Range Types

Jeff Davis
pgsql@j-davis.com

Why Range Types?

- Functionality
- Performance
- Ease-of-use

What is a Range Type?

- Represents a range of values, rather than a single value.
- “1pm until 4pm” is a range
- “3.1 – 7.7” is a range
- “192.168.1.10 through .20” is a range

Functions/Operators

- Contains “@>”
- Overlaps “&&”
- Intersection “*”
- Union “+”
- Many more...

Example

```
CREATE TYPE numrange AS RANGE
(
   SUBTYPE        = NUMERIC,
   SUBTYPE_CMP = numeric_cmp
);
```

Example

```
SELECT contains(
   range(1.7, 90.1),
   3.3 -- scalar
);
-- returns TRUE

SELECT overlaps(
   '[-2, -1]'::numrange,
   range(6.2) -- singleton range
);
-- returns FALSE
```

Alternative: Quantization

- Used to approximate range types
- Use “1:00” to mean “1:00 – 2:00”
- Unnecessarily dictates business rules
  - Business rules should govern design, not vice-versa!
- Inflexible and business-dependent
- Use Range Types, not quantization

Alternative: 2-columns

- Used to approximate range types
- Bloats queries, making even simple queries complex and error-prone
- Bloats schema
- Can't efficiently use indexes
- Use Range Types, not the 2-column approach

Alternative: 2-columns continued...

- Another major drawback – how do you prevent overlapping ranges?
  - Consider a time schedule
- With Range Types, you can take advantage of Exclusion Constraints (new in 9.0) for a simple, robust solution.
- Without range types, it's a major challenge just to prevent a schedule conflict!
  - Do not underestimate this challenge

Back to Range Types: Infinity

- Can use “-INF” and “INF” for the lower and upper bounds, respectively.
- Allow open-ended ranges

Empty Ranges

- Empty ranges are specified as:

  `'-'::numrange`
- Every range contains the empty range
- Empty ranges are equal to other empty ranges
- **NOT** like a NULL
- No range overlaps with the empty range

NULLs and Ranges

- Range boundaries cannot be NULL
- Would lead to confusion in cases like:

```
range(NULL, 10.1) –
range(NULL,  5.1)
```

- Use cases involving NULL would probably better be solved using infinity.

Inclusive/Exclusive Bounds

- Does '[1.1, 2.2)' include the point 2.2?
- “[“ and “]” mean “inclusive”
- And “(“ and “)” mean “exclusive”
- Answer: No.
- Range(1.1, 2.2) constructor function uses inclusive-exclusive form
  - Other constructors exist

[ ) and ( ]

- Inclusive-exclusive and exclusive-inclusive form
- Important because ranges can be adjacent without having any overlapping points
  - Consider a time schedule
- But singleton ranges must be [ ]

Range Join

```
SELECT
  range_intersect(t1.r1, t2.r2)
  FROM  t1, t2
  WHERE t1.r1 && t2.r2;
```

Temporal

- PERIOD is a range of TIMESTAMPs
- PERIODTZ is a range of TIMESTAMPTZs
- Use PERIOD to represent arbitrary time ranges
- Prevent schedule conflicts with Exclusion Constraints
- Improve performance with range indexing
- Simplify schema and queries

What about INTEGER?

- Is [1, 5] = [1, 6) ?
- Both represent the values 1 through 5
- Answer: yes.
- But how does the system know that INTEGER is different from NUMERIC?

Continuous Ranges

- Until now, we've been using continuous range semantics.
- Two ranges aren't equal unless boundaries are equal *and* inclusivity/exclusivity of boundaries are also equal.
- Example: NUMERIC

Discrete Ranges

- Represent a range of values with a definite “step”
- Boundaries can be transformed from inclusive to exclusive and vice-versa.
- [1,5] can become [1,6), (0, 5], or (0, 6)
- Example: INTEGER

Discrete Ranges continued...

- Specify a discrete range type by providing a “canonical” function.
- Takes a range, changes it to a canonical form, and returns the new range.
- Example: change any input range to '[ )' format.
- Allows generic range functions to work with discrete ranges just like continuous ranges

Conclusion

- Don't constrain yourself to representing individual points only
  - Especially not when it comes to time!
- Simplify queries and schema
- Solve the “non-overlapping” problem
  - Especially for scheduling!
- Use range indexing for performance