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

```
CREATE OR REPLACE FUNCTION visible_sales(user_id INT)
RETURNS SETOF sales
AS $$
  SELECT * FROM sales
  WHERE vendor_id IN (SELECT company_id
                      FROM   memberships
                      WHERE  user_id = $1);
$$ LANGUAGE sql STABLE;
```

INLINING SQL SRFS

```
=# EXPLAIN ANALYZE SELECT  *
FROM    visible_sales_slow(1) AS s
WHERE   vendor_id = 5000;
               QUERY PLAN
------------------------------------------------
 Function Scan on visible_sales_slow s
   (cost=0.25..12.75 rows=5 width=56)
   (actual time=57.415..57.670 rows=2 loops=1)
   Filter: (vendor_id = 5000)
   Rows Removed by Filter: 51688
 Planning Time: 0.129 ms
 Execution Time: 57.925 ms
(5 rows)
```

INLINING SQL SRFS

```
=# EXPLAIN ANALYZE SELECT  *
FROM    visible_sales_slow(1) AS s
WHERE   vendor_id = 5000;
              QUERY PLAN
-----------------------------------------------
 Function Scan on visible_sales_slow s
   (cost=0.25..12.75 rows=5 width=56)
   (actual time=57.415..57.670 rows=2 loops=1)
   Filter: (vendor_id = 5000)
   Rows Removed by Filter: 51688
 Planning Time: 0.129 ms
 Execution Time: 57.925 ms
(5 rows)
```

INLINING SQL SRFS

```
=# EXPLAIN ANALYZE SELECT  *
FROM    visible_sales_slow(1) AS s
WHERE   vendor_id = 5000;
               QUERY PLAN
-------------------------------------------------
 Function Scan on visible_sales_slow s
   (cost=0.25..12.75 rows=5 width=56)
   (actual time=57.415..57.670 rows=2 loops=1)
   Filter: (vendor_id = 5000)
   Rows Removed by Filter: 51688
 Planning Time: 0.129 ms
 Execution Time: 57.925 ms
(5 rows)
```

INLINING SQL SRFS

```
=# EXPLAIN ANALYZE SELECT  *
FROM    visible_sales_slow(1) AS s
WHERE   vendor_id = 5000;
               QUERY PLAN
-------------------------------------------------
 Function Scan on visible_sales_slow s
   (cost=0.25..12.75 rows=5 width=56)
   (actual time=57.415..57.670 rows=2 loops=1)
   Filter: (vendor_id = 5000)
   Rows Removed by Filter: 51688
 Planning Time: 0.129 ms
 Execution Time: 57.925 ms
(5 rows)
```

INLINING SQL SRFS

```
=# EXPLAIN ANALYZE SELECT  *
FROM    visible_sales_slow(1) AS s
WHERE   vendor_id = 5000;
               QUERY PLAN
-------------------------------------------------
 Function Scan on visible_sales_slow s
   (cost=0.25..12.75 rows=5 width=56)
   (actual time=57.415..57.670 rows=2 loops=1)
   Filter: (vendor_id = 5000)
   Rows Removed by Filter: 51688
 Planning Time: 0.129 ms
 Execution Time: 57.925 ms
(5 rows)
```

INLINING SQL SRFS

```
Nested Loop
  (cost=0.84..32.53 rows=7 width=27)
  (actual time=0.044..0.048 rows=2 loops=1)
  ->  Index Only Scan using idx_memberships_company_user c
        (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
      (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
```

INLINING SQL SRFS

```
Nested Loop
  (cost=0.84..32.53 rows=7 width=27)
  (actual time=0.044..0.048 rows=2 loops=1)
  ->  Index Only Scan using idx_memberships_company_user
        (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
      (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
```

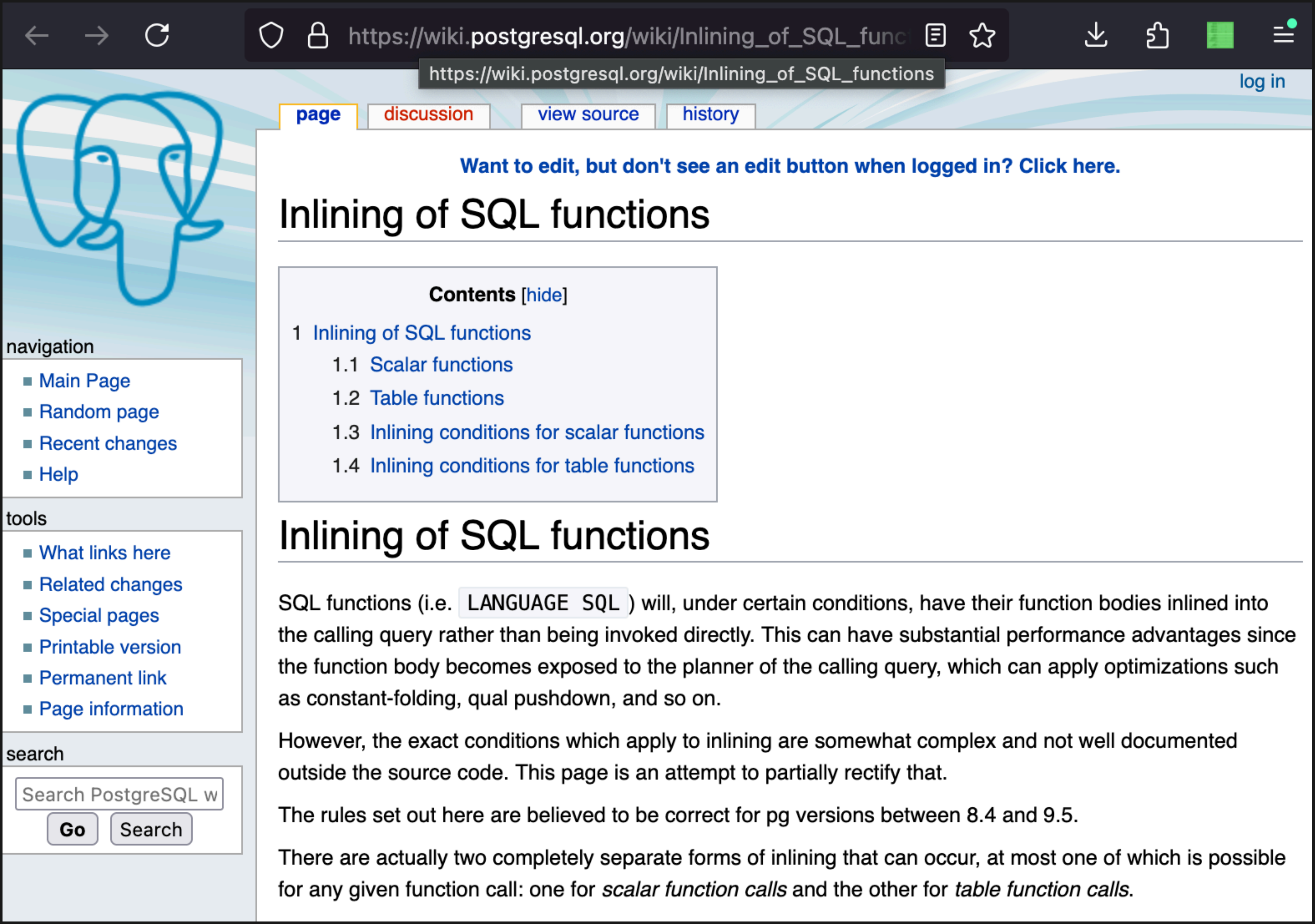
INLINING SQL SRFS

```
Nested Loop
  (cost=0.84..32.53 rows=7 width=27)
  (actual time=0.044..0.048 rows=2 loops=1)
  ->  Index Only Scan using idx_memberships_company_user
        (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
      (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
```

INLINING SQL SRFS

```
Nested Loop
  (cost=0.84..32.53 rows=7 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
      (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
```

INLINING SQL SRFS



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

Inlining of SQL functions

Contents [hide]

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*.

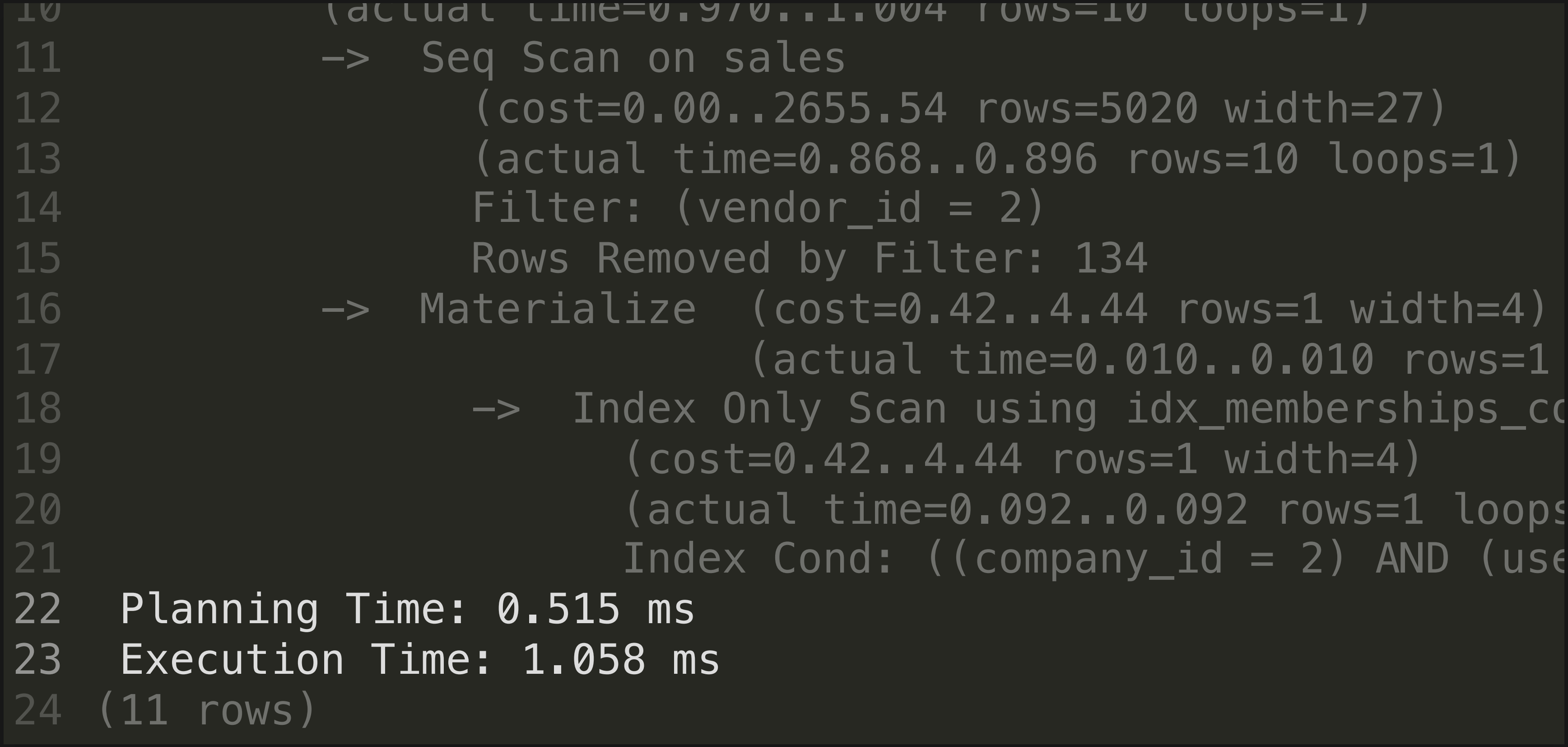
INLINING SQL SRFS

```
=# EXPLAIN (ANALYZE) SELECT  *
FROM    visible_sales(1) AS s
WHERE   vendor_id = 2 LIMIT 10;
                   QUERY PLAN
-----------------------------------------------------
 Limit  (cost=0.42..5.84 rows=10 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
```

INLINING SQL SRFS

```
=# EXPLAIN (ANALYZE) SELECT  *
FROM    visible_sales(1) AS s
WHERE   vendor_id = 2 LIMIT 10;
                   QUERY PLAN
-----------------------------------------------------
 Limit  (cost=0.42..5.84 rows=10 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
```

INLINING SQL SRFS

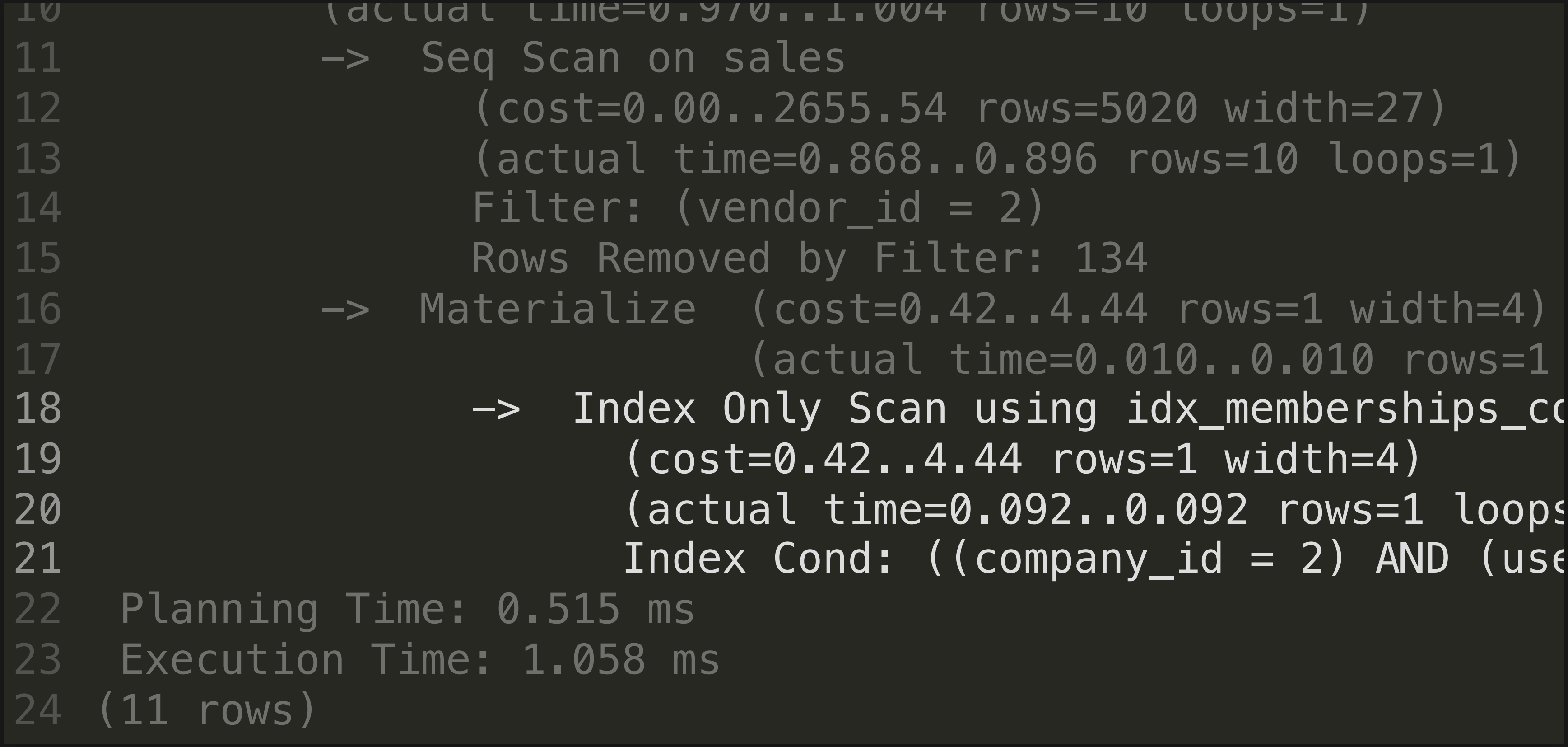


```
         (actual time=0.970..1.004 rows=10 loops=1)
         ->  Seq Scan on sales
               (cost=0.00..2655.54 rows=5020 width=27)
               (actual time=0.868..0.896 rows=10 loops=1)
               Filter: (vendor_id = 2)
               Rows Removed by Filter: 134
         ->  Materialize  (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
 Planning Time: 0.515 ms
 Execution Time: 1.058 ms
(11 rows)
```

INLINING SQL SRFS

```
         (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
         ->  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
 Planning Time: 0.515 ms
 Execution Time: 1.058 ms
(11 rows)
```

INLINING SQL SRFS



```
         (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
         ->  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
 Planning Time: 0.515 ms
 Execution Time: 1.058 ms
(11 rows)
```

inline_set_returning_function

```c
Query *inline_set_returning_function(...) {
  /* Fetch the function body */
  Datum tmp = SysCacheGetAttrNotNull(PROCOID, func_tuple,
  char *src = TextDatumGetCString(tmp);

  /* Set up to handle parameters */
  SQLFunctionParseInfoPtr pinfo = prepare_sql_fn_parse_inf
    func_tuple, (Node *) fexpr, fexpr->inputcollid);

  /* Parse, analyze, and rewrite */
  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);
```

inline_set_returning_function

```c
Query *inline_set_returning_function(...) {
  /* Fetch the function body */
  Datum tmp = SysCacheGetAttrNotNull(PROCOID, func_tuple,
  char *src = TextDatumGetCString(tmp);

  /* Set up to handle parameters */
  SQLFunctionParseInfoPtr pinfo = prepare_sql_fn_parse_info
    func_tuple, (Node *) fexpr, fexpr->inputcollid);

  /* Parse, analyze, and rewrite */
  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);
```

inline_set_returning_function

```c
Query *inline_set_returning_function(...) {
  /* Fetch the function body */
  Datum tmp = SysCacheGetAttrNotNull(PROCOID, func_tuple, A
  char *src = TextDatumGetCString(tmp);

  /* Set up to handle parameters */
  SQLFunctionParseInfoPtr pinfo = prepare_sql_fn_parse_infc
    func_tuple, (Node *) fexpr, fexpr->inputcollid);

  /* Parse, analyze, and rewrite */
  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);
```

inline_set_returning_function

```

/* Set up to handle parameters */
SQLFunctionParseInfoPtr pinfo = prepare_sql_fn_parse_info
  func_tuple, (Node *) fexpr, fexpr->inputcollid);

/* Parse, analyze, and rewrite */
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);
```

inline_set_returning_function

```c
    func_tuple, (Node *) fexpr, fexpr->inputcollid);

  /* Parse, analyze, and rewrite */
  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);

  return querytree;
}
```

inline_set_returning_function

```c
    func_tuple, (Node *) fexpr, fexpr->inputcollid);

  /* Parse, analyze, and rewrite */
  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);

  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;
```

INLINING NON-SQL

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

INLINING NON-SQL

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

INLINING NON-SQL

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

SUPPORT PROCS

```
=# \d pg_proc
                   Table "pg_catalog.pg_proc"
     Column      |     Type     | Collation | Nullable | De
-----------------+--------------+-----------+----------+---
 oid             | oid          |           | not null |
 proname         | name         |           | not null |
 pronamespace    | oid          |           | not null |
 proowner        | oid          |           | not null |
 prolang         | oid          |           | not null |
 procost         | real         |           | not null |
 prorows         | real         |           | not null |
 provariadic     | oid          |           | not null |
 prosupport      | regproc      |           | not null |
 prokind         | "char"       |           | not null |
 prosecdef       | boolean      |           | not null |
```

SUPPORT PROCS

```
proname         | name         |           | not null |
pronamespace    | oid          |           | not null |
proowner        | oid          |           | not null |
prolang         | oid          |           | not null |
procost         | real         |           | not null |
prorows         | real         |           | not null |
provariadic     | oid          |           | not null |
prosupport      | regproc      |           | not null |
prokind         | "char"       |           | not null |
prosecdef       | boolean      |           | not null |
proleakproof    | boolean      |           | not null |
proisstrict     | boolean      |           | not null |
proretset       | boolean      |           | not null |
provolatile     | "char"       |           | not null |
proparallel     | "char"       |           | not null |
```

SUPPORT REQUESTS

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

INLINING NON-SQL

```
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

```sql
CREATE OR REPLACE FUNCTION commission_cents(
  _sale_id INTEGER, _salesperson_id INTEGER
)
RETURNS INTEGER
AS $$
  SELECT  total_price_cents * COALESCE(commission_percent,
  FROM    sales AS s
  LEFT JOIN memberships AS m
  ON      m.company_id = s.vendor_id
  AND     m.user_id = _salesperson_id
  WHERE   s.id = _sale_id;
$$ LANGUAGE sql STABLE
SUPPORT commission_cents_support;
```

commission_cents_support

```sql
CREATE OR REPLACE FUNCTION commission_cents(
  _sale_id INTEGER, _salesperson_id INTEGER
)
RETURNS INTEGER
AS $$
  SELECT  total_price_cents * COALESCE(commission_percent,
  FROM    sales AS s
  LEFT JOIN memberships AS m
  ON      m.company_id = s.vendor_id
  AND     m.user_id = _salesperson_id
  WHERE   s.id = _sale_id;
$$ LANGUAGE sql STABLE
SUPPORT commission_cents_support;
```

commission_cents_support

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

commission_cents_support

```
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;

  ...
```

commission_cents_support

```
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;

  ...
```

commission_cents_support

```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;

  ...
```

commission_cents_support

```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;


  ...
```

commission_cents_support

```
typedef struct SupportRequestSimplify
{
    NodeTag     type;

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

commission_cents_support

```c
Node *node = lsecond(expr->args);
if (IsA(node, Const)) {
  Const *c = (Const *) node;
  if (c->constisnull) {
    Const *ret = makeConst(
      INT4OID,          /* type */
      -1,               /* typmod */
      0,                /* collid */
      4,                /* len */
      Int32GetDatum(0), /* value */
      false,            /* isnull */
      true              /* byval */
    );
    PG_RETURN_POINTER(ret);
  }
```

commission_cents_support

```c
Node *node = lsecond(expr->args);
if (IsA(node, Const)) {
  Const *c = (Const *) node;
  if (c->constisnull) {
    Const *ret = makeConst(
      INT4OID,          /* type */
      -1,               /* typmod */
      0,                /* collid */
      4,                /* len */
      Int32GetDatum(0), /* value */
      false,            /* isnull */
      true              /* byval */
    );
    PG_RETURN_POINTER(ret);
  }
```

commission_cents_support

```c
Node *node = lsecond(expr->args);
if (IsA(node, Const)) {
  Const *c = (Const *) node;
  if (c->constisnull) {
    Const *ret = makeConst(
      INT4OID,          /* type */
      -1,               /* typmod */
      0,                /* collid */
      4,                /* len */
      Int32GetDatum(0), /* value */
      false,            /* isnull */
      true              /* byval */
    );
    PG_RETURN_POINTER(ret);
  }
```

commission_cents_support

```c
Node *node = lsecond(expr->args);
if (IsA(node, Const)) {
  Const *c = (Const *) node;
  if (c->constisnull) {
    Const *ret = makeConst(
      INT4OID,          /* type */
      -1,               /* typmod */
      0,                /* collid */
      4,                /* len */
      Int32GetDatum(0), /* value */
      false,            /* isnull */
      true              /* byval */
    );
    PG_RETURN_POINTER(ret);
  }
```

commission_cents_support

```c
  Const *c = (Const *) node;
  if (c->constisnull) {
    Const *ret = makeConst(
      INT4OID,          /* type */
      -1,               /* typmod */
      0,                /* collid */
      4,                /* len */
      Int32GetDatum(0), /* value */
      false,            /* isnull */
      true              /* byval */
    );
    PG_RETURN_POINTER(ret);
  }
}
PG_RETURN_POINTER(NULL);
```

THERE'S A PATCH FOR THAT!

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

SupportRequestInlineSRF

```c
char *sql = "....";
List *parsed = pg_parse_query(sql);
List *analyzed = pg_analyze_and_rewrite_with_cb(
  linitial(parsed),
  sql,
  (ParserSetupHook) sql_fn_parser_setup,
  pinfo,
  NULL);
Query *q = linitial(analyzed);
PG_RETURN_POINTER(q);
```

SupportRequestInlineSRF

```c
char *sql = "....";
List *parsed = pg_parse_query(sql);
List *analyzed = pg_analyze_and_rewrite_with_cb(
  linitial(parsed),
  sql,
  (ParserSetupHook) sql_fn_parser_setup,
  pinfo,
  NULL);
Query *q = linitial(analyzed);
PG_RETURN_POINTER(q);
```

SupportRequestInlineSRF

```
char *sql = "....";
List *parsed = pg_parse_query(sql);
List *analyzed = pg_analyze_and_rewrite_with_cb(
  linitial(parsed),
  sql,
  (ParserSetupHook) sql_fn_parser_setup,
  pinfo,
  NULL);
Query *q = linitial(analyzed);
PG_RETURN_POINTER(q);
```

SupportRequestInlineSRF

```
char *sql = "....";
List *parsed = pg_parse_query(sql);
List *analyzed = pg_analyze_and_rewrite_with_cb(
  linitial(parsed),
  sql,
  (ParserSetupHook) sql_fn_parser_setup,
  pinfo,
  NULL);
Query *q = linitial(analyzed);
PG_RETURN_POINTER(q);
```

SupportRequestInlineSRF

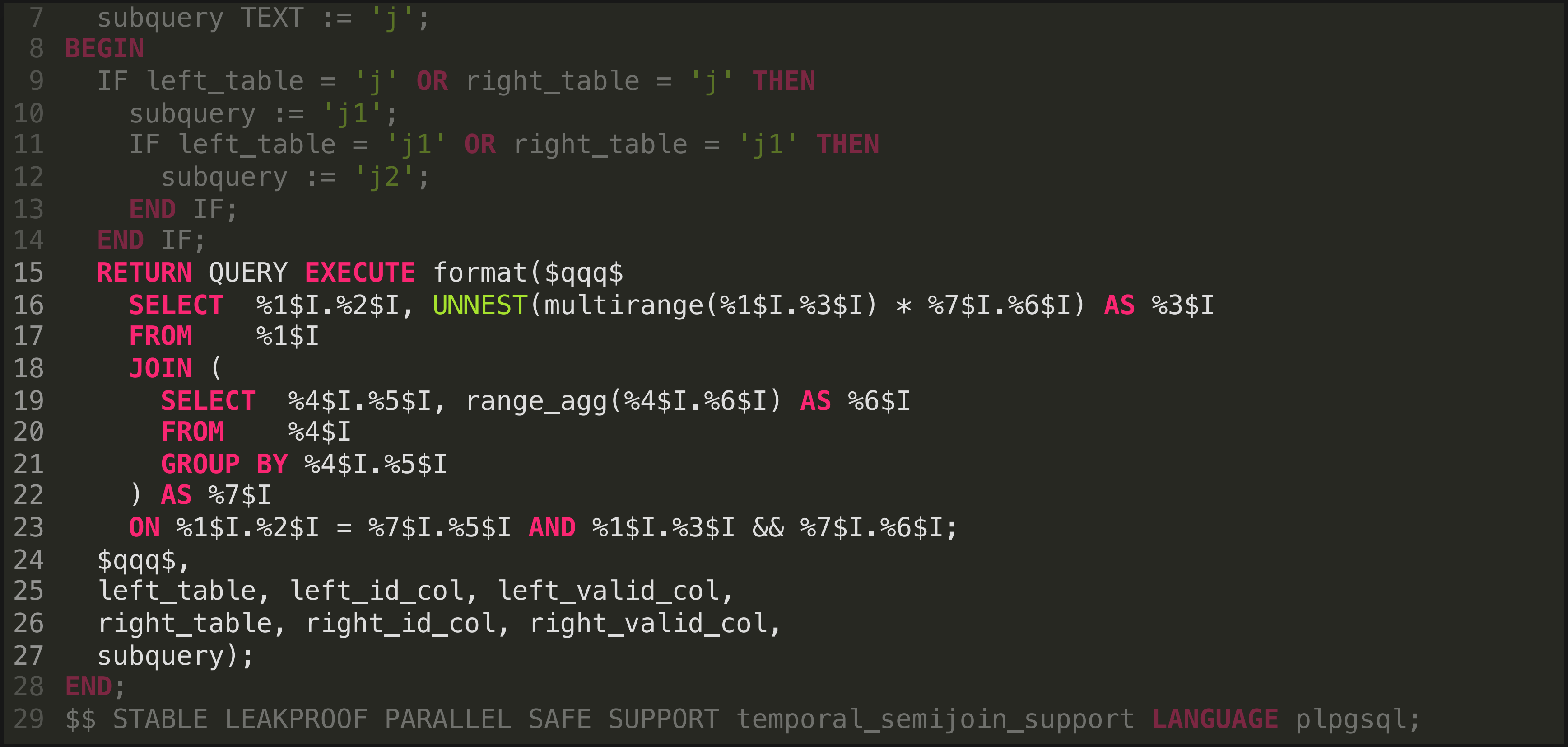
```c
char *sql = "....";
List *parsed = pg_parse_query(sql);
List *analyzed = pg_analyze_and_rewrite_with_cb(
  linitial(parsed),
  sql,
  (ParserSetupHook) sql_fn_parser_setup,
  pinfo,
  NULL);
Query *q = linitial(analyzed);
PG_RETURN_POINTER(q);
```

CAVEATS

GENERALIZING

```sql
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($qqq$
    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;
```

GENERALIZING



```
  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($qqq$
    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;
  $qqq$,
  left_table, left_id_col, left_valid_col,
  right_table, right_id_col, right_valid_col,
  subquery);
END;
$$ STABLE LEAKPROOF PARALLEL SAFE SUPPORT temporal_semijoin_support LANGUAGE plpgsql;
```

BIBLIOGRAPHY

- https://github.com/pjungwir/inlining-postgres-functio
- 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-postgres-functions