Materia: Bases de datos

Profesor: Fernando Arreola Franco

Alumno: Rueda De Oliveira Chun Shik

Tarea: Tarea 9

Semestre: 2026-1

Grupo: 1

Introducción

Los tipos de datos son fundamentales en el diseño de bases de datos, ya que definen la naturaleza de la información que se almacenará y las operaciones que podrán realizarse con ella. En bases de datos relacionales, los tipos principales incluyen:

- **Numéricos**: Para valores cuantitativos (enteros, decimales)
- **Carácter**: Para texto y cadenas alfanuméricas
- **Fecha**: Para almacenar fechas y horas

La elección adecuada del tipo de dato impacta directamente en la eficiencia del almacenamiento, la precisión de los cálculos y la integridad de la información, siendo crucial para un diseño de base de datos optimizado.

Tipos de datos postgresql

Los axiomas de Armstrong tienen tres propiedades esenciales que los hacen fundamentales para el diseño de bases de datos:

1. **Numéricos**

Tipo	Almacenamiento	Capacidad	Uso Recomendado
smallint	2 bytes	-32,768 a 32,767	Cuando el espacio es crítico
integer	4 bytes	-2,147,483,648 a 2,147,483,647	Opción estándar para números enteros
bigint	8 bytes	±92,233,372,036, 854,775,807	Para valores muy grandes
numeric	Variable	Hasta 131,072 dígitos	Precisión exacta requerida
real	4 bytes	6 decimales	Cálculos científicos simples
double precision	8 bytes	15 decimales	Cálculos científicos complejos

   a. Ejemplo:

```
-- Ejemplo de creación de columnas con diferentes tipos enteros

CREATE TABLE productos (
```

id smallint PRIMARY KEY, **-- Para IDs pequeños**

cantidad integer, **-- Para cantidades normales**

stock_total bigint **-- Para grandes volúmenes**

);

-- Insertar datos
INSERT INTO productos VALUES

(1, 100, 1000000),

(2, 500, 2500000);

b. Características importantes del tipo Numeric
   1. Precisión y Escala
   - La precisión es el número total de dígitos significativos
   - La escala es el número de dígitos después del punto decimal
   - Ejemplo: En 23.5141, la precisión es 6 y la escala es 4

   2. Almacenamiento Eficiente
   - Dos bytes por cada grupo de cuatro dígitos
   - Sin almacenamiento de ceros innecesarios
   - Overhead mínimo de 3 a 8 bytes

2. Caracter

a.

Tipo	Características	Uso Recomendado
char(n)	Longitud fija, rellena con espacios	Códigos fijos (ej: códigos postales)
varchar(n)	Longitud variable, hasta n caracteres	Textos cortos con longitud conocida
text	Longitud ilimitada	Textos largos o de longitud desconocida

b. Ejemplo
-- Ejemplo de creación de tabla con tipos char

CREATE TABLE codigos_postales (

codigo char(5), **-- Para códigos postales de 5 dígitos**
region char(2), **-- Para códigos de región de 2 caracteres**
pais char **-- Equivale a char(1)**

```
);

-- Insertar datos

INSERT INTO codigos_postales VALUES

('12345', 'NE', 'E');
```

Características Importantes

1. **char(n)** :

- Siempre ocupa n bytes de almacenamiento
- Rellena con espacios en blanco si el texto es más corto
- Los espacios finales se ignoran en comparaciones
- Sin especificar longitud (char) equivale a char(1)

2. **varchar(n)** :

- Almacena solo la longitud necesaria
- Mantiene los espacios finales
- Sin especificar longitud (varchar) acepta cualquier longitud
- Más eficiente que char para textos variables

3. **text** :

- Tipo nativo de PostgreSQL para texto
- Sin límite práctico de longitud
- Más eficiente para textos largos
- Recomendado para descripciones extensas

3. Fecha

Tipo	Formato	Rango	Uso Recomendado
date	YYYY-MM-DD	5874897 AC - 5874897 DC	Fechas sin hora
time	HH:MI:SS	00:00:00 - 24:00:00	Horas sin fecha
timestamp	YYYY-MM-DD HH:MI:SS	5874897 AC - 5874897 DC	Fecha y hora básica
timestamptz	YYYY-MM-DD HH:MI:SS+HHMM	5874897 AC - 5874897 DC	Fecha y hora con zona horaria
interval	[quantity] [unit]	Depende de la unidad	Períodos de tiempo

a. Ejemplo.

```
-- Ejemplo de creación de tabla con tipos básicos

CREATE TABLE eventos (

fecha date,                    -- Solo fecha

hora time,                     -- Solo hora

momento timestamp,             -- Fecha y hora

zona_hora timestamptz,         -- Fecha, hora y zona horaria

duracion interval              -- Período de tiempo

);

-- Insertar datos

INSERT INTO eventos VALUES

('2025-01-15',

'14:30:00',

'2025-01-15 14:30:00',

'2025-01-15 14:30:00+00',

'2 hours');
```

b. Características Importantes
c. **date** :
    a. Almacena solo la fecha
    b. Formato: YYYY-MM-DD
    c. Útil para fechas de nacimiento, aniversarios
    d. No incluye información de hora
d. **time** :
    a. Almacena solo la hora
    b. Formato: HH:MI:SS
    c. Útil para horarios de citas, turnos
    d. Puede incluir o no microsegundos
e. **timestamp** :
    a. Combina fecha y hora

b. Sin zona horaria
c. Útil para registros de eventos locales
d. Puede incluir microsegundos

f. **timestamptz** :
a. Incluye zona horaria
b. Conversión automática entre zonas horarias
c. Recomendado para aplicaciones globales
d. Más preciso para cálculos temporales

4. Fecha

Característica	json	jsonb
Almacenamiento	Texto plano	Formato binario
Normalización	No realiza normalización	Normaliza automáticamente
Operaciones	-> (acceso), ->> (acceso texto)	@>, <@, ?, ?|, ?&
Indexación	No permite indexación	Permite indexación
Rendimiento	Lento en búsquedas	Rápido en búsquedas
Uso de espacio	Variable según el JSON	Consistente, optimizado
Formato de salida	Mantiene formato original	Normalizado en salida

b. **-- Ejemplo de creación de tablas con tipos JSON**

```
CREATE TABLE productos_json (
id SERIAL PRIMARY KEY,
datos json,                    -- Para datos JSON sin normalizar
datos_bin jsonb                -- Para datos JSON optimizados
);
-- Insertar datos
INSERT INTO productos_json (datos, datos_bin) VALUES
(
    '{"nombre": "Laptop", "especificaciones": {"procesador": "Intel i7", "ram": 16}}',
    '{"nombre": "Laptop", "especificaciones": {"procesador": "Intel i7", "ram": 16}}'
```

);

Características Importantes

1. **json** :
    a. Almacenamiento exacto del texto JSON
    b. Mantiene el formato original
    c. No realiza normalización
    d. Útil para datos que deben mantener su formato exacto
2. **jsonb** :
    a. Almacenamiento binario optimizado
    b. Normaliza los datos (elimina espacios extra, estandariza mayúsculas/minúsculas)
    c. Permite indexación
    d. Mejor rendimiento en búsquedas y operaciones

Casos de Uso Recomendados

- **json**:
    - APIs que requieren formato JSON exacto
    - Documentos que deben mantener su estructura original
    - Datos que necesitan preservar su formato exacto
    - Validación estricta de JSON
- **jsonb**:
    - Búsquedas frecuentes en datos JSON
    - Operaciones complejas con datos JSON
    - Aplicaciones que requieren indexación
    - Sistemas que necesitan alta performance

5. **Array**

Característica	ARRAY
Definición	Colección ordenada de valores del mismo tipo
Sintaxis	`tipo[]` o `ARRAY[valor1, valor2, ...]`
Dimensiones	Unidimensional o multidimensional
Indexación	Basada en 1 (no en 0)
Operaciones	`||` (concatenación), <> (comparación), @> (contiene)
Uso típico	Listas de valores relacionados

b. Ejemplo

-- Ejemplo de creación de tablas con arrays

CREATE TABLE empleados (

```
id SERIAL PRIMARY KEY,
nombre VARCHAR(50),
telefonos TEXT[],                    -- Array de números telefónicos
habilidades TEXT[],                  -- Array de habilidades
experiencia INTEGER[]                -- Array de años de experiencia
);
-- Insertar datos
INSERT INTO empleados (nombre, telefonos, habilidades, experiencia) VALUES
('María García',
ARRAY['123-456-7890', '098-765-4321'],
ARRAY['SQL', 'Python', 'JavaScript'],
ARRAY[5, 3, 2]);                     -- Años en cada tecnología
-- Ejemplos de operaciones con arrays
SELECT * FROM empleados
WHERE habilidades @> ARRAY['SQL'];   -- Empleados con SQL
SELECT nombre, habilidades[1] AS primera_habilidad
FROM empleados;                      -- Primera habilidad de cada empleado
SELECT nombre, array_length(habilidades, 1) AS cantidad_habilidades
FROM empleados;                      -- Cantidad de habilidades por empleado
SELECT nombre, habilidades || ARRAY['Bash'] AS nuevas_habilidades
FROM empleados;                      -- Agregar una nueva habilidad
```

6. **UUID**

Característica	Descripción
Formato	xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
Longitud	128 bits (16 bytes)

Probabilidad de colisión	Extremadamente baja (1 en 2^122)
Uso principal	Identificadores únicos globales
Generación	Automática o manual
Indexación	Soporta índices B-tree y hash

b. Ejemplo

```
-- Ejemplo de creación de tablas con UUID

CREATE TABLE usuarios (

id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

nombre VARCHAR(50),

email VARCHAR(100) );

-- Insertar datos

INSERT INTO usuarios (nombre, email) VALUES

('María García', 'maria@example.com');

-- Verificar el UUID generado

SELECT * FROM usuarios;

-- Resultado:

-- id: 123e4567-e89b-12d3-a456-426614174000

-- nombre: María García

-- email: maria@example.com

-- Ejemplo de búsqueda por UUID

SELECT * FROM usuarios

WHERE id = '123e4567-e89b-12d3-a456-426614174000';

-- Ejemplo de actualización

UPDATE usuarios SET email = 'maria.nueva@example.com'

WHERE id = '123e4567-e89b-12d3-a456-426614174000';
```

c. Características Importantes

1. **Generación** :
    a. `gen_random_uuid()`: Genera un UUID aleatorio
    b. `uuid_generate_v4()`: Genera un UUID versión 4
    c. `uuid_generate_v1()`: Genera un UUID basado en tiempo y MAC
2. **Operaciones** :
    a. Comparación directa con =
    b. Búsqueda exacta
    c. Soporte para índices B-tree y hash
    d. Conversión a/from texto
3. **Ventajas** :
    a. Únicos globalmente
    b. Sin necesidad de coordinación central
    c. Independientes de la base de datos
    d. Escalables en sistemas distribuidos

Bibliografía

8.1. Numeric types. (2025, septiembre 25). PostgreSQL Documentation. https://www.postgresql.org/docs/current/datatype-numeric.html

Custer, C. (s/f). PostgreSQL data types: what are they, and when to use each. Cockroachlabs.com. Recuperado el 20 de octubre de 2025, de https://www.cockroachlabs.com/blog/postgres-data-types/

Data types. (2012, enero 1). PostgreSQL Documentation. https://www.postgresql.org/docs/8.1/datatype.html

Engelbert, C. (2023, agosto 23). Best practices for picking PostgreSQL data types. TigerData Blog. https://www.tigerdata.com/blog/best-practices-for-picking-postgresql-data-types

PostgreSQL data types - numeric, text, and more. (s/f). Prisma.Io. Recuperado el 20 de octubre de 2025, de https://www.prisma.io/docs/orm/more/help-and-troubleshooting/dataguide/introduction-to-data-types

Richman, J. (2023, marzo 9). PostgreSQL data types explained with examples. Estuary. https://estuary.dev/blog/postgresql-data-types/