Towards more efficient query plans

PostgreSQL 11 and beyond

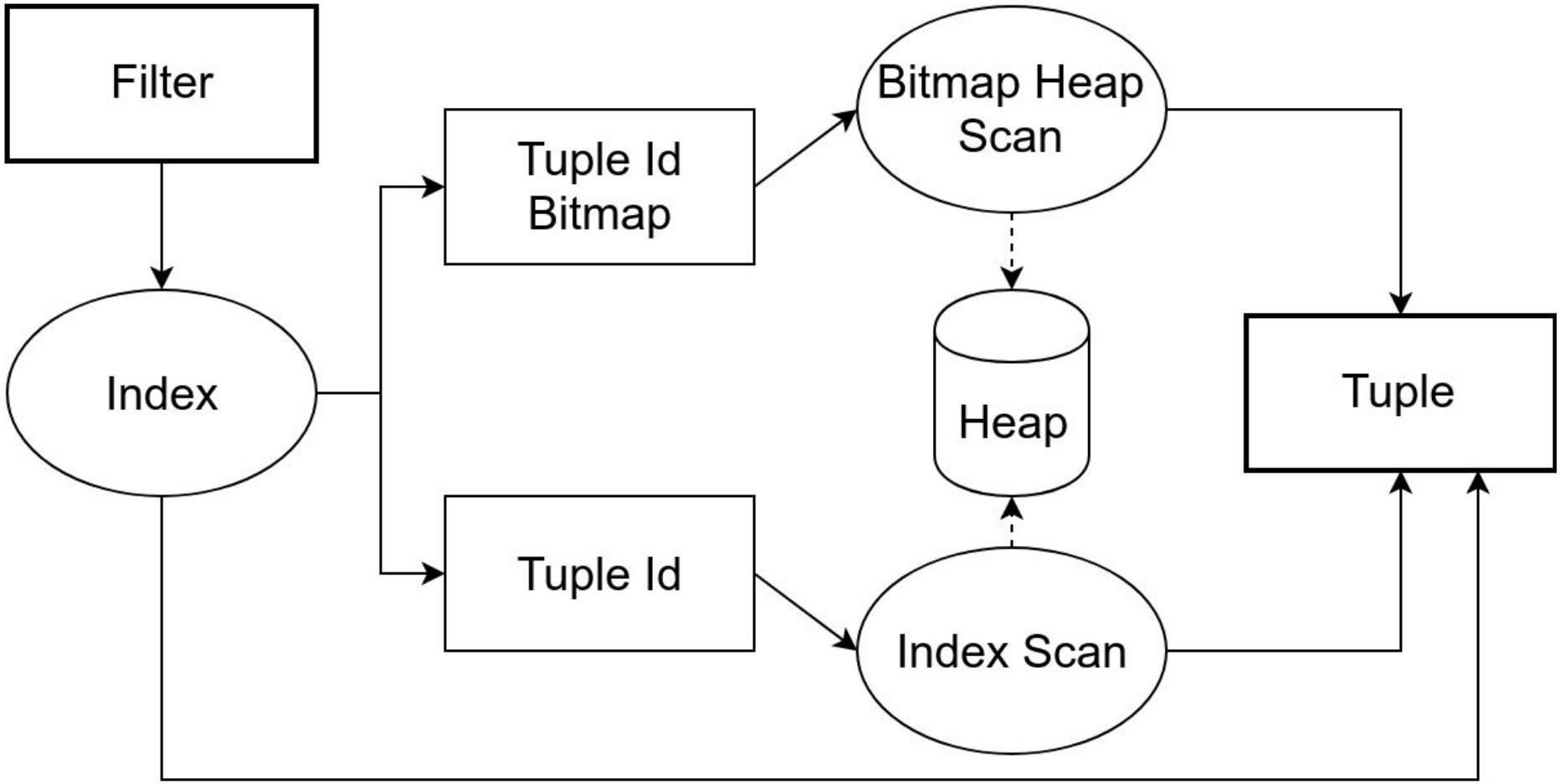
Alexander Kuzmenkov
a.kuzmenkov@postgrespro.ru

What is a plan?

- SQL is a declarative language: “what”, not “how”
- Optimizer decides how to execute queries based on statistics about data and available resources

- A plan is a tree of simple building blocks
  - Scan
    - Table
    - Index
    - Function
    - Subquery
  - Join
    - Merge
    - Nested Loop
    - Hash
  - Sort/Group/Unique
  - etc.

Index scan


Filter
Index
Tuple Id Bitmap
Tuple Id
Bitmap Heap Scan
Heap
Index Scan
Tuple

Covering B-tree indexes

- Index-only scan can return INCLUDEd columns, but these columns:
  - do not participate in UNIQUE constraint
  - do not require btree operators (e.g. point type)
- The development is ongoing for GiST

```
Table "public.pglist"
 Column   | Type
----------+-----------
id        | integer
sent      | timestamp
subject   | text
author    | text
body_plain| text
fts       | tsvector
```

```
# create unique index on pglist(id) include (subject);
# select subject from t where id < 200000;
  -- emulate a join on id that selects 1/5 of the rows
```

Index	**Plan**	**Time, ms**
unique on pglist(id)	Index Scan	150
unique on pglist(id) include (subject)	Index Only Scan	50

Index-only Bitmap Scan for count(*)

- for indexes that do not support index-only scan (e.g. GIN)
- don't fetch the tuples when we only need to count them
- fast and precise pagination without the EXPLAIN trick
- needs adequate work_mem to fit the bitmap
- works only on vacuumed pages

```
# create index on pglist using gin(fts);
# select count(*) from pglist
   where fts @@ to_tsquery('rebase');
```

Conditions	Pages read	Time, ms
not vacuumed	95k	160
vacuumed	50	90

Bitmap	
Tuple ID	☑
Page 1 Tuple 1	1
Page 1 Tuple 2	0
....	
Page N Tuple M	1
....	

Loose index scan

DEV

- Fast DISTINCT using a btree index
- Now done with Unique over sorted input

```
-- table t(a int), 100k ints [0, 500)
# create index on t(a);
# select distinct a from t;
```

Plan	**Time, ms**
Loose index scan	6
Unique over Index scan	97
Unique over Sort	160

k-nearest neighbors for B-tree indexes

DEV

- Use case: find some events closest to the given time
- Sort by distance operator inside the index
- Can use btree_gist, but it's generally slower

```
# select sent from pglist
    order by sent <-> '2010-03-05'::timestamp limit 1000;

->  Index Only Scan using sent_btree on public.pglist
       Output: sent, (sent <-> '2010-03-05')
       Order By: (pglist.sent <-> '2010-03-05')
```

Index Type	vanilla btree	btree_gist	kNN-btree
Time, ms	550	2.6	1.8

Incremental sort

DEV

- Sort partially sorted input
- Reuse one index for similar ORDER BY queries or joins
- Read less rows when LIMIT is specified
- Use less memory for sorting

Who needs sorted output?

- ORDER BY
- DISTINCT
- GROUP BY
- window functions
- merge joins

```
# create index on pglist(subject);
# select distinct on (subject) subject, sent from pglist
    order by subject, sent desc;
    -- get the newest message date for each topic
```

Plan	Sort details	Time, s
Incremental Sort over Index Scan	quicksort, 2 MB memory	5.7
Sort over Seq Scan	external merge, 1.2 GB disk	22.5

with LIMIT 100	Time, ms
Incremental Sort over Index Scan	5
top-N heapsort over Seq Scan	1000

Estimate sort costs for GROUP BY

DEV

- Make sort cost accord for cardinality and order of columns
- Choose cheapest sort order for GROUP BY
- Example
  - “p” — high cardinality, cheap to compare
  - “v” — low cardinality, expensive to compare

Sort keys	**Sort time, ms**
p, v	800
v, p	1500

```
# select i/2 as p, format('''%60s''', i%2) as v into t
    from generate_series(1, 1000000) i;
# select count(*) from t group by p, v;
```

```
 p |                                                              v
---+-------------------------------------------------------------------------------
 1 |                                                             '                0'
 1 |                                                             '                1'
 2 |                                                             '                0'
 2 |                                                             '                1'
 3 |                                                             '                0'
 3 |                                                             '                1'
```

Joins

Join types

- Inner
- Outer
- Semi/Anti

Optimizations

- Transitive equality
- Join strength reduction
- Join removal

How to choose the order of joins?

- System R
  - Finds the best join for 2 tables
  - Combines the best joins it found for N-1 tables to find the best ones for N
  - Too many combinations to try. Only used when N < geqo_threshold

- Genetic algorithm
  - Used when N >= geqo_threshold
  - A heuristic algorithm that doesn't try all the permutation

Multicolumn join selectivity

DEV

- Poor selectivity estimates for multicolumn join on correlated columns
- CREATE STATISTICS (dependencies) not helpful for joins
- Solution: create single–column statistics on composite values
- Do it automatically — there is probably and index on these columns

```
-- table t(a int, b int), a = b, a in [0..10k), 1M rows
# select * from t join t tt using (a, b);
```

Real number of join rows	**Normal stats**	**Multicolumn index stats**
10M	100 (4 orders off!)	9.97M

Joins with a unique inner side

- On the inner side, at most one row matches the join clauses
- Proved by unique index for table or GROUP BY for subquery

Semi join

- WHERE EXISTS
- Like Inner, but:
  - No inner columns
  - Skips duplicates
- Reduced to inner join when the inner side is unique [10]

Skip materialization in merge joins

- Each inner tuple only used once => don't have to materialize the inner side [10]

Self join on primary key

DEV

- Frequent in ORM-generated queries
- Also happens when reusing complex views
- Can be replaced with a scan with combined filters

```
# create view v1 as select * from pglist
   where subject like 'P%';
# create view v2 as select * from pglist
   where sent between '2010-01-01' and '2010-12-31';
# select * from v1 where exists
   (select * from v2 where id = v1.id);
```

Baseline	Join on `id` between `v1` and `v2`
With self-join removal	Scan on `pglist` where `subject like 'P%' and sent between ...`

Outer join

- Output all outer rows, nulls for inner rows when none match
- Less freedom for planning

- Can be reduced to inner join
  - when it follows from WHERE clause that some inner column is not null [before 10]
- Can be removed
  - Inner side is not used and is unique [before 10]
  - Inner side is not used and the result is made unique by GROUP BY or DISTINCT [DEV]

```
# create table parentmsg (id int primary key, parent int);


# select * from pglist left join parentmsg using (id)
    where parent = 42;
# select pglist.* from node left join parentmsg using (id);
# select distinct on (pglist.id) from pglist
    left join parentmsg on pglist.id = parent;
```

Merge join on range overlap

DEV

- Normally performed with Nested Loop
- Order ranges by comparison operator
- Perform Merge Join on range overlap (&&)

```
-- tables s, r(ir int4range) with
   r.ir = (g, g+10),
   s.ir = (g+5, g+15),
   g = 1..100k;

-- gist(ir) on s and r;
```

Plan	Time, s
Nested Loop over Seq Scan and Index Only Scan	15.7
Merge Join over Sort	4.3
Merge Join over btree Index Scan	2.8

```
# select * from s join r on s.ir && r.ir;
```

Inlining Common Table Expressions

DEV

- Can lead to better plans
  - Statistics
  - Predicate pushdown

`WITH t AS [MATERIALIZE {ON|OFF}] (…)`	
Option	**Inline?**
no option	If only one reference
ON	Never
OFF	If no side effects or RECURSIVE

```
# create index on pglist
(subject text_pattern_ops);

# with c as (
  select subject, count(*) n
  from pglist group by subject
)
select * from c
where subject like 'P%'
order by n limit 10;
```

Materialize	ON	OFF
Est. rows (actually 15k)	400	17k
Time, ms	800	30
Plan for ‘c’	Seq scan	Index scan

Precalculate stable and immutable functions

DEV

1. Cache stable functions in expressions at execution time

```
# select count(*) from messages
    where fts @@ to_tsquery('postgres');
```

- Calculate `to_tsquery` only once in Recheck step of Bitmap Heap Scan
- 1.5 s precalculated / 2.3 s baseline

2. Inline immutable functions in FROM list at planning time

```
# select count(*) from messages m,
    to_tsquery('english', 'postgres') qq where m.fts @@ qq;
```

- Bitmap Heap scan instead of Nested Loop over Function Scan + Bitmap Heap scan
- No join => faster planning, better cost estimates

Support the development

- Review the patches you need
- No need to know Postgres internals or C programming
- Read “Reviewing a Patch” at the wiki
- Usability review
  - Is the feature actually implemented?
  - Do we want it?
  - Are there dangers?
- Feature test
  - Does it work as advertised?
  - Are there any corner cases?
- Performance review
  - Are there any slowdowns?
  - If the patch claims to improve the performance, does it?

Thank you!

Alexander Kuzmenkov
a.kuzmenkov@postgrespro.ru

References

Loose index scan
https://commitfest.postgresql.org/21/1741/

k-nearest neighbors for B-tree indexes
https://commitfest.postgresql.org/21/1804/

Incremental sort
https://commitfest.postgresql.org/20/1124/

Estimate sort costs for GROUP BY
https://commitfest.postgresql.org/20/1706

Multicolumn join selectivity
https://www.postgresql.org/message-id/flat/3fcfd5e5-6849-34e6-22ab-1b62d191bedb%402ndquadrant.com#d61504c511d4b437505a05fa50047019

Self join on primary key
https://commitfest.postgresql.org/20/1712/

Unique outer join with GROUP BY
https://www.postgresql.org/message-id/flat/CAKJS1f96XNrS68NZy9s=Xkq+RAj6RE5CrCvDcy_uB-V=U4+YRw@mail.gmail.com

Merge join on range overlap
https://commitfest.postgresql.org/17/1449/

Inlining Common Table Expressions
https://commitfest.postgresql.org/21/1734/

Precalculate stable and immutable functions
1. https://commitfest.postgresql.org/20/1648/
2. https://commitfest.postgresql.org/19/1664/