Design and development of a session scoped self adaptive database framework for logical object oriented data management

1. Introduction and motivation

Modern software systems increasingly operate on data whose structure is unknown at design time and evolves continuously during execution. This course project proposes a self-adaptive database framework that abstracts database selection, schema handling, and transactional coordination away from the user. The system exposes a logical, object-oriented interface through which users interact with data, while autonomously managing schema inference, adaptive storage placement, cross-database queries, and ACID-style transactions across heterogeneous backends.

Structure of document:

- **Part 1: Problem statement**
  Core challenges in schema-less data management and autonomous storage.
- **Part 2: Proposed solution**
  Hybrid architecture involving SQL and MongoDB with session-scoped isolation.
- **Part 3: Implementation detail**
  Data ingestion, API design, and the logical dashboard.
- **Part 4: Evaluation & assignments**
  Course project phases and performance characterization.

Instructor: Dr. Yogesh K. Meena

January 16, 2026 • Semester II (2025 - 2026) • CS 432 – Databases (Course Project)

2. Problem statement

This course project addresses the following core problems:

- How can users store and query data without having to choose a database or define schemas upfront?
- How can a system observe evolving data and autonomously decide how and where it should be stored?
- How can cross-database joins and ACID-style transactions be supported transparently?
- How can users be given a stable, intuitive, and logical view of data, independent of physical storage and schema evolution?

3. Proposed solution overview

The proposed system is a database framework that provides a single logical interface for data ingestion and querying. Users never interact with tables, collections, schemas, or backend-specific query languages. This database framework operates above two backend databases:

- A relational database (SQL) for structured and stable data
- A document-oriented database (MongoDB) for evolving or semi-structured data

Key Characteristics

- Object-oriented logical interaction
- Session-scoped logical vocabularies
- Temporary buffering and delayed storage commitment
- Autonomous backend selection and schema evolution
- Framework manages joins and transactions
- Logical dashboard and JSON-based outputs

4. Session-scoped Logical Vocabulary

Session Concept

A session represents a single application context, user workflow, or experiment run. Each session is treated as an independent, logically isolated universe, with complete isolation from other sessions. Sessions do not share:

- Logical class names
- Field names
- Schema evolution history
- Storage placement decisions

Field and Class Naming Semantics

Logical class names and field names are defined at the beginning of a session. Once defined, they remain fixed for the lifetime of that session. Different sessions may use entirely different names and object structures. This ensures:

- Semantic stability within a session
- Flexibility across users and applications
- Deterministic joins and queries
- A stable dashboard experience

Importantly, only names and semantic identity are fixed. Types, optionality, schema realisation, and storage placement remain adaptive.

5. Data Ingestion and Adaptive Storage

Streaming Ingestion Model

Incoming data is first stored in a temporary staging buffer rather than being committed to a backend database immediately. Data enters the system as:

- A stream of JSON objects, or
- Python class instances converted to JSON

Observation and Analysis

While data resides in the buffer, the system observes:

- Field presence frequency
- Type variation
- Update patterns
- Object consistency over time

This observation phase allows the system to delay irreversible storage decisions.

Autonomous Backend Placement

Based on observed behaviour:

- Stable, structured fields are stored in the SQL backend
- Flexible or evolving fields are stored in MongoDB

As more data arrives, the system may:

- Promote fields to SQL
- Keep fields in MongoDB
- Migrate data transparently

All placement decisions are system-driven and invisible to the user.

6. Query Model and Transactions

Logical Query Interface

The system exposes a logical instruction set for querying data. Queries:

- Refer only to logical classes and fields
- Do not specify tables, collections, or joins
- Are declarative in nature

The framework parses queries, generates backend-specific subqueries, executes them, performs joins at the logic level, and returns results as JSON.

Cross-database Transactions

Each user operation is treated as a logical transaction. Users do not manage transactions explicitly. The framework:

- Coordinates writes across SQL and MongoDB
- Ensures atomic visibility of results
- Handles partial failures internally

7. API Design and Python Interface

Python Package and Decorators (Optional)

The system is exposed as a Python package. Users define logical objects using decorators:

```
@entity(session="session1")
class User:
    id: int
    age: int | None
    country: str | None
```

Decorators:

- Extract class and field names
- Enforce field consistency
- Convert objects into JSON

Supported Operations

- **PUT**: Insert or update objects using JSON or Python objects.
- Fields passed as `None` do not overwrite existing values.
- **GET**: Query data using the logical instruction set.
- **DELETE**: Remove objects logically, with physical deletion handled by the framework.

All outputs are returned as JSON key-value objects.

8. Logical Dashboard

The dashboard presents data exactly as the user understands it, independent of backend storage.
It shows:

- Object instances grouped by class
- Logical fields and values
- Query results in JSON form

It does **not** expose:

- SQL tables

- MongoDB collections
- Schema evolution events
- Placement decisions

9. Evaluation Plan

The system will be evaluated along three dimensions:

- **Correctness and Adaptivity**: schema inference accuracy, backend placement decisions, stability under evolving data
- **Transactional Behaviour**: atomicity across databases, correct handling of partial failures, consistent query results
- **Performance Characterisation**: buffering overhead, coordination cost, and comparison with direct database access

10. Project Phases

- **Assignment 1**: Ingestion, session management, and logical object modelling
- **Assignment 2**: Basic dashboard, schema inference and adaptive backend placement
- **Assignment 3**: ACID validation for query processing, joins, and transaction coordination
- **Assignment 4**: Dashboard development, performance testing, and final packaging

Note: Separate documents will be shared for each assignment and evaluation.

11. Expected Outcomes

- A working self-adaptive database framework
- Session-scoped logical abstraction
- Transparent use of SQL and MongoDB
- Cross-database querying and transactions
- A clean, logical dashboard
- A documented Python package interface