Project Report

Department	:	**C**omputer **S**cience and **E**ngineering
Course	:	Database Systems Laboratory (CSE 3110)
Project Title	:	EventHive
Submission Date	:	26 October, 2025

Submitted To, [1] Md. Mehrab Hossain Opi [2] Waliul Islam Sumon Department of Computer Science and Engineering, Khulna University of Engineering and Technology	Submitted By, Md. **Shifat** Hasan Roll: 2107**67** Lab-Group: **B1** Year: **3rd** Term: **1st**

Objectives

- To design a normalized event management database that balances analytical and transactional workloads.
- To enable natural language interaction with PostgreSQL through an AI-assisted application layer.
- To safeguard data integrity by enforcing relational constraints, validation rules, and indexed access paths.

Introduction

EventHive is a laboratory-scale platform that merges an event management data model with AI-driven query generation to simplify analytical exploration. The prototype showcases how PostgreSQL's relational guarantees can coexist with the flexibility of on-demand SQL authoring through Google Gemini Flash. Users receive immediate visibility into generated statements, execution notices, and tabular results within a focused single-page interface. By combining a modern web stack with a teaching schema, the project highlights best practices in database systems coursework while remaining accessible for future iteration.

Details

System Overview: The solution is organized as a Bun-powered Express backend backed by PostgreSQL and exposed through a static Tailwind interface. The server hydrates prompts with schema metadata, brokers secure execution through the `pg` (PostgreSQL) and Neon clients, and streams structured responses to the browser. Frontend modules render SQL, system notices, and paginated results without reloads, providing students with an instrumentation-rich view of every round trip.

Database Architecture: The schema prioritizes clarity and referential integrity across five core relations—`users`, `events`, `registrations`, `event_feedback`, and `event_tags`. Each table uses surrogate primary keys, cascading foreign keys, and check constraints that encode domain rules such as attendance capacity, rating ranges, and enumerated status codes. Composite indexes [for example, `events(organizer_id, event_date)` and `registrations(event_id, status)`] mirror the query fragments that the AI frequently generates, ensuring predictable performance for trend analysis and transactional lookups. Figure 1 illustrates the logical schema that underpins these integrity guarantees.

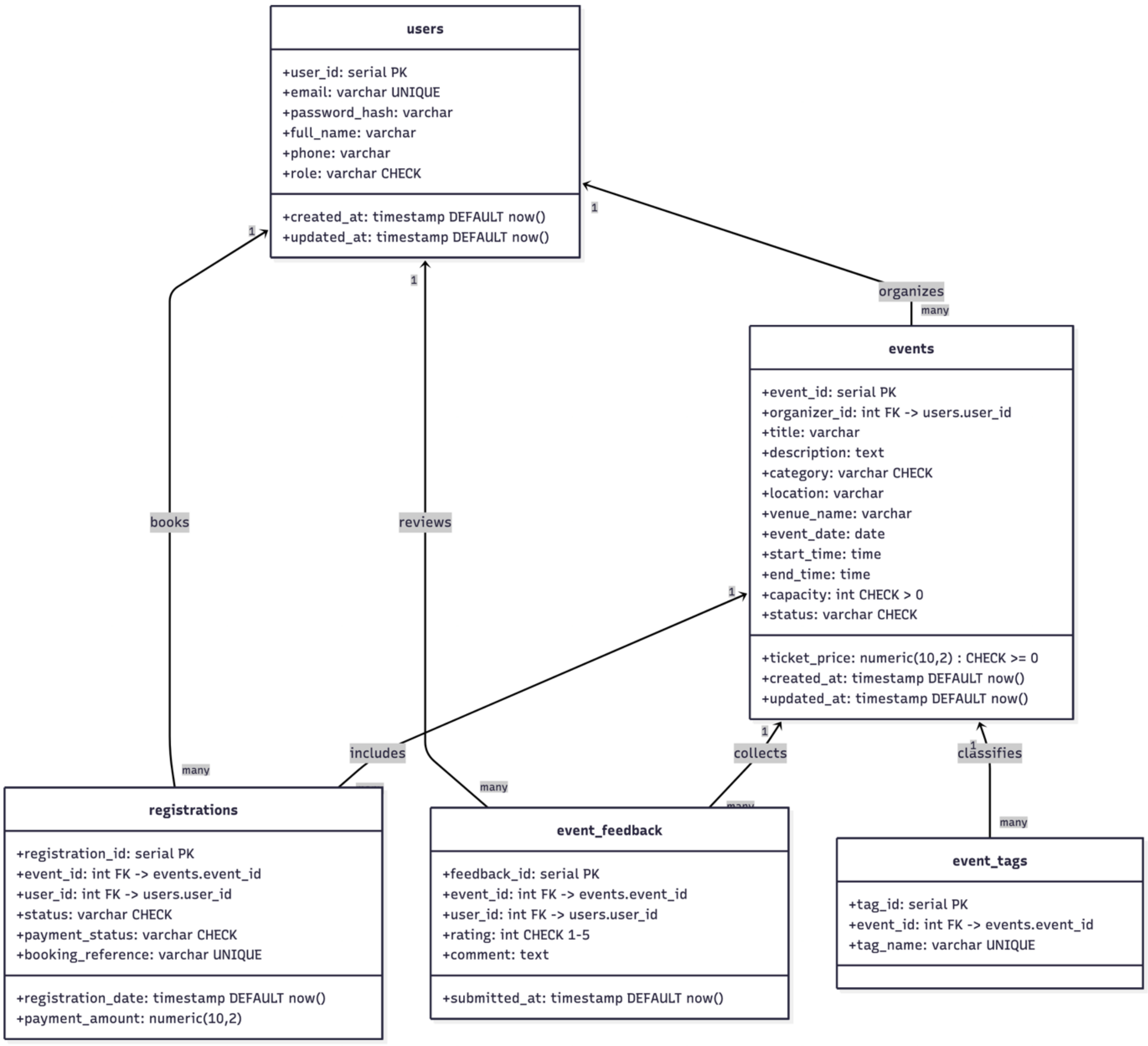
users
+user_id: serial PK
+email: varchar UNIQUE
+password_hash: varchar
+full_name: varchar
+phone: varchar
+role: varchar CHECK
+created_at: timestamp DEFAULT now()
+updated_at: timestamp DEFAULT now()
organizes
1
many
events
+event_id: serial PK
+organizer_id: int FK -> users.user_id
+title: varchar
+description: text
+category: varchar CHECK
+location: varchar
+venue_name: varchar
+event_date: date
+start_time: time
+end_time: time
+capacity: int CHECK > 0
+status: varchar CHECK
+ticket_price: numeric(10,2) : CHECK >= 0
+created_at: timestamp DEFAULT now()
+updated_at: timestamp DEFAULT now()
books
reviews
includes
collects
classifies
1
many
registrations
+registration_id: serial PK
+event_id: int FK -> events.event_id
+user_id: int FK -> users.user_id
+status: varchar CHECK
+payment_status: varchar CHECK
+booking_reference: varchar UNIQUE
+registration_date: timestamp DEFAULT now()
+payment_amount: numeric(10,2)
event_feedback
+feedback_id: serial PK
+event_id: int FK -> events.event_id
+user_id: int FK -> users.user_id
+rating: int CHECK 1-5
+comment: text
+submitted_at: timestamp DEFAULT now()
event_tags
+tag_id: serial PK
+event_id: int FK -> events.event_id
+tag_name: varchar UNIQUE

Figure 1: Physical Schema or Class Diagram

AI-Orchestrated Query Pipeline: The query route composes structured prompts with table summaries from `schema-context.js`, requests JSON-formatted plans from Gemini Flash, and executes each returned statement sequentially. A validation layer rejects destructive keywords, sequences execution to respect foreign key dependencies, and merges PostgreSQL `NOTICE` messages with AI guidance so learners can inspect both logical reasoning and runtime diagnostics. This orchestration enables dynamic PL/pgSQL generation, supporting ad hoc analytics such as capacity utilization, revenue projections, and attendee segmentation without prebuilt stored routines.

Performance, Integrity, and Security Controls: The system enforces least-privilege database connectivity through environment-scoped credentials, while the backend wraps execution in timing instrumentation to surface long-running patterns. Index coverage, uniqueness constraints, and cascading options prevent duplication and orphaned data during insertions and deletions. Tailored status enumerations and price validations reduce malformed records, and booking references give auditors deterministic identifiers for reconciliation. Together, these measures exemplify laboratory expectations for resilient relational workloads, and the cardinalities summarized in Figure 2 justify the chosen constraint strategy.

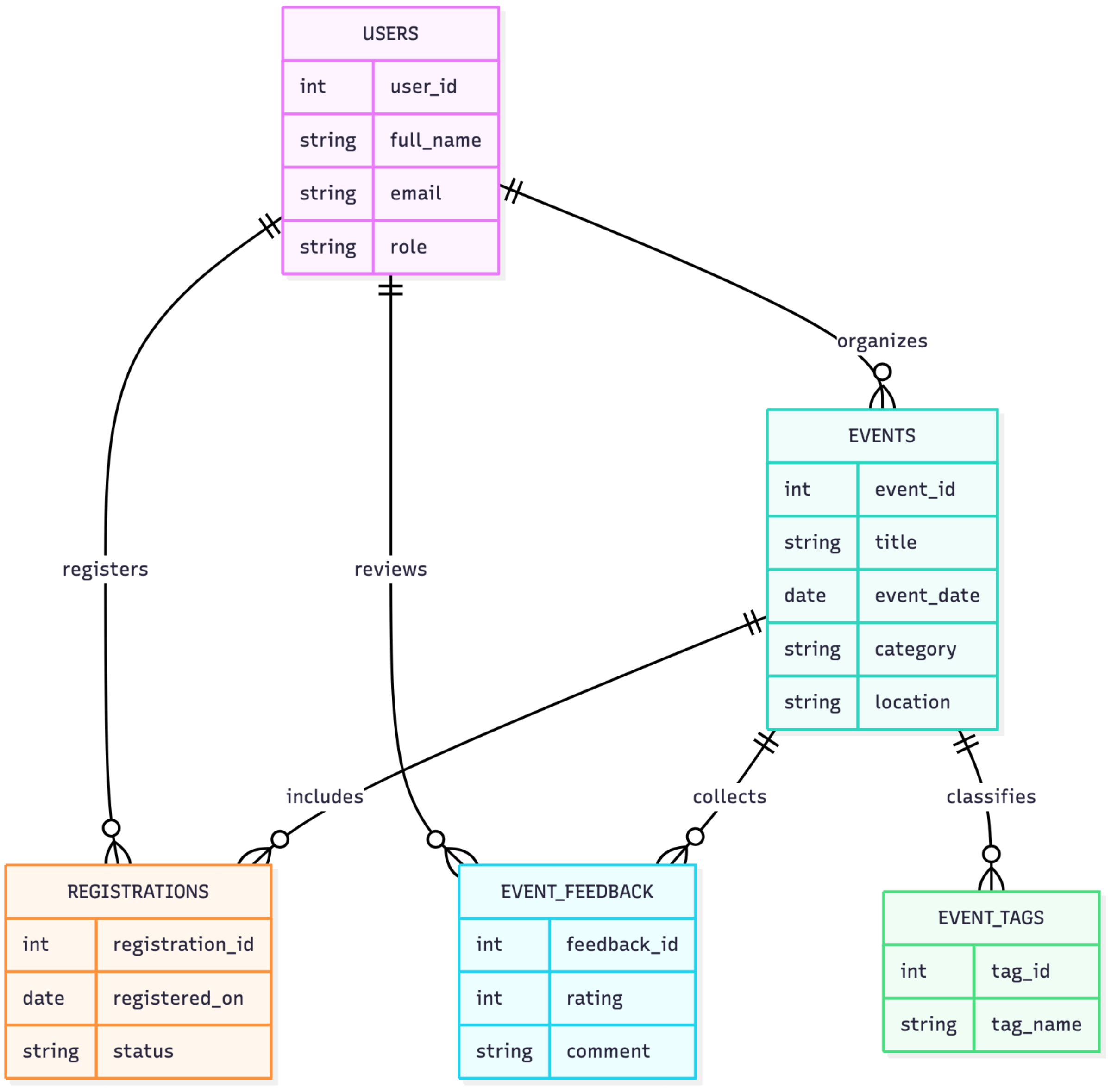

Figure 2: Conceptual entity-relationship view (showing participant, event, and tagging flows)

Deployment and User Experience: The development workflow aligns with lab delivery by bundling schema initialization scripts, health endpoints, and Tailwind build tasks within Bun. The current codebase is intentionally local-first—no cloud deployment has been provisioned yet—so students launch the stack with `bun run dev`, while schema snapshots (`public/schema.html`) aid revision. The dark UI applies copy-to-clipboard assistance, accordion-based result browsing, and status chips, making the dataset approachable for guided exercises and self-directed exploration.

Discussion

EventHive demonstrates how AI augmentation can complement, rather than replace, disciplined database design in academic settings. Continuous validation and monitoring maintain trust when autogenerated SQL is executed, giving learners guardrails while still exposing them to complex joins, CTEs, and window functions. The project also surfaces open considerations such as budgeting for Gemini API quotas, managing prompt drift across schema revisions, and hardening authentication for multi-user deployments. Scaling strategies would focus on partitioning large registration volumes and introducing caching for repeated analytical prompts. Future lab iterations could add automated test harnesses that replay exemplar questions and verify execution plans, ensuring regressions are caught before demonstrations. Finally, embedding role-based access control at the application tier would align the prototype with institutional policies for student data handling.

Conclusion

The EventHive project fulfills its laboratory objectives by coupling a well-indexed relational schema with an intelligent query assistant. Its architecture highlights best practices in data integrity, workload observability, and human-centered feedback. With incremental enhancements around security and automation, the platform is positioned to remain a relevant teaching aid for advanced database systems courses.

References

- Google Cloud: Gemini API Documentation
  https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/gemini
- Neon PostgreSQL Serverless Docs
  https://neon.tech/docs/introduction
- PostgreSQL 16 Documentation: Data Definition
  https://www.postgresql.org/docs/current/ddl.html