# Return more than one row of data from PL/pgSQL functions

## From PostgreSQL wiki

by Stephan Szabo
Last updated 4th April 2003

PostgreSQL 7.3 now supports a much more flexible system for writing set returning functions (SRFs) that when combined with some of the new function permission options allow a greater flexibility in setting up schemas. I assume in this that you already have some experience with writing functions in SQL and PL/pgSQL for PostgreSQL. We're going to work with a very simple set of tables and data and functions written in SQL and PL/pgSQL.

```
create table department(id int primary key, name text);

create table employee(id int primary key, name text, salary int, departmentid int references department);

insert into department values (1, 'Management');
insert into department values (2, 'IT');

insert into employee values (1, 'John Smith', 30000, 1);
insert into employee values (2, 'Jane Doe', 50000, 1);
insert into employee values (3, 'Jack Jackson', 60000, 2);
```

SRFs can return either a rowtype as defined by an existing table or a generic record type. First let's look at a simple SQL function that returns an existing table's rowtype.

```
create function GetEmployees() returns setof employee as 'select * from employee;' language 'sql';
```

This very simple function simply returns all the rows from employee. Let's break down this function. The return type of the function is setof employee, meaning it is going to return a rowset of employee rows. The body of the function is a very simple SQL statement to generate the output rows.

An SRF can be used in place of a table or subselect in the FROM clause of a query. For example, to use this function to get all the information on employees with an id greater than 2 you could write:

```
select * from GetEmployees() where id > 2;
```

This is great, but what if you wanted to return something more complicated, for example, a list of departments and the total salary of all employees in that department. If you want to return an existing record type, you need to make a dummy type to hold the output type, for example:

```
create type holder as (departmentid int, totalsalary int8);
```

Here we are defining a new type named holder which is a composite type of an integer named departmentid and a bigint named totalsalary. We can then define functions that return sets of this type. For this function we'll write a version in SQL and then a version in PL/pgSQL:

```
create function SqlDepartmentSalaries() returns setof holder as
'
select departmentid, sum(salary) as totalsalary from GetEmployees() group by departmentid
'
language 'sql';
create or replace function PLpgSQLDepartmentSalaries() returns setof holder as
'
declare
r holder%rowtype;
begin
for r in select departmentid, sum(salary) as totalsalary from GetEmployees() group by departmentid loop
return next r;
end loop;
return;
end
'
language 'plpgsql';
```

The SQL is very similar to the GetEmployee() function above. It returns a rowset of rows defined by the type holder (int, int8). The rows that it returns are defined by the group by query in its body.

The PL/pgSQL function is a little more complicated, but let's go through it. The function starts off by declaring a variable r to be of the rowtype holder. This variable will be used to store the rows coming from the query in the main body of the function. The main body does a loop over the group by query stated setting r to each row in sequence. The body of the loop is the new return form, 'return next' which means that an output row is queued into the return set of the function. This does not cause the function to return. Currently, SRF returning PL/pgSQL functions must generate the entire result set before returning although if the set becomes large it will be written to disk. This limitation may be removed in a future version.

These functions are used in the same fashion as the first function,

```
select * from PLpgSQLDepartmentSalaries();
```

A PL/pgSQL function can also do additional operations on the records or only queue up only some records. For example, if you wanted to get an idea of operating expenses for departments where the overhead is 75% for departments whose total salaries were greater than 70,000 and 50% otherwise and only wanted to return department ids for departments whose salaries plus overhead was greater than 100,000 you might write something like:

```
create or replace function ExpensiveDepartments() returns setof int as
'
declare
r holder%rowtype;
begin
for r in select departmentid, sum(salary) as totalsalary
from GetEmployees() group by departmentid loop

if (r.totalsalary > 70000) then
r.totalsalary := CAST(r.totalsalary * 1.75 as int8);
else
r.totalsalary := CAST(r.totalsalary * 1.5 as int8);
end if;

if (r.totalsalary > 100000) then
return next r.departmentid;
end if;

end loop;
return;
end
'
language 'plpgsql';
```

Let's look at the differences between this and PLpgSQLDepartmentSales(). This function returns a set of integers (department ids) rather than a set of a composite type because we only need to return the id for the expensive departments. The major changes to the workings of the function are inside the loop, so let's look more closely.

```
if (r.totalsalary > 70000) then
r.totalsalary := CAST(r.totalsalary * 1.75 as int8);
else
r.totalsalary := CAST(r.totalsalary * 1.5 as int8);
end if;
```

Here we're figuring out the total salary plus the overhead and updating the record appropriately. Next, we want to determine if the totalsalary is now greater than 100,000 and if so return it's identifier, so

```
if (r.totalsalary > 100000) then
return next r.departmentid;
end if;
```

Note that for the return next we are not returning the record r, but instead are returning just the departmentid because this function returns a set of integers. If we instead had wanted to return a holder to include the salary + overhead value, we could have defined the function to return setof holder and used return next r; here.

So far, the composite type returning functions only work if you're certain that you're returning a type that is made up of the same types as the one the function is declared to return. If you make a mistake, you'll get an error at creation time for SQL functions and at execute time for PL/pgSQL functions. But what happens if you only know what the details of the composite type the function needs to return at execution time? In that case, you can return a setof record. This tells PostgreSQL that you want to the function to return an composite type but that you're going to tell it what types to expect later.

Let's make a function that returns all the rows of a table whose name you pass in as a parameter.

```
create or replace function GetRows(text) returns setof record as
'
declare
r record;
begin
for r in EXECUTE ''select * from '' || $1 loop
return next r;
end loop;
return;
end
'
language 'plpgsql';
```

Calling this function is a little more complicated than calling the SRFs above. We need to give the system an idea of what types we expect this function to return as part of the query. PostgreSQL treats these functions somewhat similarly to table subselects and uses a similar syntax for providing this information as one would use to give aliases to subselect columns.

```
select * from GetRows('Department') as dept(deptid int, deptname text);
```

Here we've passed in Department as the argument which means that we expect to get rows in the general form of Department records which is an integer followed by a text string, so we tell PostgreSQL that the alias for the result should be called dept and that it is made up of an integer named deptid and a text named deptname.

Finally, we're going to make PL/pgSQL functions that synthesize rows completely from scratch. Let's do something very simple, a function that returns the numbers from 1 to an argument and those numbers doubled. The first version uses a pre-defined type as its return type and internal type.

```
create type numtype as (num int, doublenum int);

create or replace function GetNum(int) returns setof numtype as
'
declare
r numtype%rowtype;
i int;
begin
for i in 1 .. $1 loop
r.num := i;
r.doublenum := i*2;
return next r;
end loop;
return;
end
'
language 'plpgsql';
```

It's pretty simple. The function makes a variable of the rowtype numtype and for each number from 1 to the passed argument assigns num and doublenum and then does return next r; in order to enqueue a row of output. We can do the same thing using a record type so that we do not need an outside type, however it is much more complicated and involves a bogus select.

**2003/01/13 08:19 EST (via web):** ...it would still be nice just to see how the last example could be done with a RECORD type.

**2003/01/13 13:43 EST (via web):** You have to do something like (given r as record type and returning setof record): select into r 1::int as num, 1::int as doublenum; before using r in the for loop. It's pretty ugly and when I did the discussion at SFPUG it was pretty unanimous that it was a bad hack and pretty much the wrong way to go about solving the problem. :)

**2003/01/14 01:25 EST (via web):** Thank You. Please keep on adding to this section. The other documentation is very weak on this subject

**WarMage 2003/01/28 08:04 EST (via web):** If I create a function that insert something but returns nothing (or a success/error code) how would I go about it? Here is an example of my probem :
No-return function (If possible)

```
create function InsertEmployee(int,text,int,int) returns ??? as '
insert into employee values ($1, $2, $3, $4);
' language 'SQL'
```

status return function

```
create function InsertEmployee(int,text,int,int) return int as '
insert into employee values ($1, $2, $3, $4);
//How do i capture any errors???
//if no errors return success [0] else error code or something
' language 'SQL'
```

Any help will be apreciated as i am still very new to postgresql warmage@magicmail.co.za

**justinc, 2003/01/28 16:35 EST (via web):** It would be really nice if someone (other than me) with a bit of spare time would hit the "Edit this page" link at the top of this page and fix up the comments and properly line up the examples.

**2003/02/27 11:27 EST (via web):** To Warmage: In 7.3, I believe you can make a function return void if you don't want to use its value. Technically I think you still get a result set containing a NULL, but you don't have to use a final select. As for status return, if there's an error (excepting a foreign key violation that is not yet checked - like deferred constraints) right now the statement will be ended so it won't get to the next statement in the function.

**2003/03/10 08:37 EST (via web):** Technical Assistance is available through the PostgreSQL Mailing Lists, available here:

http://www.postgresql.org/community/lists

**2003/03/14 18:39 EST (via web):** If you call your set-returning function the wrong way (IOW the way you might normally call a function), you will get this error message: Set-valued function called in context that cannot accept a set. Incorrect: select sr_func(arg1, arg2, ...); Correct: select * from sr_func(arg1, arg2, ...);

**2003/03/29 13:52 EST (via web):** GREAT!!! Its a great HELP!!!

**2003/04/01 18:21 EST (via web):** Perfect! Now, what about some samples of functions that return sets in C language?

**2003/04/04 15:21 EST (via web):** For a C language one, I believe dblink in contrib does C language functions that return a set of tuples.

2003/04/17 03:39 EST (via web): I got problem while I try to use function in a Select query : i get> error executing query declare mycursor for select * from GetEmlpoyees() WHERE id > 2 ; PostgreSQL error message: ERROR: parser parse error at or near "(" PostgreSQL status:PGRES_FATAL_ERROR Does anyone know why i can't use function in a Query ?

**2003/04/17 05:51 EST (via web):** Add your comments here...

**2003/04/17 05:53 EST (via web):** How can I cath the system errors that plpgsql return ?? If someone know that please contact me at: nmogas@xlm.pt

**2003/04/24 14:52 EST (via web):** It's very important tutorial because many people donï¿½t know how crete that type of functions(procedures), and the way to make it on PostgreSQL is so diferent with other RDBMS such as MSSQL, ORACLE, INFORMIX, INTERBASE/FIREBIRD etc.. This tutorial must become part of Postgresql Function Documentation, with more examples in many other languages than SQL and PL/PGSQL such as Python, Perl, C, etc...

**2003/04/24 16:44 EST (via web):** Note that if you don't fill in all the values for the return type for each return next, old values will be used, so you have to manually null them. This becomes an issue when denormalizing data which is too complex to handle with a select, and so must be done with nested 'for select in' loops. I run into this most often when creating complex pivot tables that do not use agrigates. I would like to see 'return next' push the return row, then set all columns to null, ready for fresh data. The following simplified example shows what I'm talking about (I know this could be done with sub-selects, but in more complicated cases it must be done interatively):

```
create type returntype as ( a int, b int, c_type1 varchar, c_type2 varchar, d_type1 varchar d_type2 varchar ); create function
```

**2003/04/24 16:48 EST (via web):** Sorry, forgot the pre /pre around my code. Here it is again. The following simplified example shows what I'm talking about (I know this could be done with sub-selects, but in more complicated cases it must be done interatively):

```
create type returntype as
(
a int,
b int,
c_type1 varchar,
c_type2 varchar,
d_type1 varchar
d_type2 varchar
);

create function tst_func() returns setof returntype as
'
declare
r returntype%rowtype;
rLoopA RECORD;
rLoopB RECORD;
begin

for rLoopA IN
select a, b from foo where argle
loop

r.a := rLoopA.a;
r.b := rLoopA.b;

for rLoopB IN
select distinct on (foo, bar) foo, bar, data from sometable where bargle
-- this select may return a row for for every column of returntype
-- if a row/column is not returned, it should show null in the result set
loop

if foo = type1 and bar = c then
r.c_type1 := rLoopB;
else
r.c_type1 := NULL; -- note the explicit set to null
end if;

if foo = type2 and bar = c then
r.c_type2 := rLoopB;
else
r.c_type2 := NULL;
end if;

if foo = type1 and bar = d then
r.d_type1 := rLoopB;
else
r.d_type1 := NULL;
end if;

if foo = type2 and bar = d then
r.d_type2 := rLoopB;
else
r.d_type2 := NULL;
end if;

end loop;

return next r;

end loop;


end;
' language 'plpgsql';
```

**sszabo, 2003/05/15 19:18 EST (via web):** I'd think it'd be better to have a way to set the rowtype explicitly (perhaps to a row value constructor) since there's also cases where setting the fields to NULL is explicitly what you don't want.

**2003/05/26 08:05 EST (via web):** When i want to try run the last example create type numtype as (num int, doublenum int); create or replace function GetNum(int) returns setof numtype as ' declare r numtype%rowtype; i int; begin for i in 1 .. $1 loop r.num := i; r.doublenum := i*2; return next r; end loop; return; end ' language 'plpgsql'; It doesnt work.... It give me this error: WARNING: Error occurred while executing PL/pgSQL function getnum WARNING: line 8 at return next ERROR: Set-valued function called in context that cannot accept a set Is there any one can tell me what wrong with it??? 2003/05/27 11:31 EST (via web): Are you calling it like select GetNum(1); or select * from GetNum(1); ?

**2003/05/28 11:34 EST (via web):** Yes, I agree.. This tutorial must become part of Postgresql Function Documentation. Im confuse about set returning function when read Documentation, but after surf www.postgresql.org , search, found this tutorial, Im glad ... Thx

**2003/05/29 08:00 EST (via web):** i'm calling it as select * from GetNum(1);

**2003/06/04 08:12 EST (via web):** Add your comments here... 2003/06/26 04:31 EST (via web): Calling function GetRows(text) error: testdb=# select * from GetRows('department') as dept(deptid integer, deptname text); ERROR: parser: parse error at or near "(" testdb=# why?

**2003/06/26 12:13 EST (via web):** I agree This document should be in PostGre documentation. Do you now a better way to create the type of the result type of the function. In fact, it's a dammage to declare a type with explicit type when we already knows the type return by the function. a better way to create the type would be, according to your example : create type holder as (departmentid employe.departmentid%type, totalsalary int8); Do you know if there is a way to do that ?

**2003/06/30 08:25 EST (via web):** There seems to be some limitation with plpgsql that prevents you from using a set-valued function in the SELECT list of a query. you can do "select foo, set_of_things(bar) from mytable" if set_of_things() is an SQL function, or a C function apparently - this started from trying to figure out how the int_array_enum() function in contrib/intagg got away with it - but not if it's a PL/pgSQL function. However, that does give you a workaround: you can call the PL/pgSQL function *from* an SQL function. A simplistic example: create function pfoo(int) returns setof int language 'plpgsql' as 'declare b alias for $1; x int; begin for x in 1..b loop return next x; end loop; return; end;'; create function foo(int) returns setof int language 'sql' as 'select * from pfoo($1)'; select 1, pfoo(5); /* will give you an error */ select 1, foo(5); /* works */ (sorry for formatting this text box is tooo wide and tooo short...)

**2003/10/14 18:11 EST (via web):** If I create a sql string and the number of column are dynamicaly determined, then how I can execute the string created? In fact setof implies that I know the kind of record, but this information is know only at runtime.

**2003/10/15 03:23 EST (via web):** Hi, as I am new to postgreSQL and functions, I try to execute the first example given above GetEmployees(). I have created the tables and records as shown above but I cant get the function run. If I give a SELECT GetEmployees(); I get a list of obvious numbers. That might be ok. But If I give a SELECT * from GetEmployees(); then I get a ---> ERROR: parser: parse error at or near "(". Does someone know what is wrong with the example? I am using postgreSQL version 7.2.2 Thank you.

**2003/10/17 19:26 EST (via web):** Newbie: This article requires PostgreSQL version 7.3 or greater. You can't do it in 7.2. -Josh

**2003/10/24 05:22 EST (via web):** Fixed the ANNOYING formatting of this page. Someone had wrapped their entire comment in pre /pre and made the page layout confoundingly wide.

**2003/10/24 16:45 EST (via web):** Just a quick note for a problem I was having. One of my tables has a recursive relationship. old records-> new records. Writing a function that returned the most current row for any given entry point was a little tricky as nothing mentioned recursion that I saw. The following is what I did.

```
create or replace function get_current_rec(numeric) returns setof rec as
'
declare
r rec%rowtype;
begin
for r in select a, b, c, d as total from table where key = $1 loop
if r.replaced_by IS NULL THEN
return next r;
ELSE
raise notice ''trying to fetch record for %'',r.replaced_by;
select into r * from get_current_rec(r.replaced_by);
return next r;
end if;
end loop;
return;
end
'
language 'plpgsql';
```

For the longest time I was stuck on getting 0 records back. Turns out selecting into r and calling next fixed that. I've tested this with 4 levels of recursion so far and its worked, so I believe it is correct.

**2003/10/24 17:31 EST (via web):** Is there a way to have a function return an agregate of custom types? Something like:

```
create type foo as (blah int, blum int);
create type bar as (words text, when timestamp);
create type things as (foo, bar);

create function something() returns setof things as '
statements;
'
```

**2003/10/25 15:33 EST (via web):** Does anyone have an example of a C function which returns a SETOF RECORD? [tablefunc.c does this, but for a ROWTYPE, not a RECORD] What would be the syntax for calling this? [Maybe: SELECT * FROM c_fcn() AS (a int, b text);]

**2003/11/03 00:12 EST (via web):** Thanks, this helped quite a bit. A caviat: if you are dealing with a WHERE clause in the EXECUTE SELECT, you may want to quote_literal() your string variables (if they are being passed in). For example: CREATE FUNCTION public.sp_get_baz_for_cust(bpchar) RETURNS SETOF bpchar AS ' DECLARE cust_id ALIAS FOR $1; baz_num CHAR( 15 ); selected_baz RECORD; BEGIN FOR selected_baz IN EXECUTE *SELECT baz_number FROM baz_table WHERE customer_id* = || quote_literal( cust_id ) LOOP RETURN NEXT selected_baz.ticket_number; END LOOP; RETURN; END; Without quote_literal(), the query tends to choke on special characters (like colons, dashes, et. al.) I think it won't like spaces much either. I tried building the string as *SELECT baz_number FROM baz_table WHERE customer_id =* ***' || cust_id || '*** - no dice. quote_literal() was the solution.

**2003/11/03 00:16 EST (via web):** Sorry for the spooge in the last posting. My first here and didn't realize I'd need to format. Here it is again in (hopefully) a bit friendlier format:

Thanks, this helped quite a bit.

A caviat: if you are dealing with a WHERE clause in the EXECUTE SELECT, you may want to quote_literal() your string variables (if they are being passed in). For example:

```
CREATE FUNCTION public.sp_get_baz_for_cust(bpchar) RETURNS SETOF bpchar AS '
DECLARE cust_id ALIAS FOR $1;
baz_num CHAR( 15 );
selected_baz RECORD;

BEGIN

FOR selected_baz IN EXECUTE ''SELECT baz_number FROM baz_table WHERE customer_id = '' || quote_literal( cust_id ) LOOP
RETURN NEXT selected_baz.ticket_number;
END LOOP;

RETURN;

END;
```

Without quote_literal(), the query tends to choke on special characters (like colons, dashes, et. al.) I think it won't like spaces much either. I tried building the string as *SELECT baz_number FROM baz_table WHERE customer_id = ' **|| cust_id ||** '* - no dice. quote_literal() was the solution.

**2004/04/05 13:55 AST (via web):** Is there any way to get the n-th item in a record? something like DECLARE rec RECORD; BEGIN rec.$1 := 1; (...)

**2004/05/22 09:02 AST (via web):** If you came here wondering how you can return multiple values from a function (like in Oracle PL/SQL): CREATE FUNCTION temp() RETURNS record DECLARE v_record RECORD; BEGIN select 1, 6, 8 into v_record; return v_record; END; Then you do: select * from temp() as (int4, int4, int4)

**2005/03/13 14:59 GMT (via web):** Add your comments here...

**2005/07/11 16:59 GMT (via web):** I'm new in working with PostgreSQL!! I have a table called "events" and anoteher called "event_parameter" and some other tables that are also conected with these two. What I want is to creat a function that will manage these tables when an event occures. Can someone help me?! Thanks

**2005/08/02 10:54 GMT (via web):** Perhaps you could use triggers

Retrieved from "https://wiki.postgresql.org/index.php?title=Return_more_than_one_row_of_data_from_PL/pgSQL_functions&oldid=17343"
Category: PL/pgSQL

---

- This page was last modified on 19 May 2012, at 09:40.