Official PostgreSQL 9.0 Documentation URL: http://www.postgresql.org/docs/9.0/static/
For more examples of new features -- check out: http://wiki.postgresql.org/index.php?title=PostgreSQL_9.0

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

[1] New in this release.

[2] Enhanced in this release.

COMMON BUILT-IN DATA TYPES

Below are common data types with common alternative names.
Note: There are many more and one can define new types with create type.
All table structures create an implicit type struct as well.

datatype[] - e.g. varchar(50)[] (defines an array of a type)

```
bit
boolean
bytea
character varying(length) - varchar(length)
character(length) - char(length)
date
enum
double precision - float4 float8
integer - int4
bigint - int8
network address (inet, cidr, macaddr)
money
numeric(length,precision)
built-in (non-PostGIS) geometry types
        point, lseg,box,path,polygon,circle
oid
serial - serial4
bigserial - serial8
text
time without timezone - time
time with timezone - timez
timestamp without timezone - timestamp
timestamp with timezone - timestampz
tsquery
tsvector
uuid (aka GUID)
xml
```

CONTRIBS AND DATATYPES

means distributed separately* deprecated dep
included data types in ()

```
adminpack - pgAdmin admin pack
auto_explain[2]-- explain plan logging
citext (citext) - case insensitive text type
cube (cube) - multi-dimensional cube type
dblink - cross database/server queries
earthdistance - earth dist functions (depends on cube)
fuzzstrmatch - fuzzy string match
ltree (ltree)- hierarchical tree type
hstore[2] (hstore) - key value store type
moddatetime - moddatetime trigger
passwordcheck[1] ensure strong passwords
pg_bench benchmarking queries
pg_buffercache - inspect buffer cache
pg_trgm - trigrams for fuzzy search
pgcrypto - cryptography functions
pgAgent* - job agent
pgsphere* (scircle, sellipse, spoint,sline,spolygon, spath,strans ..)
    - spherical data types often used for astronomy
pldbg - pl debugger
postgis* (geography, geometry, raster)
    - raster packaged separately pre-2.0 (integrated in 2.0+)
jaspa* (geometry)
        - postgis like implementation of geometry and functions
        - implemented in pljava/java instead of C/c++
tablefunc -- crosstab queries
temporal* (period) -- support for time periods
uuid-ossp -- generating uuids
vacuum_lo - delete orphaned large objects

-- although xml2 is deprecated
-- some functionality exists still
-- not present in the built-in xml
-- e.g. xslt_process function
xml2[dep]
```

Admin Functions

COPY .. FROM ..
COPY .. TO ..
current_setting
pg_cancel_backend
pg_column_size
pg_database_size
pg_relation_size
pg_size_pretty
pg_tablespace_size
pg_total_relation_size
set_config
vacuum analyze verbose
vacuum full

Languages

* packaged separately
c ,plpgsql, sql[2]
pljava
plperl(u)[2]
plproxy*
plpython[2]
plpython3[1]
plr*
plruby*
plscheme*
plsh*
pltcl

Command Line

pgbench
pg_dump
pg_dumpall
pg_resetxlog
pg_restore
pg_standby
pg_upgrade [2]
psql
vacuumdb
vacuumlo

Large Object

Server

lo_create
lo_export, lo_import
lo_unlink

Client

lo_close
lo_create
lo_export, lo_import
lo_lseek
lo_open
lo_read
lo_tell
lo_unlink
lo_write

Common Functions

cast, ::
coalesce
generate_series(start, stop [,step])
generate_series(start timestamp,
stop timetamp [,step inteval])
greatest(val1,val2,val3....)
least(val1,val2,val3,...)
nullif
random

Sequence (Serial) Functions

currval
lastval(serialname)
nextval(serialname)

String Functions

||
ascii
chr
convert_from(string bytea, src_encoding name)
convert_to(string text, dest_encoding name)
encode(data bytea, ['base64', 'hex', 'escape'])
initcap
length
lower
lpad
ltrim
md5
octet_length
position(substring in string)
quote_ident
quote_literal
quote_nullable
regexp_matches
regexp_replace
regexp_split_to_array(
string, pattern [, flags])
regexp_split_to_table(
string, pattern [, flags])
repeat
replace
rpad
rtrim
split_part
string_agg[1]
strpos
substr
translate(string text, from text, to text)
trim
upper

Database Globals

current_date
current_time
current_timestamp
current_user
localtime

Date and timestamp Functions

age(timestamp[,timestamp])
date_part(text, timestamp)
century
day
decade
dow
doy
epoch
hour
minute
second
millisecond
microsecond
month
quarter
second
isodow
week
year
isoyear
millennium
date_trunc(text,timestamp)
extract(*field* from *interval*|*timestamp*)
interval
to_char
to_date
to_timestamp

Date Predicates

overlaps

Array Constructors

ARRAY[[4,5,6],..]
ARRAY()
array_agg
array_append
array_cat
array_dims
array_prepend
array_to_string
SOME(array)
regexp_split_to_array(string, pattern [, flags])
string_to_array

Array Operators

=
<>
<
>
<=
||

Array Functions Other

array_lower(anyarray, int)
array_upper(anyarray, int)
generate_subscripts(array anyarray, dim int [,reverse boolean])
unnest

Math Operators

% , ^, |/
||/,!, !!
@, &, |
#,~, <<
>>

Math Functions

This is a subset
abs
cbrt
ceiling
degrees
exp
floor
log
ln
mod
pi
power
radians
random
sqrt
trunc

Trig Functions

acos
asin
atan
atan2
cos
cot
pi()
sin
tan

Enums

> < <= >= =
enum_cmp
enum_first
enum_larger
enum_last
enum_range
enum_smaller

XML

database_to_xml
database_to_xmlschema
query_to_xml
query_to_xml_and_xmlschema
table_to_xml
xmlagg
xmlattributes
xmlcomment
xmlconcat
xmlelement
xmlforest
xpath
xmlpi
xmlroot

JOIN Types

```
CROSS JOIN
EXCEPT (ALL)
FULL JOIN
[INNER] JOIN

INTERSECT (ALL)
LEFT JOIN
NATURAL JOIN
RIGHT JOIN
UNION (ALL)
```

SQL Keywords

```
ANY(array)
BETWEEN .. AND
CASE WHEN .. END
DELETE FROM
DISTINCT
DISTINCT ON
DO [1]
    for anonymous functions (plpgsql, lolcode, plperl)
EXISTS
FROM
GROUP BY
HAVING
ILIKE
IN(..)
LIKE
LIMIT ..OFFSET
NOT
NOT IN(..)
NULLS FIRST
NULLS LAST
ORDER BY
someagg(.. ORDER BY somefield1, ..somefieldn)[1]
SELECT
SET
SIMILAR TO
TRUNCATE TABLE
UPDATE
USING
WHERE
```

COMMON TABLE EXPRESSION (CTE)

RECURSIVE keyword is required if any expression is RECURSIVE

```
WITH [RECURSIVE] tablevar1
    AS (table_sql_def),
    .., tablevarn AS (table_sql_defn)
    final_query
```

Window Keywords

```
BETWEEN frame_start AND frame_end[1]
CURRENT ROW
ORDER BY
OVER
PARTITION BY
RANGE | ROWS [2]
UNBOUNDED FOLLOWING
UNBOUNDED PRECEDING
```

Window Functions

In addition to window functions any aggregate function can be used in a window expression

```
cume_dist dense_rank
first_value
lag lead last_value
ntile nth_value
percent_rank rank
row_number
```

Aggregates

```
(For all aggregates you can also use:
        someagg(somefield ORDER BY somefield1,....somefieldn)
    someagg(DISTINCT somefield)
    someagg(DISTINCT somefield
        ORDER BY somefield) [1]
array_agg
avg
bit_and, bit_or
boolean_and, boolean_or
corr
count
covar_pop, covar_samp
every
max
min
regr_avgx, regr_avgy
regr_count
regr_intercept
regr_r2
regr_slope
regr_sxx
regr_sxy
regr_syy
stddev
stddev_pop
stddev_samp
string_agg(expression, delimeter)[1]
sum
variance
var_pop
var_samp
xmlagg
```

Key pg_catalog Tables/Views

```
pg_class
pg_rules
pg_settings
pg_stat_activity
pg_stat_database
pg_tablespaces
```

DDL

```
ADD CONSTRAINT
CREATE AGGREGATE
CREATE CAST
CREATE (DEFAULT) CONVERSION
CREATE DATABASE
CREATE DOMAIN
CREATE [OR REPLACE] FUNCTION
CREATE (UNIQUE) INDEX
CREATE LANGUAGE
CREATE OPERATOR
CREATE OPERATOR FAMILY
CREATE RULE
CREATE SCHEMA
CREATE SEQUENCE
CREATE TABLE[2]
CREATE TABLESPACE [2]
CREATE TEXT SEARCH DICTIONARY
CREATE TRIGGER [2]
ALTER TABLE
CREATE TYPE
CREATE [OR REPLACE] VIEW
DROP objecttype object_name[IF EXISTS]
ALTER ..(supported for DATABASE, TABLE, TABLESPACE)
ALTER TABLE .. DROP COLUMN [IF EXISTS] [1]
ALTER TABLE .. DROP CONSTRAINT [IF EXISTS] [1]
ALTER TABLE ..  ADD CONSTRAINT .. EXCLUDE[1]
```

DCL

```
CREATE ROLE
GRANT ALL ON SCHEMA ...
GRANT
    [ALL, INSERT, SELECT, UPDATE, DELETE,
     TRUNCATE, REFERENCES, TRIGGER]
    ON TABLES to somerole [1]
GRANT [EXECUTE] ON ALL FUNCTIONS [1]
ALTER DEFAULT PRIVILEGES IN SCHEMA[1]
REVOKE
    [ALL ..]
```

1

PostgreSQL Keywords

```
EXPLAIN ANALYZE VERBOSE
EXPLAIN (ANALYZE true, COSTS true, FORMAT json|yaml|xml) [1]
```

Key information_schema Views

```
columns
column_privileges
enabled_roles
key_column_usage
referential_constraints
routines (lists all functions)
sequences
schemata
tables
views
```

EXAMPLES OF NEW FEATURES IN POSTGRESQL 9.0 (DDL)

For some of these examples, we will use the contrived table.
Demonstrates new ability to define primary/unique key deferrable

```
CREATE TABLE passengers(
    passenger_id serial PRIMARY KEY DEFERRABLE INITIALLY DEFERRED,
        passenger_name varchar(100),
        weight integer, aisle varchar(10));
INSERT INTO passengers(passenger_name, weight, aisle)
        VALUES ('Jack', 200, '18'), ('Jill', 150, '20'),
                ('Cathy', 150, '20') , ('Simon',1000, '18');
```

SELECT INTO and CREATE TABLE AS
now return row counts to the client in their command tags

```
SELECT aisle, COUNT(*) as tally INTO passengers_tally
FROM passengers GROUP BY aisle;
-- result
SELECT 2
```

Allow IF EXISTS drop on columns and constraints

```
ALTER TABLE passengers_tally DROP COLUMN
        IF EXISTS age ;
ALTER TABLE passengers_tally DROP CONSTRAINT
        IF EXISTS pk_passengers_tally;
```

Using exclusion constraints - PostGIS example
- no overlapping point bounding box

```
CREATE TABLE poi(pt_id serial PRIMARY KEY,
        pt geography(Point,4326));
ALTER TABLE poi ADD CONSTRAINT uidxb_poi
        EXCLUDE USING gist (pt WITH &&);
```

EXAMPLES OF NEW FEATURES IN POSTGRESQL 9.0 (DML)

-- swapping keys -- Note I can get away with this
even though the update will before completion cause a key violation because passenger_id is marked as DEFERRED

```
UPDATE passengers
    SET passenger_id =
        CASE WHEN passenger_id = 4 THEN 1 ELSE passenger_id + 1 END;
```

--Using ORDERED aggregates and the new string_agg aggregate function

```
SELECT aisle, string_agg(passenger_name, '|' ORDER BY weight) As pass_list_by_wgt,
        string_agg(passenger_name, '|' ORDER BY passenger_name) As pass_list_name,
        array_agg(DISTINCT weight ORDER BY weight)   As arr_weight
FROM passengers
GROUP BY aisle ORDER BY aisle;
```

-- results

```
 aisle | pass_list_by_wgt | pass_list_name | arr_weight
-------+------------------+----------------+------------
 18    | Jack|Simon       | Jack|Simon     | {200,1000}
 20    | Jill|Cathy       | Cathy|Jill     | {150}
```

--Using WINDOW FUNCTIONS ROWS

```
SELECT aisle, passenger_name, weight,
 SUM(weight) OVER (ORDER BY weight, aisle, passenger_name
    ROWS BETWEEN 0 PRECEDING AND 2 FOLLOWING) As weight_look_2_ahead
FROM passengers
ORDER BY weight,aisle,passenger_name;
```

-- results --

```
 aisle | passenger_name | weight | weight_look_2_ahead
-------+----------------+--------+---------------------
 20    | Cathy          |    150 |                 500
 20    | Jill           |    150 |                1350
 18    | Jack           |    200 |                1200
 18    | Simon          |   1000 |                1000
```

WINDOW FUNCTION RANGE - tally weight of all people of equal or lower weight in same aisle

```
SELECT aisle, passenger_name, weight,
 SUM(weight) OVER (PARTITION BY aisle ORDER BY weight
    RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) As weight_aisle_lower
FROM passengers
ORDER BY aisle,weight,passenger_name;
```

-- result --

```
 aisle | passenger_name | weight | weight_aisle_lower
-------+----------------+--------+--------------------
 18    | Jack           |    200 |                200
 18    | Simon          |   1000 |               1200
 20    | Cathy          |    150 |                300
 20    | Jill           |    150 |                300
```

EXAMPLES OF NEW FEATURES IN POSTGRESQL 9.0 - FUNCTION ENHANCEMENTS

Anonymous function using new DO command

- adds a date_add field to all tables in public schema that don't already have one

```
DO $$
DECLARE var_searchsql text;
BEGIN
    var_searchsql :=  string_agg('ALTER TABLE '
    || t.table_schema || '.'
    || t.table_name
        || ' ADD COLUMN date_add timestamp DEFAULT(current_timestamp)', ';')
                            FROM information_schema.tables t
        LEFT JOIN information_schema.columns c
        ON (t.table_name = c.table_name AND
            t.table_schema = c.table_schema AND c.column_name = 'date_add')
    WHERE t.table_type = 'BASE TABLE' AND t.table_schema = 'public'
    AND c.table_name IS NULL;
    IF var_searchsql > '' THEN
        EXECUTE var_searchsql;
    END IF;
END$$ language plpgsql;
```

function named parameters

```
CREATE FUNCTION random_data(num_records integer,
                multiplier float)
    RETURNS SETOF float AS
$$
    SELECT random()*i*$2
        FROM generate_series(1,$1) As i;

$$ language 'sql';

-- calling the function with named parameters --
SELECT foo.i
FROM random_data(multiplier:= 1000,
    num_records:= 10) As foo(i);
```

EXAMPLES OF NEW FEATURES IN POSTGRESQL 9.0 (DCL)

These examples will use these roles,database, schema

```
CREATE ROLE jungle;
CREATE ROLE regina LOGIN CREATEDB PASSWORD 'queen^warrior';
GRANT jungle TO regina;
CREATE ROLE leo LOGIN PASSWORD 'lion@king.dom';
CREATE DATABASE kingdom OWNER regina;

REVOKE SELECT ON ALL TABLES IN SCHEMA public FROM jungle;
```

Granting/Revoking permissions on existing tables in schema public

```
GRANT ALL PRIVILEGES ON
        ALL TABLES IN SCHEMA public TO jungle;
```

Granting all permissions on future tables in schema public to jungle.

```
-- There are some nuances we won't get into such as permissions are only granted
-- to objects that the grantor has permissions to GRANT
-- using the optional [FOR role] option allows
-- you to designate a different grantor role other than current user
-- as long as current user/has rights to promote to said role.
ALTER DEFAULT PRIVILEGES IN SCHEMA public
        GRANT ALL PRIVILEGES ON TABLES TO jungle
        GRANT USAGE, SELECT, UPDATE ON SEQUENCES TO jungle;

ALTER DEFAULT PRIVILEGES IN SCHEMA public
        GRANT USAGE, SELECT, UPDATE ON SEQUENCES TO jungle;
```

ADMIN EXAMPLES

```
SELECT pg_size_pretty(pg_tablespace_size('pg_default')) as tssize,
        pg_size_pretty(pg_database_size('somedb')) as dbsize,
        pg_size_pretty(pg_relation_size('someschema.sometable')) as tblsize;

--Example importing data to table sometable
--from tab delimited where NULLs appear as NULL
COPY sometable FROM '/path/to/textfile.txt' USING DELIMITERS '\t' WITH NULL As 'NULL';
--Example importing data to table sometable
--from csv delimited that includes field headers
COPY sometable FROM 'C:/somefile.csv' WITH CSV HEADER;

--Example exporting a query to a comma separated (CSV) called textfile.csv
--setting NULLS to text NULL
COPY (SELECT * FROM sometable WHERE somevalue LIKE '%') TO '/path/to/textfile.csv'
 WITH NULL As 'NULL' CSV HEADER QUOTE AS '"';
```

Vacuuming

```
vacuum analyze verbose;
vacuum sometable;
vacuum full;
--Kills all active queries in selected db and list out process id
--and usename of process and if kill successful
SELECT procpid, usename, pg_cancel_backend(procpid)
FROM pg_stat_activity
WHERE datname = 'somedb';

--introduced in 8.3 - terminates backedn
SELECT procpid, usename, pg_terminate_backend(procpid)
FROM pg_stat_activity
WHERE datname = 'somedb';
```

DDL EXAMPLES

```
CREATE DATABASE somedb
  WITH OWNER = someuser
  WITH ENCODING='UTF8' TEMPLATE=template0 TABLESPACE = pg_default
       LC_COLLATE = 'English_United States.1252'
       LC_CTYPE = 'English_United States.1252';

CREATE TABLE orders(order_id serial NOT NULL,
  order_addeddt timestamp with time zone,
  order_rating rating,
  CONSTRAINT pk_orders_order_id PRIMARY KEY (order_id)
);

CREATE TYPE rating AS
 ENUM('none', 'bronze', 'silver', 'gold', 'platinum');
```

```
CREATE OR REPLACE FUNCTION first_element_state(anyarray, anyelement)
  RETURNS anyarray AS
$$
    SELECT CASE WHEN array_upper($1,1) IS NULL
        THEN array_append($1,$2) ELSE $1 END;
$$ LANGUAGE 'sql' IMMUTABLE;

CREATE OR REPLACE FUNCTION first_element(anyarray)
  RETURNS anyelement AS
$$
    SELECT ($1)[1] ; $$ LANGUAGE 'sql' IMMUTABLE;

CREATE AGGREGATE first(anyelement) ( SFUNC=first_element_state,
  STYPE=anyarray, FINALFUNC=first_element);
```

SELECT EXAMPLES

```
SELECT o.order_id, o.order_date, o.approved_date,
 COUNT(i.item_id) As nlineitems,
 SUM(i.unit_price*i.num_units) As total
FROM orders o
 INNER JOIN orderitems i ON o.order_id = i.order_id
GROUP BY o.order_id, o.order_date, o.approved_date
HAVING SUM(i.unit_price*i.num_units) > 200
ORDER BY o.approved_date NULLS FIRST;
```

CTE Example (8.4+)

```
WITH pt(x,y) AS (
   SELECT 100*random(), 200*random()
    FROM generate_series(1,10) As i
),
pt2(x,y) AS
( SELECT generate_series(1,2) As x, generate_series(4,5) )
SELECT pt.x + pt2.y AS a, pt2.x*pt.y AS b
FROM pt CROSS JOIN pt2;
```

UPDATE/INSERT/DELETE EXAMPLES

```
UPDATE sometable SET somevalue = 5
WHERE sometable.somename = 'stuff';

UPDATE sometable
 SET calccount = s.thecount
  FROM (SELECT COUNT(someothertable.someid) as thecount,
    someothertable.someid
    FROM someothertable
     GROUP BY someothertable.someid) s
   WHERE sometable.someid = s.someid;

DELETE FROM sometable
 WHERE somevalue = 'something';
```

--This only works on 8.1+ --

```
INSERT INTO orders(order_addeddt, order_rating)
 VALUES ('2007-10-01 20:40', 'gold'),
 ('2007-09-01 11:00 AM', 'silver'),
   ('2007-09-02 10:00 PM', 'none'), ('2007-10-10 PM', 'bronze');

--Pre 8.1+ only supports single values inserts
INSERT INTO orders(order_addeddt, order_rating)
 VALUES ('2007-10-01 20:40', 'gold');

--This is a fast delete that deletes everything in a table so be cautious.
--Only works on tables not referenced in foreign key constraints
TRUNCATE TABLE sometable;
```

COMMAND LINE EXAMPLES

These are located in bin folder of PostgreSQL

To get more info about each do a --help e.g. psql --help

```
pg_dump -i -h someserver -p 5432 -U someuser -F c -b -v -f "\somepath\somedb.backup" somedb
pg_dumpall -i -h someserver -p 5432 -U someuser -c -o -f "\somepath\alldbs.sql"
pg_restore -i -h someserver -p 5432 -U someuser -d somedb -1 "\somepath\somedb.backup"
psql -h someserver -p 5432 -U someuser -d somedb -f "\somepath\somefiletorun.sql"
psql -h someserver -p 5432 -U someuser -d somedb -c "CREATE TABLE sometable(st_id serial, st_name varchar(25))"
```

output query as xml

```
psql -h someserver -p 5432 -U someuser -d somedb -P "t" -c "SELECT query_to_xml('select * from sometable', false, false, 'sometable')" -o "outputfile.xml";
```

New -- only analyze

```
vacuumdb --analyze-only
```

http://www.postgresonline.com

This work is licensed under a Creative Commons Attribution-Share Alike 3.0 United States License
Feel free to use this material, but we ask that you please retain the Postgres OnLine website link.