

Lecture 3.2. DML

PostgreSQL Data Types

PostgreSQL supports many built-in data types:

- **Boolean**
- **Character types** – `char`, `varchar`, `text`
- **Numeric types**
 - integers
 - floating-point numbers
- **Temporal types**
 - `date`, `time`, `timestamp`, `timestamptz`, `interval`
- **UUID** – Universally Unique Identifier
- **Array** – arrays of strings, numbers, etc.
- **JSON** – `json`, `jsonb` for JSON data
- **hstore** – key-value pairs
- **Special types** – e.g., network addresses, geometric data

Boolean Type

Boolean (or `bool`) can store three values:

- `TRUE`
- `FALSE`
- `NULL` (unknown / missing)

```
is_active BOOLEAN; -- or: is_active BOOL;
```

True	False
true	false
't'	'f'
'true'	'false'
'y'	'n'
'yes'	'no'
'1'	'0'

Character Types

Character Types	Description
CHARACTER VARYING(n), VARCHAR(n)	variable-length with length limit
CHARACTER(n), CHAR(n)	fixed-length, blank padded
TEXT, VARCHAR	variable unlimited length

When to use:

- `CHAR(n)` – rarely, when all values are the same size (e.g., fixed codes).
 - `VARCHAR(n)` – user names, emails with a clear max length.
 - `TEXT` – long descriptions, comments.
-

Numeric Types

PostgreSQL has two main numeric families:

- **Integers** (no fractional part)
- **Floating-point / exact numeric** (with fractional part)

Integer

Name	Storage Size	Min	Max
SMALLINT	2 bytes	-32,768	+32,767
INTEGER	4 bytes	-2,147,483,648	+2,147,483,647
BIGINT	8 bytes	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807

Used for IDs, counters, quantities when you don't need fractions.

Floating-Point & Exact Numeric

- `FLOAT(n)`

- floating-point number with precision at least n , up to 8 bytes.
- `REAL` (often `FLOAT4`)
 - 4-byte floating-point.
- `NUMERIC` or `NUMERIC(p, s)` (a.k.a. `DECIMAL`)
 - **exact** numeric with:
 - p – total number of digits (precision)
 - s – number of digits after decimal point (scale).
 - Good for **money** and where rounding errors are not acceptable.

Example – `NUMERIC(p, s)`

```
DROP TABLE IF EXISTS products;

CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price NUMERIC(5,2)
);

INSERT INTO products (name, price)
VALUES ('Phone', 500.215),
       ('Tablet', 500.214);
```

Result table:

id	name	price
1	Phone	500.22
2	Tablet	500.21

PostgreSQL **rounds** to 2 decimal places because of `NUMERIC(5,2)`.

Temporal Data Types

- `DATE`
 - stores **date only** (year-month-day).

- **TIME**
 - stores **time of day only** (hours, minutes, seconds).
- **TIMESTAMP**
 - stores **date + time**.
- **TIMESTAMPTZ**
 - **TIMESTAMP WITH TIME ZONE** (time zone aware).
- **INTERVAL**
 - stores a **time period** (duration), e.g. "3 days 2 hours".

Abbreviation	Meaning
Y	Years
M	Months (in date part)
W	Weeks
D	Days
H	Hours
M	Minutes (in time part)
S	Seconds

```
SELECT NOW() + INTERVAL '3 days 2 hours';
```

Data Manipulation Language (DML)

- DML = **Data Manipulation Language**.
- SQL commands used to **manage and manipulate data inside tables**.

Main DML commands:

- **INSERT** – add new rows.
- **UPDATE** – modify existing rows.
- **DELETE** – delete rows.

INSERT – Adding Rows

Basic INSERT

```
INSERT INTO table_name (column1, column2, ...)  
VALUES (value1, value2, ...);
```

- `table_name` – target table.
- Column list is optional but **recommended**:
 - it defines which values go into which columns.
- If a column is omitted:
 - its **DEFAULT** is used, or
 - `NULL` if no default.

Multiple-row INSERT

```
INSERT INTO table_name (column_list) VALUES  
(value_list_1),  
(value_list_2),  
...  
(value_list_n);
```

This inserts multiple rows in a single statement (more efficient).

Example – `links` table

```
DROP TABLE IF EXISTS links;  
  
CREATE TABLE links (  
    id      SERIAL PRIMARY KEY,  
    url    VARCHAR(255) NOT NULL,  
    name   VARCHAR(255) NOT NULL,  
    description  VARCHAR(255),  
    last_update DATE  
);
```

```
INSERT INTO links (url, name)
VALUES ('https://www.postgresqltutorial.com', 'PostgreSQL Tutorial');
```

Observed result: row with `id = 1`, `url` and `name` filled, `description` and `last_update` = `NULL`.

Inserting multiple rows

```
INSERT INTO links (url, name) VALUES
('https://www.google.com', 'Google'),
('https://www.yahoo.com', 'Yahoo'),
('https://www.bing.com', 'Bing');
```

Then:

```
SELECT * FROM links;
```

shows three rows for Google, Yahoo, Bing.

UPDATE – Modifying Rows

Basic syntax

```
UPDATE table_name
SET column1 = value1,
    column2 = value2,
    ...
WHERE condition;
```

- `SET` lists columns to change and their new values.
- `WHERE` selects which rows to update.
 - If `WHERE` is omitted → **all rows** are modified (dangerous).

Example – `courses` table

Create table and insert rows:

```

DROP TABLE IF EXISTS courses;

CREATE TABLE courses (
    course_id    SERIAL PRIMARY KEY,
    course_name  VARCHAR(255) NOT NULL,
    description  VARCHAR(500),
    published_date DATE
);

INSERT INTO courses (course_name, description, published_date)
VALUES
('PostgreSQL for Developers', 'A complete PostgreSQL for Developers', '2020-07-13'),
('PostgreSQL Administration', 'A PostgreSQL Guide for DBA', NULL),
('PostgreSQL High Performance', NULL, NULL),
('PostgreSQL Bootcamp', 'Learn PostgreSQL via Bootcamp', '2013-07-11'),
('Mastering PostgreSQL', 'Mastering PostgreSQL in 21 Days', '2012-06-30');

```

Update one row:

```

UPDATE courses
SET published_date = '2020-08-01'
WHERE course_id = 3;

```

- Only the row with `course_id = 3` is changed.
- After update, that row's `published_date` becomes `'2020-08-01'`.

DELETE – Removing Rows

Basic syntax

```

DELETE FROM table_name
WHERE condition;

```

- Deletes rows where `condition` is TRUE.

- If `WHERE` is omitted → **all rows** are deleted.

Example – `links` table

```
DROP TABLE IF EXISTS links;
```

```
CREATE TABLE links (
    id      SERIAL PRIMARY KEY,
    url    VARCHAR(255) NOT NULL,
    name   VARCHAR(255) NOT NULL,
    description VARCHAR(255),
    last_update DATE DEFAULT NOW()
);
```

-- Insert several rows (PostgreSQL Tutorial, O'Reilly, Google, Yahoo, Bing, etc.)

Delete one row:

```
DELETE FROM links
WHERE id = 8;
```

Only row with `id = 8` is removed.

Delete and return deleted row:

```
DELETE FROM links
WHERE id = 7
RETURNING *;
```

- Deletes row with `id = 7` and **returns it** as a result set.

Delete all rows:

```
DELETE FROM links;
```

- Table structure stays, but it becomes empty.