INLINING POSTGRES 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

```
 1 =# EXPLAIN ANALYZE SELECT  *
 2 FROM    visible_sales_slow(1) AS s
 3 WHERE   vendor_id = 5000;
 4                 QUERY PLAN
 5 ------------------------------------------------------
 6  Function Scan on visible_sales_slow s
 7    (cost=0.25..12.75 rows=5 width=56)
 8    (actual time=57.415..57.670 rows=2 loops=1)
 9    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)
```

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

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

INLINING SQL SRFS

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

INLINING SQL SRFS

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

INLINING SQL SRFS

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

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

```
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)
15                Rows Removed by Filter: 134
16          ->  Materialize  (cost=0.42..4.44 rows=1 width=4)
17                           (actual time=0.010..0.010 rows=1
18                ->  Index Only Scan using idx_memberships_co
19                      (cost=0.42..4.44 rows=1 width=4)
20                      (actual time=0.092..0.092 rows=1 loops
21                      Index Cond: ((company_id = 2) AND (use
22  Planning Time: 0.515 ms
23  Execution Time: 1.058 ms
24 (11 rows)
```

INLINING SQL SRFS

```
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)
15                  Rows Removed by Filter: 134
16            ->  Materialize  (cost=0.42..4.44 rows=1 width=4)
17                             (actual time=0.010..0.010 rows=1
18                  ->  Index Only Scan using idx_memberships_co
19                        (cost=0.42..4.44 rows=1 width=4)
20                        (actual time=0.092..0.092 rows=1 loops
21                        Index Cond: ((company_id = 2) AND (use
22  Planning Time: 0.515 ms
23  Execution Time: 1.058 ms
24 (11 rows)
```

INLINING SQL SRFS

```
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)
15                Rows Removed by Filter: 134
16          ->  Materialize  (cost=0.42..4.44 rows=1 width=4)
17                           (actual time=0.010..0.010 rows=1
18                ->  Index Only Scan using idx_memberships_co
19                      (cost=0.42..4.44 rows=1 width=4)
20                      (actual time=0.092..0.092 rows=1 loops
21                      Index Cond: ((company_id = 2) AND (use
22  Planning Time: 0.515 ms
23  Execution Time: 1.058 ms
24 (11 rows)
```

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-SRFS

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

```
 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  prosecdef        | boolean      |          | not null |
16  proleakproof     | boolean      |          | not null |
17  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

INLINING NON-SRFS

```
SELECT  total_price_cents,
        commission_cents(id, salesperson_id)
FROM    sales
WHERE   salesperson_id = $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
 3 )
 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

```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

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

commission_cents_support

```
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   ...
```

commission_cents_support

```
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   ...
```

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

```
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   ...
```

commission_cents_support

```
typedef struct SupportRequestSimplify
{
    NodeTag     type;

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

commission_cents_support

```
 1 Node *node = lsecond(expr->args);
 2 if (IsA(node, Const)) {
 3   Const *c = (Const *) node;
 4   if (c->constisnull) {
 5     Const *ret = makeConst(
 6       INT4OID,           /* type */
 7       -1,                /* typmod */
 8       0,                 /* collid */
 9       4,                 /* len */
10       Int32GetDatum(0), /* value */
11       false,             /* isnull */
12       true               /* byval */
13     );
14     PG_RETURN_POINTER(ret);
15   }
```

commission_cents_support

```
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

```
Node *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

```
3   Const *c = (Const *) node;
4   if (c->constisnull) {
5     Const *ret = makeConst(
6       INT4OID,            /* type */
7       -1,                 /* typmod */
8       0,                  /* collid */
9       4,                  /* len */
10      Int32GetDatum(0), /* value */
11      false,              /* isnull */
12      true                /* byval */
13    );
14    PG_RETURN_POINTER(ret);
15  }
16 }
17 PG_RETURN_POINTER(NULL);
```

THERE'S A PATCH FOR THAT!

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

SupportRequestInlineSRF

```
 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);
```

SupportRequestInlineSRF

```
 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);
```

SupportRequestInlineSRF

```
 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);
```

SupportRequestInlineSRF

```
 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);
```

SupportRequestInlineSRF

```
 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

```
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