

# POSTGRES PROGRESS ON TEMPORAL DATABASES

Paul A. Jungwirth

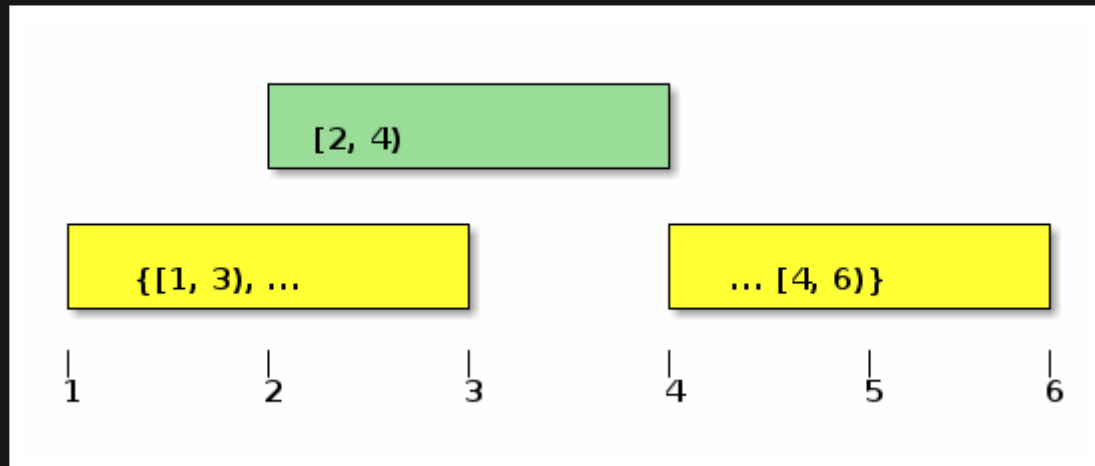
May 2020

PGCon 2020

# PATCHES

- Multiranges
- SQL:2011 Temporal
  - Primary Keys
  - Foreign Keys
  - UPDATE FOR PORTION OF
  - DELETE FOR PORTION OF

# MULTIRANGES



# MULTIRANGES

```
[1,3)  
empty
```

```
{ [1,3), [7,9) }  
{ }
```

# MULTIRANGES

```
[1,3)  
empty
```

```
{ [1,3), [7,9) }  
{ }
```

```
{empty}  
{ [1,4), [4,9), [6,9) }
```

# MULTIRANGES: CLOSURE

```
[1,4) + [7,9)      -- boom  
[1,9) - [4,7)      -- boom
```

# MULTIRANGES: CLOSURE

```
[1,4) + [7,9)      -- boom  
[1,9) - [4,7)      -- boom
```

```
{ [1,4) } + { [7,9) }  -- { [1,4) , [7,9) }  
{ [1,9) } - { [4,7) }  -- { [1,4) , [7,9) }
```

# MULTIRANGES

**ranges**

**multiranges**

int4range

int4multirange

tsrange

tsmultirange

...

...

textrange

textmultirange

...

...

anyrange

anymultirange

anycompatiblerange

anycompatiblemultirange



# MULTIRANGES: MEMORY

```
typedef struct
{
    char    vl_len_[4];          /* varlena header */
    Oid      multirangetypeid;    /* multirange type's OID */
    uint32   rangeCount;         /* the number of ranges */

    /* RangeType structs here, which are also varlena... */
} MultirangeType;
```

# MULTIRANGES: MEMORY

```
*ranges = calloc(*range_count * sizeof(RangeType *));

ptr = (char *) multirange;
end = ptr + VARSIZE(multirange);
ptr = (char *) MAXALIGN(multirange + 1);
i = 0;
while (ptr < end)
{
    r = (RangeType *) ptr;
    (*ranges)[i++] = r;

    ptr += MAXALIGN(VARSIZE(r));
}
```

# MULTIRANGES: OPERATORS

= <> < <= > >=

<< >> -|-

@> <@

&> &<

&&

+ - \*

# MULTIRANGES: IDENTITIES

$$\begin{aligned}x + \{\} &= x \\x - \{\} &= x \\x * \{\} &= \{\}\end{aligned}$$

$$\begin{aligned}x @> y &\equiv x + y = x \\x @> \{\} & \\ \{\} @> x &\equiv x = \{\}\end{aligned}$$

$$\begin{aligned}x \&\& y &\equiv x * y \neq \{\} \text{ where } x \neq \{\} \text{ and } y \neq \{\} \\x \&\& \{\} &\end{aligned}$$

# MULTIRANGES: POLYMORPHIC

| poly          | concrete       |
|---------------|----------------|
| anyelement    | integer        |
| anyarray      | integer[]      |
| anyrange      | int4range      |
| anymultirange | int4multirange |

# MULTIRANGES: FUNCTIONS

`lower(anymultirange)` returns `anyelement`  
`upper(anymultirange)` returns `anyelement`

`multirange(r anyrange)` returns `anymultirange`  
`range_merge(r anymultirange)` returns `anyrange`

`range_agg(r anyrange)` returns `anymultirange`  
`range_intersect_agg(r anyrange)` returns `anymultirange`

# MULTIRANGES: SELECTIVITY

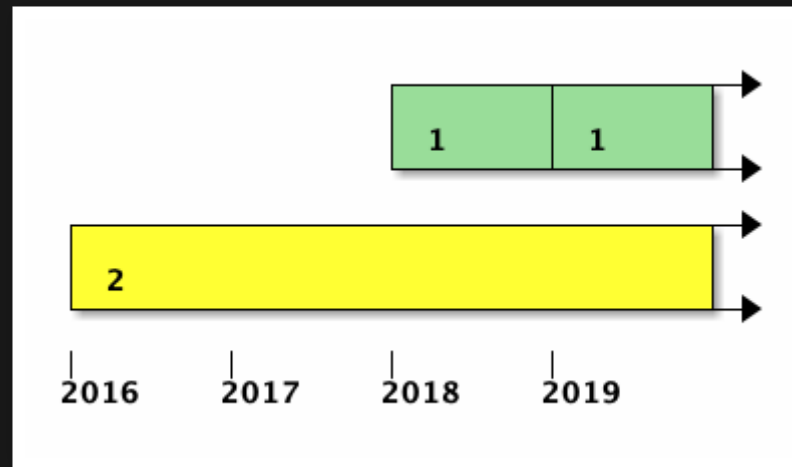
```
{ oid => '8073',  
  oid_symbol => 'OID_RANGE_OVERLAPS_MULTIRANGE_OP',  
  descr => 'overlaps', oprname => '&&',  
  oprleft => 'anyrange', oprright => 'anymultirange',  
  oprresult => 'bool',  
  oprcom => '&&(anymultirange,anyrange)',  
  oprcode => 'range_overlaps_multirange',  
  oprrest => 'multirangesel',  
  oprjoin => 'areajoinse1' },
```

# PRIMARY KEYS

```
CREATE CONSTRAINT pk_houses  
PRIMARY KEY  
(id, valid_at WITHOUT OVERLAPS);
```



# PRIMARY KEYS



# PRIMARY KEYS

```
IndexElem *iparam = makeNode(IndexElem);  
iparam->name = pstrdup(without_overlaps_str);  
index->indexParams = lappend(index->indexParams, iparam);  
  
opname = list_make1(makeString("&&"));  
index->excludeOpNames = lappend(index->excludeOpNames, opname);  
index->accessMethod = "gist";  
constraint->access_method = "gist";
```

# FOREIGN KEYS

```
CREATE CONSTRAINT fk_rooms_to_houses  
  FOREIGN KEY  
    (house_id, PERIOD valid_at)  
  REFERENCES  
    (id, PERIOD valid_at);
```

# FOREIGN KEYS

# FOREIGN KEYS

```
CREATE CONSTRAINT TRIGGER RI_ConstraintTrigger_c_454735  
  AFTER INSERT ON rooms FROM houses  
  NOT DEFERRABLE INITIALLY IMMEDIATE  
  FOR EACH ROW  
  EXECUTE PROCEDURE RI_FKey_check_ins();
```

# FOREIGN KEYS

```
SELECT 1  
FROM    houses x  
WHERE   id = $1  
FOR KEY SHARE OF x;
```

# FOREIGN KEYS

```
CREATE CONSTRAINT TRIGGER TRI_ConstraintTrigger_c_454735  
  AFTER INSERT ON rooms FROM houses  
  NOT DEFERRABLE INITIALLY IMMEDIATE  
  FOR EACH ROW  
  EXECUTE PROCEDURE TRI_FKey_check_ins();
```

# FOREIGN KEYS

```
SELECT 1
FROM (
    SELECT valid_at AS r
    FROM   houses x
    WHERE  id = $1
    AND    valid_at && $2
    FOR KEY SHARE OF x
) x1
HAVING $2 <@ range_agg(x1.r);
```



# FOREIGN KEYS

```
CREATE CONSTRAINT fk_rooms_to_houses
  FOREIGN KEY
    (house_id, PERIOD valid_at)
  REFERENCES
    (id, PERIOD valid_at)
```

# FOREIGN KEYS

```
CREATE CONSTRAINT fk_rooms_to_houses
  FOREIGN KEY
    (house_id, PERIOD valid_at)
  REFERENCES
    (id, PERIOD valid_at)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
```

# UPDATE

```
UPDATE houses
  FOR PORTION OF valid_at
FROM   '2020-01-01' TO '2021-01-01'
SET    tax_appraisal = 250000
WHERE  id = 5;
```

# DELETE

```
DELETE FROM houses
  FOR PORTION OF valid_at
  FROM   '1950-01-01' TO '1960-01-01'
 WHERE id = 7;
```

# UPDATE/DELETE

- Parse
- Analyze
- Rewrite
- Plan
- Optimize
- Execute

# UPDATE/DELETE: PARSE

parsenodes.h

```
typedef struct ForPortionOfClause
{
    NodeTag      type;
    char         *range_name;
    int          range_name_location;
    Node         *target_start;
    Node         *target_end;
} ForPortionOfClause;
```

# UPDATE/DELETE: PARSE

```
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; }
;
```

# UPDATE/DELETE: PARSE

```
UPDATE for_portion_of_test
FOR PORTION OF valid_at
  FROM '2018-03-01' AT TIME ZONE INTERVAL '1' DAY TO HOUR
  TO '2019-01-01'
  SET name = 'one^3'
  WHERE id = '[1,2)';
```



# UPDATE/DELETE: PARSE

```
state 1855
```

```
1923 opt_interval: DAY_P .  
1928           | DAY_P . TO HOUR_P  
1929           | DAY_P . TO MINUTE_P  
1930           | DAY_P . TO interval_second
```

```
TO shift, and go to state 2611
```

```
TO [reduce using rule 1923 (opt_interval)]  
$default reduce using rule 1923 (opt_interval)
```

# UPDATE/DELETE: PARSE

```
if (foo)
  if (bar)
    stmt1
  else
    stmt2
```

# UPDATE/DELETE: ANALYZE

```
FOR PORTION OF valid_at  
FROM '2020-05-30'  
TO '2021-01-31'
```

# UPDATE/DELETE: ANALYZE

```
FuncCall *fc = makeFuncCall(  
    SystemFuncName(range_type_name),  
    list_make2(result->targetStart,  
               result->targetEnd),  
    forPortionOf->range_name_location);  
result->targetRange = transformExpr(  
    pstate,  
    (Node *) fc,  
    EXPR_KIND_UPDATE_PORTION);  
result->overlapsExpr = (Node *) makeSimpleA_Expr(  
    AEXPR_OP, "&&",  
    (Node *) result->range,  
    (Node *) fc,  
    forPortionOf->range_name_location);
```

# UPDATE/DELETE: ANALYZE

```
if (stmt->forPortionOf)
{
    if (stmt->whereClause)
        whereClause = (Node *) makeBoolExpr(
            AND_EXPR,
            list_make2(qry->forPortionOf->overlapsExpr,
                      stmt->whereClause),
            -1);
    else
        whereClause = qry->forPortionOf->overlapsExpr;
}
else
    whereClause = stmt->whereClause;
```

# UPDATE: ANALYZE

```
Expr *rangeSetExpr = (Expr *) makeSimpleA_Expr(  
    AEXPR_OP, "*",  
    (Node *) result->range,  
    (Node *) fc,  
    forPortionOf->range_name_location);
```

```
rangeSetExpr = (Expr *) transformExpr(  
    pstate,  
    (Node *) rangeSetExpr,  
    EXPR_KIND_UPDATE_PORTION);
```

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

```
targetList = lappend(targetList, tle);
```

# UPDATE/DELETE: ANALYZE

```
FOR PORTION OF valid_at  
FROM '2020-05-30'  
TO 'Infinity'
```

```
FOR PORTION OF valid_at  
FROM '2020-05-30'  
TO NULL
```

# UPDATE/DELETE: ANALYZE

```
=# SELECT tstzrange(NOW(), NULL)
-#      - tstzrange(NOW(), 'infinity');
```



# UPDATE/DELETE: ANALYZE

```
=# SELECT tstzrange(NOW(), NULL)
-#      - tstzrange(NOW(), 'infinity');
   ?column?
-----
 [infinity,)
(1 row)
```

**UPDATE/DELETE:  
REWRITE**

**UPDATE/DELETE:  
PLAN**

**UPDATE/DELETE:  
OPTIMIZE**

# UPDATE/DELETE: EXECUTE

```
// executor/modifyTableNode.c
ExprContext *econtext = GetPerTupleExprContext(estate);
econtext->ecxt_scantuple = slot;

ExprState *exprState = ExecPrepareExpr(
    (Expr *) forPortionOf->targetRange, estate);
targetRange = ExecEvalExpr(exprState, econtext, &isNull);

if (isNull) elog(ERROR, "Got a NULL FOR PORTION OF target range");
targetRangeType = DatumGetRangeTypeP(targetRange);
resultRelInfo->ri_forPortionOf->fp_targetRange = targetRangeType;
```

# UPDATE/DELETE: EXECUTE

```
table_tuple_fetch_row_version(  
    resultRelInfo->ri_RelationDesc,  
    tupleid, SnapshotAny, oldtupleSlot);  
oldRange = slot_getattr(  
    oldtupleSlot,  
    forPortionOf->range_attno, &isNull);  
oldRangeType = DatumGetRangeTypeP(oldRange);
```

# UPDATE/DELETE: EXECUTE

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

# UPDATE/DELETE: EXECUTE

```
MinimalTuple oldtuple = ExecFetchSlotMinimalTuple(  
    oldtupleSlot, NULL);  
ExecForceStoreMinimalTuple(oldtuple, leftoverTuple1, false);  
  
leftoverTuple1->tts_values[forPortionOf->range_attno - 1] =  
    RangeTypePGetDatum(leftoverRangeType1);  
leftoverTuple1->tts_isnull[forPortionOf->range_attno - 1] =  
    false;  
  
ExecMaterializeSlot(leftoverTuple1);  
ExecInsert(mtstate, leftoverTuple1, planSlot,  
    estate, node->canSetTag);
```



# UPDATE/DELETE: EXECUTE

```
typedef struct TriggerData
{
    NodeTag          type;
    TriggerEvent      tg_event;
    Relation          tg_relation;
    HeapTuple         tg_trigtuple;
    HeapTuple         tg_newtuple;
    Trigger           *tg_trigger;
    TupleTableSlot    *tg_trigslot;
    TupleTableSlot    *tg_newslot;
    Tuplestorestate   *tg_oldtable;
    Tuplestorestate   *tg_newtable;
} TriggerData;
```

# UPDATE/DELETE: EXECUTE

```
typedef struct TriggerData
{
    NodeTag            type;
    TriggerEvent       tg_event;
    Relation           tg_relation;
    HeapTuple          tg_trigtuple;
    HeapTuple          tg_newtuple;
    Trigger            *tg_trigger;
    TupleTableSlot     *tg_trigslot;
    TupleTableSlot     *tg_newslot;
    Tuplestorestate    *tg_oldtable;
    Tuplestorestate    *tg_newtable;
    RangeTypeP         *tg_targetportion;
} TriggerData;
```

# THANKS!

<https://github.com/pjungwir/pg-temporal-talk-2020>