Multi-Entry Indexing in GiST & SP-GiST

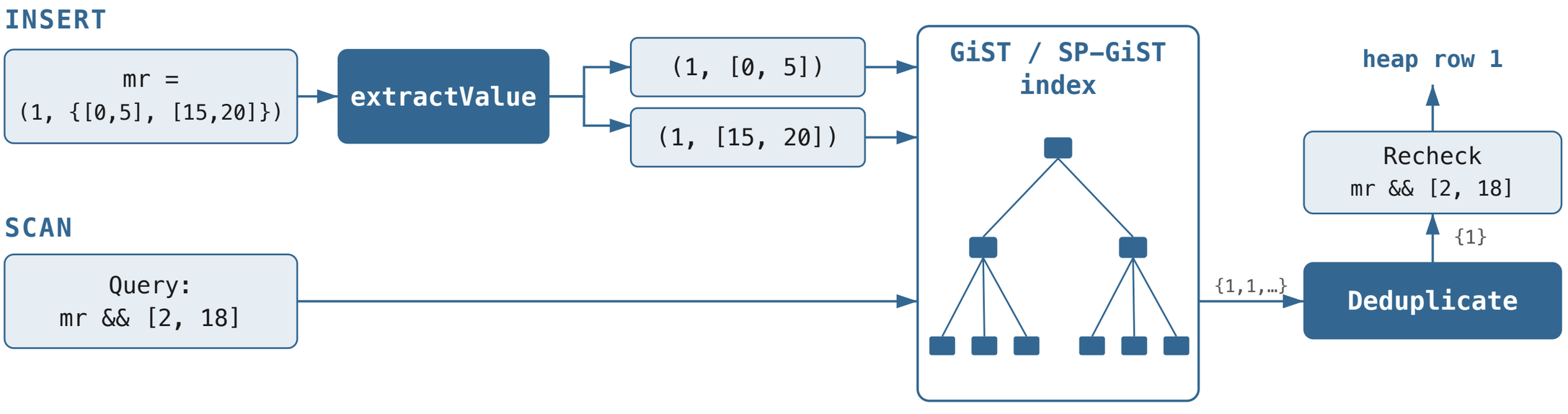
Maxime Schoemans
maxime.schoemans@enterprisedb.com
EnterpriseDB

pgsql-hackers thread

Problem

PostgreSQL indexes **multirange** values in GiST and SP-GiST by their *bounding union*, a single range covering every component. A query like mr && int4range(5,10) must visit every heap tuple whose bounding range overlaps [5,10), even when none of the multirange's actual components do.

GIN sidesteps this with extractValue: one heap tuple yields many index entries. This patch series brings the same idea into core GiST and SP-GiST, so opclasses can opt in without a new AM, and scans deduplicate TIDs on the fly. The first opt-in user is the new multirange_me_ops opclass.



How it works

Design

- New optional support proc extractValue (GiST proc 13, SP-GiST proc 8). One heap tuple becomes N index tuples, all with the same TID.
- Signature mirrors GIN:

```
Datum *extractValue(Datum value,
                    int32 *nentries,
                    bool **nullFlags)
```

- Scans deduplicate TIDs via a simplehash, so each heap tuple is returned at most once.
- Opt-in: existing opclasses are unaffected.

multirange_me_ops

```
CREATE INDEX ON bench_mr
  USING gist   (mr multirange_me_ops);
CREATE INDEX ON bench_mr
  USING spgist (mr multirange_me_ops);
```

- Each component range stored separately.
- Exact per-component: OVERLAPS, CONTAINS_ELEM.
- Empty multirange stored as an empty-range sentinel so it stays visible to operator queries.
- Non-default; sits alongside multirange_ops.

Benchmark

100k multiranges with wide gaps:

```
CREATE TABLE bench_mr (mr int4multirange);
INSERT INTO bench_mr
  SELECT int4multirange(
           int4range(g, g+10),
           int4range(g+100000, g+100010))
  FROM generate_series(1, 100000) g;
```

Query: mr @> 100000. Matches 10 rows (the value lies in the gap for nearly every row).

```
SELECT * FROM bench_mr WHERE mr @> 100000;
```

Results

Method	Time	Buf	Recheck
Seq scan	7.732 ms	834	–
GiST multirange_ops	9.504 ms	2 311	99 990
GiST multirange_me_ops	**0.056 ms**	**6**	**0**
SP-GiST multirange_me_ops	**0.112 ms**	**27**	**0**

Takeaway

Multi-entry indexes each component separately, so the bounding-union false positives disappear: 100×+ faster with zero rechecks.

PGConf.dev