

CONTENTS INCLUDE:

- About PostgreSQL
- Configuration
- Data Types
- Commonly Used Functions
- Database Objects
- Tools and more...

Essential PostgreSQL

By Leo Hsu and Regina Obe

ABOUT POSTGRESQL

PostgreSQL is the world's most advanced open source database. It runs on numerous platforms: Linux, Unix, Windows, Mac OSX. It is simple to install, fast, sports enterprise features such as advanced spatial support via PostGIS, windowing functions, and table partitioning. It supports almost all SQL:92, SQL:1999, SQL:2003 and many SQL:2006 and SQL:2008 standards. In addition to its enterprise features, it has the added benefit of supporting numerous languages for authoring stored functions and an extensible procedural language architecture to introduce new languages.

Targeted at novices and professionals alike, this Refcard will help you quickly navigate some of PostgreSQL's most popular features as well as its hidden gems. It will cover topics such as configuration, administration, backup, language support, and advanced SQL features. Items marked with [8.4] were introduced in PostgreSQL 8.4.

CONFIGURATION

PostgreSQL uses three configuration files to control overall operations. You can find these files in the initialized data cluster (the folder specified during the initialization process using `initdb -d`).



All these can be edited with a text editor. They can be edited via PgAdmin III if `contrib/adminpack.sql` is installed in master postgres db.

File	Purpose
<code>postgresql.conf</code>	Controls the listening port, IP, and default query planner settings, memory settings, path settings, and logging settings. Can be queried via <code>pg_settings</code> database view.
<code>pg_hba.conf</code>	Controls the authentication models used by PostgreSQL and can be set per user, per database, per IP range or a combination of all.
<code>pg_ident.conf</code>	Controls mapping of an OS user to a PostgreSQL user.

postgresql.conf

Many of these service settings can be overridden for each session, database, or user/role.

Option	Description
<code>listen_addresses</code>	Use '*' to listen on all IPs of the server, 'localhost' to just local or a comma separated list of IPs to listen on.
<code>port</code>	Defaults to 5432, but can be changed to allow multiple postgresql daemon clusters/versions to coexist.
<code>search_path</code>	List of default schemas that don't need schema qualification. First schema is where non-schema qualified objects are created.
<code>constraint_exclusion</code>	Options (on, off, partial). Partial was introduced in 8.4 and is the new default. Allows planner to skip over tables if constraint ensures query conditions can not be satisfied by the table. Mostly used for table partitioning via table inheritance.

pg_hba.conf

PostgreSQL supports many authentication schemes to control access to the database. The `pg_hba.conf` file dictates which schemes are used based on the rules found in this file. You can mix and match various authentication schemes at the same time. The rules are applied sequentially such that the first match fitting a connection is the one that is used. This is important to remember because if you have a more restrictive rule above a less restrictive, then the more restrictive is the one that trumps.

The most commonly used authentication schemes are `trust` (which allows connections without a password) and `md5` which authenticates with md5 encrypted passwords. Others include: `reject`, `crypt`, `password` (this is plain text), `krb5`, `ident` (authenticate simply by identity of user in OS), `pam`, `ldap`.

The example `pg_hba.conf` entries below allows all local connections to connect to all databases without a password and all remote connections to authenticate via `md5`.

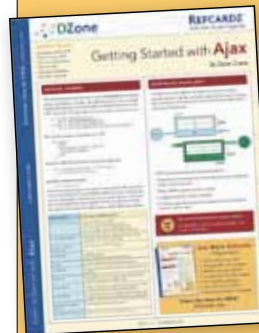
#	TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
	host	all	all	127.0.0.1/32	trust
	host	all	all	0.0.0.0/0	md5

DATA TYPES

PostgreSQL has numerous built-in types. In addition, you can define custom types. Furthermore, all tables are considered to be types in their own right, and can therefore be used within another table's column. Below are the common built-in types.

Data and Time Types

Type	Description
Date	The date is a datatype to represent dates with no time. Default representation is ISO 8601 e.g. 'YYYY-MM-DD'. Use <code>datestyle</code> configuration setting to control defaults.



Get More Refcardz (They're free!)

- Authoritative content
- Designed for developers
- Written by top experts
- Latest tools & technologies
- Hot tips & examples
- Bonus content online
- New issue every 1-2 weeks

Subscribe Now for FREE!
Refcardz.com

Timestamp	This includes both date and time and is timezone-neutral. '2009-07-01 23:00'
Timestamp with time zone	Timestamp with timezone. '2009-07-01 23:00:00-04'
Time	Time without date '23:14:20'
Time with time zone	'23:14:20-04'
Interval	A unit of time used to add and subtract from a timestamp. SELECT TIMESTAMP '2009-07-01 23:14:20' + INTERVAL '4 months 2 days 10 hours 9 seconds'
Constituents of datetime, use date_part function to extract	century, day, decade, dow (starts Sunday), doy, epoch, hour, isodow (day of week starts on Monday), minute, month, quarter, week, year

Numeric Types

Type	Description
int, int8	4 byte and 8 byte integers
serial, serial4, serial8	Sequential integers. This can be used during table creation to specify auto-numbered fields.
numeric(s,p)	Decimal numbers. s is scale and p is precision.
double precision	Floating point numbers

String Types

Type	Description
varchar(n) (aka character varying)	Max of n characters, no trailing spaces
char(n)	Padded to n characters.
text	Unlimited text

Other Types

array	Arrays in PostgreSQL are typed and you can create an array of any type. To define a column as an array of a specific type, follow with a brackets. Example: varchar(30)[]. You can also autogenerate arrays in an SQL statements with constructs such as. SELECT ARRAY['john','jane']; SELECT ARRAY(SELECT emp_name FROM employees); [Pre 8.4] SELECT array_agg(emp_name) FROM employees; [8.4]
enum	Enumerators, introduced in version 8.3 CREATE TYPE cloth_colors AS ENUM ('red','blue','green'); When used in a table, you define the column as the name of the enum. Sorting is always in the order the items appears in the enum definition.
boolean	true/false
bytea	Byte array used for storing binary objects, such as files.
lo	Large object. Stored in a separate system table with object ID reference to the large object. Useful for importing files from file system and storing and exporting back to file system.

Common Global Variables

Variable	Description
CURRENT_TIMESTAMP, now()	Returns current date and time with timezone
CURRENT_DATE	Returns current date with no time
CURRENT_TIME	Just time with no date
CURRENT_USER	returns user name of session user

COMMONLY USED FUNCTIONS

Date/Time Functions and Operators

Function	Description
age(timestamp, timestamp)	Returns an interval spanned by timestamp1 and timestamp2
age(timestamp)	Difference from current time
date_part(text,timestamp), date_part(text,interval)	Example: date_part('day', timestamp '2009-07-04 11:05:45') => 4 date_part('hour', interval '560 minutes') => 9
operators +, -, / (for intervals only)	You can add (or subtract) intervals to datetimes. You can perform subtraction between two datetimes. You can divide intervals into smaller intervals.
generate_series(timestamp, timestamp, interval) [8.4]	Generate rows of timestamps. SELECT generate_series(DATE '2009-09-12', DATE '2009-09-14', INTERVAL '10 minutes');

Text Functions and Operators

Function	Description
(string string, string number)	Concatenation
length	Number of characters in string
lpad, rpad	Left and right pad lpad('A', 5, 'X') => XXXXA rpad('A', 5, 'X') => AXXXX
lower, upper, initcap	Lower, upper, proper case.
md5	md5 hash
quote_ident	Quotes keywords and expressions not suitable for identity when unquoted. quote_ident('in') => "in" quote_ident('big') => big
quote_literal	Escapes both single and double quotes.
quote_nullable	Similar to quote_literal but doesn't quote NULL
split_part	Takes a delimited string and returns the nth item. split_part('abc def', ' ', 2) => def
strpos(text,subtext)	Returns numeric position of subtext within text.
trim, ltrim, rtrim	Trim spaces in string.

Array Functions

Function	Description
	Array concatenation ARRAY[1,2,3] ARRAY[3,4,5] => {1,2,3,3,4,5}
unnest [8.4]	Converts an array to rows SELECT anum FROM unnest(ARRAY[1,2,3]) AS anum
array_upper(anyarray, dimension) array_lower(anyarray,dimension)	returns upper/lower bound of the requested array dimension array_upper(ARRAY[ARRAY['a'], ARRAY['b']],1) => 2
array_to_string(anyarray, delimiter_text)	Converts an array to a text delimited by the delimiter. array_to_string(ARRAY[12,34], ' ') => 12 34

Other Functions

Function	Description
generate_series(int1,int2,[step])	Returns rows consisting of numbers from int1 to int2 with [step] as gaps. Step is optional and defaults to 1.
min, max, sum, avg, count	Common aggregates.
row_number,rank,dense_rank,percent_rank, lead, lag, first_value, nth_value [8.4]	window functions

DATABASE OBJECTS

Here is a listing of what you will find in a PostgreSQL server or database. An * means the object lives at the server level, not the database level.

Object	Description
Databases*	PostgreSQL supports more than one database per service/daemon.
Tablespaces*	Logical representation of physical locations where tables are stored. You can store different tables in different tablespaces, and control data storage based on database and user/group role.
User/Group roles*	Roles can have child roles. A role with login rights can be thought of as a user.
Languages	These are the procedural languages installed in the database.
Casts	PostgreSQL has the unique feature of having an extensible cast system. It has built-in casts, but allows you to define your own and override default casts. Casts allow you to define explicit behavior when casting from one object to another, as well as autocast behavior.
Schemas	These are logical groupings of objects. One can think of them as mini-databases within a larger database. An object always resides in a schema.
Tables, Views	Views are virtual tables that encapsulate an SQL SELECT statement. In PostgreSQL, tables can inherit from other tables and a query of a parent table will drill down to its children.
Rules	Rules are tied to tables or views. They are similar to triggers except they can only be written in SQL and they rewrite a statement rather than actually updating directly. Views are implemented as SELECT rules with optional DO INSTEAD inserts/update rules to make them updateable)

Functions, triggers, and aggregates	Can be written in any enabled language in the database, live in schemas. You can define your own custom aggregate functions. Trigger functions are special classes of functions that have OLD and NEW variables available that hold a pointer to the OLD and NEW data. Triggers are bound to table and a trigger function and a trigger function can be reused by many triggers.
Operators, operator classes, operator families	Live in schemas. Many are predefined, but more can be added and allow you to define things such as +, =, etc. for custom data types.
Sequences	Autogenerated when defining columns as serial. In PostgreSQL, sequences are objects in their own right and can be shared across many tables.
Types	Live in schemas. Don't forget that you have the flexibility to create your own custom data types in PostgreSQL.

TOOLS

PostgreSQL comes bundled with several tools useful for administration and query writing.

Tool	Description
psql	Command-line client packaged with PostgreSQL. Good for automating SQL jobs, copying data, outputting simple html reports.
createdb, dropdb	For creating and dropping a database from the OS shell.
PgAdminIII	Popular graphical user interface packaged with PostgreSQL.
pg_restore	Command-line tool for restoring compressed or tar backups.
pg_dump	Command-line tool for doing backups. Great for automated backups
pg_dumpall	Command-line tool for dumping all databases into a single backup.
pgAgent	A daemon/service that can be downloaded from http://www.pgadmin.org/download/pgagent.php . Used for scheduling SQL jobs and batch shell jobs. Jobs can be added easily and monitored using the PgAdmin III job interface.
pgsql2shp	Packaged with PostGIS (free spatial extender for PostgreSQL at http://www.postgis.org). Command-line tool to dump spatial data out to ESRI shapefiles and DBFs.
shp2pgsql	Packaged with PostGIS. Command-line tool that can load ESRI shapefiles or plain DBF files into PostgreSQL.
phpPgAdmin	Not packaged with PostgreSQL, but downloadable or installable via PostgreSQL application stackbuilder. Similar to phpMyAdmin, it allows administration of PostgreSQL via web interface. Also downloadable separately at http://phpPgAdmin.sourceforge.net

PSQL COMMON TASKS

PSQL is a command-line tool that allows you to run ad-hoc queries, scripts, and other useful database management routines. PSQL runs in both a non-interactive mode (straight from the OS shell prompt) and an interactive mode (PSQL terminal prompt). In both modes, the following arguments apply:

Argument	Description
-d	Database. Defaults to the user (via system identification if no user is specified).
-h	Server host. Defaults to localhost if not specified.
-p	Port. Defaults to 5432 if not specified.
-U	Username you are trying to log in with. Defaults to system user name.

PSQL Non-Interactive Mode

Getting help

```
psql -help
```

Execute an SQL script stored in a file

```
psql -h localhost -U postgres -p 5432 -f pgdumpall.sql
```

Output data in html format

```
psql -h someserver -p 5432 -U postgres -d dzone -H -c "SELECT * FROM pg_tips" -o mydata.html
```

Execute a single statement against a db

```
psql -U postgres -p 5432 -d dzone -c "CREATE TABLE test(some_id serial PRIMARY KEY, some_text text);"
```

Execute an SQL batch script against a database and send output to file

```
psql -h localhost -U someuser -d dzone -f scriptfile.sql -o outputfile.txt
```

PSQL Interactive Mode

To initiate interactive PSQL, type

```
psql -U username -p 5432 -h localhost -d dzone
```

Once you are in the the psql terminal you can perform a myriad of tasks. Below are some of the common ones.

Quit	\q
Cancel out of more screen	:q
Help on psql commands	\?
Help on SQL commands	\h some command
Switch database	\connect somedatabase
List all databases	\l
\dtv p*	List tables and views that start with p.
\du	List user/group roles and their group memberships and server level permissions
\d sometable	List columns, data types, and constraints for a table
\i somefile	Execute SQL script stored in a file.
\o somefile	Output contents to file.
Retrieve prior commands	Use up and down arrows
\timing	Toggle query timing on and off. When on, query output includes timing information.
\copy	Copy from client computer to server and from server to client computer. Example: The following command string copies data to local client computer in CSV format with header. \copy (SELECT * FROM sometable) TO 'sometable.csv' WITH HEADER CSV FORCE QUOTE

ADMIN TASKS

Backup and Restore

Below are common backup and restore statements

Create a compressed backup

```
pg_dump -h someserver -p 5432 -U someuser -F c -b -v -f "somedb.backup" somedb
```

Create a compressed backup of select tables

```
pg_dump -h localhost -p 5432 -U someuser -F c -b -f "somedb.backup" -t "someschema.table1" -t "someschema.table2" -v somedb
```

Create a compressed backup excluding a particular schema

```
pg_dump -h localhost -p 5432 -U someuser -F c -b -f "somedb.backup" -N someschema -v somedb
```

Restore a compressed backup

```
pg_restore -h localhost -d db_to_restore_to -U someuser somedb.backup
```

Restore select schemas from backup

```
pg_restore -h localhost -d db_to_restore_to -U someuser -n someschema1 -n someschema2 somedb.backup
```

Output a table of contents from backup file

```
pg_restore -l -f "toc.txt" "somedb.backup"
```

Restore only items in the table of contents

```
pg_restore -h localhost -d db_to_restore -U someuser -L "toc.txt"
"somedb.backup"
```



pg_dumpall currently only dumps to plain text sql. pg_dumpall backups must be restored with psql. For space savings and flexibility, use pg_dump. With pg_dump compressed and tar backups, you can selectively restore objects. You can not selectively restore with plain text backups.

Below are common switches used with pg_dump [D], pg_restore [R], pg_dumpall [A]. These tools are packaged with PostgreSQL and are in the bin folder. They are also packaged with pgAdmin III and are in the PgAdmin III/1.10/ folder.

Switch	Tool	Description
-b, --blobs	D	Include large objects in dump.
-d, --dbname=NAME	R	Specify name of database to restore to.
-F, --format=c t p	D R	Specify backup file format (c = compressed, t = tar, p = plain text). Plain text backups must be restored with psql.
-c, --clean	D R A	Clean (drop) schema prior to create (for pg_dumpall drop database prior to create).
-g, --globals-only	A	Dump only global objects (roles, schemas, tablespaces), no databases.
-i	D	This will ignore the version of the command-line tool and will allow processing even if an older command-line is used to dump a newer PostgreSQL server.
-j, --jobs=NUM [8.4]	R	Use this multiple parallel jobs to restore. This is especially useful for large backups and speeds them up significantly in many cases.
-l, --list	R	Print summarized TOC of the archive.
-L, use-list=filename	R	Use TOC from this file for selecting/ordering output.
-n, --schema=NAME	D R	Dump/restore only select objects in schema(s).
-N, --exclude-schema=SCHEMA	D R	Exclude from dump/restore named schema(s).
-r, --roles-only	A	Dump only roles, no database or tablespace.
-t, --table=NAME	D	Backup only named table(s) along with associated indexes, constraints, and rules.
-T, --exclude-table=NAME	D	Exclude named table(s) from backup.
-v --verbose	D R A	Controls verbosity.

User Rights Management

These are SQL commands you can use to control rights. They can be run in the PSQL interactive, via an SQL script, or via PgAdmin.

Create a new role with login rights that can create objects	CREATE ROLE somerole LOGIN NOSUPERUSER INHERIT CREATEDB NOCREATEROLE;
Create a group role with no login rights and members inherit rights of role	CREATE ROLE somerole NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE;
Add a role to another role	GRANT somerole TO someotherrole;
Give rights to a role	Example uses: GRANT SELECT, UPDATE ON TABLE sometable TO somerole; GRANT ALL ON TABLE sometable TO somerole; GRANT EXECUTE ON FUNCTION somefunction TO somerole; -- Grant execute to all users GRANT EXECUTE ON FUNCTION somefunction TO public;
Revoke rights	REVOKE ALL ON TABLE sometable FROM somerole;
Give insert/update rights to select columns [8.4]	GRANT INSERT, UPDATE (somecolumn) ON sometable TO somerole;

DATA DEFINITION (DDL)

Many of the examples we have below use named schemas. If you leave out the schema, objects created will be in the first schema defined in the search_path and dropped by searching the search path sequentially for the named object.

Create a new database	CREATE DATABASE somedatabase WITH OWNER = someuser;
Create a schema	CREATE SCHEMA someschema;
Changing database schema search path	Sets the default schema to someschema ALTER DATABASE somedatabase SET search_path = someschema, public;
Dropping objects with no dependents	A drop without a CASCADE clause will not drop an object if there are objects that depend on it, such as views, functions, and tables. For drop database you should be connected to a database other than the one you're dropping. DROP DATABASE somedatabase; DROP VIEW someview; ALTER TABLE sometable DROP COLUMN somecolumn; DROP FUNCTION somefunction;
Dropping object and all dependents. (Use with caution.)	DROP SCHEMA someschema CASCADE;
Create a table in a schema	CREATE TABLE sometable (id serial PRIMARY KEY, name character varying(150), status boolean NOT NULL DEFAULT true);
Create a child table	CREATE TABLE somechildtable (CONSTRAINT pk_somepk PRIMARY KEY (id)) INHERITS (someparenttable);
Create a check constraint	ALTER TABLE sometable ADD CONSTRAINT somecheckconstraint CHECK (id > 0);
Create or alter a view	CREATE OR REPLACE VIEW someview AS SELECT * FROM sometable [Prior to version 8.4 adding new columns to a view requires dropping and recreating]
Add a column to a table	ALTER TABLE sometable ADD COLUMN somecolumn timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP;
Add a functional index to a table	CREATE INDEX idx_someindex ON sometable USING btree (upper(somecolumn));
Create a new type	CREATE TYPE sometype AS (somecolumn integer, someothercolumn integer[]);
Create a trigger	CREATE OR REPLACE FUNCTION sometriggger() RETURNS trigger AS \$\$ BEGIN IF OLD.somecolumn <> NEW.somecolumn OR (OLD.somecolumn IS NULL AND NEW.somecolumn IS NOT NULL) THEN NEW.sometimestamp := CURRENT_TIMESTAMP; END IF; RETURN NEW; END; \$\$ LANGUAGE 'plpgsql' VOLATILE;
Add trigger to table	CREATE TRIGGER sometriggger BEFORE UPDATE ON sometable FOR EACH ROW EXECUTE PROCEDURE sometrigggerupdate();
Suppress redundant updates [8.4]	A built in trigger that prevents updates that would not change any data. CREATE TRIGGER trig_01_suppress_redundant BEFORE UPDATE ON sometable FOR EACH ROW EXECUTE PROCEDURE suppress_redundant_updates_trigger();



A table can have multiple triggers, and each trigger for a particular event on a table is run in alphabetical order of the named trigger. So if order is important, name your triggers such that they are sorted in the order you need them to run

QUERY AND UPDATE (DML)

These are examples that showcase some of PostgreSQL query features.

Retrieving data

View running queries	<pre>SELECT * FROM pg_stat_activity;</pre>
Selecting the first record of each distinct set of data	<pre>-- this example selects the store and product -- where the given store has the lowest price -- for the product. This uses PostgreSQL -- DISTINCT ON and an order by to resort -- results by product_name. SELECT r.product_id, r.product_name, r.product_price FROM (SELECT DISTINCT ON(p.product_id) p.product_id, p.product_name, s.store_name, i.product_price FROM products AS p INNER JOIN inventory As i ON p.product_id = i.product_id INNER JOIN store AS s ON i.store_id = s.store_id ORDER BY p.product_id, i.product_price) As r;</pre>
Using window function to number records (row_num)*	<pre>SELECT ROW_NUMBER() OVER(ORDER BY p.product_price) As row_num, p.product_name FROM products;</pre>
Using window function to number records by category and price *	<pre>--numbering restarts for each category SELECT ROW_NUMBER() OVER(PARTITION BY p.product_category ORDER BY p.product_price), p.product_category, p.product_price FROM products AS p ORDER BY p.product_category, p.product_price</pre>
Non-Recursive CTE with 2 CTE expressions. Note a CTE expression has only one WITH, each subexpression is separated by a , and the final query follows. Example returns the lowest priced car in each category	<pre>WITH c AS (SELECT country_code, conv_us FROM country), prices AS (SELECT p.car, p.category, p.price*c.conv_us As us_price FROM cars As p INNER JOIN c ON p.country_code = c.country_code) SELECT DISTINCT ON(category) category, car, us_price FROM prices ORDER BY category, us_price;</pre>
Recursive CTE * inventory, gives full name which includes parent tree name e.g Paper->Color->Red->20 lbs	<pre>WITH RECURSIVE tree AS (SELECT id, parentid, CAST(item As text) As fullname FROM products WHERE parentid IS NULL UNION ALL SELECT p.id,p.parentid, CAST(t.fullname '->' p.item As text) As fullname FROM products As p INNER JOIN tree AS t ON (p.parentid = t.id)) SELECT id, fullname FROM tree;</pre>

Adding and Updating Data

Insert statement with multirows	<pre>--requires 8.2+ INSERT INTO tableA(id,price) VALUES (1,5.00), (1,5.25)</pre>
Insert statement from select only load items not already in table	<pre>INSERT INTO tableA(id,price) SELECT invnew.id,invnew.price FROM tableB As invnew LEFT JOIN tableA As invold ON (invnew.id = invold.id) WHERE invold.price IS NULL;</pre>
Cross update only update items for a particular store where price has changed	<pre>UPDATE tableA SET price = invnew.price FROM tableB AS invnew WHERE invnew.id = tableA.id AND NOT (invnew.price = tableA.price);</pre>
Insert from a tab delimited file no header	<pre>COPY products FROM "/tmp/productslist.txt" WITH DELIMITER '\t' NULL As 'NULL';</pre>
Insert from a comma delimited file with header row	<pre>--these copy from the server's file system COPY products FROM "/tmp/productslist.csv" WITH CSV HEADER NULL As 'NULL';</pre>
Copy data to comma delimited file and include header	<pre>--this outputs to the server's file system COPY (SELECT * FROM products WHERE product_rating = 'A') TO '/tmp/productslist.csv' WITH CSV HEADER NULL As 'NULL';</pre>

PROCEDURAL LANGUAGES

PostgreSQL stands out from other databases in its extensive and extendable support for different languages to write database stored functions. It allows you to call out to libraries native to that language. We will list the key as well as some esoteric ones. The ones with * are preinstalled with PostgreSQL and can be enabled. Some require additional installs in addition to the language handler.

You can create set returning functions, simple scalar functions, triggers, and aggregate functions with most of these languages. This allows for languages highly optimized for a particular task to work directly with data without having to always copy it out to process as you normally would need to with a simple database storage device. Language handlers can be of two flavors trusted and untrusted. An untrusted language can access the file system directly.

```
CREATE PROCEDURAL LANGUAGE 'plpythonu' HANDLER plpython_call_handler;
CREATE OR REPLACE somename(arg1 arg1type)
RETURNS result_argtype AS
$$
body goes here
$$
LANGUAGE 'someLang';
```

Language	Description	Req
sql* (trusted)	Enabled in all dbs. Allows to write simple functions and set returning functions in just sql. The function internals are visible to the planner so in many cases performs better than other functions since the planner can strategize how to navigate based on the bigger query. It is simple, fast, but limited in functionality. CREATE OR REPLACE FUNCTION prod_state(prev numeric, e1 numeric, e2 numeric) RETURNS numeric AS \$\$ SELECT COALESCE(\$1,0) + COALESCE(\$2*\$3,0); \$\$ LANGUAGE 'sql' IMMUTABLE;	none
c*	Built in and always enabled. Often used to extend PostgreSQL e.g. postgis, pgsphere, tablefunc or for example introduce new windowing functions (introduced in PostgreSQL 8.4). Functions are referenced from a .so or .dll file. CREATE OR REPLACE FUNCTION summary(geometry) RETURNS text AS '\$libdir/postgis-1.4', 'LWGEOM_summary' LANGUAGE 'c' IMMUTABLE STRICT;	none
plpgsql* (trusted)	Not always enabled but packaged so can be installed. CREATE FUNCTION cp_upd(p_key integer, p_value varchar) RETURNS void AS \$\$ BEGIN IF EXISTS(SELECT test_id FROM testtable WHERE test_id = p_key) THEN UPDATE testtable SET test_stuff = p_value WHERE test_id = p_key; ELSE INSERT INTO testtable (test_id, test_stuff) VALUES(p_key, p_value); END IF; RETURN; END; \$\$ LANGUAGE 'plpgsql' VOLATILE;	none
perl (trusted), perl* (untrusted)	CREATE OR REPLACE FUNCTION use_quote(TEXT) RETURNS text AS \$\$ my \$text_to_quote = shift; my \$qfunc = \$_SHARED(myquote); return &\$qfunc(\$text_to_quote); \$\$ LANGUAGE perl;	perl
plpythonu (untrusted)	CREATE FUNCTION infileexists(IN fname text) RETURNS boolean AS \$\$ import os return os.path.exists(fname) \$\$ LANGUAGE 'plpythonu';	python
plr	Good for doing advanced stats and plotting using R statistical language. CREATE FUNCTION r_quantile(float8[]) RETURNS float8[] AS \$\$ quantile(arg1, probs = seq(0, 1, 0.25), names = FALSE) \$\$ LANGUAGE 'plr';	R

<p>plsh (untrusted)</p>	<p>Allows to write in shell script CREATE FUNCTION callscript(id integer) RETURNS text AS \$\$ #!/bin/sh wget -q "http://somesite.com/somepage?id=\$1" >/dev/null 2>&1 echo "done" \$\$ LANGUAGE 'plsh' VOLATILE;</p>	<p>sh</p>
------------------------------------	---	------------------

Common Procedural Tasks

Create a trigger and use in table

```
CREATE FUNCTION mytable_ft_trigger() RETURNS trigger AS $$
BEGIN
NEW.tsv :=
setweight(to_tsvector('pg_catalog.english',
coalesce(new.field1, '')), 'A') ||
setweight(to_tsvector('pg_catalog.english',
coalesce(new.field2, '')), 'B');
RETURN NEW;
END
$$ LANGUAGE plpgsql;
CREATE TRIGGER mytable_trigiu BEFORE INSERT OR UPDATE
ON mytable FOR EACH ROW EXECUTE PROCEDURE mytable_ft_trigger();
```

Return sets and use of out params

```
CREATE OR REPLACE FUNCTION
fn_sqltestmulti(param_subject varchar,
OUT test_id integer,
OUT test_stuff text)
RETURNS SETOF record
AS
$$
SELECT test_id, test_stuff
FROM testtable
WHERE test_stuff LIKE $1;
$$
LANGUAGE 'sql' STABLE;
--example
SELECT * FROM fn_sqltestmulti('%stuff%');
```

Create an aggregate function

```
CREATE AGGREGATE sum(text) (
SFUNC=textcat,
STYPE=text,
INITCOND='');
SELECT item, sum(category || ' ') as categories
FROM item_cat GROUP BY item;
```

ABOUT THE AUTHORS



The wife and husband team of Leo Hsu and Regina Obe founded Paragon Corporation in 1997 specializing in database technology. Paragon Corporation works with numerous organizations to design, develop and maintain database and web applications. In 2002, Leo and Regina started to dabble in the growing field of spatial analysis and have become active participants in the on-going development of PostGIS which is a spatial extension of PostgreSQL. Regina is a member of the PostGIS core development team and Project Steering Committee.

Through Paragon Corporation, Leo and Regina have helped many clients using PostgreSQL, SQL Server and MySQL. Paragon Corporation takes on database projects in a wide range of industries and advocates database-driven development.

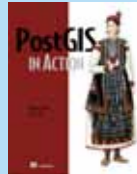
Leo and Regina met at MIT and both graduated with engineering degrees. Leo went on to obtain a master's degree from Stanford University in Engineering of Economic Systems.

They maintain two sites: <http://www.postgresonline.com> -- provides tips and tricks for using PostgreSQL and <http://www.bostongis.com> - provides tips and tricks for using PostGIS, SQL Server 2008 Spatial and other open source and open GIS tools.

Email contact: lr@pcorp.us

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

RECOMMENDED BOOK

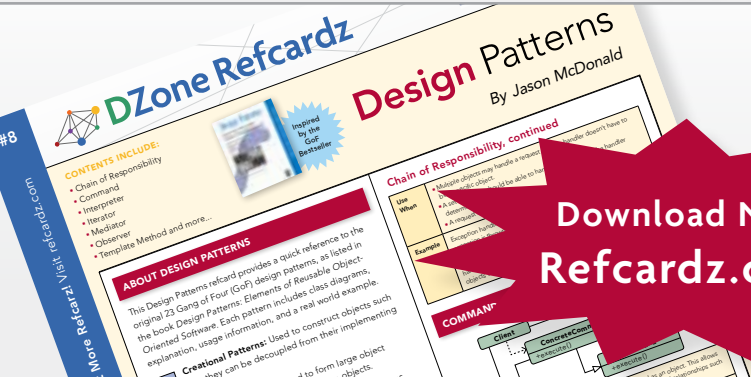


PostGIS in Action is the first book devoted entirely to PostGIS. It will help both new and experienced users write spatial queries to solve real-world problems. For those with experience in more traditional relational databases, this book provides a background in vector-based GIS so you can quickly move to analyzing, viewing, and mapping data. Advanced users will learn how to optimize queries for maximum speed, simplify geometries for greater efficiency, and create custom functions suited specifically to their applications. It also discusses the new features available in PostgreSQL 8.4 and provides tutorials on using additional open source GIS tools in conjunction with PostGIS.

Pre-Order Now

<http://www.manning.com/obe/>

Professional Cheat Sheets You Can Trust



Download Now
Refcardz.com

"Exactly what busy developers need: simple, short, and to the point."

James Ward, Adobe Systems

Upcoming Titles

- RichFaces
- Agile Software Development
- BIRT
- JSF 2.0
- Adobe AIR
- BPM&BPMN
- Flex 3 Components

Most Popular

- Spring Configuration
- jQuery Selectors
- Windows Powershell
- Dependency Injection with EJB 3
- Netbeans IDE JavaEditor
- Getting Started with Eclipse
- Very First Steps in Flex



DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513
888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com



\$7.95