Python for PostgresSQL developers

# PostgresSQL Data Types.

PostgresSQL has the following data types available.

|  |  |
| --- | --- |
| Data Type Syntax | Explanation |
| char(size) | Where size is the number of characters to store. Fixed-length strings. Space padded on right to equal size characters. |
| character(size) | Where size is the number of characters to store. Fixed-length strings. Space padded on right to equal size characters. |
| varchar(size) | Where size is the number of characters to store. Variable-length string. |
| character varying(size) | Where size is the number of characters to store. Variable-length string. |
| text | Variable-length string. |

Numeric data types

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Storage Size | Description | Range |
| Smallint | 2 bytes | small-range integer | -32768 to +32767 |
| integer - | 4 bytes | typical choice for integer | 2147483648 to +2147483647 |
| bigint - | 8 bytes | large-range integer | 9223372036854775808 to 9223372036854775807 |
| decimal | variable | user-specified precision, exact | up to 131072 digits before the decimal point; up to 16383 digits after the decimal point |
| numeric | variable | user-specified precision, exact | up to 131072 digits before the decimal point; up to 16383 digits after the decimal point |
| real | 4 bytes | variable-precision, inexact | 6 decimal digits precision |
| double precision | 8 bytes | variable-precision, inexact | 15 decimal digits precision |
| serial | 4 bytes | autoincrementing integer | 1 to 2147483647 |
| bigserial | 8 bytes | large autoincrementing integer | 1 to 9223372036854775807 |

Binary data types

|  |  |  |
| --- | --- | --- |
| Name | Storage Size | Description |
| bytea | 1 or 4 bytes plus the actual binary string | variable-length binary string |

## Which data type to use?

### Integers.

* Smallint. Use only if space is at a premium, for example embedded systems.
* BigInt. BigInt has a performance penalty compared to Int.
* Int. For everything else.

### Numeric.

* Provides scale and precision
* Scale. Number of digits to the right of the decimal point.
* Precision. Total number of digits in a number.
* Be clear on what you use and why.
  + The precision should be large enough to provide the ability for the application to handle larger numbers at a future time. Example, handling amounts in thousands today and millions tomorrow.
  + The scale has to be sufficient, for example if you have an accounting application that needs to store monetary values with a fraction of the smallest currency account, for example using a scale of 3 or 4 rather than the two needed for USD pennies.
  + Be mindful of rounding and truncation in the decimal fraction and inadvertent NaN’s.
  + Avoid floating point data types for currency appplications. Floating point is designed for performance, not accuracy. In currency applications, accuracy is the more meaningful choice.
* Declarations

numeric(precision,scale)

* + Maximum number declarable is 1000
  + Max scale is 100
  + Has a special value NaN which means Not a Numer.

numeric(precision)

* + This is effectively an integer.

An example of the numeric data type:

SELECT 100 \* (0.08875)::numeric;

---

8.875

SELECT 100 \* (0.08875)::numeric(7,2);

---

9.0

SELECT (100 \* 0.08875)::numeric(7,2);

---

8.88

### Numbers – Floating Point.

* Uses the IEEE 754 standard for floating point representation
* Not exact. Unexpected behavior is possible including:
  + Overflow/Underflow
  + Equality imprecision.
* Constants
  + ‘NaN’, ‘Infinity’,’-Infinity’
* Types
  + Real => 1E-37 <=> 1E+37
  + double precision => 1E-308 <=> 1E+308
  + float(1) <=> float(24) = real
  + float(25) <=> float(53) = double precision

An example of using floats vs. the numeric data type.

|  |
| --- |
| \timing  CREATE TABLE floats (x double precision);  CREATE TABLE numerics (x numeric(15, 15));  INSERT INTO floats  SELECT random() FROM generate\_series(1,1000000);  INSERT INTO numerics  SELECT \* FROM floats;  CREATE INDEX floats\_idx ON floats (x);  CREATE INDEX numerics\_idx ON numerics (x);  SELECT \* FROM floats WHERE x >= 0.7;  -- avg 280ms  SELECT \* FROM numerics WHERE x >= 0.7;  -- avg 120ms |

* Generally better to use numeric rather than float.
* Floating Point usage is application specific
  + Reading data from a thermometer, for example.
  + When you have too many rows for larger numeric data types
  + Don’t requires a specific precision.
* You should understand the ramifications of your choice before making it.

Serial Types.

|  |  |  |
| --- | --- | --- |
| Name | Storage Size | Range |
| smallserial | 2 bytes | 1 to 32767 |
| serial | 4 bytes | 1 to 2147483647 |
| bigserial | 8 bytes | 1 to 9223372036854775807 |

### Serial Data Type.

* Serial is not really a data type, but it is a very useful convenience.

|  |
| --- |
| CREATE TABLE awesome (  id serial  ); |

|  |
| --- |
| CREATE SEQUENCE awesome\_id\_seq;  CREATE TABLE awesome (  id integer NOT NULL DEFAULT nextval(‘awesome\_id\_seq’)  );  ALTER SEQUENCE awesome\_colname\_seq OWNED BY awesome.id; |

Monetary data types

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Storage Size | Description | Range |
| money | 8 bytes | currency amount | -92233720368547758.08 to +92233720368547758.07 |

### Monetary Data Type.

* Stores monetary amounts based on a the ‘lc\_monetary’ setting.
* Output based on lc\_monetary. E.g.
  + ‘$1000.00’
* The reality.
  + Don’t use the monetary type.
  + Store currency as Integer or Numeric types.
  + Money is based on a database wide environment setting.
  + This setting can change widely between instances.
  + You cannot control the environment swetting for specific database columns.
  + Money is not a standard SQL data type. Postgres includes this for the convenience of users who are importing data from other database systems.

Datetime Types

|  |  |
| --- | --- |
| Data Type Syntax | Explanation |
| date | Displayed as 'YYYY-MM-DD'.  timestamp |
| timestamp without time zone | Displayed as 'YYYY-MM-DD HH:MM:SS'. |
| timestamp with time zone | Displayed as 'YYYY-MM-DD HH:MM:SS-TZ'.  Equivalent to timestamptz. |
| time | Displayed as 'HH:MM:SS' with no time zone. |
| time without time zone | Displayed as 'HH:MM:SS' with no time zone. |
| time with time zone. | Displayed as 'HH:MM:SS-TZ' with time zone. Equivalent to timetz. |

### Datetime Data Type.

* Format can be adjusted using the following:
  + Command: SET <datestyle>
  + Modify postgressql.conf – ‘DateStyle’ parameter
  + Environment variable: PGDATESTYLE

Examples of using the datetime data type in PostgresSql.

|  |
| --- |
| postgres=# BEGIN;  postgres=# SELECT now();  now  -------------------------------  2013-08-26 12:17:43.182331+02  postgres=# SELECT now();  now  -------------------------------  2013-08-26 12:17:43.182331+02  postgres=# SELECT clock\_timestamp();  clock\_timestamp  -------------------------------  2013-08-26 12:17:50.698413+02  postgres=# SELECT clock\_timestamp();  clock\_timestamp  -------------------------------  2013-08-26 12:17:51.123905+02 |

### Intervals.

* YEAR
* MONTH
* DAY
* HOUR
* MINUTE
* SECOND
* YEAR TO MONTH
* DAY TO HOUR
* DAY TO MINUTE
* DAY TO SECOND
* HOUR TO MINUTE
* HOUR TO SECOND
* MINUTE TO SECOND

Intervals allow us to select datetime intervals very easily.

|  |
| --- |
| postgres=# SELECT now() - interval '3 days';  ?column?  -------------------------------  2013-08-23 12:23:40.069717+02 |

Extracting datetime fields.

|  |
| --- |
| postgres=# SELECT extract(DAY FROM now());  date\_part  -----------  26  postgres=# SELECT extract(DOW FROM now());  date\_part  -----------  1 |

Converting between timezones.

|  |
| --- |
| postgres=# BEGIN;  BEGIN  postgres=# SELECT now();  now  -------------------------------  2013-08-26 12:39:39.122218+02  postgres=# SELECT now() AT TIME ZONE 'GMT';  timezone  ----------------------------  2013-08-26 10:39:39.122218  postgres=# SELECT now() AT TIME ZONE 'GMT+1';  timezone  ----------------------------  2013-08-26 09:39:39.122218  postgres=# SELECT now() AT TIME ZONE 'PST';  timezone  ----------------------------  2013-08-26 02:39:39.122218 |