PostgreSQL All Commands: Full Guide with Examples & Use Cases

Basics (Core SQL Syntax)

SELECT: Retrieves data from one or more tables.

`Example:`

```
SELECT name FROM users;
```

Use Case: Use when you need to fetch data from a table for display or processing.

INSERT INTO: Inserts new rows into a table.

`Example:`

```
INSERT INTO users (name, age) VALUES ('Alice', 30);
```

Use Case: Use when adding new records such as user signups.

UPDATE: Modifies existing data in a table.

`Example:`

```
UPDATE users SET age = 31 WHERE name = 'Alice';
```

Use Case: Use when modifying values based on user actions or data corrections.

DELETE: Removes rows from a table.

`Example:`

```
DELETE FROM users WHERE name = 'Alice';
```

Use Case: Use when removing obsolete or user-deleted data.

CREATE TABLE: Creates a new table.

`Example:`

```
CREATE TABLE users (id SERIAL, name TEXT);
```

Use Case: Use during initial schema design or feature expansions.

DROP TABLE: Deletes a table.

`Example:`

```
DROP TABLE users;
```

Use Case: Use when a table is no longer needed and should be removed from the schema.

ALTER TABLE: Modifies table structure.

`Example:`

```
ALTER TABLE users ADD COLUMN email TEXT;
```

Use Case: Use when you need to add or remove columns or constraints.

Filtering and Sorting

WHERE: Filters rows based on condition.

```
Example:

SELECT * FROM users WHERE age > 30;
```

Use Case: Use to retrieve only records that match specific criteria.

ORDER BY: Sorts result set.

```
Example:

SELECT * FROM users ORDER BY name ASC;
```

Use Case: Use to sort output alphabetically, numerically, or chronologically.

LIMIT / OFFSET: Restricts number of rows returned.

```
Example:

SELECT * FROM users LIMIT 5 OFFSET 10;
```

Use Case: Use for pagination in web apps or to restrict large datasets.

DISTINCT: Removes duplicates.

```
Example:

SELECT DISTINCT country FROM users;
```

Use Case: Use to get unique values from a column.

IN / NOT IN: Checks if value is in a list.

```
Example:

WHERE age IN (25, 30, 35)
```

Use Case: Use when matching a field against multiple values.

BETWEEN: Checks for range.

```
Example:

WHERE age BETWEEN 18 AND 30
```

Use Case: Use to filter values within a continuous range.

LIKE / ILIKE: Pattern matching (ILIKE = case-insensitive).

```
Example:

WHERE name ILIKE 'a%'
```

Use Case: Use to find values that start with, end with, or contain specific patterns.

Joins and Relationships

INNER JOIN: Matches records in both tables.

```
Example:

SELECT * FROM a INNER JOIN b ON a.id = b.a_id;
```

Use Case: Use when you want only matched rows from both tables.

LEFT JOIN: All rows from left table + matches from right.

```
Example:

SELECT * FROM a LEFT JOIN b ON a.id = b.a_id;
```

Use Case: Use to include unmatched rows from the left table.

RIGHT JOIN: All rows from right table + matches from left.

```
Example:

SELECT * FROM a RIGHT JOIN b ON a.id = b.a_id;
```

Use Case: Use to include unmatched rows from the right table.

FULL OUTER JOIN: All rows from both tables.

```
Example:

SELECT * FROM a FULL OUTER JOIN b ON a.id = b.a_id;
```

Use Case: Use when you want to preserve all data regardless of match.

CROSS JOIN: Cartesian product.

```
Example:

SELECT * FROM a CROSS JOIN b;
```

Use Case: Use for testing or generating combinations across datasets.

Grouping and Aggregation

GROUP BY: Groups rows for aggregation.

```
Example:

SELECT dept, COUNT(*) FROM employees GROUP BY dept;
```

Use Case: Use to calculate totals or stats by category.

HAVING: Filters groups (after GROUP BY).

```
Example:

SELECT dept FROM employees GROUP BY dept HAVING COUNT(*) > 5;
```

Use Case: Use when filtering aggregated results.

COUNT, SUM, AVG: Aggregate functions.

```
Example:

SELECT AVG(salary) FROM employees;
```

Use Case: Use to summarize numeric data.

Window Functions

RANK(): Assigns rank, with gaps for ties.

`Example:`

```
RANK() OVER (ORDER BY salary DESC)
```

Use Case: Use for leaderboard-style ranking.

DENSE_RANK(): Like RANK() but no gaps for ties.

`Example:`

```
DENSE_RANK() OVER (ORDER BY salary DESC)
```

Use Case: Use for dense ranking in analytical reports.

ROW_NUMBER(): Unique row number in partition.

`Example:`

```
ROW_NUMBER() OVER (PARTITION BY dept ORDER BY salary)
```

Use Case: Use to uniquely identify rows in ordered groups.

NTILE(n): Buckets rows into n quantiles.

`Example:`

```
NTILE(4) OVER (ORDER BY score)
```

Use Case: Use to assign rows to percentiles.

LAG(): Previous row value.

`Example:`

```
LAG(score) OVER (PARTITION BY player ORDER BY date)
```

Use Case: Use to compare current value with a previous one.

LEAD(): Next row value.

`Example:`

```
LEAD(score) OVER (ORDER BY id)
```

Use Case: Use to access future row data.

FIRST_VALUE(): First value in window frame.

`Example:`

```
FIRST_VALUE(salary) OVER (...)
```

Use Case: Use to retrieve the earliest value in a group.

LAST_VALUE(): Last value in window frame.

`Example:`

```
LAST_VALUE(score) OVER (...)
```

Use Case: Use to get the final value from a window.

NTH_VALUE(): Nth value in window.

```
Example:

NTH_VALUE(salary, 2) OVER (...)
```

Use Case: Use to retrieve a specific rank/positioned value.

Pivot and Unpivot

PIVOT with FILTER: Transforms rows into columns using aggregate filters.

```
Example:

SELECT employee,

  SUM(salary) FILTER (WHERE year = 2022) AS salary_2022,

  SUM(salary) FILTER (WHERE year = 2023) AS salary_2023

FROM salaries

GROUP BY employee;
```

Use Case: Use when summarizing multiple values per category into columns.

PIVOT with CASE: Alternative pivot using conditional aggregation.

```
Example:

SELECT employee,

  MAX(CASE WHEN year = 2022 THEN salary END) AS salary_2022,

  MAX(CASE WHEN year = 2023 THEN salary END) AS salary_2023

FROM salaries

GROUP BY employee;
```

Use Case: Use for simple pivoting without FILTER.

UNPIVOT with UNION ALL: Converts columns into rows by using UNION ALL.

```
Example:

SELECT employee, '2022' AS year, salary_2022 AS salary

FROM employee_salaries

UNION ALL

SELECT employee, '2023' AS year, salary_2023 AS salary

FROM employee_salaries;
```

Use Case: Use to flatten columnar data for analytics.

Common Table Expressions (CTE)

WITH: Defines temporary named result sets for reuse in a query.

```
Example:

WITH recent_orders AS (
  SELECT * FROM orders WHERE order_date > NOW() - INTERVAL '30 days'
)
SELECT * FROM recent_orders WHERE total > 100;
```

Use Case: Use when breaking down complex queries or referencing the same subquery multiple times.

Set Operations

UNION: Combines results from two queries and removes duplicates.

```
Example:
SELECT name FROM a UNION SELECT name FROM b;
```

Use Case: Use when merging data sources with distinct values.

UNION ALL: Combines results from two queries and keeps duplicates.

```
Example:
SELECT name FROM a UNION ALL SELECT name FROM b;
```

Use Case: Use when combining datasets without deduplication.

INTERSECT: Returns common rows between two queries.

```
Example:
SELECT name FROM a INTERSECT SELECT name FROM b;
```

Use Case: Use to find overlapping data.

EXCEPT: Returns rows from the first query not in the second.

```
Example:
SELECT name FROM a EXCEPT SELECT name FROM b;
```

Use Case: Use to identify differences between datasets.

DDL and Advanced Schema Features

CREATE INDEX: Creates an index to speed up queries.

```
Example:
CREATE INDEX idx_users_name ON users(name);
```

Use Case: Use to optimize performance for search-heavy columns.

CREATE VIEW: Defines a saved virtual query.

```
Example:
CREATE VIEW active_users AS SELECT * FROM users WHERE active = true;
```

Use Case: Use to simplify frequent query patterns.

CREATE MATERIALIZED VIEW: Stores query results on disk for performance.

```
Example:
```

```
CREATE MATERIALIZED VIEW sales_summary AS SELECT region, SUM(total) FROM sales
GROUP BY region;
```

Use Case: Use for expensive queries that don't need real-time updates.

FOREIGN KEY: Enforces referential integrity.

```
Example:

FOREIGN KEY (user_id) REFERENCES users(id)
```

Use Case: Use to ensure relational consistency.

CHECK: Adds custom validation logic.

```
Example:

CHECK (age >= 18)
```

Use Case: Use to enforce business rules at the database level.

Procedural and Custom Logic

CASE: Adds conditional logic inside queries.

```
Example:

SELECT name, CASE WHEN age < 18 THEN 'Minor' ELSE 'Adult' END FROM users;
```

Use Case: Use to implement logic-based transformations.

COALESCE(): Returns the first non-null value in the list.

```
Example:

SELECT COALESCE(phone, 'N/A') FROM users;
```

Use Case: Use to handle null values in results.

NULLIF(): Returns NULL if two expressions are equal.

```
Example:

SELECT NULLIF(a, b) FROM table;
```

Use Case: Use to prevent divide-by-zero errors.

GREATEST(): Returns the largest value in a list.

```
Example:

SELECT GREATEST(score1, score2, score3) FROM test_scores;
```

Use Case: Use to compare multiple columns.

LEAST(): Returns the smallest value in a list.

```
Example:

SELECT LEAST(score1, score2, score3) FROM test_scores;
```

Use Case: Use to find minimums across multiple values.

Indexing & Performance

EXPLAIN: Displays the query plan for a SQL statement.

```
Example:

EXPLAIN SELECT * FROM users WHERE name = 'Alice';
```

Use Case: Use to understand and optimize query performance.

ANALYZE: Collects statistics about database contents.

```
Example:

ANALYZE users;
```

Use Case: Use to update planner stats after major changes.

VACUUM: Cleans up outdated/deleted rows.

```
Example:

VACUUM FULL users;
```

Use Case: Use to reclaim disk space and improve performance.

Procedural / Advanced Control

DO: Executes an anonymous code block.

```
Example:

DO $$ BEGIN RAISE NOTICE 'Hello'; END $$;
```

Use Case: Use for one-off logic without creating a function.

CREATE FUNCTION: Defines a reusable database function.

```
Example:

CREATE FUNCTION add(a int, b int) RETURNS int AS $$ BEGIN RETURN a + b; END $$
LANGUAGE plpgsql;
```

Use Case: Use to encapsulate reusable business logic.

RAISE NOTICE: Outputs debug messages inside PL/pgSQL blocks.

```
Example:

RAISE NOTICE 'Processing row: %', id;
```

Use Case: Use for debugging or status updates.

EXCEPTION: Handles errors in PL/pgSQL.

```
Example:

BEGIN ... EXCEPTION WHEN OTHERS THEN ... END;
```

Use Case: Use to catch and handle runtime errors in stored procedures.

PostgreSQL-Specific Extras

SERIAL / BIGSERIAL: Auto-incrementing integers for IDs.

```
Example:

id SERIAL PRIMARY KEY
```

Use Case: Use for auto-generated primary keys.

GENERATED AS IDENTITY: SQL-standard syntax for sequences.

```
Example:

id INT GENERATED ALWAYS AS IDENTITY
```

Use Case: Use instead of SERIAL for standard compliance.

ARRAY: Stores and queries multiple values in a single column.

```
Example:

SELECT * FROM users WHERE 'admin' = ANY(roles);
```

Use Case: Use to store lists without normalization.

ENUM: Defines a type with a fixed set of values.

```
Example:

CREATE TYPE mood AS ENUM ('happy', 'sad');
```

Use Case: Use for status fields with limited options.

UUID: Universally unique identifier type.

```
Example:

id UUID DEFAULT gen_random_uuid()
```

Use Case: Use when high uniqueness is needed across systems.

tsvector / tsquery: Used for full-text search indexing and queries.

```
Example:

SELECT to_tsvector('english', content) FROM articles;
```

Use Case: Use to support efficient text search capabilities.