Domain-Driven Design in Practice: Crafting an AI Assistant Step by Step

Imagine you're part of the data team at a large company. Your team, consisting of data scientists, data engineers, and analytics professionals, has been approached with a challenge. The business analysts in your company frequently need to query the company's vast database to gain insights, but not all of them are proficient in SQL.

They come to you with a request (and a use case emerges):

"***Can we create a self service tool that will allow us to make ad hoc queries without the users knowing SQL?***"

This scenario presents an exciting challenge that combines natural language processing, database management, and AI. It's a common problem in data-driven organizations: how to democratize data access without requiring everyone to become a SQL expert.

How do we even start? Should we just start solutionizing right away?

No! While it might be tempting to dive into coding, especially if we've tackled similar challenges before, that approach often leads to solutions that don't fully address the complexities of the problem or align with broader business goals.

Instead, let's take a step back and consider a more structured approach. We need a method that will help us:

- Fully understand the business context and user needs
- Break down the problem into manageable components
- Identify the core areas where we can provide unique value
- Design a solution that's flexible and can evolve with changing requirements

Enter Domain-Driven Design (DDD) and the DDD Starter Modelling Process.

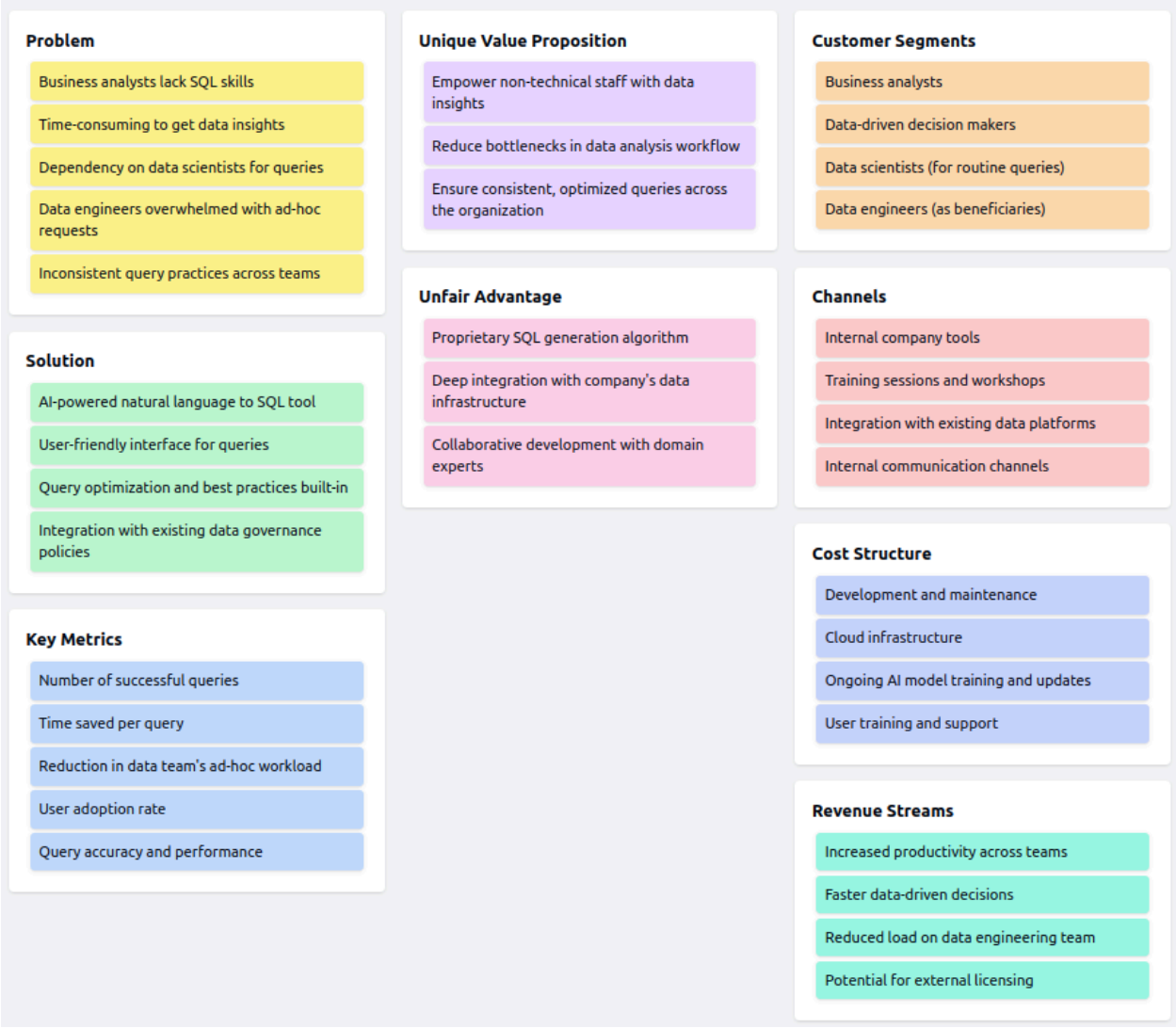
The DDD Starter Modelling Process

The DDD Starter Modelling Process is a step-by-step guide for learning and practically applying each aspect of Domain-Driven Design. It's particularly useful for those new to DDD or when kicking off a new project. I like to make it my own with some variations but I use this as a good starting off point.

Let's walk through each step of this process, applying it to our assistant:

1. Understand

In this step, we align our focus with the organization's business model and user needs. For our application, we might use tools like the classic lean canvas to understand how this tool fits into the company's overall strategy. There are a ton of different variations available as templates.



Key Questions:

- How does this AI assistant contribute to our company's value proposition?
- What are some of the problems we are trying to solve with this tool?
- What key resources and activities are required to build and maintain this tool?

- Who are the primary users, and what are their needs?

2. Discover

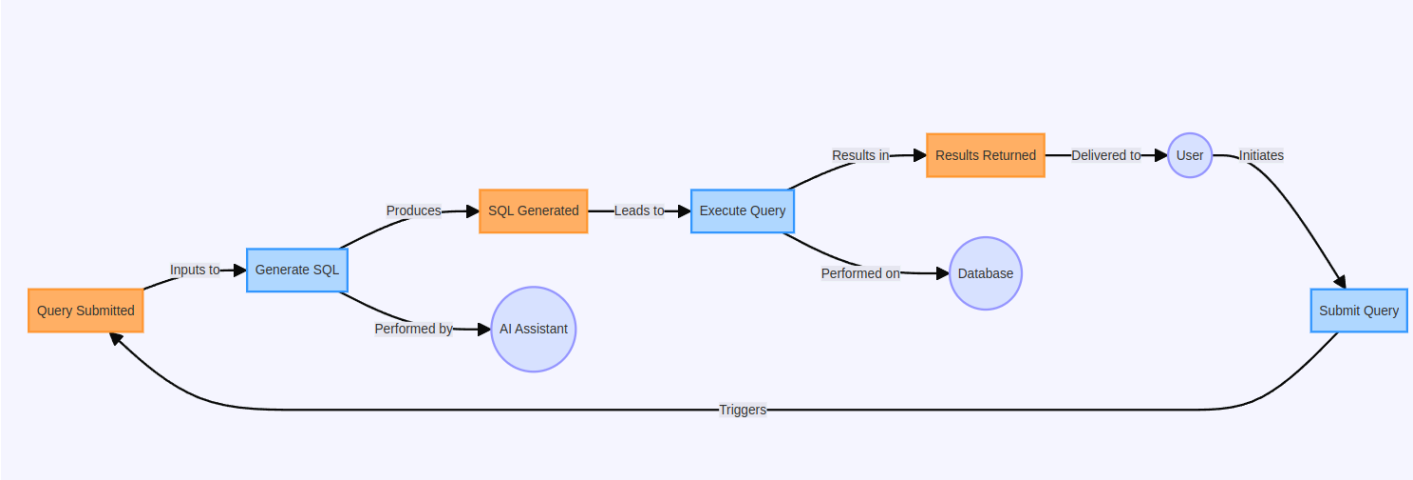
This crucial step involves visually and collaboratively discovering the domain. I personally find this part the most interesting. For our project, we might use classic EventStorming to map out the process of turning a natural language query into a SQL statement.

Don't get too caught up in the process, just have fun with it, lean in and learn with your team!

Key Activities:

- Conduct an EventStorming session with business/data analysts, data scientists, and data engineers
- Identify key domain events like "QueryReceived", "SQLGenerated", "ResultsReturned"
- Map out the flow of information and decision points in the query process

Here's what that might look like (oversimplified but you get the idea):

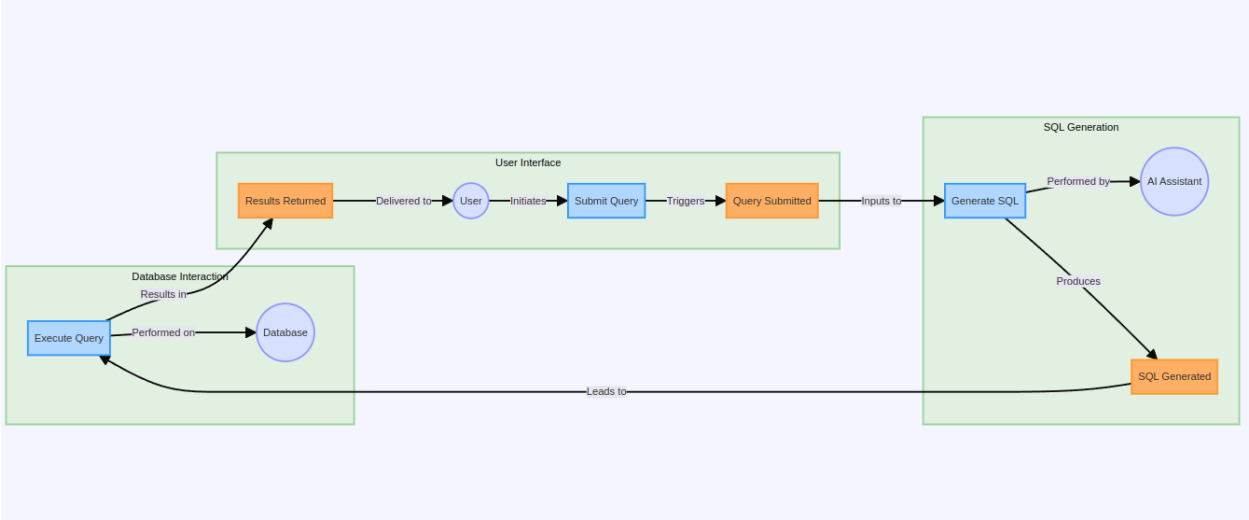


3. Decompose

Here, we break down our main domain **Self-Service Business Intelligence** into subdomains.

For our AI assistant, we identify the following subdomains:

- **SQL Generation**: Transforms natural language queries into accurate SQL statements.
- **Database Interaction:** Manages connections with databases and executes SQL queries.
- **User Interface**: Provides an intuitive interface for users to input queries and view results.

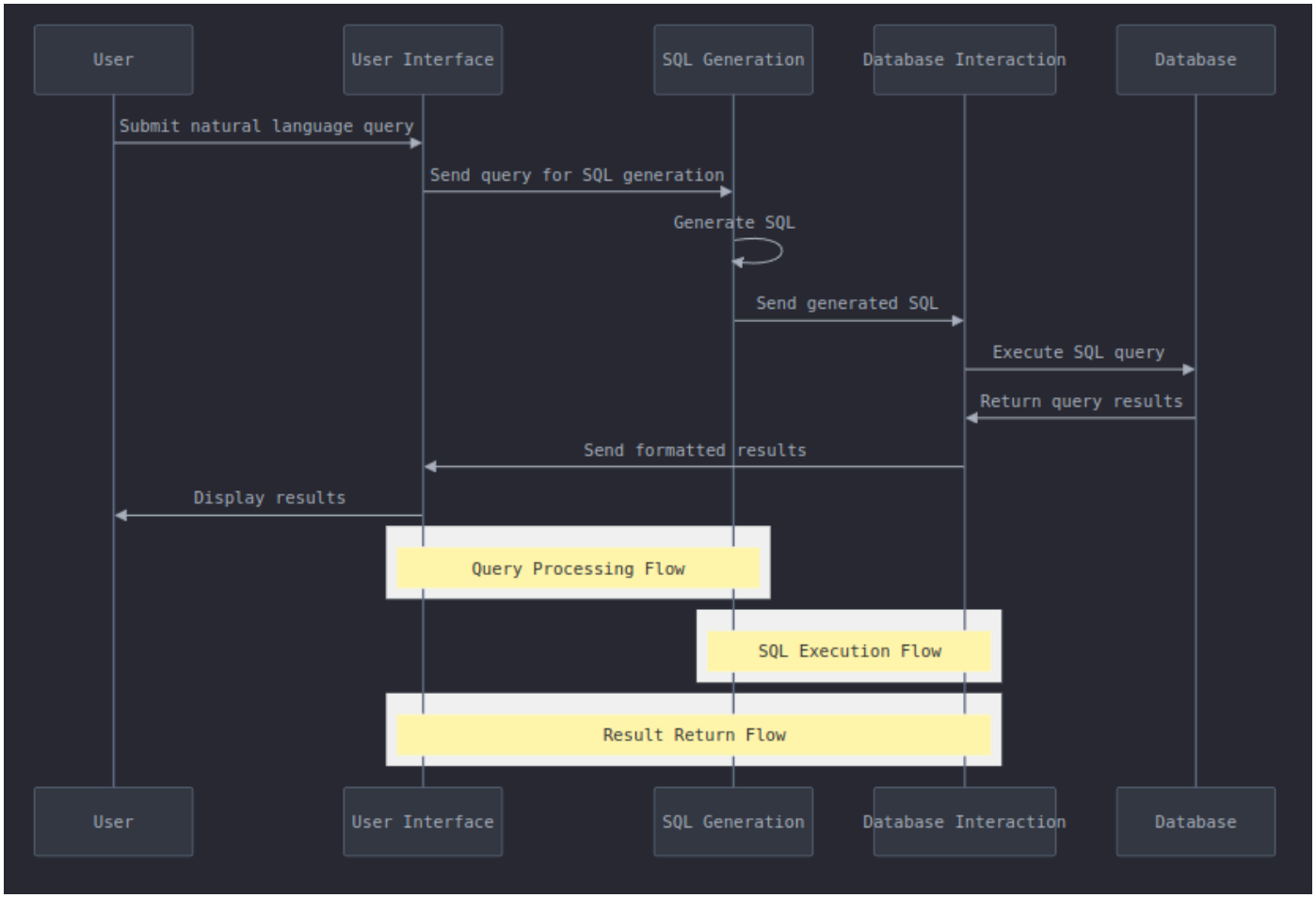


4. Strategize

In this step, we identify our core domains - the areas with the greatest potential for business differentiation. For our project, the SQL Generation sub-domain might be identified as a core domain, as it's where our competitive advantage lies. We can replace the UI and the Database with any number of other alternatives but the core engine can remain the same.

5. Connect

Now we design how our sub-domains interact to fulfill end-to-end business use cases. We might use **Domain Message Flow Modelling** or something similar and my favorite is a good old fashioned **sequence diagram** to visualize how a user query flows through our system, from the UI to to the SQL generator, to the database, and back.



6. Organize

This step involves organizing autonomous teams aligned with context boundaries. We might create separate teams for:

- SQL Generation (our core domain)
- User Interface and Experience
- Database Management and Optimization

As you can see here this is really going to help us with the actual execution of the project. If you can't or don't have the resources to organize entirely separate teams I've also had great success with smaller cross functional teams. I ensure that the team members have the capabilities that align with my domains. This is not a popular opinion in the DDD world but its a practical one.

7. Define

Here, we define the roles and responsibilities of each bounded context. For our SQL Generation context, we might use the Bounded Context Canvas to clearly outline its responsibilities, interactions, and strategic classification. For this use case it might be a little overkill but its very useful for more complex use cases.

8. Code

Finally, we “code” our domain model. We could write this entirely from scratch but why do that when we have some amazing tools like langflow, langchain and ollama at our disposal. Remember the EventStorming session we did?

What if we could translate that into a real time functional diagram that actually works.

Let's do it!

Note: Before you begin you will need to install docker desktop and complete. The instructions are here.

Step 1.

```
git clone https://github.com/ShawnKyzer/nlq-to-sql-assistant.git
```

```
cd nlq-to-sql-assistant
```

And navigate to the root project directory.

Step 2.

```
docker compose up -d
```

Step 3.

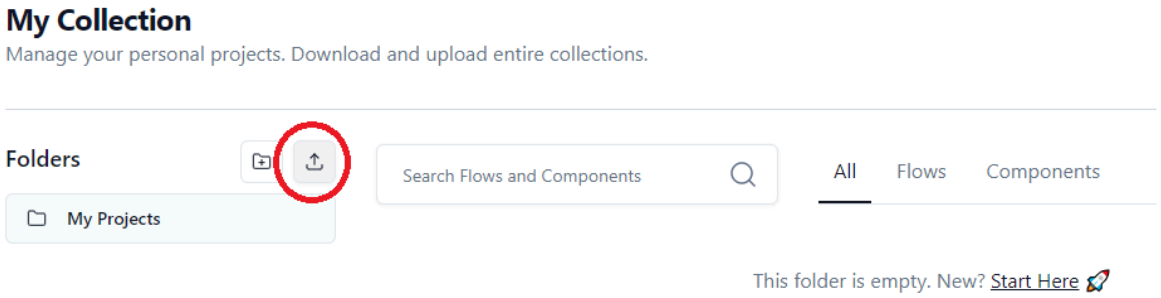
Wait a while for all things to pull and start. Note that it will also pull the llama3 7B model which does take some time. Seriously go have a ☕!

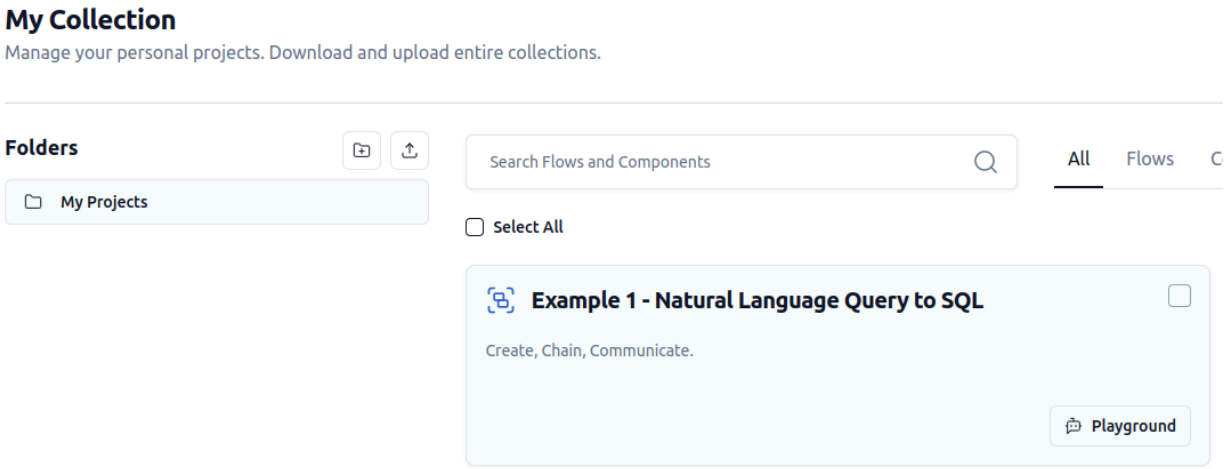
Step 4.

Navigate to langflow typically at http://localhost:7860

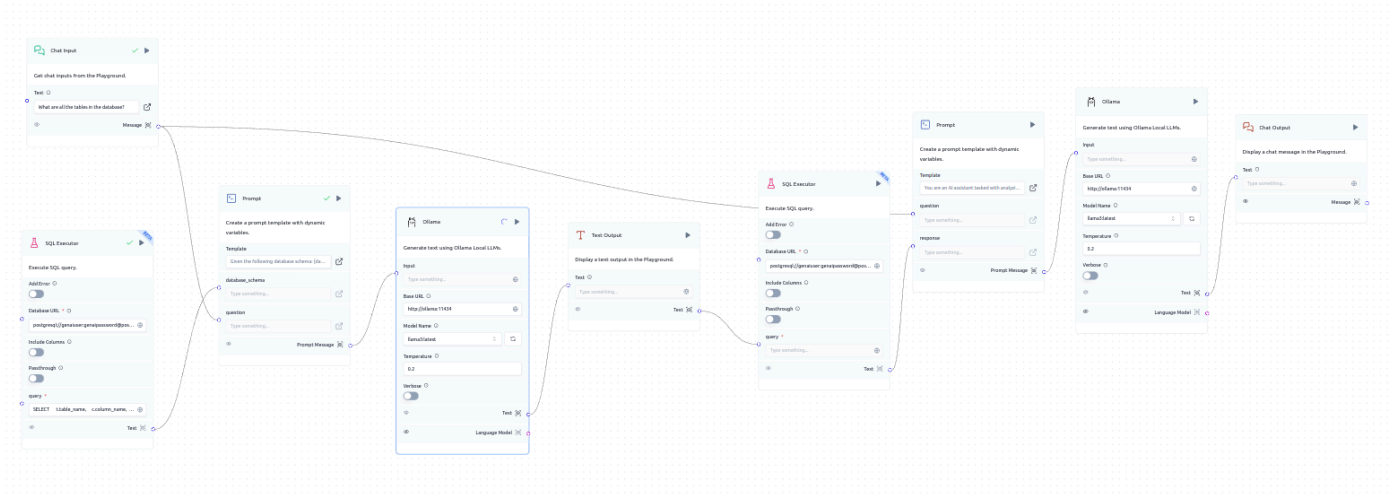
Step 5.

Import the pipeline from app_data/langflow/Example 1 - Natural Language Query to SQL .json



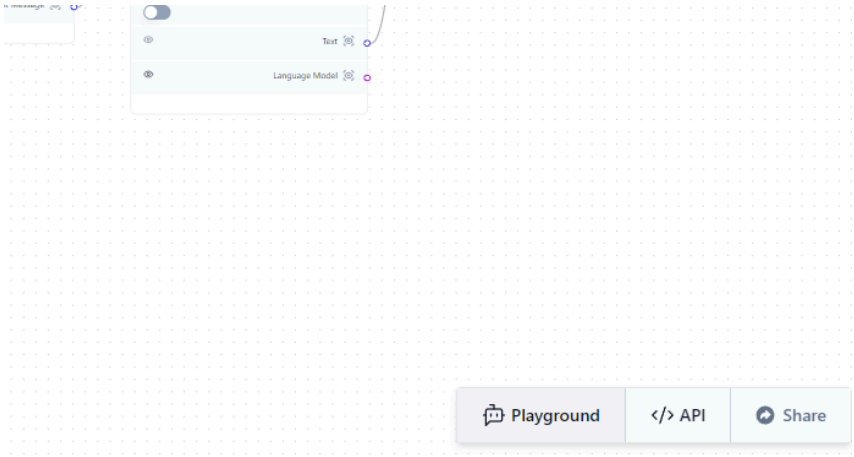


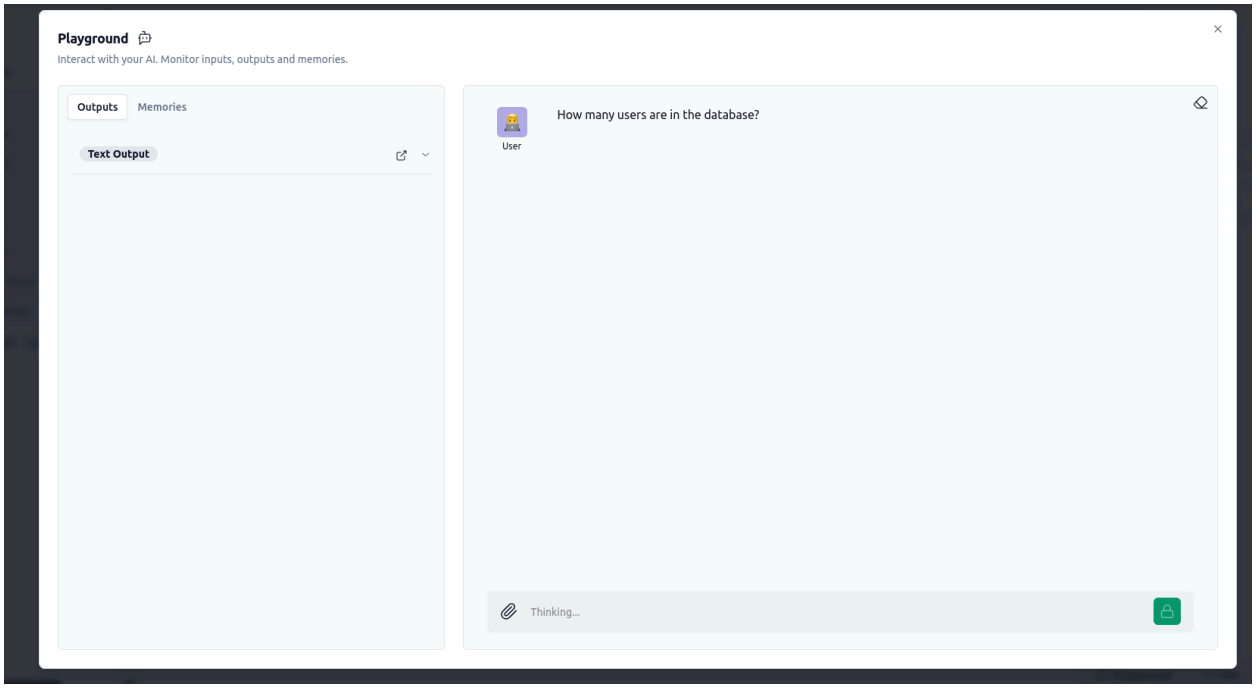
Once you have imported the pipeline you can now see the flow that we adapted from our DDD modeling exercises above. Double click on the example flow in the My Projects workspace.

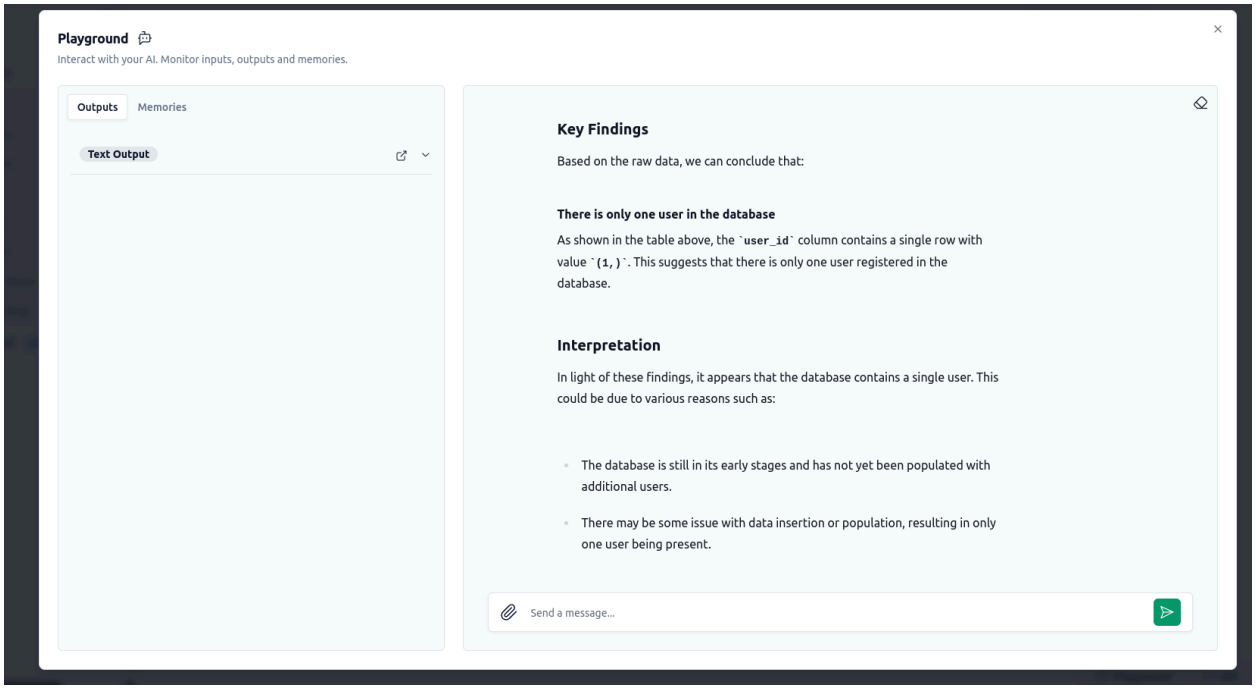


In order to initially create this pipeline we used the data collected from all the previous DDD modeling exercises.

Conveniently we have our UI built in now we just open the playground (lower right icon) and we can start asking questions of the database.







This code structure and modularity reflects our domain model, with clear separations between different aspects of the system and a focus on the core domain of **SQL generation.** It also gives us a quick and dirty UI and of course a deployable API for our UI team to build on top of.

Since we decided to build everything in a configurable and modular fashion we can now easily swap out components like the LLM with any of our choosing and we can share this template with the broader teams.

Personally, I am using La Plateforme with Mistal AI since I'm too impatient to wait for the response from a locally deployed model, they open source their model weights and it's a pretty good price point. That said there are a lot of other great alternatives such as AWS Bedrock and Microsoft OpenAI Service.

Benefits of Using the DDD Modelling Process

1. **Structured Approach** - The process provides a clear roadmap for applying DDD principles, reducing overwhelm and ensuring all key aspects are considered.

2. **Business Alignment** - By starting with understanding the business model and user needs, we ensure our technical solutions are aligned with business goals.

3. **Collaborative Modeling** - The process encourages collaboration between technical and domain experts, fostering a shared understanding of the problem space.

4. **Strategic Focus** - By identifying core domains, we can allocate resources more effectively, focusing on areas that provide the most business value.

5. **Scalable Architecture -** The decomposition and connection steps help design a modular, scalable system architecture.

6. **Team Organization** - The process considers team structure alongside technical architecture, promoting autonomous, efficient teams.

7. **Iterative Refinement -** While presented linearly, the process is designed to be iterative, allowing for continuous refinement of our understanding and design.

Conclusion

By applying Domain-Driven Design to our AI-powered SQL assistant, we've created a solution that goes beyond technical implementation to deliver real business value. This approach has allowed us to:

- **Focus on our core domain (SQL Generation)**, ensuring we address the heart of our business challenge.
- Design a **scalable** and **adaptable** system that can **evolve** with changing needs.
- **Bridge** the gap between **business requirements and technical implementation**.

- Truly **democratize data access**, empowering non-technical users to derive insights independently.

In essence, we've not just built a tool; we've enabled a fundamental shift in how our organization interacts with data. This SQL assistant, born from a business need and shaped by DDD principles, stands as a testament to the power of aligning technology closely with business strategy.

Want to dive deeper into applying DDD principles to AI system development? Want to learn about even AI Architectural Design Patterns? Join me at my upcoming ODSC workshop, where we'll walk through the DDD Starter Modelling Process for complex AI use cases, get hands-on experience with collaborative modeling techniques, and explore the latest advancements in combining DDD with AI technologies. See you there!

—

About the Author:

Shawn is a passionate technologist with over 15 years of experience in data engineering, strategy, and analytics. His expertise spans distributed computing, cloud architecture, data science, AI and machine learning. Shawn's innovative approach to technology helps businesses uncover new opportunities and deliver tangible outcomes.

https://www.linkedin.com/in/shawn-kyzer-msit-mba-b5b8a4b/

We are expanding our Data & AI teams among other roles! If you are interested in working for AstraZeneca Spain have a look at our latest career postings here.

—

References

- DDD Starter Modelling Process. GitHub. https://github.com/ddd-crew/ddd-starter-modelling-process
- Event Storming. Wikipedia. https://en.wikipedia.org/wiki/Event_storming
- Langflow. Official Website. https://www.langflow.org/
- LangChain. GitHub. https://github.com/langchain-ai
- Ollama. Official Website. https://ollama.com/
- Docker Desktop Installation Guide. Docker Documentation. https://docs.docker.com/compose/install/
- La Plateforme with Mistral AI. Mistral AI News. https://mistral.ai/news/la-plateforme/
- AWS Bedrock. Amazon Web Services. https://aws.amazon.com/bedrock
- Microsoft OpenAI Service. Microsoft Azure. https://azure.microsoft.com/en-us/products/ai-services/openai-service/
- ODSC Workshop: Designing AI Architectures with Domain-Driven Design: A Use Case-Centric Approach. ODSC Speakers. https://odsc.com/speakers/designing-ai-architectures-with-domain-driven-design-a-use-case-centric-approach/
- AstraZeneca Spain Career Postings. AstraZeneca Careers. https://careers.astrazeneca.com/location/spain-jobs/7684/2510769/2
- Fowler, M. Domain Driven Design. Martin Fowler's Bliki. https://martinfowler.com/bliki/DomainDrivenDesign.html
- Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional.