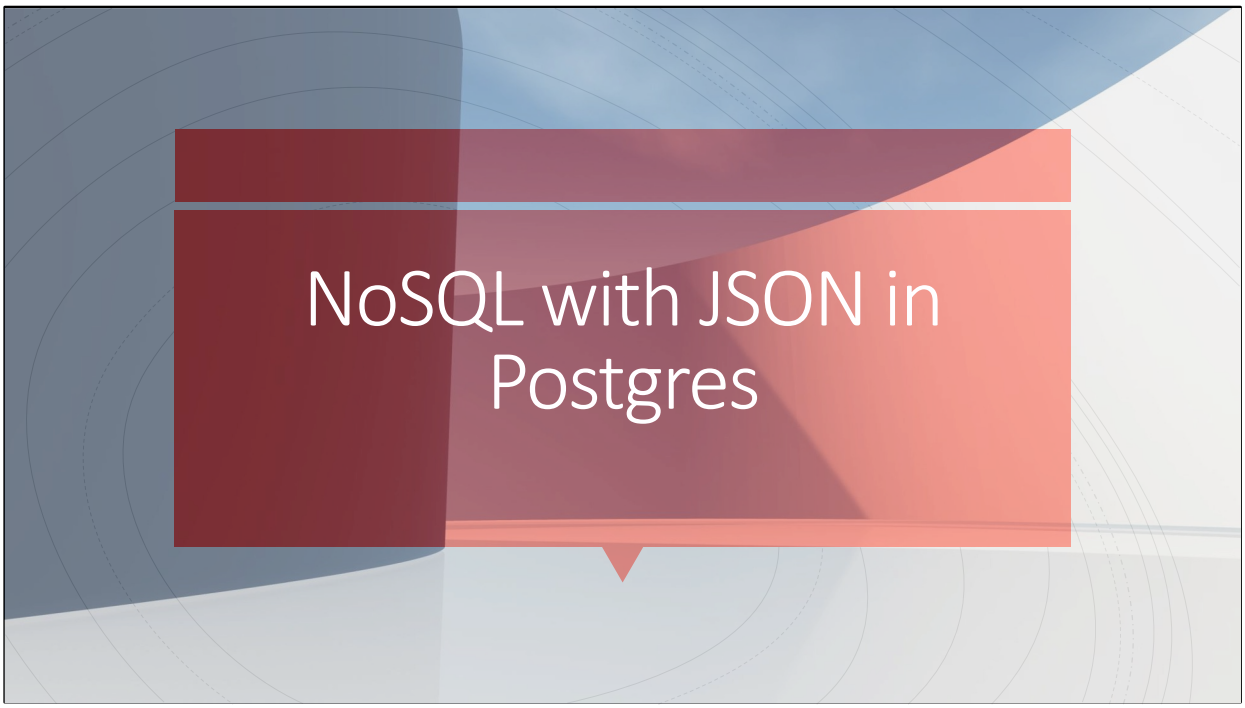
NoSQL with JSON in Postgres

1. Overview of AI Workflows	2. Why Postgres as a vector store	3. Storing and managing vectors	4. Querying the vector store
• Look at high-level architecture - LLMs, vector stores and JSON • Look at key vocabulary and concepts (embeddings, vectors, hybrid queries, etc.)	• What is a vector store? Key concepts and use cases. • Why Postgres and how does it compare with other market tools • Setting up Postgres with vector capabilities (pgvector) • **Lab**: Install and configure Postgres using Docker	• Generating embeddings: Overview of tools and workflows • Storing and organizing embeddings in Postgres • Strategies for handling large datasets including chunking • Dense and sparse vectors • **Lab:** Generate embeddings for a dataset and store them	• Techniques for similarity search: k-NN, cosine similarity • Using indexes to optimize vector queries • Reranking results • **Lab:** Query stored vectors to retrieve similar items (document/image search)
5. Querying LLMs with retrieved data	**6. NoSQL with JSON in Postgres**	**7. Integrating Vector, Relational and JSON Data**	**8. Putting it all together**
• Recap on querying LLMs vis APIs • Best practices for combining vector retrieval with LLM prompts • Prompt configuration parameters (temperature, top-k, etc) • **Lab:** Build a pipeline where vector store results enhance LLM responses (context-aware Q&A, etc)	• Overview of JSON/JSONB support in Postgres • Querying JSONB data with SQL • Indexing JSONB data for performance • **Lab:** Design a schema mixing vector, relational and JSONB data for a sample project	• Building hybrid queries to power advanced workflows • **Case study:** Combining embeddings, metadata (relational) and configurations (JSON) • **Lab:** Implement a hybrid query to support a sample AI use case	• Full stack pipeline demo: Retrieve data, query the LLM and return results • Debugging and optimising the workflow • Spotlight on LLM frameworks • **Lab:** Build a working application combining all elements

Plan for the session

- **Understand JSON and JSONB Support in PostgreSQL**
- **Query JSONB Data Using SQL**
- **Optimize Performance by Indexing JSONB Columns**
- **Lab: Design a Schema Mixing Vector, Relational, and JSONB Data**

Structured + Flexible: The Power of JSON in Postgres

- **PostgreSQL is a relational database**, but it supports **NoSQL-style JSON storage**.
- Storing JSON allows **flexible, schema-less data handling** inside a structured database.
- **Use Cases:**
  - Storing **semi-structured data** (API responses, logs, metadata).
  - **Handling dynamic attributes** without modifying the schema.
  - **Combining relational and document-style data** in a single system.

📌 **Key Takeaway:**
PostgreSQL bridges the gap between relational and NoSQL databases.

Comparing JSON and JSONB in Postgres

Feature	JSON (Text-based)	JSONB (Binary, Optimized)
Storage Format	Stored as raw **text**	Stored as **binary**
Read Performance	**Slower** (parses every time)	**Faster** (pre-parsed & indexed)
Write Performance	**Faster** (raw insert)	**Slower** (parses & indexes)
Indexing Support	❌ No native indexing	✅ Supports **GIN indexes**
Operators Supported	✅ ->, ->>	✅ ->, ->>, @>, ?, jsonb_path_query()
Best For	Storing JSON **as-is**	Querying & searching JSON efficiently

📌 **Key Takeaway:**
Use **JSONB when querying and indexing**. Use **JSON if only storing text data.**

Choosing JSONB for Semi-Structured Data

✅ **Use JSONB When:**

- You need **fast querying and indexing** of JSON fields.
- Your data is **semi-structured**, but needs to be searchable.
- You want to **store API responses, logs, metadata, or user preferences**.
- Schema evolution is **frequent**, but you still need relational features.

❌ **Avoid JSONB When:**

- Data should be **strictly relational** (use **tables & foreign keys**).
- Queries mostly filter on **relational fields** rather than JSON content.
- JSON data is **write-heavy** but rarely read (use **JSON instead**).

💡 **Example Schema Choice:**

- **User table** stores structured info (id, email, created_at).
- **Preferences column (JSONB)** holds flexible settings (theme, notifications, language).

Extracting & Filtering JSONB Data in PostgreSQL

📌 **Key Points:**

- JSONB allows **storing structured data** inside a relational database.
- We need efficient **querying techniques** to extract and filter JSON data.
- PostgreSQL provides **specialized operators and functions** to interact with JSONB.

💡 **Example Use Case:**

- A **product catalog** stores structured attributes (name, price) and flexible data (specifications, reviews) in JSONB.
- Queries must **filter products by price range, extract metadata, and search text within JSONB fields**.

Essential JSONB Operators for Querying Data

Operator	Description	Example Query
->	Extracts a JSON **object or array** as JSONB.	SELECT item_data->'category' FROM items;
->>	Extracts a **text value** from JSONB.	SELECT item_data->>'category' FROM items;
@>	Checks if JSONB contains a **specific key-value pair**.	WHERE item_data @> '{"category": "programming"}'
?	Checks if a **key exists** in JSONB.	WHERE item_data ? 'price'
jsonb_set()	Updates a specific key in JSONB.	UPDATE items SET item_data = jsonb_set(item_data, '{price}', '29.99');

Demo

Lab

Design a Schema Mixing Vector, Relational, and JSONB Data *(Hands-on Lab)*
•Integrate **vector embeddings (pgvector)**, structured **relational data**, and **semi-structured JSONB** in a unified model.
•Implement **queries that combine relational, vector, and JSON search** efficiently.


Questions
?