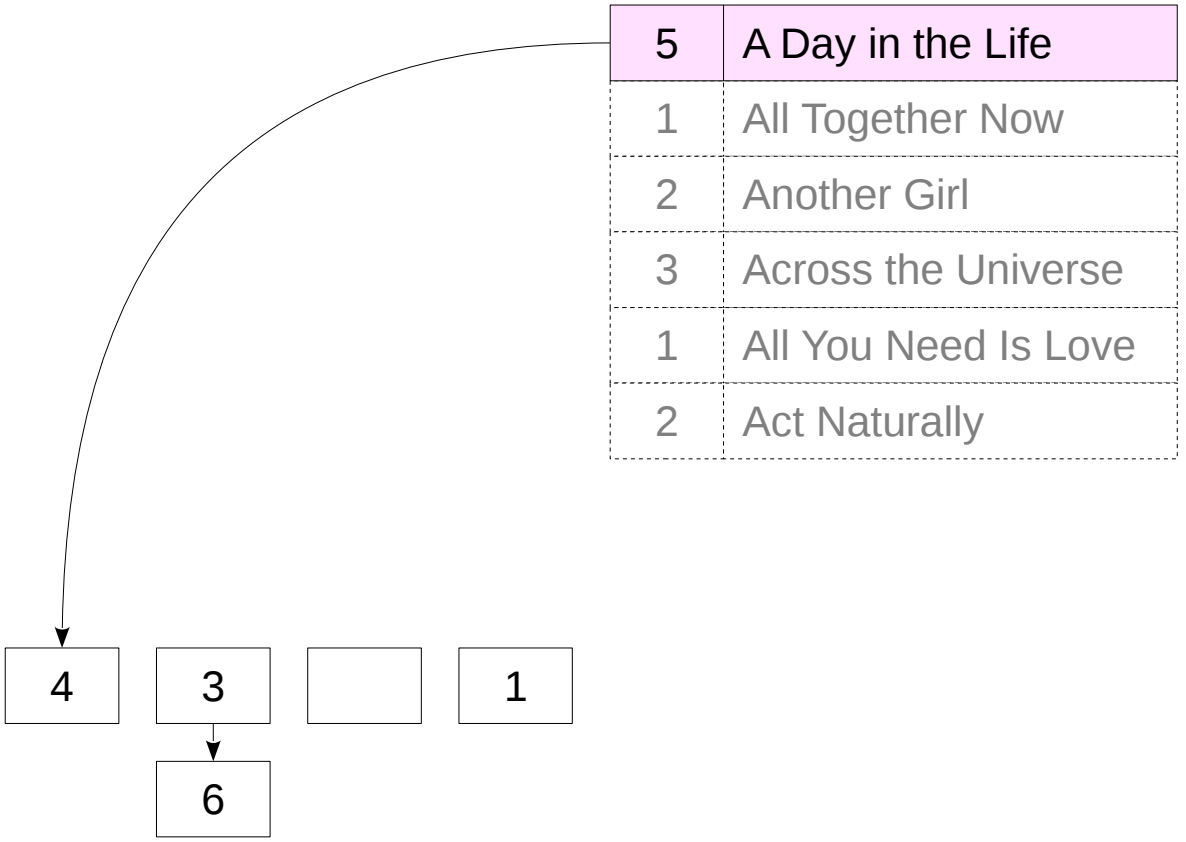
3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles



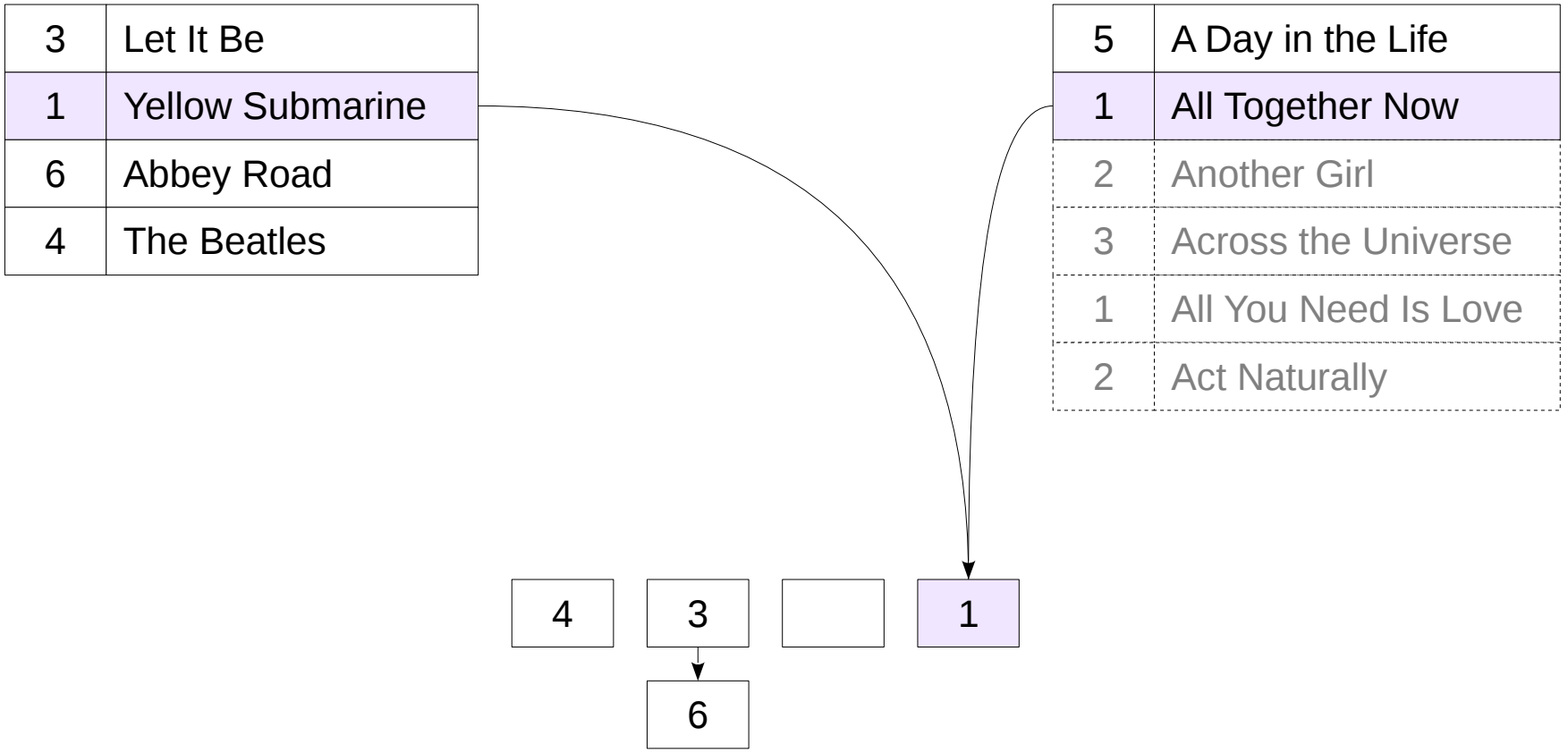
5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

Hash Join

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles



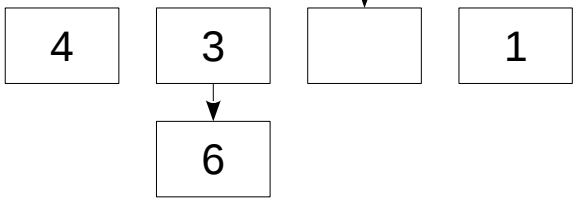
Hash Join


3 Let It Be
1 Yellow Submarine
6 Abbey Road
4 The Beatles
5 A Day in the Life
1 All Together Now
2 Another Girl
3 Across the Universe
1 All You Need Is Love
2 Act Naturally
4
3
1
6

Hash Join

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

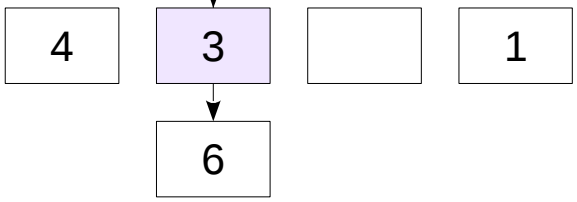
5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally



Hash Join

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

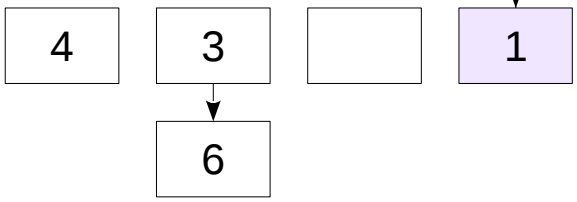
5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally



Hash Join

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

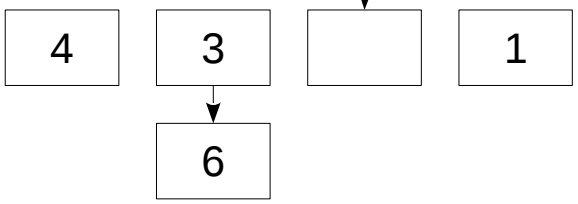
5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally



Hash Join

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally



Hash Join

Требует подготовительных действий

- надо построить хэш-таблицу

Эффективен для больших выборок

- предполагается полный перебор обоих наборов строк

Зависит от порядка соединения

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

Поддерживает только эквисоединения

- для хэш-кодов «больше» и «меньше» не имеют смысла

Merge Join

1	Yellow Submarine
3	Let It Be
4	The Beatles
6	Abbey Road

сортировка

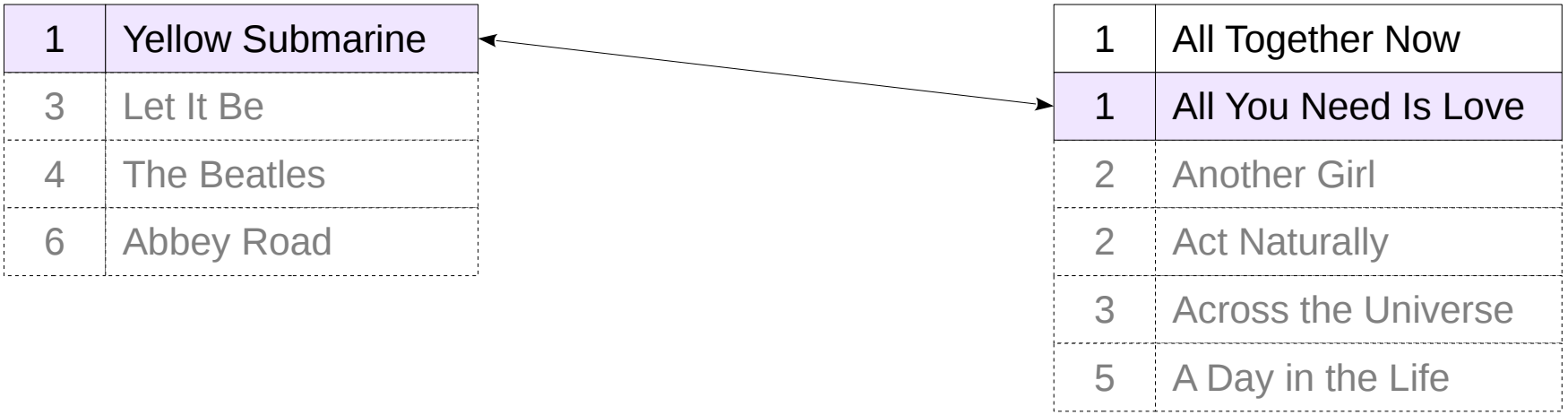
1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life

Слияние предварительно отсортированных строк

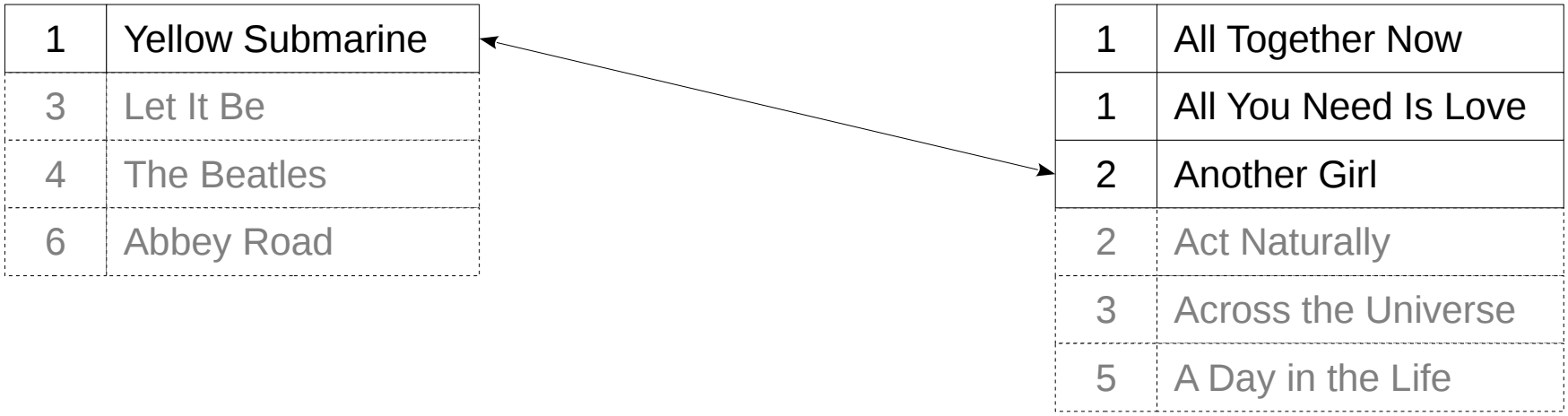
Merge Join


1 Yellow Submarine
3 Let It Be
4 The Beatles
6 Abbey Road
1 All Together Now
1 All You Need Is Love
2 Another Girl
2 Act Naturally
3 Across the Universe
5 A Day in the Life

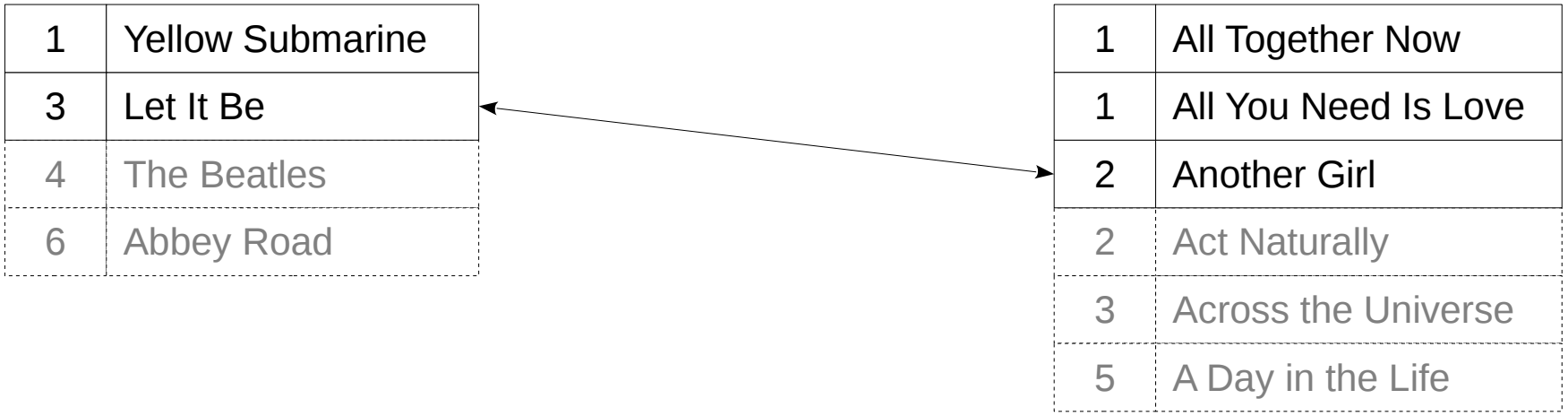
Merge Join


1 Yellow Submarine
3 Let It Be
4 The Beatles
6 Abbey Road
1 All Together Now
1 All You Need Is Love
2 Another Girl
2 Act Naturally
3 Across the Universe
5 A Day in the Life

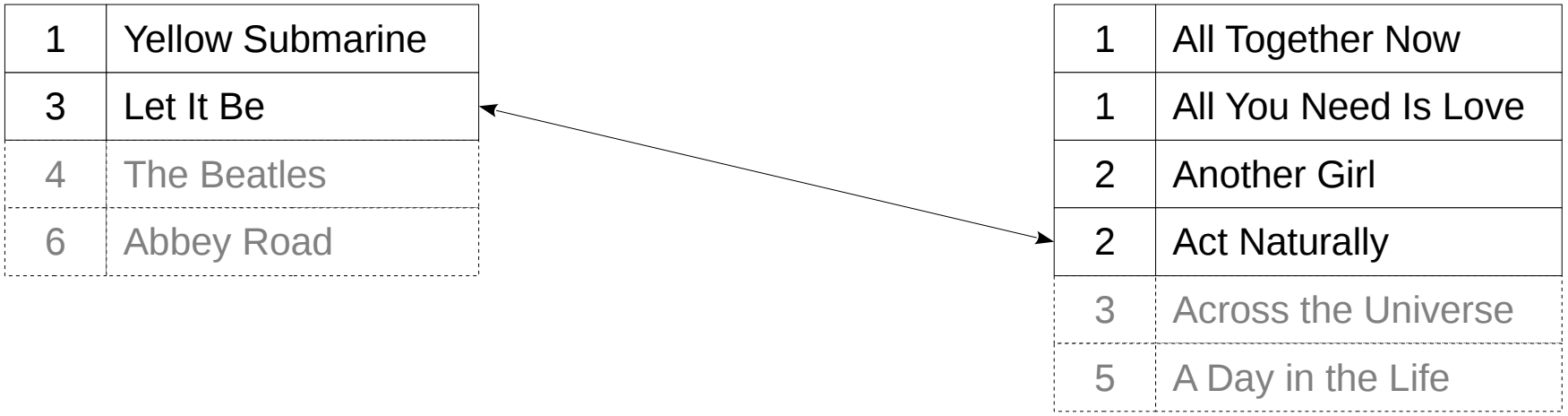
Merge Join


1 Yellow Submarine
3 Let It Be
4 The Beatles
6 Abbey Road
1 All Together Now
1 All You Need Is Love
2 Another Girl
2 Act Naturally
3 Across the Universe
5 A Day in the Life

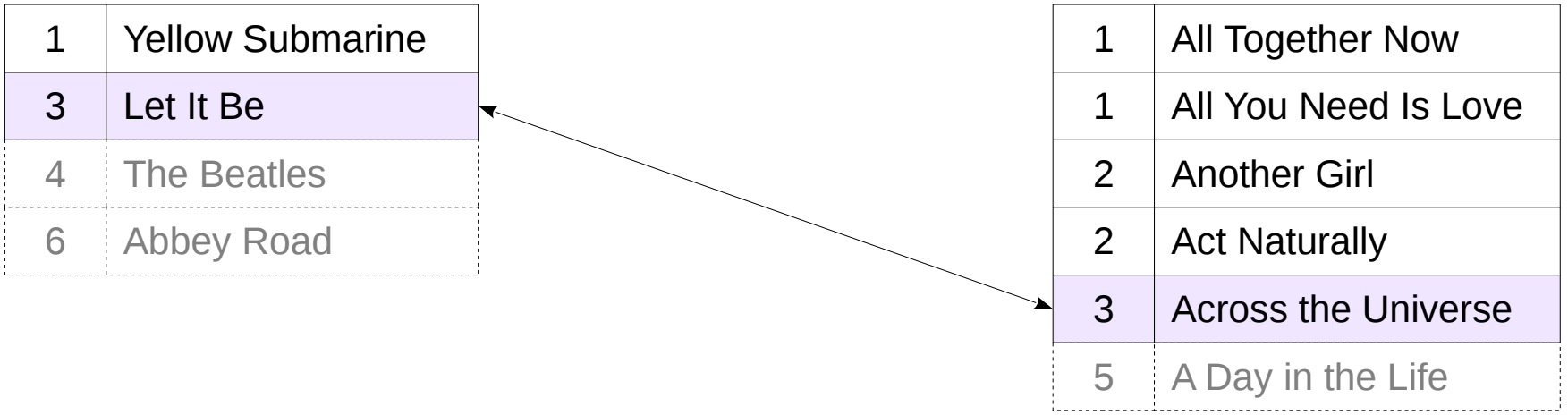
Merge Join


1 Yellow Submarine
3 Let It Be
4 The Beatles
6 Abbey Road
1 All Together Now
1 All You Need Is Love
2 Another Girl
2 Act Naturally
3 Across the Universe
5 A Day in the Life

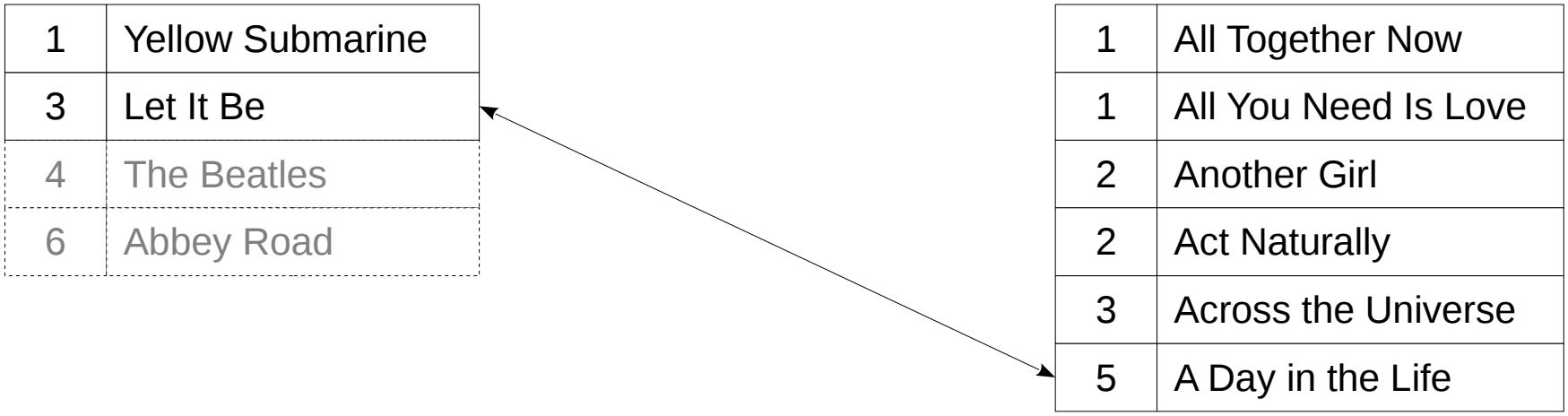
Merge Join


1 Yellow Submarine
3 Let It Be
4 The Beatles
6 Abbey Road
1 All Together Now
1 All You Need Is Love
2 Another Girl
2 Act Naturally
3 Across the Universe
5 A Day in the Life

Merge Join

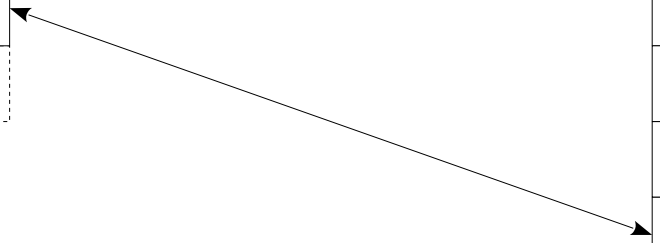

1 Yellow Submarine
3 Let It Be
4 The Beatles
6 Abbey Road
1 All Together Now
1 All You Need Is Love
2 Another Girl
2 Act Naturally
3 Across the Universe
5 A Day in the Life

Merge Join


1 Yellow Submarine
3 Let It Be
4 The Beatles
6 Abbey Road
1 All Together Now
1 All You Need Is Love
2 Another Girl
2 Act Naturally
3 Across the Universe
5 A Day in the Life

Merge Join

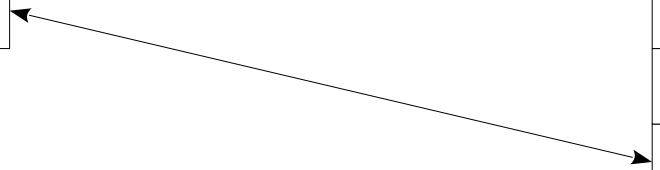
1	Yellow Submarine
3	Let It Be
4	The Beatles
6	Abbey Road



1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life

Merge Join

1	Yellow Submarine
3	Let It Be
4	The Beatles
6	Abbey Road



1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life

Merge Join

Требует подготовительных действий

- надо отсортировать наборы строк
- или получить их заранее отсортированными

Эффективен для больших выборок

- предполагается полный перебор обоих наборов строк
- хорошо, если наборы уже отсортированы
- хорошо, если нужен отсортированный результат

Не зависит от порядка соединения

Поддерживает только эквисоединения

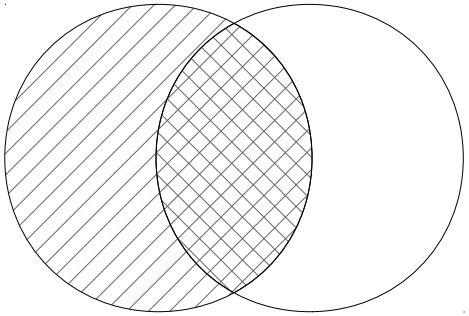
- другие не реализованы, но принципиальных ограничений нет

Модификации

Left (right)

возвращает строки, даже если
для одного набора строк
не нашлось соответствия в другом наборе

SQL: `a left|right join b`

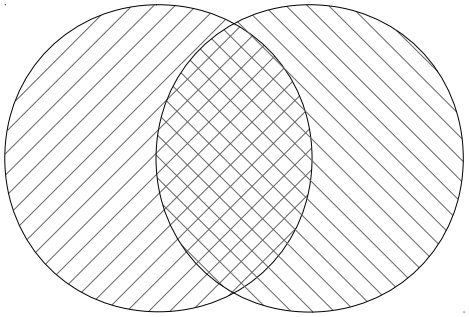


Full

возвращает строки, даже если
для любого набора строк
не нашлось соответствия в другом наборе

SQL: `a full join b`

только эквисоединение

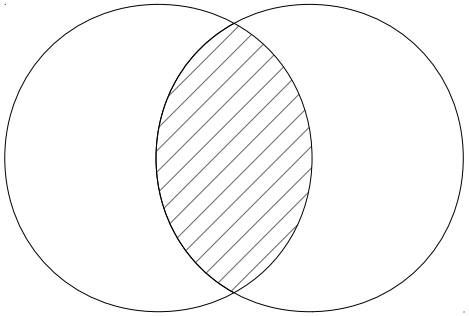


Модификации

Semi

возвращает строки одного набора, если только для них нашлось хотя бы одно соответствие в другом наборе

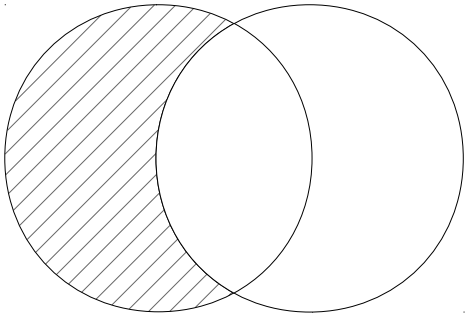
SQL: `exists`



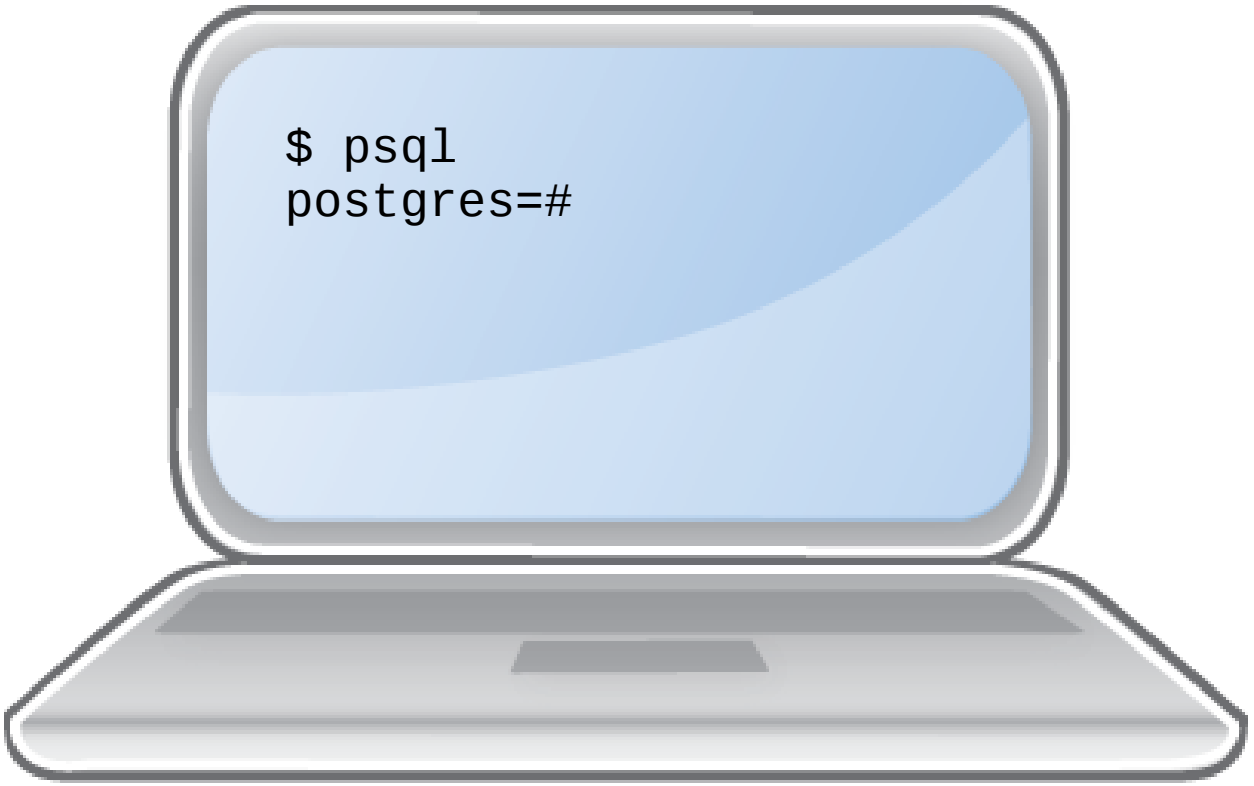
Anti

возвращает строки одного набора, если только для них не нашлось соответствия в другом наборе

SQL: `not exists`



Демонстрация



Итоги

Существуют три способа соединения

- соединение вложенными циклами
- соединение хэшированием
- соединение слиянием

На применимость и эффективность влияет много факторов

- способ SQL-соединения
- условия соединения
- размер соединяемых наборов строк
- возможности получения данных из набора строк (постепенно; быстро; в отсортированном виде)
- нужен ли отсортированный результат
- и другие

Практика

1. Создайте базу DB16 и в ней две таблицы — заказы и позиции:

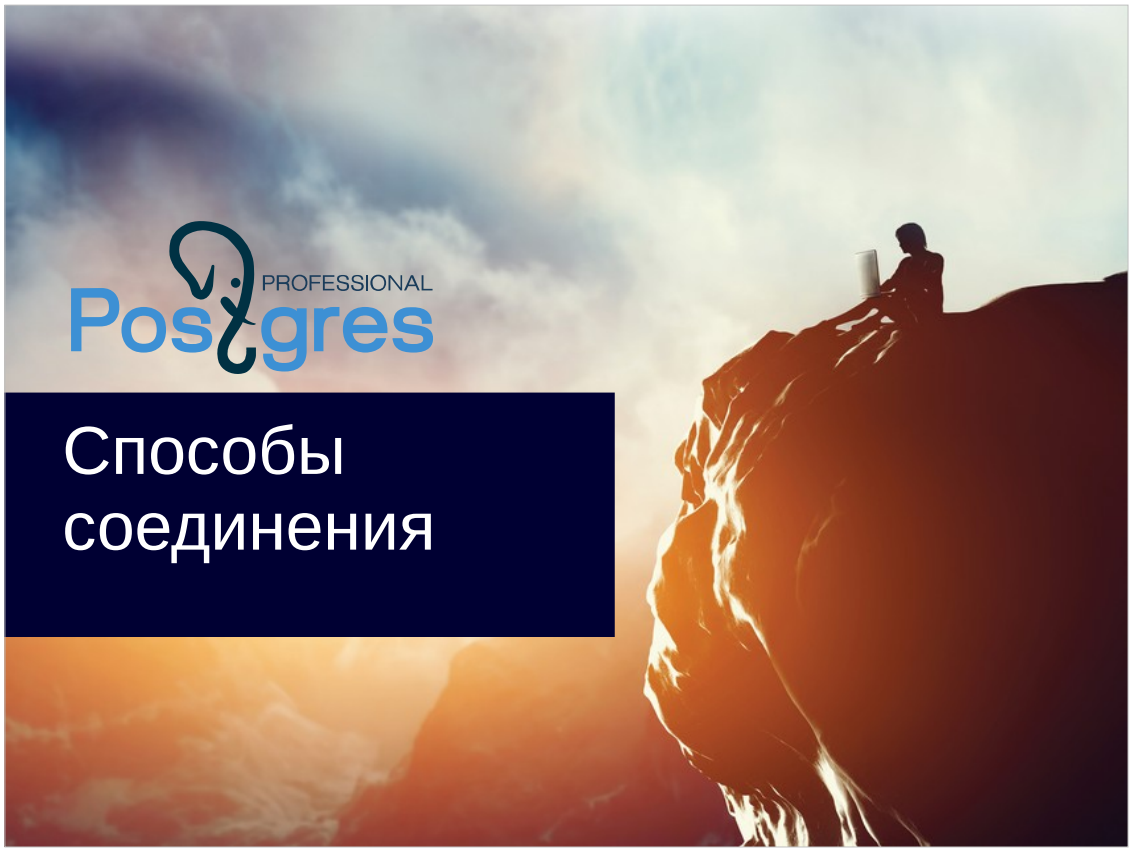
```
orders (id serial primary key,
        placed date);
items  (id serial primary key,
        order_id integer references orders(id),
        amount money);
```

   Заполните таблицы так, чтобы в заказах содержался один миллион строк, а в позициях — десять миллионов. Проиндексируйте items.order_id, проанализируйте таблицы.

2. Какое соединение будет выбрано оптимизатором, если требуется:
   — по номеру найти ровно один заказ со всеми позициями?
   — выбрать все заказы и их суммы?

3. Измените план, чтобы использовался другой способ соединения, и сравните скорость выполнения запросов до и после изменения.

4*. Допустим, предложение select ... group by выполняется с помощью сортировки. Будут ли данные еще раз сортироваться, если добавить фразу order by? A order by ... desc?
   Постройте пример и проверьте, посмотрев на план запроса.



Авторы: Егор Рогов, Павел Лузанов

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

Темы

Соединение вложенными циклами

Соединение хэшированием

Соединение слиянием

Модификации:
левые, правые, полные, полу- и анти-соединения

Соединения

Способы соединения — не соединения SQL

- inner/left/right/full/cross join — логические операции
- способы соединения — механизм реализации

Соединяются не таблицы, а наборы строк

- могут быть получены от любого узла дерева плана

Наборы строк соединяются попарно

- порядок соединений важен с точки зрения производительности
- обычно важен и порядок внутри пары

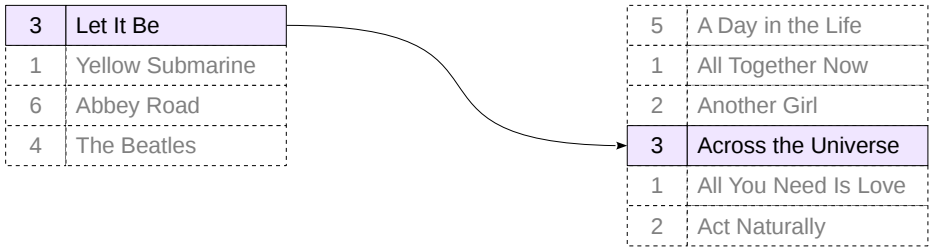
Мало получать данные с помощью методов доступа, надо уметь объединять их. Для этого PostgreSQL предоставляет несколько способов.

Способы соединения являются алгоритмами, с помощью которых можно объединить два набора строк. Их можно рассматривать как механизмы, с помощью которых реализуются операции соединения в языке SQL. Не стоит путать одно с другим: SQL-соединения — это логические операции с двумя множествами; способы соединения PostgreSQL — это возможные реализации таких соединений, учитывающие вопросы производительности.

Часто можно услышать, что соединяются *таблицы*. Это удобное упрощение, но на самом деле в общем случае соединяются *наборы строк*. Эти наборы действительно могут быть получены из таблицы (с помощью одного из методов доступа), но с тем же успехом могут быть и результатом соединения других наборов строк.

Наконец, наборы строк всегда соединяются попарно. Порядок, в котором происходит соединение не важен для SQL (то есть с точки зрения логики запроса), но очень важен с точки зрения производительности. Как мы увидим дальше, важен и порядок, в котором соединяются два набора строк.

Nested Loop



Вложенный цикл

для каждой строки одного набора
перебираем подходящие строки другого набора

Начнем с соединения вложенными циклами (nested loop), как с самого простого. Его алгоритм таков: для каждой строки одного из наборов перебираем и возвращаем соответствующие ему строки второго набора. По сути, это два вложенных цикла, отсюда и название способа.

Заметим, что ко второму набору мы будем обращаться столько раз, сколько строк в первом наборе. Если нет эффективного метода доступа для поиска «соответствующих» строк во втором наборе (то есть, попросту говоря, индекса на таблице), то придется неоднократно перебирать большое количество строк, не относящихся к делу. Очевидно, это будет не лучший выбор.

Рисунки проиллюстрируют этот способ соединения. На них:

- Серым цветом обозначены строки, к которым еще не было доступа.

- Фиолетовым цветом выделены пары соответствующих друг другу строк, которые должны войти в результирующий набор.

- Розовым цветом выделены строки, для которых не нашлось пары.

Строки соединяются по условию равенства числовых идентификаторов.

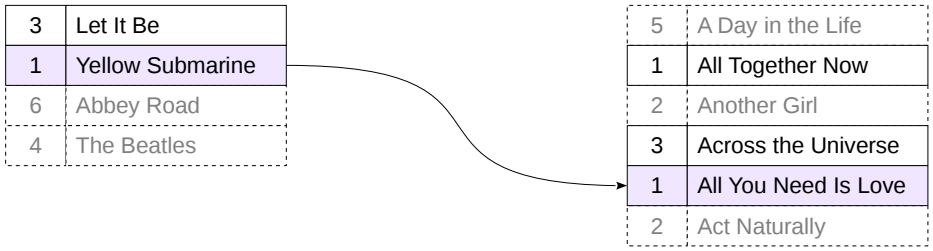
Сначала мы читаем первую строку первого набора и находим ее пару во втором наборе. Соответствие нашлось, и у нас уже есть первая строка результата, которую можно вернуть вышестоящему узлу плана: («Let It Be», «Across the Universe»).

Nested Loop

Другой пары для первой строки нет, поэтому читаем вторую строку первого набора.

Для нее тоже есть пара из второго набора: возвращаем («Yellow Submarine», «All Together Now»).

Nested Loop



В данном случае есть еще одно соответствие: («Yellow Submarine», «All You Need Is Love»).

Nested Loop

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

Переходим к третьей строке первого набора. Для нее соответствия нет.

Nested Loop

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

Для четвертой строки тоже нет соответствий. На этом работа соединения заканчивается.

Заметим, что к части строк из второго набора мы не обращались вовсе.

(Точный алгоритм можно увидеть в файле src/backend/executor/nodeNestloop.c.)

Nested Loop

Не требует подготовительных действий

- может отдавать результат соединения без задержек

Эффективен для небольших выборок

- внешний набор строк не очень велик
- к внутреннему набору есть эффективный доступ (обычно по индексу)

Зависит от порядка соединения

- обычно лучше, если внешний набор меньше внутреннего

Поддерживает соединение по любому условию

- как эквисоединения, так и любые другие

Сильной стороной способа соединения вложенными циклами является его простота: не требуется никаких подготовительных действий, мы может начать возвращать результат практически моментально.

Обратная сторона состоит в том, что этот способ крайне неэффективен для больших объемов данных. Ситуация та же, что и с индексами: чем больше выборка, чем больше накладных расходов.

Таким образом, соединение вложенными циклами имеет смысл применять, если:

- один из наборов строк небольшой,

- к другому набору есть эффективный доступ по условию соединения,

- общее количество строк результата не велико.

Это обычная ситуация для, например, запросов от пользовательского интерфейса: веб-страница (или экранная форма) должны открыться быстро и не содержат большого объема информации.

Еще одна особенность, которую стоит отметить: соединение вложенными циклами может работать для любого условия соединения. Подходит и эквисоединение (по условию равенства, как в примере), так и любое другое.

Hash Join

3	Let It Be
1	Yellow Submarine
6	Abbey Road
4	The Beatles

5	A Day in the Life
1	All Together Now
2	Another Girl
3	Across the Universe
1	All You Need Is Love
2	Act Naturally

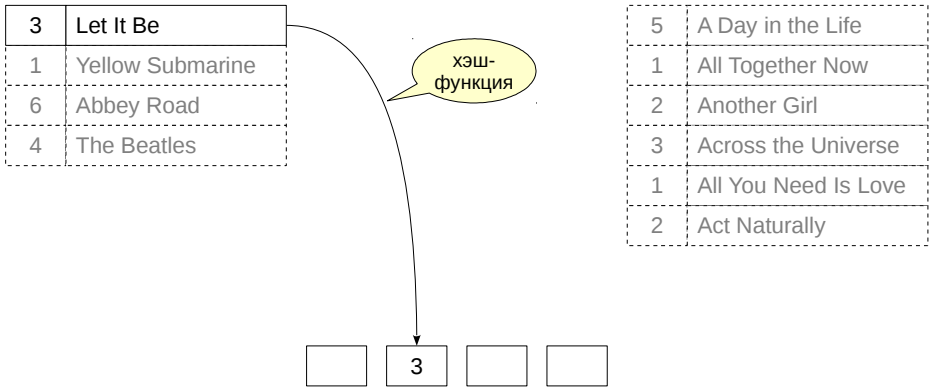
Соединение с использованием хэш-таблицы

Рассмотрим теперь соединение хэшированием.

Первым этапом в памяти строится хэш-таблица. Ключом хэширования являются значения полей, по которым происходит соединение (в нашем примере — числовые идентификаторы).

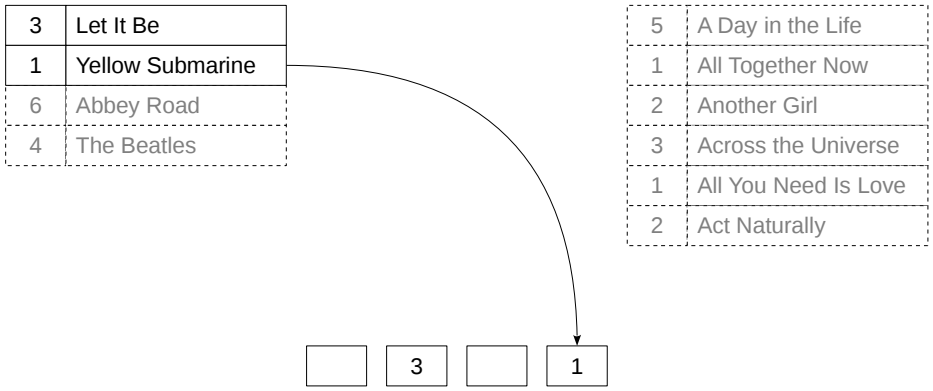
(Интересующиеся могут посмотреть алгоритм в файле src/backend/executor/nodeHashjoin.c.)

Hash Join



Строки первого набора читаются последовательно, и для каждой из них в хэш-таблицу помещается ссылка на саму строку.

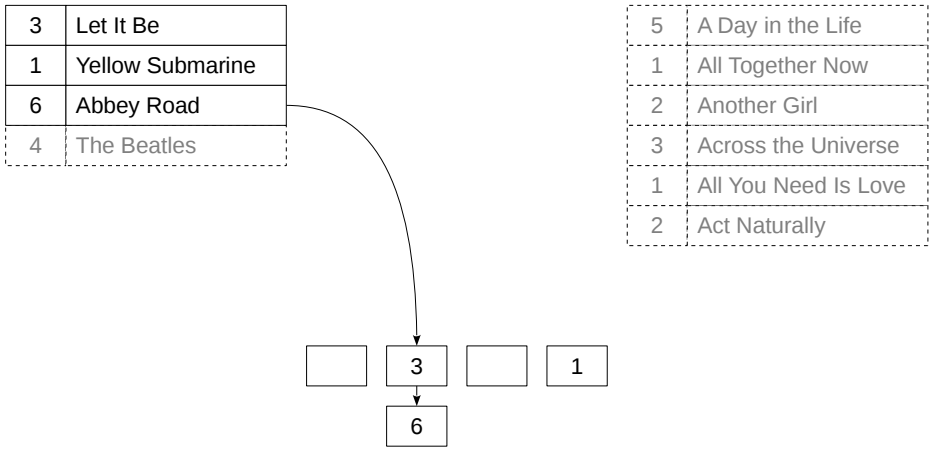
Hash Join



Идея хэширования состоит в том, что функция хэширования будет *равномерно* распределять значения по ограниченным корзинам хэш-таблицы.

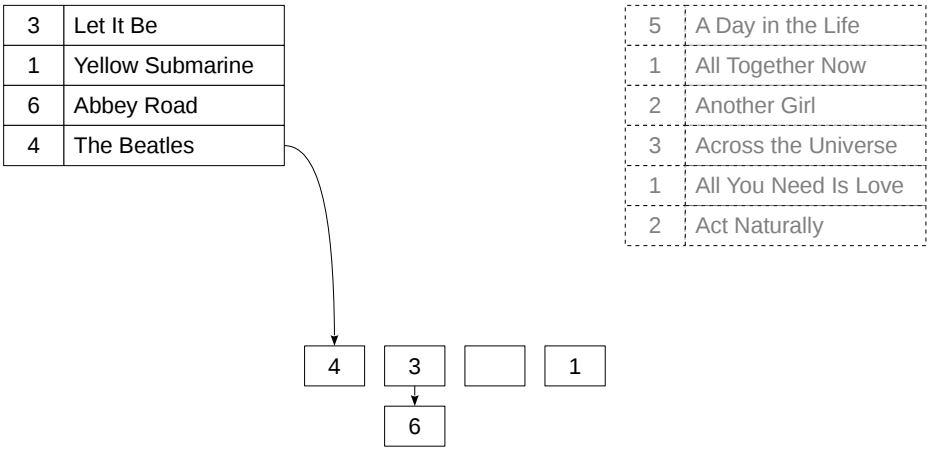
В таком случае разные значения как правило будут попадать в разные корзины хэш-таблицы.

Hash Join



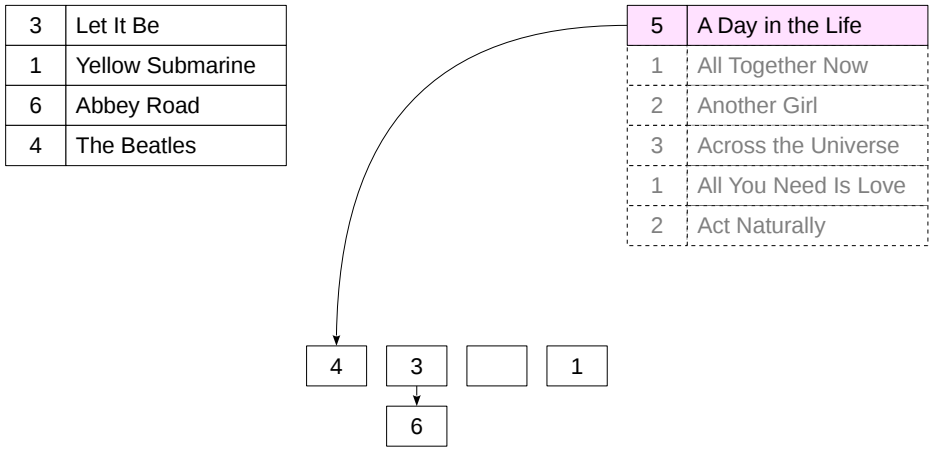
Еслиравномерности не будет, в одну корзину может попасть много значений. В таком случае они будут выстраиватся в список, и по мере увеличения длины списка эффективность хэш-таблицы будет падать.

Hash Join



На этом построение хэш-таблицы завершено, и мы переходим ко второму этапу.

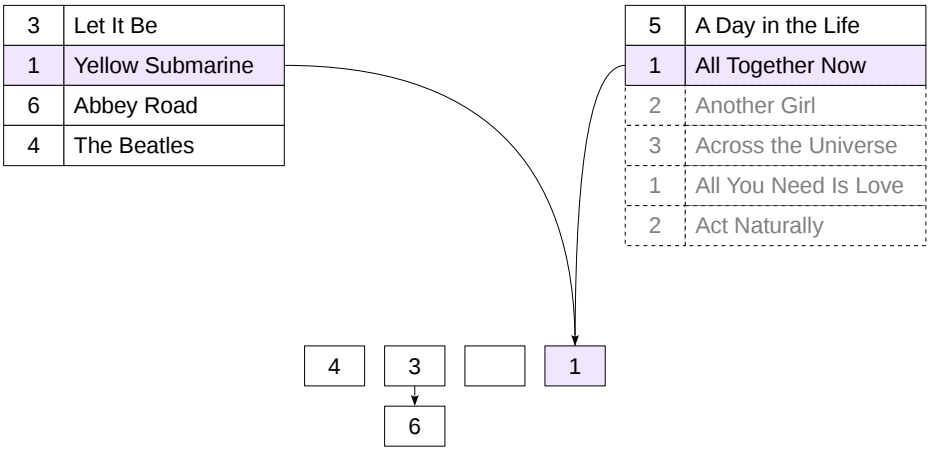
Hash Join



На втором этапе мы последовательно читаем второй набор строк. По мере чтения мы вычисляем хэш-функцию от значения полей, участвующих в условии соединения. Если в вычисленной корзине хэш-таблицы обнаруживается соответствующее значение — мы нашли пару.

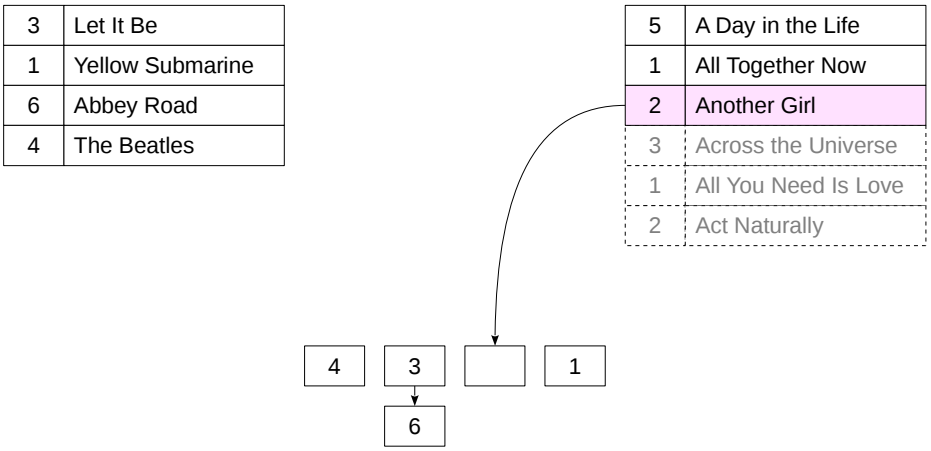
В данном случае соответствия нет.

Hash Join



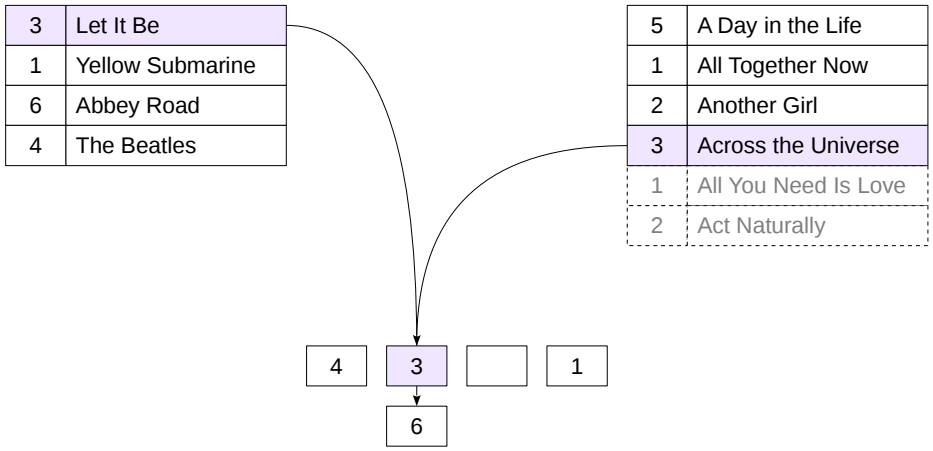
Вторая строка второго набора дает первое соответствие, которое уже можно вернуть вышестоящему узлу плана: («Yellow Submarine», «All Together Now»).

Hash Join



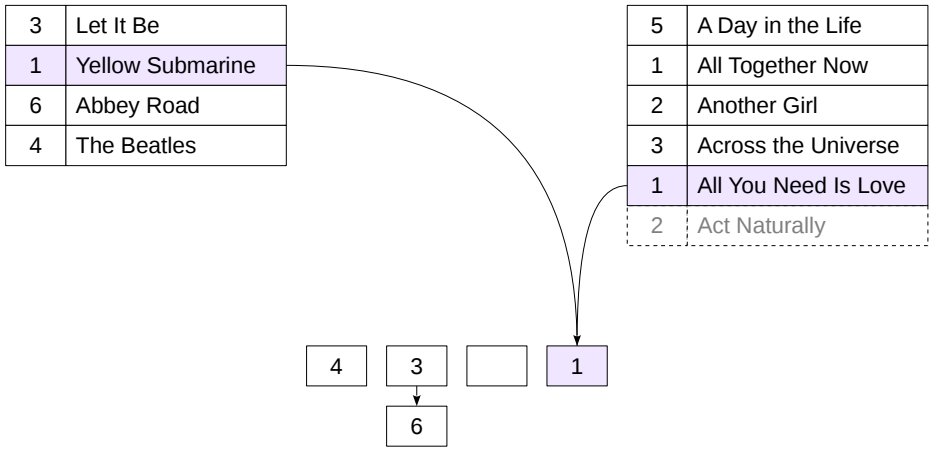
Для третьей строки соответствия нет.

Hash Join



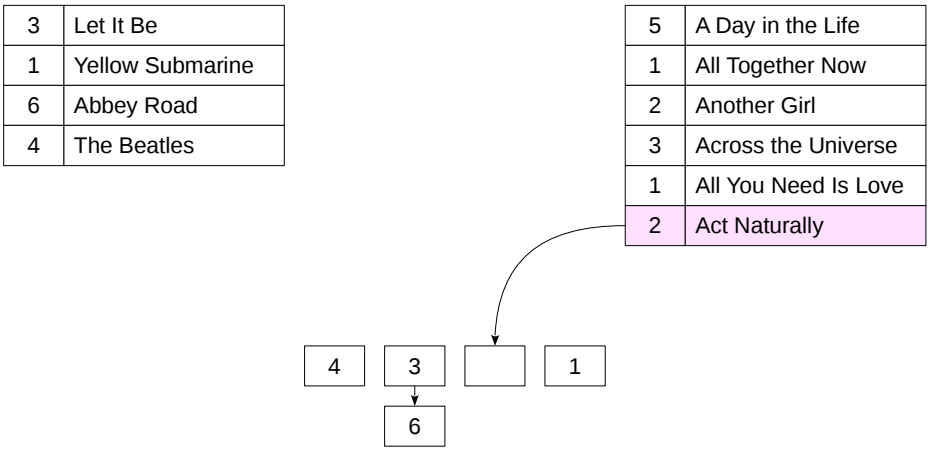
Для четверной получаем («Let It Be», «Across the Universe»).

Hash Join



Для пятой — («Yellow Submarine», «All You Need Is Love»).

Hash Join



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

Hash Join

Требует подготовительных действий

- надо построить хэш-таблицу

Эффективен для больших выборок

- предполагается полный перебор обоих наборов строк

Зависит от порядка соединения

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

Поддерживает только эквисоединения

- для хэш-кодов «больше» и «меньше» не имеют смысла

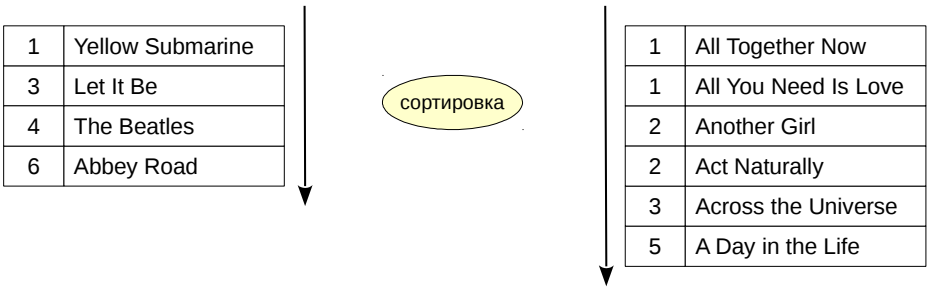
В отличие от соединения вложенными циклами, хэш-соединение требует подготовки: построения хэш-таблицы. Пока таблица не построена, ни одна результирующая строка не может быть получена.

Зато соединение хэшированием эффективно работает на больших объемах данных. Оба набора строк читаются последовательно и один раз.

Хотя мы и не рассматриваем пока вопрос использования оперативной памяти, тем не менее отметим: набор строк, по которому строится хэш-таблица, должен быть наименьшим из двух. Подробнее мы рассмотрим это в следующей теме «Использование памяти».

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

Merge Join

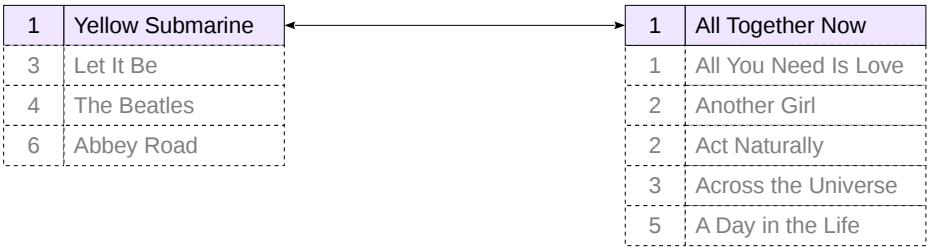


Слияние предварительно отсортированных строк

Третий, и последний, способ — соединение слиянием.

Подготовительным этапом для него случит сортировка обоих наборов строк. Сортировка — довольно дорогая операция, но иногда этого этапа удается избежать, если можно сразу получить отсортированные наборы строк (например, за счет индексного доступа к таблице).

Merge Join

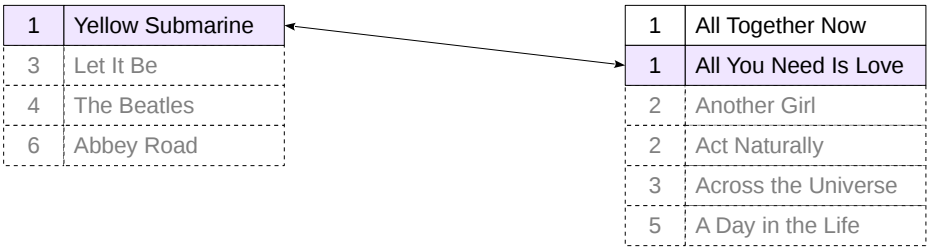


Само слияние устроено просто. Сначала берем первые строки обоих наборов и сравниваем. В данном случае мы сразу нашли соответствие и можем вернуть первую строку результата: («Yellow Submarine», «All Together Now»).

Общий алгоритм таков: читаем следующую строку того набора, для которого значение поля, по которому происходит соединение, меньше (один набор «догоняет» другой). Если же значения одинаковы, как в нашем примере, то читаем следующую строку второго набора.

(На самом деле алгоритм сложнее — что, если и в первом наборе строк может быть несколько одинаковых значений? Но не будем загромождать картину деталями. Псевдокод алгоритма можно посмотреть в файле src/backend/executor/nodeMergejoin.c.)

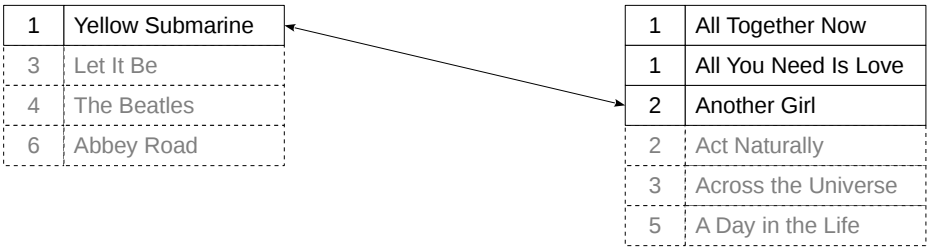
Merge Join



Вновь соответствие: («Yellow Submarine», «All You Need Is Love»).

Снова читаем следующую строку второго набора.

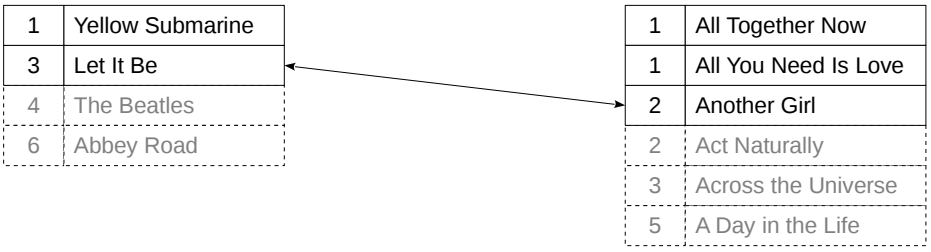
Merge Join



В данном случае соответствия нет.

Поскольку 1 < 2, читаем следующую строку первого набора.

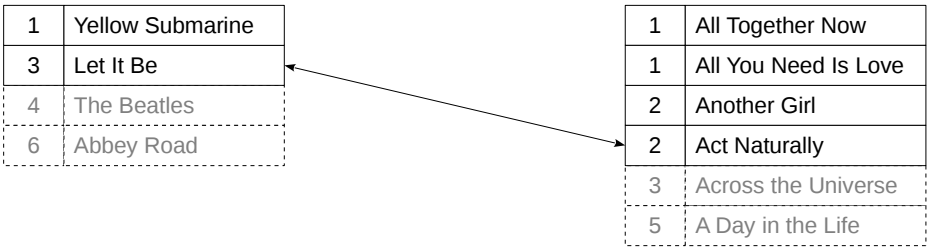
Merge Join



Соответствия нет.

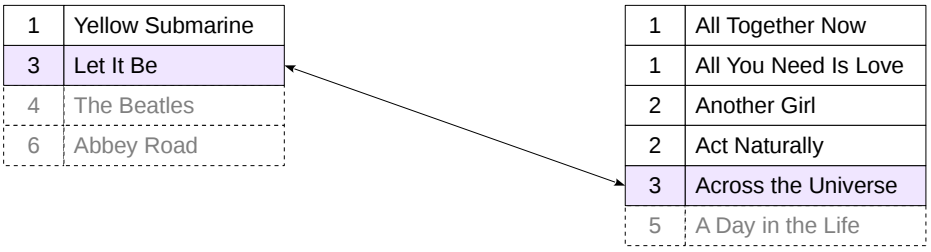
3 > 2, поэтому читаем следующую строку второго набора.

Merge Join



Снова нет соответствия, снова 3 < 2, снова читаем строку второго набора.

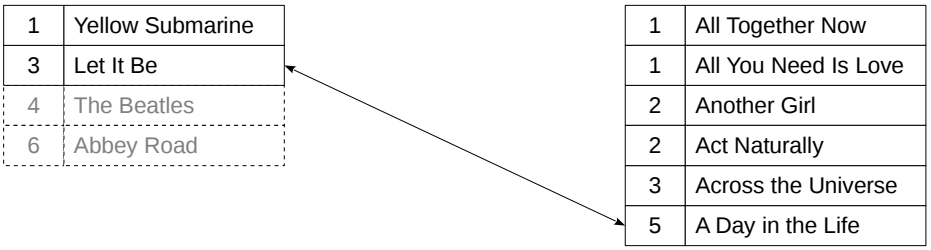
Merge Join



Есть соответствие: («Let It Be», «Across the Universe»).

3 = 3, читаем следующую строку второго набора.

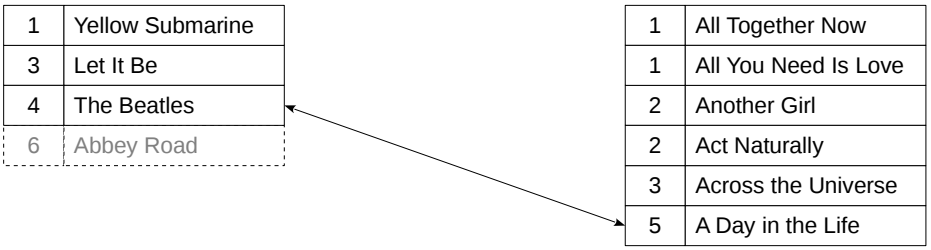
Merge Join



Соответствия нет.

3 < 5, читаем строку первого набора.

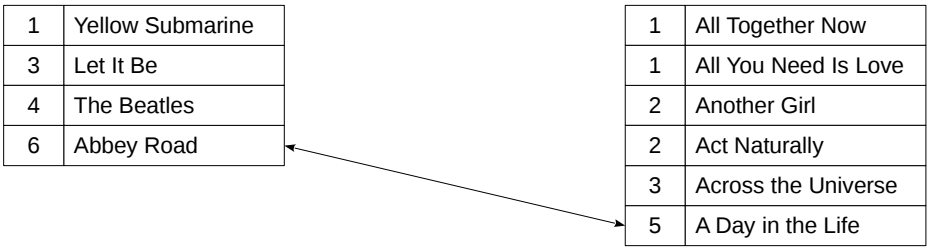
Merge Join



Соответствия нет.

4 < 5, читаем строку первого набора.

Merge Join



1	Yellow Submarine
3	Let It Be
4	The Beatles
6	Abbey Road

1	All Together Now
1	All You Need Is Love
2	Another Girl
2	Act Naturally
3	Across the Universe
5	A Day in the Life

И последний шаг: снова нет соответствия.

Merge Join

Требует подготовительных действий

- надо отсортировать наборы строк
- или получить их заранее отсортированными

Эффективен для больших выборок

- предполагается полный перебор обоих наборов строк
- хорошо, если наборы уже отсортированы
- хорошо, если нужен отсортированный результат

Не зависит от порядка соединения

Поддерживает только эквисоединения

- другие не реализованы, но принципиальных ограничений нет

Чтобы начать соединение слиянием, оба набора строк должны быть отсортированы. Хорошо, если удается получить данные уже в нужном порядке; если нет — требуется выполнить сортировку.

Само слияние выполняется очень эффективно даже для больших наборов строк. В качестве приятного бонуса результирующая выборка тоже упорядочена, поэтому такой способ соединения привлекателен, если вышестоящим узлам плана также требуется сортировка. Простой пример: запрос с фразой order by.

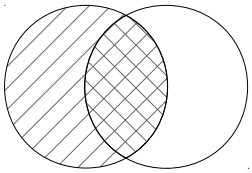
В настоящее время соединение слиянием поддерживает только эквисоединения, соединение по операциям «больше» или «меньше» не реализованы.

Модификации

Left (right)

возвращает строки, даже если
для одного набора строк
не нашлось соответствия в другом наборе

SQL: `a left|right join b`

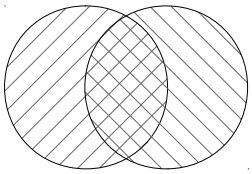


Full

возвращает строки, даже если
для любого набора строк
не нашлось соответствия в другом наборе

SQL: `a full join b`

только эквисоединение



Для того, чтобы поддерживать внешние соединения SQL, алгоритмы рассмотренных способов соединения надо изменить, но совсем немного.

В плане запроса такие узлы будут содержать в своем имени слово Left, Right или Full, помимо основного названия способа соединения.

Поскольку метод вложенных циклов в принципе не может работать с полным соединением, а хэширование и слияние поддерживают только эквисоединения, то полное соединение в настоящее время возможно только для условия равенства. В качестве обходного пути можно выразить полное соединение через левое и антисоединение.

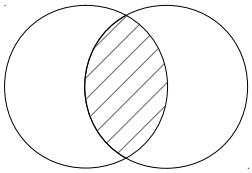
Модификации

Postgres PROFESSIONAL

Semi

возвращает строки одного набора, если только для них нашлось хотя бы одно соответствие в другом наборе

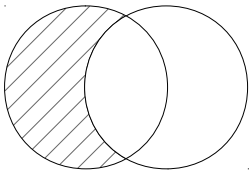
SQL: `exists`



Anti

возвращает строки одного набора, если только для них не нашлось соответствия в другом наборе

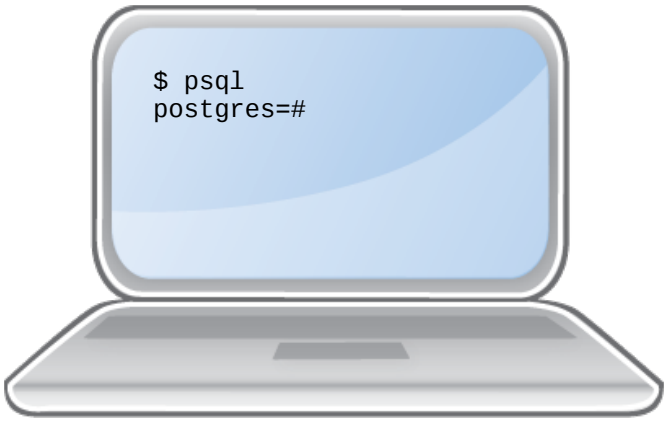
SQL: `not exists`



Для эффективной поддержки предиката exists в PostgreSQL есть модификация Semi. Способы соединения с такой модификацией работают как обычное внутреннее соединение (inner join), но результатом являются строки только первого набора данных.

Для поддержки not exists существует модификация Anti. Результатом антисоединения также являются строки только первого набора данных, но в том случае, если во втором наборе им не нашлось соответствия.

Демонстрация



Итоги

Существуют три способа соединения

- соединение вложенными циклами
- соединение хэшированием
- соединение слиянием

На применимость и эффективность влияет много факторов

- способ SQL-соединения
- условия соединения
- размер соединяемых наборов строк
- возможности получения данных из набора строк (постепенно; быстро; в отсортированном виде)
- нужен ли отсортированный результат
- и другие

Практика

1. Создайте базу DB16 и в ней две таблицы — заказы и позиции:

```
orders (id serial primary key,
        placed date);
items  (id serial primary key,
        order_id integer references orders(id),
        amount money);
```

   Заполните таблицы так, чтобы в заказах содержался один миллион строк, а в позициях — десять миллионов. Проиндексируйте items.order_id, проанализируйте таблицы.

2. Какое соединение будет выбрано оптимизатором, если требуется:
   — по номеру найти ровно один заказ со всеми позициями?
   — выбрать все заказы и их суммы?

3. Измените план, чтобы использовался другой способ соединения, и сравните скорость выполнения запросов до и после изменения.

4*. Допустим, предложение select ... group by выполняется с помощью сортировки. Будут ли данные еще раз сортироваться, если добавить фразу order by? A order by ... desc?
   Постройте пример и проверьте, посмотрев на план запроса.

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