Love Your Database

Postgres Types

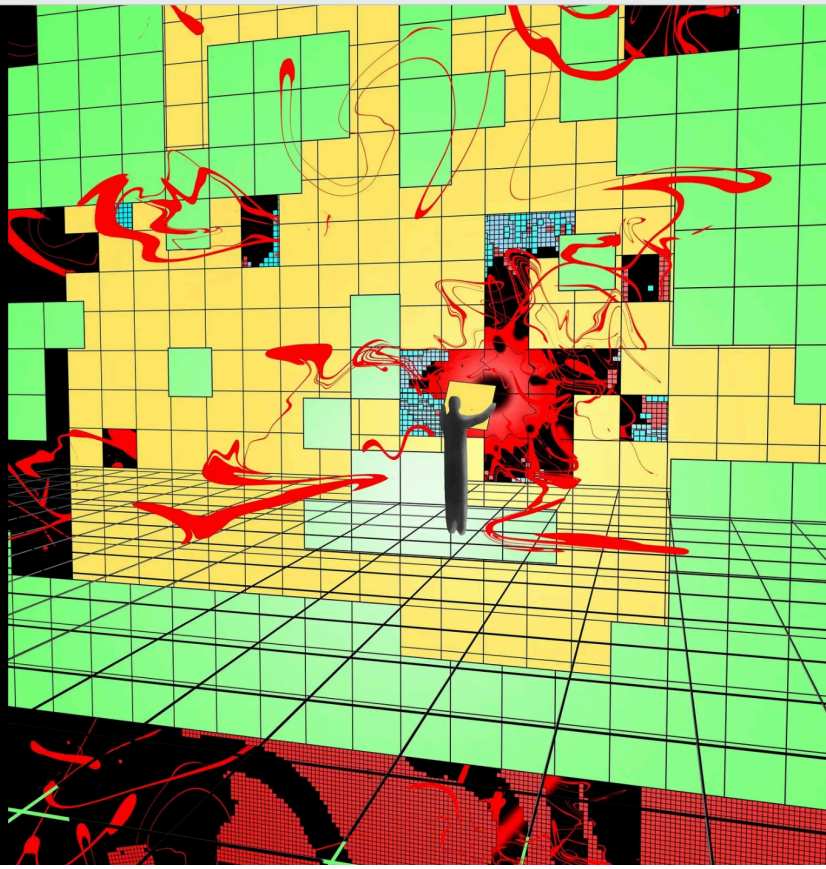
Guyren G Howe

nytimes.com/2024/04/03/technology/prevent-cyberattack-linux.html

THE SHIFT

Did One Guy Just Stop a Huge Cyberattack?

A Microsoft engineer noticed something was off on a piece of software he worked on. He soon discovered someone was probably trying to gain access to computers all over the world.



Timestamp With Time Zone (aka timestamptz)

```
SHOW timezone;
     TimeZone
----------------------
America/Los_Angeles
```

```
# CREATE TABLE tz_test(
    x TIMESTAMP WITH TIMEZONE
);
CREATE TABLE
```

```
# INSERT INTO tz_test(x)
    VALUES
        ('2024-04-17 09:00:00 -07');


# SELECT x FROM tz_test;
            x
------------------------
 2024-04-17 09:00:00-07
```

For timestamp with time zone, **the internally stored value is always in UTC…**

An input value that has an explicit time zone specified is **converted to UTC** using the appropriate offset for that time zone

If no time zone is stated in the input string, then it is **assumed to be in the time zone indicated by the system's TimeZone** parameter

When a timestamp with time zone value is output, it is always **converted from UTC to the current timezone**

```
INSERT INTO tz_test(x) VALUES (
    '2024-04-17 09:00:00 +00');
# SELECT x FROM tz_test;
            x
------------------------
 2024-04-17 09:00:00-07
 2024-04-17 02:00:00-07
```

```
# CREATE TABLE timestamp_test(
    x TIMESTAMP);
CREATE TABLE
```

```
# INSERT INTO timestamp_test(x)
VALUES('2024-04-18 12:00:00+03');
# SELECT x FROM timestamp_test ;
            x
---------------------
 2024-04-18 12:00:00
```

```
# INSERT INTO timestamp_test(x)
VALUES(
    '2024-04-18 12:00:00+03'
    ::timestamptz
);
```

```
SELECT
    x
FROM
    timestamp_test;
          x
---------------------
 2024-04-18 06:00:00
```

Time With Time Zone (aka timetz)

Money

The money type stores a currency amount with a **fixed fractional precision**… The fractional precision is determined by the database's lc_monetary setting

The money type stores a currency amount with a fixed fractional precision… The **fractional precision is determined by the database's lc_monetary setting**

```
# SET lc_monetary='en_US.utf8';
SET
# CREATE TABLE t (m MONEY);
```

```
# INSERT INTO t VALUES('$1000.00');
INSERT 0 1
# TABLE t;
     m
------------
 $1,000.00
(1 row)
```

```
# SET lc_monetary='ja_JP.utf8';
SET
# TABLE t;
     m
------------
 ¥100,000
```

CHARACTER(…)

```
# SELECT 'a'::character(10);
    bpchar
  ================
  a
```

```
SELECT LENGTH('a'::character(10));
   length
 ==========
        1
```

```
# SELECT
    'a'::CHARACTER(10) ||
    'b'::CHARACTER(10);
   ?column?
  ================
   ab
```

Values of type character are physically **padded with spaces to the specified width n**, and are stored and displayed that way. However, trailing spaces are treated as semantically insignificant and disregarded when comparing two values of type character.

Values of type character are physically padded with spaces to the specified width n, and are **stored and displayed that way**. However, trailing spaces are treated as semantically insignificant and disregarded when comparing two values of type character.

Values of type character are physically padded with spaces to the specified width n, and are stored and displayed that way. However, **trailing spaces are treated as semantically insignificant** and disregarded when comparing two values of type character.

Trailing spaces are removed when converting a character value to one of the other string types. Note that trailing spaces are semantically significant in character varying and text values, and when using pattern matching

Trailing spaces are removed when converting a character value to one of the other string types. Note **that trailing spaces are semantically significant in character varying and text values, and when using pattern matching**

Thanks!

- Laurenz Albe
- Ian
- Adrian Klaver
- David G Johnston
- Tom Lane
- Paul Jungwirth

Enums

Enums

Advantages

- Performance
- Simpler SQL

Enums

Disadvantages

- Performance
  - “An enum value occupies four bytes on disk.”
- Changes involve DDL
- Fixed order
- Fixed case
- Non-localisation friendly
- Less Portable

Range Types

Thanks to Paul Jungwirth

Inclusive Exclusive

```
SELECT '[3,7)'::int4range;
 int4range
-----------
 [3,7)
(1 row)

/* contains 3, 4, 5, 6 */
```

```
SELECT int4range(3,7, '[)');
 int4range
-----------
 [3,7)
(1 row)
```

```
SELECT '{[3,7), [8,9)}'::int4multirange;
/* contains 3, 4, 5, 6, 8 */
```

Operator	Description	Example	Result
=	equal	`int4range(1,5) = '[1,4]'::int4range`	t
<>	not equal	`numrange(1.1,2.2) <> numrange(1.1,2.3)`	t
<	less than	`int4range(1,10) < int4range(2,3)`	t
>	greater than	`int4range(1,10) > int4range(1,5)`	t
<=	less than or equal	`numrange(1.1,2.2) <= numrange(1.1,2.2)`	t
>=	greater than or equal	`numrange(1.1,2.2) >= numrange(1.1,2.0)`	t
@>	contains range	`int4range(2,4) @> int4range(2,3)`	t
@>	contains element	`'[2011-01-01,2011-03-01)'::tsrange @> '2011-01-10'::timestamp`	t
<@	range is contained by	`int4range(2,4) <@ int4range(1,7)`	t
<@	element is contained by	`42 <@ int4range(1,7)`	f
&&	overlap (have points in common)	`int8range(3,7) && int8range(4,12)`	t
<<	strictly left of	`int8range(1,10) << int8range(100,110)`	t
>>	strictly right of	`int8range(50,60) >> int8range(20,30)`	t
&<	does not extend to the right of	`int8range(1,20) &< int8range(18,20)`	t
&>	does not extend to the left of	`int8range(7,20) &> int8range(5,10)`	t
-|-	is adjacent to	`numrange(1.1,2.2) -|- numrange(2.2,3.3)`	t
+	union	`numrange(5,15) + numrange(10,20)`	[5,20)
*	intersection	`int8range(5,15) * int8range(10,20)`	[10,15)
-	difference	`int8range(5,15) - int8range(10,20)`	[5,10)

```
CREATE TYPE
    inetrange
AS RANGE (
    subtype = inet
);
```

```
SELECT
'[1.2.3.4,1.2.3.8]'::inetrange;
     inetrange
--------------------
 [1.2.3.4,1.2.3.8]
```

```
SELECT inetrange(
    '1.2.3.4',
    '1.2.3.8',
    '[]'
);
     inetrange
--------------------
 [1.2.3.4,1.2.3.8]
```

```
SELECT
    '1.2.3.6'::inet
        <@
    '[1.2.3.4,1.2.3.8]'::inetrange;
 ?column?
----------
 t
(1 row)
```

```
CREATE TABLE geoips (
  ips inetrange NOT NULL,
  country_code TEXT NOT NULL,
  latitude REAL NOT NULL,
  longitude REAL NOT NULL,
  CONSTRAINT geoips_dont_overlap
    EXCLUDE USING gist (ips WITH &&)
    DEFERRABLE INITIALLY IMMEDIATE
);
```

```
CREATE OR REPLACE FUNCTION
    inet_diff(x inet, y inet)
RETURNS DOUBLE PRECISION AS
$$
BEGIN
  RETURN x - y;
END;
$$
LANGUAGE 'plpgsql'
STRICT IMMUTABLE;
```

```
CREATE TYPE
    inetrange
AS RANGE (
  subtype = inet,
  subtype_diff = inet_diff
);
```

```
SELECT
    '[1.1.1.1,1.1.1.3)'::inetrange
    =
    '(1.1.1.2,1.1.1.2]'::inetrange;
 ?column?
----------
 f
(1 row)
```

Composite Types

A composite type **represents the structure of a row or record**; it is essentially just a list of field names and their data types.

A composite type represents the structure of a row or record; it is essentially just a **list of field names and their data types**.

```
//Java
class inventory_item {
    String name;
    int    supplier_id;
    float  price;
}
```

```
# 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(
    ('fuzzy dice', 42, 1.99),
    1000
);
```

Not a space

```
SELECT
    (item).name
FROM
    on_hand
WHERE
    item.price > 9.99;
```

```
SELECT
    (on_hand.item).name
FROM
    on_hand
WHERE
    (on_hand.item).price > 9.99;
```

```
UPDATE
    on_hand
SET
    item.price =
        (item).price * 1.1;
```

```
CREATE TYPE address AS (
    street text,
    city   text,
    zip    int,
    state  text
);
```

```
CREATE TYPE person AS (
    addr   address,
    given  text,
    family text
);
```

```
CREATE TABLE folks(
    them   person,
    notes  text
);
```

```
INSERT INTO
    folks(them, notes)
VALUES(((  '1 Main St',
           'Herotown',
           90210,
           'CA'),
        'Andreas',
        'Freund'),
    'The hero of the hour');
```

```
INSERT INTO
    folks(them, notes)
VALUES(((  '1 Main St',
           'Herotown',
           90210,
           'CA'),
        'Andreas',
        'Freund'),
     'The hero of the hour');
```

```
CREATE FUNCTION
    pp(addr address)
RETURNS text
LANGUAGE plpgsql
PARALLEL SAFE LEAKPROOF COST 1
AS $function$BEGIN
RETURN
    addr.street || E'\n' ||
    addr.city || E'\n' ||
    addr.state || E'\n' ||
    addr.zip;
END;$function$
```

```
# SELECT
pp((them).addr)
FROM
    folks;
    pp
-----------
 1 Main St+
 Herotown +
 CA       +
 90210
(1 row)
```

```
# SELECT
    (them).addr.pp
FROM
    folks;
    pp
-----------
 1 Main St+
 Herotown +
 CA       +
 90210
(1 row)
```

```
# SELECT
   folks.notes
FROM
    folks;
     notes
------------
 The hero of the hour
(1 row)
```

```
# SELECT
   notes(folks)
FROM
    folks;
     notes
------------
 The hero of the hour
(1 row)
```

CREATE OPERATOR

CREATE OPERATOR — define a new operator

Synopsis

```
CREATE OPERATOR name (
    {FUNCTION|PROCEDURE} = function_name
    [, LEFTARG = left_type ] [, RIGHTARG = right_type ]
    [, COMMUTATOR = com_op ] [, NEGATOR = neg_op ]
    [, RESTRICT = res_proc ] [, JOIN = join_proc ]
    [, HASHES ] [, MERGES ]
)
```

```
# CREATE TABLE inventory_item
AS (
    name                text,
    supplier_id         integer,
    price               numeric
);
```

```
# CREATE TABLE
    inventory_items
OF
    inventory_item;
```

```
# CREATE TABLE
    inventory_items
OF
    inventory_item
 (name NOT NULL,
price NOT NULL);
```

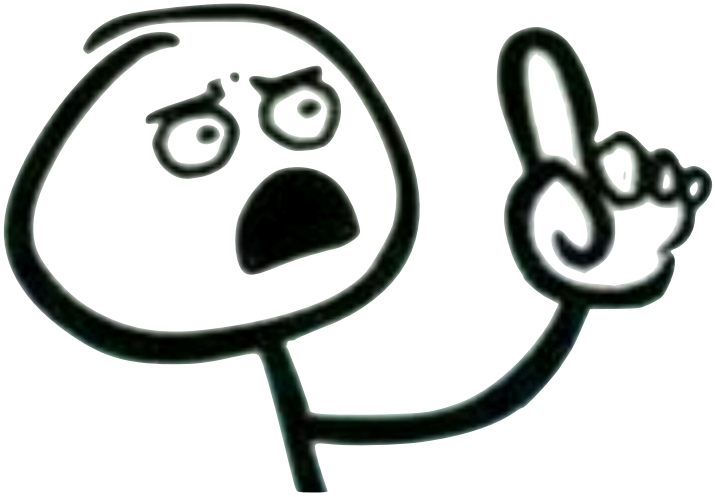
```
CREATE FUNCTION
    pp(f folks)
RETURNS
    text
LANGUAGE sql
PARALLEL SAFE LEAKPROOF COST 1
RETURN
    (f.them).given || ' ' ||
    (f.them).family || E'\n' ||
    (f.them).addr.pp;
```

```
# SELECT
    folks.pp
FROM
    folks;
    pp
-----------
 1 Main St+
 Herotown +
 CA       +
 90210
(1 row)
```

```
# select them from folks;
                              them
-----------------------------------------------------------
 ("(""1 Main St"",Herotown,90210,CA)",Andreas,Freund)
(1 row)
```

```
# select (them).* from folks;
               addr                  |  given   | family
-------------------------------------+----------+--------
 ("1 Main St",Herotown,90210,CA) | Andreas | Freund
(1 row)
```

“But that’s not *Relational*”



Wrong 1

From Wikipedia, the free encyclopedia

In database theory, a relation, as originally defined by E. F. Codd,[1] is a set of tuples (d1,d2,...,dn), where each element dj is a member of Dj, a data domain.

Wrong #1

From Wikipedia, the free encyclopedia

In database theory, a relation, as originally defined by E. F. Codd,[1] is a set of tuples (d1,d2,...,dn), where **each element dj is a member of Dj, a data domain**.

Wrong #2

From Wikipedia, the free encyclopedia

…each element [of a row] is termed an attribute value. An attribute is a name paired with a … data type.

Wrong #2

From Wikipedia, the free encyclopedia

…each element [of a row] is termed an attribute value. **An attribute is a name paired with a … data type**.

```
# SELECT
   folks.notes
FROM
    folks;
     notes
-----------
 The hero of the hour
(1 row)
```

There is an issue…

Thanks to Steven Frost

```
CREATE TYPE
    twoint AS (
        a int,
        b int
);
```

```
CREATE TABLE
    t1 (
        a int,
        b int
);
```

```
CREATE TABLE
    t2 (
        c1 twoint
);
```

```
INSERT INTO
   t1 VALUES
(1,2);

INSERT INTO
   t2 VALUES
((1,2));
```

```
# SELECT
   pg_column_size(t1.*)
FROM
   t1;
 pg_column_size
----------------
              32
(1 row)
```

```
# SELECT
   pg_column_size(t2.*)
FROM
   t2;
 pg_column_size
----------------
              53
(1 row)
```

LYDB

lydb.xyz