



## Table Of Contents

### From the Editors

[PostGIS 1.3.4 is finally out the door](#)

[PostGIS 1.3.5 out the door critical patch to 1.3.4 and Testing Enhancements](#)

### What's new and upcoming in PostgreSQL

[PostGIS 1.3.4 almost out the Door and 8.4 highlights](#)

### Basics

[Backing up Login Roles aka Users and Group Roles](#) *Beginner*

[Yum addendum for 8.3.5 and PgAgent](#) *Beginner*

### PL Programming

[Quick Guide to writing PLPGSQL Functions: Part 3 - NOTICES, RECURSION, and more](#) *Intermediate*

### Application Development

[Fusion Charts and PostgreSQL Part 1: Database Analysis of USDA DB](#) *Beginner*

### Product Showcase

[Fusion Charts for Sprucing up Data](#)

### Special Feature

[PostgreSQL 8.3 PLPGSQL Cheatsheet Overview](#)

### *Reader Comments*

A Product of Paragon Corporation

<http://www.paragoncorporation.com/>

<http://www.postgresonline.com/>





































We cover only a subset of what we feel are the most useful constructs that we could squash in a single cheatsheet page

**commonly used** <sup>1</sup> New in this release.

FUNCTION CACHING	RETURN constructs
<pre>IMMUTABLE STABLE VOLATILE</pre>	<pre>RETURN somevariable RETURN NEXT rowvariable RETURN QUERY <sup>1</sup></pre>
CONTROL FLOW	RAISE FAMILY
<pre>FOR i IN 1 ... numtimes LOOP     statements END LOOP;  FOR i IN REVERSE numtimes ... 1 LOOP     statements END LOOP;  FOR var_e IN EXECUTE('somedynamicsql') LOOP     statements     RETURN NEXT var_e; END LOOP;  FOR var_e IN somesql LOOP     statements     RETURN NEXT var_e; END LOOP;  IF condition THEN     : END IF;  IF condition THEN     : ELSE     : END IF;  IF condition. THEN     : ELSIF condition THEN     : ELSE     : END IF;  WHILE condition LOOP     : END LOOP;  LOOP     -- some computations     EXIT WHEN count &gt; 100;     CONTINUE WHEN count &lt; 50;     -- some computations for count IN [50 .. 100] END LOOP;</pre>	<pre>RAISE DEBUG[1-5] RAISE EXCEPTION RAISE INFO RAISE LOG RAISE NOTICE  EXCEPTION Handling  RAISE EXCEPTION 'Exception notice: %', var EXCEPTION     WHEN condition THEN         do something or         leave blank to ignore END;</pre>
	COMMON States and ERROR constants
	<pre>FOUND ROW_COUNT division_by_zero no_data_found too_many_rows unique_violation</pre>
	Variable Setting
	<pre>DECLARE     somevar sometype := somevalue;     somevar sometype     curs1 refcursor;     curs2 CURSOR FOR SELECT * FROM sometable;  somevar := somevalue  SELECT field1, field2 INTO somevar1, somevar2 FROM sometable WHERE .. LIMIT 1;</pre>
	Return types
	<pre>RETURNS somedatatype RETURNS SETOF somedatatype RETURNS refcursor RETURNS trigger RETURNS void</pre>
	QUALIFIERS
	<pre>SECURITY DEFINER STRICT  COST cost_metric <sup>1</sup> ROWS est_num_rows <sup>1</sup></pre>

### PLPGSQL FUNCTION SAMPLES

<pre>CREATE OR REPLACE FUNCTION fn_test(param_arg1 integer, param_arg2 text) RETURNS text AS \$\$ DECLARE     var_a integer := 0;     var_b text := 'test test test'; BEGIN     RAISE NOTICE 'Pointless example to demonstrate a point';     RETURN var_b    ' - '            CAST(param_arg1 AS text)    ' - '            param_arg2; END \$\$ LANGUAGE 'plpgsql' STABLE;  SELECT fn_test(10, 'test');</pre>	<pre>--Perform action-- CREATE OR REPLACE FUNCTION cp_updatesometable(param_id bigint,     param_lname varchar(50), param_fname varchar(50)) RETURNS void AS \$\$ BEGIN     UPDATE people SET first_name = param_fname, last_name = param_lname     WHERE name_key = param_id; END; \$\$ LANGUAGE 'plpgsql' VOLATILE SECURITY DEFINER;</pre>
<pre>--Example to RETURN QUERY -- CREATE OR REPLACE FUNCTION fnpgsql_get_peoplebylname_key(param_lname text) RETURNS SETOF int AS \$\$ BEGIN     RETURN QUERY SELECT name_key     FROM people WHERE last_name LIKE param_lname; END \$\$ LANGUAGE 'plpgsql' STABLE;</pre>	<pre>--Sample logging trigger taken from docs CREATE TABLE emp_audit(     operation      char(1) NOT NULL,     stamp          timestamp NOT NULL,     userid        text NOT NULL,     empname       text NOT NULL,     salary        integer ); CREATE OR REPLACE FUNCTION process_emp_audit() RETURNS TRIGGER AS \$\$ BEGIN     -- Create a row in emp_audit to reflect the operation performed on emp,     -- make use of the special variable TG_OP to work out the operation.     IF (TG_OP = 'DELETE') THEN         INSERT INTO emp_audit SELECT 'D', now(), current_user, OLD.*;         RETURN OLD;     ELSIF (TG_OP = 'UPDATE') THEN         INSERT INTO emp_audit SELECT 'U', now(), current_user, NEW.*;         RETURN NEW;     ELSIF (TG_OP = 'INSERT') THEN         INSERT INTO emp_audit SELECT 'I', now(), current_user, NEW.*;         RETURN NEW;     END IF;     RETURN NULL; -- result is ignored since this is an AFTER trigger END; \$\$ LANGUAGE plpgsql;</pre>
<pre>--Example using dynamic query -- CREATE OR REPLACE FUNCTION cp_addtextfield(param_schema_name text, param_table_name text,     param_column_name text) RETURNS text AS \$\$ BEGIN     EXECUTE 'ALTER TABLE '            quote_ident(param_schema_name)    '.'    quote_ident(param_table_name)            ' ADD COLUMN '    quote_ident(param_column_name)    ' text ';     RETURN 'done'; END; \$\$ LANGUAGE 'plpgsql' VOLATILE; SELECT cp_addtextfield('public', 'employees', 'resume');</pre>	<pre>CREATE TRIGGER emp_audit AFTER INSERT OR UPDATE OR DELETE ON emp FOR EACH ROW EXECUTE PROCEDURE process_emp_audit();</pre>