Database Internals + Application Design
DSCI 551 Course Project, Spring'26

Exploring and Applying Modern Database Systems Beyond MySQL and MongoDB

Project Overview

This project has two integrated goals:

1. **Analyze the internal architecture** of a modern database system (excluding MySQL and MongoDB), focusing on storage, indexing, query execution, and (if applicable) distribution.
2. **Design and implement a small application** that uses the chosen database in a way that reflects its internal design choices.

The project is divided into **three phases** to support steady progress and feedback.

Group Size Options

Students may work in **groups of 1, 2, or 3**. Expectations scale with group size as described below.

Allowed Database Systems (Open Source)

Relational / NewSQL

- PostgreSQL
- CockroachDB
- TiDB
- YugabyteDB
- VoltDB
- OceanBase (open-source edition)

Columnar / Analytical

- DuckDB
- ClickHouse
- Apache Doris
- Apache Druid
- Apache Pinot

- Greenplum

Key–Value / Storage Engines

- RocksDB
- LevelDB
- FoundationDB
- BadgerDB
- Pebble

Distributed / NoSQL

- Apache Cassandra
- ScyllaDB
- Apache HBase
- Apache Kudu
- Vitess

Embedded / Specialized

- SQLite (requires internal analysis such as B-tree or Virtual Database Engine)
- LMDB (Lightning Memory-Mapped Database)
- Berkeley DB
- H2 (https://www.h2database.com/html/main.html)

Group Size

Students may work in groups of 1, 2, or 3. Project scope and depth are expected to scale with group size.

Phase 1: Project Proposal (10%)

Due: Week 6 (2/20, Friday)

Submit a 1–2 page proposal that includes:

- Chosen database system and brief motivation
- Planned internal focus areas (storage, indexing, execution, distribution, etc) and briefly explain why the focuses
- Preliminary application idea
- Team information and responsibilities
- 2–3 initial references

- **Group-Size Expectations**
  - **1 person:** One core internal focus area identified

- **2 people:** Two internal focus areas clearly outlined
- **3 people:** Broad internal coverage proposed, including distribution if applicable

Phase 2: Midterm Report & Implementation Checkpoint (10%)

Due: Week 9 (3/13, Friday)

Written midterm report (3–4 pages) should include:

- Database system overview and architecture
- Details on the focus areas (e.g., how it works, and how it differs from other DB systems)
- Preliminary application design
- Preliminary discussions on the mappings between DB internals and application behaviors (see examples at the end).
- Progress update and challenges

Implementation checkpoint requirements:

- Database installed and configured
- Initial schema or data model implemented
- At least one working application feature or query

Phase 3: Demo, Implementation, & Final Report (80%)

Application Demo, 4/20 (all sections), 4/22 (MW section only), and 4/27 (reserved for any leftover groups) (10%)

- 5–10 minute live demo, conducted online over Zoom, in class meeting times
- Focus on database usage and application of system focus areas (UI polish not graded)

Implementation (50%)

- For each major application operation, students must explicitly explain:
  - What the application does
  - What the database does internally
  - Why that internal behavior matters (i.e., mapping between internals & applications)
- Examples of acceptable mappings (see more details at the end of this guideline):
  - Query → index traversal vs full scan
  - Insert workload → B-tree vs LSM (log-structured merge tree) behavior
  - Aggregation → columnar vs row-based execution
  - Range scan → leaf-page or partition traversal

- Note that your application should be fully developed by the demo time. The mapping should be completed and documented in the final report.
- The submission must include:
  - Application source code
  - Database schema / setup scripts
  - Instructions to run the application
  - A small dataset (real or synthetic)
- The application must be **runnable by the instructor or TA**.

Final Report, 5/8, Friday (20%)

- 1 person: 6–8 pages
- 2 people: 8–10 pages
- 3 people: 10–12 pages

The final report must include the following sections:

- Introduction & Motivation
- Database System Overview
- Internal Architecture (storage, indexing, execution, distribution if applicable)
- Application Design
- Mapping Internals to Application Behavior
- Comparison with MySQL and MongoDB
- Limitations & Lessons Learned
- Link to Google drive where you upload all your project codes and documentation

Examples of mapping between application behavior and database internals

Example 1: SQLite — Task Management App

Application behavior

The application frequently queries tasks by due_date and status.

```
SELECT * FROM tasks
WHERE status = 'open'
ORDER BY due_date;
```

Database internals mapping

- SQLite stores table data in a **B-tree organized by rowid**
- Creating an index on (status, due_date) builds a **separate B-tree**

- The query planner selects an **index scan** instead of a full table scan (show & explain high-level execution plan).
- SQLite's **VDBE bytecode** executes an ordered index traversal (show & explain bytecode generated)

Note: SQLite can be chosen only by 1–2 person groups.

Example 2: PostgreSQL — Content Management App

Application behavior

The application retrieves published articles by category.

```
SELECT title, published_at
FROM articles
WHERE category = 'database'
ORDER BY published_at DESC;
```

Database internals mapping

- PostgreSQL uses **heap storage + secondary B-tree indexes**
- Index on (category, published_at)
- Query planner chooses an **index scan**
- **MVCC** allows reads without blocking writers (show & explain how MVCC is used to handle concurrent read & write)

Example 3: DuckDB — Analytical Dashboard

Application behavior

The application runs aggregations over large CSV datasets.

```
SELECT region, SUM(sales)
FROM sales_data
GROUP BY region;
```

Database internals mapping

- DuckDB uses **columnar storage**
- Query execution is **vectorized**
- Only the region and sales columns are scanned
- Execution occurs in **cache-friendly batches** (e.g., 1024 values of a specific column such as region & sales)

Example 4: RocksDB — Metadata Service

Application behavior

The application performs frequent writes and point lookups.

```
PUT(user_id → profile)
GET(user_id)
```

Database internals mapping

- RocksDB uses an Log-Structured Merge-tree (**LSM-tree**)
- Writes go to **memtables**
- Reads may consult multiple SSTables
- Compaction reorganizes data in the background

Example 5: Cassandra — Event Logging App

Application behavior

The application queries recent events for a given user.

```
SELECT *
FROM events
WHERE user_id = ?
ORDER BY timestamp DESC;
```

Database internals mapping

- Cassandra partitions data by **partition key**
- Clustering key defines **on-disk sort order**
- Query maps directly to a **single partition scan**