SQL
Занятие 9
План запроса и оптимизация

Бояр Владислав

Центр непрерывного образования ФКН ВШЭ 2024

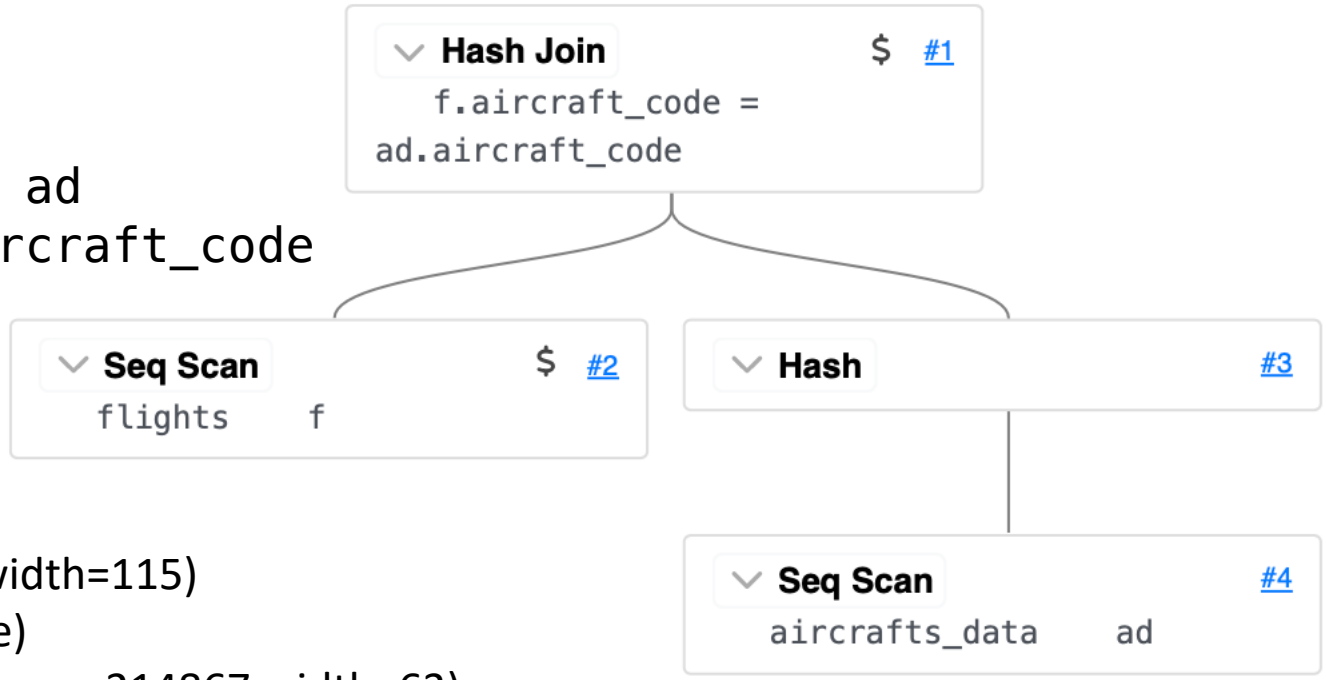
План запроса

План запроса

- Прежде, чем запустить выполнение запроса, СУБД выстраивает план его выполнения
- Посмотреть план выполнения запроса можно с помощью команды **EXPLAIN**
- Пример визуализации плана выполнения запроса: https://explain.dalibo.com/

Пример плана запроса

```
EXPLAIN
 SELECT *
   FROM bookings.flights f
   JOIN bookings.aircrafts_data ad
     ON f.aircraft_code = ad.aircraft_code
```




```
Hash Join  (cost=1.20..5603.50 rows=214867 width=115)
  Hash Cond: (f.aircraft_code = ad.aircraft_code)
  ->  Seq Scan on flights f  (cost=0.00..4772.67 rows=214867 width=63)
  ->  Hash  (cost=1.09..1.09 rows=9 width=52)
        ->  Seq Scan on aircrafts_data ad  (cost=0.00..1.09 rows=9 width=52)
```

Структура плана запроса

- Структура плана запроса представляет собой дерево узлов плана
- Узлы могут быть разных типов в зависимости от содержимого запроса:
  - Узлы на нижнем уровне дерева — это узлы сканирования, которые возвращают необработанные данные таблицы;
  - Если запрос включает объединения, агрегатные вычисления, сортировки или другие операции с исходными строками, над узлами сканирования появляются узлы, обозначающие эти операции.
- В выводе команды EXPLAIN для каждого узла в дереве плана отводится одна строка, где показывается базовый тип узла плюс оценка стоимости выполнения данного узла, которую сделал для него планировщик.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. (cost = number_1 .. number_2) – затраты (среднее время доступа до страницы в БД):
3. number_1 - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. number_2 - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. (rows= …) – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = …) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. number_1 - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. number_2 - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. (rows= …) – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = …) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. number_2 - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. (rows= …) – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = …) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. (rows= …) – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = …) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= …)** – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. (width = …) – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= …) –** ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. **(width = …)** – ожидаемый средний размер возвращаемых строк в байтах.
7. Planning time (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= …)** – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. **(width = …)** – ожидаемый средний размер возвращаемых строк в байтах.
7. **Planning time** (Время планирования) - время, затраченное на построение плана запроса и его оптимизацию.
8. Execution time (Время выполнения) - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

Структура вывода EXPLAIN

1. СУБД проверяет все возможные сценарии выполнения запроса. Она знает, сколько у вас строк и сколько строк (вероятнее всего) попадут под заданные критерии;
2. **(cost = number_1 .. number_2)** – затраты (среднее время доступа до страницы в БД):
3. **number_1** - приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки;
4. **number_2** - приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки;
5. **(rows= …)** – ожидаемое количество строк, которое запрос способен вернуть (он может вернуть меньше, например, в случае использования LIMIT);
6. **(width = …)** – ожидаемый средний размер возвращаемых строк в байтах.
7. **Planning time (Время планирования)** - время, затраченное на построение плана запроса и его оптимизацию.
8. **Execution time (Время выполнения)** - включает продолжительность запуска и остановки исполнителя запроса, а также время выполнения всех сработавших триггеров.

EXPLAIN ANALYZE

- Точность оценок планировщика можно проверить, используя команду EXPLAIN с параметром ANALYZE;
- С этим параметром EXPLAIN на самом деле выполняет запрос, а затем выводит фактическое число строк и время выполнения, накопленное в каждом узле плана, вместе с теми же оценками, что выдаёт обычная команда EXPLAIN;
- Не забывайте, что EXPLAIN ANALYZE действительно выполняет запрос, хотя его результаты могут не показываться, а заменяться выводом команды EXPLAIN. Поэтому при таком анализе возможны побочные эффекты. Если вы хотите проанализировать запрос, изменяющий данные, но при этом сохранить прежние данные таблицы, вы можете откатить транзакцию после запроса:

```
BEGIN;

EXPLAIN ANALYZE
 UPDATE bookings.aircrafts_data
    SET "range" = 1;

ROLLBACK;
```

EXPLAIN ANALYZE

- В ряде случаев EXPLAIN ANALYZE выводит дополнительную статистику по выполнению, включающую не только время выполнения узлов и число строк.
- К примеру, для узла Sort показывается использованный метод (Sort Method) и задействованный объём памяти (Memory):

Sort (cost=717.34..717.59 rows=101 width=488) (actual time=7.761..7.774 rows=100 loops=1)

Sort Key: t1.column_name **Sort Method:** quicksort **Memory:** 77kB

- Для узла Hash выводится число групп (Buckets) и пакетов хеша (Batches), а также максимальный объём, который заняла в памяти хеш-таблица (Memory Usage):

Hash (cost=229.20..229.20 rows=101 width=244) (actual time=0.659..0.659 rows=100 loops=1)

Buckets: 1024 **Batches:** 1 **Memory Usage:** 28kB

EXPLAIN ANALYZE

- Другая полезная дополнительная информация — число строк, удалённых условием фильтра (Rows Removed by Filter):

```
EXPLAIN ANALYZE
 SELECT *
   FROM bookings.aircrafts_data ad
  WHERE ad."range" < 5000
```

Seq Scan on aircrafts_data ad (cost=0.00..1.11 rows=3 width=52) (actual time=0.008..0.009 rows=4 loops=1)
  Filter: (range < 5000)
  Rows Removed by Filter: 5
Planning Time: 0.040 ms
Execution Time: 0.029 ms

EXPLAIN ANALYZE

- В некоторых планах запросов некоторый внутренний узел может выполняться неоднократно. Например, внутреннее сканирование индекса будет выполняться для каждой внешней строки во вложенном цикле верхнего уровня. В таких случаях значение loops (циклы) показывает, сколько всего раз выполнялся этот узел, а фактическое время и число строк вычисляется как среднее по всем итерациям.

Index Scan using tenk2_unique2 on tenk2 t2 (cost=0.29..7.91 rows=1 width=244)
(actual time=0.021..0.022 rows=1 **loops**=10)

Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования;
2. Parallel Seq Scan (параллельное последовательное сканирование) – используется несколько ядер процессора;
3. Index Scan (индексное сканирование) - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся;
4. Index Only Scan (исключительно индексное сканирование) – используется, когда индекс уже содержит все необходимые для запроса данные;
5. Bitmap Heap Scan (сканирование по битовой карте) - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования;
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора;
3. Index Scan (индексное сканирование) - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся;
4. Index Only Scan (исключительно индексное сканирование) – используется, когда индекс уже содержит все необходимые для запроса данные;
5. Bitmap Heap Scan (сканирование по битовой карте) - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования;
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора;
3. **Index Scan (индексное сканирование)** - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся;
4. Index Only Scan (исключительно индексное сканирование) – используется, когда индекс уже содержит все необходимые для запроса данные;
5. Bitmap Heap Scan (сканирование по битовой карте) - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

Виды операций сканирования

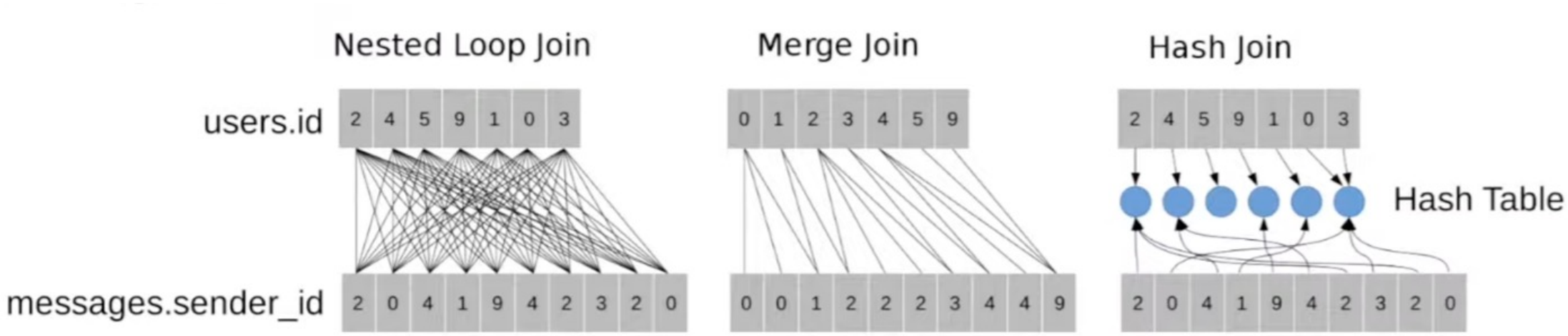
1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования;
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора;
3. **Index Scan (индексное сканирование)** - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся;
4. **Index Only Scan (исключительно индексное сканирование)** – используется, когда индекс уже содержит все необходимые для запроса данные;
5. Bitmap Heap Scan (сканирование по битовой карте) - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

Виды операций сканирования

1. **Seq Scan (последовательное сканирование)** – последовательно считывается каждая строка таблицы и проверяется на соответствие заданному условию. Наиболее медленный тип сканирования;
2. **Parallel Seq Scan (параллельное последовательное сканирование)** – используется несколько ядер процессора;
3. **Index Scan (индексное сканирование)** - возвращает значения TID (tuple id – идентификатор строки) по одному, до тех пор, пока подходящие строки не закончатся;
4. **Index Only Scan (исключительно индексное сканирование)** – используется, когда индекс уже содержит все необходимые для запроса данные;
5. **Bitmap Heap Scan (сканирование по битовой карте)** - сначала получает все подходящие под условие индексного поиска TID из индекса, затем по порядку запрашивает соответствующие блоки самой таблицы.

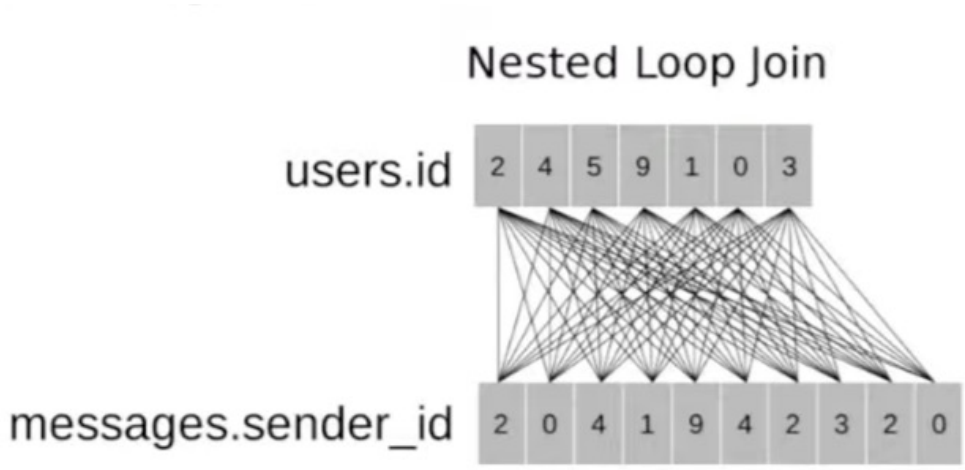
Виды операций объединения

- Nested Loop Join (соединение вложенных циклов);
- Merge Join (соединение слиянием);
- Hash Join (хэш-соединение).



Nested Loop Join

- сравнивает каждую строку первой таблицы с каждой строкой второй таблицы
- выводит строки, которые удовлетворяют условию объединения
- самый медленный тип объединения

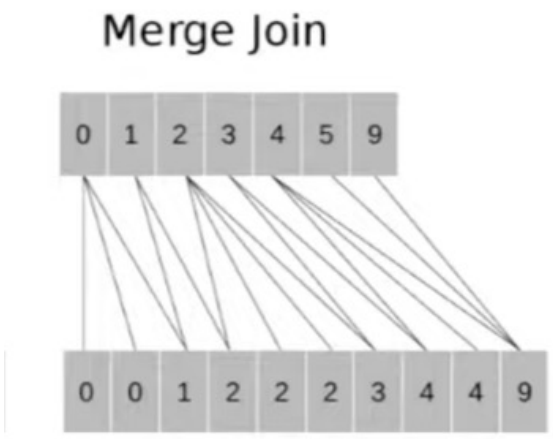


Псевдокод:

```
for each row R1 in the outer table
    for each row R2 in the inner table
        if R1 joins with R2
            return (R1, R2)
```

Merge Join

- используется, если объединяемые наборы данных отсортированы (или могут быть отсортированы с небольшими затратами) по ключам объединения
- работает на индексах, которые возвращают отсортированные данные

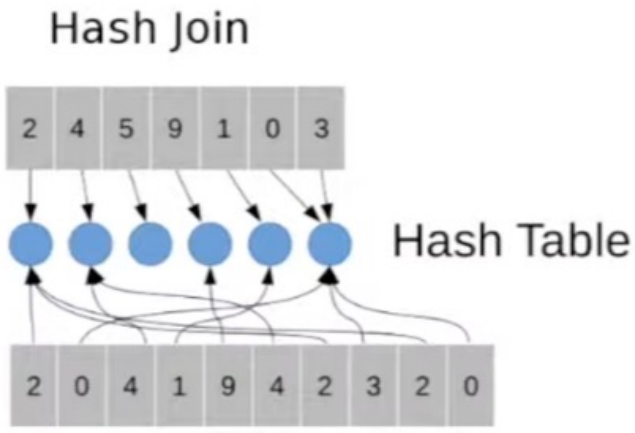


Псевдокод:

```
get first row R1 from   input 1
get first row R2 from   input 2
while not at the end   of either input
    begin
        if R1 joins with R2
            begin
                get next row R2 from input 2
                return (R1, R2)
            end
        else if R1 < R2
            get next row R1 from input 1
        else
            get next row R2 from input 2
    end
```

Hash Join

- хэш-соединение строит хэш-таблицу (словарь) на основе ключа соединения
- планировщик обычно решает выполнить хэш-соединение, когда таблицы имеют более или менее одинаковый размер, и хэш-таблица может разместиться целиком в памяти



Псевдокод:

```
for each row R1 in the build table
   begin
      calculate hash value on R1 join key(s)
      insert R1 into the appropriate hash bucket
   end
for each row R2 in the probe table
   begin
      calculate hash value on R2 join key(s)
      for each row R1 in the corresponding hash
bucket
         if R1 joins with R2
            return (R1, R2)
   end
```

Оптимизация запросов

Общие рекомендации

- Объединяйте данные по колонкам, на которых построен индекс;
- По возможности не объединяйте данные по колонкам с типом данных строка;
- При запросе данных используйте фильтрацию;
- Учитывайте индексы и ключи партиционирования при объединениях и фильтрации.

Не используйте UNION без необходимости удаления дублей

UNION ALL – объединяет два множества без дедубликации (работает быстро).

UNION – объединяет два множества и удаляет дублирующиеся строки (выполняет операцию DISTINCT);

Избегайте повторения кода

- использование CTE (Common Table Expression) - конструкция WITH;
- сохранение результатов подзапросов во временную таблицу (temporary table – существует только в рамках вашей сессии);

TRUNCATE быстрее DELETE

- При необходимости удаления всех данных из таблицы лучше использовать **TRUNCATE**, чем **DELETE;**
- **DELETE** физически удаляет данные;
- **TRUNCATE** физически в моменте не удаляет данные, а проставляет «флаг» удаления;

Unlogged Tables

- Данные при записи и обновлении не записываются в журнал логирования операций (wal-log);
- Существующую таблицу нельзя переформатировать в unlogged, необходимо указывать данный параметр при создании таблицы:

```
CREATE UNLOGGED TABLE table_name;
```

Практика