

An LLM-Assisted Heterogeneous DBMS Migration Model

Dong-Bin Lim*, Eun-jin Oh**, Jae-Woong Kim***

*Student, Dept. of Software, Kongju National University, Kongju, Korea

**Ph.D. Candidate, Dept. of Software, Kongju National University, Kongju, Korea

***Professor, Dept. of Software, Kongju National University, Kongju, Korea

[Abstract]

This study proposes an LLM-assisted migration model for heterogeneous DBMS environments that integrates T0-based initial loading, LLM-based SQL transformation and normalization, deterministic SQL post-processing, and CDC-based synchronization. Unlike prior studies that treat SQL dialect translation, LLM-based SQL generation, and change replication separately, the proposed model combines these functions in a single migration workflow. Source metadata and object definitions are embedded in DBMS-pair-specific prompts, and executable SQL is extracted and corrected before target application. MariaDB-to-PostgreSQL validation confirmed table loading, procedural-object transformation, CDC application, checkpoint management, and execution logging. Source and target row counts matched for four tables containing 2,980 rows; two procedures, two functions, and two triggers were applied without SQL execution errors; and all 162 CDC events were processed without failure. No manual SQL correction was required, and total processing time was 449.328 seconds. Academically, the model structures LLM-based SQL transformation as a post-processed migration workflow; practically, it reduces manual correction and improves traceability. Future work will evaluate additional DBMS pairs, LLMs, recovery behavior, and larger workloads.

▶ **Key words:** Heterogeneous DBMS Migration, LLM, SQL Transformation, CDC, Procedural Objects

[요 약]

본 연구는 이기종 DBMS 마이그레이션에서 발생하는 SQL 문법 차이, 절차적 객체 변환 및 초기 적재 이후 변경분 반영 문제를 완화하기 위해 T0 기반 초기 적재, LLM 기반 SQL 변환·정규화·결정적 후처리, CDC 기반 동기화를 통합한 모델을 제안한다. 기존 연구가 SQL 방언 변환, LLM 기반 SQL 생성 및 변경 데이터 복제를 개별적으로 다룬 것과 달리, 제안 모델은 이를 하나의 마이그레이션 흐름으로 구성한다. MariaDB-PostgreSQL 환경에서 검증한 결과, 4개 테이블 2,980건의 소스·타겟 행 수가 일치하였고, 프로시저 2개, 함수 2개, 트리거 2개와 CDC 이벤트 162건이 오류 없이 적용되었다. 수동 SQL 보정은 0회였으며 전체 처리시간은 449.328초였다. 본 연구는 LLM 기반 SQL 변환을 후처리 및 CDC 동기화와 결합된 실행 절차로 구조화하고, 수동 보정과 이관 결과 추적 부담을 줄일 가능성을 제시한다. 향후 다양한 DBMS와 LLM 조합을 대상으로 확장 검증한다.

▶ **주제어:** 이기종 DBMS 마이그레이션, 대규모 언어 모델, SQL 변환, 변경 데이터 캡처, 절차적 객체

- First Author: Dong-Bin Lim, Corresponding Author: Jae-Woong Kim
- *Dong-Bin Lim (dblim@j-it.co.kr), Dept. of Software, Kongju National University
- **Eun-jin Oh (oej73@daum.net), Dept. of Software, Kongju National University
- ***Jae-Woong Kim (jykim@kongju.ac.kr), Dept. of Software, Kongju National University
- Received: 2026. 07. 21, Revised: 2026. 07. 27, Accepted: 2026. 08. 19.

I. Introduction

현대 IT 인프라 환경에서 데이터는 기업의 핵심 자산이며, 데이터를 관리하는 DBMS의 유연한 운영은 비즈니스 연속성을 위한 필수 요건이다. 그러나 특정 DBMS 벤더에 대한 기술 종속은 라이선스 비용 상승과 시스템 확장성 저해의 원인이 될 수 있다[1]. 특히 상용 DBMS에서 다른 상용 DBMS 또는 오픈소스 기반 DBMS로 마이그레이션하려는 수요는 증가하고 있으나, 이기종 DBMS 간 구조적 차이로 인해 마이그레이션의 기술적 복잡성과 비용 부담이 발생한다[2]. 이기종 DBMS 마이그레이션에서 발생하는 스키마 계층 구조의 비호환성, 데이터 타입 차이, 절차적 객체(프로시저, 함수, 트리거 등)의 문법 차이는 마이그레이션 복잡성을 심화시키는 핵심 요인이다[3][4]. 또한 기존 마이그레이션 방식은 DBA가 장기간 스키마 재설계와 SQL 변환을 직접 수행해야 하는 경우가 많아, 반복 작업과 휴먼 에러에 따른 데이터 품질 및 무결성 위험이 발생할 수 있다[4].

기존 규칙 기반(rule-based) SQL 변환 방식은 사전에 정의된 변환 규칙에 의존하므로, DBMS별 특수 문법이나 복잡한 절차적 객체를 처리하는 데 한계가 있다. 특히 프로시저, 함수, 트리거와 같은 절차적 객체는 변수 선언, 반환 타입, 예외 처리, 이벤트 참조 방식 등 실행 문맥과 DBMS별 절차 언어 특성을 함께 고려해야 하므로 단순 규칙 기반 접근만으로는 완전한 자동화가 어렵다[3][4]. 최근 대규모 언어 모델(LLM)과 머신러닝(ML) 기술의 발전은 이러한 복잡 업무의 자동화 가능성을 확대하고 있다[5][6]. LLM은 SQL 및 객체 정의문의 문맥 정보를 학습하여 다양한 문법 패턴을 일반화할 수 있으므로, 이기종 DBMS 환경에서 보다 유연한 SQL 변환을 지원할 수 있다. 그러나 LLM의 생성 결과가 항상 실행 가능한 SQL임을 보장하지는 않기 때문에, 변환 결과에 대한 정규화와 후처리 절차가 함께 요구된다[7][8].

이를 위해 LLM 기반 SQL 변환 파이프라인을 중심으로 이기종 DBMS 마이그레이션 자동화 시스템을 설계하였다. 제안 시스템은 소스 DBMS의 메타데이터와 객체 정의를 수집하고, 이를 기반으로 DBMS 조합별 프롬프트 템플릿을 로딩한 뒤 메타데이터와 객체 정의를 변수로 주입하여 LLM 변환 요청을 구성한다. 이후 LLM 기반 SQL 변환 및 후처리 과정을 통해 실행 가능한 SQL을 추출·보정하고 타깃 DBMS에 적용한다. 또한 데이터 타입 매핑, 스키마 문맥 반영, SQL 정규화, 문법 보정 과정을 통합하여 스키마 객체와 절차적 객체의 변환 안정성을 높이고자 한다[7]. 이

와 함께 CDC(Change Data Capture) 기반 변경분 동기화 구조와 체크포인트 관리를 적용하여 초기 이관 이후 발생하는 변경 데이터를 타깃 DBMS에 지속적으로 반영하고, 서비스 중단과 데이터 불일치 가능성을 줄이고자 한다[9][10].

본 연구의 주요 기여는 다음과 같다.

첫째, 이기종 DBMS 환경에서 스키마와 절차적 객체의 변환을 지원하기 위한 LLM 기반 SQL 변환 및 후처리 구조를 설계하였다.

둘째, 생성된 SQL의 타깃 DBMS 실행 가능성과 구문적 적합성을 높이기 위해 SQL 추출, 정규화, 결정적 후처리 및 제한적 재변환 절차를 적용하였다.

셋째, T0 및 일관 스냅샷 기반 초기 적재와 체크포인트 기반 CDC 동기화를 연결하여 초기 적재 이후 발생한 변경 데이터를 반영하고 CDC 재시작 위치를 관리하도록 구성하였다.

넷째, MariaDB에서 PostgreSQL로의 마이그레이션 환경에서 4개 테이블 2,980건, 절차적 객체 6개 및 CDC 이벤트 162건을 대상으로 기능 검증을 수행하여 제안 절차의 적용 가능성을 확인하였다.

II. Preliminaries

1. Heterogeneous DBMS Migration

마이그레이션(Migration)은 기존 시스템의 데이터, 스키마, 객체 정의, 권한 정보 등을 다른 환경으로 이전하는 과정을 의미한다[2]. 동일 DBMS 간 마이그레이션은 구조와 문법 차이가 상대적으로 작지만, 이기종 DBMS 마이그레이션(Heterogeneous DBMS Migration)은 서로 다른 DBMS 사이에서 수행되므로 데이터 타입, 스키마 계층 구조, 제약 조건, 객체 정의 문법 등의 차이를 함께 고려해야 한다[4].

이기종 DBMS 마이그레이션에서는 원본 환경을 소스 DBMS, 변환 결과가 적용되는 환경을 타깃 DBMS라고 한다. 소스 DBMS와 타깃 DBMS 사이의 구조적 차이가 클수록 단순 데이터 이동만으로는 완전한 마이그레이션을 보장하기 어렵다. 따라서 데이터 이전뿐 아니라 스키마 분석, 객체 변환, 데이터 동기화, 오류 복구 등을 포함하는 통합적인 절차가 요구된다.

스키마(Schema)는 DBMS 내부에서 데이터 구조를 정의하는 논리적 단위이며 테이블, 컬럼, 데이터 타입, 기본 키, 외래 키, 인덱스, 제약 조건 등을 포함한다. 테이블

(Table)은 실제 데이터를 저장하는 기본 객체이며 마이그레이션 과정에서 가장 먼저 분석되는 대상이다.

메타데이터(Metadata)는 데이터베이스 구조를 설명하는 정보로서 테이블 이름, 컬럼 이름, 데이터 타입, 제약 조건, 인덱스, 객체 정의 등을 포함한다. 이기종 DBMS 환경에서는 동일한 의미를 가지는 데이터 타입이나 제약 조건도 서로 다른 문법으로 표현될 수 있으므로, 메타데이터는 변환 대상 식별과 변환 규칙 결정의 핵심 자료로 활용된다[4].

이기종 DBMS 마이그레이션에서는 테이블 구조뿐 아니라 절차적 객체(Procedural Objects)의 변환도 중요한 과제로 다루어진다. 대표적인 절차적 객체에는 프로시저(Procedure), 함수(Function), 트리거(Triple)가 있다. 프로시저는 반복적으로 수행되는 처리 절차를 DBMS 내부에 저장한 실행 단위이며, 함수는 입력값을 받아 특정 결과를 반환하는 객체이다. 트리거는 특정 테이블에서 'INSERT', 'UPDATE', 'DELETE' 등의 이벤트가 발생했을 때 자동으로 실행되는 객체이다.

절차적 객체는 변수 선언, 조건문, 반복문, 예외 처리, 반환 타입, 내장 함수 호출, 트랜잭션 처리 등의 로직을 포함하며, 이러한 요소는 DBMS별 절차 언어와 실행 모델에 강하게 의존한다[3]. 예를 들어 변수 선언 방식, 반환 타입 정의 방식, 예외 처리 구조, 트리거 이벤트 참조 방식 등은 DBMS마다 다르게 구현될 수 있다. 따라서 절차적 객체 변환은 단순 문자열 치환이나 규칙 기반 변환만으로 해결하기 어렵고, 객체의 의미와 실행 문맥을 고려한 변환 과정이 요구된다[11][12].

2. LLM-based SQL Transformation

대규모 언어 모델(Large Language Model, LLM)은 자연어와 프로그램 코드의 구조적 패턴을 학습한 생성 모델이다[5]. 최근에는 LLM을 활용하여 SQL 문장이나 데이터베이스 객체 정의를 분석하고, 이를 다른 DBMS 문법에 맞게 변환하는 연구가 수행되고 있다[6][7].

이기종 DBMS의 SQL 변환에서는 데이터 타입, 내장 함수, 제약 조건, 반환 타입 및 절차 언어 문법 등의 차이를 함께 고려해야 한다. LLM은 다양한 SQL과 프로그램 코드의 패턴을 바탕으로 이러한 차이를 문맥적으로 처리할 수 있으므로, 사전에 정의된 규칙만으로 처리하기 어려운 SQL 및 절차적 객체 변환을 보완하는 방법으로 활용될 수 있다[7].

그러나 LLM이 생성한 결과는 타깃 DBMS에서의 실행 가능성과 의미적 동등성을 항상 보장하지 않는다. 생성 결

과에는 설명문, 주석, Markdown 코드 블록과 같은 비실행 텍스트가 포함될 수 있으며, 데이터 타입 불일치, 반환 타입 오류, 제약 조건 오류, 객체 생성 순서 문제 및 예약어 충돌 등이 발생할 수 있다[7][8]. 따라서 LLM 응답을 타깃 DBMS에 적용하기 전에 SQL 정규화와 후처리 과정이 필요하다.

본 연구에서 SQL 정규화(Normalization)는 LLM 응답에 포함된 설명문, Markdown 형식 및 기타 비실행 텍스트를 제거하고 실행 가능한 SQL 본문을 추출하는 과정을 의미한다. 후처리(Post-processing)는 데이터 타입 불일치, 외래키 및 CHECK 제약 오류, 반환 타입 불일치, 객체 생성 순서 문제와 DBMS별 예약어 충돌 등의 오류 가능 패턴을 점검하고 보정하여, 타깃 DBMS에서의 실행 가능성과 구문적 적합성을 높이는 절차를 의미한다.

3. Data Synchronization

운영 중인 데이터베이스를 마이그레이션하는 경우 초기 데이터 적재가 수행되는 동안에도 소스 DBMS에서는 새로운 데이터 변경이 지속적으로 발생할 수 있다. 따라서 특정 기준 시점의 데이터 상태를 확보하고, 이후 발생한 변경분을 별도로 반영하는 동기화 전략이 필요하다.

T0 Point는 초기 데이터 적재와 이후 변경분 반영을 연결하는 기준 시점을 의미한다. 마이그레이션 시스템은 T0 시점의 로그 위치를 기록하고, 일관 스냅샷(Consistent Snapshot)을 이용하여 동일한 기준 시점의 데이터를 읽는다. 이를 통해 초기 데이터 복사 중 소스 데이터가 변경되더라도 읽기 시점의 불일치를 줄이고, 이후 변경분 동기화를 위한 기준 상태를 확보할 수 있다[13].

CDC(Change Data Capture)는 소스 DBMS에서 발생하는 INSERT, UPDATE, DELETE 등의 변경 이벤트를 감지하여 타깃 DBMS에 반영하는 기술이다[9][10]. 초기 적재와 CDC를 함께 적용하면 전체 데이터를 반복적으로 복사하지 않고도 T0 이후 발생한 변경분을 반영할 수 있으며, 최종 전환 시점에서 소스와 타깃 데이터 간의 차이를 줄일 수 있다.

체크포인트(Checkpoint)는 CDC 이벤트의 최종 처리 위치를 저장하는 기준점이다[13]. 변경 이벤트가 타깃 DBMS에 성공적으로 반영된 후 체크포인트를 갱신하면, 동기화 작업이 중단되거나 장애가 발생하더라도 마지막 처리 위치부터 작업을 재개할 수 있다.

따라서 T0, 일관 스냅샷, CDC 및 체크포인트를 함께 적용함으로써 초기 적재와 이후 변경분 반영을 하나의 연속적인 동기화 절차로 구성할 수 있다.

4. Comparison with Related Approaches

앞서 검토한 DBMS 마이그레이션, LLM 기반 SQL 처리 및 데이터 동기화 접근의 주요 범위와 제안 모델의 차이를 Table 1에 요약하였다.

Table 1. Comparison with Related Approaches

Approach	Scope and Key Difference
Conventional DBMS Migration [2][4]	Focuses on schema and data migration, but does not integrate LLM-based procedural-object transformation or deterministic SQL post-processing.
LLM-based SQL Generation [5][8]	Focuses on Text-to-SQL generation and self-correction, but does not address migration of existing DBMS objects or CDC-based synchronization.
Hybrid SQL Dialect Translation [7]	Combines rule-based processing and LLM-based SQL translation, but does not include T0-based loading, checkpointed CDC, or execution logging.
CDC and Replication [9][10]	Focuses on change capture and near-real-time replication, but does not transform schemas or procedural objects.
Proposed Model	Integrates LLM-based transformation, SQL normalization and deterministic post-processing, T0-based loading, CDC synchronization, checkpoints, and execution logs.

본 연구의 핵심 차별성은 메타데이터 기반 LLM 객체 변환과 SQL 정규화·결정적 후처리를 T0 기반 초기 적재 및 체크포인트 기반 CDC와 결합하여, 변환 결과의 적용과 추적이 가능한 단일 이기종 DBMS 마이그레이션 흐름으로 구성한 데 있다.

III. The Proposed Scheme

1. Overview

본 장에서는 LLM 기반 SQL 변환과 CDC 기반 데이터 동기화를 결합한 이기종 DBMS 마이그레이션 절차를 설명한다. 제안 절차는 소스 및 타깃 DBMS에 연결한 후 소스 DBMS의 테이블 메타데이터, 객체 정의 및 스키마 문맥을 수집하는 것으로 시작한다. 수집된 정보를 DBMS 조합별 프롬프트 템플릿에 반영하여 타깃 테이블 DDL을 생성하고, LLM 응답에서 실행 가능한 SQL을 추출·정규화·보정한 후 타깃 DBMS에 적용한다.

타깃 테이블이 준비되면 소스 DBMS의 로그 위치를 T0 커서로 기록하고 일관 스냅샷을 시작한다. 이후 스냅샷 데이터를 타깃 테이블에 적재하고, 프로시저·함수·트리거와

같은 절차적 객체를 타깃 DBMS 문법으로 변환하여 적용한다. 초기 적재와 객체 적용이 완료된 후에는 기존 체크포인트 또는 T0 위치부터 CDC를 재개하여 변경 이벤트를 반영하고, 최종 검증 결과와 실행 이력을 저장한다.

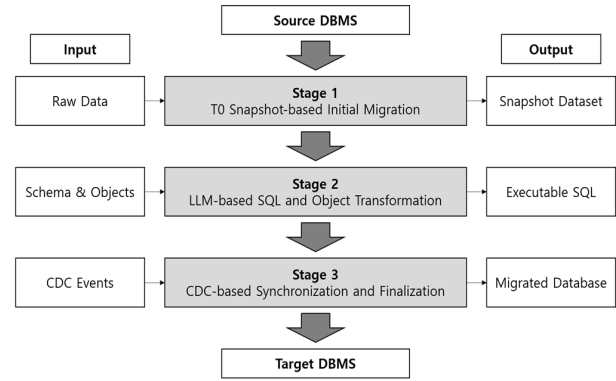


Fig. 1. Overall Architecture of the Proposed Migration Scheme

Fig. 1은 제안 모델을 구성하는 처리 단계와 단계별 산출물의 관계를 나타낸다. Algorithm 1은 Fig. 1의 처리 흐름을 입력, 출력 및 주요 처리 순서를 중심으로 정리한 의사코드이다.

Algorithm 1. Proposed Heterogeneous DBMS Migration Procedure

Input: Source and target DBMS connections, selected tables and procedural objects, and a DBMS-pair-specific prompt template
Output: Migrated data and objects, applied CDC events, a CDC checkpoint, and execution logs

- 1: Collect source metadata, object definitions, and schema context.
- 2: Transform the target table DDL using the LLM.
- 3: Extract executable SQL, normalize it, and apply deterministic post-processing.
- 4: If predefined risk patterns are detected, retry the transformation within the predefined retry limit.
- 5: Create the target tables.
- 6: Pause CDC processing if active, capture the T0 cursor, and begin a consistent snapshot.
- 7: Load the snapshot data into the target tables.
- 8: Transform and apply procedures, functions, and triggers.
- 9: Resume CDC processing from T0 and apply change events.
- 10: Update the checkpoint after each successful CDC application, perform final validation, and record the execution results.

2. T0 Snapshot-based Initial Migration

첫 번째 단계는 초기 마이그레이션의 기준 시점인 T0를 설정하고, 해당 시점의 테이블 데이터를 타깃 DBMS에 일

관된 상태로 적재하는 단계이다. MariaDB/MySQL 계열 소스 DBMS에서는 스냅샷 데이터의 기준점과 변경 로그의 재생 기준점이 어긋나지 않도록, 실행 중인 CDC 프로세스가 있는 경우 이를 일시적으로 중지한다. 이후 소스 DBMS의 로그 위치를 T0 커서로 기록하고, 동일한 기준 시점의 데이터를 읽기 위해 일관 스냅샷 트랜잭션을 시작한다[13][14].

일관 스냅샷이 시작되면 선택된 테이블의 데이터를 읽어 미리 생성된 타깃 테이블에 적재한다. 각 테이블의 DDL 적용 결과와 데이터 적재 결과는 실행 로그에 기록하며, 데이터 적재가 완료되면 스냅샷 트랜잭션을 종료한다. 이후 기존 CDC 체크포인트가 없는 경우에는 기록된 T0 커서를 CDC 재시작 위치로 사용한다.

3. LLM-based SQL and Object Transformation

LLM 기반 SQL 및 객체 변환 단계는 소스 DBMS에서 수집한 스키마 및 객체 정의를 타깃 DBMS에서 실행 가능한 SQL로 변환하는 과정이다. 이 단계에서는 테이블 정의, 프로시저, 함수, 트리거의 원문 정의를 수집하고, 각 객체가 참조하는 테이블과 컬럼 정보를 포함한 스키마 문맥을 함께 구성한다[3][11][12]. 이후 DBMS 조합별 Markdown 프롬프트 템플릿에 메타데이터, 원문 객체 정의, 스키마 문맥을 주입하여 LLM에 SQL 변환을 요청한다[7].

LLM이 생성한 결과는 그대로 실행하지 않고, SQL 정규화와 후처리 과정을 거친다. 이 과정에서는 SQL 코드 블록 또는 실행 가능한 SQL 시작 구문을 기준으로 SQL 본문을 추출하고, 잘못된 외래키 참조, 날짜-시간 타입 오변환, 소스에 없는 CHECK 제약 등을 보정한다. 프로시저와 함수 변환에서는 RETURNS TABLE과 RETURN QUERY의 반환 타입 정합성 문제가 예상되는 경우 재시도 프롬프트를 추가하여 다시 변환을 요청한다[7]. 최종적으로 정리된 SQL은 타깃 DBMS에서 실행되어 객체 변환 결과를 반영한다. 이러한 LLM 기반 SQL 및 객체 변환의 세부 처리 흐름을 Fig. 2에 나타내었다.

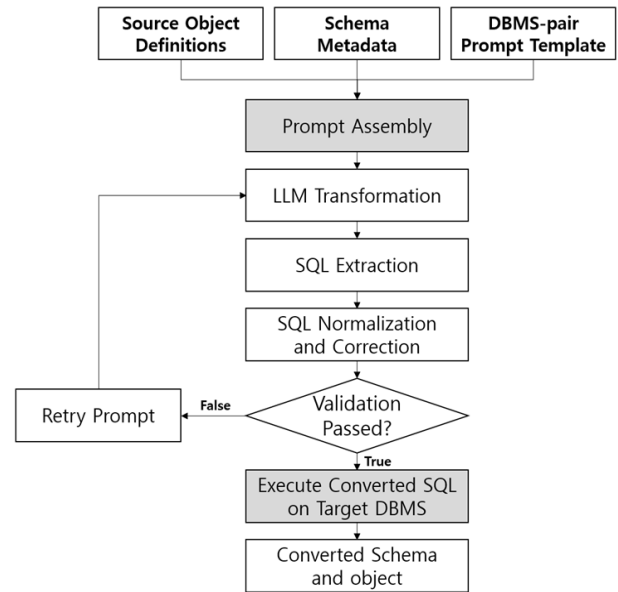


Fig. 2. Workflow of LLM-based SQL and Object Transformation

4. CDC-based Synchronization and Finalization

세 번째 단계는 초기 적재 이후 T0 시점부터 발생한 변경 데이터를 반영하고 마이그레이션의 결과를 최종 정리하는 단계이다. 초기 적재와 객체 변환이 완료되면 앞 단계에서 확보한 T0 커서를 체크포인트로 기록하고, CDC를 T0 위치부터 재시작한다. 이후 소스 DBMS의 INSERT, UPDATE, DELETE 변경 이벤트를 타깃 DBMS에서 실행 가능한 SQL로 변환하여 순차적으로 적용한다[9][13].

CDC 기반 동기화는 초기 적재 시점 이후 발생한 변경 이벤트를 반영함으로써, 초기 적재 결과와 최종 소스 데이터 사이의 불일치를 줄이는 역할을 한다. 변경분 적용 이후에는 선택적으로 최종 동기화(Finalize Sync)를 수행하여 소스 DBMS와 타깃 DBMS의 메타데이터를 비교하고, 누락되거나 실패한 테이블 및 절차적 객체를 재실행 대상으로 등록한다. 마지막으로 전체 실행 결과, 체크포인트, 오류 이력을 로그로 기록하여 마이그레이션 과정의 추적 가능성을 확보한다.

IV. Experimental Results

1. Experimental Setup and Procedure

본 장에서는 제안 모델의 주요 기능이 의도한 흐름에 따라 수행되는지 확인하기 위한 실험 결과를 정리한다. 본 연구의 평가는 실제 프로덕션 부하를 재현한 성능 평가가 아니라, 이기종 DBMS 마이그레이션 과정에서 핵심 기능

이 정상적으로 동작하는지 확인하는 기능 검증 중심으로 수행하였다.

실험은 소스 DBMS와 타깃 DBMS를 연결한 뒤, 선택된 테이블과 절차적 객체를 대상으로 마이그레이션 실행, LLM 기반 SQL 변환, SQL 정규화, CDC 동기화, 실행 로그 기록 여부를 확인하는 방식으로 진행하였다. AI 기반 변환은 Ollama 환경에서 실행되는 llama3를 사용하여 검증하였다[15]. Table 2는 실험 환경, 검증 데이터 규모 및 검증 범위를 보여준다.

Table 2. Experimental Environment and Validation Scope

Item	Description
Evaluation Type	Functional Validation
Source DBMS	MariaDB
Target DBMS	PostgreSQL
LLM Runtime	Ollama 0.20.2
AI Model	llama3 (8.0B, Q4_0)
Validation Dataset	4 tables; 2,980 rows; 2 procedures; 2 functions; 2 triggers
CDC Workload	162 events: 100 INSERTs, 42 UPDATEs, and 20 DELETEs
Validation Scope	SQL transformation and normalization, data loading, procedural-object application, CDC synchronization, checkpoint management, and execution logging

실험 검증 절차는 실험 환경을 구성하고 소스 DBMS와 타깃 DBMS의 연결을 확인하였다. 이후 검증 대상 객체를 선택하고 사전점검을 수행한 뒤, 마이그레이션을 실행하였다. 실행 과정에서는 LLM 기반 SQL 변환 결과와 절차적 객체 변환 결과를 확인하고, CDC 기반 변경분 동기화가 정상적으로 수행되는지 검토하였다. 마지막으로 실행 결과와 로그를 확인하여 각 단계의 처리 상태를 검증하였다.

2. Functional Validation Results

기능 검증은 제안 모델의 핵심 단계별로 수행하였다. 검증 대상은 테이블 변환 및 데이터 적재, 프로시저, 함수, 트리거 변환, CDC 기반 변경분 반영, 실행 로그 기록으로 구성하였다. 각 항목은 변환 결과가 타깃 DBMS에 적용 가능한지, 실행 과정에서 필요한 보정이 이루어졌는지, 그리고 결과를 추적할 수 있는 로그가 남는지를 중심으로 확인하였다.

본 연구는 실제 프로덕션 부하를 재현한 성능 벤치마크가 아니라, LLM 기반 SQL 변환과 후처리, 데이터 적재,

CDC 동기화 및 체크포인트 관리가 하나의 마이그레이션 흐름으로 수행되는지를 확인하는 기능 검증으로 진행하였다. 기능 검증 결과의 객관성을 보완하기 위해 객체 적용 성공률, 소스·타깃 행 수 일치 여부, CDC 이벤트 반영률, 재시도 및 수동 보정 횟수, 처리시간과 웹 애플리케이션 프로세스의 자원 사용량을 함께 측정하였다. Table 3에서 객체 적용 성공률은 변환을 시도한 객체 중 생성된 SQL이 타깃 DBMS에 오류 없이 적용된 객체의 비율을 의미한다.

각 검증 항목은 다음 기준에 따라 판정하였다. 테이블 마이그레이션은 선택된 모든 테이블이 타깃 DBMS에 생성되고 소스와 타깃의 행 수가 일치한 경우 Pass로 판정하였다. 프로시저, 함수 및 트리거 변환은 생성된 객체 정의가 타깃 DBMS에 실행 오류 없이 적용된 경우 Pass로 판정하였으며, SQL 정규화는 모든 LLM 응답에서 실행 가능한 SQL이 추출·정규화된 경우 Pass로 판정하였다. 수동 보정은 생성된 SQL에 대한 사용자의 직접 수정이 발생하지 않은 경우, CDC 동기화는 검증용으로 생성된 변경 이벤트가 누락 없이 모두 반영된 경우, 체크포인트 및 실행 로그는 유효한 체크포인트와 실행 기록이 생성되고 미해결 오류 로그가 없는 경우 각각 Pass로 판정하였다.

Table 3. Functional Validation Results

Verification Item	Observed Result
Table Migration	4/4 tables; 2,980 rows; row-count match = 4/4; Pass
Procedural Objects	Procedures = 2/2, retries = 0; functions = 2/2, retries = 0; triggers = 2/2; Pass
SQL Normalization	10/10 SQL statements normalized; recorded corrections = 0; Pass
Manual Correction	Manual corrections = 0; Pass
CDC Synchronization	162/162 events applied; I/U/D = 100/42/20; Pass
Checkpoint and Logging	Checkpoint records = 1; log records = 481; error logs = 0; Pass
Note: Pass for procedural objects indicates successful application to the target DBMS without SQL execution errors; it does not represent full semantic equivalence with the source objects.	

보조 성능 측정 결과, 전체 처리시간은 449.328초였으며, 이 중 LLM 응답시간은 446.747초, 타깃 DBMS 반영 시간은 2.220초로 측정되었다. 웹 애플리케이션 프로세스 기준 CPU 사용률은 0.7%, 메모리 변화량은 1.30 MB였다.

V. Conclusions

본 연구는 이기종 DBMS 마이그레이션에서 발생하는 스키마 차이, SQL 문법 차이, 절차적 객체 변환, 초기 적재 이후의 데이터 변경 반영 문제를 완화하기 위해 T0 기반 초기 적재, LLM 기반 SQL 변환 및 SQL 정규화, CDC 기반 변경분 동기화를 결합한 마이그레이션 모델을 제안하였다. 제안 모델은 소스 DBMS의 메타데이터와 객체 정의를 활용하여 DBMS 조합별 변환 프롬프트를 구성하고, LLM 응답에서 실행 가능한 SQL을 추출·정규화·보정한 뒤 타깃 DBMS에 적용하도록 설계되었다.

기능 검증 결과, 제한된 MariaDB-PostgreSQL 환경에서 4개 테이블의 총 2,980건에 대해 소스와 타깃의 행 수가 일치하였고, 프로시저 2개, 함수 2개, 트리거 2개와 CDC 이벤트 162건이 오류 없이 적용되었다. 이는 제안 모델이 단순한 SQL 변환 기능에 그치지 않고, 초기 적재, 절차적 객체 변환, 변경 데이터 동기화 및 실행 이력 추적을 연계하는 통합형 마이그레이션 구조로 동작할 수 있음을 보여준다. 또한 프로시저와 함수 변환의 재시도는 0회였고 수동 SQL 보정도 발생하지 않아, 제한된 검증 범위에서 LLM 생성 결과를 SQL 추출·정규화 및 후처리 절차와 연결한 방식의 적용 가능성을 확인하였다.

기존 연구 및 도구가 LLM 기반 SQL 생성·변환[6][7][8]과 CDC 기반 변경 데이터 처리[9][10]를 각각 다루어 온 것과 비교하면, 본 연구의 학술적 기여는 LLM 기반 SQL 변환을 SQL 후처리와 CDC 동기화 절차 안에 배치하여 이기종 DBMS 마이그레이션 흐름으로 구조화했다는 점에 있다. 기술적 측면에서는 DBMS 조합별 프롬프트 구성, 실행 가능한 SQL 추출, 정규화 및 보정, 체크포인트 기반 CDC 반영, 실행 로그 기록을 연계함으로써 변환 결과의 적용 과정을 추적 가능한 절차로 제시하였다. 실무적으로는 SQL 변환 결과, CDC 반영 상태 및 실행 이력을 동일한 처리 흐름에서 확인할 수 있어, 이기종 DBMS 이관 과정의 수동 확인과 추적 부담을 줄이는 데 활용될 수 있다.

다만 본 연구의 실험은 실제 프로덕션 부하를 재현한 성능 평가가 아니라 제한된 환경에서 수행한 기능 검증에 해당한다. 따라서 대규모 실서비스 데이터 처리 시간, 장시간 CDC 운영 안정성, CDC 지연 시간, 동시 작업 처리 성능, 장애 복구 시간, 체크포인트 복구 성능을 본 연구 결과만으로 일반화하기에는 한계가 있다. 또한 AI 변환 품질은 단일 LLM 모델과 제한된 DBMS 조합에서 확인되었으므로, 다른 LLM 모델이나 DBMS 조합에서도 동일한 수준의 변환 결과가 나타난다고 단정하기는 어렵다.

향후 연구에서는 실제 운영 환경에 가까운 데이터 규모와 장애 조건에서 제안 모델을 검증하고, 다양한 DBMS 조합 및 여러 LLM 모델을 대상으로 정량적 비교 실험을 수행할 필요가 있다. 구체적으로 MariaDB-PostgreSQL 외에 PostgreSQL-MariaDB, MySQL-PostgreSQL 등 서로 다른 DBMS 조합을 실험 대상으로 확장하고, 동일한 스키마와 절차적 객체 집합에 대해 LLM 모델별 SQL 변환 성공률, 후처리 필요 빈도, 재변환 횟수, 전체 처리 시간, LLM 응답 시간, 타깃 DBMS 반영 시간, CDC 지연 시간, 체크포인트 복구 시간을 비교하는 방식으로 평가 계획을 구성할 수 있다. 또한 테이블, 프로시저, 함수, 트리거와 같이 객체 유형별 결과를 분리하여 분석하면 DBMS 조합과 LLM 모델에 따라 변환이 어려운 객체 유형을 보다 구체적으로 확인할 수 있다. 이를 통해 제안 모델의 실무 적용 가능성과 운영 안정성을 보다 명확하게 평가할 수 있을 것이다.

REFERENCES

- [1] Nayan B. Ruparelia, Cloud Computing, revised and updated ed., Cambridge, MA, USA: MIT Press, Aug. 2023.
- [2] Amazon Web Services, "What is AWS Database Migration Service?," AWS Database Migration Service User Guide, Amazon Web Services Documentation, accessed Jul. 7, 2026.
- [3] PostgreSQL Global Development Group, "Chapter 41. PL/pgSQL - SQL Procedural Language," PostgreSQL 18 Documentation, PostgreSQL Global Development Group, accessed Jul. 7, 2026.
- [4] Maria J. Ortin, Jose R. Hoyos, and Jesus Garcia-Molina, "A Model-Driven Approach to Database Migration with a Unified Data Model," arXiv:2604.22415, Apr. 2026. DOI: 10.48550/arXiv.2604.22415.
- [5] Ruoxi Sun, Sercan O. Arik, Alexandre Muzio, Lesly Miculicich, Satya Kesav Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister, "SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL," Transactions on Machine Learning Research, 2024.
- [6] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo, "A Survey of Text-to-SQL in the Era of LLMs: Where are We, and Where are We Going?," IEEE Transactions on Knowledge and Data Engineering, vol. 37, no. 10, pp. 5735-5754, Oct. 2025, DOI: 10.1109/TKDE.2025.3592032.
- [7] Wei Zhou, Yuyang Gao, Xuanhe Zhou, and Guoliang Li, "Cracking SQL Barriers: An LLM-based Dialect Translation System," Proceedings of the ACM on Management of Data, Vol. 3, No.

- 3, Article 141, pp. 1-26, June 2025. DOI: 10.1145/3725278.
- [8] Mohammadreza Pourreza and Davood Rafiei, "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction," *Advances in Neural Information Processing Systems* 36, 2023, DOI: 10.52202/075280-1577.
- [9] Debezium Community, "Reference Documentation," Debezium Documentation, Debezium, accessed Jul. 7, 2026.
- [10] Dennis Butterstein, Daniel Martin, Knut Stolze, Felix Beier, Jia Zhong, and Lingyun Wang, "Replication at the Speed of Change: A Fast, Scalable Replication Solution for Near Real-Time HTAP Processing," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3245-3257, 2020, DOI: 10.14778/3415478.3415548.
- [11] MariaDB Foundation, "CREATE TRIGGER," MariaDB Server Documentation, MariaDB, accessed Jul. 7, 2026.
- [12] PostgreSQL Global Development Group, "CREATE TRIGGER," PostgreSQL 18 Documentation, PostgreSQL Global Development Group, accessed Jul. 7, 2026.
- [13] MariaDB Foundation, "SHOW MASTER STATUS," MariaDB Server Documentation, MariaDB, accessed Jul. 7, 2026.
- [14] MariaDB Foundation, "START TRANSACTION," MariaDB Server Documentation, MariaDB, accessed Jul. 7, 2026.
- [15] Ollama, "Introduction," Ollama API Reference, Ollama, accessed Jul. 7, 2026.

Authors



Dong-Bin Lim received the B.S. degree in Computer Engineering from Youngdong University, Korea, in 2008. He is currently the head of the Technical Support Team at JIT Inc., where he works as a System

Engineer. M.E. Candidate, Kongju National University, since 2025. His research interests include AI, database systems, and distributed systems.



Eun-jin Oh received the B.S. degree in Computer Science from Korea National Open University, Korea, in 2007, and the M.S. degree in Computer Engineering from Kongju National University, Korea, in 2002.

She completed her Ph.D. coursework in Computer Engineering at Kongju National University in 2007. She has been a part-time lecturer in the Department of Software at Kongju National University, Korea, since March 2004. Her research interests include software engineering, big data analytics, artificial intelligence, and generative AI.



Jae-Woong Kim received the B.S. degree in Electronic Computer Science from Chung-Ang University, Korea, in 1983, the M.S. degree in Computer Engineering from Chung-Ang University, Korea, in 1988, and the Ph.D.

degree in Computer Engineering from Daejeon University, Korea, in 2000. Dr. Kim joined the faculty of the Department of Software at Kongju National University, Korea, in February 1992. He is currently a Professor in the Department of Software at Kongju National University. He is interested in software engineering, big data, and multimedia engineering.