SQL Visual Optimizer

An Interactive Query Planning, Cost Analysis, and SQL Performance Laboratory

INFOGRAPHIC GUIDEBOOK
FOR THE GITHUB REPOSITORY

A 10-page visual guide to the SQL Visual Optimizer project. Explore features, concepts, and practical insights through clear infographics.

GITHUB REPOSITORY
github.com/FaramarzKowsari/sql-visual-optimizer

Understand Queries Deeply
Visualize the full journey of your SQL from parse tree to execution plan.

Analyze Costs Intelligently
Inspect operator costs, cardinalities, and data flow with rich, interactive visualizations.

Optimize with Confidence
Explore index candidates and rewrite suggestions backed by quantitative insights.

Engineer for Performance
Built for developers, researchers, and DBAs who care about performance.

10-PAGE INFOGRAPHIC GUIDEBOOK

Your visual companion to the SQL Visual Optimizer repository.
Perfect for onboarding, learning, and quick reference.

- Complete visual tour of the repository
- Key concepts, architecture, and workflows
- Step-by-step examples and insights
- Designed for developers, DBAs, and learners

Interactive by Design
Explore plans, costs, and indexes with dynamic, intuitive visuals.

Data-Driven Insights
Quantitative analysis that turns guesswork into knowledge.

Performance Focused
Identify bottlenecks, cut costs, and ship faster queries.

Built for Everyone
From developers to DBAs to researchers–optimize with clarity.

Open Source & Extensible
Transparent, collaborative, and designed to grow with the community.

Reproducible & Trusted
Reproducible results and versioned analyses you can rely on.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

What Is SQL Visual Optimizer?

From raw SQL text to visual performance insight

Think of it as your **SQL x-ray machine.** See what the engine sees—and act on it.

1 SQL Query

```
SELECT c.customer_id, c.name,
SUM(o.total_amount) AS total_spent
FROM customers c
JOIN orders o ON o.customer_id
   = c.customer_id
WHERE o.order_date >= '2024-01-01'
   AND o.status = 'completed'
GROUP BY c.customer_id, c.name
ORDER BY total_spent DESC
LIMIT 50;
```

2 Parser

- Tokenize
- Validate
- Normalize
- Bind types

Valid SQL

3 AST

4 Logical Plan

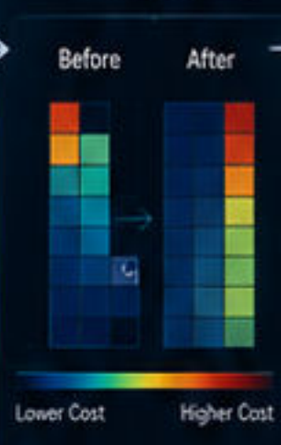
5 Findings

- 2 High Impact (Costly operations)
- 3 Medium Impact (Potential issues)
- 4 Low Impact (Minor improvements)
- 5 Info (Informational)

6 Index Hypotheses

```
CREATE INDEX
idx_orders_customer_date
ON orders (customer_id,
order_date);
----------------------
CREATE INDEX
idx_orders_status
ON orders (status);
----------------------
CREATE INDEX
idx_customers_id
ON customers (customer_id);
```

7 EXPLAIN JSON Comparison



- Total Cost -72%
- Rows Read -68%
- Plan Nodes -18%

8 Actionable Insight

- What's wrong
- Why it matters
- What to change
- What to expect
- Validate & iterate

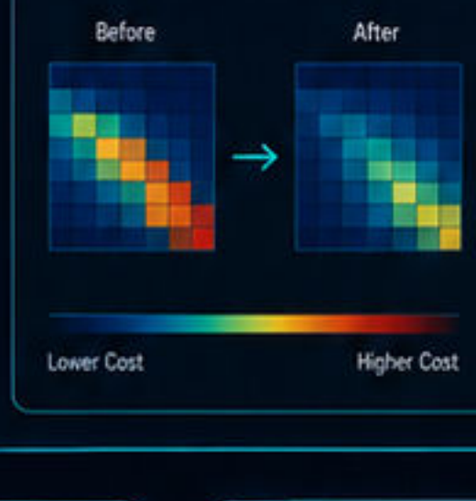
Faster, Smarter SQL

Overview

A browser-first educational SQL performance laboratory that parses queries, models likely cost centers, surfaces anti-patterns, and visualizes real PostgreSQL-compatible EXPLAIN JSON.

- **No Install** — Runs in your browser
- **PostgreSQL Compatible** — Realistic EXPLAIN JSON
- **AI-Powered Explanations** — Clear, concise rationales
- **Privacy-First** — No query data leaves your device
- **Open Source** — Transparent, trusted, extensible

Cost Heatmap (Before vs After)



Top Improvement

Total Cost **-72%** Improvement

Key Change

Add index on orders(customer_id, order_date)

Estimated Time Saved: 184 ms (72%)

Why it matters

- **Makes hidden query behavior visible** — See the plan the engine will (likely) use.
- **Supports learning and teaching** — Great for classrooms, workshops, and self-study.
- **Encourages evidence-based optimization** — Compare plans, measure impact, and validate improvements.
- **Runs without project-owned servers or embedded secrets** — You connect using your own database (optional) or use offline/estimated plans.

Three evidence labels

Estimated Plan — Estimated

Generated from query analysis and cardinality heuristics when no real plan is imported. Great for early exploration.

Imported Real Plan — Real

You import actual EXPLAIN (FORMAT JSON) from PostgreSQL-compatible databases. The source of truth.

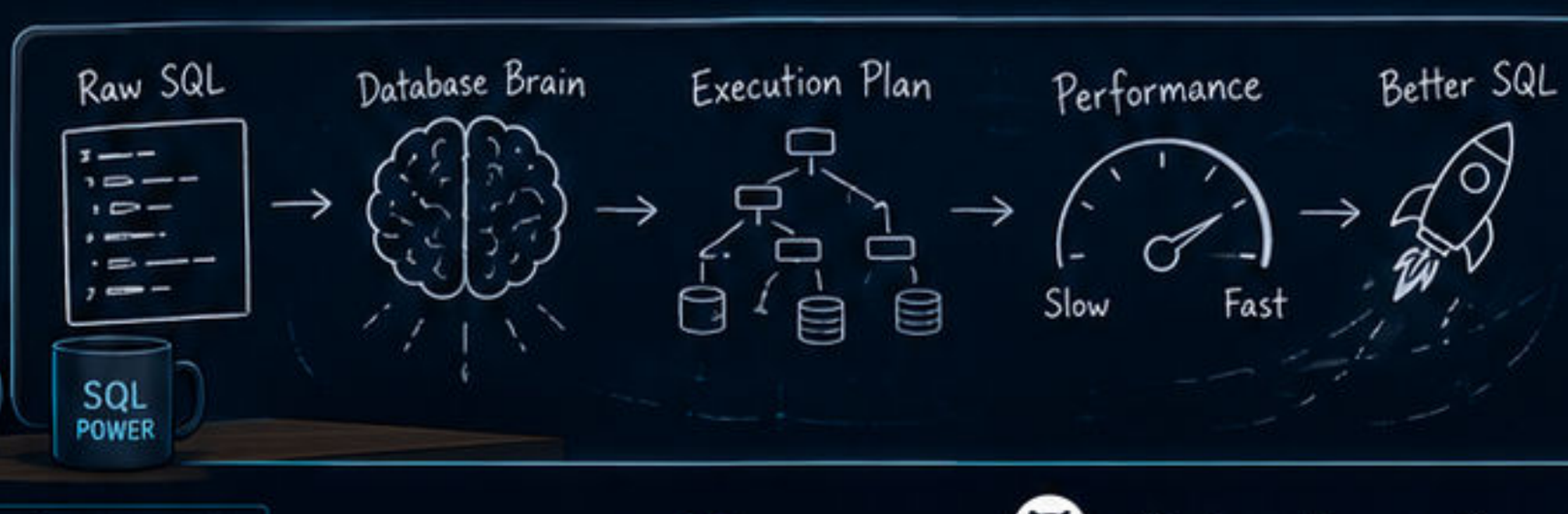
AI Explanation — AI

AI-generated natural language explanations that clarify the plan, identify risks, and suggest actions in plain English.

Label shown on every insight so you always know the evidence behind it.

Who is it for?

- **Developers** — Speed up queries and ship better code.
- **DBAs** — Diagnose, tune, and validate at scale.
- **Data Engineers** — Design efficient pipelines and models.
- **Students** — Learn how the database thinks.
- **Researchers** — Experiment and benchmark ideas.
- **Instructors** — Teach with real examples and visuals.



Golden Rule

Measure.
Understand.
Change.
Measure again.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

SQL Visual Optimizer
Mini-Guidebook

Core Features

What the laboratory can do

1 Multi-dialect SQL support

Write and analyze SQL in your preferred dialect.

- PostgreSQL
- MySQL
- SQLite
- T-SQL

One tool. Many dialects.
Consistent insights.

2 AST visualization

See your query as a clear Abstract Syntax Tree.

Understand structure.
Write better queries.

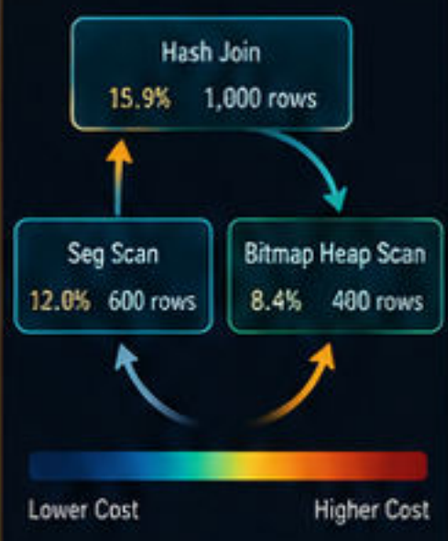
3 Transparent browser-side logical plan estimation

Instant logical plan with estimated costs—computed in your browser.

No server needed.
Fast, private, and portable.

4 Modeled row flow and relative cost heat

Understand how rows move and where cost is concentrated.



Follow the data.
Focus your optimizations.

5 SQL anti-pattern detection

Find common pitfalls that hurt performance.

- SELECT *
- Implicit conversions
- Functions on columns
- Missing WHERE filters
- NOT IN with NULs
- Non-SARGable predizations

Catch issues early.
Write efficient SQL.

6 Index candidate generation

Get data-driven index suggestions with impact estimates.

Index Candidate	Impact
orders(customer_id, order_date)	High
orders(status, order_date)	High
orders(customer_id)	Medium
customers(country, created_at)	Low

Better indexes.
Better performance.

7 Schema and table statistics editing

Edit statistics to reflect your environment and see the impact.

Table: orders	
Row Count	1,250,000
Width (avg)	128 bytes
Distinct: customer_id	320,000
Distinct: status	6
Null %: shipped_at	12%

Accurate stats.
Accurate plans.

8 PostgreSQL-compatible EXPLAIN JSON import

Import plans exported from PostgreSQL.

```
{
  "Plan": {
    "Node Type": "Hash Join",
    "Startup Cost": 0.71,
    "Total Cost": 15.96,
    "Plan Rows": 50,
    "Plan Width": 72,
    "Plans": [
    {
      "Node Type": "Seq Scan",
      "Relation Name": "customers"
    },  ...
  ]
}
```

Bring your DB's plan.
Analyze offline.

9 Local rule-based explanation without AI

Get clear, deterministic explanations generated by built-in rules.

- ✓ Deterministic
- ✓ Always available
- ✓ Private & offline
- ✓ Fast & transparent
- ✓ No data leaves your browser

Knowledge you can trust.
No network required.

10 Optional Gemini BYOK explanation

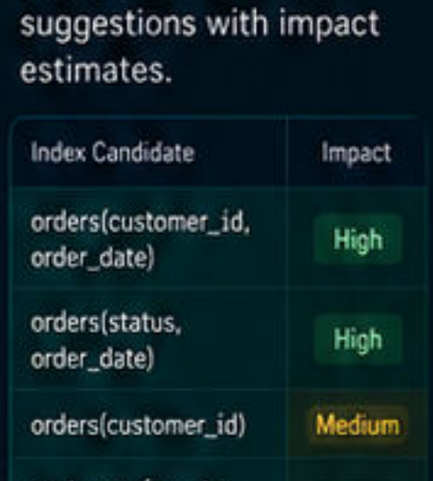
Use your own Google Gemini API key for deeper natural language explanations.

Gemini (BYOK)

Status	Connected
Model	gemini-1.5-flash

Explain with Gemini

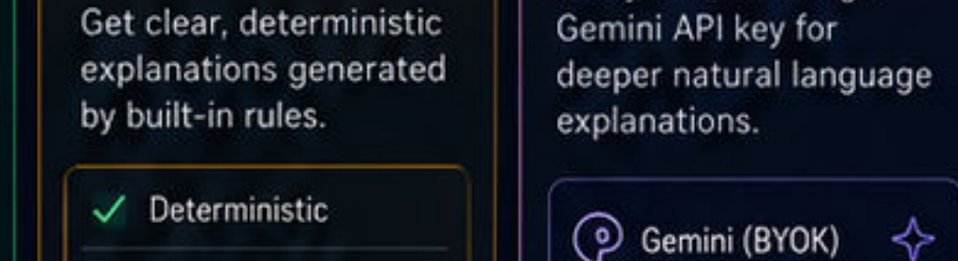
You control the key.
You control the data.



No paid project infrastructure required

SQL Visual Optimizer is a fully client-side web app.
It runs as static files on GitHub Pages—no servers, no database, no backend.

- ☑ Open source
- ☑ Free to host
- ☑ Easy to deploy
- ☑ Private: your queries never leave your browser
- ☑ Always available
- ☑ Works offline after loading




```
$ git clone https://github.com/FaramarzKowsari/sql-visual-optimizer.git
$ cd sql-visual-optimizer
$ open index.html    # or open in your browser
```

That's it. You're ready to optimize!

Private by design — Everything runs in your browser.
Secure — No data is sent to any server.
Fast — Instant feedback and visualization.
Explainable — Understand every step and decision.
Data-driven — Better stats lead to better plans.
For everyone — DBAs, developers, researchers, students.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

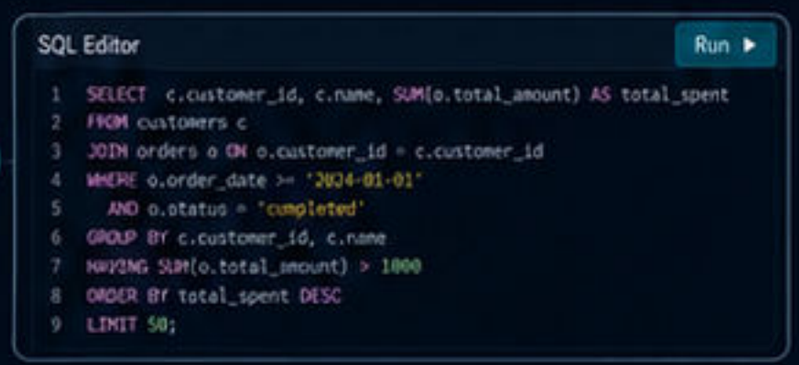
How the Workflow Works

A guided path through the application

1 Choose a sample clinic or paste your own SQL.

Start fast with curated samples or bring your own query.

OR



2 Select the SQL dialect.

Pick the engine to get accurate rules and plan behavior.

3 Format or edit the query.

Clean, consistent SQL is easier to read and optimize.

4 Review schema statistics and declared indexes.

Accurate stats and good indexes are the foundation of reliable analysis.

5 Run the analysis.

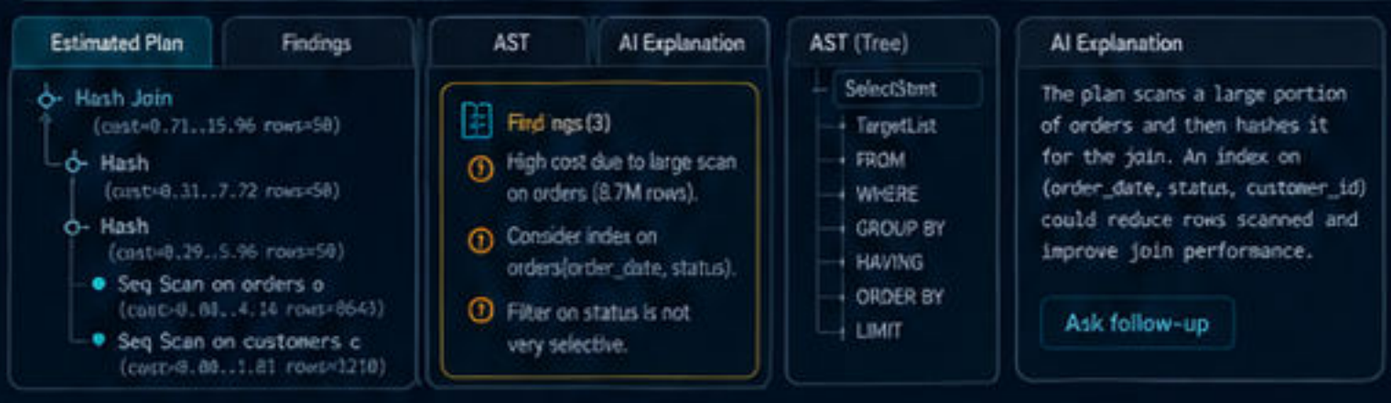
Let the optimizer analyze costs, cardinalities, and plan alternatives.

6 Inspect the health score, estimated cost, complexity, and rows scanned.

High-level signals that tell you where to focus.

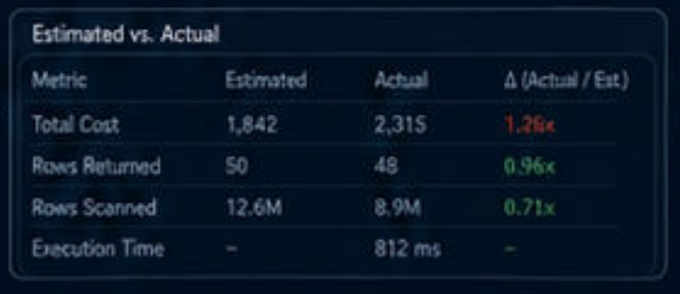
7 Explore the Estimated Plan, Findings, AST, and AI Explanation.

Understand why the plan looks the way it does.



8 Import real EXPLAIN JSON for comparison.

Compare estimated vs. actual to understand gaps and tune better.



9 Turn findings into index experiments or query rewrites.

Iterate, compare, and keep what actually helps.

Best practices

Use real plans for verification.
Import EXPLAIN (ANALYZE) JSON to validate estimates with actual execution.

Treat estimates as hypotheses.
Optimizers estimate. Validate, measure, and iterate.

Edit table statistics for realism.
Keep row counts and distributions up to date for trustworthy plans.

Compare alternatives, not absolutes.
The "best" plan depends on your workload and data distribution.

Keep AI interpretation separate from evidence.
AI explanations help you understand—plans and metrics show what actually happens.

> "Good data in. Clear evidence out. Better decisions in between."
>
> — SQL Visual Optimizer

Interactive by Design
Explore plans, costs, and indexes with dynamic, intuitive visuals.

Data-Driven Insights
Quantitative analysis that turns guesswork into knowledge.

Performance Focused
Identify bottlenecks, cut costs, and ship faster queries.

Built for Everyone
From developers to DBAs to researchers—optimize with clarity.

Open Source & Extensible
Transparent, collaborative, and designed to grow with the community.

Reproducible & Trusted
Reproducible results and versioned analyses you can rely on.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

Interface Deep Dive

A guided map of the live dashboard

How to use this page

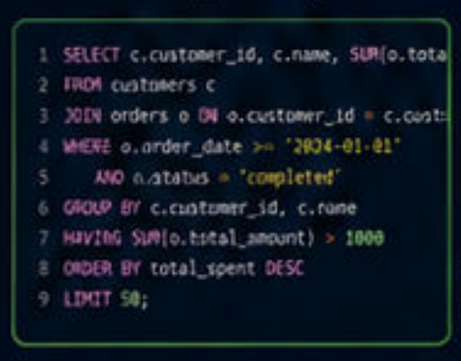
Each numbered callout corresponds to a part of the interface. Follow the map, explore the features, and build better, faster SQL.

1 Site header and navigation

Access key modules, save workspaces, manage settings, and sync with GitHub. Start here.

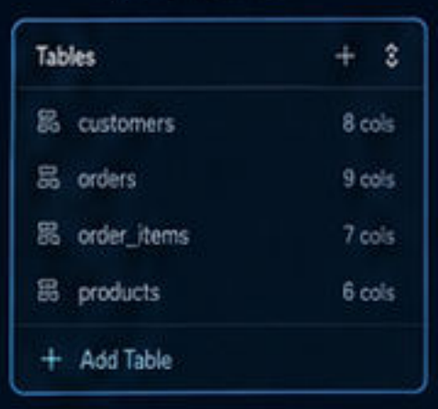
3 SQL editor

Write or paste your query, use smart formatting, and get live linting. Supports auto-complete and snippet library.



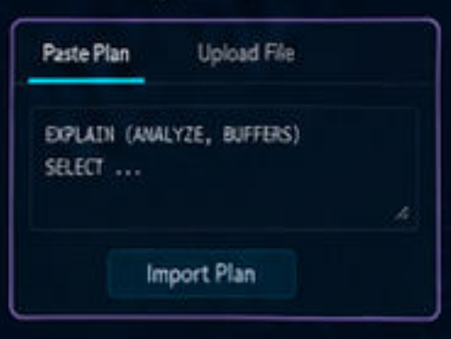
9 Schema editor

Define or edit tables, columns, and indexes used by the optimizer.



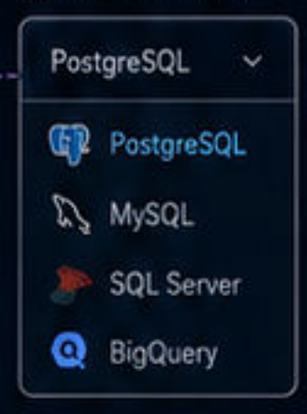
10 Explain importer

Paste or upload an EXPLAIN / ANALYZE plan from your database.



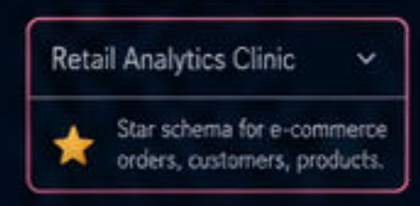
4 Dialect selector

Switch SQL dialects to match your database engine and optimizer behavior.



5 Example clinic selector

Load curated scenarios and datasets. Best for quick experimentation.



6 Format / show schema controls

Format your SQL and toggle table/column visibility to focus on what matters.

7 Metric cards

At-a-glance health of your query. Use this to compare assumptions with evidence.

8 Parser status & issue summary

Know if your query parsed cleanly and what to fix next.

9 Results tabs

Explore different views of your query and let the AI explain the "why".

How to explore (recommended flow)

1 Write or paste your SQL → Pick a clinic & dialect → Run & review metrics → Inspect plan & findings → Iterate & save improvements

Pro tip

Use clinics to test ideas quickly, then save and compare your best plans with real execution data.

SQL Visual Optimizer
An interactive query planning, cost analysis, and SQL performance laboratory.

GitHub Repository
Star, fork, and contribute to the project.

Documentation
Deep guides and reference material.

Discussions
Ask questions and share optimization stories.

Releases
Track updates and new features.

Roadmap
See what's coming next.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

Repository Architecture

SQL Visual Optimizer
MINI-GUIDEBOOK

How the project is organized under the hood

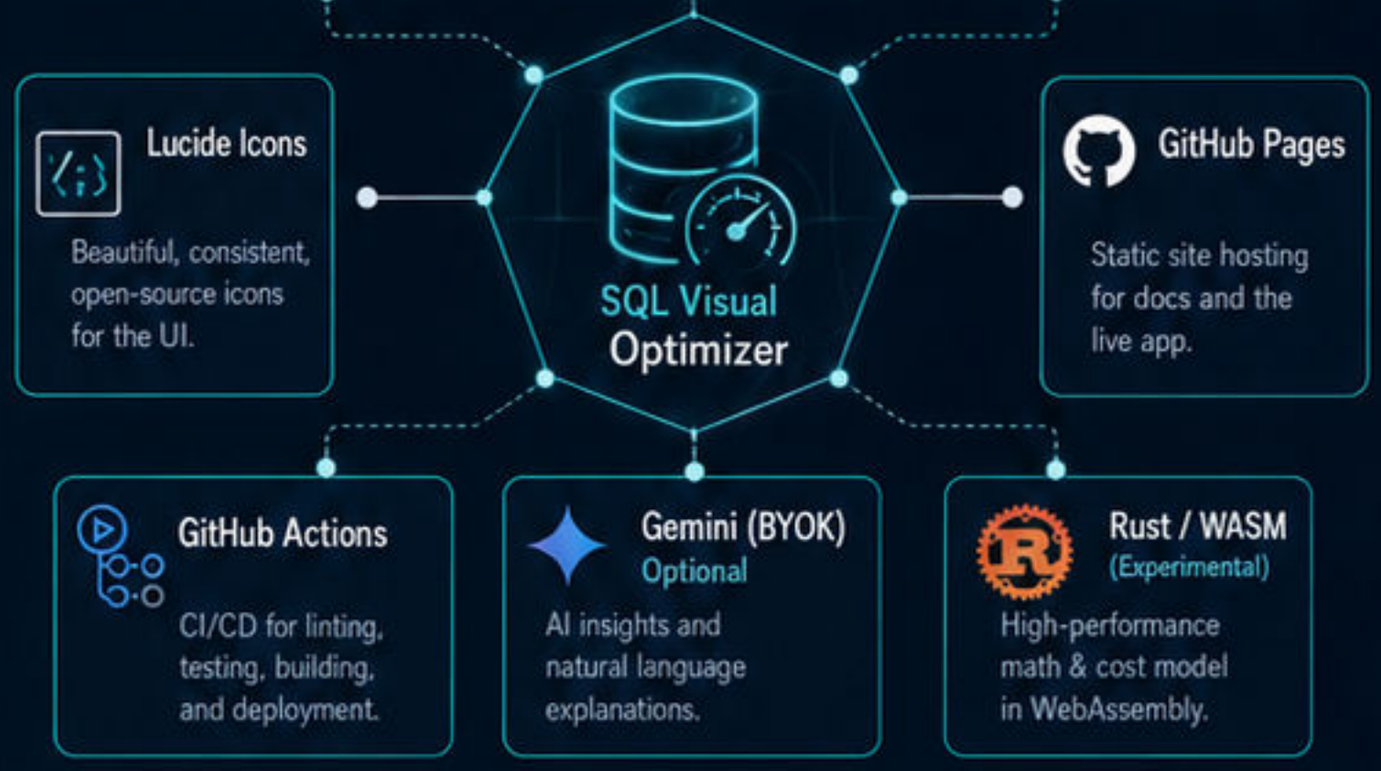
A) Repository tree

Key directories and what they do.

sql-visual-optimizer/	
src/	
components/	Reusable UI components and visual panels.
data/	Static data, cost models, examples, and lookups.
lib/	
analyzer.ts	Deterministic SQL analysis (engine core).
explain.ts	Parsers for EXPLAIN/PLAN (JSON, text, adapters).
gemini.ts	Optional AI insights via Gemini (BYOK).
types.ts	Shared TypeScript types and interfaces.
crates/	
query-math/	Experimental Rust library (for advanced math).
examples/	Example queries and usage scenarios.
benchmarks/	Performance benchmarks and datasets.
docs/	Documentation, guides, and architecture notes.
.github/	
workflows/	CI/CD pipelines and automations.
README.md	Project overview and getting started.
CITATION.cff	Scholarly citation metadata.
LICENSE	MIT License.
package.json	Project metadata & scripts.

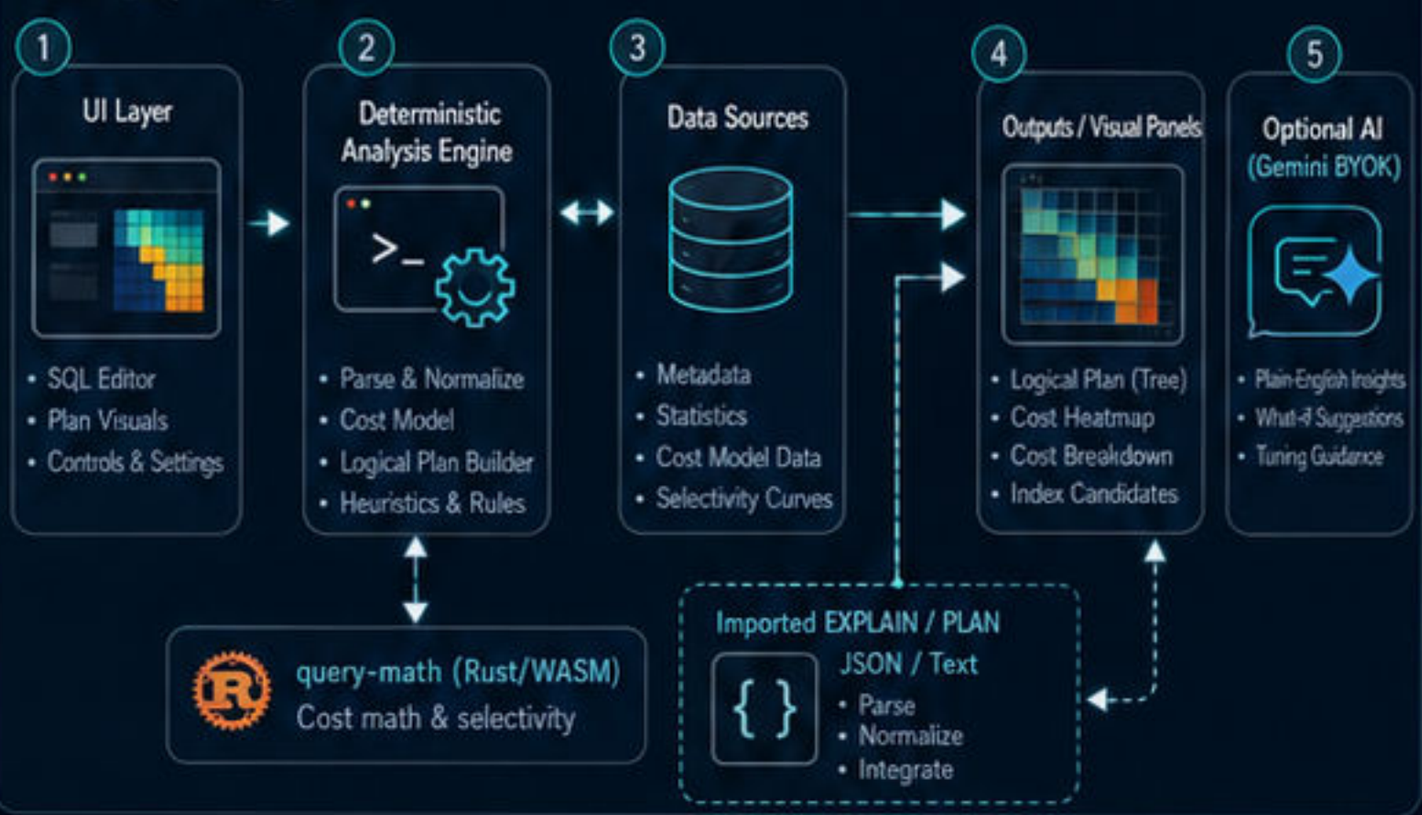
B) Tech stack map

The technologies that power SQL Visual Optimizer.



C) Architecture flow

From query to insights.



D) Scholarly and repository metadata

Everything you need to cite, trust, and build upon this work.

README.md
Project overview, installation, usage, and contribution guide.
Start here

CITATION.cff
Scholarly citation metadata for academic and research use.
Cite this work

Zenodo DOI
DOI: 10.5281/zenodo.21501361
Permanent, citable release with versioned archives.
View on Zenodo

Documentation
Deep dives, architecture notes, and best practices for advanced users.
Explore docs

Transparent
Open-source code, clear algorithms, and full reproducibility.

Modular
Clean separation of concerns for easy extension.

Performant
WASM-accelerated math and efficient analysis engine.

Extensible
Plug in new adapters, cost models, and visualizations.

Community-Driven
Built for researchers, engineers, and data professionals.

Scholarly-Ready
Citations, DOI, and docs for academic and practical impact.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

Inside the Analyzer

How the deterministic engine reasons about SQL

The SQL Visual Optimizer runs entirely in your browser. Given your SQL, chosen dialect, and schema statistics, it deterministically parses, models, estimates costs, finds issues, and proposes index improvements.

Project Reference

SQL Visual Optimizer
github.com/FaramarzKowsari/sql-visual-optimizer

1 Input — SQL text + selected dialect + schema statistics.

2 Detection — Extract tables, aliases, joins, filters, aggregates, sorts, limits, offsets.

3 Modeling — Estimate row flow, cost signals, and complexity score.

4 Findings — Detect anti-patterns and generate warnings.

5 Index Hypotheses — Propose candidate indexes that can improve the plan.

6 Evidence Boundary — Relative cost units are educational, not latency.

1 INPUT

SQL TEXT

```sql
SELECT c.customer_id, c.name, SUM(o.total_amount) AS total_spent
FROM customers c
JOIN orders o ON o.customer_id = c.customer_id
WHERE o.order_date >= '2024-01-01'
  AND o.status = 'completed'
GROUP BY c.customer_id, c.name
HAVING SUM(o.total_amount) > 1000
ORDER BY total_spent DESC
LIMIT 50 OFFSET 1000;
```

DIALECT: PostgreSQL

SCHEMA STATISTICS (SAMPLE)

Table	Rows	Distinct	Notes
customers	1.2M	customer_id (1.2M)	PK: customer_id
orders	28.6M	customer_id (1.2M)	PK: order_id
order_items	72.4M	order_id (28.6M)	PK: order_item_id

2 DETECTION (What we extract)

Tables & Aliases
- customers AS c
- orders AS o

Joins
- o.customer_id = c.customer_id

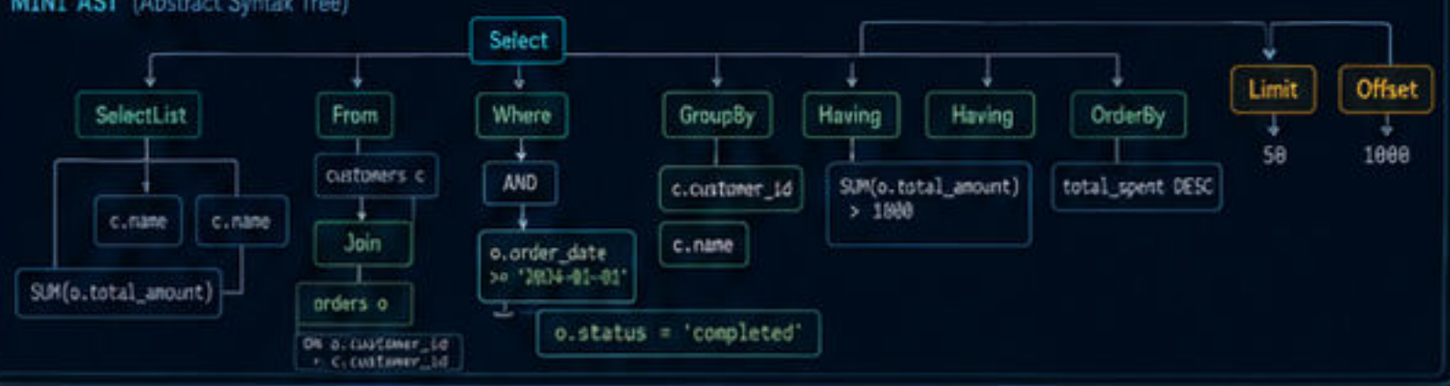
Filters
- o.order_date >= '2024-01-01'
- o.status = 'completed'

Aggregates
- SUM(o.total_amount)
- GROUP BY c.customer_id, c.name

Sorts
ORDER BY total_spent DESC (derived alias)

Limits / Offsets
LIMIT 50
OFFSET 1000

MINI AST (Abstract Syntax Tree)



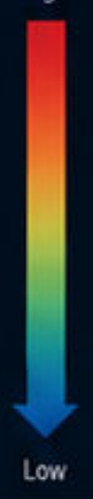
ANALYZER LOGIC (High-Level)

```
function analyzeSql(sql, dialect, stats):
  ast = parse(sql, dialect)
  clauses = extractClauses(ast)
  sem = buildSemanticModel(ast, stats)
  plan = estimatePlan(sem, stats)
  findings = computeFindings(sem, plan, stats)
  indexes = proposeIndexes(sem, plan, stats)
  score = computeComplexity(plan, findings)
  return { plan, findings, indexes, score }
```

3 MODELING (Deterministic cost reasoning)

PLAN TREE (Estimated)

ROW FLOW (Approx.)



COST SIGNALS WE USE

- ✓ Table/Cardinality estimates from stats
- ✓ Selectivity of predicates
- ✓ Join type & join key coverage
- ✓ Sort / Hash / Aggregate overheads
- ✓ I/O vs CPU mix (heuristic)
- ✓ Penalty for deep OFFSET
- ✓ Wide rows & SELECT * penalty

COMPLEXITY SCORE

Moderate → High

Drivers
- Deep OFFSET (1000)
- Large Aggregation
- Sort on derived alias

4 FINDINGS (Anti-patterns & Warnings)

- **Deep OFFSET pagination** — OFFSET 1000 forces the engine to process and discard many rows. — High
- **SELECT list could be narrower** — Avoid SELECT * or unnecessary columns to reduce row width. — Medium
- **GROUP BY on non-key columns** — Grouping on c.name may increase cardinality and memory. — Medium
- **HAVING on aggregate** — HAVING SUM(o.total_amount) > 1000 may be non-sargable early. — Medium
- **Status filter may be low-selective** — If 'completed' is most rows, index benefit may be limited. — Low
- **Consider keyset pagination** — Use WHERE (total_spent, c.customer_id) < (:last_val, :last_id) — Info

5 INDEX HYPOTHESES (Generated from predicates & joins)

Candidate Index	Why it helps	Impact	Cost Reduction (Est.)
orders(customer_id, order_date, status) INCLUDE (total_amount)	Supports join + filter + date range + aggregation	High	58%
orders(status, order_date) INCLUDE (customer_id)	Helps filter by status and date range	Medium	31%
customers(customer_id)	Join key lookup	Low	12%
orders(customer_id)	Join key only (fallback)	Low	8%

Heuristics: *Prefer composite indexes aligned with equality → range → include columns.*

6 EVIDENCE BOUNDARY

All costs, rows, and impact percentages are relative educational estimates, not production latency or SLAs.

Relative Units — "Cost" is in relative units based on heuristics and statistics—not time.

Heuristic Model — The engine uses deterministic rules, not live execution plans or runtime metrics.

Stats Dependence — Estimates vary with table statistics quality and data distribution.

Use Responsibly — Validate suggestions with EXPLAIN (ANALYZE) and real workloads.

Think of it as a smart whiteboard, not a stopwatch.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

Privacy, BYOK AI, Deployment & CI/CD

Why the project is lightweight, safe, and reproducible

SQL Visual Optimizer

An Interactive Query Planning, Cost Analysis, and SQL Performance Laboratory

github.com/FaramarzKowsari/sql-visual-optimizer

DOI 10.5281/zenodo.21501361

A Browser-first deployment

Static files · GitHub Pages · No project-owned server · No paid runtime · No project-owned API key

- No project-owned server
- No paid runtime
- No database
- No project-owned API key embedded

What this means

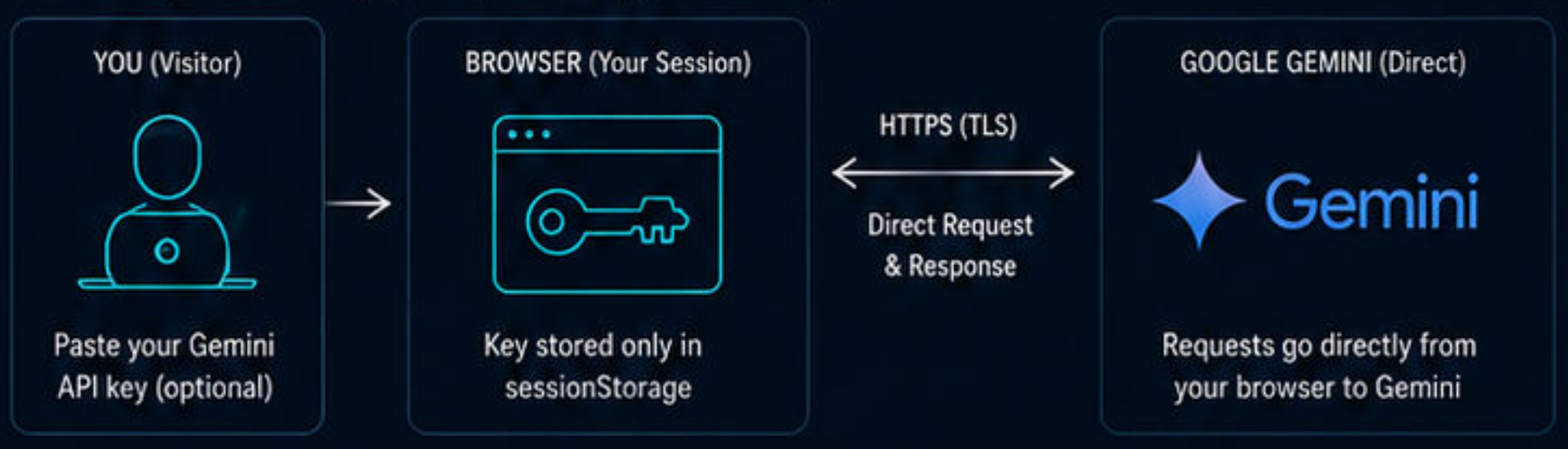
- Runs entirely in your browser.
- No backend costs.
- No project access to your data or queries.
- Easy to audit, fork, and mirror.

You stay in control.

The project serves only static files—nothing more.

B Optional Gemini BYOK mode

Bring Your Own Key (BYOK) · Your key, your control, your session



How it works

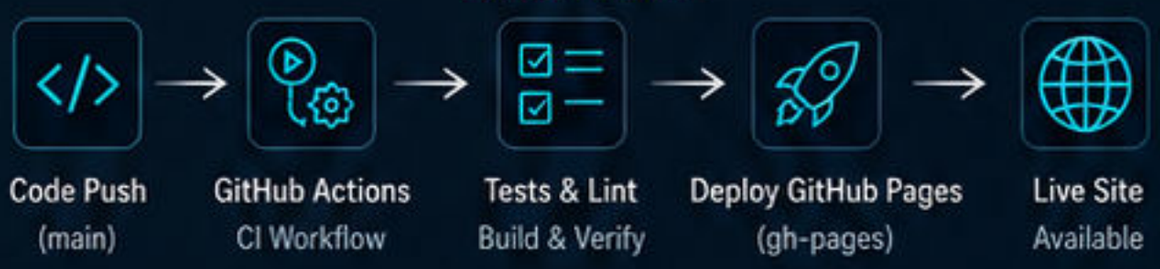
- You provide your own API key.
- Requests are sent directly from your browser to Gemini.
- No proxy. No project server.
- Key is stored only in sessionStorage.
- Cleared when the tab/session ends.

Important

Do not submit confidential production queries without authorization.

C Reproducibility and trust

Open workflows · Versioned releases · Archived · Citable · Licensed



RELEASES & ARCHIVAL

- **Versioned Releases** — Tagged snapshots
- **Zenodo Archiving** — Automatic archive
- **DOI (Citable)** — 10.5281/zenodo.21501361
- **MIT License** — Open & permissive

Why it matters

- Deterministic builds and public workflow logs.
- Anyone can reproduce, audit, and extend.
- Archived for the long term.
- Citable and creditable.
- Open source under the MIT License.

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software...

D Security principles

Transparent labels · No fake execution claims · Estimates vs real plans

EXPLAIN Plans (Estimated)

- Estimated by the optimizer
- Uses statistics & assumptions
- May differ from real runtime

≠

Execution (Real)

- Actual runtime & I/O
- Environment dependent
- Measure with real execution

Our commitments

- We show estimates, not guaranteed results.
- We do not execute your queries.
- We do not store or log your queries.
- You remain in control of your data.
- Transparent, honest, and user-first.

Honest labels. Accurate context. Better decisions.

Interactive by Design

Explore plans, costs, and indexes with dynamic, intuitive visuals.

Data-Driven Insights

Quantitative analysis that turns guesswork into knowledge.

Performance Focused

Identify bottlenecks, cut costs, and ship faster queries.

Built for Everyone

From developers to DBAs to researchers—optimize with clarity.

Open Source & Extensible

Transparent, collaborative, and designed to grow with the community.

Reproducible & Trusted

Reproducible results and versioned analyses you can rely on.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

Use Cases, Research Value & Citation

Why this repository matters beyond the demo

1 USE CASES

Who benefits and how this repository is used in the real world.

Classroom Teaching

Demystify the optimizer with visual plans, costs, and clear explanations.

Self-Study & Upskilling

Learn by doing—write queries, inspect plans, and refine iteratively.

Technical Demos

Engage audiences with live, data-driven explanations.

SQL Clinics

Diagnose slow queries, recommend fixes, and show the impact.

Onboard Junior Analysts

Build mental models of plans, costs, and performance trade-offs.

Database Workshops

Hands-on labs for indexes, joins, stats, and query rewrites.

Exploratory Optimization

What-if analysis with hints, indexes, and alternative strategies.

Document Performance Thinking

Capture experiments, insights, and decisions for future reference.

2 RESEARCH VALUE

Why this project contributes meaningfully to research and education.

Evidence Labels

Every insight is backed by observable evidence (EXPLAIN plans, costs, and runtime outcomes).

EVIDENCE-DRIVEN

Transparent Heuristics

Cost assumptions, rules, and heuristics are explicit, inspectable, and open to critique.

INTERPRETABLE

Reproducible Releases

Versioned datasets, queries, and environments enable reproducible experiments.

REPRODUCIBLE

Scholarly Software Identity

Citable DOI, open source, and documented methods for academic and professional use.

CITABLE & TRUSTED

Pedagogical Clarity

Designed to turn complex optimizer behavior into clear, teachable concepts.

TEACHING-FIRST

3 CITATION

If you use this work, please cite it.

SQL VISUAL OPTIMIZER

Title	SQL Visual Optimizer: An Interactive Query Planning, Cost Analysis, and SQL Performance Laboratory
Version	v1.0.0
DOI	10.5281/zenodo.21501361
Repository	github.com/FaramarzKowsari/sql-visual-optimizer

4 ROADMAP — WHERE WE'RE HEADING NEXT

A living plan shaped by community feedback and research needs.

1 → 2 → 3 → 4 → 5

Richer Cost Models

More accurate models across engines, statistics, and hardware profiles.

DEEPER INSIGHT

Benchmark Suites

Curated workloads and microbenchmarks for repeatable evaluation.

STRONGER EVIDENCE

More EXPLAIN Dialects

Support for additional databases and plan formats (Postgres, MySQL, SQL Server, BigQuery, Oracle, DuckDB...).

WIDER COVERAGE

Expanded Teaching Clinics

Scenario packs, exercises, and instructor guides for classrooms and workshops.

BETTER PEDAGOGY

Stronger Visualization Tooling

Interactive what-if, diff views, and explain-aware recommendations.

BETTER DECISIONS

5 GET INVOLVED — YOUR CONTRIBUTION MATTERS

Use it, teach with it, cite it, and help others learn.

Explore the repo

- Browse the code
- Read the docs
- Star the project

github.com/FaramarzKowsari/sql-visual-optimizer

Run the live app

- Try queries
- Inspect plans
- Compare costs

Launch the App

Cite the software

- Use the DOI
- Give proper credit
- Boost open science

View Citation

Use it in teaching

- Labs & exercises
- Classroom demos
- Workshop materials

Teaching Resources

Scan to visit the repository

Interactive by Design

Explore plans, costs, and indexes with dynamic, intuitive visuals.

Data-Driven Insights

Quantitative analysis that turns guesswork into knowledge.

Performance Focused

Identify bottlenecks, cut costs, and ship faster queries.

Built for Everyone

From developers to DBAs to researchers—optimize with clarity.

Open Source & Extensible

Transparent, collaborative, and designed to grow with the community.

Reproducible & Trusted

Reproducible results and versioned analyses you can rely on.

About the Author

Faramarz Kowsari

Author · Software Engineer · AI Researcher

Faramarz Kowsari is an author, Software Engineer and AI researcher based in Istanbul. Focusing on the intersection of technology, education, and personal growth, he has published over 80 digital titles on international platforms.

His areas of expertise span Artificial Intelligence, prompt engineering, modern trading strategies (Smart Money Concepts & algorithmic trading), as well as classical literature and mindfulness.

In addition to writing, he develops web-based educational tools and creates specialized instructional video content.

Official Profiles & Repositories

ORCID:
https://orcid.org/0000-0003-1692-0453

Google Scholar:
https://scholar.google.com/citations?user=G7tP5WMAAAAJ&hl=en

GitHub:
https://github.com/FaramarzKowsari

LinkedIn:
https://www.linkedin.com/in/faramarzkowsari

Google Books:
https://play.google.com/store/search?q=Faramarz_Kowsari&c=books

Official Website:
https://FaramarzKowsari.github.io

Zenodo Records:
https://zenodo.org/search?q=creators.orcid%3A%220000-0003-1692-0453%22&l=list&p=1&s=10&sort=bestmatch

About SQL Visual Optimizer

SQL Visual Optimizer is an interactive, data-driven platform for SQL query planning, cost analysis, and performance insights.

Visualize. Analyze. Optimize.
Transform complex SQL execution plans and cost metrics into clear, interactive visualizations.

Data-Driven Intelligence
Inspect operator costs, cardinalities, and data flow to make informed optimization decisions.

Built for Developers & DBAs
A practical toolkit for query tuning, performance engineering, and query plan exploration.

Open Source & Community-Driven
Transparent, extensible, and designed to grow with the community.

DOI 10.5281/zenodo.21501361

github.com/FaramarzKowsari/sql-visual-optimizer

Interactive by Design
Explore plans, costs, and insights with intuitive visuals.

Data-Driven Insights
Quantitative analysis that turns guesswork into knowledge.

Performance Focused
Identify bottlenecks, cut costs, and ship faster queries.

Built for Everyone
From developers to DBAs to researchers–optimize with clarity.

Open Source & Extensible
Transparent, collaborative, and designed to grow with the community.

Reproducible & Trusted
Reproducible results and versioned analyses you can rely on.

Thank you for exploring this mini-guidebook.
If you find SQL Visual Optimizer useful, please star the repository, share your feedback, *and contribute to make it even better for everyone.*

Scan to visit the GitHub repository and get involved!