ПЛАН ЗАПРОСА .2

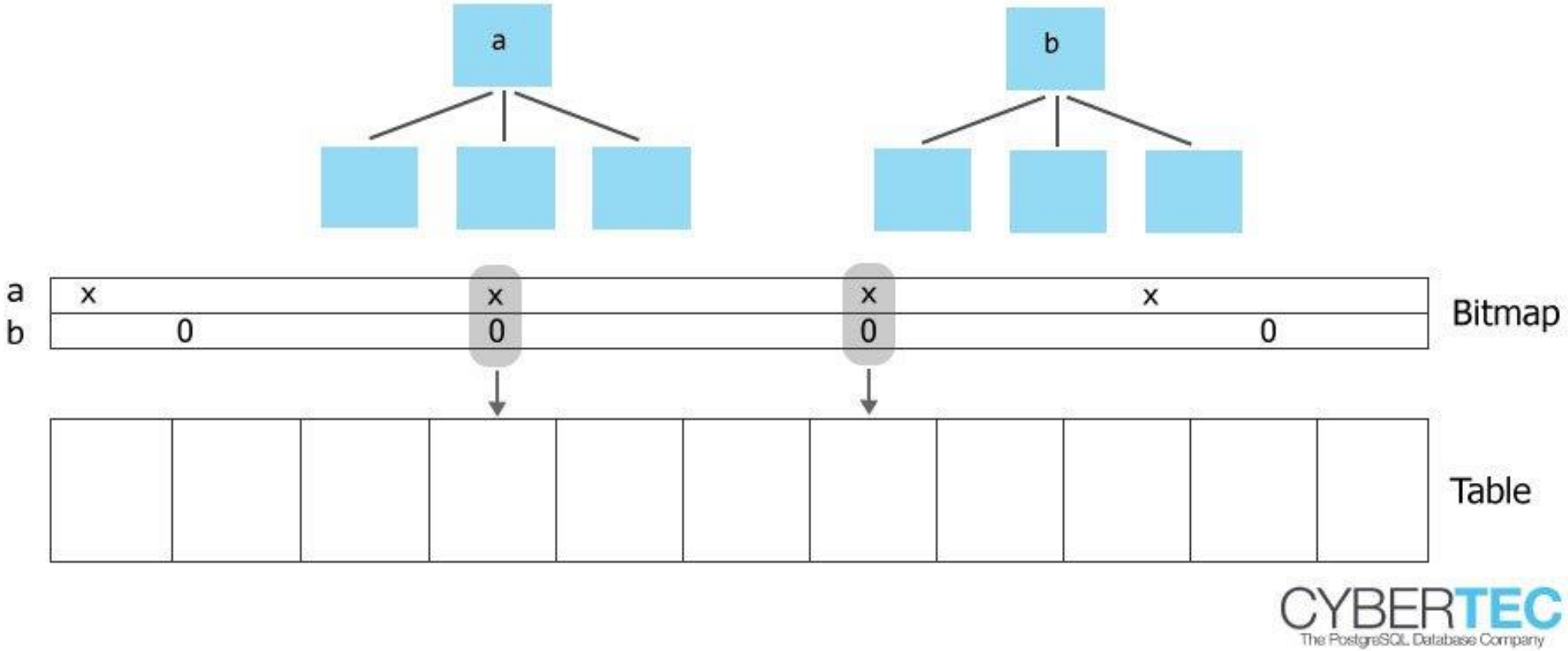
Лекция 11

Scan Types

- Sequential Scan
  - *Сканирует всю таблицу целиком*
  - *работает быстро на маленьких таблицах*
- Index Scan
  - *Сканирует строки в индексе, после ищет их на страницах*
  - *Быстрее, чем последовательное сканирование, если выбирается маленькое количество строк из большой таблицы*
- Index Only Scan
  - *Сканирует строки в индексе*
  - *Не нужно искать данные в таблице, потому что все данные уже хранятся в индексе*
- Bitmap Heap Scan
  - *Сканирует индекс, строит битовую карту страниц для посещения*
  - *Обходит только нужные страницы в таблице*

PostgreSQL Bitmap-scan

```
SELECT ... FROM  tab WHERE a=10 and b=20
```



CYBERTEC
The PostgreSQL Database Company

Join Types

- Nested Loops
  - *Для каждой строки во внешней таблице выбирается совпадающая строка во внутренней таблице (по сути последовательное сканирование)*
  - *Хорошо работает с маленькими таблицами*
- Merge Join
  - *Быстрое соединение отсортированных данных*
  - *Хорошо работает с большими таблицами*
  - *Стоимость увеличивается, если нужна начальная сортировка*
- Hash Join
  - *Строится хэш внутренней таблицы, внешняя таблица сканируется для совпадения значений*
  - *Используется только для условий равенства*
  - *Высока стоимость начала, но быстрое исполнение*

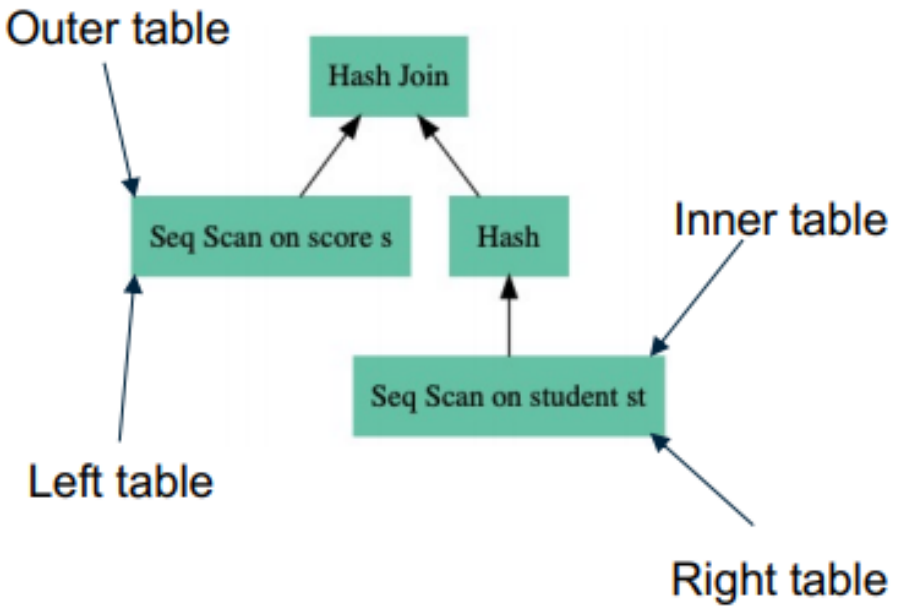
Hash Join

Фазы Hash Join:

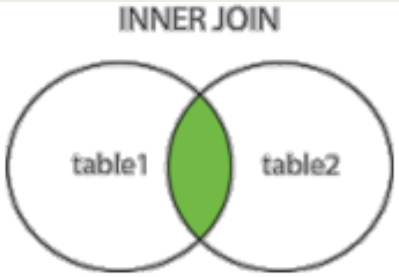
- Build phase (фаза построения): строится хэш-таблица по меньшей таблице после применения всех условий. Эта таблица называется внешней
- Probe phase - сканирование кортежей другой таблицы и попытка соединить таблицы по условию. Другая таблица называется "внешней"

SELECT name, subject, score FROM student st INNER JOIN score s ON st.id = s.stu_id

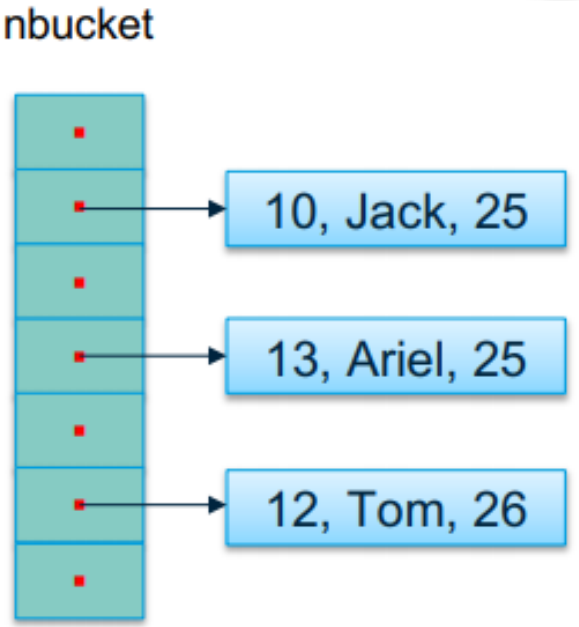
```
QUERY PLAN
-----------------------------------------------------
Hash Join  (cost=35.42..297.73 …)
   Hash Cond: (st.id = s.stu_id)
   ->  Seq Scan on student st  (cost=0.00..22.00)
   ->  Hash  (cost=21.30..21.30 rows=1130 width=8)
         ->  Seq Scan on score s  (cost=0.00..21.30)
(5 rows)
```



Inner join: build phase

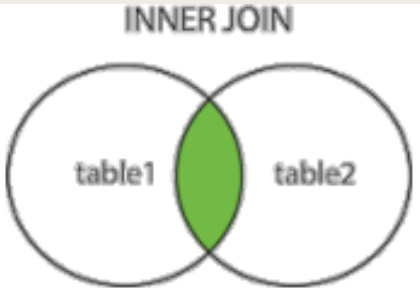



```
SELECT * FROM student;
 id | name   | age
----+--------+-----
 10 | Jack   |  25
 13 | Ariel  |  25
 12 | Tom    |  26
```



SELECT name, subject, score FROM student st INNER JOIN score s ON st.id = s.stu_id

Inner join: probe phase

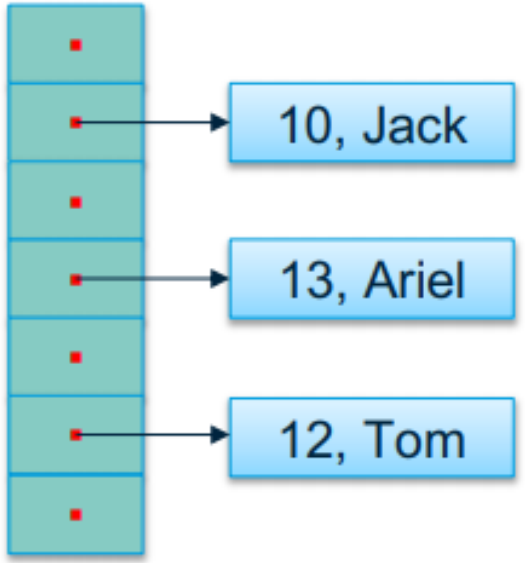


Score table

id	stu_id	subject	score
1	10	math	95
2	10	history	98
3	12	math	97
4	15	history	92

Jack	math	95
Jack	hist	98
Tom	math	97

Hash table for student




```
SELECT name, subject, score FROM student st INNER JOIN score s ON  st.id = s.stu_id
```

Визуализация

- Merge Join: http://sqlity.net/wp-content/uploads/2012/12/merge-join-algorithm.gif
- Hash Join: https://habr.com/ru/company/otus/blog/459314/

Полезные ссылки

- Описание всех операций
- https://www.pgmustard.com/docs/explain

Вложенность плана запроса

- Дан запрос:

```
EXPLAIN ANALYZE SELECT * FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2
ORDER BY t1.fivethous;
```

```
 QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=717.34..717.59 rows=101 width=488) (actual time=7.761..7.774 rows=100 loops=1)
   Sort Key: t1.fivethous
   Sort Method: quicksort  Memory: 77kB
   ->  Hash Join  (cost=230.47..713.98 rows=101 width=488) (actual time=0.711..7.427 rows=100 loops=1)
         Hash Cond: (t2.unique2 = t1.unique2)
         ->  Seq Scan on tenk2 t2  (cost=0.00..445.00 rows=10000 width=244) (actual time=0.007..2.583 rows=10000
loops=1)
         ->  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
               ->  Bitmap Heap Scan on tenk1 t1  (cost=5.07..229.20 rows=101 width=244) (actual time=0.080..0.526
rows=100 loops=1)
                     Recheck Cond: (unique1 < 100)
                     ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.04 rows=101 width=0) (actual
time=0.049..0.049 rows=100 loops=1)
                           Index Cond: (unique1 < 100)
 Planning time: 0.194 ms
 Execution time: 8.008 ms
```

Вложенность плана запроса

- Планировщик строит дерево узлов, символ -> указывает на каждый узел

```
Sort
└── Hash Join
    ├── Seq Scan
    └── Hash
        └── Bitmap Heap Scan
            └── Bitmap Index Scan
```

- Каждая ветвь – это под-действие, сначала выполняются самые вложенные действия, а после - внешние

Recheck Cond

```
->  Bitmap Heap Scan on tenk1 t1  (cost=5.07..229.20 rows=101 width=244) (actual time=0.080..0.526 rows=100 loops=1)
        Recheck Cond: (unique1 < 100)
        ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.04 rows=101 width=0) (actual time=0.049..0.049 rows=100 loops=1)
            Index Cond: (unique1 < 100)
```

- Битовая карта стала слишком большой, поэтому в ней могут быть потери, для этого происходит её пересчёт

Примеры!

- Для примеров используем учебную БД PostgreSQL

https://postgrespro.ru/education/demodb