Section 2.3: Geometric Types

Name	Storage Size	Description	Representation
`point`	16 bytes	Point on a plane	(x,y)
`line`	32 bytes	Infinite line	{A,B,C}
`lseg`	32 bytes	Finite line segment	((x1,y1),(x2,y2))
`BOX`	32 bytes	Rectangular box	((x1,y1),(x2,y2))
`path`	16+16n bytes	Closed path (similar to polygon)	((x1,y1),...)
`path`	16+16n bytes	Open path	[(x1,y1),...]
`polygon`	40+16n bytes	Polygon (similar to closed path)	((x1,y1),...)
`CIRCLE`	24 bytes	Circle	<(x,y),r> (center point and radius)

Section 2.4: Network Adress Types

Name	Storage Size	Description
`CIDR`	7 or 19 bytes	IPv4 and IPv6 networks
`INET`	7 or 19 bytes	IPv4 and IPv6 hosts and networks
`macaddr`	6 bytes	MAC addresses

Section 2.5: Character Types

Name	Description
`CHARACTER varying(n)`, `varchar(n)`	variable-length with limit
`character(n)`, `char(n)`	fixed-length, blank padded
`TEXT`	variable unlimited length

Section 2.6: Arrays

In PostgreSQL you can create Arrays of any built-in, user-defined or enum type. In default there is no limit to an Array, but you *can* specify it.

Declaring an Array

```
SELECT INTEGER[];
SELECT INTEGER[3];
SELECT INTEGER[][];
SELECT INTEGER[3][3];
SELECT INTEGER ARRAY;
SELECT INTEGER ARRAY[3];
```

Creating an Array

```
SELECT '{0,1,2}';
SELECT '{{0,1},{1,2}}';
SELECT ARRAY[0,1,2];
SELECT ARRAY[ARRAY[0,1],ARRAY[1,2]];
```

Accessing an Array

By default PostgreSQL uses a one-based numbering convention for arrays, that is, an array of n elements starts with `ARRAY[1]` and ends with `ARRAY[n]`.

```
--accesing a spefific element
```

```
WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT int_arr[1] FROM arr;

int_arr
---------
        0
(1 ROW)

--sclicing an array
WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT int_arr[1:2] FROM arr;

int_arr
---------
    {0,1}
(1 ROW)
```

Getting information about an array

```
--array dimensions (as text)
WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT ARRAY_DIMS(int_arr) FROM arr;

array_dims
------------
       [1:3]
(1 ROW)

--length of an array dimension
 WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT ARRAY_LENGTH(int_arr,1) FROM arr;

 array_length
 --------------
              3
 (1 ROW)

--total number of elements across all dimensions
 WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT cardinality(int_arr) FROM arr;

 cardinality
 -------------
             3
 (1 ROW)
```

Array functions

> will be added