POSTGRESQL CONFERENCE · SAN JOSE

Building Real-Time, Data-Aware Intelligence with Postgres & Model Context Protocol

Eliminating Hallucinations with Postgres and the Model Context Protocol

Yogesh Jain
Staff SDE

LLM MCP POSTGRES

ABOUT ME

Yogesh Jain

Staff SDE @ EDB · Curious One · Full Stack Developer

- Building a **Postgres Hybrid Manager** for managing Postgres across different deployment platforms
- Focused on **Unified Observability** and **AI Ecosystem integration** at EDB
- Enabling a **Single Pane of Glass**: metrics, logs & AI-driven insights across all platforms

Postgres

Monitoring & Observability

Kubernetes

Cloud-Native Infrastructure

AI Integration

LLMs & MCP for Databases

THE ORIGIN STORY

Where It All Started

I wanted to query APIs and get **real-time data** in a chatbot using natural language.
Before MCP or agents existed, I built my own solution.

1. Pass API spec to LLM

Fed the OpenAPI/Swagger spec as context and asked the LLM to generate a `curl` command.

2. Execute the curl

Wrote a function to parse and execute the generated `curl`, then fed the response back to the LLM.

3. It worked!

Natural language → API call → human-readable answer. The loop closed.

```
// My DIY "agent" - early 2024

User:
"What's the current CPU usage and total DB connections?"
↓
LLM + API Spec →
curl -X GET https://monitor.curiousone.in/
/api/v1/metrics?names=cpu,db_conn
-H Authorization: Bearer ..."
↓
execute() → { cpu: 72%, conn: 84 }
↓
LLM:
"CPU is at 72% and there are 84 active DB connections."
```

THE SCALING WALL

But It Didn't Scale

Every new data source needed custom glue code. I wasn't the only one facing this.

Custom Tooling (2024)

- New parser for every API format
- Custom auth handling per source
- Fragile prompt engineering for each spec
- No shared standard across teams
- Every engineer reinventing the wheel

MCP (Today)

- **One protocol** for all data sources
- Standardized tool discovery & invocation
- Auth, safety & validation built in
- Growing ecosystem of MCP servers
- Adopted across the AI industry

Anthropic introduced MCP as an open standard - and it solved exactly this problem for everyone.

AGENDA

What We'll Cover

01 **The Problem**: Why LLMs hallucinate SQL

02 **MCP**: What it is & how it works

03 **Postgres + MCP**: pg_catalog, tools & flow

04 **EXPLAIN**: Optimizing the queries

05 **Security**: Read-only roles, RLS & audit

06 **Hybrid**: Custom MCP + Postgres MCP

THE PROBLEM

LLMs Are Blind to Your Data

LLMs understand SQL syntax perfectly. But they don't know *your* table names, column names, or relationships. They guess based on training data patterns.

What the LLM generates

```
SELECT u.name, o.total
FROM   users u
JOIN   orders o
       ON u.id = o.user_id
```

ERROR: relation "users" does not exist

What actually exists

```
SELECT c.full_name, o.total_price
FROM   app_customers c
JOIN   customer_orders o
       ON c.id = o.customer_id
```

SUCCESS: 47 rows returned

ROOT CAUSE

The Context Gap

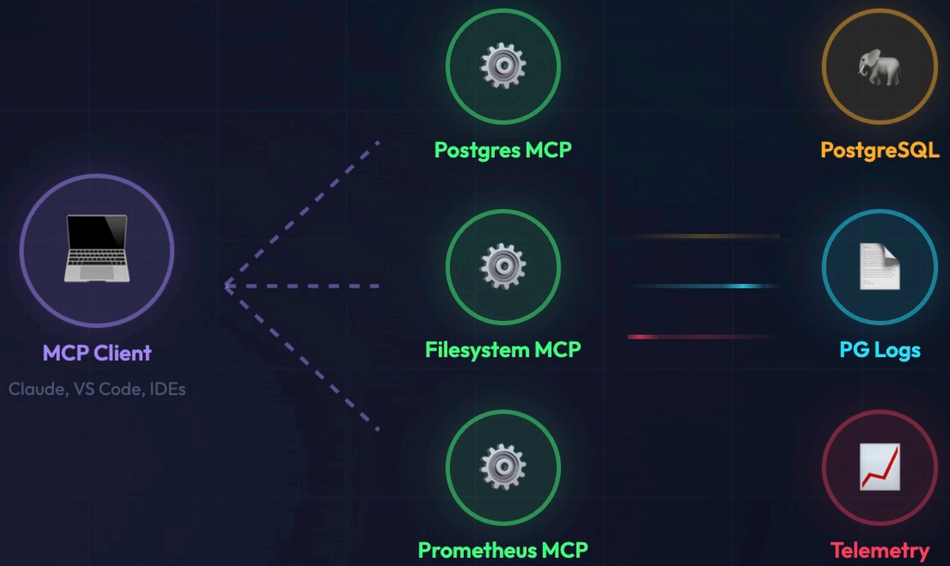
Every type of SQL hallucination traces back to missing context:

Hallucination	Example	Missing Context
Wrong table	`FROM users`	Real table: `app_customers`
Wrong column	`SELECT email FROM orders`	Column doesn't exist
Wrong JOIN	`ON users.id = orders.id`	Should be `orders.user_id`
Wrong types	`WHERE created_at = '2024'`	It's a `timestamptz`
Wrong dialect	`DATEADD(day, 7, now())`	That's SQL Server, not PG

THE SOLUTION

The Model Context Protocol

MCP is an open standard that connects AI models to external data sources and tools. Think of it as **USB-C for AI**.



JSON-RPC 2.0 • stdio / SSE transport • One client → many servers

MCP PRIMITIVES

MCP Building Blocks

MCP defines three primitives. For databases, **Tools** matter most - they return *live* data, not stale snapshots.

🔧 Tools

Functions the LLM can call. Returns live, real-time results.

`EXECUTE_SQL` `LIST_SCHEMAS`

📄 Resources

Read-only, static data via URIs. Schema snapshots go stale fast.

💬 Prompts

Reusable templates that bundle context and instructions.

PG-AIRMAN-MCP

Tools for Data-Aware AI

list_schemas

Returns all database schemas, categorized as system or user. The starting point for discovery.

list_objects

Retrieves tables, views, sequences & functions within a schema, including comments.

get_object_details

Comprehensive details: columns, types, constraints, indexes & comments for any object.

explain_query

Runs EXPLAIN plans and simulates hypothetical indexes via **HypoPG** without creating them.

execute_sql

Executes queries with configurable access control, read-only mode & safe SQL parsing.

analyze_query_indexes

Explores thousands of possible indexes to find optimal solutions for your workload.

THE QUERY FLOW

From Question to Correct SQL

User asks: "Show me top 5 customers by revenue"

1. LLM calls `list_schemas` → discovers available schemas
2. LLM calls `list_objects` → gets real table & view names
3. LLM calls `get_object_details` → columns, types, constraints & indexes
4. LLM calls `explain_query` → validates query plan, used to optimize the query
5. LLM calls `execute_sql` → runs query with access control & safety parsing

THE SECRET WEAPON

pg_catalog - Postgres Knows Itself

The MCP server doesn't need external config. It queries Postgres's own metadata catalog for real-time, always-accurate schema info.

pg_class

Complete map of all tables

pg_attribute

Every column name & exact type

pg_constraint

PKs, FKs, CHECK constraints

pg_index

All indexes for query optimization

pg_namespace

Namespaces (schemas) in the database

pg_description

COMMENT ON annotations as docs

UNDER THE HOOD

What the MCP Server Queries

```sql
-- list_schemas
SELECT
  schema_name,
  schema_owner,
  CASE
    WHEN schema_name LIKE 'pg_%'
      THEN 'System Schema'
    WHEN schema_name =
      'information_schema'
      THEN 'System Info Schema'
    ELSE 'User Schema'
  END AS schema_type
FROM information_schema.schemata
ORDER BY schema_type,
  schema_name;
```

```sql
-- list_objects
SELECT
  CASE c.relkind
    WHEN 'r' THEN 'table'
    WHEN 'v' THEN 'view'
  END AS object_type,
  n.nspname AS table_schema,
  c.relname AS table_name,
  d.description AS comment
FROM pg_catalog.pg_class c
JOIN pg_catalog.pg_namespace n
  ON n.oid = c.relnamespace
LEFT JOIN pg_catalog.pg_description d
  ON d.objoid = c.oid
  AND d.objsubid = 0
WHERE n.nspname = $1
  AND c.relkind IN ('r','v')
ORDER BY c.relname;
```

```sql
-- get_object_details
SELECT
  column_name, data_type,
  is_nullable, column_default
FROM information_schema.columns
WHERE table_schema = $1
  AND table_name = $2
ORDER BY ordinal_position;
```

QUERY-OPTIMIZATION

EXPLAIN: Before Optimization

```
-- LLM's initial query
SELECT c.full_name,
       SUM(o.total_price)
FROM   app_customers c
JOIN   customer_orders o
       ON c.id = o.customer_id
GROUP BY c.full_name
ORDER BY total DESC;
```

Seq Scan on ~500K rows

No date filter - scans ALL orders

~2,999 ms execution time

The LLM reads EXPLAIN output and rewrites...

QUERY-OPTIMIZATION

EXPLAIN: After Optimization

```sql
-- LLM's optimized query
SELECT c.full_name,
       SUM(o.total_price)
FROM   app_customers c
JOIN   customer_orders o
       ON c.id = o.customer_id
WHERE  o.created_at ⩾ NOW() - INTERVAL '90 days'
GROUP BY c.id, c.full_name
ORDER BY total DESC
LIMIT 10;
```

Index Scan on ~3,000 rows

Date filter + LIMIT added

20 ms execution time

~150x faster

SECURITY

Guard Rails

Read-Only Database Role

GRANT SELECT only. Even DROP TABLE gets rejected by Postgres.

Row-Level Security (RLS)

Multi-tenant isolation at the engine level. No SQL injection bypass.

Safe SQL Execution (`execute_sql`)

Configurable access control, read-only mode, safe SQL parsing, and statement_timeout.

Audit Logging + Rate Limits

Every query logged with context. Rate limiting and statement_timeout enforced.

SECURITY IN PRACTICE

Read-Only Role Setup

```
-- Create a dedicated read-only role for MCP
CREATE ROLE mcp_reader LOGIN PASSWORD '...';
GRANT CONNECT ON DATABASE mydb TO mcp_reader;
GRANT USAGE ON SCHEMA public TO mcp_reader;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO mcp_reader;

-- Multi-tenant isolation with RLS
ALTER TABLE customer_orders ENABLE ROW LEVEL SECURITY;

CREATE POLICY tenant_isolation ON customer_orders
  FOR SELECT USING (
    tenant_id = current_setting('app.tenant_id')::int
  );
```

ARCHITECTURE

How It All Fits Together

LET'S TRY IT LIVE

Demo Time

```
# Configure Claude Desktop (or any MCP client)
# ~/Library/Application Support/Claude/claude_desktop_config.json

{
  "mcpServers": {
    "postgres": {
      "command": "docker",
      "args": [
        "run", "-i", "--rm",
        "-e", "AIRMAN_MCP_DATABASE_URL",
        "enterprisedb/pg-airman-mcp",
        "--access-mode=unrestricted"
      ],
      "env": {
        "AIRMAN_MCP_DATABASE_URL":
          "postgresql://mcp_reader:demo@localhost:5432/demo"
      }
    }
  }
}
```

THE REAL PROBLEM

It Runs. It Returns. It's Wrong.

Ask an LLM "what's our revenue this month?" against a real production schema. The SQL executes fine. The number looks plausible. It's **silently wrong**.

What the LLM doesn't know

```sql
-- LLM's guess
SELECT SUM(o.total_price)
FROM   customer_orders o
WHERE  o.created_at ⩾ '2026-04-01'
```

✗ Includes cancelled & refunded orders

✗ Misses soft-delete filter `is_deleted = false`

What the query should be

```sql
-- Correct query
SELECT SUM(o.total_price)
FROM   customer_orders o
WHERE  o.created_at ⩾ '2026-04-01'
  AND o.status ≠ 'cancelled'
  AND o.is_deleted = false
  AND o.is_refunded = false
```

Tribal knowledge that lives in people's heads, not in the schema

SOLUTION

Custom MCP Server with FastMCP

Encode your **business logic** into an MCP tool. The LLM calls it instead of guessing the SQL.

```python
# revenue_mcp.py
from fastmcp import FastMCP
import psycopg2

mcp = FastMCP("revenue-server")

@mcp.tool()
def get_monthly_revenue(month: str, year: int) → dict:
    """Get actual revenue excluding cancelled,
    refunded, and soft-deleted orders."""
    conn = psycopg2.connect(DB_URL)
    cur = conn.cursor()
    cur.execute("""
        SELECT SUM(o.total_price)
        FROM   customer_orders o
        WHERE  DATE_TRUNC('month', o.created_at)
               = DATE_TRUNC('month', %s::date)
        AND    o.status ≠ 'cancelled'
        AND    o.is_deleted = false
        AND    o.is_refunded = false
    """, [f"{year}-{month}-01"])
    result = cur.fetchone()
```

HYBRID APPROACH

Postgres MCP + Custom MCP

Postgres MCP

Schema, tables, columns, FKs

"What exists in the database"

+

Custom MCP

Business logic, filters, compliance

"What the data actually means"

- ✓ Legacy schemas with cryptic names → **Postgres MCP** discovers them live
- ✓ Soft-deletes, status flags, tribal joins → **Custom MCP** encodes the rules
- ✓ LLM picks the right tool for the question → no guessing/ hallucination

CONTEXT STRATEGIES

Why Not Just...

Approach	Freshness	Scalability	Verdict
Paste schema in prompt	Stale instantly	Wastes tokens	Quick hack only
RAG over docs	Embeddings drift from schema	Good	Good for semantics, not structure
Fine-tuned model	Stale on deploy	Expensive	Overkill for schema
MCP (live connection)	Always current	Composable	**Best for structure**

KEY TAKEAWAYS

From Blind Guessing to Data-Aware Intelligence

- ✓ LLMs hallucinate SQL because they **lack live database context**
- ✓ **MCP** is a universal protocol connecting LLMs to external data
- ✓ **pg-airman-mcp** uses **pg_catalog** for real-time schema discovery
- ✓ **Custom MCP servers** (FastMCP) encode business logic, soft-deletes, and tribal knowledge
- ✓ **Postgres MCP** + **Custom MCP** = the hybrid approach for **real enterprise schemas**

POSTGRES MCP · CUSTOM MCP · FASTMCP · POSTGRES

REFERENCES

Resources & Links

MCP — Model Context Protocol Specification - `modelcontextprotocol.io`

AIRMAN — pg-airman-mcp - `github.com/EnterpriseDB/pg-airman-mcp`

PGVECTOR — Vector Similarity Search for Postgres - `github.com/pgvector/pgvector`

PG DOCS — PostgreSQL System Catalogs - `postgresql.org/docs/current/catalogs.html`

THANK YOU

Yogesh Jain

Staff SDE @ EDB

Let's connect to talk about:

OBSERVABILITY · POSTGRES · KUBERNETES · CLOUDNATIVEPG · OPEN SOURCE · MCP

SCAN TO CONNECT

EMAIL
contactyogeshjain@gmail.com

LINKEDIN
yogeshjain96

BLOG
curiousone.in