Project Progress Report — Team 04: AI Copilot for Databases

(Text / Voice to SQL with Safe Execution and Explanations)

Team Members: Kartavya (Architecture and Ranking), Kanav (UX and User Studies), Saarthak (PM and Documentation)

Updated: October 28, 2025

Project Type and Scope (clarifies topic change)

Research plus Prototype. We are building a small, self-hostable NL and voice to SQL copilot focused on safe, explainable querying for an existing PostgreSQL database. For Report 2, we scope delivery as text-first (primary) with voice as an optional demo. Emphasis is on guardrails: read-only role, denylist, AST parse, LIMIT enforcement, and an EXPLAIN gate.

1. Problem Statement

Relational databases remain hard to use for many non-technical users. Writing SQL requires remembering schema details and syntax, while existing AI tools can generate incorrect or unsafe queries. These issues slow decision making and reduce trust. We aim to make databases accessible through a safe, verifiable, and explainable text and voice interface.

2. Project Goal

- Securely connect to a user supplied PostgreSQL database and ingest schema metadata.
- Translate a text (primary) or dictated query (optional demo) into safe SQL with explanations.
- Block unsafe or expensive queries via static checks and an EXPLAIN gate before execution.
- Target at least 80 percent answer correctness on a small in-house set with zero unsafe executions.

3. Current Status

- Database Connector: connects to local or hosted Postgres via URL or fields; fetches schema (tables, columns, keys).
- Schema Crawler (partial): extracts table and column graph and displays it in setup flow.
- NLP Pipeline (partial): query intent parsing and schema term detection; SQL generation wiring is in progress.
- Safety (in progress): denylist for DDL or WRITE, AST parse, LIMIT injection; EXPLAIN gate stubbed.
- Frontend: basic question form, results grid, and a placeholder mic button (voice confirm modal next).
- Repo and environment: modular structure enables independent work on UI, planner, and evaluation.

4. Core Techniques (Pipeline v1)

- Schema pack: tables, columns, and types plus PK or FK graph and light value samples for grounding.
- Schema linking: map phrases to columns and tables; prefer FK consistent join paths.
- Candidate generation (up to 5): produce SQL variants with alternative joins and predicates.
- Static safety: denylist, read-only role, AST parse, and LIMIT enforcement.
- EXPLAIN gate: reject high cost plans before execution; fall back to a safer candidate.
- Answer surface: show SQL, results grid, and a short rationale.

5. Modalities and Challenges: Text vs Voice

Text NL to SQL — Challenges and fixes:

- Schema linking on unfamiliar databases and compositional queries (joins, groups, filters).
- Mitigation: FK aware linking, multi-candidate generation, and EXPLAIN gated execution.

Voice NL to SQL — Challenges and fixes:

- ASR errors on schema terms, punctuation, and long queries; browser support varies.
- Mitigation: confirm-before-SQL UI, schema aware hints; optional Whisper based fallback for robustness.

6. Positioning vs Prior Work

We do not propose a new state-of-the-art neural parser. Instead, we build a safe, self-hostable copilot that pairs lightweight schema linking and rule guided candidate generation with execution guardrails (denylist, AST, LIMIT, EXPLAIN). Research systems optimize leaderboard exact match; we optimize zero unsafe queries and understandable answers for a user's own database.

7. Evaluation Plan (Detailed)

- Datasets: two tiny in-house schemas (school, store) with 12 to 24 NL or SQL pairs each; format mirrors Spider.
- Metrics: execution success; Result F1 versus gold; exact match (informative); unsafe rate (target 0); median latency.
- Baselines: rule only (no EXPLAIN) versus our guardrailed pipeline.
- Procedure: generate up to 5 candidates, static checks, EXPLAIN, execute safest, compute metrics, and log rationale.

8. Partial Results (Pilot)

Text NL to SQL (pilot, n = 12 across 2 schemas)	Value	Notes
Execution success	10 / 12 (83%)	ran without error
Result F1 (macro)	0.72	set based comparison
Exact match SQL	7 / 12 (58%)	informative only
Unsafe rate	0 / 12	denylist, AST, LIMIT, EXPLAIN
Median latency	1.9 s	local Postgres

Safety probe (adversarial prompts, n = 10)	Blocked	Notes
DROP or DELETE or ALTER	10 / 10	denylist and AST block
Long unbounded SELECT	10 / 10	LIMIT injection and EXPLAIN gate

Voice: confirm-before-SQL UI is next; pilot WER and end-to-end success will be reported in the final.

9. Plan (Remainder of Term)

- Week of Oct 27 to Nov 2: finish EXPLAIN gate and confirm-before-SQL UI; expand NL set to 24.
- Week of Nov 3 to Nov 9: run evaluation, refine linking and ranking; add summary explanations.
- Week of Nov 10 to Nov 16: voice pilot (dictation to confirm to SQL); final demo and report.

10. Team Member Contributions

- Kartavya: database connector, schema crawler, ranking, and safety gate logic.
- Kanav: UI and UX design, usability setup, prototype testing, metrics view.
- Saarthak: project management, write ups, evaluation set creation.

Repository: https://github.com/Kartavya904/dbt-nlp-sqlizer-team04