PgCuckoo

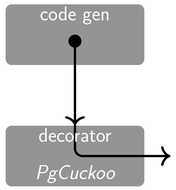
Laying Plan Eggs in PostgreSQL's Nest

Denis Hirn, Torsten Grust

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

1 Generating Code Externally

Numerous data-centric languages are compiled into intermediate SQL text that is then fed into PostgreSQL. The resulting queries are often large or non-idiomatic and fail to fully exploit front-end language semantics.



Instead, algebraic code in the form of skeleton plan trees allows us to:

- gain full control over the plan shape,
- assemble operator constellations not derivable from SQL,
- use PostgreSQL's executor, table storage, and index support,
- focus on the font-end language development.

PgCuckoo's *plan decorator* completes the skeleton plan trees through **rule-based property inference**.

```
SELECT c.c_name, o.o_orderkey, o.o_orderdate,
       abs(o.o_totalprice -
           SUM(l.l_extendedprice *
               (1 - l.l_discount) * (1 + l.l_tax))) AS deviation
FROM   orders AS o, lineitem AS l, customer AS c
WHERE  o.o_orderkey = l.l_orderkey
AND    o.o_custkey  = c.c_custkey
AND    o.o_orderdate > date '1998-01-01'
GROUP BY o.o_orderkey, o.o_totalprice, o.o_orderdate, c.c_name
HAVING SUM(l.l_extendedprice *
           (1 - l.l_discount) * (1 + l.l_tax)) <> o.o_totalprice
ORDER BY c.c_name, o.o_orderkey;
```

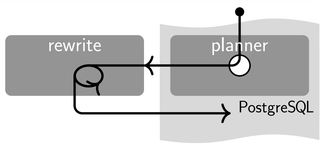
Overall/Inferred Plan Properties	Query Plan	Seed Properties
SSSSS	PLANNEDSTMT	
BBBBBBBBBBBBBBBRRRRMMMMMMMM	└GROUP AGGREGATE	BBBRRRMM
BBBBBBBBBBBBBBBRRRRMMMMMMMM	└SORT	BBBBRRRMMMM
BBBBBBBBBBBBBBBMM	└HASH JOIN	BBBBMM
BBBBBBBBBBBBBBBMM	├HASH JOIN	BBBMM
BBBBBBBBBM	├SEQ SCAN	BBM
	│└lineitem	
BBBBBMM	└HASH	BBM
BBBBBR	└SEQ SCAN	BM
	└orders	
BBMM	├HASH	BBM
BM	└SEQ SCAN	BM
	└customer	

B expression (×30) R intra-plan reference
M meta data, misc. flags S schema information (×10)

Required plan tree properties overall (left), seed from which *PgCuckoo* can infer the rest (right).

3 Improving Original Plan Trees

PgCuckoo is a framework in which the **experimentation with advanced plan generation approaches** may be performed *outside* the database kernel. Thus,



- a SQL query Q is submitted to PostgreSQL,
- the *planner hook* is used to receive Q's initial plan tree,
- an external plan rewriter performs its task, before
- the rewritten plan is injected back into PostgreSQL for regular execution.

We obtain an experimental version of the system which explores portions of the plan tree space that an off-the-shelf PostgreSQL would not consider to enter on its own. Examples:

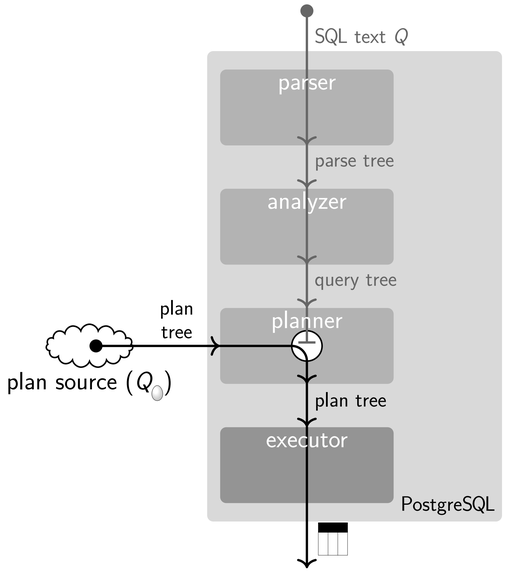
- Implement the query unnesting strategy employed by the *HyPer* DBMS
- Plan trees may be abstracted and understood as *logical* relational algebra

External Plan Sources for PostgreSQL

PgCuckoo. We build on on the so-called *planner hook* to significantly alter PostgreSQL's operation: we use the hook to inject query *plan trees from outside the system* and have these foreign plans be executed by the system's query executor – like a cuckoo lays an egg in a victim bird's nest. Several opportunities arise:

1. Given an **external code generator** for a foreign (maybe even non-relational) query or data processing language, we may count on PostgreSQL as a runtime and execution back-end for that language.
2. We can **stitch together several plan pieces** to fully control the evaluation of subqueries (or query parts, in general) in a fine-grained fashion.
3. We may **improve original plan trees** through rewriting strategies—expressed on the surface query level as well as on plan trees themselves—that are not present in PostgreSQL itself.
4. We may quickly **retrieve canned "plan favorites"** based on the original SQL query text and other system or environment parameters, foregoing costly (and sometimes unpredictable) planning from scratch.

Closing Down PostgreSQL's Query Front End

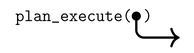


PostgreSQL invokes its *planner hook* just before query planning begins. The called user code receives a representation of a parsed SQL query Q and is expected to return a *plan tree* for Q.

- Plan trees are self-contained and carry all information needed for execution.
- We return a plan for a query Q_0 of our choosing.
- PostgreSQL's executor will evaluate Q_0 and return its tabular result.

We thus "short-circuit" the standard planner and effectively close down PostgreSQL's query front end—the executor does not depend on it (symbolized by).

2 Stitching Together Individual Plan Pieces



For experimentation, benchmarking, but also in educational settings it is valuable to be able to **exercise precise control over the plan** that the RDBMS generates for a SQL query.

- Vanilla PostgreSQL provides few levers we can pull to influence plan generation.
- Switches like `set enable_hashjoin = on/off` govern physical operator choice for the *entire* plan tree.

With *PgCuckoo*, we are able to control plan generation at the granularity of individual expressions or subqueries:

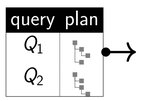
- PostgreSQL generates the global shape of the plan.
- `plan_execute(·)` is a table-valued SQL function whose argument is a plan piece. This allows to control plan details locally.
- Stitch together several plan pieces to form a complete plan.

```
SELECT bills.c_name, bills.o_orderkey, bills.o_orderdate,
       abs(bills.o_totalprice - total.price) AS deviation
FROM   plan_execute('{PLANNEDSTMT A}') AS total,
       plan_execute('{PLANNEDSTMT B}') AS bills,
WHERE  total.l_orderkey = bills.o_orderkey
AND    bills.o_totalprice <> total.price
AND    bills.o_orderdate > date '1998-01-01' -- C
ORDER BY bills.c_name, bills.o_orderkey;
```

```
PLANNEDSTMT
└SORT
 └HASH JOIN
  ├ FINALIZE HASH AGGREGATE  A   } locally exploit
  │ └GATHER                      } parallelism
  │  └PARTIAL HASH AGGREGATE     }
  │   └PARALLEL SEQ SCAN         }
  │    └lineitem                 }
  └HASH
   └FILTER  C  ····· PostgreSQL-placed predicate
    └ MERGE JOIN  B   } locally switch to
     ├INDEX SCAN      } sorted/merge-based processing
     │ └customer      }
     └SORT            }
      └SEQ SCAN       }
       └orders        }
```

Alter the plan of the original query to enforce the use of parallelism in plan piece A and to request sort-based (as opposed to hash-based) processing in piece B.

4 Saving and Retrieving Canned Plan Favorites



PgCuckoo supports the ingestion of externally generated plans but also provides a **foundation for the construction of *plan caches*** or *Query Stores* as in MS SQL Server. Such caches may save repeated plan generation effort and provide predictable performance for queries in a workload:

- Associate a query ID (or hash) Q_i with the best or designated plan tree.
- Retrieve the plan tree of Q_i from the cache and pass it to the executor.
- Multiple entries per Q_i may be present to record the query's history of plans.
- The cache can be augmented with plan trees *not* hatched up by PostgreSQL.
- All building blocks are in place for a learning-based plan generator for PostgreSQL.

SIGMOD/PODS 2019 AMSTERDAM

Visit us at `http://db.inf.uni-tuebingen.de`