Data Types in PostgreSQL

CODEUS_

Agenda

- Enum
- Array
- Range
- Composite
- Domain

Enum

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
CREATE TABLE person (name text, current_mood mood);
```

```
INSERT INTO person VALUES ('John Doe', 'happy');
```

```
SELECT * FROM person WHERE current_mood != 'sad';
```

```
UPDATE person SET current_mood = 'ok' WHERE name = 'John Doe';
```

```
ALTER TYPE mood ADD VALUE 'angry';
```

Enum

```
'Ordering is defined by creation: ENUM values have an inherent order as declared.'
SELECT 'happy' > 'ok';  -- returns true, based on ENUM order
```

```
'Can be used with comparison operators, which respect the declared ENUM order.'
SELECT * FROM person WHERE current_mood > 'ok';
```

```
'Use Cases: user status, task priority, workflow states'
```

CODEUS_

Array

```
CREATE TABLE sal_emp (name text, pay_by_quarter integer[], schedule text[][]);
```

```
INSERT INTO sal_emp VALUES ('Bill', ARRAY[10000, 11000, 12000, 13000], '{{meeting,lunch},{training,presentation}}');
```

```
SELECT name FROM sal_emp WHERE array_length(pay_by_quarter, 1) > 4;
SELECT name FROM sal_emp WHERE pay_by_quarter[1] = 10000;
```

```
UPDATE sal_emp SET pay_by_quarter[1] = 10500 WHERE name = 'Bill';
```

Array

```
'Use unnest() to flatten an array:'
SELECT unnest(pay_by_quarter) FROM sal_emp WHERE name = 'Bill';
```

```
'Use @> (contains), <@ (is contained by), and && (overlap) operators:'
-- Does the array contain all listed elements?
SELECT name FROM sal_emp WHERE pay_by_quarter @> ARRAY[10000];
-- Is the array contained within the listed elements?
SELECT name FROM sal_emp WHERE pay_by_quarter <@ ARRAY[10000, 11000];
-- Do two arrays overlap?
SELECT name FROM sal_emp WHERE pay_by_quarter && ARRAY[11000, 12000];
```

```
'Use Cases: tags, quiz answers, time series, user roles'
```

CODEUS_

Range

```sql
CREATE TABLE reservation (room int, during tsrange);
CREATE TYPE float8range AS RANGE (subtype = float8, subtype_diff = float8mi);
```

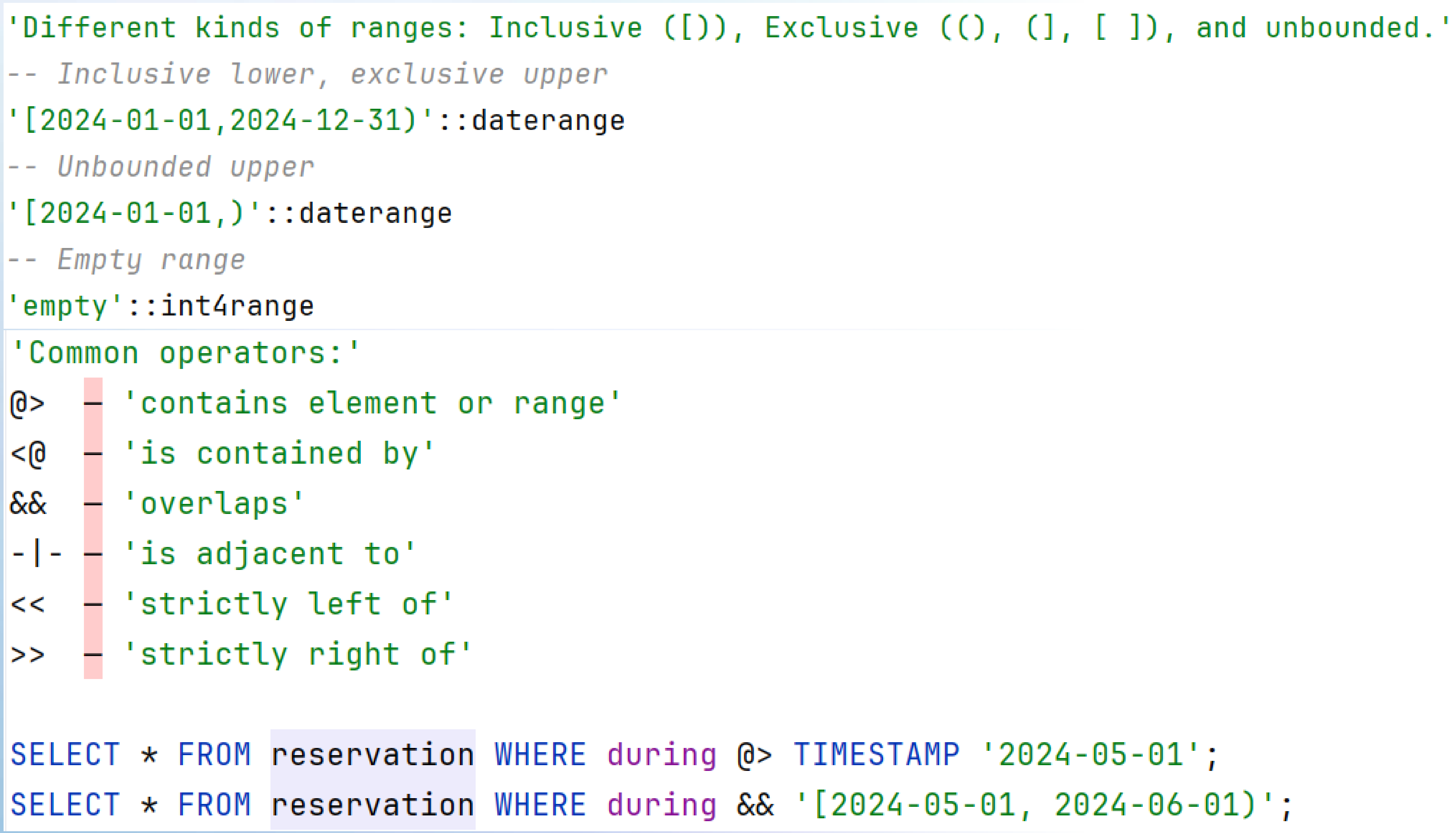
```sql
INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
```

```sql
-- Find reservations that overlap with a specific time window
SELECT * FROM reservation WHERE during && '[2010-01-01 15:00, 2010-01-01 16:00)';
```

```sql
UPDATE reservation SET during = '[2010-01-01 13:00, 2010-01-01 14:00)' WHERE room = 1108;
```

CODEUS_

Range



```
'Different kinds of ranges: Inclusive ([)), Exclusive ((), (], [ ]), and unbounded.'
-- Inclusive lower, exclusive upper
'[2024-01-01,2024-12-31)'::daterange
-- Unbounded upper
'[2024-01-01,)'::daterange
-- Empty range
'empty'::int4range
```

```
'Common operators:'
@>  – 'contains element or range'
<@  – 'is contained by'
&&  – 'overlaps'
-|- – 'is adjacent to'
<<  – 'strictly left of'
>>  – 'strictly right of'

SELECT * FROM reservation WHERE during @> TIMESTAMP '2024-05-01';
SELECT * FROM reservation WHERE during && '[2024-05-01, 2024-06-01)';
```

CODEUS_

Range

```
'Use lower(), upper(), isempty() to inspect ranges:'
SELECT lower(during), upper(during), isempty(during) FROM reservation;
```

```
'Use cases: booking periods, product availability windows, numeric ranges'
```

Composite

```
CREATE TYPE inventory_item AS (name text, supplier_id integer, price numeric);
CREATE TABLE on_hand (item inventory_item, count integer);
```

```
INSERT INTO on_hand VALUES (ROW('fuzzy dice', 42, 1.99), 100);
```

```
SELECT (item).name, count FROM on_hand WHERE (item).name LIKE '%dice%';
```

```
UPDATE on_hand SET item.price = 2.49 WHERE (item).name = 'fuzzy dice';
```

CODEUS_

Composite

```
'You can compare full composite values:'
SELECT * FROM on_hand WHERE item = ROW('fuzzy dice', 42, 1.99);
```

```
'Can be used with functions and aggregates by accessing fields:'
SELECT AVG((item).price) FROM on_hand;
```

```
'A composite type can be returned from a function.'
```

```
'Use Cases: grouped attributes like address, specs, coordinates'
```

CODEUS_

Domain

```
CREATE DOMAIN rating AS integer CHECK (VALUE BETWEEN 1 AND 5);
CREATE TABLE reviews (id serial, score rating);
```

```
INSERT INTO reviews(score) VALUES (4);
```

```
SELECT * FROM reviews WHERE score > 3;
```

```
UPDATE reviews SET score = 5 WHERE id = 1;
```

```
ALTER DOMAIN rating ADD CONSTRAINT rating_max CHECK (VALUE <= 5);
```

Domain

```
'Custom Operators: can define based on base type logic'
CREATE DOMAIN mytext AS text CHECK(...);
CREATE FUNCTION mytext_eq_text (mytext, text) RETURNS boolean AS ...;
CREATE OPERATOR = (procedure=mytext_eq_text, leftarg=mytext, rightarg=text);
CREATE TABLE mytable (val mytext);

SELECT * FROM mytable WHERE val = text 'foo';
```

```
'Use Cases: positive integers, email/zipcode validation, normalized enums'
```

CODEUS_

Thank you

- Author: Serhii Kravchuk
- My LinkedIn: Link
- Date: May 2025
- Join Codeus community in Discord
- Join Codeus community in LinkedIn