LlamaIndex 기반 LLM 시스템의 정형 데이터 질의 처리 워크플로우

워크플로우 개요

LlamaIndex를 활용한 LLM 시스템에서 사용자의 자연어 질문이 **정형 데이터 질의** (예: 데이터베이스 조회)로 분류되면, 시스템은 OracleDB 등의 데이터베이스에서 관련 데이터를 조회한 후 **표** 또는 **차트** 형태로 결과를 전달합니다. 전체 프로세스는 질의 이해부터 SQL 생성, DB 조회, 시각화 준비, 최종 응답 생성에 이르는 단계로 이루어지며, 각 단계에서 LLM과 LlamaIndex의 적절한 기능 및 **툴 호출**(Tool Calling)을 활용합니다. 아래에서는 단계별 워크플로우와 사용 기술을 설명합니다.

단계별 워크플로우

1. **의도 분류 및 파라미터 추출:** 사용자의 질문을 받으면 먼저 LLM으로 해당 질의의 유형을 판별합니다. 예를 들어 "우리 회사 분기별 실적을 알려줘"라는 질문은 정형 데이터(테이블) 조회 의도로 분류됩니다. 이 단계에서 LLM은 질문을 분석해 **조회 대상** (예: 분기별 실적 테이블, 지표 종류)과 **조건/파라미터** (예: 우리 회사, 분기별)를 추출합니다. LlamaIndex의 RouterQueryEngine 등을 사용하면 이런 분류를 LLM을 통해 자동화할 수 있습니다 [1]. RouterQueryEngine은 정의된 여러 **Query Engine** 중 어느 것을 사용할지 LLM이 선택하도록 하며, 사전에 정형 질의용 엔진과 일반 질의용 엔진을 등록해 두면 LLM이 질문에 따라 경로를 선택합니다 [1]. 의도 분류 결과 해당 질문이 **구조화 데이터 질의**로 판단되면, 이후 단계에서 DB 조회 흐름을 밟게 됩니다.

2. **SQL 쿼리 생성:** 질의 의도가 DB 조회로 결정되면, 추출된 정보로 실제 SQL 쿼리를 만듭니다. 이때 SQL은 **LLM을 통해 생성**하거나, **룰 기반 템플릿**에 파라미터를 채워 구성하는 두 가지 방식을 고려합니다. LLM을 활용하는 경우 LlamaIndex의 **NLSQLTableQueryEngine**(자연어->SQL 엔진)을 사용할 수 있습니다. LlamaIndex는 자연어 질문을 이해하여 적절한 SQL 쿼리를 자동으로 작성해주므로, 사용자는 복잡한 SQL 문법을 몰라도 됩니다 [2] [3]. 예를 들어 LlamaIndex의 NLSQLTableQueryEngine에 데이터베이스와 테이블 스키마를 지정하면, "우리 회사 분기별 실적" 같은 질문을 받아 내부적으로 GPT 모델을 이용해 `SELECT 분기, 매출, 영업이익 FROM 실적테이블 ...`과 같은 SQL을 생성합니다 [4]. 대안으로, 질문 유형별로 미리 정의된 SQL 템플릿이 있다면 분기/연도 등의 파라미터만 채워서 쿼리를 구성할 수도 있습니다.

3. **툴 호출 시점:** SQL 생성에는 LLM의 능력을 활용하므로, 필요하면 이 단계를 **툴**로 추상화할 수 있습니다. 예를 들어 LLM 에이전트가 `"GenerateSQL"`이라는 툴을 호출하면 내부적으로 GPT를 통해 SQL을 생성하도록 만들 수 있습니다. 다만 LlamaIndex의 SQLQueryEngine을 쓰는 경우 이미 엔진 내부에서 LLM이 호출되므로 별도 툴 호출이 드러나지 않고 자동화됩니다.

4. **데이터베이스 조회 실행:** 생성된 SQL 쿼리를 OracleDB (또는 연결된 RDB)에 실행하여 결과 데이터를 가져옵니다. LlamaIndex의 SQLQueryEngine(NLSQLTableQueryEngine 등)은 SQLAlchemy 등의 커넥터로 DB에 접속해 쿼리를 수행하고 결과를 받아옵니다 [5] [6]. 예를 들어, Text-to-SQL QueryEngine을 사용하면 `.query()` 호출 시 **LLM이 생성한 SQL**을 실제 DB에 보내 결과 레코드를 얻고, 이를 LLM이 읽기 좋은 형식으로 변환해줍니다 [6]. 이 단계는 **툴 호출**을 통해 수행할 수도 있습니다. LLM 에이전트 아키텍처를 취한다면, LLM이 SQL 문을 얻은 뒤 `"DBQuery"`라는 툴을 호출하여 쿼리를 실행하고 결과를 반환받는 형태로 구현 가능합니다. LlamaIndex를 활용하면 이러한 DB 질의 실행을 QueryEngine 내부 로직으로 처리하거나, 필요한 경우 **Tool abstraction**으로 커스텀 툴을 정의해 사용할 수도 있습니다 [7]. (예: `SQLQueryTool`로 Python의 DB 조회 함수를 등록하여 LLM이 선택적으로 호출)

5. **결과 시각화 준비 (표 또는 차트):** DB에서 조회된 **결과 데이터**를 사용자가 이해하기 쉽게 **표**(table) 또는 **차트**(graph) 형태로 가공합니다.

6. **표 형식 응답:** 결과 행(row)이 적당한 규모라면 LLM이 결과를 요약하거나 표 형태의 마크다운으로 직접 출력하게 할 수 있습니다. 예를 들어 분기별 실적 데이터를 표로 정리해 Markdown으로 생성하면 사용자에게 바로 표가 보입니다. 이는 LlamaIndex의 `synthesize_response=True` 옵션을 사용하면 LLM이 쿼리 결과를 자연어로 요약/응답하도록 해주는데 [5] [6], 표 형태로 출력하도록 프롬프트를 설계할 수도 있습니다.
7. **차트 시각화:** 시간 경과에 따른 실적처럼 **추세**를 보여주는 답변의 경우 차트로 시각화하면 더 직관적입니다. 시스템은 Python의 `matplotlib`나 `seaborn`으로 **서버 측에서 이미지 차트**를 생성할 수 있고, 생성된 차트를 사용자 응답에 이미지로 포함할 수 있습니다. 이러한 서버 사이드 렌더링은 LLM의 응답 본문에 `![](image_url)` 또는 해당 시스템의 이미지 embedding 규칙에 따라 삽입됩니다. LlamaIndex 자체에 차트 생성 기능이 있는 것은 아니지만, 워크플로우 상에서 **LLM 이후 단계**로서 백엔드가 차트를 그리고, 이를 **파일 경로**로서 LLM이 응답에 포함하도록 할 수 있습니다.

8. **툴 호출 시점:** 만약 에이전트 형태로 구현한다면, LLM이 쿼리 결과를 받은 후 **"ChartTool"** 등을 호출하여 파이썬으로 차트를 그리게 하고, 반환된 이미지 경로를 응답에 포함시키도록 할 수 있습니다. 즉, 차트 생성도 툴로써 LLM이 필요 시 선택하게 하는 구조를 고려할 수 있습니다.

9. **최종 응답 구성 및 전달:** 마지막으로 사용자가 보게 될 답변을 구성합니다. 표나 차트가 준비되었다면, **LLM은 해당 데이터를 간략히 설명하는 문장**을 붙여 사용자에게 전달할 답변을 만듭니다. 예를 들어 "최근 4개 분기 실적 추이를 아래 차트로 나타냈습니다."와 같은 문구입니다. LLM이 최종 응답을 꾸미는 동안 이전 단계에서 생성한 **표 Markdown**이나 **차트 이미지**를 응답에 포함시킵니다. LlamaIndex의 ResponseSynthesizer나 `compose` 기능을 활용하면 여러 소스의 정보(텍스트 요약, 표 데이터 등)를 하나의 응답으로 합성할 수 있습니다. 결국 사용자에게는 **요약 설명** + **표/차트**가 함께 제공되어, 질의에 대한 이해와 데이터 확인이 용이해집니다.

Tool Calling 활용 방안

Tool Calling이란 LLM이 질문에 답하는 과정에서 외부 **도구**(함수, API 등)를 호출하여 필요한 작업을 수행하는 방식입니다 [7]. 본 시나리오에서는 **DB 조회**나 **차트 생성** 같은 단계를 툴로 만들어, LLM이 자체 논리에 따라 적절한 시점에 호출하도록 설계할 수 있습니다. 예를 들어 LangChain 스타일의 에이전트를 만든다면, LLM이 "분기별 실적을 알려줘"라는 질문을 받고 다음과 같은 생각을 할 수 있습니다:

- **생각 1:** "이 질문은 데이터베이스 조회가 필요하군. SQL 질의를 만들어야겠어."
- **행동 1:** `GenerateSQL` 툴 호출 → (LLM이 SQL 생성 수행)
- **생각 2:** "SQL 결과를 얻었어. 데이터를 시각화하면 좋겠어."
- **행동 2:** `DBQuery` 툴 호출 → (DB에서 쿼리 실행, 결과 반환)
- **생각 3:** "결과를 차트로 보여주면 이해하기 쉽겠어."
- **행동 3:** `PlotChart` 툴 호출 → (파이썬으로 차트 생성, 이미지 파일 경로 반환)
- **생각 4:** "이제 사용자에게 설명과 함께 차트를 제공해야지."
- **행동 4:** 최종 답변 구성 (설명 텍스트 + 차트 이미지 포함)

이처럼 **툴 호출**은 LLM이 여러 작업을 단계적으로 수행하도록 하여 복잡한 질의에도 정확한 처리를 가능케 합니다. LlamaIndex도 이러한 에이전트적 흐름을 지원하며, 자체 Query Engine들을 **툴**처럼 사용할 수 있게 해줍니다 [1] [7]. 예를 들어, RouterQueryEngine 설정 시 `query_engine_tools`로 각 엔진을 등록해두면 LLM이 마치 툴처럼 특정 엔진(예: SQL 엔진)을 선택해 사용합니다 [1]. **단일 단계 질문**의 경우 RouterQueryEngine만으로 충분하지만, 복잡한 멀티스텝 작업엔 LLM 에이전트+툴 구조가 도움이 됩니다. 요약하면: - **의도 분류** 단계: RouterQueryEngine 같은 **자동 경로 선택** 도구 활용 (LLM 내재 호출). - **SQL 생성/실행** 단계: LlamaIndex SQLQueryEngine (내부적으로 LLM 및 DB툴 사용) 또는 명시적 에이전트 툴 호출. - **차트 생성** 단계: 필요한 경우 커스

텀 툴로 구현하여 LLM이 호출 or 백엔드 자동 수행. - LlamaIndex의 **Tool abstraction**을 사용하면 이러한 툴들을 쉽게 통합하여 에이전트가 활용할 수 있습니다 [7].

차트 렌더링: 서버 측 vs 프론트엔드

데이터 시각화를 사용자에게 전달하는 방법으로 **서버 측 이미지 생성**과 **프론트엔드 렌더링** 두 가지가 있습니다. 각각의 선택 기준과 권장 방식은 다음과 같습니다:

- **서버 측 이미지 렌더링:** 백엔드에서 `matplotlib` 등의 라이브러리로 차트 이미지를 생성한 후 응답에 포함합니다. **장점:** 사용자는 별도 조치 없이 바로 이미지를 볼 수 있고, LLM의 응답 내에 이미지(예: `![Chart](url)` 형태)로 포함되므로 일관된 형식으로 전달됩니다. Chatbot 인터페이스나 PDF 리포트 등에는 이 방식이 편리합니다. **단점:** 차트가 정적 이미지이므로 상호작용이 불가능하며, 서버에서 이미지 생성에 추가 리소스가 듭니다. 또한 스타일이나 해상도를 사전에 정의해야 합니다.

- **프론트엔드 렌더링 (JSON 데이터 전송):** 백엔드는 차트로 그릴 **데이터를 JSON** 등 구조로 전달하고, 프론트엔드에서 JavaScript 라이브러리(e.g. D3.js, Chart.js, 혹은 Enterprise Dashboards)를 통해 차트를 그립니다. **장점:** 클라이언트에서 차트를 동적으로 렌더링하므로 **인터랙티브 기능**(마우스오버 설명, 확대 등)이나 **스타일 일관성**(여러 답변 간 동일 테마 적용)을 쉽게 구현할 수 있습니다. 또한 데이터 자체를 주고받으므로 해상도 제약이 없고, 클라이언트 기기 해상도에 맞춰 렌더링됩니다. **단점:** 프론트엔드에 추가 구현이 필요하고, 실시간 채팅 답변에 바로 이미지를 띄우는 것보다는 구현 복잡도가 높습니다. 또한 JSON으로 큰 데이터를 보내면 전송량이 늘어날 수 있습니다.

권장 접근: 시스템의 목적과 클라이언트 환경에 따라 혼용 가능합니다. **챗봇 형태**의 응답이라면 서버 측에서 **이미지로 차트를 포함**시키는 방법이 구현이 쉬우며, LLM 답변과 함께 즉시 볼 수 있어 좋습니다. 반면 사내 대시보드나 웹 애플리케이션에 임베드 되는 경우 **프론트엔드 렌더링**이 향후 확장성과 사용자 경험 측면에서 유리합니다. 예컨대, 사용자가 차트를 클릭해 상세 데이터를 보거나 기간 필터를 조정할 수 있게 하려면 클라이언트 측에서 렌더링해야 합니다. 따라서 **정적인 Q&A 응답**에는 서버 측 렌더링을, **인터랙티브 BI 애플리케이션**에는 JSON 데이터 전달 및 프론트렌더링을 권장합니다. 경우에 따라 두 방식을 혼합해, 기본은 이미지를 보여주되 "데이터 다운로드" 옵션으로 원본 JSON을 제공할 수도 있습니다.

LLM과 LlamaIndex의 역할 분담

이 워크플로우에서 **LLM**과 **LlamaIndex**가 맡는 역할을 구분하면 다음과 같습니다:

- **LLM (Large Language Model):** 자연어 **이해**와 **생성**을 담당합니다. 사용자 질문의 의도를 파악하고 필요한 정보를 추론하며, 최종 답변 문장을 만들어 내는 주체입니다. 예를 들어 "분기별 실적"이라는 요구를 보고 어떤 데이터를 원하고 어떤 형식이 적합할지 결정하는 사고를 합니다. 또한 쿼리 결과를 사용자 친화적으로 설명하는 문장을 생성하는 것도 LLM의 몫입니다. SQL 생성 역시 LLM의 언어 능력에 의해 이뤄집니다 (프롬프트에 스키마 정보를 주면 LLM이 SQL문 작성) [2]. 요약하면, **자연어↔구조화된 요청** 간 변환과 답변 서술을 LLM이 수행합니다.

- **LlamaIndex:** LLM의 이러한 능력을 실제 어플리케이션에 **연결**시켜 주는 **플랫폼/프레임워크** 역할을 합니다. LlamaIndex는 데이터베이스, 문서 등 다양한 **데이터 소스와 LLM을 중개**하여 질의 응답 시스템을 구축하는데, 본 시나리오에서는 **DB 연동과 질의 라우팅**에 활용됩니다. 예를 들어 LlamaIndex의 SQLQueryEngine (또는 NLSQLTableQueryEngine)은 LLM이 생성한 SQL을 실행하고 결과를 다시 LLM이 이해할 수 있는 형태로 전달합니다 [5]. 또한 RouterQueryEngine을 통해 질의 유형에 따라 LLM의 처리 경로를 분기시켜주는 로직을 제공하며, 이를 통해 LLM이 **적절한 툴/데이터소스**를 선택하게 합니다 [1]. LlamaIndex의 **Tool abstraction**은 LLM이 외부 함수를 호출하는 기능을 추상화하여, 개발자가 Python 함수들을 손쉽게 LLM의 도구로 등록할

수 있게 해줍니다 [7]. 요약하면, LlamaIndex는 **LLM의 능력을 효과적으로 끌어내어 데이터에 접근**하게 해주는 **구조화된 틀**이며, 개발 편의 기능 (SQL 엔진, 라우터, 툴 등)을 제공하여 전체 시스템을 구성합니다.

아키텍처 다이어그램 (예시)

아래는 설명한 워크플로우의 아키텍처를 도식화한 예시입니다. 각 단계에서 LLM과 LlamaIndex 모듈, 툴이 어떻게 흐름을 이루는지 보여줍니다:

```
User Query ("분기별 실적 알려줘")
      ⬇
[LLM 기반 의도 분류 (LlamaIndex Router)]
      ⬇ (정형 데이터 질의로 분류됨)
[LLM Text-to-SQL 생성 (LlamaIndex SQLQueryEngine)]
      ⬇ (생성된 SQL 쿼리)
[OracleDB 실행 – DB Tool 또는 QueryEngine 내부]
      ⬇ (조회된 결과 데이터)
[시각화 처리: 차트 생성 또는 표 포맷팅]
      ⬇
[LLM 최종 응답 구성]
      ⬇
사용자에게 표/차트 응답 전달
```

이 그림에서 보듯, **프론트엔드**는 질의를 보내고 최종 답변(차트/표 포함)을 받는 단순한 형태이고, **백엔드**에서는 LLM+LlamaIndex가 협력하여 질의를 분석하고, 필요하면 툴을 호출해(DB 조회, 차트 생성) 결과를 준비한 뒤 응답을 생성합니다. 이렇게 구성된 시스템은 사용자가 자연어로 데이터 질의를 하면 백엔드에서 자동으로 SQL을 통해 데이터를 가져오고 시각화하여 제공하므로, 기술적 세부사항을 몰라도 손쉽게 데이터에 대한 **인사이트**를 얻을 수 있습니다 [2].

---

[1] [7] Agentic RAG Application using LlamaIndex — Router Query Engine | by Abdul Samad | Medium
https://medium.com/@samad19472002/agentic-rag-application-using-llamaindex-router-query-engine-5b3f7b7feb75

[2] [3] [5] [6] “Using Llama Index: Querying Databases in Plain English without SQL Expertise” | by Shaelander Chauhan | Medium
https://medium.com/@shaelanderchauhan/using-llama-index-querying-databases-in-plain-english-without-sql-expertise-428b32568c17

[4] Text-to-SQL Guide (Query Engine + Retriever) - LlamaIndex
https://docs.llamaindex.ai/en/stable/examples/index_structs/struct_indices/SQLIndexDemo/