Bazy danych 2024

Piotr Wieczorek

30 kwietnia 2024

Funkcje PL/Python

https://www.postgresql.org/docs/current/plpython.html

```
 sudo apt install postgresql-plpython3-14
```

```
piotrek=# CREATE EXTENSION plpython3u;
```

Funkcje PL/Python

https://www.postgresql.org/docs/current/plpython.html

```
 sudo apt install postgresql-plpython3-14
```

```
piotrek=# CREATE EXTENSION plpython3u;
CREATE FUNCTION funcname (argument-list)
RETURNS return-type
AS $$
  # PL/Python function body
$$ LANGUAGE plpython3u;

CREATE FUNCTION pymax (a integer, b integer)
RETURNS integer
AS $$
    if a > b:
      return a
    return b
```

Funkcje PL/Python

```
https://www.postgresql.org/docs/current/plpython.html
```

```
 sudo apt install postgresql-plpython3-14
```

```
piotrek=# CREATE EXTENSION plpython3u;
CREATE FUNCTION funcname (argument-list)
RETURNS return-type
AS $$
  # PL/Python function body
$$ LANGUAGE plpython3u;

CREATE FUNCTION pymax (a integer, b integer)
RETURNS integer
AS $$
    if a > b:
      return a
    return b
$$ LANGUAGE plpython3u;

  piotrek=# select pymax(3, 4);
   pymax
  -------
       4
```

Funkcje PL/Python

```
CREATE FUNCTION pystrip(x text)
RETURNS text AS $$
  x = x.strip()  # error
  return x
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python

```
CREATE FUNCTION pystrip(x text)
RETURNS text AS $$
  x = x.strip()  # error
  return x
$$ LANGUAGE plpython3u;
```

Treat the function parameters as read-only!

Funkcje PL/Python: null is None

```
CREATE FUNCTION pymax (a integer, b integer)
  RETURNS integer
AS $$
  if (a is None) or (b is None):
    return None
  if a > b:
    return a
  return b
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: null is None

```
CREATE FUNCTION pymax (a integer, b integer)
  RETURNS integer
AS $$
  if (a is None) or (b is None):
    return None
  if a > b:
    return a
  return b
$$ LANGUAGE plpython3u;
```

Data types conversion: https://www.postgresql.org/docs/current/plpython-data.html

Funkcje PL/Python: returning sets

```
CREATE TYPE greeting AS (
  how text,
  who text
);
```

Funkcje PL/Python: returning sets

```
CREATE TYPE greeting AS (
  how text,
  who text
);
CREATE FUNCTION greet (how text)
  RETURNS SETOF greeting
AS $$
  # return tuple containing lists as composite types
  # all other combinations work also
  return ( [ how, "World" ], [ how, "PostgreSQL" ], [ how, "PL/Python" ] )
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: returning sets

```
CREATE TYPE greeting AS (
  how text,
  who text
);
CREATE FUNCTION greet (how text)
  RETURNS SETOF greeting
AS $$
  # return tuple containing lists as composite types
  # all other combinations work also
  return ( [ how, "World" ], [ how, "PostgreSQL" ], [ how, "PL/Python" ] )
$$ LANGUAGE plpython3u;
CREATE FUNCTION greet (how text)
  RETURNS SETOF greeting
AS $$
  for who in [ "World", "PostgreSQL", "PL/Python" ]:
    yield ( how, who )
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: accessing database

```
#plpy.execute(query [, limit])

rv = plpy.execute("SELECT * FROM my_table", 5)

foo = rv[i]["my_column"]

plan = plpy.prepare("SELECT last_name FROM my_users WHERE first_name = $1", ["text"])
rv = plpy.execute(plan, ["name"], 5)

# rv = plan.execute(["name"], 5)
```

Funkcje PL/Python: cursors

```
CREATE FUNCTION count_odd_iterator() RETURNS integer AS $$
odd = 0
for row in plpy.cursor("select num from largetable"):
    if row['num'] % 2:
        odd += 1
return odd

# by one row
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: cursors

```
CREATE FUNCTION count_odd_fetch(batch_size integer) RETURNS integer AS $$
odd = 0
cursor = plpy.cursor("select num from largetable")
while True:
    rows = cursor.fetch(batch_size)
    if not rows:
        break
    for row in rows:
        if row['num'] % 2:
            odd += 1
return odd

# in batches
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: cursors

```
CREATE FUNCTION count_odd_prepared() RETURNS integer AS $$
odd = 0
plan = plpy.prepare("select num from largetable where num % $1 <> 0", ["integer"])
rows = list(plpy.cursor(plan, [2]))  # or: = list(plan.cursor([2]))

return len(rows)
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: Accessing database - errors

- Errors cause functions to abort and raise an exception - an instance of `plpy.SPIError`

Funkcje PL/Python: Accessing database - errors

- Errors cause functions to abort and raise an exception - an instance of `plpy.SPIError`
- By default they will terminate the function.

Funkcje PL/Python: Accessing database - errors

- Errors cause functions to abort and raise an exception - an instance of `plpy.SPIError`
- By default they will terminate the function.
- This can be handled just like any other Python exception: try/except

Funkcje PL/Python: Accessing database - errors

- Errors cause functions to abort and raise an exception - an instance of `plpy.SPIError`
- By default they will terminate the function.
- This can be handled just like any other Python exception: try/except
- However: Recovering from errors caused by database access can lead to the incosistent state → subtransactions.

Funkcje PL/Python: Accessing database - errors

```
CREATE FUNCTION try_adding_joe() RETURNS text AS $$
from plpy import spiexceptions
    try:
        plpy.execute("INSERT INTO users(displayname) VALUES ('joe')")
    except spiexceptions.UniqueViolation:
        return "already have Joe"
    except plpy.SPIError:
        return "something went wrong"
    else:
        return "Joe added"
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: Accessing database - errors

```
CREATE FUNCTION try_adding_joe() RETURNS text AS $$
from plpy import spiexceptions
    try:
        plpy.execute("INSERT INTO users(displayname) VALUES ('joe')")
    except spiexceptions.UniqueViolation:
        return "already have Joe"
    except plpy.SPIError:
        return "something went wrong"
    else:
        return "Joe added"
$$ LANGUAGE plpython3u;
```

https://www.postgresql.org/docs/current/errcodes-appendix.html#ERRCODES-TABLE

Funkcje PL/Python: subtransactions

```
CREATE FUNCTION transfer_funds() RETURNS void AS $$
try:
  plpy.execute("UPDATE accounts SET blnce = blnce - 100 WHERE acc_name = 'joe'")
  plpy.execute("UPDATE accounts SET blnce = blnce + 100 WHERE acc_name = 'mary'")
except plpy.SPIError, e:
  result = "error transferring funds: %s" % e.args
else:
  result = "funds transferred correctly"

plan = plpy.prepare("INSERT INTO operations (result) VALUES ($1)", ["text"])
plpy.execute(plan, [result])
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: subtransactions

```
CREATE FUNCTION transfer_funds() RETURNS void AS $$
try:
  with plpy.subtransaction():
    plpy.execute("UPDATE accounts SET blnce = blnce - 100 WHERE acc_name = 'joe'")
    plpy.execute("UPDATE accounts SET blnce = blnce + 100 WHERE acc_name = 'mary'")
except plpy.SPIError, e:
  result = "error transferring funds: %s" % e.args
else:
  result = "funds transferred correctly"

plan = plpy.prepare("INSERT INTO operations (result) VALUES ($1)", ["text"])
plpy.execute(plan, [result])
$$ LANGUAGE plpython3u;
```

Funkcje PL/Python: controlling transactions (top level)

```
CREATE PROCEDURE transaction_test1() LANGUAGE plpython3u
AS $$
for i in range(0, 10):
    plpy.execute("INSERT INTO test1 (a) VALUES (%d)" % i)
    if i % 2 == 0:
        plpy.commit()
    else:
        plpy.rollback()
$$;
CALL transaction_test1();
```

Funkcje PL/Python: global dictionaries

- The dictionary SD stores private data between repeated calls to the same function.

Funkcje PL/Python: global dictionaries

- The dictionary SD stores private data between repeated calls to the same function.
- The dictionary GD is available to all Python functions within a session; use with care.

Funkcje PL/Python: global dictionaries

- The dictionary SD stores private data between repeated calls to the same function.
- The dictionary GD is available to all Python functions within a session; use with care.
- When a function is used as a trigger, the dictionary TD contains trigger-related values

Funkcje PL/Python: global dictionaries

- The dictionary SD stores private data between repeated calls to the same function.
- The dictionary GD is available to all Python functions within a session; use with care.
- When a function is used as a trigger, the dictionary TD contains trigger-related values
- e.g. `TD["event"]` contains the event as a string: INSERT, UPDATE, DELETE, or TRUNCATE.

Funkcje PL/Python: global dictionaries

- The dictionary SD stores private data between repeated calls to the same function.
- The dictionary GD is available to all Python functions within a session; use with care.
- When a function is used as a trigger, the dictionary TD contains trigger-related values
- e.g. `TD["event"]` contains the event as a string: INSERT, UPDATE, DELETE, or TRUNCATE.
- more: `https://www.postgresql.org/docs/current/plpython-trigger.html`

Funkcje SECURITY DEFINER

```
CREATE FUNCTION check_password(uname TEXT, pass TEXT)
RETURNS BOOLEAN AS $$
DECLARE passed BOOLEAN;
BEGIN
        SELECT  (pwd = $2) INTO passed
        FROM    pwds
        WHERE   username = $1;

        RETURN passed;
END;
$$  LANGUAGE plpgsql
    SECURITY DEFINER
    -- Set a secure search_path: trusted schema(s), then 'pg_temp'.
    SET search_path = admin, pg_temp;
```

Funkcje SECURITY DEFINER

```
BEGIN;
CREATE FUNCTION check_password(uname TEXT, pass TEXT) ... SECURITY DEFINER;
REVOKE ALL ON FUNCTION check_password(uname TEXT, pass TEXT) FROM PUBLIC;
GRANT EXECUTE ON FUNCTION check_password(uname TEXT, pass TEXT) TO wwwapplication;
COMMIT;
```

https://www.postgresql.org/docs/16/sql-createfunction.html#SQL-CREATEFUNCTION-SECURITY

Metody indeksowania

B-tree — indeksowanie wg klucza, dostosowane do dużych danych przechowywanych na dysku; zbalansowane drzewo poszukiwań o bardzo "grubych" wierzchołkach i bardzo dużej arności (w związku z tym płytkie); operatory `<,<=,=, >=,>`

hash — rozrzucanie indeksowanych danych wg funkcji haszującej do "dużych" kubełków; specyficzne metody obsługi przepełnienia kubełków (podwajanie, haszowanie liniowe, haszowanie rozszerzalne); operator =

GiST, SP-GiST, GIN, BRIN p. dokumentacja

`https://www.postgresql.org/docs/current/gist-intro.html` – balanced, tree-structured access method (R-Tree, text-search,)

`https://www.postgresql.org/docs/current/spgist.html` – non-balanced data structures (quad-trees, k-d trees, and radix trees (tries))

`https://www.postgresql.org/docs/current/gin.html` – Generalized Inverted Index, (key, posting list) pairs, (array, json)

`https://www.postgresql.org/docs/current/brin.html`– Block Range Index, very large tables with columns correlated with their physical location, e.g. sale orders with a date column (e.g., min&max within a range)

Indeksowanie za pomocą btree. Rozwijanie postgresql

- Lehman-Yao High-Concurrency btrees

Indeksowanie za pomocą btree. Rozwijanie postgresql

- Lehman-Yao High-Concurrency btrees

- postgres@github
- Developer FAQ
- pgsql-hackers
- Przykładowa dyskusja (inlinowanie podzapytań z WITH aka CTEs)
- todo

Co jest czym?

```
Nested Loop  (cost=4.65..118.62 rows=10 width=488)
                       (actual time=0.128..0.377 rows=10 loops=1)
```

- Estimated start-up cost. e.g., time to do the sorting in a sort node.
- Estimated total cost. (assumed to be run to completion, no LIMIT).
- Estimated number of rows output by this plan node. (assumed to be run to completion).
- Estimated average width of rows output by this plan node (in bytes).
- “actual time” values are in milliseconds of real time, whereas the cost estimates are expressed in arbitrary units;
- since no output rows are delivered to the client, network transmission costs and I/O conversion costs are not included.
- results on a toy-sized table cannot be assumed to apply to large tables (e.g., table on a single disk page).

EXPLAIN ANALYZE actually runs the query, any side-effects will happen

```
BEGIN;

EXPLAIN ANALYZE UPDATE tenk1 SET hundred = hundred + 1 WHERE unique1 < 100;
                                                           QUERY PLAN
----------------------------------------------------------------------------------------------------
 Update on tenk1  (cost=5.07..229.46 rows=101 width=250) (actual time=14.628..14.628 rows=0 loops=1)
   ->  Bitmap Heap Scan on tenk1  (cost=5.07..229.46 rows=101 width=250) (actual time=0.101..0.439 ro
         Recheck Cond: (unique1 < 100)
         ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.04 rows=101 width=0) (actual time=0.04
               Index Cond: (unique1 < 100)
 Planning time: 0.079 ms
 Execution time: 14.727 ms

ROLLBACK;
```

Indeksowanie

```
SELECT * FROM users WHERE displayname= 'Isaac';

CREATE INDEX i_users_displayname ON users (displayname);
```

Indeksowanie

```
SELECT * FROM users WHERE displayname= 'Isaac';

CREATE INDEX i_users_displayname ON users (displayname);


SELECT * FROM users WHERE lower(displayname)= 'isaac';

CREATE INDEX i_users_displayname ON users (lower(displayname));
```

Metody dostępu

- Sequential scan (czyta wszystko z tabeli)

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli)

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało, random access

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało, random access
- Bitmap Index Scan (najpierw zaznacza sobie strony z pasującymi krotkami i potem je przegląda)

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało, random access
- Bitmap Index Scan (najpierw zaznacza sobie strony z pasującymi krotkami i potem je przegląda) dobry gdy ?, unika random access

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało, random access
- Bitmap Index Scan (najpierw zaznacza sobie strony z pasującymi krotkami i potem je przegląda) dobry gdy ?, unika random access ale robi dwa skany

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało, random access
- Bitmap Index Scan (najpierw zaznacza sobie strony z pasującymi krotkami i potem je przegląda) dobry gdy ?, unika random access ale robi dwa skany
- Łączenie kilku Bitmap Index Scanów - możliwe za pomocą BitmapAnd / BitmapOr.

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało, random access
- Bitmap Index Scan (najpierw zaznacza sobie strony z pasującymi krotkami i potem je przegląda) dobry gdy ?, unika random access ale robi dwa skany
- Łączenie kilku Bitmap Index Scanów - możliwe za pomocą BitmapAnd / BitmapOr.
- Index Only Scan (patrzy tylko do indeksu, nie dotyka tabeli)

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało, random access
- Bitmap Index Scan (najpierw zaznacza sobie strony z pasującymi krotkami i potem je przegląda) dobry gdy ?, unika random access ale robi dwa skany
- Łączenie kilku Bitmap Index Scanów - możliwe za pomocą BitmapAnd / BitmapOr.
- Index Only Scan (patrzy tylko do indeksu, nie dotyka tabeli) - działa gdy wybieramy tylko kolumny z indeksu

Metody dostępu

- Sequential scan (czyta wszystko z tabeli) - dobry gdy pasujących krotek jest dużo
- Index Scan (sprawdza tylko pasujące krotki z tabeli) - dobry gdy pasujących krotek jest mało, random access
- Bitmap Index Scan (najpierw zaznacza sobie strony z pasującymi krotkami i potem je przegląda) dobry gdy ?, unika random access ale robi dwa skany
- Łączenie kilku Bitmap Index Scanów - możliwe za pomocą BitmapAnd / BitmapOr.
- Index Only Scan (patrzy tylko do indeksu, nie dotyka tabeli) - działa gdy wybieramy tylko kolumny z indeksu
- Jeśli indeks nie zawiera wszystkich potrzebnych kolumn to można sztucznie dodać kolumnę (→ covering index):
  np. dla `SELECT y FROM tab WHERE x = 'key';`
  `CREATE INDEX tab_x_y ON tab(x, y);`
  lub `CREATE INDEX tab_x_y ON tab(x) INCLUDE (y);`

```
student=> EXPLAIN ANALYZE SELECT * FROM users WHERE lower(displayname)= 'isaac';
-------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on users  (cost=4.31..15.48 rows=3 width=457)
                            (actual time=0.039..0.046 rows=3 loops=1)
   Recheck Cond: (lower(displayname) = 'isaac'::text)
   Heap Blocks: exact=3
   ->  Bitmap Index Scan on users_lower_idx  (cost=0.00..4.31 rows=3 width=0)
                                             (actual time=0.030..0.030 rows=3 loops=1)
         Index Cond: (lower(displayname) = 'isaac'::text)
 Planning Time: 0.157 ms
 Execution Time: 0.090 ms
```

```
student=> EXPLAIN ANALYZE SELECT * FROM users WHERE lower(displayname)= 'isaac';
---------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on users  (cost=4.31..15.48 rows=3 width=457)
                            (actual time=0.039..0.046 rows=3 loops=1)
   Recheck Cond: (lower(displayname) = 'isaac'::text)
   Heap Blocks: exact=3
   ->  Bitmap Index Scan on users_lower_idx  (cost=0.00..4.31 rows=3 width=0)
                                             (actual time=0.030..0.030 rows=3 loops=1)
         Index Cond: (lower(displayname) = 'isaac'::text)
 Planning Time: 0.157 ms
 Execution Time: 0.090 ms


student=> DROP index users_lower_idx; -- DROP INDEX
student=> EXPLAIN ANALYZE SELECT * FROM users WHERE lower(displayname)= 'isaac';
----------------------------------------------------------------------------------------------------
 Seq Scan on users  (cost=0.00..447.48 rows=43 width=457)
                    (actual time=0.029..5.877 rows=3 loops=1)
   Filter: (lower(displayname) = 'isaac'::text)
   Rows Removed by Filter: 8629
 Planning Time: 0.137 ms
 Execution Time: 5.921 ms
```

```
student=>  EXPLAIN SELECT * FROM users WHERE id=3;
--------------------------------------------------------------------------
Index Scan using users_pkey on users  (cost=0.15..8.17 rows=1 width=36)
Index Cond: (id = 3)
```

```
student=>  EXPLAIN SELECT * FROM users WHERE id=3;
-------------------------------------------------------------------------
Index Scan using users_pkey on users  (cost=0.15..8.17 rows=1 width=36)
Index Cond: (id = 3)

student=> EXPLAIN SELECT id FROM users WHERE id=3;
-----------------------------------------------------------------------------
Index Only Scan using users_pkey on users  (cost=0.15..8.17 rows=1 width=4)
Index Cond: (id = 3)
```

GiST

https://www.postgresql.org/docs/current/gist-builtin-opclasses.html#GIST-BUILTIN-OPCLASSES-TABLE

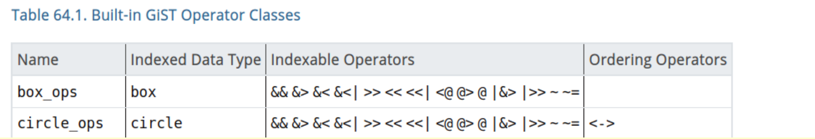

Table 64.1. Built-in GiST Operator Classes

Name	Indexed Data Type	Indexable Operators	Ordering Operators
box_ops	box	&& &> &< &<| >> << <<| <@ @> @ |&> |>> ~ ~=	
circle_ops	circle	&& &> &< &<| >> << <<| <@ @> @ |&> |>> ~ ~=	<->

. . .

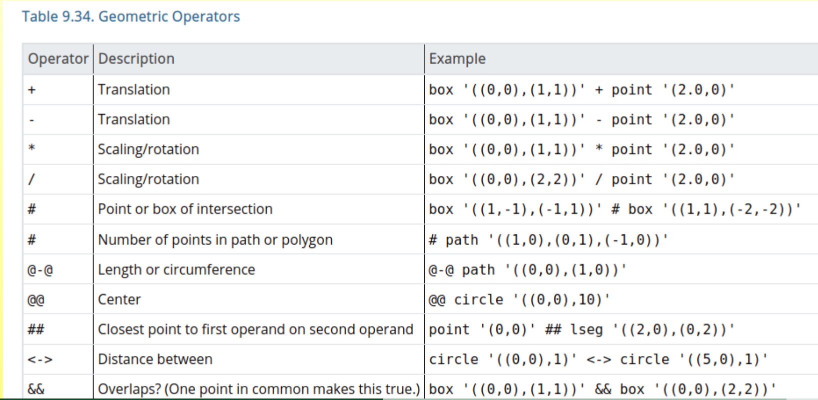

Table 9.34. Geometric Operators

Operator	Description	Example
+	Translation	`box '((0,0),(1,1))' + point '(2.0,0)'`
-	Translation	`box '((0,0),(1,1))' - point '(2.0,0)'`
*	Scaling/rotation	`box '((0,0),(1,1))' * point '(2.0,0)'`
/	Scaling/rotation	`box '((0,0),(2,2))' / point '(2.0,0)'`
#	Point or box of intersection	`box '((1,-1),(-1,1))' # box '((1,1),(-2,-2))'`
#	Number of points in path or polygon	`# path '((1,0),(0,1),(-1,0))'`
@-@	Length or circumference	`@-@ path '((0,0),(1,0))'`
@@	Center	`@@ circle '((0,0),10)'`
##	Closest point to first operand on second operand	`point '(0,0)' ## lseg '((2,0),(0,2))'`
<->	Distance between	`circle '((0,0),1)' <-> circle '((5,0),1)'`
&&	Overlaps? (One point in common makes this true.)	`box '((0,0),(1,1))' && box '((0,0),(2,2))'`

GiST

```
CREATE TABLE points(p POINT);

INSERT INTO points(p) VALUES
  (point '(1,1)'), (point '(1,4)'), (point '(4,1)'),
  (point '(4,4)'), (point '(2,2)');

INSERT INTO points(p)
 SELECT point(n*random()/10000, n*random()/10000)
 FROM generate_series(1,10000) AS n;

CREATE INDEX ON points USING GIST (p)

SELECT p FROM points WHERE p <@ box '(3,3),(7,7)'

SELECT * FROM points ORDER BY p <-> point '(0,0)' LIMIT 10;
```

https://www.postgresql.org/docs/current/functions-geometry.html#FUNCTIONS-GEOMETRY-CONV-TABLE

GiST

```
CREATE TABLE lectures(during tsrange);
INSERT INTO lectures(during) VALUES
 ('["2021-04-22 10:15","2021-04-22 12:00")');

CREATE index ON lectures USING GIST(during);

SELECT * FROM lectures where during && '[2021-04-22 11:00, 2021-04-22 11:00]';
SELECT * FROM lectures where during && '[2021-04-22 11:00, 2021-04-22 11:15)';
SELECT * FROM lectures where during <@ '(2021-04-22 11:00, 2021-04-22 11:15]'; -- no
SELECT * FROM lectures where during @> '[2021-04-22 11:00, 2021-04-22 11:15]';
```

ranges

```
(lower-bound,upper-bound)
(lower-bound,upper-bound]
[lower-bound,upper-bound)
[lower-bound,upper-bound]
empty
```

ranges

```
(lower-bound,upper-bound)
(lower-bound,upper-bound]
[lower-bound,upper-bound)
[lower-bound,upper-bound]
empty
```

- `int4range, int8range` — Range of integer/bigint
- `numrange` — Range of numeric
- `tsrange, tstzrange` — Range of timestamp without/with time zone
- `daterange` — Range of date

ranges

```
-- Containment
SELECT int4range(10, 20) @> 3;

-- Overlaps
SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);

-- Extract the upper bound
SELECT upper(int8range(15, 25));

-- Compute the intersection, i.e.,  [15,20)
SELECT int4range(10, 20) * int4range(15, 25);

-- Is the range empty?
SELECT isempty(numrange(1, 5));
SELECT isempty(numrange('empty'));
```

GIN

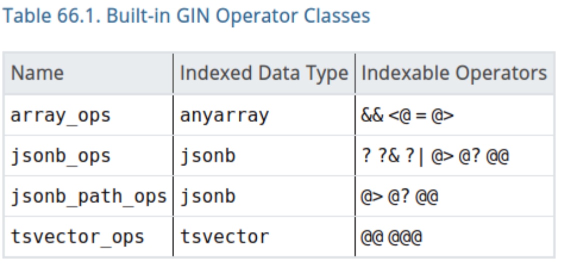

Table 66.1. Built-in GIN Operator Classes

Name	Indexed Data Type	Indexable Operators
array_ops	anyarray	&& <@ = @>
jsonb_ops	jsonb	? ?& ?| @> @? @@
jsonb_path_ops	jsonb	@> @? @@
tsvector_ops	tsvector	@@ @@@

```
-- contains
  ARRAY[1,4,3] @> ARRAY[3,1,3]
-- is contained by
  ARRAY[2,2,7] <@ ARRAY[1,7,4,2,6]
-- overlap (have elements in common)
  ARRAY[1,4,3] && ARRAY[2,1]
```