Journey of a DML query

23-Feb-2023 @PGConf India

Vaibhav Dalvi

EDB

Who am I?

- My name is Vaibhav Dalvi
- Database developer at **EnterpriseDB**, Pune, India
- Started hacking Postgresql 4 years ago
- Email: vaibhav.dalvi@enterprisedb.com

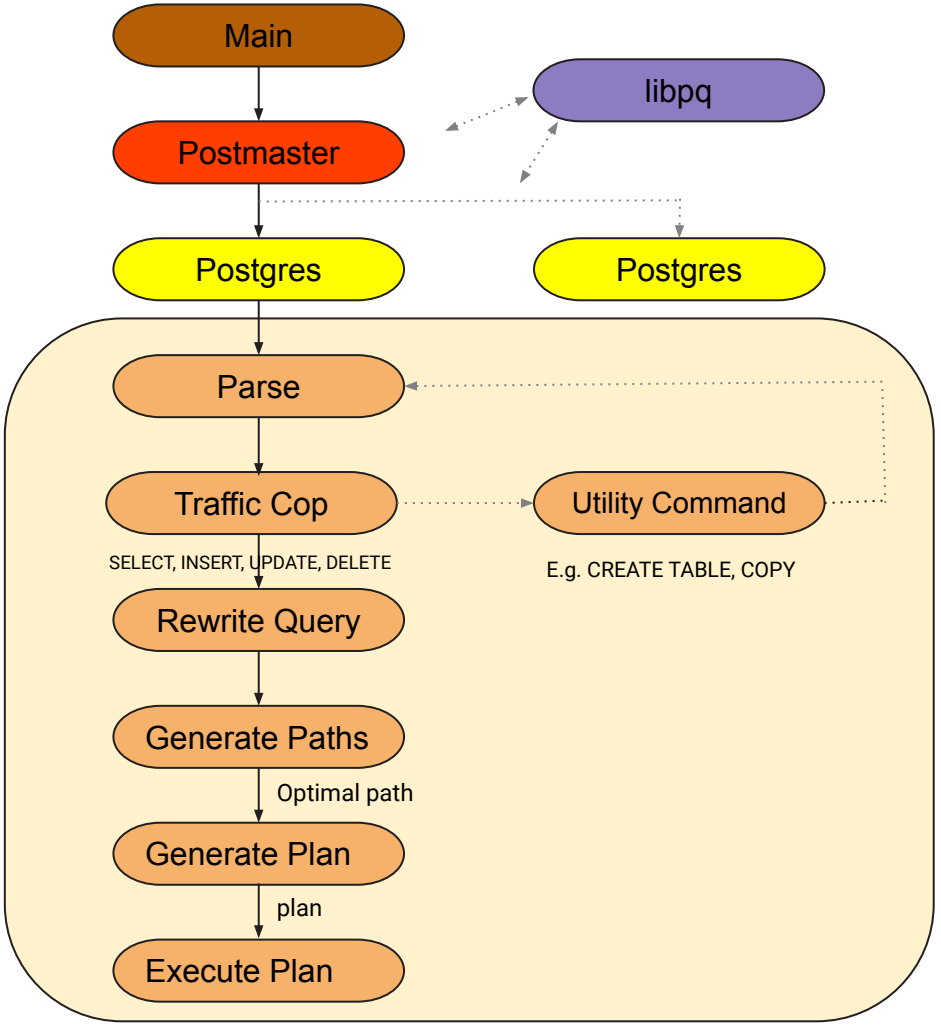
Agenda:

- Brief about backend
- List of DML queries
- All queries entry point
- Debug Postgresql
- Parse
- Analyze
- Rewrite
- Plan
- Execution
- Q&A

EDB™

Backend:

- Initialization
  - Bootstrap
  - Main
  - Postmaster
  - Libpq
- Main query flow
  - Parser
  - Analyzer
  - Planner/Optimizer
  - Executor



DML statements:

- **INSERT** to create new rows
- **UPDATE** to change existing rows
- **DELETE** to remove existing rows

```
CREATE TABLE products (
    product_no INTEGER,
    name       TEXT,
    price      NUMERIC
);
```

DML queries:

```
INSERT INTO products
(product_no, name, price)
VALUES(1, 'Cheese', 9.99);
```

```
UPDATE products SET price = 10
WHERE price = 9.99;
```

```
DELETE FROM products WHERE price
= 10;
```

EDB

Entry point:

```
/*
 * exec_simple_query
 *
 * Execute a "simple Query" protocol message.
 */
static void
exec_simple_query(const char *query_string)
{
      .
      .
      parsetree_list = pg_parse_query(query_string);
      foreach(parsetree_item, parsetree_list)
      {
            querytree_list = pg_analyze_and_rewrite_fixedparams(parsetree, query_string, NULL, 0,NULL);
            .
            .

            plantree_list = pg_plan_queries(querytree_list, query_string, CURSOR_OPT_PARALLEL_OK,NULL);
            .
            .

            /* subroutine to establish a portal's query */
            (void) PortalRun(portal, FETCH_ALL, true, true, receiver, receiver, &qc);
            .
            .
      }
}
```

Debugging Postgresql:

Step 1:

```
$ ./db/bin/psql   -d   postgres
psql (16devel)

#SELECT   pg_backend_pid();
 pg_backend_pid
----------------------------
        18418
```

Step 2:

```
$ gdb   -p   18418

Loaded symbols for /lib64/libnss_dns.so.2
0x00007f69400f80c3 in __epoll_wait_nocancel () from /lib64/libc.so.6
(gdb) b exec_simple_query
Breakpoint 1 at 0x9b2b0c: file postgres.c, line 981.
```

Step 3:

```
#INSERT INTO products VALUES (
postgres(# 1, 'Cheese', 9.99
postgres(# );
```

Step 4:

```
(gdb) c
Continuing.
Breakpoint 1, exec_simple_query (query_string=0x2ebfc78 "INSERT INTO
products VALUES (\n1, 'Cheese', 9.99\n);") at postgres.c:981
981             CommandDest dest = whereToSendOutput;
(gdb)
```

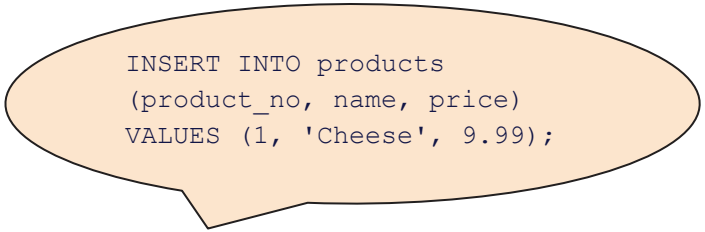
Parsing:

- `pg_parse_query()`
- `raw_parser()`
- `scanner_init()`
- `parser_init()`
- `base_yyparse()`

pg_parse_query:

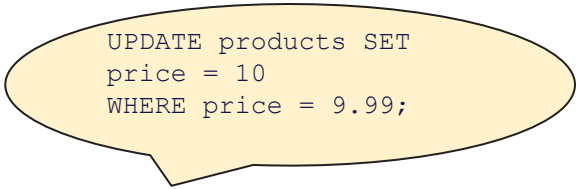
```
/* Do raw parsing (only). */
List *
pg_parse_query(const char *query_string)
{
   raw_parsetree_list = raw_parser(query_string, RAW_PARSE_DEFAULT);
   return raw_parsetree_list;
}

/* raw_parser: Given a query in string form, do lexical and grammatical analysis. */
List *
raw_parser(const char *str, RawParseMode mode)
{
   yyscanner = scanner_init(str, &yyextra.core_yy_extra, &ScanKeywords, ScanKeywordTokens);

   parser_init(&yyextra);

   yyresult = base_yyparse(yyscanner);
}
```

INSERT Syntax:



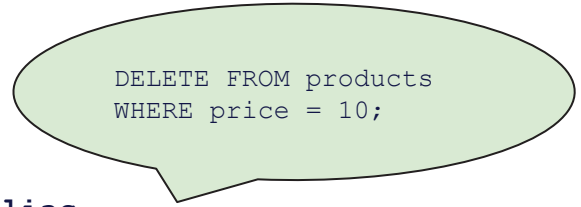

```
InsertStmt:
            opt_with_clause INSERT INTO insert_target insert_rest
            opt_on_conflict returning_clause
                {
                    $5->relation = $4;
                    $5->onConflictClause = $6;
                    $5->returningList = $7;
                    $5->withClause = $1;
                    $$ = (Node *) $5;
                }
        ;
```

UPDATE Syntax:




```
UpdateStmt: opt_with_clause UPDATE relation_expr_opt_alias
            SET set_clause_list from_clause
            where_or_current_clause
            returning_clause
              {
                  UpdateStmt *n = makeNode(UpdateStmt);

                  n->relation = $3;
                  n->targetList = $5;
                  n->fromClause = $6;
                  n->whereClause = $7;
                  n->returningList = $8;
                  n->withClause = $1;
                  $$ = (Node *) n;
              }
        ;
```

DELETE Syntax:




```
DeleteStmt: opt_with_clause DELETE_P FROM relation_expr_opt_alias
            using_clause where_or_current_clause returning_clause
               {
                    DeleteStmt *n = makeNode(DeleteStmt);

                    n->relation = $4;
                    n->usingClause = $5;
                    n->whereClause = $6;
                    n->returningList = $7;
                    n->withClause = $1;
                    $$ = (Node *) n;
               }
     ;
```

Analyzing & Rewriting:

- `pg_analyze_and_rewrite_fixedparams()`
- `parse_analyze_fixedparams()`
- `transformStmt()`
- `setTargetTable()`
- `pg_rewrite_query()`
- `QueryRewrite()`
- `RewriteQuery()`

Analyze & Rewrite:

```
/*
 * Given a raw parsetree (gram.y output), and optionally information about
 * types of parameter symbols ($n), perform parse analysis and rule rewriting.
 */
List *
pg_analyze_and_rewrite_fixedparams(RawStmt *parsetree, const char *query_string, const Oid,
                                   *paramTypes, int numParams, QueryEnvironment *queryEnv)
{
     /* (1) Perform parse analysis.*/
     query = parse_analyze_fixedparams(parsetree, query_string, paramTypes, numParams, queryEnv);

     /* (2) Rewrite the queries, as necessary */
     querytree_list = pg_rewrite_query(query);

     return querytree_list;
}
```

Analyze:

```
Query *
transformStmt(ParseState *pstate, Node *parseTree)
{
    switch (nodeTag(parseTree))
    {
     case T_InsertStmt:
          result = transformInsertStmt(pstate, (InsertStmt *) parseTree);
          break;

     case T_DeleteStmt:
          result = transformDeleteStmt(pstate, (DeleteStmt *) parseTree);
          break;

     case T_UpdateStmt:
          result = transformUpdateStmt(pstate, (UpdateStmt *) parseTree);
          Break;
     .
     .
    }
}
```

Rewrite:

```
List *
QueryRewrite(Query *parsetree)
{
   /* Step 1: Apply all non-SELECT rules possibly getting 0 or many queries */
   querylist = RewriteQuery(parsetree, NIL, 0);

   /* Step 2: Apply all the RIR rules on each query */
   foreach(l, querylist)
   {
       Query      *query = (Query *) lfirst(l);
       query = fireRIRrules(query, NIL);
       query->queryId = input_query_id;
       results = lappend(results, query);
   }

   /*
    * Step 3: Determine which, if any, of the resulting queries is supposed to set
    * the command-result tag; and update the canSetTag fields accordingly.
    */
}
```

RewriteQuery():

```
static List *
RewriteQuery(Query *parsetree, List *rewrite_events, int orig_rt_length)
{
    /* First, recursively process any insert/update/delete statements in WITH clauses. */
    foreach(lc1, parsetree->cteList)
    {
        newstuff = RewriteQuery(ctequery, rewrite_events, 0);
    }

    /* If the statement is an insert, update, delete, or merge, adjust its targetlist */
    if (event != CMD_SELECT && event != CMD_UTILITY)
    {
        /* Rewrite the targetlist as needed for the command type. */
      if (event == CMD_INSERT) { }
      else if (event == CMD_UPDATE) { }
      else if (event == CMD_DELETE) { /* Nothing to do here */}

      /* Collect and apply the appropriate rules. */
    }
    return rewritten;
}
```

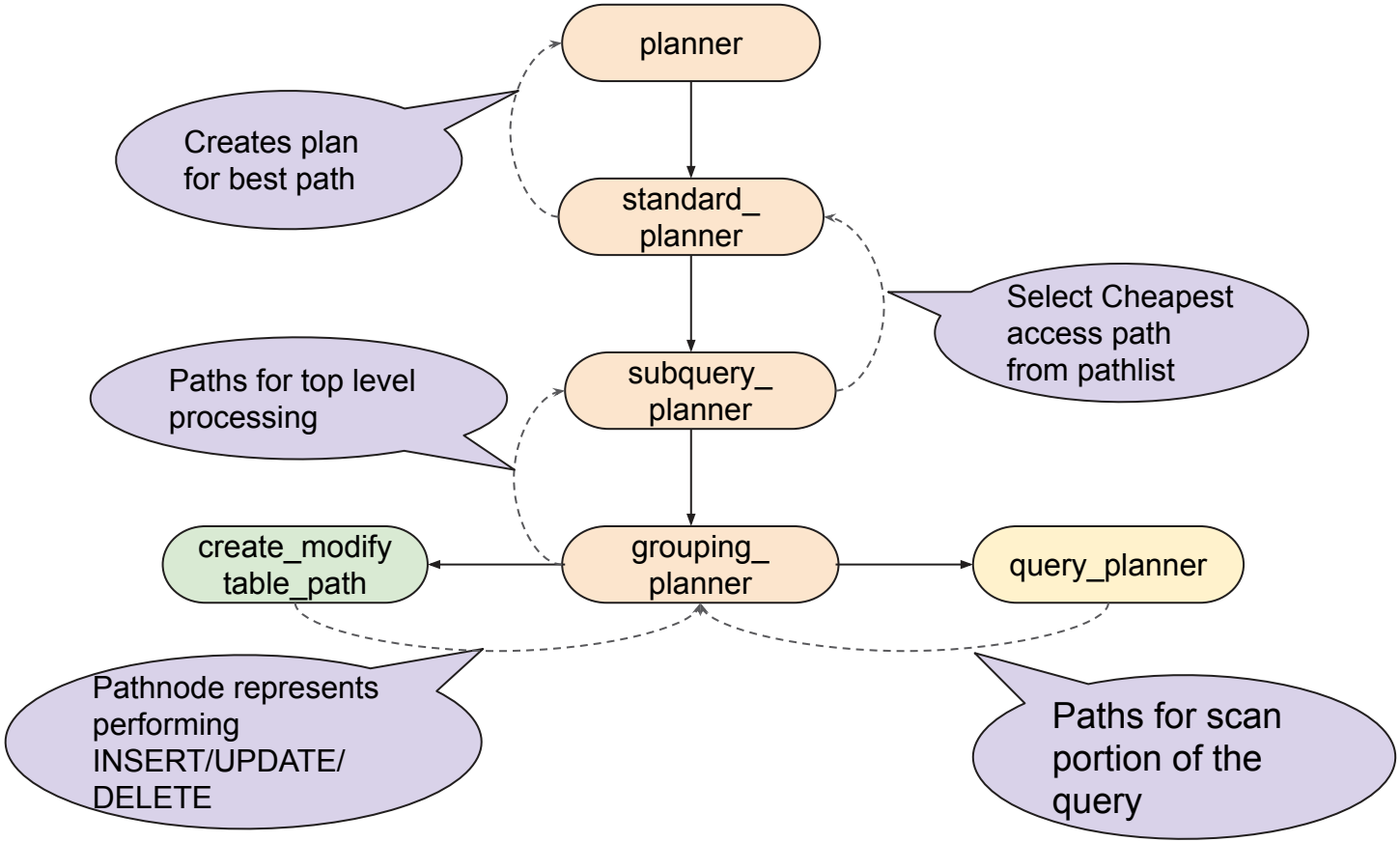
Plan:

- `planner()`
- `standard_plan()`
- `subquery_planner()`
- `grouping_planner()`
- `query_planner()`
- `create_modifytable_path()`

pg_plan_queries:

```
List *
pg_plan_queries(List *querytrees, const char *query_string, int cursorOptions, ParamListInfo boundParams)
{
      foreach(query_list, querytrees)
      {
          if (query->commandType == CMD_UTILITY)
          {
                /* Utility commands require no planning. */
          }
          else
          {
                stmt = pg_plan_query(query, query_string, cursorOptions, boundParams);
          }

          stmt_list = lappend(stmt_list, stmt);
      }

      return stmt_list;
}

PlannedStmt *
pg_plan_query(Query *querytree, const char *query_string, int cursorOptions, ParamListInfo boundParams)
{
      plan = planner(querytree, query_string, cursorOptions, boundParams);
}
```

Planning flow:



Add ModifyTable node:

```
ModifyTablePath *
create_modifytable_path(PlannerInfo *root, RelOptInfo *rel, Path *subpath, CmdType operation, ...)
{
     ModifyTablePath *pathnode = makeNode(ModifyTablePath);

     return pathnode;
}

/* ModifyTablePath represents performing INSERT/UPDATE/DELETE/MERGE */
typedef struct ModifyTablePath
{
     Path      path;
     Path     *subpath;       /* Path producing source data */
     CmdType   operation;     /* INSERT, UPDATE, DELETE, or MERGE */
     bool      canSetTag;     /* do we set the command tag/es_processed? */
     Index     nominalRelation;    /* Parent RT index for use of EXPLAIN */
     Index     rootRelation;  /* Root RT index, if target is partitioned */
     bool      partColsUpdated;    /* some part key in hierarchy updated? */
     List      *resultRelations;   /* integer list of RT indexes */
     List      *updateColnosLists; /* per-target-table update_colnos lists */
     List      *withCheckOptionLists;   /* per-target-table WCO lists */
     List      *returningLists; /* per-target-table RETURNING tlists */
     List      *rowMarks;     /* PlanRowMarks (non-locking only) */
     OnConflictExpr *onconflict; /* ON CONFLICT clause, or NULL */
     int       epqParam;      /* ID of Param for EvalPlanQual re-eval */
     List      *mergeActionLists;  /* per-target-table lists of actions for MERGE */
} ModifyTablePath;
```

Executor:

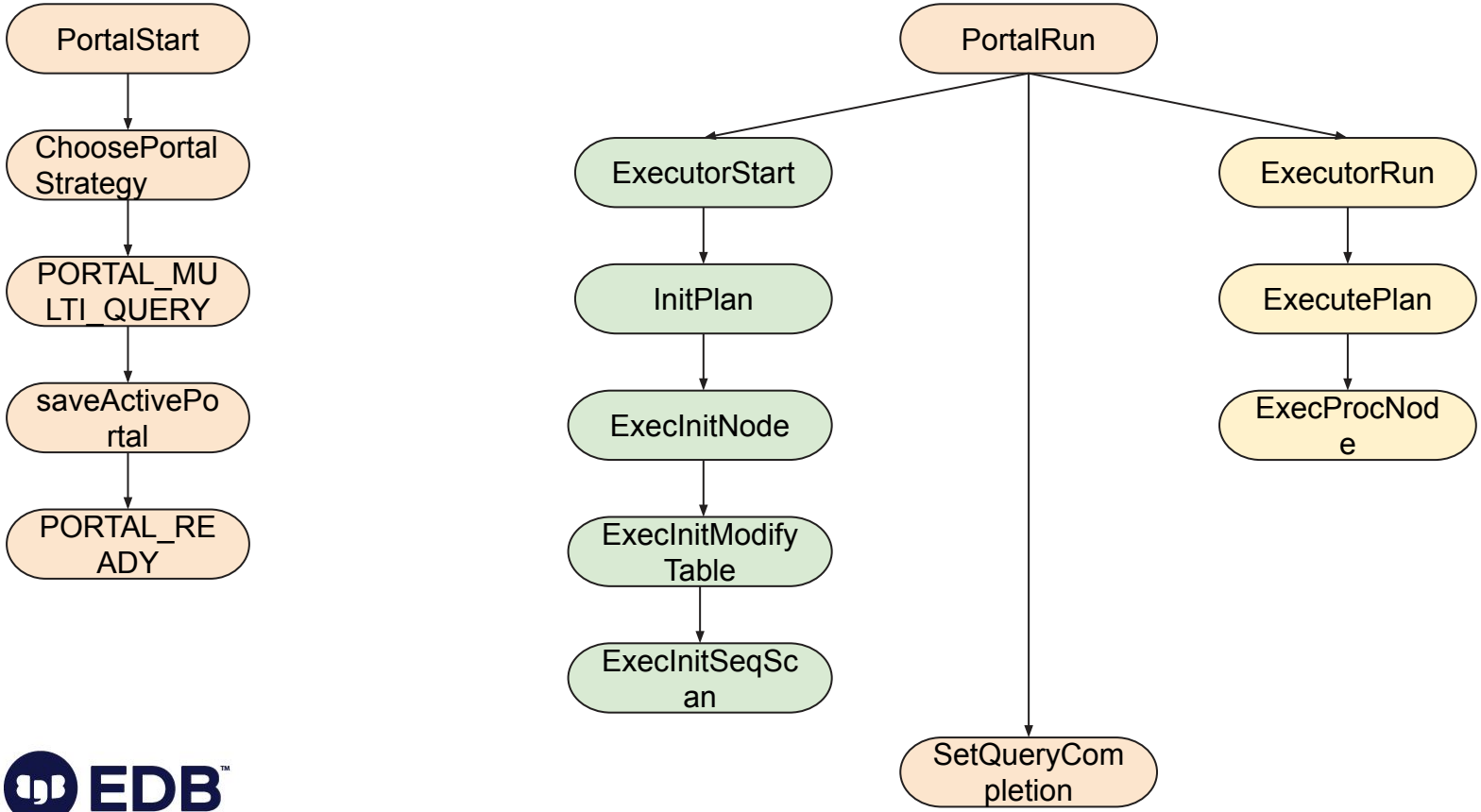
- `CreatePortal()`
- `PortalDefineQuery()`
- `PortalStart()`
- `PortalSetResultFormat()`
- `CreateDestReceiver()`
- `PortalRun()`
- `PortalDrop()`

EDB

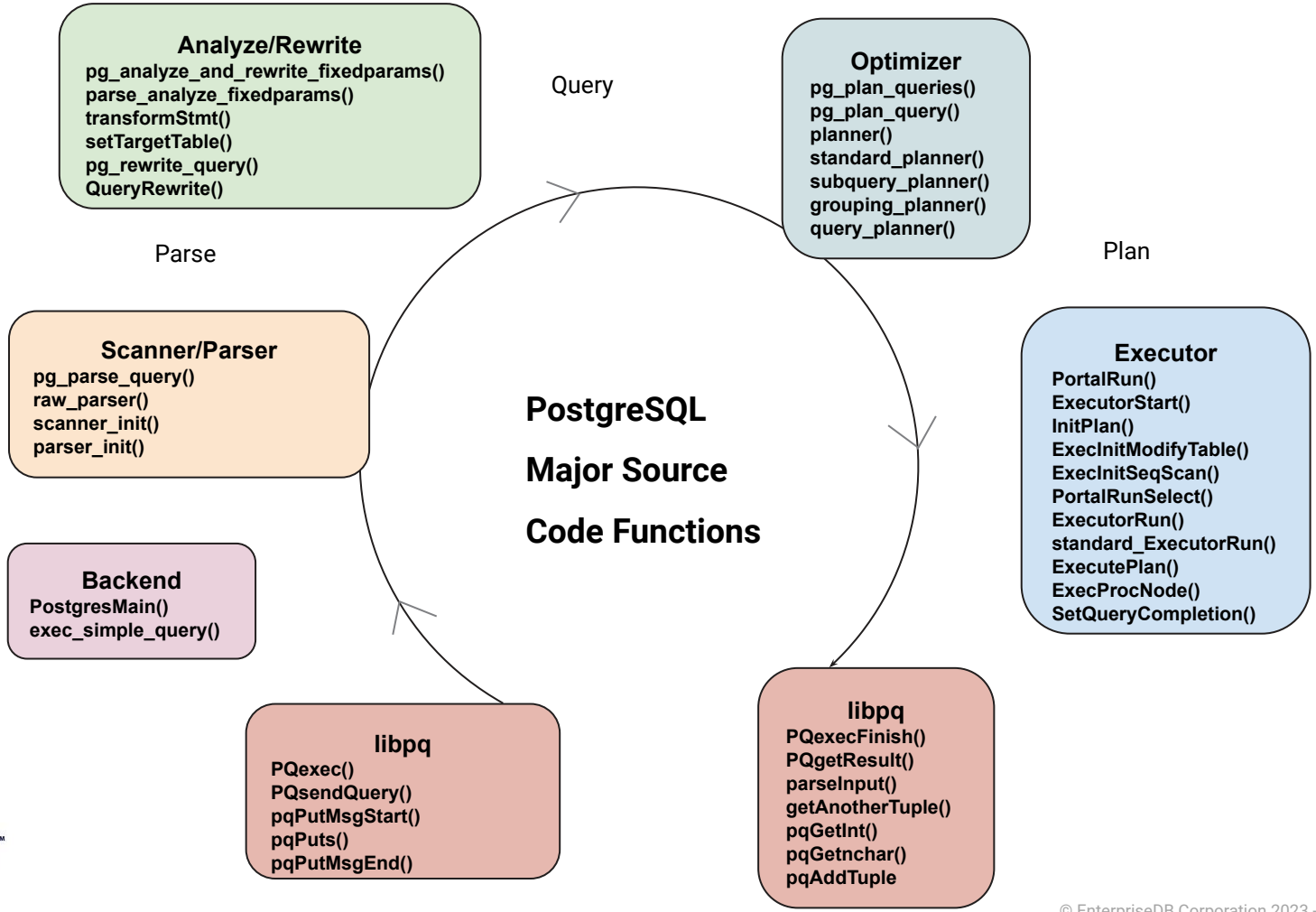
Execute:

```
static void
exec_simple_query(const char *query_string)
{
      .
      .
      querytree_list = pg_analyze_and_rewrite_fixedparams(parsetree, query_string, NULL, 0, NULL);
      .
      .
      plantree_list = pg_plan_queries(querytree_list, query_string, CURSOR_OPT_PARALLEL_OK, NULL);
      .
      .
      portal = CreatePortal("", true, true);

      PortalDefineQuery(portal, NULL, query_string, commandTag, plantree_list, NULL);

      /* Start the portal.  No parameters here.*/
      PortalStart(portal, NULL, 0, InvalidSnapshot);

      PortalSetResultFormat(portal, 1, &format);

      /* Run the portal to completion, and then drop it (and the receiver). */
      (void) PortalRun(portal, FETCH_ALL, true, true, receiver, receiver, &qc);

      PortalDrop(portal, false);
}
```

Executor Flow:



Recap:



References:

- PostgreSQL 15 official documentation
- PostgreSQL Internals Through Pictures by Bruce Momjian
- Source flow diagram by Bruce Momjian
- The internals of PostgreSQL by Hironobu SUZUKI

EDB

Questions?

Thank you

EDB

EDB™

POWER TO POSTGRES

We are hiring. Be part of our team!

Scan Now to Apply!

careers@enterprisedb.com

@EDBPostgres /company/EDBPostgres www.enterprisedb.com