**Project Proposal — Team 04**

AI Copilot for Databases (Text-/Voice-to-SQL with Safe Execution & Explanations)

**0) Cover Info**

Team: Kartavya (Lead: architecture & ranking), Kanav (UX & user studies), Saarthak (PM, documentation)

Project Type: Research + Prototype

Target: Natural-language (and voice) interface to any SQL database, with verified query generation, safe execution, and human-readable explanations.

**1) Problem Statement**

Non-experts struggle to extract correct, timely answers from relational databases. Even experts waste time recalling schema details, writing boilerplate SQL, and cross-checking results. Current “AI SQL helpers” often hallucinate, produce unsafe queries, or fail to explain results clearly.

**2) Proposed Solution (High-Level)**

A small, self-hostable application that connects to a target database and lets users ask questions by text or voice. The system (1) connects securely to a database (via URL or manual host/port/user/password), (2) ingests schema metadata to build a schema graph and lightweight statistics, (3) generates candidate SQL from natural language using a constrained planner, (4) verifies and safely executes the best candidate(s) in a sandbox (read-only by default), and (5) returns the SQL, the results, and a short explanation of how the query answers the question.

**3) Existing Techniques We Build On**

Text-to-SQL planning with schema-aware prompt construction and constrained decoding; schema graphs with FK/PK-driven join reasoning; ranking of multiple SQL candidates using features like parse validity, predicate alignment, and estimated cost; safety via read-only roles, whitelisting, and EXPLAIN-first checks; optional voice input using local speech-to-text.

**4) Detailed Design (Rough Method)**

4.1 Connection & Onboarding: Two options—(a) full DB URL string, (b) manual host/port/DB/user/password. Health checks validate connectivity, role permissions, and safe defaults (read-only, timeouts).

4.2 Learning Phase: No model fine-tuning; we index metadata (schemas, tables, columns, PK/FK graph, basic stats, synonyms) to build a schema context pack for planning.

4.3 NL→SQL Planner: Generate 3–5 candidates via rule-guided decomposition, FK-aware join path search, and type-checked predicates; run static validity checks (parse, identifiers).

4.4 Safety & Execution: Allow SELECT/WITH/EXPLAIN; block writes/DDL by default. Enforce LIMIT/time caps, EXPLAIN cost gates, and least-privileged read-only roles.

4.5 Ranking & Answer Synthesis: Score candidates by intent coverage, join plausibility, filter alignment, and cost. Execute the top candidate; fall back to alternatives or a minimal clarifier when needed. Outputs include SQL, paged results, CSV export, and a concise explanation.

**5) Example**

Question: “List students who took course X in Fall 2015 with a grade of C but a score < 75.”

SELECT s.student\_id, s.name, e.course\_id, e.term, e.year, e.grade, e.score  
FROM enrollment e  
JOIN students s ON s.student\_id = e.student\_id  
WHERE e.course\_id = 'X'  
 AND e.term = 'Fall' AND e.year = 2015  
 AND e.grade = 'C' AND e.score < 75  
ORDER BY s.student\_id;

Explanation: Filters by course X, Fall 2015 term, letter grade C, and numeric score under 75; joins enrollment to students to return identities.

**6) Data, Privacy, and Security**

No PII extraction; we read schema and minimal samples only (sampling optional). Connection strings are not stored on disk by default; read-only DB roles and per-schema allowlists are supported. All executed SQL is logged with the originating question; literals can be redacted on request.

**7) Evaluation Plan (Testing Plan)**

Offline Accuracy: Curate ~50–100 task questions with known-good SQL. Metrics: execution success rate, result-set F1 (set similarity), and where applicable exact-match SQL and NDCG@k for candidate ranking. Targets: ≥85% F1 on core tasks; ≥90% execution success.

Online/UX Satisfaction: Measure time-to-first-correct-answer, query reformulation rate, and UMUX-Lite 2-item score after tasks. Targets: −30% median time vs manual SQL; +0.4 UMUX-Lite uplift from baseline.

Safety: Zero production of write/DDL statements under default policy; zero queries breaching row/time caps in tests.

**8) Schedule / Milestones**

W1: Connection + schema crawler (Postgres) → demo on sample DB.

W2: NL→SQL candidate generator + static checks → ≥80% exec success offline.

W3: Safety (read-only role, LIMIT/timeout, EXPLAIN gate) → 0 unsafe queries in tests.

W4: Ranking + answer synthesis → ≥80% F1 on core tasks.

W5: Voice input + UX polish; user test #1 (UMUX-Lite baseline).

W6: Failure analysis + fixes → ≥85% F1, −30% time-to-answer vs manual SQL.

W7: Documentation, demo script, final report.

**9) Each Member’s Contribution**

Kartavya: system design, planner/ranker, safety gates.

Kanav: UX flows, usability tests, metrics instrumentation.

Saarthak: PM, write-ups, presentations, repo hygiene.