INLINING FUNCTIONS NOW AND THEN

Paul A. Jungwirth

12 May 2025

LINEUP

- Inlining SQL Set-Returning Functions
- Inlining Non-Set-Returning SQL Functions
- Inlining Non-SQL Set-Returning Functions

SRFS

```
1 =# EXPLAIN ANALYZE SELECT *
 2 FROM visible sales slow(1) AS s
 3 WHERE vendor_id = 5000;
 4
               QUERY PLAN
 6
    Function Scan on visible_sales_slow s
      (cost=0.25..12.75 \text{ rows}=5 \text{ width}=56)
 8
      (actual time=57.415..57.670 rows=2 loops=1)
      Filter: (vendor_id = 5000)
10
      Rows Removed by Filter: 51688
11 Planning Time: 0.129 ms
12
    Execution Time: 57.925 ms
13 (5 rows)
```

```
1 =# EXPLAIN ANALYZE SELECT *
2 FROM visible sales slow(1) AS s
 3 WHERE vendor_id = 5000;
                  QUERY PLAN
  Function Scan on visible sales slow s
      (cost=0.25.12.75 \text{ rows}=5 \text{ width}=56)
    (actual time=57.415..57.670 rows=2 loops=1)
      Filter: (vendor id = 5000)
      Rows Removed by Filter: 51688
11 Planning Time: 0.129 ms
    Execution Time: 57.925 ms
13 (5 rows)
```

```
1 =# EXPLAIN ANALYZE SELECT *
 2 FROM visible sales slow(1) AS s
 3 WHERE vendor_id = 5000;
 4
               QUERY PLAN
 6
    Function Scan on visible_sales_slow s
      (cost=0.25..12.75 \text{ rows}=5 \text{ width}=56)
 8
      (actual time=57.415..57.670 rows=2 loops=1)
      Filter: (vendor_id = 5000)
10
      Rows Removed by Filter: 51688
11 Planning Time: 0.129 ms
12
    Execution Time: 57.925 ms
13 (5 rows)
```

```
1 =# EXPLAIN ANALYZE SELECT *
2 FROM visible sales slow(1) AS s
 3 WHERE vendor_id = 5000;
                  OUERY PLAN
 6 Function Scan on visible_sales_slow s
      (cost=0.25.12.75 \text{ rows}=5 \text{ width}=56)
 8
      (actual time=57.415..57.670 rows=2 loops=1)
      Filter: (vendor_id = 5000)
10
      Rows Removed by Filter: 51688
11 Planning Time: 0.129 ms
    Execution Time: 57.925 ms
13 (5 rows)
```

```
1 =# EXPLAIN ANALYZE SELECT *
2 FROM visible sales slow(1) AS s
  WHERE vendor_id = 5000;
                  OUERY PLAN
 6 Function Scan on visible_sales_slow s
      (cost=0.25.12.75 \text{ rows}=5 \text{ width}=56)
    (actual time=57.415..57.670 rows=2 loops=1)
      Filter: (vendor id = 5000)
      Rows Removed by Filter: 51688
11 Planning Time: 0.129 ms
12
    Execution Time: 57.925 ms
13 (5 rows)
```

```
Nested Loop
      (cost=0.84..32.53 rows=7 width=27)
 3
      (actual time=0.044..0.048 rows=2 loops=1)
          Index Only Scan using idx_memberships_company_user c
 5
             (cost=0.42..4.44 \text{ rows}=1 \text{ width}=4)
 6
             (actual time=0.025..0.026 rows=1 loops=1)
            Index Cond: ((company_id = 5000) AND (user_id = 1)
      -> Index Scan using uq_sales_po_number on sales
8
           (cost=0.42..28.02 rows=7 width=27)
10
           (actual time=0.013..0.016 rows=2 loops=1)
             Index Cond: (vendor_id = 5000)
11
12
    Planning Time: 0.258 ms
13
    Execution Time: 0.071 ms
```

```
Nested Loop
  (cost=0.84..32.53 \text{ rows}=7 \text{ width}=27)
  (actual time=0.044..0.048 rows=2 loops=1)
      Index Only Scan using idx_memberships_company_user o
         (cost=0.42...4.44 \text{ rows}=1 \text{ width}=4)
         (actual time=0.025..0.026 rows=1 loops=1)
        Index Cond: ((company_id = 5000) AND (user_id = 1)
  -> Index Scan using uq_sales_po_number on sales
       (cost=0.42...28.02 rows=7 width=27)
       (actual time=0.013..0.016 rows=2 loops=1)
        Index Cond: (vendor_id = 5000)
Planning Time: 0.258 ms
Execution Time: 0.071 ms
```

```
Nested Loop
      (cost=0.84...32.53 \text{ rows}=7 \text{ width}=27)
      (actual time=0.044..0.048 rows=2 loops=1)
         Index Only Scan using idx_memberships_company_user c
5
            (cost=0.42..4.44 \text{ rows}=1 \text{ width}=4)
6
            (actual time=0.025..0.026 rows=1 loops=1)
            Index Cond: ((company_id = 5000) AND (user_id = 1)
     -> Index Scan using uq_sales_po_number on sales
          (cost=0.42...28.02 rows=7 width=27)
          (actual time=0.013..0.016 rows=2 loops=1)
            Index Cond: (vendor_id = 5000)
   Planning Time: 0.258 ms
   Execution Time: 0.071 ms
```

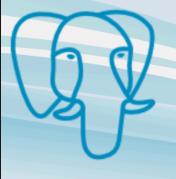
```
Nested Loop
      (cost=0.84...32.53 \text{ rows}=7 \text{ width}=27)
      (actual time=0.044..0.048 rows=2 loops=1)
          Index Only Scan using idx_memberships_company_user o
             (cost=0.42..4.44 rows=1 width=4)
             (actual time=0.025...0.026 rows=1 loops=1)
            Index Cond: ((company_id = 5000) AND (user_id = 1)
      -> Index Scan using uq_sales_po_number on sales
8
           (cost=0.42..28.02 rows=7 width=27)
10
           (actual time=0.013..0.016 rows=2 loops=1)
             Index Cond: (vendor_id = 5000)
11
    Planning Time: 0.258 ms
    Execution Time: 0.071 ms
```



view source

https://wiki.postgresql.org/wiki/Inlining_of_SQL_functions

log in



navigation

- Main Page
- Random page
- Recent changes
- Help

tools

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link
- Page information

search

Search PostgreSQL w Go Search

Want to edit, but don't see an edit button when logged in? Click here.

history

Inlining of SQL functions

Contents [hide]

- 1 Inlining of SQL functions
 - 1.1 Scalar functions

discussion

page

- 1.2 Table functions
- 1.3 Inlining conditions for scalar functions
- 1.4 Inlining conditions for table functions

Inlining of SQL functions

SQL functions (i.e. LANGUAGE SQL) will, under certain conditions, have their function bodies inlined into the calling query rather than being invoked directly. This can have substantial performance advantages since the function body becomes exposed to the planner of the calling query, which can apply optimizations such as constant-folding, qual pushdown, and so on.

However, the exact conditions which apply to inlining are somewhat complex and not well documented outside the source code. This page is an attempt to partially rectify that.

The rules set out here are believed to be correct for pg versions between 8.4 and 9.5.

There are actually two completely separate forms of inlining that can occur, at most one of which is possible for any given function call: one for scalar function calls and the other for table function calls.

```
1 =# EXPLAIN (ANALYZE) SELECT *
           visible sales(1) AS s
 2 FROM
 3 WHERE vendor_id = 2 LIMIT 10;
 4
                      QUERY PLAN
    Limit (cost=0.42..5.84 rows=10 width=27)
           (actual time=0.972..1.010 rows=10 loops=1)
 8
          Nested Loop Semi Join
            (cost=0.42..2722.73 rows=5020 width=27)
10
            (actual time=0.970..1.004 rows=10 loops=1)
11
            -> Seq Scan on sales
12
                  (cost=0.00..2655.54 rows=5020 width=27)
13
                  (actual time=0.868..0.896 rows=10 loops=1)
14
                  Filter: (vendor_id = 2)
                  Rows Removed by Filter: 134
15
```

```
1 =# EXPLAIN (ANALYZE) SELECT *
2 FROM
          visible sales(1) AS s
3 WHERE vendor_id = 2 LIMIT 10;
                      QUERY PLAN
   Limit (cost=0.42...5.84 \text{ rows}=10 \text{ width}=27)
           (actual time=0.972..1.010 rows=10 loops=1)
         Nested Loop Semi Join
            (cost=0.42..2722.73 rows=5020 width=27)
            (actual time=0.970..1.004 rows=10 loops=1)
           -> Seq Scan on sales
                  (cost=0.00..2655.54 rows=5020 width=27)
                  (actual time=0.868..0.896 rows=10 loops=1)
                  Filter: (vendor id = 2)
                  Rows Removed by Filter: 134
```

```
(actuat time=0.9/0.11.004 tows=10 toops=1)
            -> Seg Scan on sales
                   (cost=0.00..2655.54 rows=5020 width=27)
                   (actual time=0.868..0.896 rows=10 loops=1)
                  Filter: (vendor_id = 2)
                  Rows Removed by Filter: 134
            -> Materialize (cost=0.42..4.44 rows=1 width=4)
                              (actual time=0.010..0.010 rows=1
                  -> Index Only Scan using idx_memberships_co
                         (cost=0.42..4.44 rows=1 width=4)
                         (actual time=0.092..0.092 rows=1 loops
                        Index Cond: ((company_id = 2) AND (use
22
    Planning Time: 0.515 ms
23
    Execution Time: 1.058 ms
   (11 rows)
```

```
(actual time=0.970..1.004 rows=10 loops=1)
11
            -> Seq Scan on sales
12
                   (cost=0.00..2655.54 rows=5020 width=27)
13
                   (actual time=0.868..0.896 rows=10 loops=1)
                  Filter: (vendor_id = 2)
14
15
                  Rows Removed by Filter: 134
            -> Materialize (cost=0.42..4.44 rows=1 width=4)
                              (actual time=0.010..0.010 rows=1
                  -> Index Only Scan using idx_memberships_co
                         (cost=0.42..4.44 rows=1 width=4)
                         (actual time=0.092..0.092 rows=1 loops
                         Index Cond: ((company_id = 2) AND (use
22
    Planning Time: 0.515 ms
23
    Execution Time: 1.058 ms
   (11 rows)
```

```
(actuat time=0.9/0.11.004 tows=10 toops=1)
             -> Seg Scan on sales
                   (cost=0.00..2655.54 rows=5020 width=27)
                   (actual time=0.868..0.896 rows=10 loops=1)
                   Filter: (vendor_id = 2)
                   Rows Removed by Filter: 134
            -> Materialize (cost=0.42..4.44 rows=1 width=4)
                               (actual time=0.010..0.010 rows=1
18
                   -> Index Only Scan using idx_memberships_cc
19
                          (cost=0.42..4.44 \text{ rows}=1 \text{ width}=4)
20
                          (actual time=0.092..0.092 rows=1 loops
21
                         Index Cond: ((company_id = 2) AND (use
    Planning Time: 0.515 ms
    Execution Time: 1.058 ms
   (11 rows)
```

```
Query *inline_set_returning_function(...) {
     /* Fetch the function body */
     Datum tmp = SysCacheGetAttrNotNull(PROCOID, func_tuple, /
     char *src = TextDatumGetCString(tmp);
 6
     /* Set up to handle parameters */
     SQLFunctionParseInfoPtr pinfo = prepare_sql_fn_parse_info
8
       func_tuple, (Node *) fexpr, fexpr->inputcollid);
10
     /* Parse, analyze, and rewrite */
     List *raw_parsetree_list = pg_parse_query(src);
11
     List *querytree_list = pg_analyze_and_rewrite_withcb(
12
       linitial(raw_parsetree_list), src, ..., pinfo, NULL);
13
     Query *querytree = linitial(querytree_list);
14
15
```

```
Query *inline_set_returning_function(...) {
    /* Fetch the function body */
    Datum tmp = SysCacheGetAttrNotNull(PROCOID, func_tuple, /
3
4
    char *src = TextDatumGetCString(tmp);
    SQLFunctionParseInfoPtr pinfo = prepare_sql_fn_parse_info
      func_tuple, (Node *) fexpr, fexpr->inputcollid);
    List *raw_parsetree_list = pg_parse_query(src);
    List *querytree_list = pg_analyze_and_rewrite_withcb(
      linitial(raw_parsetree_list), src, ..., pinfo, NULL);
    Query *querytree = linitial(querytree_list);
```

```
Query *inline_set_returning_function(...) {
    Datum tmp = SysCacheGetAttrNotNull(PROCOID, func_tuple, /
    char *src = TextDatumGetCString(tmp);
6
    /* Set up to handle parameters */
    SQLFunctionParseInfoPtr pinfo = prepare_sql_fn_parse_info
8
      func_tuple, (Node *) fexpr, fexpr->inputcollid);
    List *raw_parsetree_list = pg_parse_query(src);
    List *querytree_list = pg_analyze_and_rewrite_withcb(
      linitial(raw_parsetree_list), src, ..., pinfo, NULL);
    Query *querytree = linitial(querytree_list);
```

```
SQLFunctionParseInfoPtr pinfo = prepare_sql_fn_parse_info
       func_tuple, (Node *) fexpr, fexpr->inputcollid);
10
     /* Parse, analyze, and rewrite */
     List *raw_parsetree_list = pg_parse_query(src);
11
     List *querytree_list = pg_analyze_and_rewrite_withcb(
12
       linitial(raw_parsetree_list), src, ..., pinfo, NULL);
13
     Query *querytree = linitial(querytree list);
14
     querytree = substitute_actual_srf_parameters(
       querytree,
       funcform->pronargs,
       fexpr->args);
```

```
Tunc tuple, (Node *) Texpr, Texpr->inputcollid);
     List *raw_parsetree_list = pg_parse_query(src);
     List *querytree_list = pg_analyze_and_rewrite_withcb(
       linitial(raw_parsetree_list), src, ..., pinfo, NULL);
     Query *querytree = linitial(querytree list);
16
     querytree = substitute_actual_srf_parameters(
17
       querytree,
       funcform->pronargs,
18
       fexpr->args);
19
     return querytree;
```

```
Tunc tuple, (Node *) Texpr, Texpr->inputcollid);
     List *raw_parsetree_list = pg_parse_query(src);
     List *querytree_list = pg_analyze_and_rewrite_withcb(
       linitial(raw_parsetree_list), src, ..., pinfo, NULL);
     Query *querytree = linitial(querytree list);
     querytree = substitute_actual_srf_parameters(
       querytree,
       funcform->pronargs,
       fexpr->args);
21
     return querytree;
```

temporal_semijoin

```
SELECT a.id,
        UNNEST(multirange(a.valid_at) * j.valid_at) AS valid_at
FROM a
JOIN (
    SELECT b.id, range_agg(b.valid_at) AS valid_at
    FROM b
    GROUP BY b.id
) AS j
ON a.id = j.id AND a.valid_at && j.valid_at;
```

from https://github.com/pjungwir/temporal_ops

temporal_semijoin

```
CREATE OR REPLACE FUNCTION temporal semijoin(
  left_table text, left_id_col text, left_valid col text,
  right table text, right id col text, right valid col text
RETURNS SETOF RECORD AS $$
DECLARE
  subquery TEXT := 'j';
BEGIN
  IF left_table = 'j' OR right_table = 'j' THEN
    subquery := 'j1';
    IF left table = 'j1' OR right table = 'j1' THEN
      subquery := 'j2';
    END IF:
  END IF;
  RETURN QUERY EXECUTE format($j$
    SELECT %1$I.%2$I, UNNEST(multirange(%1$I.%3$I) * %7$I.%6$I) AS %3$I
    FROM
            %1$I
    JOIN (
      SELECT %4$I.%5$I, range agg(%4$I.%6$I) AS %6$I
      FROM
              %4$I
      GROUP BY %4$I.%5$I
    ) AS %7$I
    ON \$1\$I.\$2\$I = \$7\$I.\$5\$I AND \$1\$I.\$3\$I && \$7\$I.\$6\$I;
```

```
1 CREATE OR REPLACE FUNCTION commission cents(
 2 _sale_id INTEGER, _salesperson_id INTEGER
 4 RETURNS INTEGER
 5 AS $$
6 SELECT s.total price cents *
              COALESCE(m.commission_percent, 0)
8 FROM sales AS s
 9 LEFT JOIN memberships AS m
10 ON
            m.company_id = s.vendor_id
11 AND m.user_id = _salesperson_id
12 WHERE s.id = _sale_id;
13 $$ LANGUAGE sql STABLE;
```

```
1 CREATE OR REPLACE FUNCTION commission_cents(
2 _sale_id INTEGER, _salesperson_id INTEGER
4 RETURNS INTEGER
5 AS $$
6 SELECT s.total price cents *
              COALESCE(m.commission_percent, 0)
8 FROM sales AS s
  LEFT JOIN memberships AS m
            m.company_id = s.vendor_id
11 AND m.user_id = _salesperson_id
12 WHERE s.id = _sale_id;
13 $$ LANGUAGE sql STABLE;
```

```
1 CREATE OR REPLACE FUNCTION commission_cents(
2 _sale_id INTEGER, _salesperson_id INTEGER
4 RETURNS INTEGER
5 AS $$
6 SELECT s.total_price_cents *
              COALESCE(m.commission_percent, 0)
8 FROM sales AS s
   LEFT JOIN memberships AS m
10
    ON
            m.company_id = s.vendor_id
11 AND m.user_id = _salesperson_id
12 WHERE s.id = _sale_id;
13 $$ LANGUAGE sql STABLE;
```

SUPPORT PROCS

1 2	=# \d pg_proc Table "pg catalog.pg proc"				
3	Column		Collation		D€
5	oid	 oid	1	not null	
6	proname	name		not null	
7	pronamespace	oid		not null	
8	proowner	oid		not null	
9	prolang	oid		not null	
10	procost	real		not null	
11	prorows	real		not null	
12	provariadic	oid		not null	
13	prosupport	regproc		not null	
14	prokind	"char"		not null	
15	nrosecdef	hoolean		not null	

SUPPORT PROCS

6	proname	name	not null
	pronamespace	oid	not null
8	proowner	oid	not null
9	prolang	oid	not null
10	procost	real	not null
	prorows	real	not null
	provariadic	oid	not null
13	prosupport	regproc	not null
14	prokind	"char"	not null
15	prosecdef	boolean	not null
16	proleakproof	boolean	not null
	proisstrict	boolean	not null
18	proretset	boolean	not null
19	provolatile	"char"	not null
20	proparallel	"char"	not null

SUPPORT REQUESTS

- SupportRequestRows
- SupportRequestSelectivity
- SupportRequestCost
- SupportRequestIndexCondition
- SupportRequestWFuncMonotonic
- SupportRequestOptimizeWindowClause
- SupportRequestModifyInPlace
- SupportRequestSimplify

```
SELECT total_price_cents,
```

commission_cents(id, \$1)

FROM sales

WHERE salesperson_id IS NOT DISTINCT FROM \$1

AND sold_at BETWEEN start_of_month(\$2)

AND end_of_month(\$2)

commission_cents_support

```
1 CREATE OR REPLACE FUNCTION commission_cents(
 2 _sale_id INTEGER, _salesperson_id INTEGER
4 RETURNS INTEGER
5 AS $$
6 SELECT total_price_cents * COALESCE(commission_percent,
7 FROM sales AS s
8 LEFT JOIN memberships AS m
 9 ON m.company_id = s.vendor_id
10 AND m.user_id = _salesperson_id
11 WHERE s.id = _sale_id;
12 $$ LANGUAGE sql STABLE
13 SUPPORT commission_cents_support;
```

commission_cents_support

```
1 CREATE OR REPLACE FUNCTION commission_cents(
2 _sale_id INTEGER, _salesperson_id INTEGER
4 RETURNS INTEGER
5 AS $$
6 SELECT total_price_cents * COALESCE(commission_percent,
7 FROM sales AS s
8 LEFT JOIN memberships AS m
  10 AND m.user_id = _salesperson_id
11 WHERE s.id = _sale_id;
12 $$ LANGUAGE sql STABLE
  SUPPORT commission_cents_support;
```

```
CREATE OR REPLACE FUNCTION commission_cents_support(INTERNAL)
RETURNS INTERNAL
AS 'commission_cents', 'commission_cents_support'
LANGUAGE C;
```

```
Datum commission_cents_support(PG_FUNCTION_ARGS) {
   Node *rawreq = (Node *) PG_GETARG_POINTER(0);
   SupportRequestSimplify *req;

if (!IsA(req, SupportRequestSimplify)) {
   PG_RETURN_POINTER(NULL);

req = (SupportRequestSimplify *) rawreq;

FuncExpr *expr = req->fcall;

...
```

```
1 Datum commission_cents_support(PG_FUNCTION_ARGS) {
2   Node *rawreq = (Node *) PG_GETARG_POINTER(0);
3   SupportRequestSimplify *req;
4   if (!IsA(req, SupportRequestSimplify)) {
6     PG_RETURN_POINTER(NULL);
7   req = (SupportRequestSimplify *) rawreq;
9   FuncExpr *expr = req->fcall;
11   ...
```

```
Datum commission_cents_support(PG_FUNCTION_ARGS) {
   Node *rawreq = (Node *) PG_GETARG_POINTER(0);
   SupportRequestSimplify *req;

if (!IsA(req, SupportRequestSimplify)) {
   PG_RETURN_POINTER(NULL);

req = (SupportRequestSimplify *) rawreq;

FuncExpr *expr = req->fcall;

...
```

```
1 Datum commission_cents_support(PG_FUNCTION_ARGS) {
2   Node *rawreq = (Node *) PG_GETARG_POINTER(0);
3   SupportRequestSimplify *req;
4
5   if (!IsA(req, SupportRequestSimplify)) {
6     PG_RETURN_POINTER(NULL);
7
8   req = (SupportRequestSimplify *) rawreq;
9
10   FuncExpr *expr = req->fcall;
11
12   ...
```

```
typedef struct SupportRequestSimplify
{
   NodeTag type;

   struct PlannerInfo *root;
   FuncExpr *fcall;
} SupportRequestSimplify;
```

```
1 Node *node = lsecond(expr->args);
 2 if (IsA(node, Const)) {
    Const *c = (Const *) node;
 4 if (c->constisull) {
      Const *ret = makeConst(
 6
        INT40ID, /* type */
                    /* typmod */
        -1,
8
        0,
                     /* collid */
        4,
                      /* len */
        Int32GetDatum(0), /* value */
10
11
      false,
                     /* isnull */
                     /* byval */
12
        true
13
      PG_RETURN_POINTER(ret);
14
15
```

```
1 Node *node = lsecond(expr->args);
2 if (IsA(node, Const)) {
   Const *c = (Const *) node;
 if (c->constisnull) {
  Const *ret = makeConst(
       INT40ID, /* type */
       -1,
       0,
   Int32GetDatum(0), /* value */
     PG RETURN POINTER(ret);
```

```
1 Node *node = lsecond(expr->args);
2 if (IsA(node, Const)) {
   Const *c = (Const *) node;
3
4 if (c->constisnull) {
  Const *ret = makeConst(
       INT40ID, /* type */
       -1,
       0,
   Int32GetDatum(0), /* value */
     PG RETURN POINTER(ret);
```

```
1 Node *node = lsecond(expr->args);
 2 if (IsA(node, Const)) {
    Const *c = (Const *) node;
 4 if (c->constisull) {
      Const *ret = makeConst(
 6
        INT40ID, /* type */
                    /* typmod */
        -1,
8
        0,
                     /* collid */
        4,
                      /* len */
        Int32GetDatum(0), /* value */
10
11
      false,
                     /* isnull */
                     /* byval */
12
        true
13
      PG_RETURN_POINTER(ret);
14
15
```

```
COUST *C = \{COUST * i | HOUE;
    if (c->constisnull) {
      Const *ret = makeConst(
        INT40ID, /* type */
        -1,
        0,
     Int32GetDatum(0), /* value */
      PG RETURN POINTER(ret);
   PG_RETURN_POINTER(NULL);
17
```

THERE'S A PATCH FOR THAT!

https://commitfest.postgresql.org/patch/5083/

```
1 char *sql = "...";
2 List *parsed = pg_parse_query(sql);
3 List *analyzed = pg_analyze_and_rewrite_with_cb(
4    linitial(parsed),
5    sql,
6    (ParserSetupHook) sql_fn_parser_setup,
7    pinfo,
8    NULL);
9 Query *q = linitial(analyzed);
10 PG_RETURN_POINTER(q);
```

```
1 char *sql = "...";
2 List *parsed = pg_parse_query(sql);
3 List *analyzed = pg_analyze_and_rewrite_with_cb(
4    linitial(parsed),
5    sql,
6    (ParserSetupHook) sql_fn_parser_setup,
7    pinfo,
8    NULL);
9 Query *q = linitial(analyzed);
10 PG_RETURN_POINTER(q);
```

```
1 char *sql = "...";
2 List *parsed = pg_parse_query(sql);
3 List *analyzed = pg_analyze_and_rewrite_with_cb(
4    linitial(parsed),
5    sql,
6    (ParserSetupHook) sql_fn_parser_setup,
7    pinfo,
8    NULL);
9 Query *q = linitial(analyzed);
10 PG_RETURN_POINTER(q);
```

```
1 char *sql = "...";
2 List *parsed = pg_parse_query(sql);
3 List *analyzed = pg_analyze_and_rewrite_with_cb(
4    linitial(parsed),
5    sql,
6    (ParserSetupHook) sql_fn_parser_setup,
7    pinfo,
8    NULL);
9 Query *q = linitial(analyzed);
10 PG_RETURN_POINTER(q);
```

```
1 char *sql = "...";
2 List *parsed = pg_parse_query(sql);
3 List *analyzed = pg_analyze_and_rewrite_with_cb(
4    linitial(parsed),
5    sql,
6    (ParserSetupHook) sql_fn_parser_setup,
7    pinfo,
8    NULL);
9 Query *q = linitial(analyzed);
10 PG_RETURN_POINTER(q);
```

CAVEATS

GENERALIZING

```
1 CREATE OR REPLACE FUNCTION temporal semijoin(
     left table text, left id col text, left valid col text,
     right table text, right id col text, right valid col text
 4)
 5 RETURNS SETOF RECORD AS $$
 6 DECLARE
     subquery TEXT := 'j';
 8 BEGIN
     IF left table = 'j' OR right table = 'j' THEN
       subquery := 'i1':
10
       IF left_table = 'j1' OR right_table = 'j1' THEN
11
12
         subquery := 'i2':
13
       END IF:
14
     END IF;
     RETURN QUERY EXECUTE format($ggg$
15
16
       SELECT %1$I.%2$I, UNNEST(multirange(%1$I.%3$I) * %7$I.%6$I) AS %3$I
17
       FROM
               %1$I
18
       JOIN (
19
         SELECT %4$I.%5$I, range agg(%4$I.%6$I) AS %6$I
20
         FROM
                 %4$I
21
         GROUP BY %4$I.%5$I
22
       ) AS %7$I
23
       ON %1$I.%2$I = %7$I.%5$I AND %1$I.%3$I && %7$I.%6$I;
```

GENERALIZING

```
IF left table = 'j1' OR right table = 'j1' THEN
15
     RETURN QUERY EXECUTE format($qqq$
       SELECT %1$I.%2$I, UNNEST(multirange(%1$I.%3$I) * %7$I.%6$I) AS %3$I
17
       FROM
               %1$I
       JOIN (
18
19
         SELECT %4$I.%5$I, range agg(%4$I.%6$I) AS %6$I
20
         FROM
                 %4$I
21
         GROUP BY %4$I.%5$I
22
       ) AS %7$I
23
       ON %1$I.%2$I = %7$I.%5$I AND %1$I.%3$I && %7$I.%6$I;
24
     $qqq$,
     left table, left id col, left valid col,
25
26
     right table, right id col, right valid col,
27
     subquery);
29 $$ STABLE LEAKPROOF PARALLEL SAFE SUPPORT temporal semijoin support LANGUAGE plpgsgl;
```

BIBLIOGRAPHY

- https://github.com/pjungwir/inlining-postgres-function
- https://wiki.postgresql.org/wiki/Inlining_of_SQL_func
- https://commitfest.postgresql.org/patch/5083/
- https://github.com/pjungwir/temporal_ops

THANK YOU!

https://github.com/pjungwir/inlining-postgresfunctions