HowTo create an abstract data type (ADT) in Postgresql

Very fast howto

V 0.1 of 2nd December 2003, by Laurent FALLET

Goal : Our objective here is to create an ADT called "complex" which consists in the description of a complex number, with a form $z = a + i * b$, a and b having type double.

First, let's have a look to the SQL request to define the "complex" type :

```
CREATE TYPE complex (
   internallength = 16,
   input = complex_in,
   output = complex_out,
   alignment = double
);
```

We are giving the type's name, its length (in bytes), its alignment (storage alignment), and input and output functions. These functions will realize the conversion from the external textual representation to the internal representation for *input_function* (and vice-versa for *output_function*). We'll enter complex in the table by putting values like *(2,-5)* for $z = 2 – 5i$ and this representation will be used by the operators and functions defined for the type.

So we need to define complex_in and complex_out before creating the type. We'll do this in C language. Here is an abstract of the file "**complex.c**" :

```
#include "postgres.h"

typedef struct Complex
{
      double         x;
      double         y;
}     Complex;

/* Prototypes */
Complex    *complex_in(char *str);
char       *complex_out(Complex * complex);
Complex    *complex_add(Complex * a, Complex * b);

/* Input and Output functions */
Complex *
complex_in(char *str)
{
      double         x;
      double         y;
      Complex    *result;

      if (sscanf(str, " ( %lf , %lf )", &x, &y) != 2)
      {
            elog(ERROR, "complex_in: error in parsing \"%s\"", str);
            return NULL;
      }
      result = (Complex *) palloc(sizeof(Complex));
      result->x = x;
      result->y = y;
      return result;
}

char *
complex_out(Complex * complex)
```

```
{
	char	*result;

	if (complex == NULL)
		return NULL;

	result = (char *) palloc(100);
	snprintf(result, 100, "(%g,%g)", complex->x, complex->y);
	return result;
}

/* New Operators */
Complex *
complex_add(Complex * a, Complex * b)
{
	Complex	*result;

	result = (Complex *) palloc(sizeof(Complex));
	result->x = a->x + b->x;
	result->y = a->y + b->y;
	return result;
}
```

As everybody understood, postgresql data base will only manipulate string instead of complex. complex_out transform the complex into a string. The operator complex_add was also created, we'll see later how to declare it to the data base.

Note : it's highly recommanded to write a tiny main() to test your code. Using postgres to debug your C function will be very painful.

If you try to create the complex type before adding the conversion functions into postgresql, you'll get an error (). So let's do that before :

```
CREATE FUNCTION complex_in(cstring)
   RETURNS complex
   AS 'complex.c'
   LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION complex_out(complex)
   RETURNS cstring
   AS 'complex.c'
   LANGUAGE C IMMUTABLE STRICT;
```

Then create the new type :

```
CREATE TYPE complex (
  internallength = 16,
  input = complex_in,
  output = complex_out,
  alignment = double
);
```

We can create a function whose aim will be to add 2 complex :

```
CREATE FUNCTION complex_add(complex, complex)
   RETURNS complex
   AS 'complex.c', 'complex_add'
   LANGUAGE C IMMUTABLE STRICT;
```

Finally an operator, which is going to use the function defined above :

```
CREATE OPERATOR + (
```

```
    leftarg = complex,
    rightarg = complex,
    procedure = complex_add,
    commutator = +
);
```

You can use it this way :

```
SELECT (a + b) AS c FROM test_complex;
```

There you are, you can now create any table you want with an attribute “complex”. Below you'll find some example of queries (create a table, a select, an insert) :

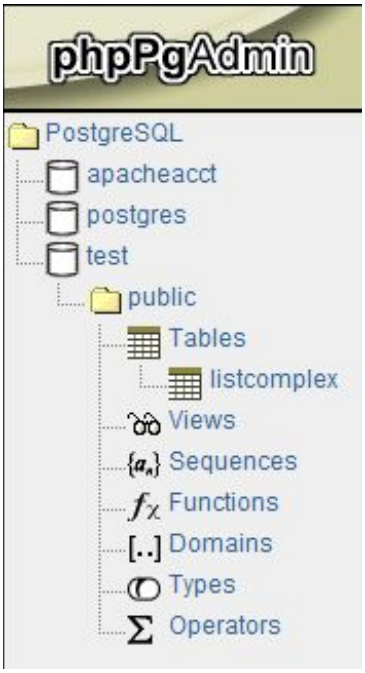
```
CREATE TABLE listcomplex (id integer, thecomplex complex)

INSERT INTO listcomplex VALUES ('4', '(2.67,5.67)')
```

Cancel the suppression of the tuple whose id is 2 by mean of an rule :

```
CREATE RULE "CancelDelete" AS ON DELETE TO listcomplex WHERE (old.id = 2) DO INSTEAD NOTHING;
```

Utilisation de phpPgAdmin :

phpPgAdmin is a web interface to manage postgresql databases, as phpMyAdmin for MySQL bases. We can see for the figure herein that it gives a whole view of tables, relations, functions, types and operators.



It enables to test queries directly without logging directly on the machine. It's very useful, as you can have the list of types easily.