

????? ??? ?????? ???? ?
????????? Postgres ??????
????????????????? ??? ??????????
????????? json, ?? ??? ??????
????? ? ?????????? ???, ??????????
XML ??? ??????????????

Для такой задачи **не подходит** чистый тип `JSON` или `JSONB`, так как PostgreSQL начнет проверять валидность JSON-структуры и выдаст ошибку при попытке сохранить XML или бинарные данные (изображение).

Наилучшим решением будет использование одного из следующих подходов в зависимости от ваших приоритетов по размеру и производительности.

??????? 1: `BYTEA` (????????? ??????) — ?????? ?????????????????? ?
????????????????????

Тип `BYTEA` (byte array) хранит данные «как есть» в виде сырого массива байт. Ему абсолютно неважно, что именно там находится — JSON, XML, PNG/JPEG или текстовый файл.

SQL

```
CREATE TABLE payload_store (  
    id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
    content_type VARCHAR(50) NOT NULL, -- 'application/json', 'application/xml', 'image/png'  
    payload BYTEA NOT NULL,  
    created_at TIMESTAMPTZ DEFAULT NOW()  
);
```

- **Плюсы:**

- Абсолютная универсальность (принимает всё: от текстовых форматов до бинарных изображений).
- Минимальные накладные расходы при записи/чтении (нет перекодировок в текст).
- Корректно сохраняет изображения без раздувания размера.

• Минусы:

- Из C# нужно передавать данные как `byte[]` (для JSON/XML это будет байтовый массив строки UTF-8).
- Внутри PostgreSQL нельзя делать прямое обращение к полям JSON (например, `payload->>'name'`).

?????? 2: `TEXT` — ???????? ??? ????????, ?? ?? ???????? ???
 ????????? ???????

Если вы хотите видеть содержимое пакетов (JSON и XML) напрямую через консоль или клиенты (DBever, pgAdmin) без декодирования байтов, можно использовать `TEXT`.

SQL
 ■ ■

```
CREATE TABLE payload_store (
  id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  content_type VARCHAR(50) NOT NULL,
  payload TEXT NOT NULL
);
```

• Плюсы:

- Текстовые данные (JSON, XML) хранятся в читаемом виде.
- Текст можно при необходимости привести к типу `JSONB` налету: `SELECT (payload::jsonb)->>'key' WHERE content_type = 'application/json'`.

• Минусы:

- **Изображения придется кодировать в Base64** перед сохранением из C#.
- Кодирование в Base64 увеличивает объем хранимого файла (изображения) примерно на **33%**.

????????? ??????????

Критерий	BYTEA	TEXT (с Base64 для фото)
Универсальность	100% (любые бинарники и текст)	100% (но изображения нужно кодировать)
Размер изображений	Исходный (1:1)	+33% к размеру файла из-за Base64
Удобство просмотра JSON/XML в СУБД	Требуется <code>encode(payload, 'escape')</code>	Читается из «коробки»
Сложность в С# (Npgsql)	Работа через <code>byte[]</code>	Работа через <code>string</code>

?????????????

1. Если **изображений много или они большие** — однозначно используйте `BYTEA`.
2. Если хранение изображений — редкий случай, а **JSON/XML нужно часто просматривать напрямую в БД** — используйте `TEXT` (сохраняя картинки как Base64-строки).
3. **Обязательно добавьте колонку** `content_type` (MIME-тип) или `payload_kind` — без нее приложение на С# не сможет понять, как правильно интерпретировать вычитанные данные (как раскодировать JSON/XML или как отдать картинку).

??? ????????? ? `BYTEA` ?? С# (Npgsql):

С#
■ ■

```
// Для JSON/XML кодируем строку в UTF-8 байты:
byte[] jsonBytes = System.Text.Encoding.UTF8.GetBytes(jsonString);

// Для изображений берем байты файла напрямую:
byte[] imageBytes = await System.IO.File.ReadAllBytesAsync("photo.png");

// Передача в NpgsqlParameter:
cmd.Parameters.Add(new NpgsqlParameter("payload", NpgsqlTypes.NpgsqlDbType.Bytea) { Value =
imageBytes });
```