Cheatography

PostgreSQL

by cmok1996 via cheatography.com/136331/cs/28464/

PSQL commands

Help	psql --help
List all databases	\l
Quit sessions / psql	\q
Connecting to database	\c <database_name>; / psql -h <host_name> -p <port> -U <username>;
List all schemas	\dn
List all tables in current database	\dt
List all stored procedure and functions	\df
List all views	\dv
Describe table	\d <tbl_name>
Expanded display	\x
Execute script	\i path/<name>.sql
Export query to CSV	\copy (<SELECT_query>) TO 'path/<name>.csv' DELIMITER ',' CSV HEADER;
Import CSV to table	COPY <tbl_name> (col1, col2, ...) FROM 'path/' DELIMITER ',' CSV HEADER;

Database

Create database	CREATE DATABASE <name>;
Drop database	DROP DATABASE <name>;
Create Table	CREATE TABLE <name> (<col_name> <datatype(length)> <constraint>, ...);
Create new table from query	SELECT <select_query> INTO <new_table_name> FROM <existing_table> <remaining_query>; / CREATE TABLE <new_table_name> AS <query>
Create temporary table (only exist within session	CREATE TEMP TABLE <tbl_name>(<col> <datatype> <constraint>, ...);
Update values	UPDATE <tbl_name> SET <col_name> = <>, <col_name> = <>, ...);
Add columns	ALTER TABLE <tbl_name> ADD COLUMN <column_name> <datatype> <constraint>);
Drop column	ALTER TABLE <tbl_name> DROP COLUMN <col_name>;
Rename column	ALTER TABLE <tbl_name> RENAME COLUMN <old_column> TO <new_column>;
Insert rows	INSERT INTO <tbl_name> (col1, col2, ...) VALUES (val1, val2, ...) , (val1, val2, ...) ...);

Database (cont)

Insert conficts	INSERT INTO <tbl_name> (col1, col2, ...) VALUES (val1, val2, ...) ON CONFLICT (<unique_col>) DO UPDATE SET <col1> = EXCLUDED.col1, <col2> = EXCLUDED.col2, ...;
Delete rows	DELETE <tbl_name> WHERE <logic>
Delete entire (large) table	TRUNCATE TABLE <table_name>
Add PK Constraint	ALTER TABLE <tbl_name> ADD PRIMARY KEY ;
Add Check Constraint	ALTER TABLE <tbl_name> ADD <constraint_name> CHECK (<logic>, ...);
Add Unique Constraint	ALTER TABLE <tbl_name> ADD <constraint_name> UNIQUE (<col_name>);
Not Null Constraint	ALTER TABLE <tbl_name> ALTER COLUMN <column_name> SET NOT NULL, <column_name> SET NOT NULL, ...)
Set Default	ALTER TABLE <tbl_name> ALTER COLUMN <column_name> SET DEFAULT <value>;
Add PK (constraint name auto = col_pkey)	ALTER TABLE <tbl_name> ADD PRIMARY KEY (<col>);
Add FK	ALTER TABLE <tbl_name> ADD FOREIGN KEY <constraint_name> (<fk_column>) REFERENCES <ref_tbl> (<pk_column>) ON DELETE CASCADE;

Database (cont)

Drop Constraints	ALTER TABLE <tbl_name> DROP CONSTRAINT <constraint_name>

Basic query

Select columns	SELECT col1, col2, ... FROM <table> WHERE <col = / !=/ > / < / >=/ <= val OR 'text' / col in (val1, val2...)> ORDER BY<col> LIMIT 5;
WHERE + null	SELECT col1, col2 ... FROM <table> WHERE <col IS NULL / col IS NOT NULL>;
Aggregate functions	SUM(<col>) / AVG(<col>) / MAX(<col>) / MIN(<col>) / COUNT(<col>)
GROUP BY + HAVING	SELECT col1, COUNT(col2), ... FROM <table> WHERE ... GROUP BY <col1> ... HAVING COUNT(col2) <logic> ORDER BY COUNT(col2);
Joins	SELECT a.col1, ... , b.col2 FROM <table1> a [INNER, LEFT, RIGHT, FULL OUTER] JOIN <table2> ON a.col1 = b.col1;
CTE	WITH <cte_name> AS <query> SELECT ... FROM <cte_name>...
Subquery	SELECT ... FROM <subquery> <alias> / ... WHERE <col> IN <subquery>
Casting	SELECT CAST(<col> AS <datatype>) / SELECT <col>::<datatype> ;

Basic query (cont)

Concatenate	SELECT CONCAT('text1', 'text2', ...) AS <name>; / SELECT 'text1' || 'text2' ||.. AS <name>;
COALESCE / NULLIF (return null if match arg2)	SELECT COALESCE(<col>, <value>) / SELECT COALESCE(NULLIF(<col>, 0), <value>);
Case When	SELECT CASE WHEN <logic> THEN <value> ELSE <value> WHEN <logic> THEN <value> ELSE <value> [...] END AS <name>;

Dates

Convert to date	SELECT CAST(<col> AS DATE) / SELECT <col>::DATE ;
Filter based on date	SELECT <...> WHERE <col>::DATE > TO_DATE('201701-03','YYYYMMDD');
Extract date features	SELECT EXTRACT(<field> FROM <col>::TIMESTAMP) AS <name>

Window Functions

General syntax	SELECT AGG_FUNC(<col>) OVER (ORDER BY <col> PARTITION BY <col>) AS <name> ...;
Cumulative sum	SELECT SUM(<col1>) OVER (ORDER BY <col2> [PARTITON BY <col3>);
Grouped sum printed in all rows	SELECT SUM(<col1>) OVER (PARTITION BY <col2>);

Window Functions (cont)

Ranking	SELECT [ROW_NUMBER(), DENSE_RANK(), RANK(), CUME_DIST(), NTILE(<n>)] OVER (ORDER BY <col2> [PARTITION BY <col3>]) AS <name>;
Lag / lead	SELECT [LAG(<col>, <n>), LEAD(<col>, <n>)] OVER (ORDER BY <col2> [PARTITION BY <col3>]) AS <name>;
First / Last value (for strings)	SELECT [FIRST_VALUE(<text_col>), LAST_VALUE(<text_col>)] OVER(ORDER BY (<col2>) [PARTITION BY <col3>] AS <name>;

Correlated subqueries

Return outer query if meets any of inner query	SELECT <...> FROM <table> WHERE <col> > ANY (<subquery>);
Return outer query if meet all of inner query	SELECT <...> FROM <table> WHERE <col> > ALL (<subquery>);
Return outer query if subquery returns at least one row	SELECT <col1> FROM <table1> WHERE [NOT] EXISTS(SELECT <...> FROM <table2> WHERE col1 = table1.col1)

Advanced

Explain query stats	EXPLAIN <query>
Create [Partial] Index (improve efficiency for where clause	CREATE [UNIQUE] INDEX <idx_name> <table>(<col>) [WHERE <col> = <value>];
Create sub-blocks	DO $$ <<outer_block>> DELCARE counter integer:=0; BEGIN counter := counter + 1; DECLARE counter interger := 0 BEGIN counter := counter + 10 - outer_block.counter; END; END outer_block $$
Dollar--quoted string constant	$optional_tag$ <to_escape_backslash/quotes/query> $optional_tag$
Create anonymous block	DO $$ <<first_block>> DECLARE film_count integer:=0 BEGIN SELECT COUNT(*) INTO film_count FROM <table> RAISE NOTICE 'The number of films is %', film_count; END first_block;
Variable setting	DO $$ DECLARE <var_name> <datatype>:=<initialized_value>; END $$; / DO $$ DECLARE <var_name> <datatype>; BEGIN <select_into query>; END $$;

Advanced (cont)

Functions (call by SELECT func())	CREATE OR REPLACE FUNCTION <func_name(val1 <datatype>, val2 < datatype> ...)> RETURNS <datatype> LANGUAGE PLPGSQL AS tag DECLARE <var_name> <datatype> BEGIN SELECT val1+val2 INTO <var_name> RETURN <var_name>; END; tag / CREATE OR REPLACE FUNCTION <func_name(IN val1 <datatype, IN val2 <datatype> ..., OUT out_val <datatype>, ...) AS tag BEGIN <func_code>; END; tag LANGUAGE PLPGSQL; / CREATE OR REPLACE <func(val1 = <datatype>, ...)> RETURNS TABLE(var1 <datatype>, var2 <datatype> ...) AS $body$ BEGIN RETURN QUERY <query>; END; LANGUAGE PLPGSQL;
Drop function	DROP FUNCTION <func_name> / --if function is not unique / overloaded -- DROP FUNCTION <func_name(...)>
Undo Transactions	--Start new transaction-- BEGIN; --undo-- ROLLBACK; --Commit transaction-- COMMIT;
Create stored procedure	CREATE PROCEDURE
Print	BEGIN ... RAISE NOTICE '... %... %...', <var1> , <var2>, ...; END;
Set row variable	DO $$ DECLARE actor_info%rowtype; BEGIN SELECT * INTO actor_info FROM table WHERE id=10; END $$;

Advanced (cont)

Set constant	DO $$ DECLARE vat CONSTANT NUMERIC := 0.1; net_price NUMERIC := 20.5; BEGIN RAISE NOTICE 'The selling price is %', net_price * (1 + vat); END $$;
if else	...BEGIN ... IF .. THEN ; ELSIF ... THEN; ELSE ...; END IF; END;
Procedures (work same way as functions but able to rollback, mostly used for DML statements)	CREATE OR REPLACE PROCEDURE <procedure_name(...)> / DROP PROCEDURE <procedure_name>
Create /Drop Views	CREATE OR REPLACE VIEW <name> AS <query>; DROP VIEW <view_name> [CASCADE];
Renaming Views	ALTER VIEW <view_name> RENAME TO <new_name>;
Materialized Views (for concurrently, unique index on view must exist first)	CREATE MATERIALIZED VIEW <view_name> AS <query> WITH [NO] DATA; REFRESH MATERIALIZED VIEW [CONCURRENTLY] <view_name>;