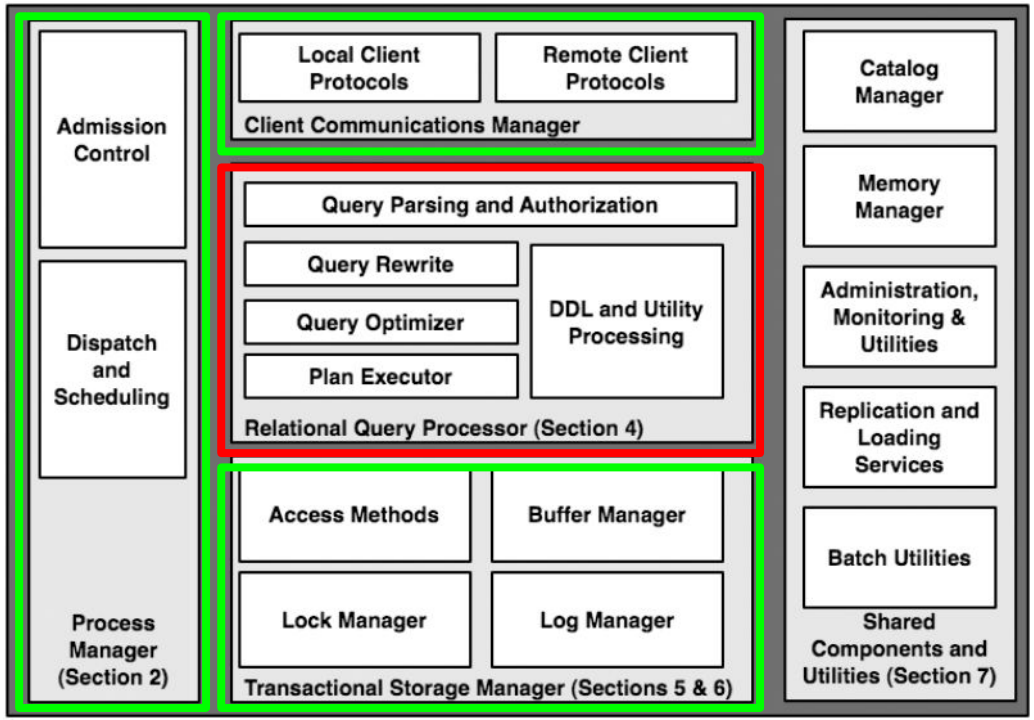
Технологии и разработка СУБД

Выполнение запросов

Анастасия Лубенникова
Александр Алексеев


Admission Control
Dispatch and Scheduling
Process Manager (Section 2)
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor (Section 4)
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager (Sections 5 & 6)
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities (Section 7)

План лекции

- Часть 1. Путь запроса
- Часть 2. Методы сканирования
- Часть 3. Методы соединения
- Часть 4. Настройки планировщика

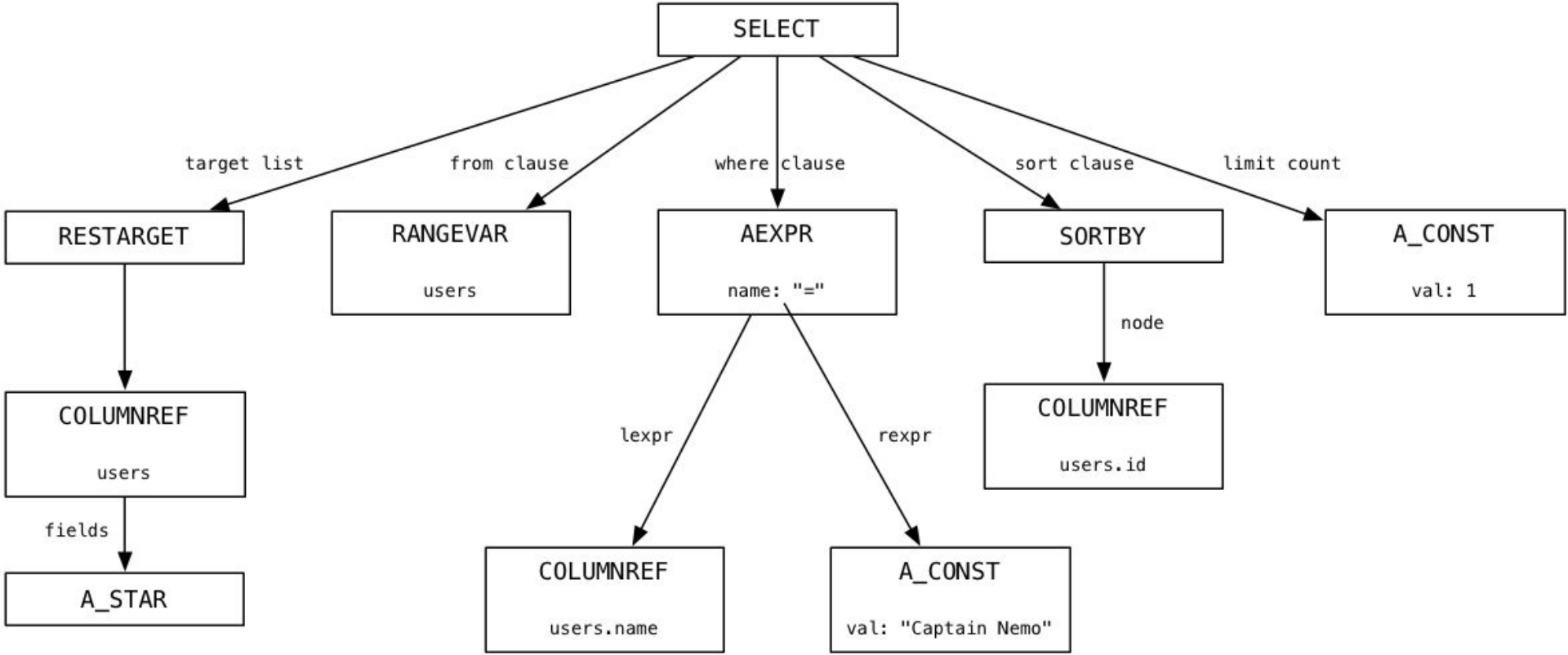
Простой запрос

```
SELECT * FROM users
WHERE name = 'Captain Nemo'
ORDER BY id ASC
LIMIT 1
```

Стадии обработки запроса

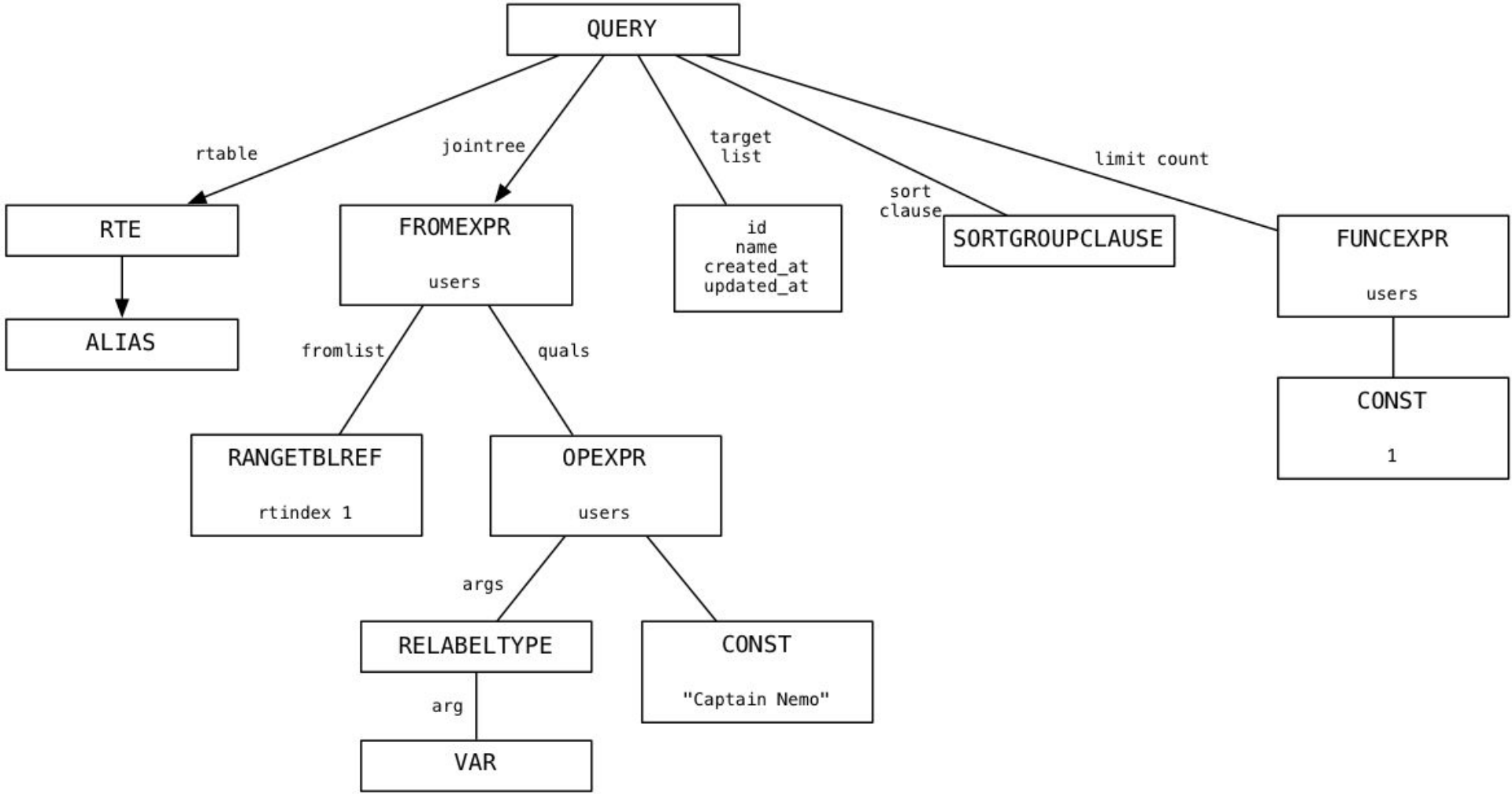
- Синтаксический анализ (Parser)
- Авторизация пользователя
  - проверка прав доступа к данным
- выполнение DDL запроса
  - Переписывание запроса (Rewriter)
  - DDL and Utility processor
- выполнение DML запроса
  - Переписывание запроса (Rewriter)
  - Планировщик (Planner)
  - Оптимизатор (Optimizer)
  - Исполнитель (Executor)

Дерево разбора (Parse Tree)



Система правил (RULE)

- Позволяет определить альтернативное действие, заменяющее операции добавления, изменения или удаления данных в таблицах базы данных.
- В отличие от триггеров, правило применяется только один раз на этапе разбора (как макрос).
- Например, при создании представления (VIEW) в таблице каталога pg_depend создается запись о правиле преобразования запроса. При запросе к этому VIEW, система правил выполняет данные преобразования.


QUERY
rtable
jointree
target list
sort clause
limit count
RTE
FROMEXPR
users
id
name
created_at
updated_at
SORTGROUPCLAUSE
FUNCEXPR
users
ALIAS
fromlist
quals
CONST
1
RANGETBLREF
rtindex 1
OPEXPR
users
args
RELABELTYPE
CONST
"Captain Nemo"
arg
VAR

Path

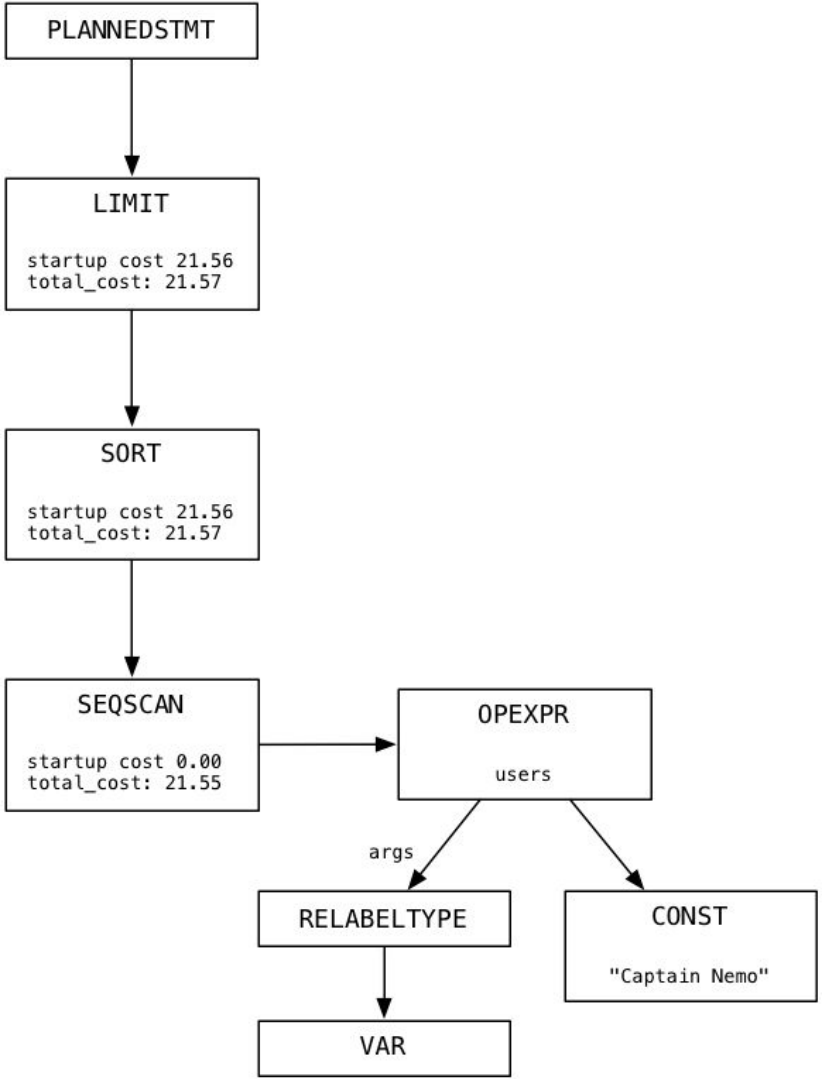
```
typedef struct Path
 {
     NodeTag     type;
     NodeTag     pathtype;       /* tag identifying scan/join method */
     RelOptInfo *parent;         /* the relation this path can build */
     PathTarget *pathtarget;     /* list of Vars/Exprs, cost, width */
     ParamPathInfo *param_info;  /* parameterization info, or NULL if none */

     /* estimated size/costs for path (see costsize.c for more info) */
     double      rows;           /* estimated number of result tuples */
     Cost        startup_cost;   /* cost expended before fetching any tuples */
     Cost        total_cost;     /* total cost (assuming all tuples fetched) */

     List       *pathkeys;       /* sort ordering of path's output */
     /* pathkeys is a List of PathKey nodes; see above */
 } Path;
```

Pathlist

- path #1
- path #2
- path #3

Лист сортируется по **total_cost**, также учитываются **startup_cost** и **rows.**

В итоге выбирается оптимальный путь и по нему строится детальный план запроса.


PLANNEDSTMT
LIMIT
startup cost 21.56
total_cost: 21.57
SORT
startup cost 21.56
total_cost: 21.57
SEQSCAN
startup cost 0.00
total_cost: 21.55
OPEXPR
users
args
RELABELTYPE
CONST
"Captain Nemo"
VAR

EXPLAIN

```
psql (9.3.3)
Type "help" for help.

pat=> EXPLAIN SELECT  "users".* FROM "users"  WHERE "users"."name" = 'Captain Nemo'
      ORDER BY "users"."id" ASC LIMIT 1;
                                QUERY PLAN
-------------------------------------------------------------------------
 Limit  (cost=21.56..21.57 rows=1 width=551)
   ->  Sort  (cost=21.56..21.57 rows=1 width=551)
         Sort Key: id
         ->  Seq Scan on users  (cost=0.00..21.55 rows=1 width=551)
               Filter: ((name)::text = 'Captain Nemo'::text)
(5 rows)
```

EXPLAIN (ANALYZE)

- команда EXPLAIN выводит план выполнения
- с указанием ANALYZE план действительно выполняется

Чем отличаются разные пути?

- Методы сканирования
- Методы соединения
- Порядок соединения

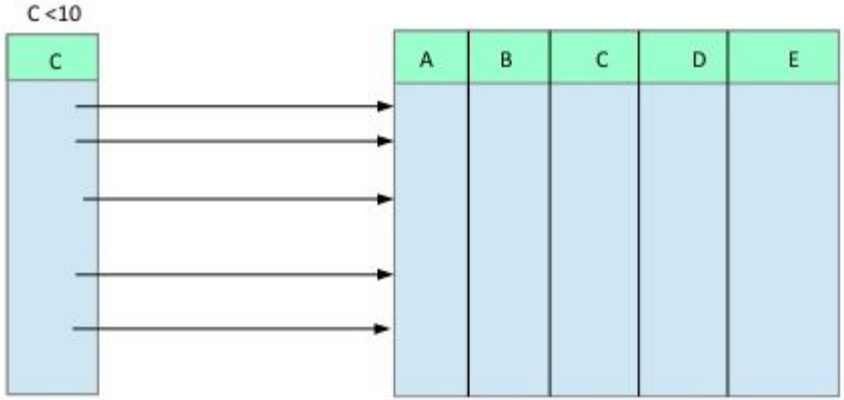
Методы сканирования

- Sequential Scan
- Index Scan
- Index Only Scan
- Bitmap Index Scan

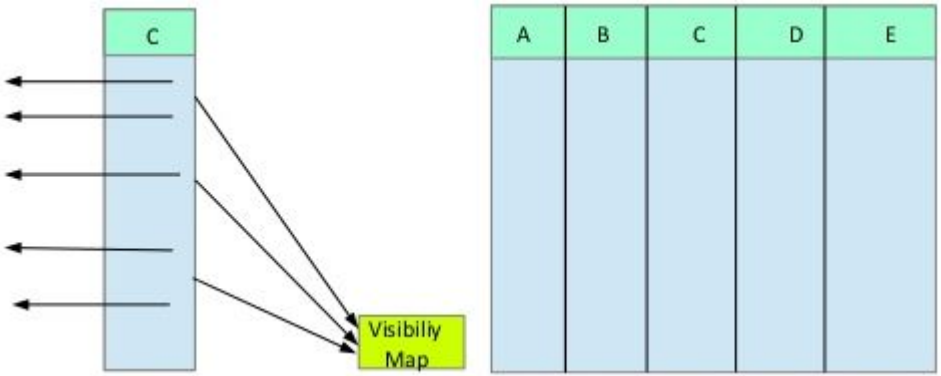
Sequential Scan

- последовательное сканирование таблицы

Index Scan

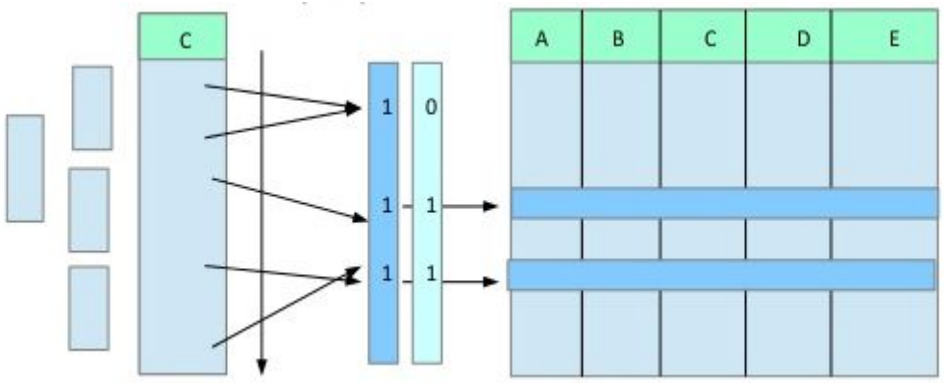


Index Only Scan



- если все данные, требуемые в запросе есть в индексе и отмечены как видимые всем транзакциям, можно избежать обращения к таблице

Bitmap Index Scan

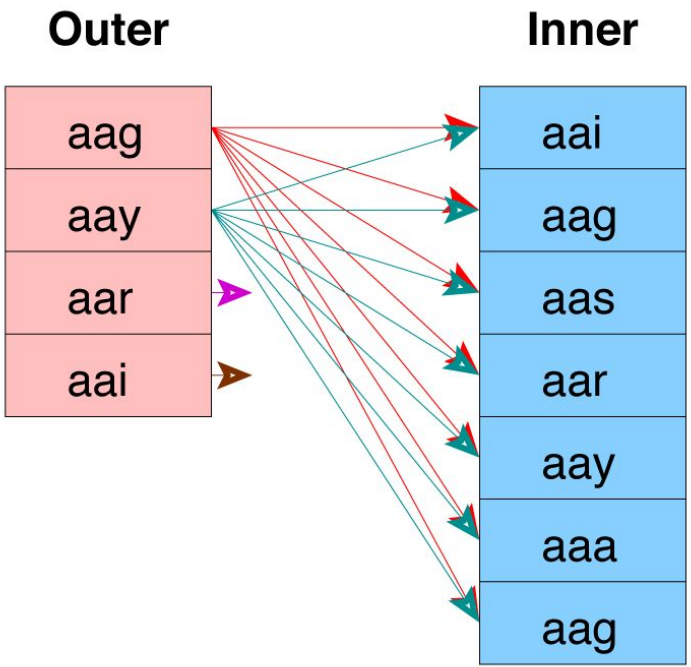


- Для определения подходящих записей выполняется поиск по индексу.
- Для выбранных кортежей составляется битовая карта. Затем выполняется Bitmap Heap Scan.
- Используется,если в запросе присутствует несколько условий. Перед поиском по таблице производится совмещение битовых карт для каждого из ключей поиска. За счет этого уменьшается количество обращений к таблице.

Методы соединения

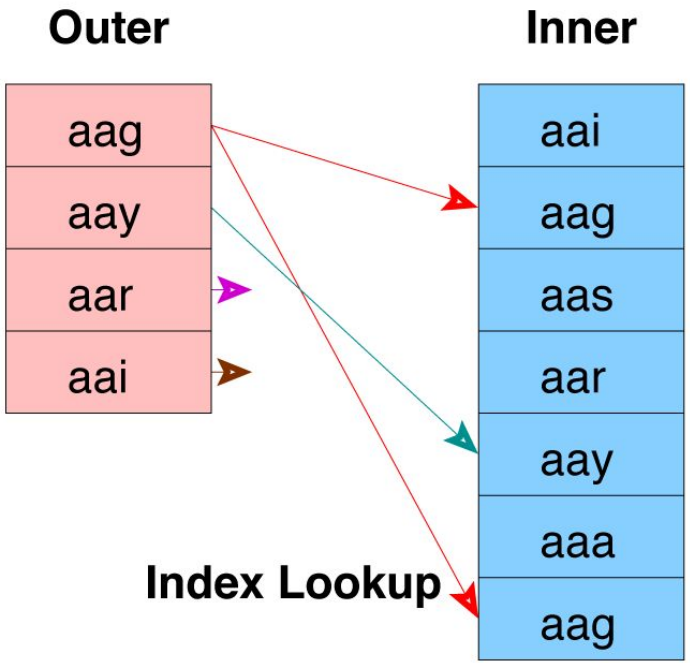
- Nested Loop
  - Inner Sequential Scan
  - Inner Index Scan
- Hash Join
- Merge Join

Nested Loop with Seq Scan



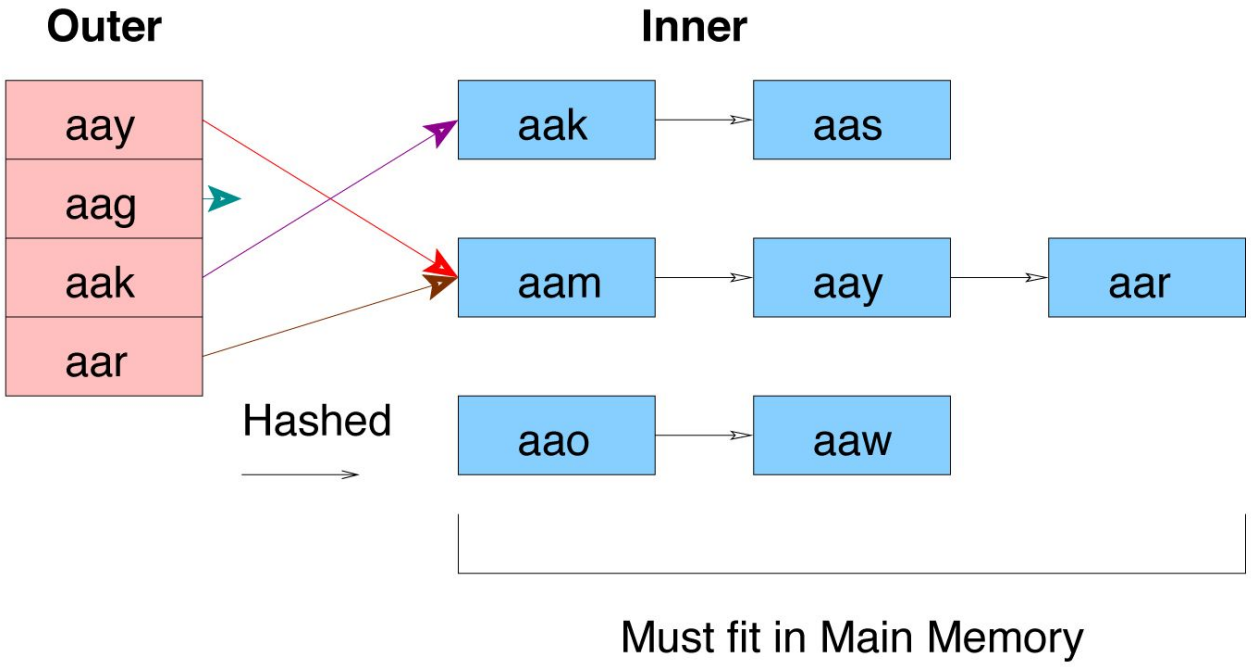
- не требуется предварительная подготовка
- используется для маленьких таблиц

Nested Loop with Index Scan

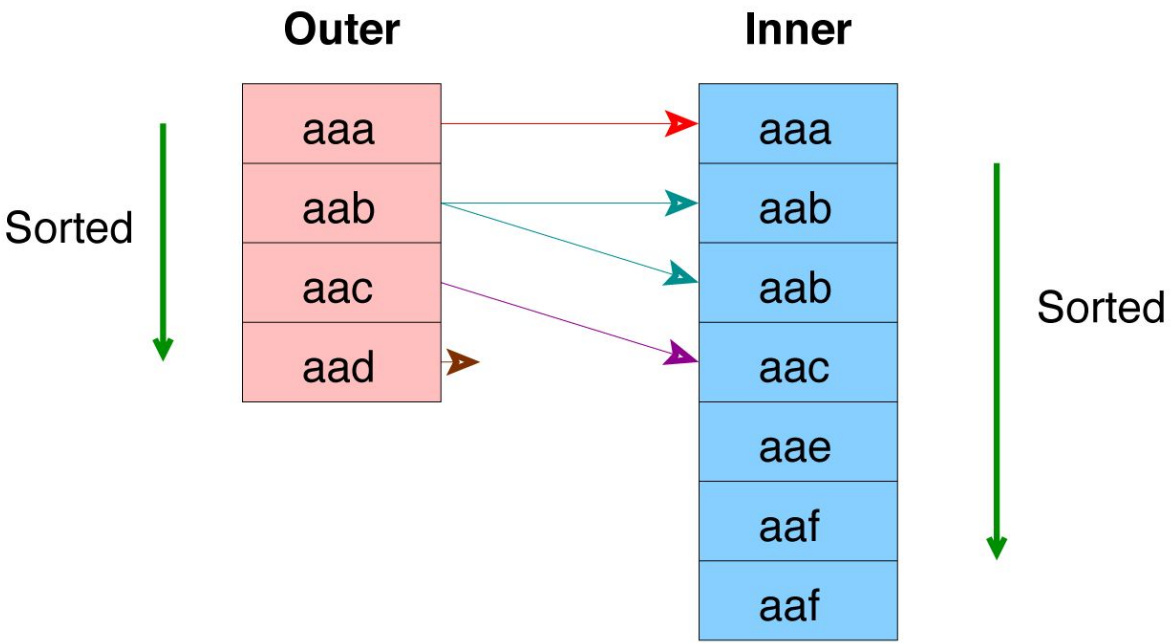


- не требуется предварительная подготовка
- соответствующий индекс должен существовать

Hash Join


Outer
Inner
aay
aag
aak
aar
aak
aas
aam
aay
aar
aao
aaw
Hashed
Must fit in Main Memory

Merge Join



- используется для больших таблиц
- можно использовать индекс, чтобы не делать сортировку

Порядок соединения

- Для небольшого числа таблиц - полный перебор
- Для большого - Genetic Query Optimization
  https://postgrespro.ru/docs/postgresql/10/geqo.html

Настройки планировщика (1)

- enable_seqscan
- enable_indexscan
- enable_indexonlyscan
- enable_bitmapscan

- enable_nestloop
- enable_hashjoin
- enable_mergejoin

Настройки планировщика (2)

- seq_page_cost
- random_page_cost

Дополнительные материалы

- Обзор обработки запроса
  https://postgrespro.ru/docs/postgresql/10/overview.html
- Путешествие запроса Select через внутренности Постгреса
  https://habrahabr.ru/post/304258/
- Настройки планирования запросов
  https://postgrespro.ru/docs/postgresql/10/runtime-config-query.html
- http://etutorials.org/SQL/Postgresql/Part+I+General+PostgreSQL+Use/Chapter+4.+Performance/Understanding+How+PostgreSQL+Executes+a+Query/
- Explaining the Postgres Query Optimizer
  https://momjian.us/main/writings/pgsql/optimizer.pdf

Вопросы и ответы.

- a.lubennikova@postgrespro.ru
- a.alekseev@postgrespro.ru
- Telegram: https://t.me/dbmsdev