POSTGRES PIPELINE: ESPECIALLY EMPHASIZING EXECUTION

by Paul Jungwirth

Illuminated Computing

May 2023

PHASES

- Parsing
- Analysis
- Rewriting
- Planning & Optimizing
- Executing

FOR EXAMPLE

```
ALTER TABLE ADD PERIOD valid_at (valid_from, valid_til)

PRIMARY KEY/UNIQUE (id, valid_at WITHOUT OVERLAPS)

UPDATE/DELETE FROM t
  FOR PORTION OF valid_at
  FROM '2020-01-01' TO '2030-01-01'

FOREIGN KEY (id, PERIOD valid_at)
  REFERENCES parent (id, PERIOD valid_at)
```

FOR EXAMPLE

```
ALTER TABLE ADD PERIOD valid_at (valid_from, valid_til)

PRIMARY KEY/UNIQUE (id, valid_at WITHOUT OVERLAPS)

UPDATE/DELETE FROM t
  FOR PORTION OF valid_at
  FROM '2020-01-01' TO '2030-01-01'

FOREIGN KEY (id, PERIOD valid_at)
  REFERENCES parent (id, PERIOD valid_at)
```

FOR EXAMPLE

```
ALTER TABLE ADD PERIOD valid_at (valid_from, valid_til)

PRIMARY KEY/UNIQUE (id, valid_at WITHOUT OVERLAPS)

UPDATE/DELETE FROM t
  FOR PORTION OF valid_at
  FROM '2020-01-01' TO '2030-01-01'

FOREIGN KEY (id, PERIOD valid_at)
  REFERENCES parent (id, PERIOD valid_at)
```

FOR EXAMPLE

```
ALTER TABLE ADD PERIOD valid_at (valid_from, valid_til)

PRIMARY KEY/UNIQUE (id, valid_at WITHOUT OVERLAPS)

UPDATE/DELETE FROM t
  FOR PORTION OF valid_at
  FROM '2020-01-01' TO '2030-01-01'

FOREIGN KEY (id, PERIOD valid_at)
  REFERENCES parent (id, PERIOD valid_at)
```

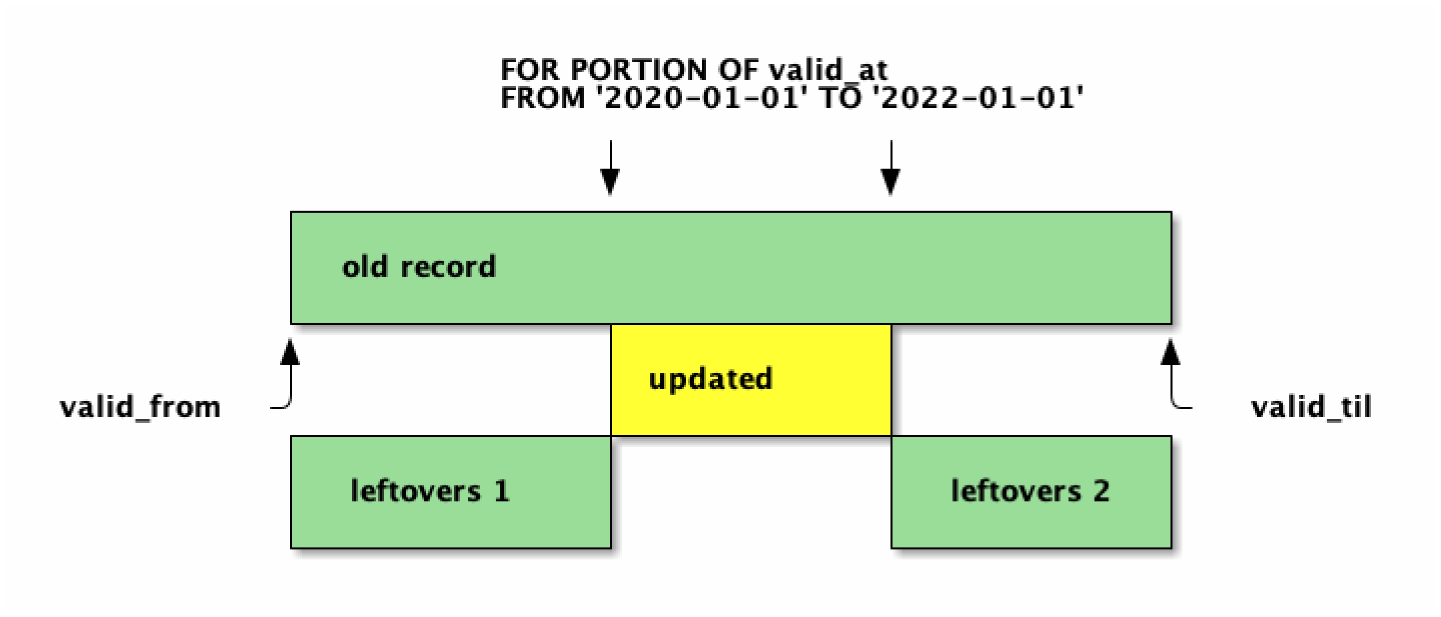
FOR EXAMPLE

```
ALTER TABLE ADD PERIOD valid_at (valid_from, valid_til)

PRIMARY KEY/UNIQUE (id, valid_at WITHOUT OVERLAPS)

UPDATE/DELETE FROM t
  FOR PORTION OF valid_at
  FROM '2020-01-01' TO '2030-01-01'

FOREIGN KEY (id, PERIOD valid_at)
  REFERENCES parent (id, PERIOD valid_at)
```

FOR EXAMPLE

```
ALTER TABLE ADD PERIOD valid_at (valid_from, valid_til)

PRIMARY KEY/UNIQUE (id, valid_at WITHOUT OVERLAPS)

UPDATE/DELETE FROM t
  FOR PORTION OF valid_at
  FROM '2020-01-01' TO '2030-01-01'

FOREIGN KEY (id, PERIOD valid_at)
  REFERENCES parent (id, PERIOD valid_at)
```

FOR EXAMPLE


FOR PORTION OF valid_at
FROM '2020-01-01' TO '2022-01-01'
old record
updated
valid_from
valid_til
leftovers 1
leftovers 2

WHY

```
On Tue, Nov 14, 2017 at 9:43 AM Tom Lane <tgl@sss.pgh.pa.us> w
>
> Robert is correct that putting this into the parser
> is completely the wrong thing.
> If you do that, then for example views using the features
> will reverse-list in the rewritten form,
> which we Do Not Want,
> even if the rewritten form is completely valid SQL
> (is it?).
>
> . . .
>
>       regards, tom lane
```

TCOP/POSTGRES.C

```
exec_simple_query
  pg_parse_query
  pg_analyze_and_rewrite
   parse_analyze
   pg_rewrite_query
     QueryRewrite
       RewriteQuery
  pg_plan_queries
  PortalDefineQuery
  PortalStart
    ExecutorStart
      ExecInitModifyTable
  PortalRun
    FillPortalStore
    PortalRunSelect
      ExecutorRun
        ExecModifyTable
```

TCOP/POSTGRES.C

```
exec_simple_query
  pg_parse_query
  pg_analyze_and_rewrite
   parse_analyze
   pg_rewrite_query
     QueryRewrite
       RewriteQuery
  pg_plan_queries
  PortalDefineQuery
  PortalStart
    ExecutorStart
      ExecInitModifyTable
  PortalRun
    FillPortalStore
    PortalRunSelect
      ExecutorRun
        ExecModifyTable
```

TCOP/POSTGRES.C

```
exec_simple_query
  pg_parse_query
  pg_analyze_and_rewrite
   parse_analyze
   pg_rewrite_query
     QueryRewrite
       RewriteQuery
  pg_plan_queries
  PortalDefineQuery
  PortalStart
    ExecutorStart
      ExecInitModifyTable
  PortalRun
    FillPortalStore
    PortalRunSelect
      ExecutorRun
        ExecModifyTable
```

TCOP/POSTGRES.C

```
exec_simple_query
  pg_parse_query
  pg_analyze_and_rewrite
   parse_analyze
   pg_rewrite_query
     QueryRewrite
       RewriteQuery
  pg_plan_queries
  PortalDefineQuery
  PortalStart
    ExecutorStart
      ExecInitModifyTable
  PortalRun
    FillPortalStore
    PortalRunSelect
      ExecutorRun
        ExecModifyTable
```

TCOP/POSTGRES.C

```
exec_simple_query
  pg_parse_query
  pg_analyze_and_rewrite
   parse_analyze
   pg_rewrite_query
     QueryRewrite
       RewriteQuery
  pg_plan_queries
  PortalDefineQuery
  PortalStart
    ExecutorStart
      ExecInitModifyTable
  PortalRun
    FillPortalStore
    PortalRunSelect
      ExecutorRun
        ExecModifyTable
```

TCOP/POSTGRES.C

```
exec_simple_query
  pg_parse_query
  pg_analyze_and_rewrite
   parse_analyze
   pg_rewrite_query
     QueryRewrite
       RewriteQuery
  pg_plan_queries
  PortalDefineQuery
  PortalStart
    ExecutorStart
      ExecInitModifyTable
  PortalRun
    FillPortalStore
    PortalRunSelect
      ExecutorRun
        ExecModifyTable
```

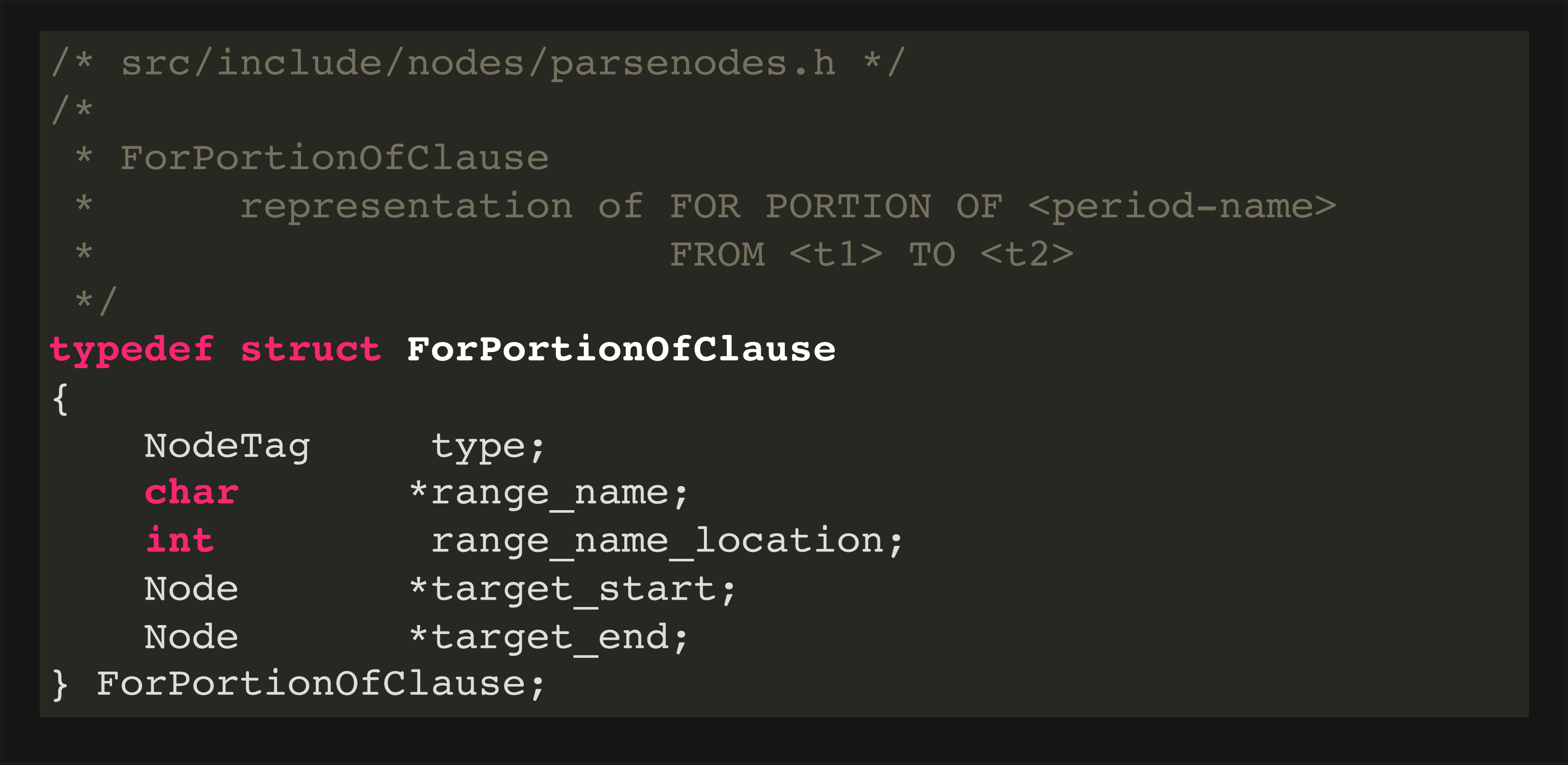
TCOP/POSTGRES.C

```
exec_simple_query
  pg_parse_query
  pg_analyze_and_rewrite
   parse_analyze
   pg_rewrite_query
     QueryRewrite
       RewriteQuery
  pg_plan_queries
  PortalDefineQuery
  PortalStart
    ExecutorStart
      ExecInitModifyTable
  PortalRun
    FillPortalStore
    PortalRunSelect
      ExecutorRun
        ExecModifyTable
```

PARSING

```
/* src/backend/parser/gram.y */

for_portion_of_clause:
    FOR PORTION OF ColId FROM a_expr TO a_expr
      {
        ForPortionOfClause *n = makeNode(ForPortionOfClause);
        n->range_name = $4;
        n->range_name_location = @4;
        n->target_start = $6;
        n->target_end = $8;
        $$ = n;
      }
    | /*EMPTY*/                 { $$ = NULL; }
  ;
```

PARSE NODES

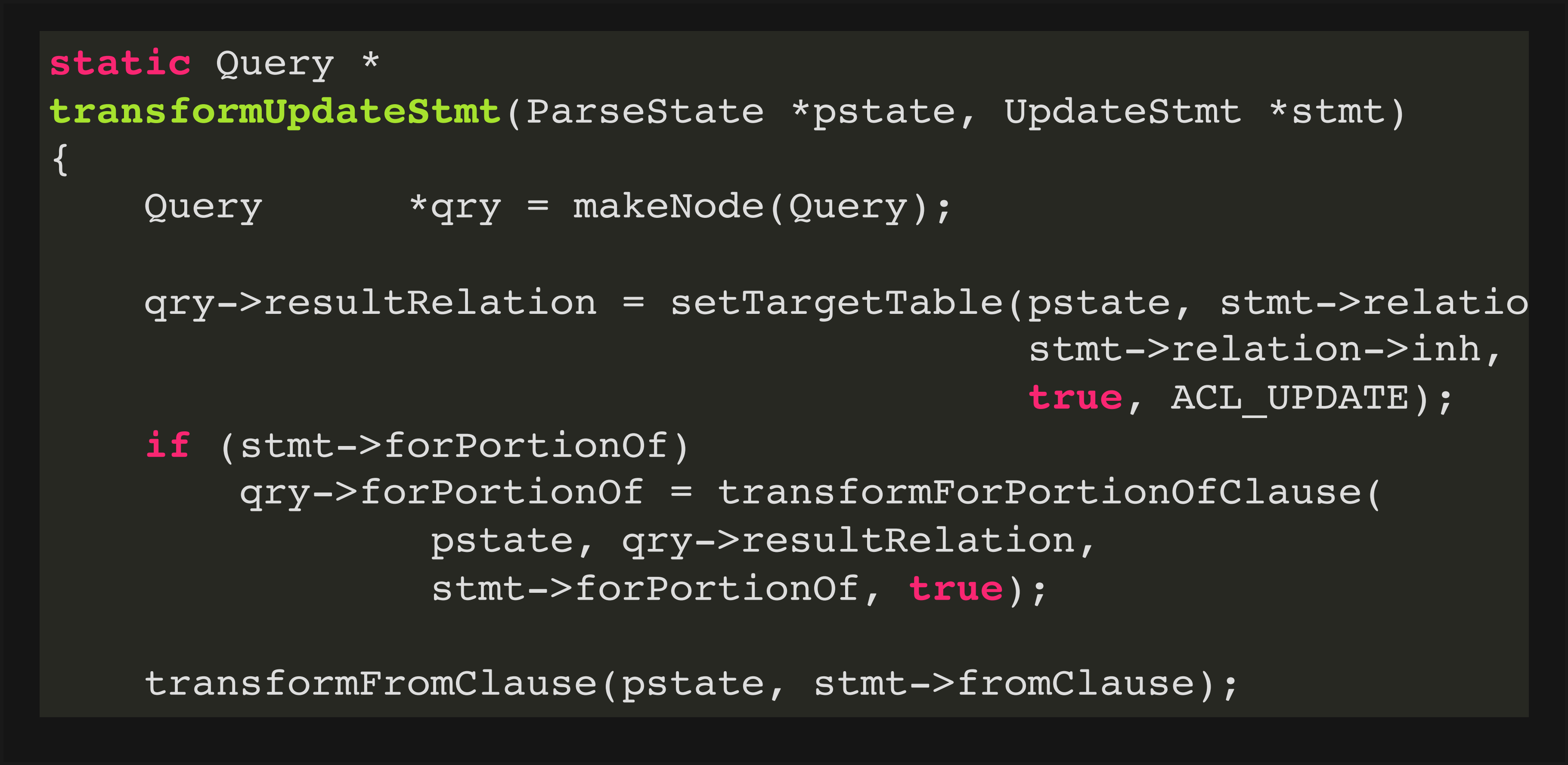


```
/* src/include/nodes/parsenodes.h */
/*
 * ForPortionOfClause
 *      representation of FOR PORTION OF <period-name>
 *                        FROM <t1> TO <t2>
 */
typedef struct ForPortionOfClause
{
    NodeTag      type;
    char        *range_name;
    int          range_name_location;
    Node        *target_start;
    Node        *target_end;
} ForPortionOfClause;
```

MEMORY CONTEXTS

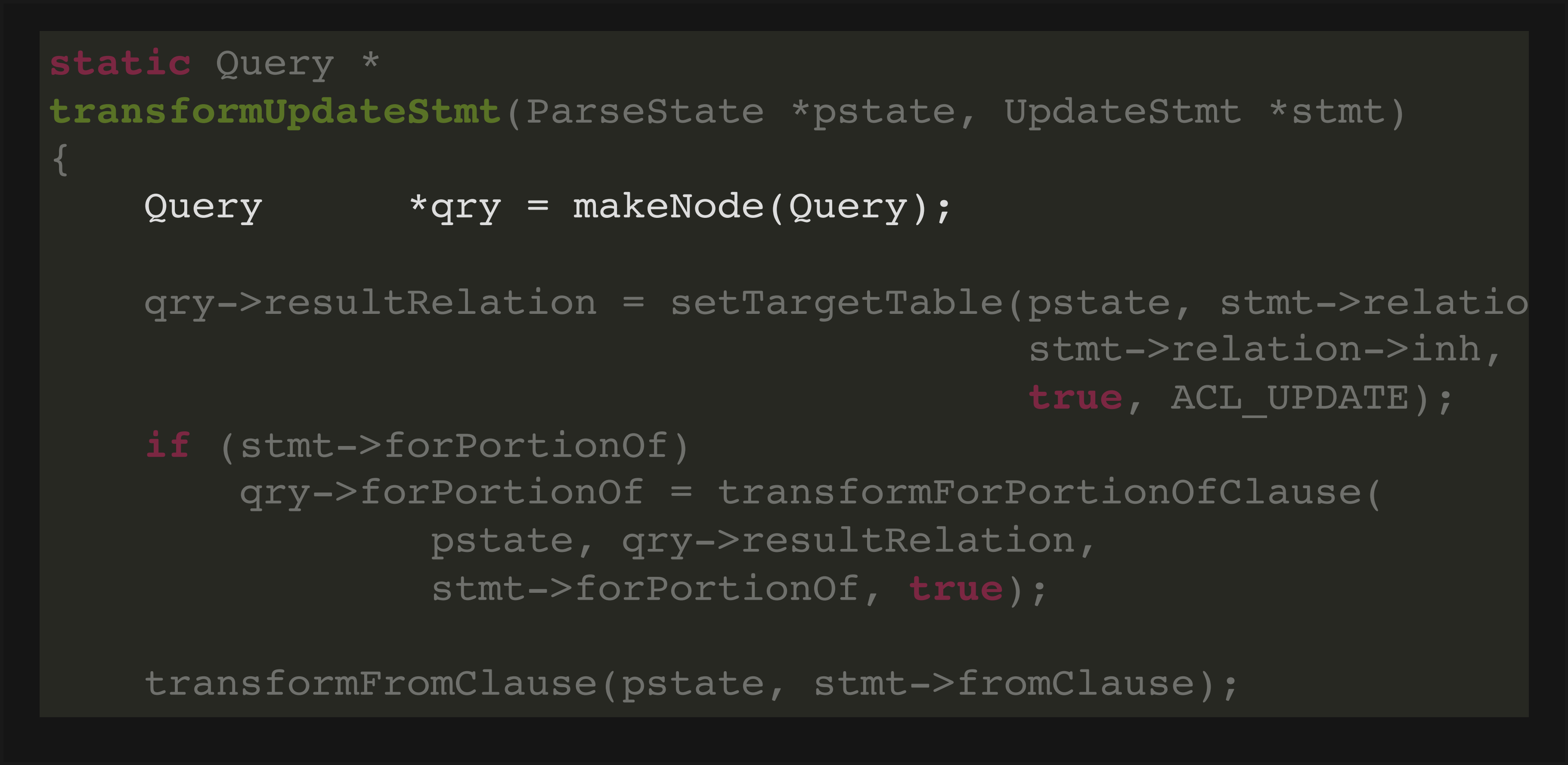
```
TopMemoryContext
PostmasterContext
CacheMemoryContext
MessageContext
TopTransactionContext
CurTransactionContext
PortalContext
ErrorContext
and more!
```

ANALYSIS



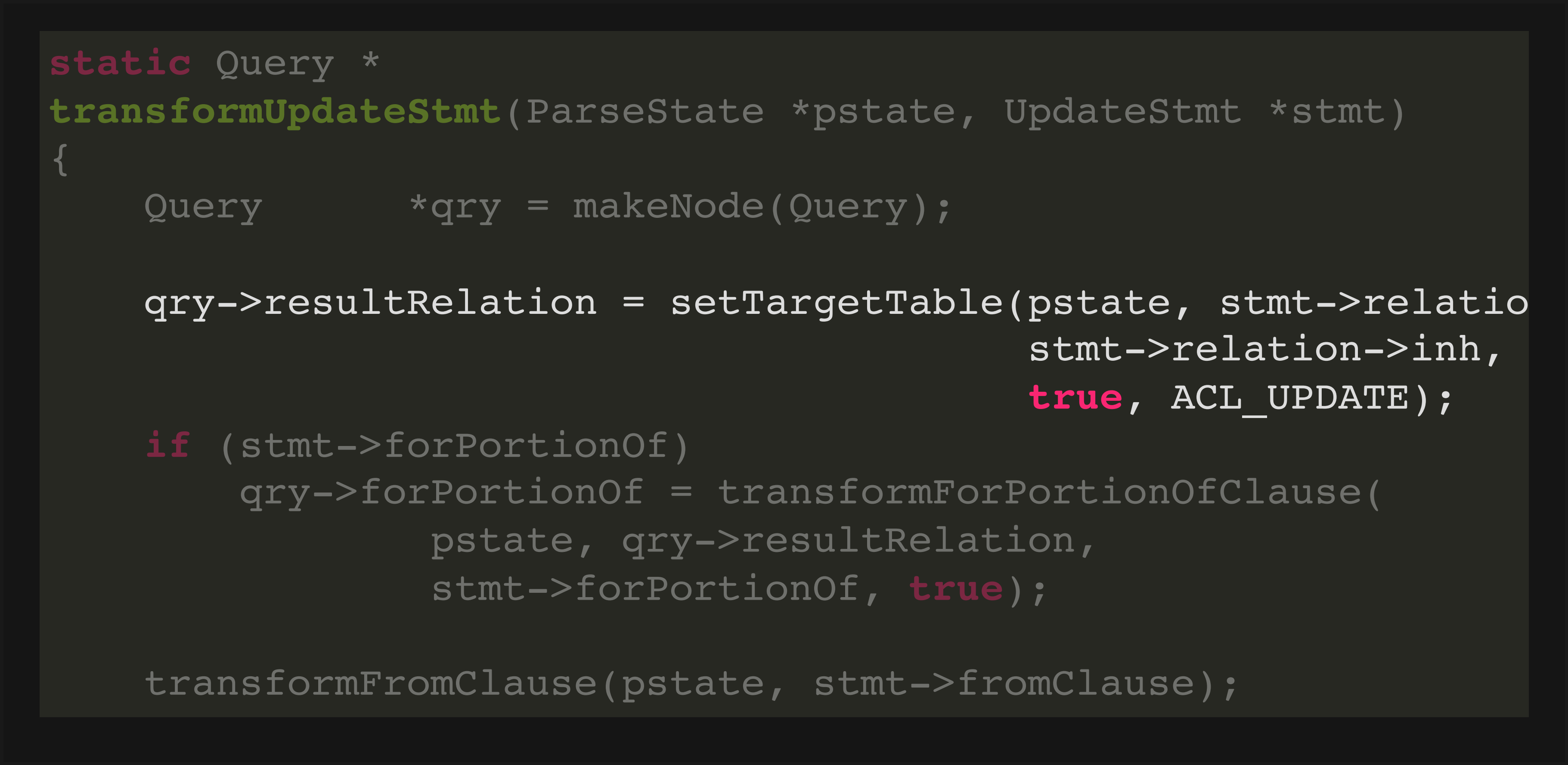
```
static Query *
transformUpdateStmt(ParseState *pstate, UpdateStmt *stmt)
{
    Query      *qry = makeNode(Query);

    qry->resultRelation = setTargetTable(pstate, stmt->relatio
                                         stmt->relation->inh,
                                         true, ACL_UPDATE);
    if (stmt->forPortionOf)
        qry->forPortionOf = transformForPortionOfClause(
                pstate, qry->resultRelation,
                stmt->forPortionOf, true);

    transformFromClause(pstate, stmt->fromClause);
```

ANALYSIS



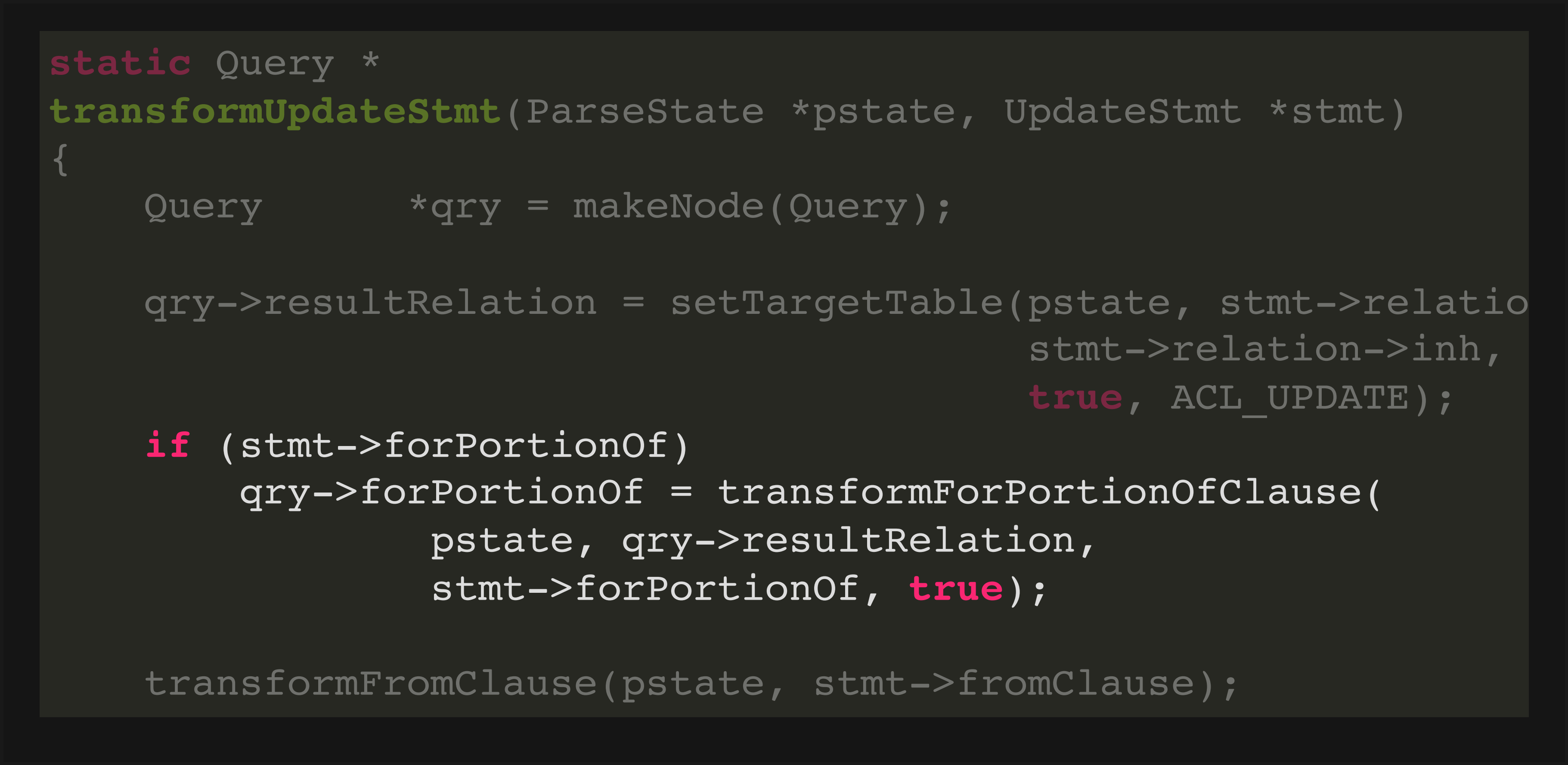
```
static Query *
transformUpdateStmt(ParseState *pstate, UpdateStmt *stmt)
{
    Query      *qry = makeNode(Query);

    qry->resultRelation = setTargetTable(pstate, stmt->relatio
                                         stmt->relation->inh,
                                         true, ACL_UPDATE);
    if (stmt->forPortionOf)
        qry->forPortionOf = transformForPortionOfClause(
                pstate, qry->resultRelation,
                stmt->forPortionOf, true);

    transformFromClause(pstate, stmt->fromClause);
```

ANALYSIS



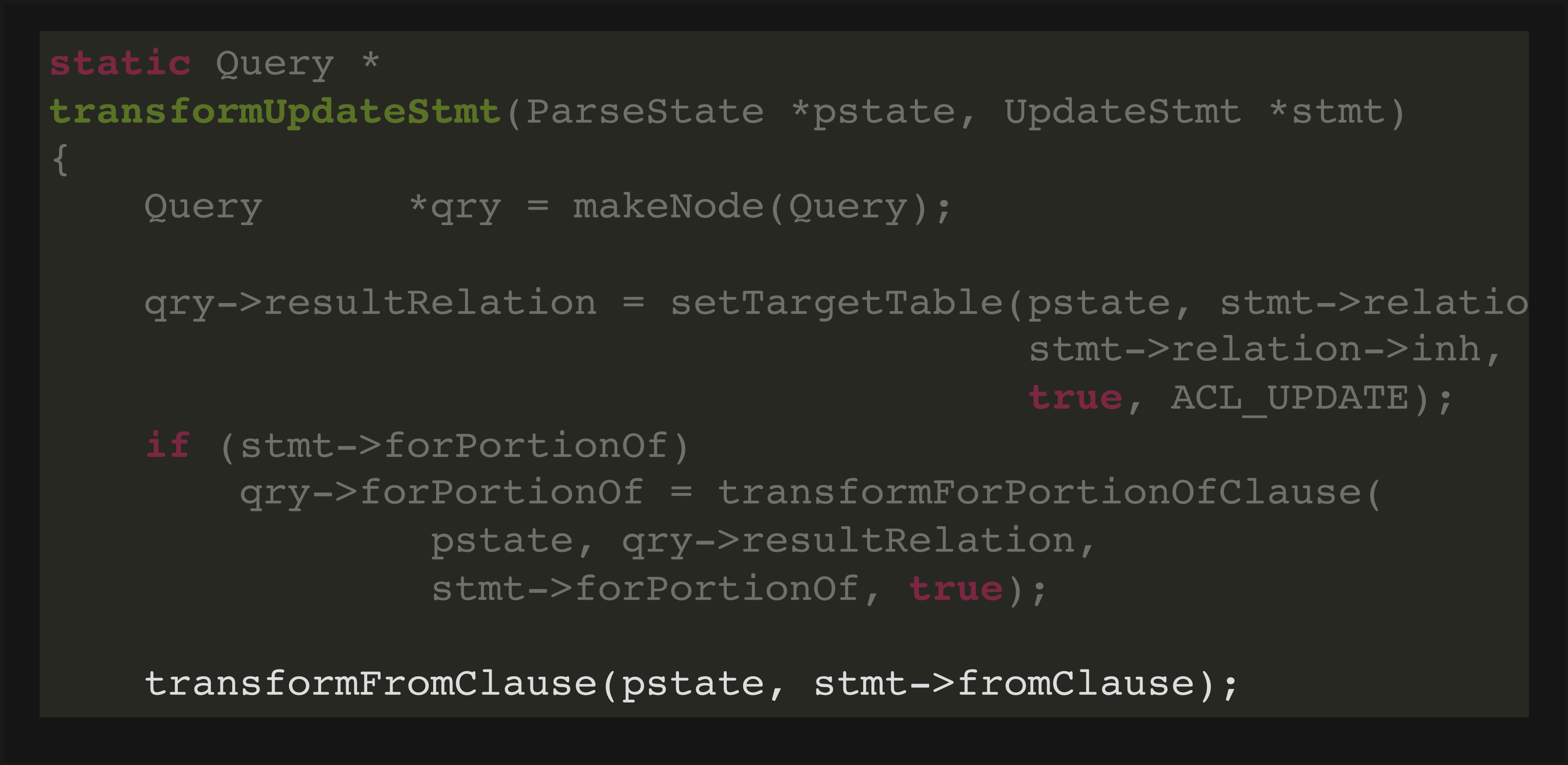
```
static Query *
transformUpdateStmt(ParseState *pstate, UpdateStmt *stmt)
{
    Query      *qry = makeNode(Query);

    qry->resultRelation = setTargetTable(pstate, stmt->relatio
                                         stmt->relation->inh,
                                         true, ACL_UPDATE);
    if (stmt->forPortionOf)
        qry->forPortionOf = transformForPortionOfClause(
                pstate, qry->resultRelation,
                stmt->forPortionOf, true);

    transformFromClause(pstate, stmt->fromClause);
```

ANALYSIS



```
static Query *
transformUpdateStmt(ParseState *pstate, UpdateStmt *stmt)
{
    Query      *qry = makeNode(Query);

    qry->resultRelation = setTargetTable(pstate, stmt->relatio
                                         stmt->relation->inh,
                                         true, ACL_UPDATE);
    if (stmt->forPortionOf)
        qry->forPortionOf = transformForPortionOfClause(
                pstate, qry->resultRelation,
                stmt->forPortionOf, true);

    transformFromClause(pstate, stmt->fromClause);
```

ANALYSIS

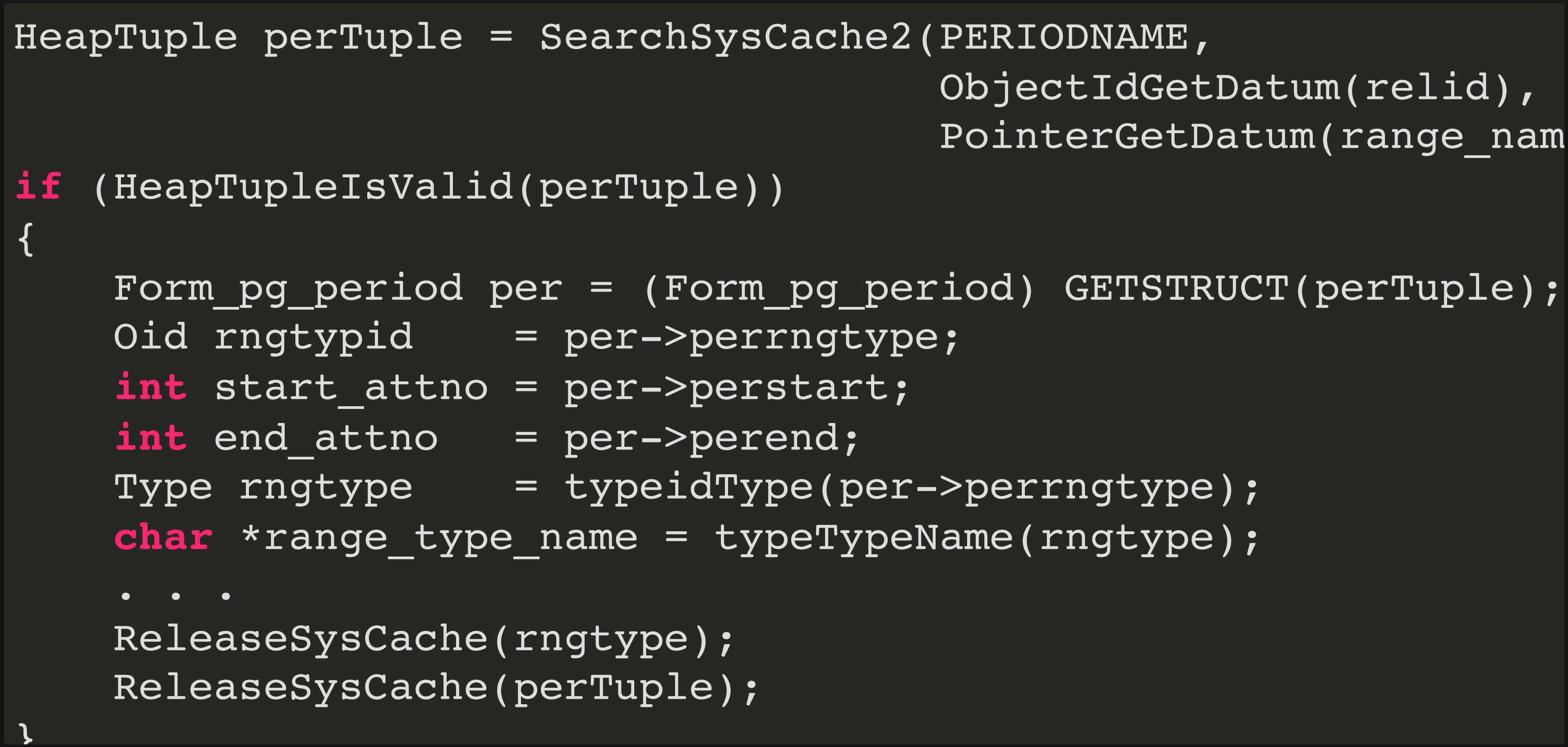


```
static Query *
transformUpdateStmt(ParseState *pstate, UpdateStmt *stmt)
{
    Query      *qry = makeNode(Query);

    qry->resultRelation = setTargetTable(pstate, stmt->relatio
                                         stmt->relation->inh,
                                         true, ACL_UPDATE);
    if (stmt->forPortionOf)
        qry->forPortionOf = transformForPortionOfClause(
                pstate, qry->resultRelation,
                stmt->forPortionOf, true);

    transformFromClause(pstate, stmt->fromClause);
```

RANGETBLENTRY

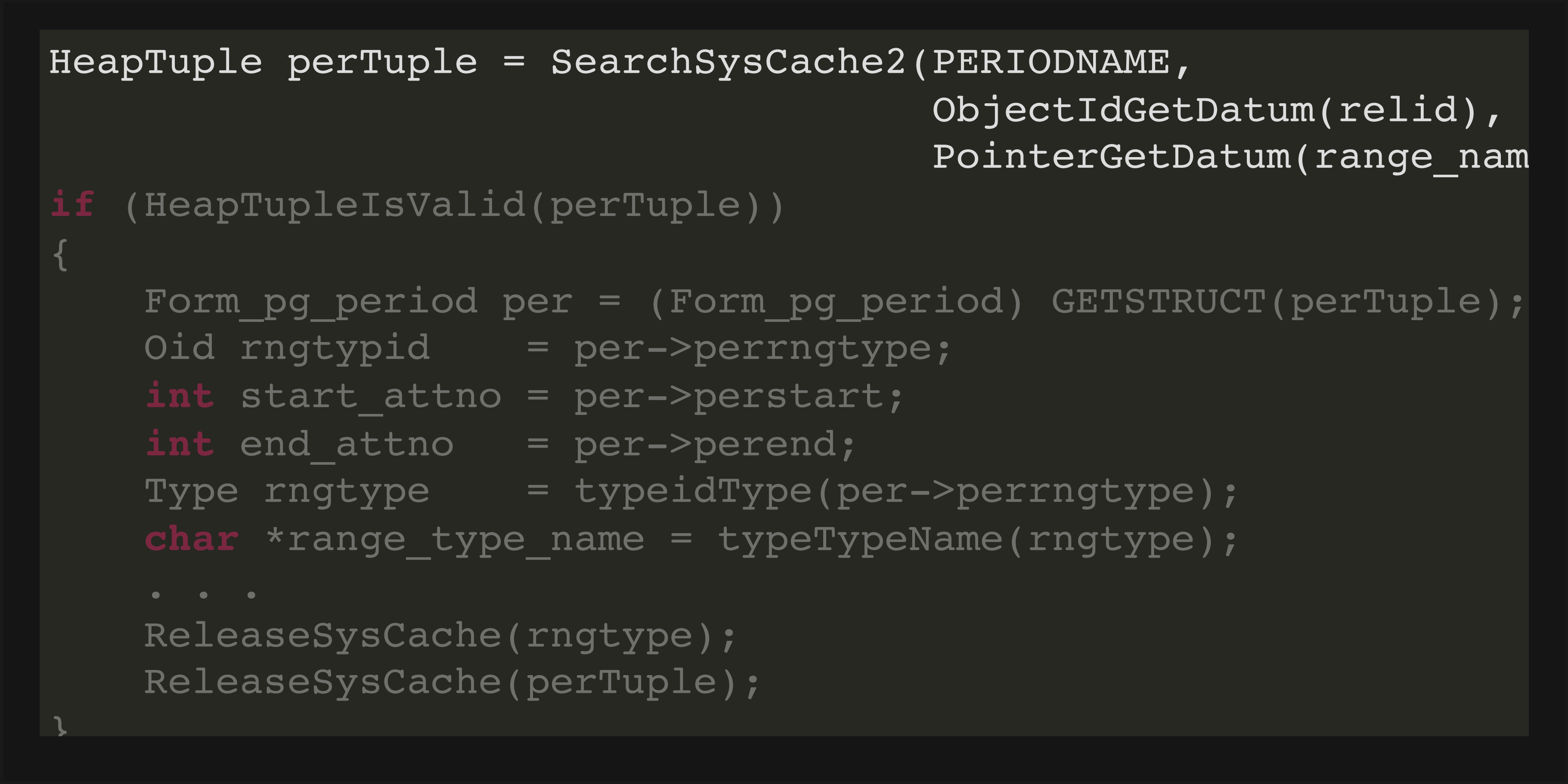
```
typedef struct RangeTblEntry
{
    NodeTag     type;

    RTEKind     rtekind;        /* see above */
    Oid         relid;          /* OID of the relation */
    char        relkind;        /* relation kind (see pg_class
    int         rellockmode;    /* lock level that query requi
    Index       perminfoindex;
    Query      *subquery;       /* the sub-query */
    /* . . . */
```

SYSCACHE



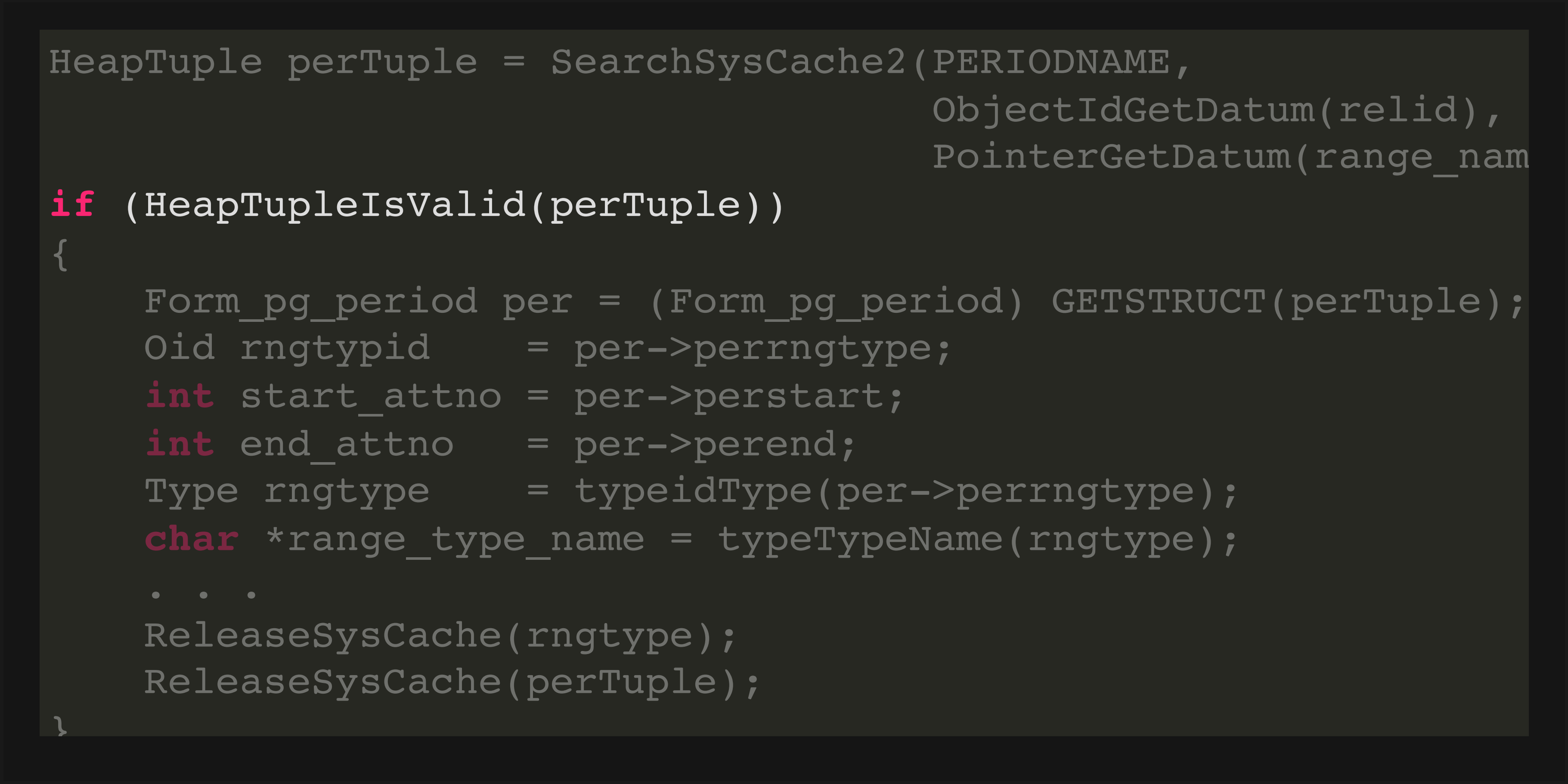
```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

SYSCACHE



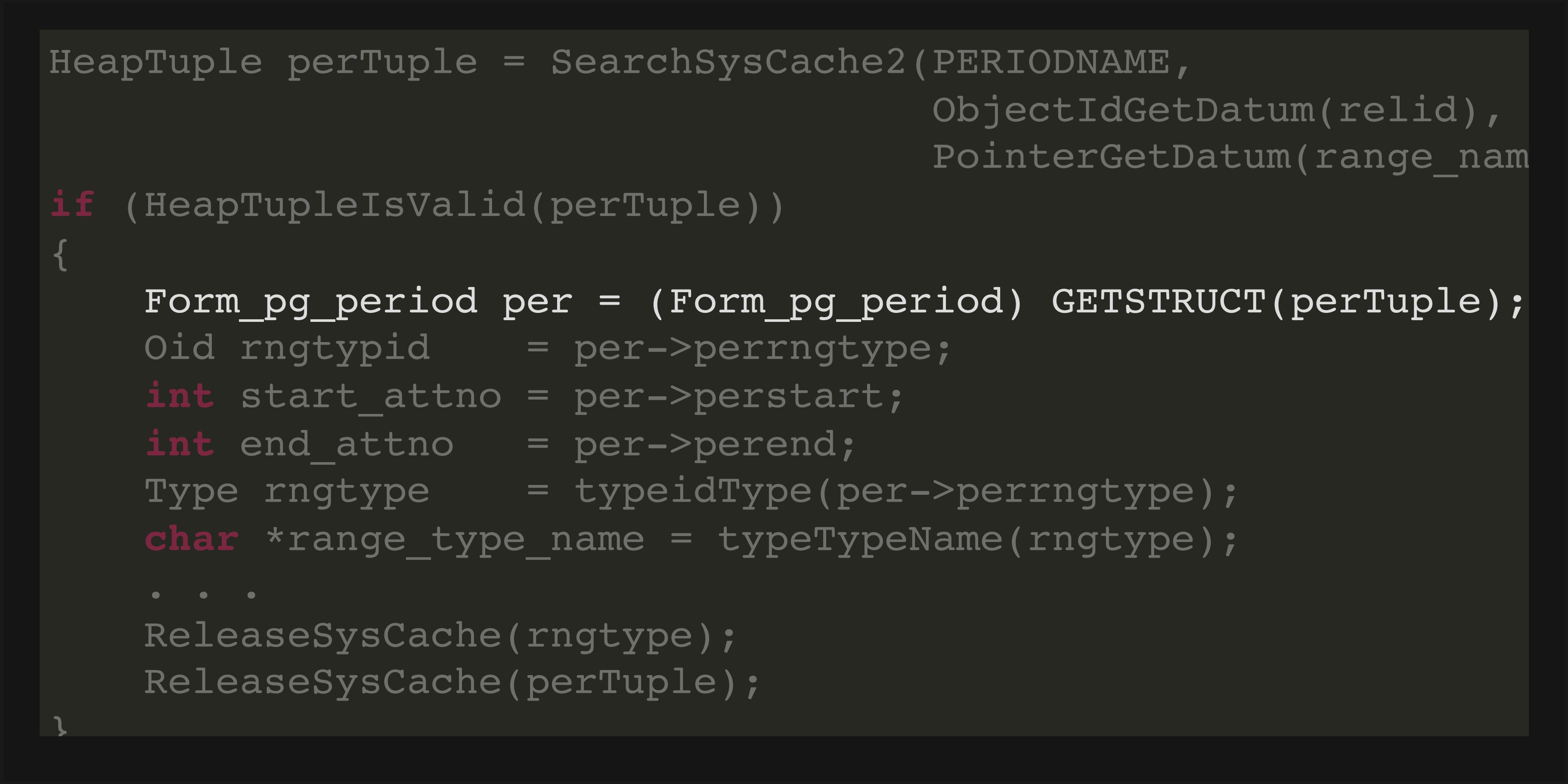
```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

SYSCACHE



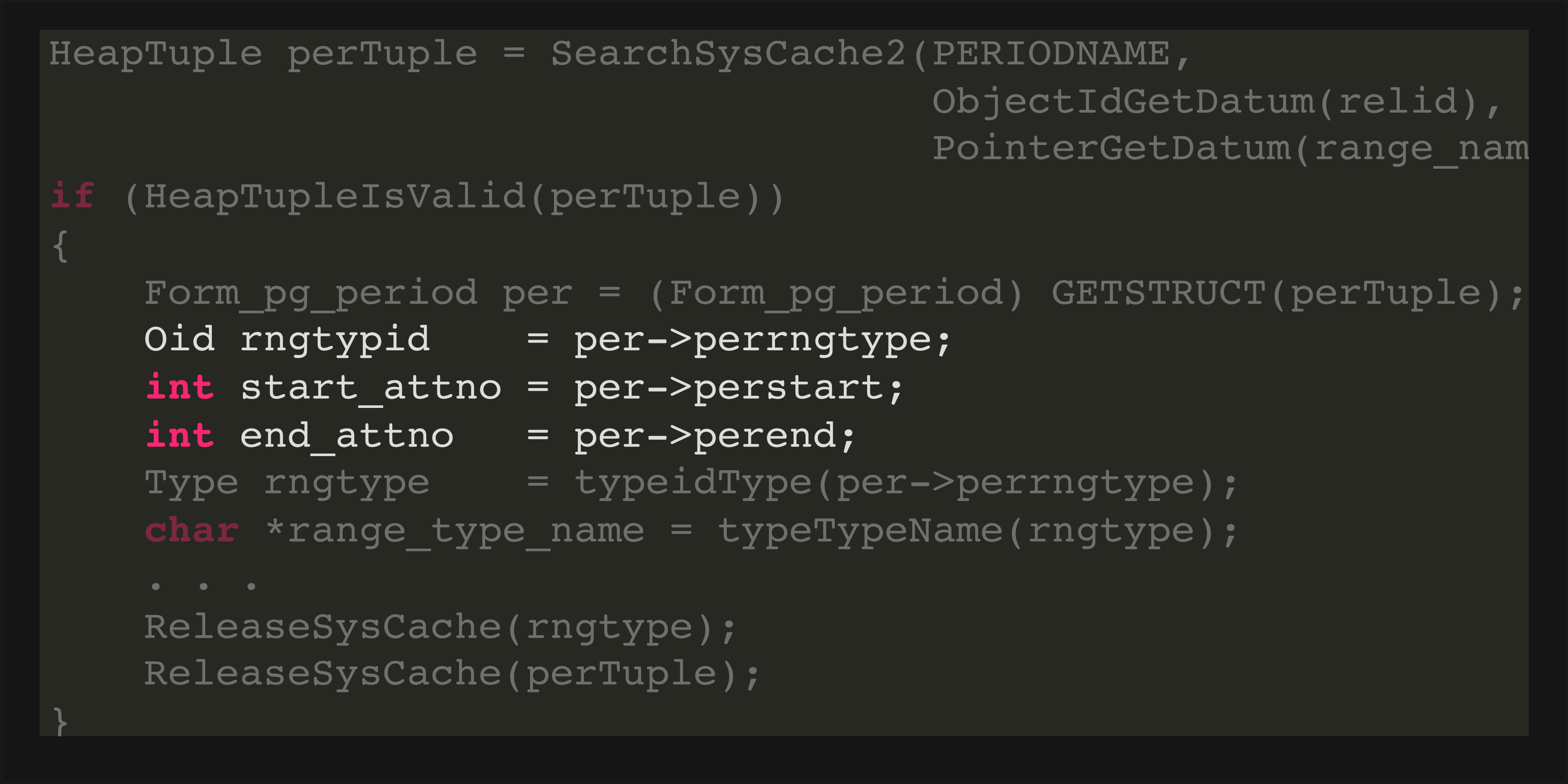
```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

SYSCACHE



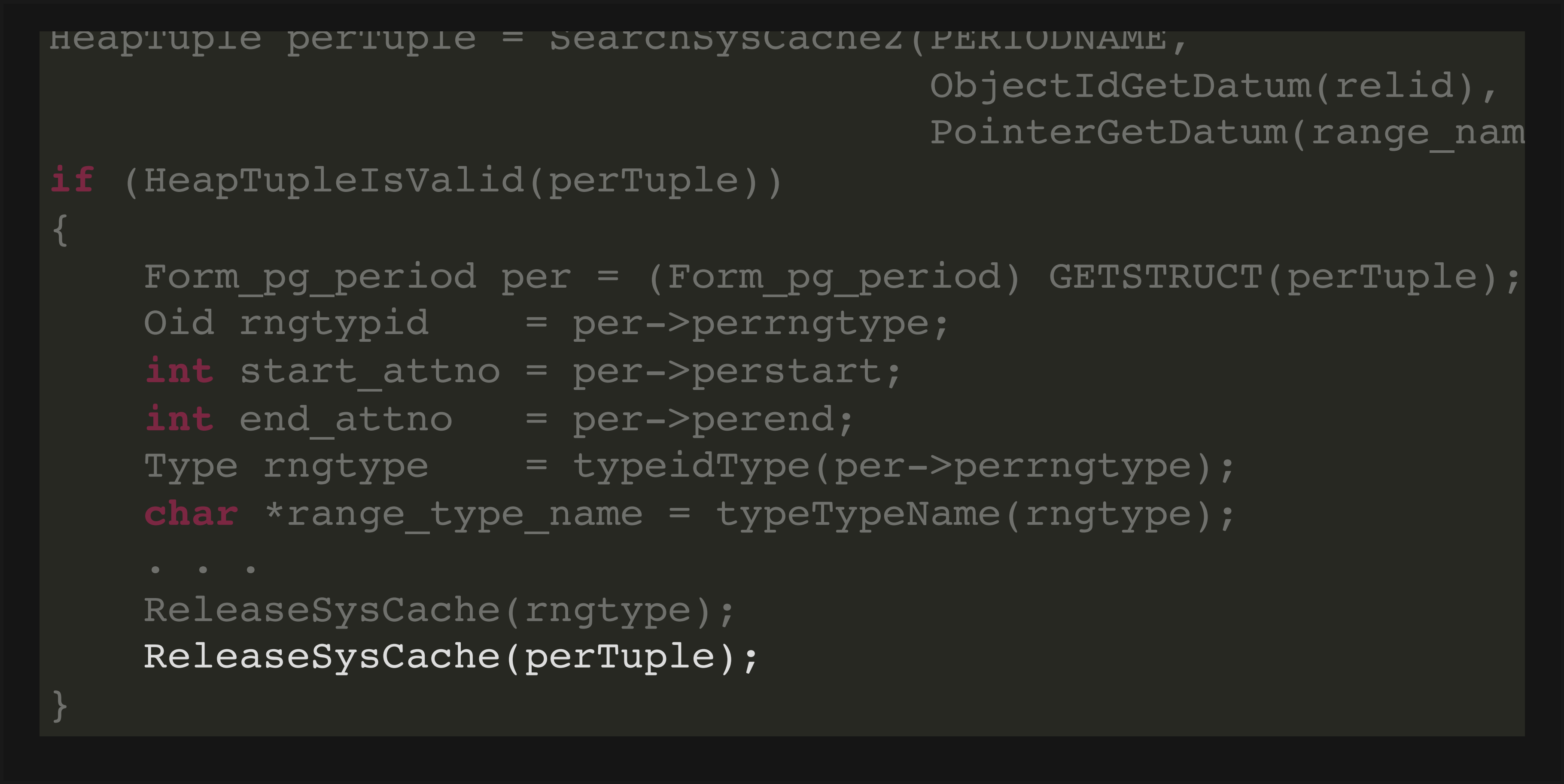
```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

SYSCACHE



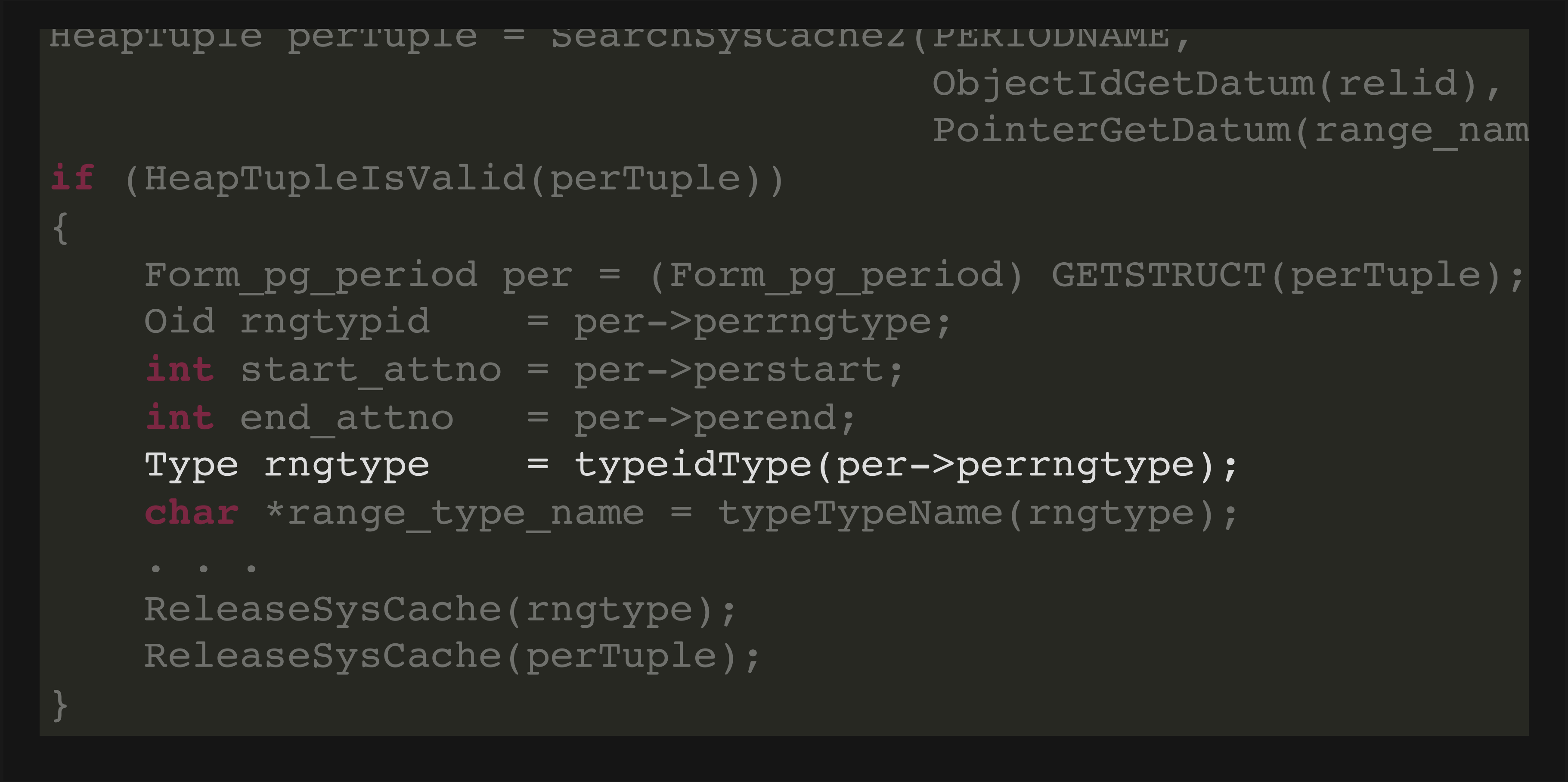
```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

SYSCACHE



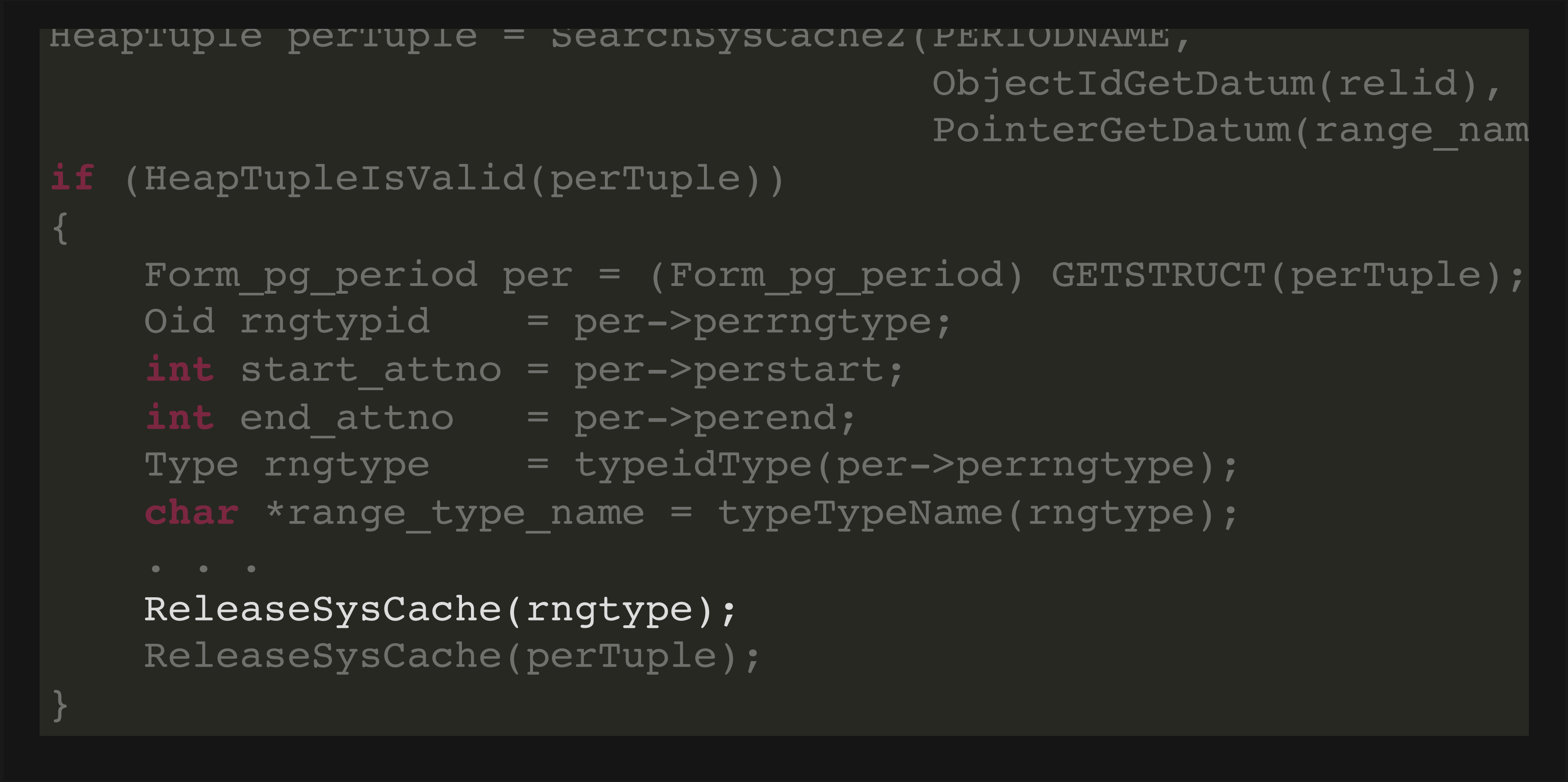
```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

SYSCACHE



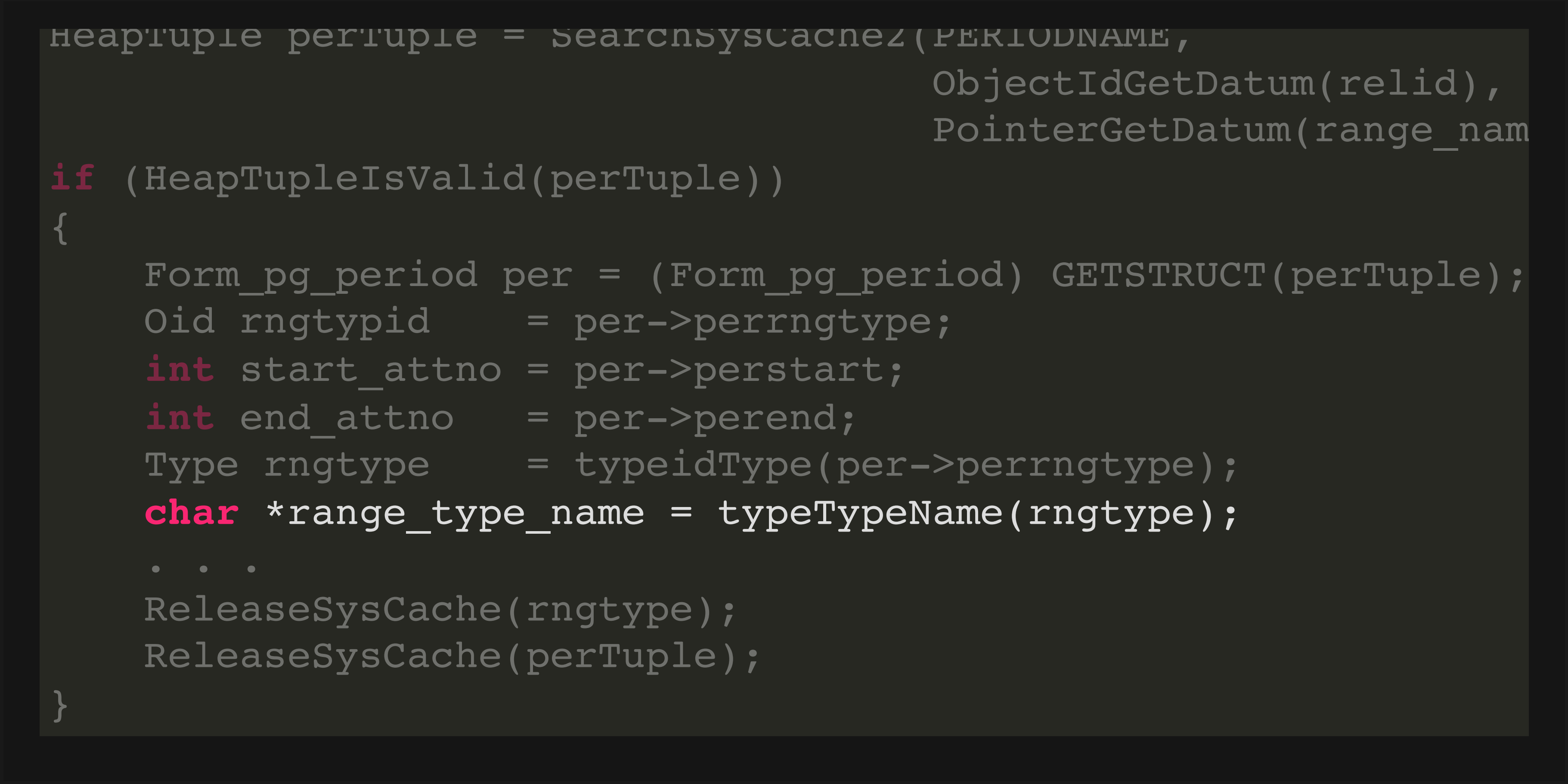
```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

SYSCACHE



```c
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

SYSCACHE



```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    Type rngtype    = typeidType(per->perrngtype);
    char *range_type_name = typeTypeName(rngtype);
    . . .
    ReleaseSysCache(rngtype);
    ReleaseSysCache(perTuple);
}
```

LSYSCACHE

```
char *get_periodname(Oid periodid, bool missing_ok) {
 HeapTuple tp = SearchSysCache1(PERIODOID,
                                ObjectIdGetDatum(periodid));
 if (HeapTupleIsValid(tp)) {
   Form_pg_period period_tup = (Form_pg_period) GETSTRUCT(tp);
   char *result = pstrdup(NameStr(period_tup->pername));
   ReleaseSysCache(tp);
   return result;
 }

 if (!missing_ok)
   elog(ERROR, "cache lookup failed for period %d", periodid);
 return NULL;
}
```

TYPCACHE

```
RangeType *r = DatumGetRangeTypeP(src->fp_targetRange);
TypeCacheEntry *typcache =
    lookup_type_cache(RangeTypeGetOid(r),
                      TYPECACHE_RANGE_INFO);
dst->fp_targetRange = datumCopy(src->fp_targetRange,
                                typcache->typbyval,
                                typcache->typlen);
```

TYPCACHE

```
RangeType *r = DatumGetRangeTypeP(src->fp_targetRange);
TypeCacheEntry *typcache =
    lookup_type_cache(RangeTypeGetOid(r),
                      TYPECACHE_RANGE_INFO);
dst->fp_targetRange = datumCopy(src->fp_targetRange,
                                typcache->typbyval,
                                typcache->typlen);
```

TYPCACHE

```
RangeType *r = DatumGetRangeTypeP(src->fp_targetRange);
TypeCacheEntry *typcache =
    lookup_type_cache(RangeTypeGetOid(r),
                      TYPECACHE_RANGE_INFO);
dst->fp_targetRange = datumCopy(src->fp_targetRange,
                                typcache->typbyval,
                                typcache->typlen);
```

ANALYSIS

```
Node *target_start = transformForPortionOfBound(
        forPortionOf->target_start, true);
Node *target_end   = transformForPortionOfBound(
        forPortionOf->target_end, false);
FuncCall *fc = makeFuncCall(SystemFuncName(range_type_name),
              list_make2(target_start, target_end),
              COERCE_EXPLICIT_CALL,
              forPortionOf->range_name_location);
result->targetRange = transformExpr(
        pstate, (Node *) fc, EXPR_KIND_UPDATE_PORTION);
```

ANALYSIS

```
Node *target_start = transformForPortionOfBound(
        forPortionOf->target_start, true);
Node *target_end   = transformForPortionOfBound(
        forPortionOf->target_end, false);
FuncCall *fc = makeFuncCall(SystemFuncName(range_type_name),
              list_make2(target_start, target_end),
              COERCE_EXPLICIT_CALL,
              forPortionOf->range_name_location);
result->targetRange = transformExpr(
        pstate, (Node *) fc, EXPR_KIND_UPDATE_PORTION);
```

ANALYSIS

```
Node *target_start = transformForPortionOfBound(
        forPortionOf->target_start, true);
Node *target_end   = transformForPortionOfBound(
        forPortionOf->target_end, false);
FuncCall *fc = makeFuncCall(SystemFuncName(range_type_name),
              list_make2(target_start, target_end),
              COERCE_EXPLICIT_CALL,
              forPortionOf->range_name_location);
result->targetRange = transformExpr(
        pstate, (Node *) fc, EXPR_KIND_UPDATE_PORTION);
```

ANALYSIS

```
Node *target_start = transformForPortionOfBound(
        forPortionOf->target_start, true);
Node *target_end   = transformForPortionOfBound(
        forPortionOf->target_end, false);
FuncCall *fc = makeFuncCall(SystemFuncName(range_type_name),
              list_make2(target_start, target_end),
              COERCE_EXPLICIT_CALL,
              forPortionOf->range_name_location);
result->targetRange = transformExpr(
        pstate, (Node *) fc, EXPR_KIND_UPDATE_PORTION);
```

ANALYSIS

```
Expr *rangeSetExpr = (Expr *) makeSimpleA_Expr(
        AEXPR_OP, "*", (Node *) copyObject(rangeExpr),
        (Node *) fc, forPortionOf->range_name_location);
rangeSetExpr = (Expr *) transformExpr(
        pstate, (Node *) rangeSetExpr, EXPR_KIND_UPDATE_PORTIO

TargetEntry *tle = makeTargetEntry(
        rangeSetExpr, range_attno, range_name, false);
targetList = lappend(targetList, tle);

/* Mark the range column as requiring update permissions */
target_perminfo->updatedCols =
        bms_add_member(target_perminfo->updatedCols,
                       range_attno - FirstLowInvalidHeapAttrib
```

ANALYSIS

```
Expr *rangeSetExpr = (Expr *) makeSimpleA_Expr(
        AEXPR_OP, "*", (Node *) copyObject(rangeExpr),
        (Node *) fc, forPortionOf->range_name_location);
rangeSetExpr = (Expr *) transformExpr(
        pstate, (Node *) rangeSetExpr, EXPR_KIND_UPDATE_PORTIO

TargetEntry *tle = makeTargetEntry(
        rangeSetExpr, range_attno, range_name, false);
targetList = lappend(targetList, tle);

/* Mark the range column as requiring update permissions */
target_perminfo->updatedCols =
        bms_add_member(target_perminfo->updatedCols,
                       range_attno - FirstLowInvalidHeapAttrib
```

REWRITING

- `VIEW`s
- `RULE`s
- `Query` -> List (of `Query`)

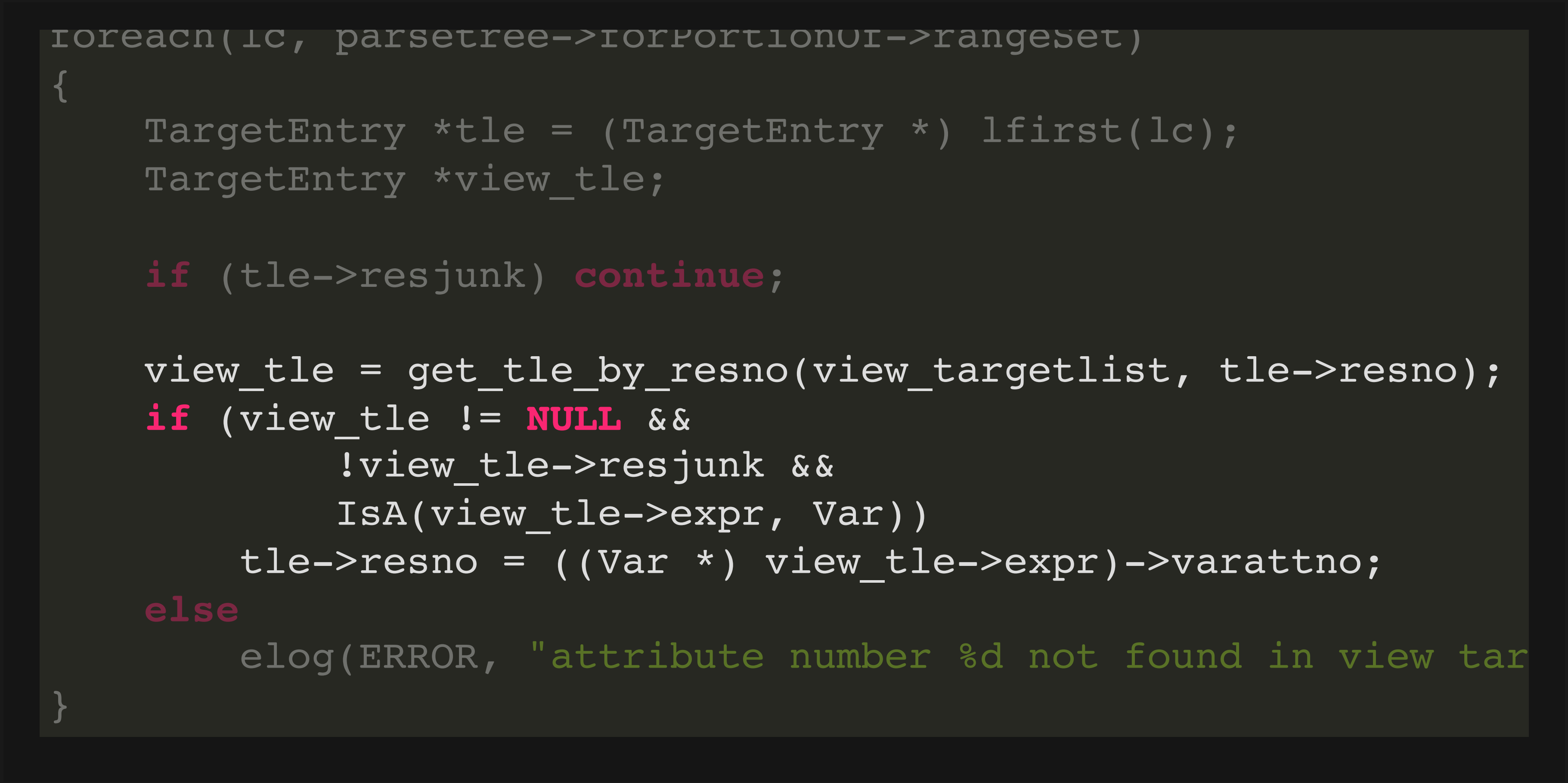
REWRITING

```
foreach(lc, parsetree->forPortionOf->rangeSet)
{
    TargetEntry *tle = (TargetEntry *) lfirst(lc);
    TargetEntry *view_tle;

    if (tle->resjunk) continue;

    view_tle = get_tle_by_resno(view_targetlist, tle->resno);
    if (view_tle != NULL &&
            !view_tle->resjunk &&
            IsA(view_tle->expr, Var))
        tle->resno = ((Var *) view_tle->expr)->varattno;
    else
        elog(ERROR, "attribute number %d not found in view tar
}
```

REWRITING

```
foreach(lc, parsetree->forPortionOf->rangeSet)
{
    TargetEntry *tle = (TargetEntry *) lfirst(lc);
    TargetEntry *view_tle;

    if (tle->resjunk) continue;

    view_tle = get_tle_by_resno(view_targetlist, tle->resno);
    if (view_tle != NULL &&
            !view_tle->resjunk &&
            IsA(view_tle->expr, Var))
        tle->resno = ((Var *) view_tle->expr)->varattno;
    else
        elog(ERROR, "attribute number %d not found in view tar
}
```

REWRITING



```
foreach(lc, parsetree->forPortionOf->rangeSet)
{
    TargetEntry *tle = (TargetEntry *) lfirst(lc);
    TargetEntry *view_tle;

    if (tle->resjunk) continue;

    view_tle = get_tle_by_resno(view_targetlist, tle->resno);
    if (view_tle != NULL &&
            !view_tle->resjunk &&
            IsA(view_tle->expr, Var))
        tle->resno = ((Var *) view_tle->expr)->varattno;
    else
        elog(ERROR, "attribute number %d not found in view tar
}
```

PLANNING & OPTIMIZING

- `Query` -> `PlannedStmt`

EXECUTOR

```
PortalStart
  ExecutorStart
    CreateExecutorState
    InitPlan
      ExecInitNode
        ...
        ExecInitModifyTable
        ...
PortalRun
  PortalRunSelect
    ExecutorRun
      ExecProcNode
        ExecModifyTable
```

EXECUTOR

```
PortalStart
  ExecutorStart
    CreateExecutorState
    InitPlan
      ExecInitNode
        ...
        ExecInitModifyTable
        ...
PortalRun
  PortalRunSelect
    ExecutorRun
      ExecProcNode
        ExecModifyTable
```

EXECUTOR

```
PortalStart
  ExecutorStart
    CreateExecutorState
    InitPlan
      ExecInitNode
        ...
        ExecInitModifyTable
        ...
PortalRun
  PortalRunSelect
    ExecutorRun
      ExecProcNode
        ExecModifyTable
```

EXECUTOR

```
PortalStart
  ExecutorStart
    CreateExecutorState
    InitPlan
      ExecInitNode
        ...
        ExecInitModifyTable
        ...
PortalRun
  PortalRunSelect
    ExecutorRun
      ExecProcNode
        ExecModifyTable
```

EXECUTOR

```
PortalStart
  ExecutorStart
    CreateExecutorState
    InitPlan
      ExecInitNode
        ...
        ExecInitModifyTable
        ...
PortalRun
  PortalRunSelect
    ExecutorRun
      ExecProcNode
        ExecModifyTable
```

EXECUTOR

```
typedef struct PlanState
{
    pg_node_attr(abstract)
    NodeTag     type;
    Plan       *plan;
    EState     *state;
    ExecProcNodeMtd ExecProcNode;
    ...
```

EXECUTOR

```
typedef struct PlanState
{
    pg_node_attr(abstract)
    NodeTag     type;
    Plan       *plan;
    EState     *state;
    ExecProcNodeMtd ExecProcNode;
    ...
```

EXECUTOR

```
typedef struct PlanState
{
    pg_node_attr(abstract)
    NodeTag     type;
    Plan       *plan;
    EState     *state;
    ExecProcNodeMtd ExecProcNode;
    ...
```

EXECUTOR

```
typedef struct PlanState
{
    pg_node_attr(abstract)
    NodeTag     type;
    Plan       *plan;
    EState     *state;
    ExecProcNodeMtd ExecProcNode;
    ...
```

EXECUTOR

```
typedef struct PlanState
{
    pg_node_attr(abstract)
    NodeTag     type;
    Plan       *plan;
    EState     *state;
    ExecProcNodeMtd ExecProcNode;
    ...
```

EXECUTOR

```
ExprState *exprState = ExecPrepareExpr(
        (Expr *) forPortionOf->targetRange, estate);
Datum targetRange = ExecEvalExpr(exprState, econtext, &isNull)
resultRelInfo->ri_forPortionOf->fp_targetRange = targetRange;
```

EXECUTOR

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

EXECUTOR

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

EXECUTOR

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

EXECUTOR

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

EXECUTOR

```
ExecutorRun
  ExecutePlan
    for (;;) {
      TupleTableSlot *slot = node->ExecProcNode(node);
      if (TupIsNull(slot))
          break;
    }
}
```

EXECUTOR

```
for (;;) {
  context.planSlot = ExecProcNode(subplanstate);
  if (TupIsNull(context.planSlot)) break;

  switch (operation) {
    case CMD_UPDATE:
      slot = ExecUpdate(...);
  }
}
```

EXECUTOR

```
for (;;) {
  context.planSlot = ExecProcNode(subplanstate);
  if (TupIsNull(context.planSlot)) break;

  switch (operation) {
    case CMD_UPDATE:
      slot = ExecUpdate(...);
  }
}
```

EXECUTOR

```
for (;;) {
  context.planSlot = ExecProcNode(subplanstate);
  if (TupIsNull(context.planSlot)) break;

  switch (operation) {
    case CMD_UPDATE:
      slot = ExecUpdate(...);
  }
}
```

EXECUTOR

```
RangeType *oldRange = slot_getattr(oldtupleSlot,
        forPortionOf->rangeVar->varattno, &isNull);
RangeType *targetRange = DatumGetRangeTypeP(
        resultRelInfo->ri_forPortionOf->fp_targetRange);

range_leftover_internal(typcache,
        oldRangeType, targetRangeType,
        &leftoverRangeType1, &leftoverRangeType2);
```

EXECUTOR

```
RangeType *oldRange = slot_getattr(oldtupleSlot,
        forPortionOf->rangeVar->varattno, &isNull);
RangeType *targetRange = DatumGetRangeTypeP(
        resultRelInfo->ri_forPortionOf->fp_targetRange);

range_leftover_internal(typcache,
        oldRangeType, targetRangeType,
        &leftoverRangeType1, &leftoverRangeType2);
```

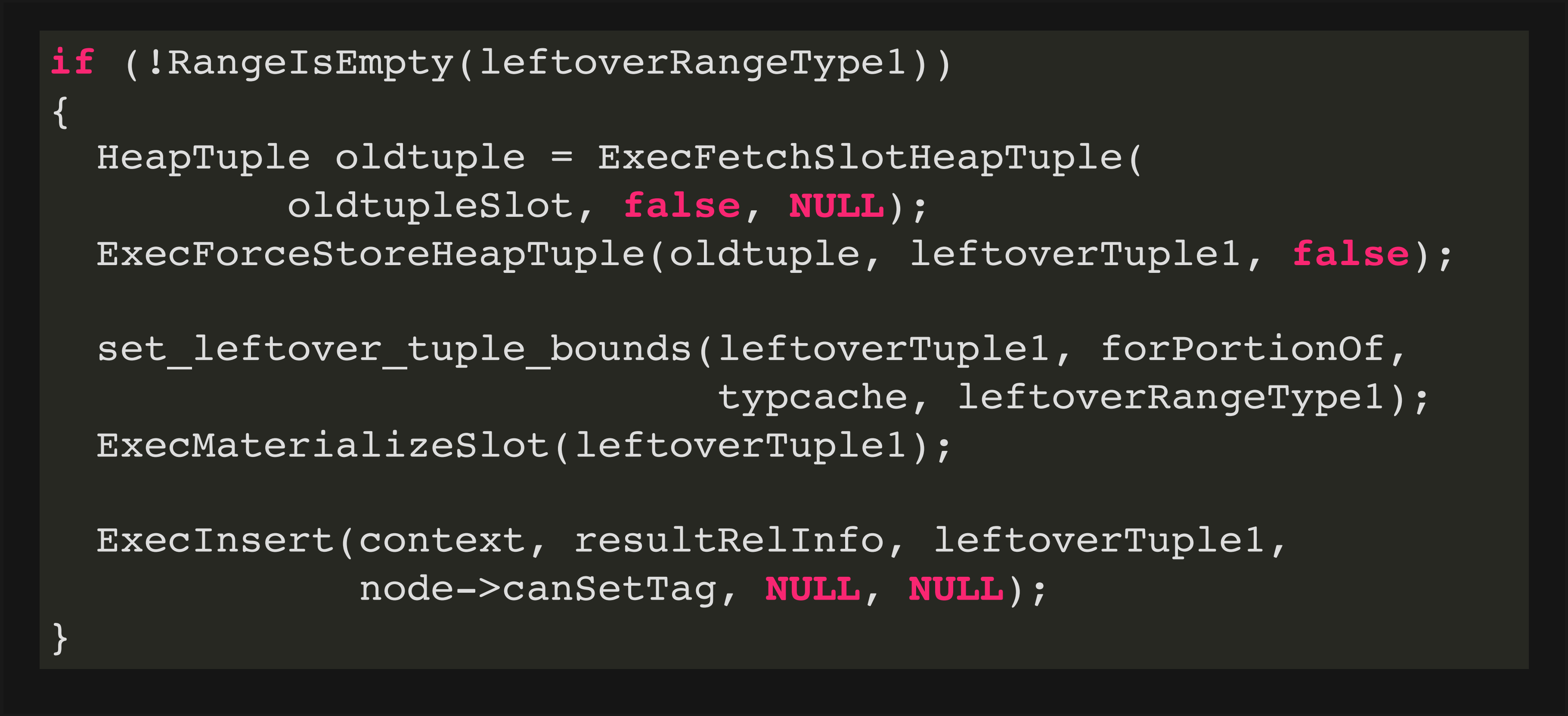
EXECUTOR

```
RangeType *oldRange = slot_getattr(oldtupleSlot,
        forPortionOf->rangeVar->varattno, &isNull);
RangeType *targetRange = DatumGetRangeTypeP(
        resultRelInfo->ri_forPortionOf->fp_targetRange);

range_leftover_internal(typcache,
        oldRangeType, targetRangeType,
        &leftoverRangeType1, &leftoverRangeType2);
```

EXECUTOR

```
RangeType *oldRange = slot_getattr(oldtupleSlot,
        forPortionOf->rangeVar->varattno, &isNull);
RangeType *targetRange = DatumGetRangeTypeP(
        resultRelInfo->ri_forPortionOf->fp_targetRange);

range_leftover_internal(typcache,
        oldRangeType, targetRangeType,
        &leftoverRangeType1, &leftoverRangeType2);
```

EXECUTOR



```
if (!RangeIsEmpty(leftoverRangeType1))
{
  HeapTuple oldtuple = ExecFetchSlotHeapTuple(
          oldtupleSlot, false, NULL);
  ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

  set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                            typcache, leftoverRangeType1);
  ExecMaterializeSlot(leftoverTuple1);

  ExecInsert(context, resultRelInfo, leftoverTuple1,
             node->canSetTag, NULL, NULL);
}
```

EXECUTOR

```
if (!RangeIsEmpty(leftoverRangeType1))
{
  HeapTuple oldtuple = ExecFetchSlotHeapTuple(
          oldtupleSlot, false, NULL);
  ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

  set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                            typcache, leftoverRangeType1);
  ExecMaterializeSlot(leftoverTuple1);

  ExecInsert(context, resultRelInfo, leftoverTuple1,
             node->canSetTag, NULL, NULL);
}
```

EXECUTOR

```
if (!RangeIsEmpty(leftoverRangeType1))
{
  HeapTuple oldtuple = ExecFetchSlotHeapTuple(
          oldtupleSlot, false, NULL);
  ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

  set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                            typcache, leftoverRangeType1);
  ExecMaterializeSlot(leftoverTuple1);

  ExecInsert(context, resultRelInfo, leftoverTuple1,
             node->canSetTag, NULL, NULL);
}
```

EXECUTOR

```
if (!RangeIsEmpty(leftoverRangeType1))
{
  HeapTuple oldtuple = ExecFetchSlotHeapTuple(
          oldtupleSlot, false, NULL);
  ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

  set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                            typcache, leftoverRangeType1);
  ExecMaterializeSlot(leftoverTuple1);

  ExecInsert(context, resultRelInfo, leftoverTuple1,
             node->canSetTag, NULL, NULL);
}
```

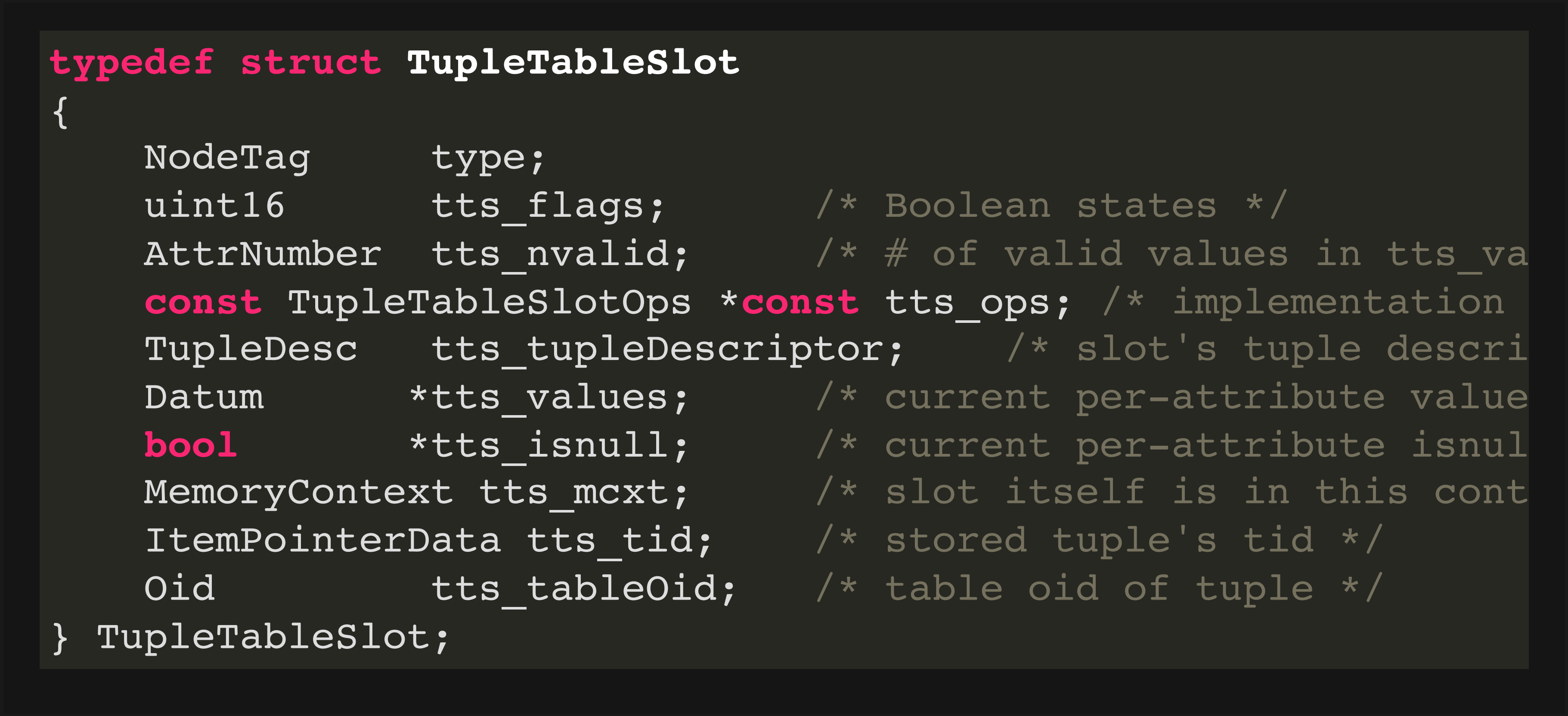
EXECUTOR

```
if (!RangeIsEmpty(leftoverRangeType1))
{
  HeapTuple oldtuple = ExecFetchSlotHeapTuple(
          oldtupleSlot, false, NULL);
  ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

  set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                            typcache, leftoverRangeType1);
  ExecMaterializeSlot(leftoverTuple1);

  ExecInsert(context, resultRelInfo, leftoverTuple1,
             node->canSetTag, NULL, NULL);
}
```

EXECUTOR

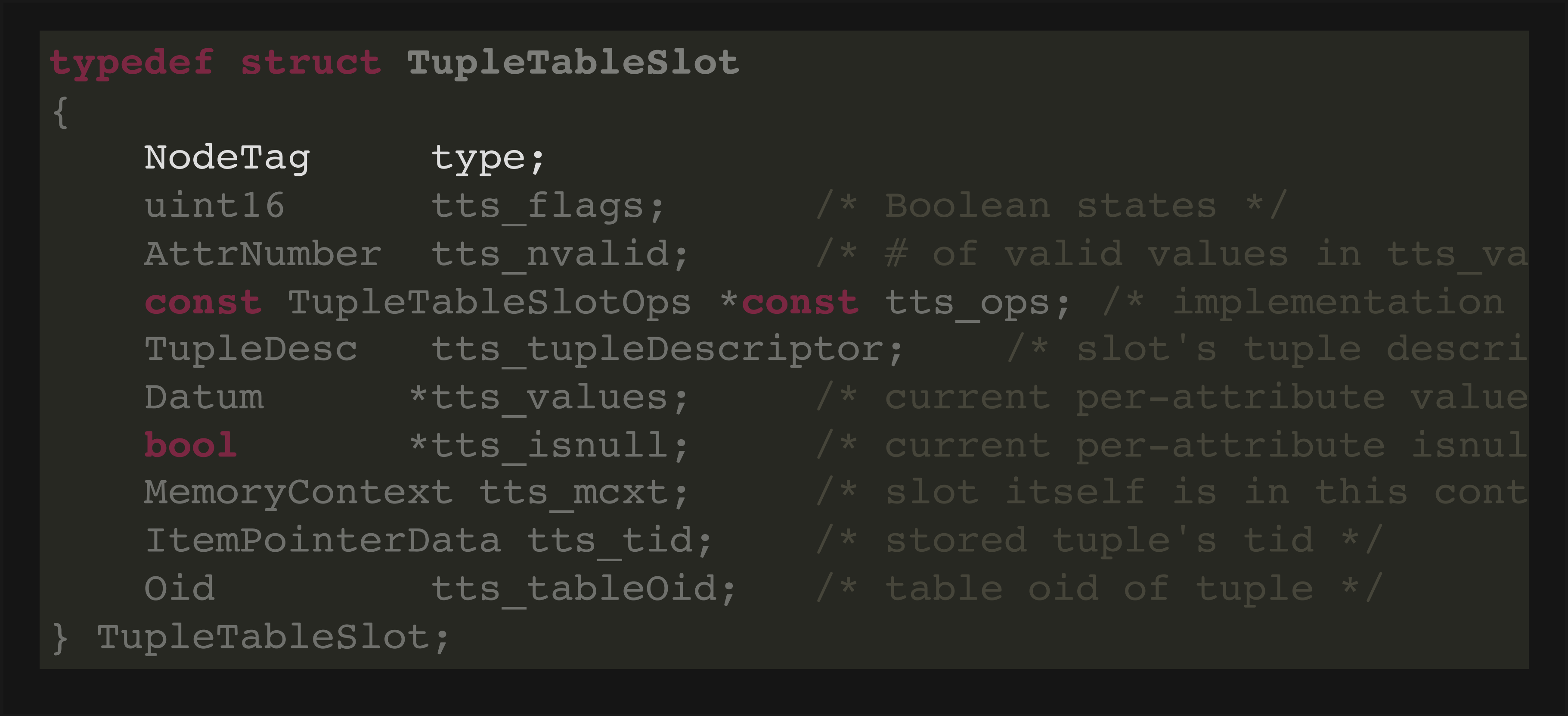
```
if (!RangeIsEmpty(leftoverRangeType1))
{
  HeapTuple oldtuple = ExecFetchSlotHeapTuple(
          oldtupleSlot, false, NULL);
  ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

  set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                            typcache, leftoverRangeType1);
  ExecMaterializeSlot(leftoverTuple1);

  ExecInsert(context, resultRelInfo, leftoverTuple1,
             node->canSetTag, NULL, NULL);
}
```

TUPLE TABLE SLOTS



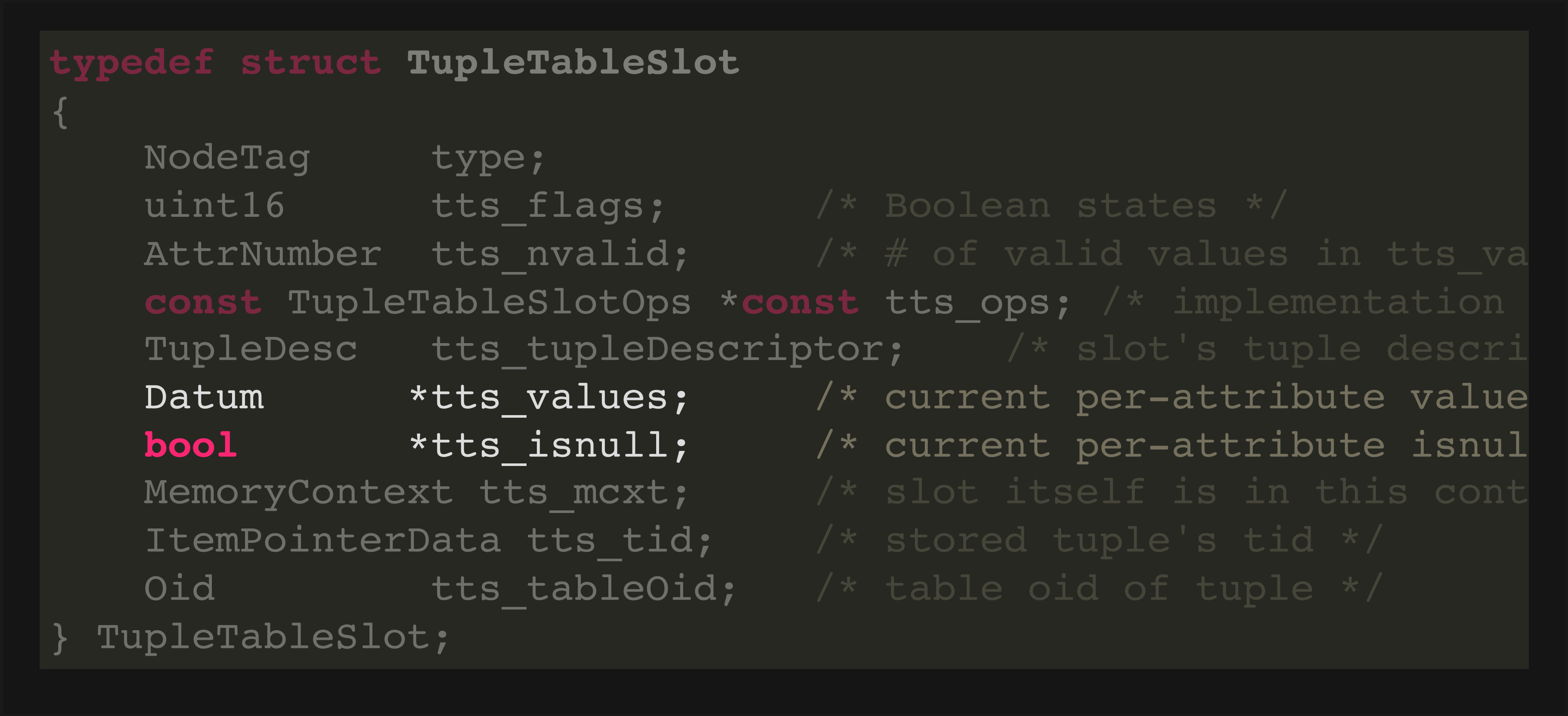
```
typedef struct TupleTableSlot
{
    NodeTag      type;
    uint16       tts_flags;      /* Boolean states */
    AttrNumber   tts_nvalid;     /* # of valid values in tts_va
    const TupleTableSlotOps *const tts_ops; /* implementation
    TupleDesc    tts_tupleDescriptor;    /* slot's tuple descri
    Datum       *tts_values;     /* current per-attribute value
    bool        *tts_isnull;     /* current per-attribute isnul
    MemoryContext tts_mcxt;      /* slot itself is in this cont
    ItemPointerData tts_tid;     /* stored tuple's tid */
    Oid          tts_tableOid;   /* table oid of tuple */
} TupleTableSlot;
```

TUPLE TABLE SLOTS



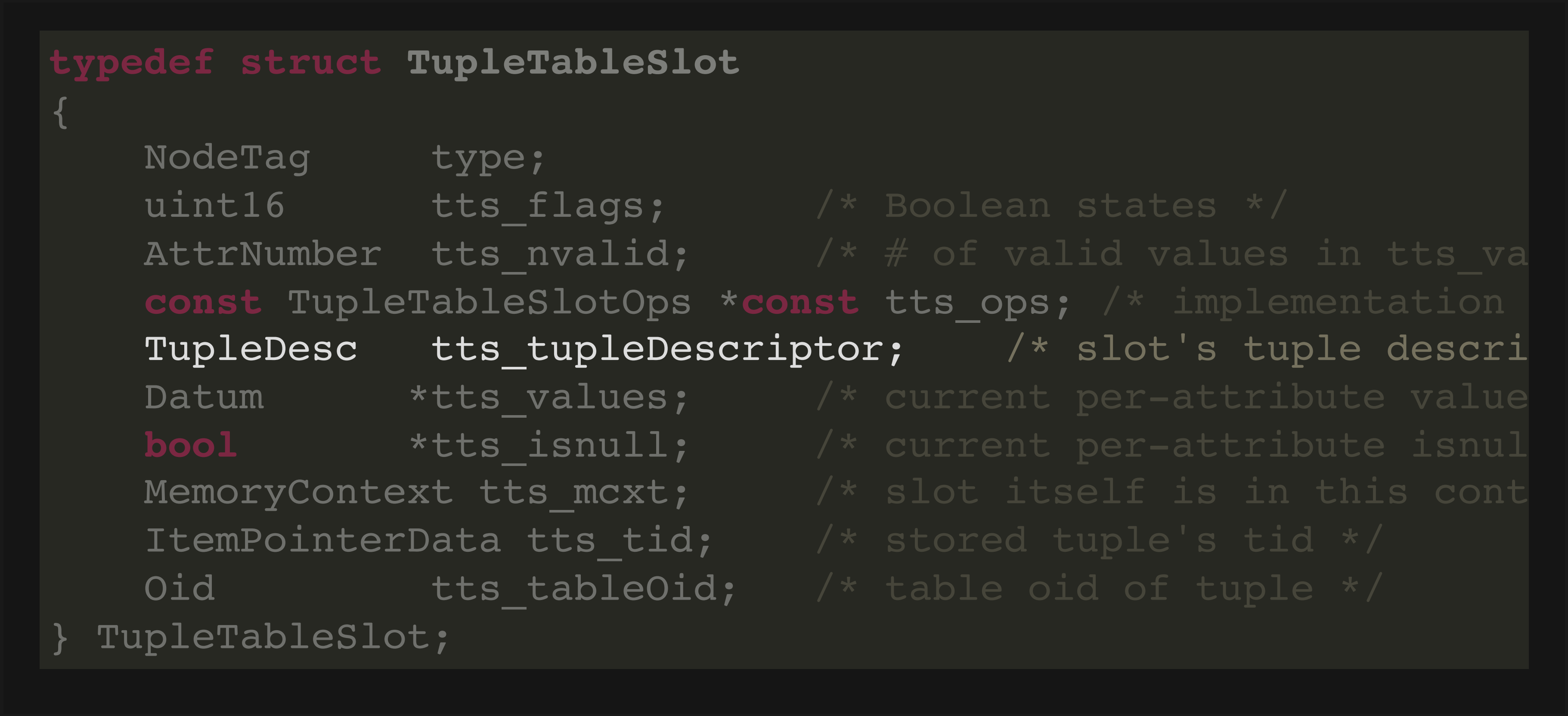
```
typedef struct TupleTableSlot
{
    NodeTag         type;
    uint16          tts_flags;      /* Boolean states */
    AttrNumber      tts_nvalid;     /* # of valid values in tts_va
    const TupleTableSlotOps *const tts_ops; /* implementation
    TupleDesc       tts_tupleDescriptor;    /* slot's tuple descri
    Datum          *tts_values;     /* current per-attribute value
    bool           *tts_isnull;     /* current per-attribute isnul
    MemoryContext tts_mcxt;         /* slot itself is in this cont
    ItemPointerData tts_tid;        /* stored tuple's tid */
    Oid             tts_tableOid;   /* table oid of tuple */
} TupleTableSlot;
```

TUPLE TABLE SLOTS



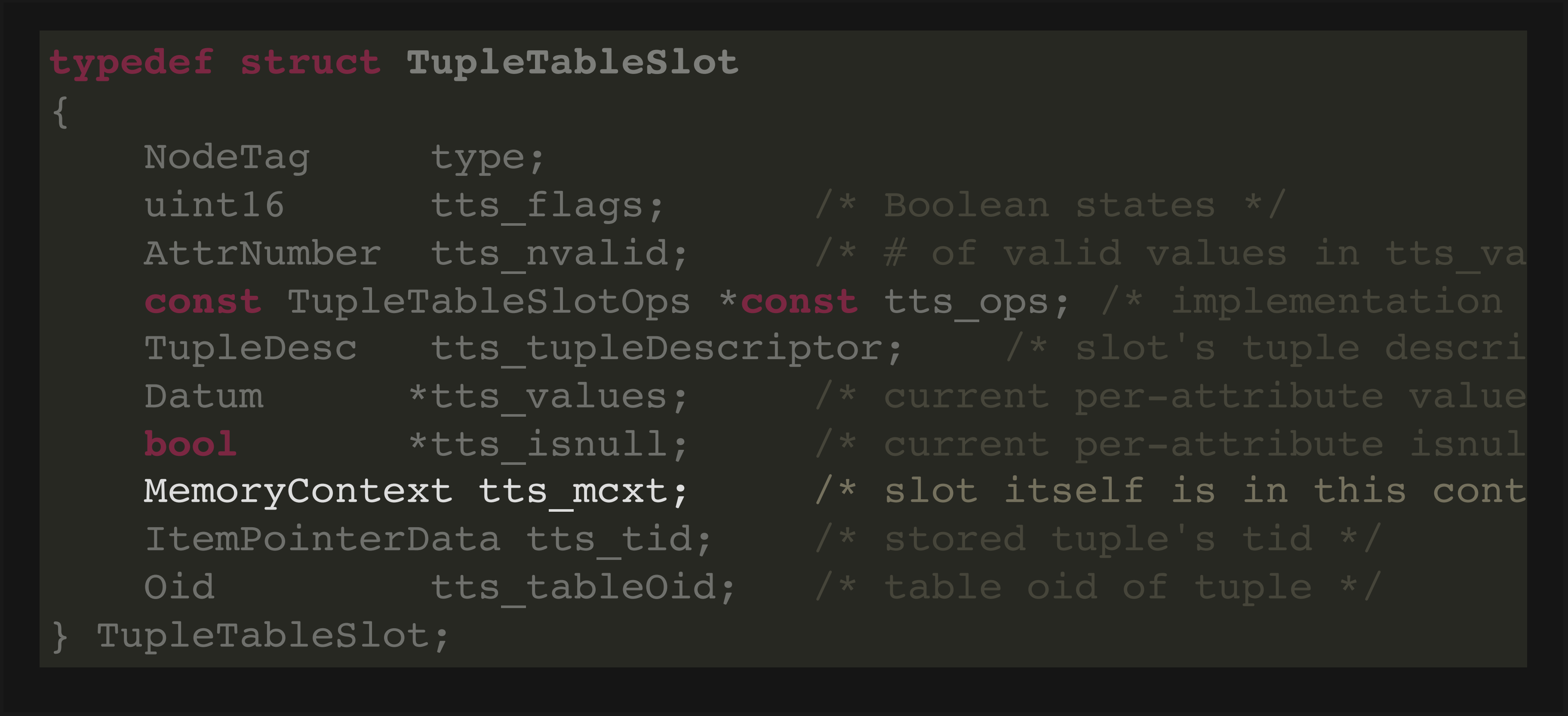
```
typedef struct TupleTableSlot
{
    NodeTag     type;
    uint16      tts_flags;      /* Boolean states */
    AttrNumber  tts_nvalid;     /* # of valid values in tts_va
    const TupleTableSlotOps *const tts_ops; /* implementation
    TupleDesc   tts_tupleDescriptor;    /* slot's tuple descri
    Datum      *tts_values;     /* current per-attribute value
    bool       *tts_isnull;     /* current per-attribute isnul
    MemoryContext tts_mcxt;     /* slot itself is in this cont
    ItemPointerData tts_tid;    /* stored tuple's tid */
    Oid         tts_tableOid;   /* table oid of tuple */
} TupleTableSlot;
```

TUPLE TABLE SLOTS



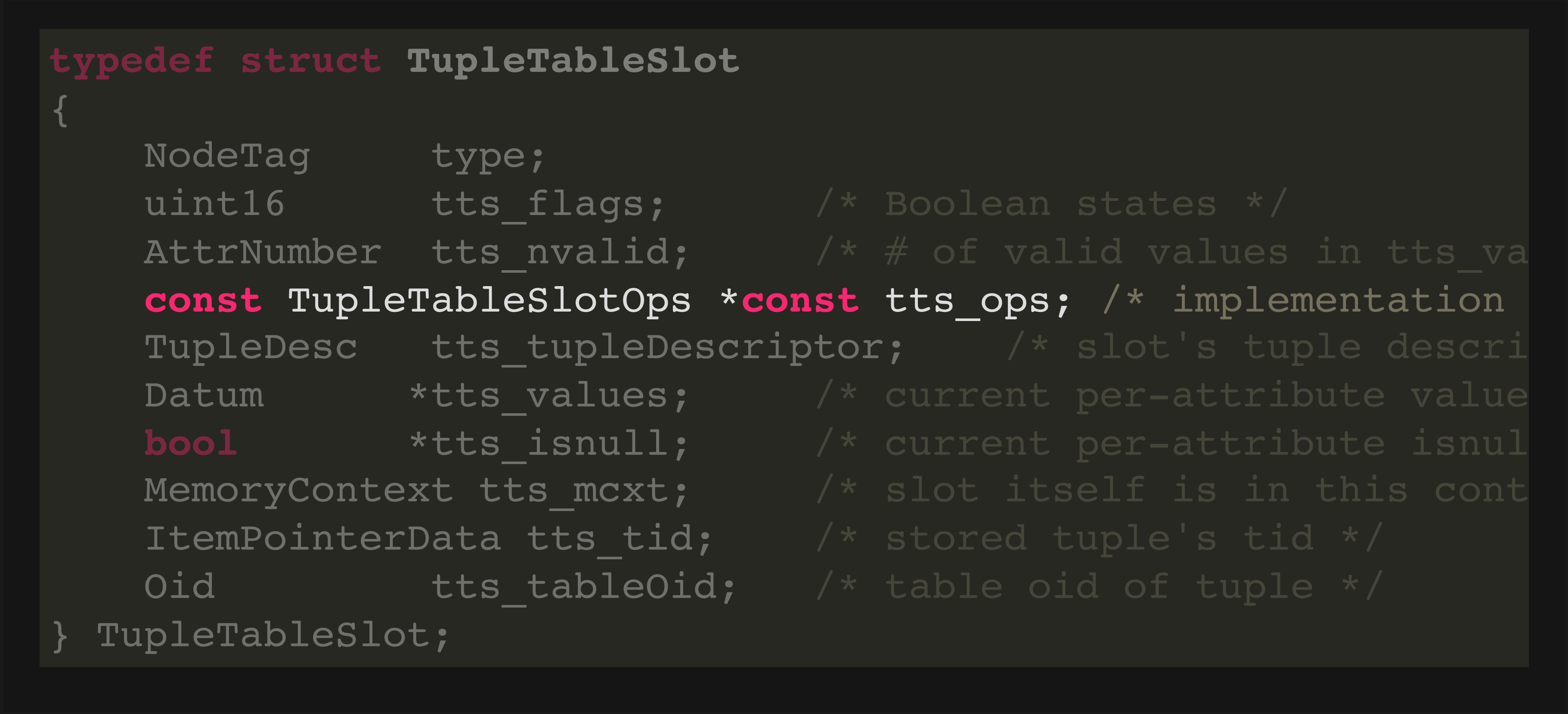
```
typedef struct TupleTableSlot
{
    NodeTag         type;
    uint16          tts_flags;      /* Boolean states */
    AttrNumber      tts_nvalid;     /* # of valid values in tts_va
    const TupleTableSlotOps  *const tts_ops; /* implementation
    TupleDesc       tts_tupleDescriptor;    /* slot's tuple descri
    Datum          *tts_values;     /* current per-attribute value
    bool           *tts_isnull;     /* current per-attribute isnul
    MemoryContext tts_mcxt;         /* slot itself is in this cont
    ItemPointerData tts_tid;        /* stored tuple's tid */
    Oid             tts_tableOid;   /* table oid of tuple */
} TupleTableSlot;
```

TUPLE TABLE SLOTS



```
typedef struct TupleTableSlot
{
    NodeTag     type;
    uint16      tts_flags;      /* Boolean states */
    AttrNumber  tts_nvalid;     /* # of valid values in tts_va
    const TupleTableSlotOps *const tts_ops; /* implementation
    TupleDesc   tts_tupleDescriptor;    /* slot's tuple descri
    Datum      *tts_values;     /* current per-attribute value
    bool       *tts_isnull;     /* current per-attribute isnul
    MemoryContext tts_mcxt;     /* slot itself is in this cont
    ItemPointerData tts_tid;    /* stored tuple's tid */
    Oid         tts_tableOid;   /* table oid of tuple */
} TupleTableSlot;
```

TUPLE TABLE SLOTS



```
typedef struct TupleTableSlot
{
    NodeTag     type;
    uint16      tts_flags;      /* Boolean states */
    AttrNumber  tts_nvalid;     /* # of valid values in tts_va
    const TupleTableSlotOps *const tts_ops; /* implementation
    TupleDesc   tts_tupleDescriptor;    /* slot's tuple descri
    Datum      *tts_values;     /* current per-attribute value
    bool       *tts_isnull;     /* current per-attribute isnul
    MemoryContext tts_mcxt;     /* slot itself is in this cont
    ItemPointerData tts_tid;    /* stored tuple's tid */
    Oid         tts_tableOid;   /* table oid of tuple */
} TupleTableSlot;
```

TUPLE TABLE SLOTS

TTSOpsHeapTuple

```
typedef struct HeapTupleTableSlot
{
    pg_node_attr(abstract)
    TupleTableSlot base;
    HeapTuple   tuple;              /* physical tuple */
    uint32      off;                /* saved state for slot_deform
    HeapTupleData tupdata;          /* optional workspace for stor
} HeapTupleTableSlot;
```

TUPLE TABLE SLOTS

TTSOpsHeapTuple

```
typedef struct HeapTupleTableSlot
{
    pg_node_attr(abstract)
    TupleTableSlot base;
    HeapTuple   tuple;            /* physical tuple */
    uint32      off;              /* saved state for slot_deform
    HeapTupleData tupdata;        /* optional workspace for stor
} HeapTupleTableSlot;
```

TUPLE TABLE SLOTS

TTSOpsHeapTuple

```
typedef struct HeapTupleTableSlot
{
    pg_node_attr(abstract)
    TupleTableSlot base;
    HeapTuple   tuple;          /* physical tuple */
    uint32      off;            /* saved state for slot_deform
    HeapTupleData tupdata;      /* optional workspace for stor
} HeapTupleTableSlot;
```

TUPLE TABLE SLOTS

TTSOpsHeapTuple

```
RangeType *oldRange = slot_getattr(oldtupleSlot,
        forPortionOf->rangeVar->varattno, &isNull);
RangeType *targetRange = DatumGetRangeTypeP(
        resultRelInfo->ri_forPortionOf->fp_targetRange);

range_leftover_internal(typcache,
        oldRangeType, targetRangeType,
        &leftoverRangeType1, &leftoverRangeType2);
```

TUPLE TABLE SLOTS

HeapTuple

```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid     = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno    = per->perend;
    . . .
    ReleaseSysCache(perTuple);
}
```

TUPLE TABLE SLOTS

HeapTuple

```
HeapTuple perTuple = SearchSysCache2(PERIODNAME,
                                     ObjectIdGetDatum(relid),
                                     PointerGetDatum(range_nam
if (HeapTupleIsValid(perTuple))
{
    Form_pg_period per = (Form_pg_period) GETSTRUCT(perTuple);
    Oid rngtypid    = per->perrngtype;
    int start_attno = per->perstart;
    int end_attno   = per->perend;
    . . .
    ReleaseSysCache(perTuple);
}
```

TUPLE TABLE SLOTS

TTSOpsBufferHeapTuple

```
typedef struct BufferHeapTupleTableSlot
{
    pg_node_attr(abstract)

    HeapTupleTableSlot base;

    Buffer      buffer;         /* tuple's buffer, or InvalidB
} BufferHeapTupleTableSlot;
```

TUPLE TABLE SLOTS

`TTSOpsMinimalTuple`

TUPLE TABLE SLOTS

TTSOpsVirtual

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
/* Initialize slot for the existing tuple */

resultRelInfo->ri_forPortionOf->fp_Existing =
    table_slot_create(resultRelInfo->ri_RelationDesc,
                      &mtstate->ps.state->es_tupleTable);

/* Create the tuple slots for INSERTing the leftovers */

resultRelInfo->ri_forPortionOf->fp_Leftover1 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
resultRelInfo->ri_forPortionOf->fp_Leftover2 =
    ExecInitExtraTupleSlot(mtstate->ps.state, tupDesc,
                           &TTSOpsVirtual);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
HeapTuple oldtuple = ExecFetchSlotHeapTuple(
        oldtupleSlot, false, NULL);
ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                          typcache, leftoverRangeType1);
ExecMaterializeSlot(leftoverTuple1);

ExecInsert(context, resultRelInfo, leftoverTuple1,
           node->canSetTag, NULL, NULL);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
HeapTuple oldtuple = ExecFetchSlotHeapTuple(
        oldtupleSlot, false, NULL);
ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                          typcache, leftoverRangeType1);
ExecMaterializeSlot(leftoverTuple1);

ExecInsert(context, resultRelInfo, leftoverTuple1,
           node->canSetTag, NULL, NULL);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
HeapTuple oldtuple = ExecFetchSlotHeapTuple(
        oldtupleSlot, false, NULL);
ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                          typcache, leftoverRangeType1);
ExecMaterializeSlot(leftoverTuple1);

ExecInsert(context, resultRelInfo, leftoverTuple1,
           node->canSetTag, NULL, NULL);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
HeapTuple oldtuple = ExecFetchSlotHeapTuple(
        oldtupleSlot, false, NULL);
ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                          typcache, leftoverRangeType1);
ExecMaterializeSlot(leftoverTuple1);

ExecInsert(context, resultRelInfo, leftoverTuple1,
           node->canSetTag, NULL, NULL);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
HeapTuple oldtuple = ExecFetchSlotHeapTuple(
        oldtupleSlot, false, NULL);
ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                          typcache, leftoverRangeType1);
ExecMaterializeSlot(leftoverTuple1);

ExecInsert(context, resultRelInfo, leftoverTuple1,
           node->canSetTag, NULL, NULL);
```

TUPLE TABLE SLOTS

TTSOpsVirtual

```
HeapTuple oldtuple = ExecFetchSlotHeapTuple(
        oldtupleSlot, false, NULL);
ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                          typcache, leftoverRangeType1);
ExecMaterializeSlot(leftoverTuple1);

ExecInsert(context, resultRelInfo, leftoverTuple1,
           node->canSetTag, NULL, NULL);
```

TUPLE TABLE SLOTS

`TTSOpsVirtual`

```
HeapTuple oldtuple = ExecFetchSlotHeapTuple(
        oldtupleSlot, false, NULL);
ExecForceStoreHeapTuple(oldtuple, leftoverTuple1, false);

set_leftover_tuple_bounds(leftoverTuple1, forPortionOf,
                          typcache, leftoverRangeType1);
ExecMaterializeSlot(leftoverTuple1);

ExecInsert(context, resultRelInfo, leftoverTuple1,
           node->canSetTag, NULL, NULL);
```

THANK YOU!

REFERENCES

1. Selena Deckelmann, *So, you want to a developer*, 2011. https://wiki.postgresql.org/wiki/So,_you_want_to_be_a_developer%3F

2. Laetitia Avrot, *Demystifying Contributing to PostgreSQL*, 2018. https://www.slideshare.net/LtitiaAvrot/demystifying-contributing-to-postgresql

3. Neil Conway and Gavin Sherry, *Introduction to Hacking PostgreSQL*, 2007. http://www.neilconway.org/talks/hacking/hack_slides.pdf and https://www.cse.iitb.ac.in/infolab/Data/Courses/CS631/PostgreSQL-Resources/hacking_intro.pdf

4. Greg Smith, *Exposing PostgreSQL Internals with User-Defined Functions*, 2010. https://www.pgcon.org/2010/schedule/attachments/142_HackingWithUDFs.pdf

5. Hironobu Suzuki, *The Internals of PostgreSQL*, 2012. http://www.interdb.jp/pg/

6. Egor Rogov, *Indexes in PostgreSQL*, 2019. https://habr.com/ru/companies/postgrespro/articles/441962/

7. Tom Lane, *Re: [HACKERS] [PROPOSAL] Temporal query processing with range types*, pgsql-hackers mailing list, 2017. https://www.postgresql.org/message-id/32265.1510681378@sss.pgh.pa.us

8. Robert Haas, *Re: MERGE SQL statement for PG12*, pgsql-hackers mailing list, 2019. https://www.postgresql.org/message-id/CA%2BTgmoZj8fyJGAFxs%3D8Or9LeNyKe_xtoSN_zTeCSgoLrUye%3D9Q%40mail.gmail.com

9. Paul Jungwirth, https://github.com/pjungwir/pgcon-2023-talk-exec-phase