1. What you’re building (scope for v1.0)

Deliverables

queryshield-probe (Python pkg) — instruments Django DB calls, classifies problems, can (optionally) run EXPLAIN (FORMAT JSON) safely, and emits a machine‑readable queryshield\_report.json. Uses Django’s official database instrumentation API (connection.execute\_wrapper). Django Project

queryshield (Python CLI) — runs your project’s tests or a target script under the probe, compares results to a budgets YAML, and sets exit codes for CI.

QueryShield (JetBrains plugin, Kotlin) — for PyCharm (Community/Pro) and IntelliJ with the Python plugin; shows in‑editor inlay‑style counters, a Tool Window dashboard, and intention actions (“add select\_related()”, etc.). Integrates JetBrains Marketplace licensing for paid distribution. JetBrains Marketplace +3 JetBrains Marketplace +3 JetBrains Marketplace +3

Non‑goals (v1.0): No production APM; no server; no telemetry; PostgreSQL first (safe EXPLAIN only). (MySQL support and agent PR bots come in v1.1.)

1. Repo layout

Monorepo recommended:

queryshield/ ├─ probe/ # Python package: queryshield-probe │ ├─ queryshield\_probe/ │ │ ├─ **init**.py │ │ ├─ capture.py # execute\_wrapper, stack capture, redaction │ │ ├─ classify.py # heuristics (N+1, seq scan, sort w/o index, select \*) │ │ ├─ explain\_pg.py # EXPLAIN (FORMAT JSON) helpers │ │ ├─ report.py # JSON schema & writer │ │ ├─ budgets.py # YAML parsing, thresholds │ │ ├─ runners/ │ │ │ ├─ django\_runner.py # run tests under instrumentation │ │ │ └─ pytest\_plugin.py # optional pytest integration │ │ └─ utils.py │ ├─ pyproject.toml │ └─ tests/ │ ├─ cli/ # Python CLI: queryshield │ ├─ queryshield\_cli/ │ │ ├─ **init**.py │ │ ├─ main.py # click/typer entrypoint │ │ └─ commands/ # analyze, budget-check, record-baseline, verify-patch │ ├─ pyproject.toml │ └─ tests/ │ ├─ ide-plugin/ # JetBrains plugin (Kotlin) │ ├─ build.gradle.kts │ ├─ gradle.properties │ ├─ settings.gradle.kts │ ├─ resources/META-INF/plugin.xml │ └─ src/main/kotlin/ │ ├─ Settings.kt │ ├─ InlayProvider.kt │ ├─ ToolWindowFactory.kt │ ├─ RunAnalysisAction.kt │ ├─ Intentions.kt │ ├─ CliProcess.kt # Execution API wrapper │ └─ Licensing.kt │ └─ sample-django-app/ # integration tests & demos ├─ docker-compose.yml # PostgreSQL + seed data ├─ manage.py ├─ settings.py (minimal) ├─ app/ (Author, Book models; N+1 route) └─ tests/

1. Python probe (core instrumentation) 2.1 Install hook

Use Django’s connection.execute\_wrapper() around any code block to intercept every DB execution (including executemany). The wrapper receives (execute, sql, params, many, context); call execute(...) inside and record timing + metadata. This is the canonical and supported API. Django Project

# queryshield\_probe/capture.py

import time, inspect, threading from contextlib import contextmanager from django.db import connection

\_local = threading.local()

class QueryEvent: **slots** = ("sql", "params", "duration\_ms", "many", "stack", "error") ...

def \_stack\_signature(skip=0, depth=8): frames = inspect.stack()[skip+1:skip+1+depth] return [(f.filename, f.function, f.lineno) for f in frames if "/site-packages/" not in f.filename]

class ProbeWrapper: def **init**(self, recorder): self.recorder = recorder def **call**(self, execute, sql, params, many, context): start = time.perf\_counter() err = None try: return execute(sql, params, many, context) except Exception as e: err = repr(e) raise finally: ev = QueryEvent() ev.sql = sql ev.params = params ev.duration\_ms = (time.perf\_counter() - start) \* 1000.0 ev.many = bool(many) ev.stack = \_stack\_signature(skip=1) ev.error = err self.recorder.record(ev)

@contextmanager def install\_probe(recorder): with connection.execute\_wrapper(ProbeWrapper(recorder)): yield

Why this API: execute\_wrapper is official, context‑managed, and works per‑thread. Avoid monkey‑patching drivers. Django Project

2.2 Redaction & normalization

Store normalized SQL for grouping (replace literals with placeholders).

Redact params in reporting (keep types/shape only).

2.3 Heuristics (v1 set)

Implement in classify.py:

N+1 detector

Group by (normalized\_sql, top\_of\_stack\_signature); if count ≥ N (default 5) and query touches the same table/foreign key path, flag N+1; attach suggested fix: .select\_related() for FKs/OneToOne; .prefetch\_related() for M2M/reverse FK.

Show before/after estimate (count → 1–3) based on relation shape.

Full scan / missing index (PostgreSQL)

If EXPLAIN JSON shows a Seq Scan on a table with a filter (and row estimate >> 1000 by heuristic or table‑size hint), flag MISSING\_INDEX; suggest basic/compound index from equality predicates; mark SAFE (we used EXPLAIN only). PostgreSQL

Sort without index

If plan has a Sort node with large input and no matching index prefix for ORDER BY, suggest composite index with equality predicates first, then order-by keys.

SELECT \* on large tables

If normalized SQL selects \* from table estimated large, suggest .only()/explicit fields for Django or .values() when appropriate.

Use EXPLAIN (FORMAT JSON), not ANALYZE, by default (no data changes; no long‑running exec). Provide a “verify with ANALYZE” opt‑in for local dev only. Cite official docs. PostgreSQL +1

2.4 Explain helper

explain\_pg.py:

If DB vendor is PostgreSQL, run EXPLAIN (FORMAT JSON) with params bound safely.

Return parsed JSON plan and surface key nodes (Node Type, Relation Name, Index Name, Sort Key, estimated rows/cost). PostgreSQL

2.5 Report schema (JSON)

report.py writes a single file (default: .queryshield/queryshield\_report.json):

{ "version": "1", "project\_root": "/abs/path", "timestamp": "2025-10-09T12:34:56Z", "framework": { "name": "django", "version": "5.0+" }, "db": { "vendor": "postgresql", "version": ">=14" }, "run": { "mode": "tests|script", "budgets\_file": "queryshield.yml" }, "tests": [ { "name": "app.tests.test\_views::test\_list\_view", "duration\_ms": 540, "queries\_total": 41, "queries\_p95\_ms": 18.2, "problems": [ { "type": "N+1", "evidence": { "cluster\_count": 41, "example\_sql": "SELECT ... WHERE ...", "stack": [["views.py","list",42], ...] }, "suggestion": { "kind": "select\_related", "args": ["author","category"] }, "explain": null } ], "queries": [ { "normalized\_sql": "SELECT ... WHERE id = ?", "duration\_ms": 2.4, "stack": [...], "tags": ["n+1\_cluster\_1"] } ] } ] }

2.6 Budgets YAML

budgets.py supports global and per‑test budgets:

defaults: max\_queries: 5 max\_total\_db\_time\_ms: 200 forbid: - type: "N+1" - type: "MISSING\_INDEX"

tests: "app.tests.test\_views::test\_list\_view": max\_queries: 3 max\_total\_db\_time\_ms: 120

2.7 Runners

Django: import settings via DJANGO\_SETTINGS\_MODULE, call django.setup(), and run tests using DiscoverRunner. Wrap the entire run in install\_probe(recorder). Django Project

pytest (optional): ship pytest\_plugin.py that installs the wrapper in pytest\_sessionstart; CLI passes -p queryshield\_probe.runners.pytest\_plugin. docs.pytest.org

1. Python CLI (queryshield)

CLI implemented with Typer or Click.

Commands

queryshield analyze --runner=django|pytest [--explain] [--budgets=queryshield.yml] [--output=.queryshield/queryshield\_report.json] Runs tests, writes report, prints a summary table, exits 0.

queryshield budget-check [--budgets=queryshield.yml] [--report=.queryshield/queryshield\_report.json] Fails (exit 2) if any budget is exceeded or forbidden problem appears.

queryshield record-baseline Saves current metrics as baseline for later diff.

queryshield verify-patch --baseline baseline.json --report current.json Fails if deltas < expected improvement thresholds.

1. JetBrains plugin (PyCharm / IntelliJ) 4.1 Project setup

Use the IntelliJ Platform Gradle Plugin 2.x (plugin id org.jetbrains.intellij.platform), JDK 17+, Gradle 8.5+. Add dependency on PyCharm and the Python plugin IDs as needed, and declare com.intellij.modules.python in plugin.xml. JetBrains Marketplace +1

build.gradle.kts (sketch)

plugins { id("org.jetbrains.intellij.platform") version "2.9.0" } repositories { mavenCentral(); intellijPlatform { defaultRepositories() } } dependencies { intellijPlatform { // target PyCharm Community; switch to PY for Pro pycharmCommunity("2025.2.3") bundledPlugin("PythonCore") // Python APIs (Community) bundledPlugin("com.intellij.json") // if you parse JSON in plugin } } intellijPlatform { // signing/publishing configured later // ... }

Note: JSON APIs were split to a separate module in recent builds; add com.intellij.modules.json to plugin.xml if you use JSON PSI. JetBrains Marketplace

plugin.xml (essentials)

io.queryshield QueryShield Your Name com.intellij.modules.python com.intellij.modules.json

See paid‑plugin product descriptor requirements (product code rules, release‑date/version semantics) here. JetBrains Marketplace

4.2 UI & UX

Inlay hints above pytest/Django test functions and DRF view methods showing: DB calls: 41 · p95: 18 ms · Problems: N+1. Implement with Inlay Hints API (nonintrusive, inline). JetBrains Marketplace

Tool Window “QueryShield”: shows last run summary, problem list, and a diff view (before/after). JetBrains Marketplace +1

Intentions/Quick‑fix (Alt+Enter): on N+1 reports, offer an IntentionAction that inserts a .select\_related(...) / .prefetch\_related(...) call into the nearest QuerySet chain (when resolvable), or inserts a snippet in the editor with a link to docs. JetBrains Marketplace +2 JetBrains Marketplace +2

Run action (“Analyze with QueryShield”): triggers CLI run and streams output into the Run Console using the Execution API. JetBrains Marketplace

4.3 Calling the CLI from the plugin

Resolve Python interpreter path:

Default: project’s configured Python SDK if available (Python plugin), else a user‑configurable path in plugin settings. JetBrains Marketplace

Run python -m queryshield\_cli analyze ... via IntelliJ Execution API and show a console with a progress indicator. JetBrains Marketplace

4.4 Licensing (paid plugin)

Add Marketplace license checks using JetBrains’ sample LicensingFacade code in plugin startup and before gated actions. The IntelliJ Platform performs built‑in daily license validation; your plugin should additionally check periodically and gate premium features. Test on the Marketplace Demo instance. JetBrains Marketplace +1

4.5 Publishing

Build: ./gradlew buildPlugin → ZIP.

First upload manually; later use Gradle publish with a Marketplace token. JetBrains Marketplace

1. IDE features — technical notes

Inlay hints: implement InlayHintsProvider for Python PSI elements representing test functions or APIView/viewset methods, reading latest report JSON to render counters. JetBrains Marketplace

Tool Window: register a ToolWindowFactory, show a table grouped by test; provide “Re‑run with budgets” button (calls CLI). JetBrains Marketplace +1

Intentions & Inspections:

Start with Intentions (no continuous highlighting burden). Use IntentionAction to propose code changes where a Django QuerySet call chain is PSI‑resolvable. Add a conservative fallback “copyable snippet” when AST context is ambiguous. JetBrains Marketplace +1

Add an optional Inspection (extends LocalInspectionTool) to flag obvious anti‑patterns found statically (e.g., .all() inside loops). JetBrains Marketplace

1. Heuristics — implementable details

Normalization:

Strip whitespace & comments; replace parameter literals with ?; collapse IN (?, ?, ?) to IN (?); store normalized\_sql.

Stack clustering:

Form a signature from top K frames (path relative to project root). Group repeated queries by (normalized\_sql, top\_frame).

N+1 (thresholds: ≥5, 70%+ identical normalized SQL; executed in same test/function):

Evidence = size of cluster, sample stack, earliest loop site (frame that repeats).

Fix = propose .select\_related() for FK/OneToOne detected from ORM metadata; else .prefetch\_related() (for M2M/reverse). Provide a snippet.

Full scan (Postgres):

Run EXPLAIN (FORMAT JSON): if plan root (or child) contains Seq Scan with Filter and estimated rows large (e.g., >10k or using table-size hints), mark MISSING\_INDEX; suggest index on equality predicates. PostgreSQL

Sort without index:

If plan contains Sort and no Index Scan on Sort Key (or index does not match prefix of where‑equalities + order keys), propose composite index.

SELECT \*:

If SELECT \* on tables with many columns or large estimate, suggest .only("..") or .values().

All EXPLAIN invocations in safe mode (no ANALYZE). Offer an opt‑in to EXPLAIN ANALYZE for local dev only. PostgreSQL

1. CI integration (headless)

Provide a GitHub Actions example:

name: QueryShield on: [pull\_request] jobs: analyze: runs-on: ubuntu-latest services: postgres: image: postgres:16 env: { POSTGRES\_PASSWORD: postgres, POSTGRES\_DB: appdb } ports: ["5432:5432"] options: >- --health-cmd="pg\_isready -U postgres" --health-interval=10s --health-timeout=5s --health-retries=5 steps: - uses: actions/checkout@v4 - uses: actions/setup-python@v5 with: { python-version: '3.11' } - run: pip install -e ./probe -e ./cli . - run: | export DJANGO\_SETTINGS\_MODULE=sample\_django\_app.settings queryshield analyze --runner=django --explain --budgets=queryshield.yml - run: queryshield budget-check --budgets=queryshield.yml

Exit non‑zero when budgets fail (blocks merge).

1. Developer environment & commands

Prereqs

Python 3.11+, PostgreSQL 14+, JDK 17, Gradle 8.5+, PyCharm or IntelliJ + Python plugin.

Local run

# Python

cd probe && pip install -e . cd ../cli && pip install -e .

# Sample app

cd ../sample-django-app docker compose up -d # starts Postgres with seed data export DJANGO\_SETTINGS\_MODULE=sample\_django\_app.settings queryshield analyze --runner=django --explain --budgets=../queryshield.yml

# IDE plugin

cd ../ide-plugin ./gradlew runIde # launches a sandbox IDE with plugin

Build & package plugin:

./gradlew buildPlugin # produces build/distributions/QueryShield-.zip

Publish later with a Marketplace token (first upload is manual). JetBrains Marketplace

1. Paid distribution (JetBrains Marketplace)

Request paid status; prepare plugin; you can test the purchase & license flow on the Demo marketplace. JetBrains Marketplace

In plugin.xml, add <product-descriptor ...> with product code, release-date, and release-version as required. JetBrains Marketplace

Add license verification calls using JetBrains’ reference LicensingFacade sample (2024.3+), and gate premium features (e.g., budgets & Explain). JetBrains Marketplace

Build & upload; later automate with Gradle publish. JetBrains Marketplace

1. Security & privacy defaults

No data leaves the machine.

Redact query params (show types/shape only).

Cap stored queries per test (e.g., 500) and truncate SQL length.

EXPLAIN runs without ANALYZE unless user opts in. PostgreSQL

1. Definition of Done (v1.0)

On sample-django-app, create an intentional N+1 (e.g., list view iterating related objects).

queryshield analyze detects N+1 and suggests select\_related('author','category').

After applying, total queries drop (e.g., 41 → ≤3).

Budgets file enforces ≤3 queries on that test; budget-check passes/fails accordingly.

Plugin shows inlay counters on the test and a Tool Window summary; an Intention appears near the offending QuerySet. JetBrains Marketplace +1

Packaging: buildPlugin succeeds; plugin runs in sandbox; license check gates a premium feature. JetBrains Marketplace

1. v1.1 (agent‑native, optional after v1.0 ships)

Headless JSON API: analyze, budget\_check, suggest\_fixes, verify\_patch.

PR comment bot: post minimal repro + suggested patch; rerun verify on apply.

Framework packs: add SQLAlchemy.

1. Implementation hints & gotchas

Django instrumentation: Prefer execute\_wrapper over connection.queries or monkey‑patches; it’s official and stable. Django Project

Postgres EXPLAIN: Use JSON format for machine consumption; keep timeouts; always parameterize. PostgreSQL

Plugin dependency: For PyCharm API availability, declare com.intellij.modules.python in plugin.xml and add the corresponding Python plugin via Gradle (PythonCore for Community or Pythonid for Pro). JetBrains Marketplace

Gradle plugin: Prefer the IntelliJ Platform Gradle Plugin 2.x (org.jetbrains.intellij.platform) going forward. JetBrains Marketplace

UI choices: Use Inlay Hints (lightweight) rather than heavy gutter renderers; use a Tool Window for detailed tables; use IntentionAction for code tweaks/scaffolds. JetBrains Marketplace +2 JetBrains Marketplace +2

Running the CLI: Use IntelliJ Execution API rather than Runtime.exec to get an integrated Run Console and proper lifecycle control. JetBrains Marketplace

1. Minimal code stubs (for orientation)

CLI entry

# queryshield\_cli/main.py

import sys, json, typer from queryshield\_probe.runners.django\_runner import run\_django\_tests from queryshield\_probe.report import write\_report from queryshield\_probe.budgets import load\_budgets, check\_budgets app = typer.Typer()

@app.command() def analyze(runner: str = "django", explain: bool = False, budgets: str = "queryshield.yml", output: str = ".queryshield/queryshield\_report.json"): results = run\_django\_tests(explain=explain) if runner=="django" else run\_pytest(explain=explain) write\_report(results, output) print(f"Wrote {output}")

@app.command("budget-check") def budget\_check(budgets: str, report: str = ".queryshield/queryshield\_report.json"): rules = load\_budgets(budgets) data = json.load(open(report)) violations = check\_budgets(rules, data) if violations: for v in violations: print(v) raise typer.Exit(code=2) print("Budgets OK")

if **name** == "**main**": app()

Plugin run action (outline)

// RunAnalysisAction.kt class RunAnalysisAction : AnAction("Analyze with QueryShield") { override fun actionPerformed(e: AnActionEvent) { val project = e.project ?: return val settings = QueryShieldSettings.getInstance(project) val cmd = GeneralCommandLine(settings.pythonPath, "-m", "queryshield\_cli", "analyze", "--runner=django", "--budgets=${settings.budgetsPath}") .withWorkDirectory(project.basePath) .withEnvironment(System.getenv()) val handler = OSProcessHandler(cmd) val console = TextConsoleBuilderFactory.getInstance().createBuilder(project).console console.attachToProcess(handler) handler.startNotify() } }

(Uses IntelliJ Execution API to run the CLI and stream output.) JetBrains Marketplace

1. QA checklist

Unit tests for each heuristic (feed canned EXPLAIN JSON with Seq Scan, Sort, etc.). PostgreSQL

Integration tests on sample-django-app:

Test proves 41→≤3 query drop with select\_related on list view.

Budgets fail when regression introduced.

Plugin UI smoke tests in sandbox:

Inlay appears on a test function.

Tool Window lists problems; double‑click navigates to top stack frame.

Intention inserts select\_related() in a resolvable QuerySet call.

Marketplace paid flow tested on Demo; license checks gate premium features. JetBrains Marketplace +1

References you’ll likely consult while building

Django DB instrumentation (execute\_wrapper). Django Project

PostgreSQL EXPLAIN & JSON format. PostgreSQL +1

IntelliJ Platform Gradle Plugin 2.x (setup, tasks). JetBrains Marketplace

PyCharm plugin dev & Python plugin dependency. JetBrains Marketplace

Inlay Hints / Tool Windows / Intentions (plugin UI). JetBrains Marketplace +2 JetBrains Marketplace +2

Publishing and paid licensing (product descriptor + license checks + Demo). JetBrains Marketplace +2 JetBrains Marketplace +2

If you follow the spec above, you’ll have a working v1: a Django‑aware probe, a CI‑friendly CLI, and a JetBrains plugin with meaningful, actionable UX—ready to submit to the Marketplace with paid licensing