CS 411 Stage 4 Project Report
Team: fa25-cs411-team126-ysql

Deployed Webpage: https://paperscope-frontend-ahrq6r24nq-uc.a.run.app/review-papers
Recorded Video: https://mediaspace.illinois.edu/media/t/1_8c6yk39v
Constraints, triggers, procedures and transactions are inside Database folder in our Github repo.

1. Changes in the Direction of the Project

The direction of the project changed in several ways as we moved from the Stage 1 proposal to the final implementation. In the proposal, the application focused mainly on basic paper management, authorship, venue assignments, and reviews. As development progressed, we expanded the scope to introduce AI-assisted draft generation, batch paper insertion workflows, and stronger data validation mechanisms.

The biggest directional change was the introduction of AI-generated draft papers, which did not exist in the original proposal. This required structural changes to the schema-specifically, adding ai_generated and source_paper_id to the Papers table and creating a self-referencing FK to allow provenance tracking:

```
ALTER TABLE Papers
ADD COLUMN ai_generated TINYINT(1) DEFAULT 0,
ADD COLUMN source_paper_id VARCHAR(50),
ADD CONSTRAINT fk_papers_source
FOREIGN KEY (source_paper_id) REFERENCES Papers(paper_id);
```

Another direction shift involved building an advanced batch paper creation workflow to support real-world academic needs (e.g., entering dozens of papers at once). This feature became one of our main advanced transactions and was not part of the initial design.

We also expanded our use of constraints and triggers to enforce business rules inside the database instead of relying solely on application logic. This included enforcing valid review comments, preventing authors from reviewing their own papers, and ensuring AI drafts follow specific promotion rules.

2. Usefulness: What the Application Achieved and What It Fell Short On

We believe the application succeeded in providing a complete research paper management platform with meaningful interactions between authors, papers, venues, datasets, and reviews. The system supports full CRUD operations, keyword search, author analytics, and safe deletion of papers. The introduction of AI-assisted related paper recommendations further increased the usefulness by helping authors explore similar research directions.

The batch paper creation workflow significantly improves usability for scenarios like research assistants entering many papers or importing large datasets. The following backend code snippet shows how the system prevents duplicate submissions using row-level locking:

```
SELECT p.venue_id, p.paper_title
FROM Papers p
WHERE (p.venue_id, p.paper_title) IN ((?, ?), (?, ?), ...)
FOR UPDATE;
```

However, there were some limitations. While we originally planned to include a reviewer anonymity feature and topic classification for papers, both were removed due to time constraints. Additionally, while our keyword search works effectively, we were not able to implement full-text search or ranking, which would have made the search far more powerful. Similarly, AI recommendations currently work on demand rather than being cached or precomputed.

3. Changes to Schema or Data Source

We did not change the source of the data; all data continues to live inside our MySQL instance hosted on GCP. However, the schema changed significantly to accommodate new features and improve data integrity.

New columns added to the Papers table:

```
ai_generated TINYINT(1),
source_paper_id VARCHAR(50)
```

We also added five CHECK constraints to enforce data quality, such as email format validation:

`CHECK (email LIKE '%@%.%')`
and review comment enforcement:
`CHECK (CHAR_LENGTH(TRIM(comment)) > 0)`
Triggers were also introduced to enforce relationship-based rules, such as preventing self-reviews, which had a direct impact on how data behaves within the schema.

4. Changes to the ER Diagram and Table Implementations

Compared to the original ER diagram, the final design includes additional relationships and constraints that reflect new workflow requirements:

A self-referencing relationship in the Papers table (source_paper_id ,Üí paper_id) to support AI draft lineage.

New CHECK constraints across the schema.
The addition of triggers on Papers and Reviews to validate transitions and prevent invalid actions.

Difference example: originally, Papers had no connection to itself. In the final design, AI drafts are linked like this:
Paper P123 (ai_generated=1) ,Üí source_paper_id = P045

This change makes the schema more expressive and better aligned with the expanded functionality.

The final design is more suitable because it consolidates business logic directly into the database. For example, the trigger enforcing AI draft promotion rules ensures consistent behavior regardless of how the data is submitted:

```
IF NEW.status = 'Under Review' AND NEW.ai_generated = 1 THEN
  IF NEW.pdf_url IS NULL OR CHAR_LENGTH(NEW.abstract) < 50 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'AI drafts require a PDF and long abstract';
  END IF;
END IF;
```

This version of the design is more robust and prevents invalid or incomplete records from entering the system.

5. Functionalities Added or Removed

Added
AI Draft Generation: Generates draft papers from Gemini recommendations and stores them safely using a stored procedure.
Batch Paper Creation: Allows inserting multiple papers and authors at once using an advanced transaction.
Per-venue summary reporting: Provided by a GROUP BY query after batch inserts.

Collapsible multi-paper UI on the frontend: Improves usability when entering many papers.

Removed
Topic classification for papers (removed due to limited time).
Reviewer anonymity (removed because it introduced complex schema changes).

The added features aligned better with real-world academic workflows, while the removed features either required too much time or were less critical for meeting course requirements.

6. How the Advanced Database Programs Complement the Application

The advanced database programs act as the backbone of the application by enforcing business logic, ensuring safety of operations, and providing analytical capabilities.

Stored procedures support insights and portfolio features with queries such as:

```
SELECT COUNT(DISTINCT p.paper_id) AS total_papers,
COALESCE(AVG(r.rating),0) AS avg_rating
FROM Authorship a
LEFT JOIN Papers p ON a.paper_id = p.paper_id
LEFT JOIN Reviews r ON p.paper_id = r.paper_id
WHERE a.user_id = p_user_id;
```

Triggers prevent invalid states, such as self-reviews or incomplete AI draft promotions. Transactions guarantee safety in write-heavy workflows. The batch-insert transaction ensures atomicity:

If any step fails, the entire batch is rolled back. All these programs reduce errors, simplify backend logic, and ensure a consistent user experience.

7. Technical Challenges Encountered

Challenge 1 Handling Collation Mismatches Across Tables

We encountered a persistent issue where several JOIN queries failed with "Illegal mix of collations" errors. This happened because our tables were created at different times using different default collations (utf8mb4_unicode_ci and utf8mb4_0900_ai_ci). These errors only appeared when writing multi-table stored procedures and during batch validation queries. The solution was to explicitly enforce a consistent collation in problematic queries:... ON u.user_id = a.user_id COLLATE utf8mb4_unicode_ci

Future teams should standardize collation during schema creation to avoid debugging complexity later.

Challenge 2 - Designing the Batch Insert Transaction

The transaction needed to validate foreign keys in bulk and prevent duplicate titles per venue. The challenge was to prevent race conditions. The solution was row-level locking:

Challenge 3 - Frontend State Management

The batch-create UI initially became unmanageable because each new paper added multiple uncontrolled form elements. The solution was to create collapsible panels and restrict interaction to one expanded section at a time. This significantly improved usability.

8. Other Differences Compared to the Original Proposal

Other differences include stronger constraints, broader use of triggers, and a shift toward enforcing rules inside the database rather than in the application layer. We also added success modals, improved error handling, and more structured insights pages. These enhancements were not explicitly planned in Stage 1 but improved overall usability and reliability.

9. Future Work Beyond the Interface

Future improvements include:
Full-text search to support ranked keyword search.
Caching for AI recommendations to reduce API cost and latencies.
Paper versioning, so updates to papers are tracked historically.
Notifications or email triggers when reviews or paper status changes occur.
These enhancements would make the system more scalable and closer to a production-ready research management tool.

10. Final Division of Labor and Teamwork Evaluation

Our work was divided across three major areas: database design and implementation, backend API development, and frontend interface development. Although each of us had primary responsibilities, we made a conscious effort to involve all four members in the database portion of the project because this course emphasizes database systems. This ensured that every team member contributed meaningfully to constraints, triggers, stored procedures, and transactions.

From a database perspective, Bharath, Chitsimran, Hardik, and Satwik all worked on different aspects. We collaborated on designing the schema extensions required for Stage 4, including the new AI-related fields and the self-referencing foreign key. We also jointly refined the CHECK constraints, reviewed one another's stored procedures, and helped debug triggers. When implementing the batch-insert transaction and safe delete procedure, everyone participated in code review and SQL testing. This shared responsibility allowed each member to gain experience with advanced database features and ensured redundancy in understanding the system.

For backend development, Bharath and Hardik implemented the Express API, validation logic, and transaction-controlled endpoints. This included the single-paper creation endpoint, the batch-insert workflow, and the integration points for calling stored procedures. They ensured that all database programs were triggered through the API rather than executed manually. They also handled error propagation from triggers and constraints so that the UI displayed meaningful messages to users. Their coordination was important because backend routes had to follow the database rules exactly for the system to remain consistent.

For frontend development, Satwik and Chitsimran built the React interface and connected it to the backend. They implemented the batch-create page, author insights view, portfolio display, and the paper details modal. They also ensured that all CRUD operations for Papers and Reviews were available through the UI, making each database function accessible as required by Stage 4. The collapsible card system on the batch-create page, the keyword search interface, and the error/success banners all came from their work. They also tested frontend workflows with the backend transaction logic, which helped uncover several SQL constraint violations during development.

As a team, we managed the project smoothly and with consistent communication. We met frequently to align API contracts, confirm data formats, and discuss database logic before implementation. This prevented mismatches between the UI, backend, and database layers. When bugs appeared especially collation issues, trigger misfires, or batch-validation failures we worked through them together rather than leaving them to whoever originally wrote the component. The final application reflects coordinated work rather than isolated contributions, and each member had visibility into the full stack. This collaborative structure made integration straightforward and helped the project progress without major conflicts.