# Applying Differential Privacy to DBMS Bug Reporting: A Reproducibility-Retention Study with Synthetic Relational Data

Master Thesis

Master Thesis

Damian Hrabąszcz

Aalborg University
CS-IT

**Dept. of Computer Science**
Selma Lagerløfsvej 300
DK-9220 Aalborg
https://www.cs.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

**Title:**
Applying Differential Privacy to DBMS Bug Reporting: A Reproducibility-Retention Study with Synthetic Relational Data

**Theme:**
Differential Privacy, Relational Synthetic data, DBMS Bug Reproducibility, Query Optimization, Performance Regression,

**Project Period:**
Summer Semester 2026

**Participant(s):**
Damian Hrabąszcz

**Supervisor(s):**
Søren Kejser Jensen

**Date of Completion:**
February 24, 2026

**Abstract:**

This thesis examines whether data dependent DBMS bug behavior can be retained after differentially private relational synthesis, so bugs can be reported without sharing raw production data. The method combines LLM assisted candidate discovery from public bug reports with schema constrained adaptation and deterministic execution validation in a MySQL harness. Adaptation follows acceptance criteria focused on trigger preservation, realistic schema implementation, and observability.
We evaluate two MySQL optimizer bugs, 111669 and 74602, on StackOverflow derived relational data synthesized with dp-relational. Results show synthetic reproduction is feasible but sensitive to configuration. For bug 111669, timeout asymmetry between fast component subqueries and slower combined predicates is retained across tested epsilon sweeps without post synthesis scaling in the default capped setup. For bug 74602, a fixed structure boundary sweep reproduces down to epsilon 1.8, and a separate operating point search identifies a practical reproducible setting.
We report supplementary robustness checks alongside primary execution based criteria and clarify the privacy boundary: formal differential privacy guarantees apply to synthesis stages, while data dependent wrapper steps outside synthesis limit strict end to end claims over raw inputs. These results support privacy aware bug reporting and motivate further automation of bug targeted synthesis configuration search.

# Thesis Summary

This thesis studies whether data dependent database bug behavior can be retained after differentially private relational synthesis, so bugs can be reported without sharing raw production data. The motivation is practical: bug reports often need realistic data characteristics to reproduce execution failures, but sharing production data can violate privacy and policy constraints. The work therefore focuses on a reproducibility-retention question: can synthetic data preserve bug triggering behavior well enough to support actionable reporting?

The methodology follows a two-stage pipeline. First, candidate bugs are identified and filtered from public reports using structured acceptance criteria. Second, selected cases are adapted into a reproducible MySQL harness with schema constrained synthesis and deterministic execution checks. The adaptation process prioritizes trigger preservation, realistic schema level implementation, and measurable observability. This is intended to make results both auditable and repeatable across runs.

The evaluation centers on two MySQL optimizer bugs, 111669 and 74602, using StackOverflow-derived relational source data and corresponding synthetic datasets produced with dp-relational. Experiments sweep configuration settings, including privacy and workload parameters, and compare source-data and synthetic-data outcomes under matched execution conditions. The thesis reports both primary reproduction outcomes and supplementary robustness checks so that retention results are interpreted with clear operational context.

Results show that synthetic-data bug reproduction is feasible, but strongly configuration dependent. For bug 111669, the key timeout asymmetry between fast component subqueries and slower combined predicates is retained across tested epsilon sweeps in the default capped setup, even without post synthesis scaling. For bug 74602, a fixed-structure boundary sweep retains reproduction down to epsilon 1.8, then fails at lower tested points, while a separate operating-point search identifies a practical reproducible setting.

The thesis also clarifies the privacy accounting boundary. Formal differential privacy guarantees apply to synthesis stages, while data dependent wrapper steps outside synthesis limit strict end-to-end claims over raw input data. Taken together, the findings provide a reproducible basis for privacy aware bug reporting workflows and motivate future automation of bug targeted synthesis configuration search.

# Applying Differential Privacy to DBMS Bug Reporting: A Reproducibility-Retention Study with Synthetic Relational Data

Damian Hrabąszcz
Aalborg University
Aalborg, Denmark
dhraba23@student.aau.dk

## Abstract

This thesis examines whether data dependent DBMS bug behavior can be retained after differentially private relational synthesis, so bugs can be reported without sharing raw production data. The method combines LLM assisted candidate discovery from public bug reports with schema constrained adaptation and deterministic execution validation in a MySQL harness. Adaptation follows acceptance criteria focused on trigger preservation, realistic schema implementation, and observability.

We evaluate two MySQL optimizer bugs, 111669 and 74602, on StackOverflow derived relational data synthesized with dp-relational. Results show synthetic reproduction is feasible but sensitive to configuration. For bug 111669, timeout asymmetry between fast component subqueries and slower combined predicates is retained across tested epsilon sweeps without post synthesis scaling in the default capped setup. For bug 74602, a fixed structure boundary sweep reproduces down to epsilon 1.8, and a separate operating point search identifies a practical reproducible setting.

We report supplementary robustness checks alongside primary execution based criteria and clarify the privacy boundary: formal differential privacy guarantees apply to synthesis stages, while data dependent wrapper steps outside synthesis limit strict end to end claims over raw inputs. These results support privacy aware bug reporting and motivate further automation of bug targeted synthesis configuration search.

## Keywords

differential privacy, relational synthetic data, DBMS bug reproducibility, query optimization, performance regression

## 1 Introduction

Many Database Management System (DBMS) bugs are data-dependent. Whether a bug appears often depends on cardinality, skew, correlation, and join structure rather than query text alone. Reproducing such bugs therefore requires realistic relational data at non-trivial scale. At the same time, sharing production-like data for bug reproduction creates confidentiality risk, because reports and replay artifacts can expose sensitive values, derived attributes, or linkage cues.

Simple identifier removal is not sufficient in this setting. Foundational disclosure-control work frames anonymity as contextual, linkage-driven, and probabilistic rather than binary [25]. As that work emphasizes, de-identified data are not automatically anonymous: "The term anonymous implies that the data cannot be manipulated or linked to identify an individual." [25] This framing matters for bug-report workflows, where disclosure artifacts range from schema hints, filtered results, and traces to full row-level dumps; individually or in combination, these artifacts can enable re-identification.

This thesis studies whether differential privacy (DP) can support reproducibility-retention: preserving bug-triggering behavior while reducing data-leakage risk. Concretely, we evaluate bug-targeted DP-relational synthesis on StackOverflow-derived relational data and test whether known MySQL bug signatures remain reproducible across privacy budgets and scale factors.

### 1.1 Privacy Risks

Bug-report workflows create multiple privacy risks. This thesis directly mitigates data-value leakage from report artifacts. Schema-structure and presence leakage are discussed as contextual risks in the current scope.

Data leakage is the primary mitigation target. Users or developers may include production values in `INSERT`/`UPDATE` statements or expose `SELECT` outputs, logs, traces, and diagnostics. These artifacts can reveal personally identifiable information (PII), financial information, or proprietary content. Reducing this leakage is the core privacy objective of the workflow, including bug-specific adaptation and tuning of DP-relational drivers for #111669, #74602, and related cases; formal DP guarantees in this thesis are scoped to the synthesis stages.

Schema-structure leakage is contextual in this thesis. Even without raw values, table organization, key constraints, and index choices can expose sensitive business or system information. We acknowledge this risk but do not claim a separate formal mitigation mechanism beyond artifact minimization.

Presence leakage is also contextual. The mere presence of specific tables, fields, or records can disclose sensitive facts. For example, evidence that a known identifier appears in a released artifact may leak information even when values are masked.

### 1.2 Re-identification and Reconstruction Attacks

Quasi-identifiers (QIDs) are attributes that are not unique on their own but can be linked with other QIDs or external datasets to re-identify individuals. Typical examples include date of birth, ZIP code, and gender. Individually these values may appear harmless, but in small or homogeneous populations their combination can uniquely identify records.

While k-anonymity-style methods (and related models such as l-diversity and t-closeness) are relevant for quasi-identifier risk reduction [17, 18, 26], they are not the formal privacy basis of this thesis. We use differential privacy (DP) as the primary guarantee because it is composable and robust to auxiliary information under the $(\epsilon, \delta)$ framework.

This thesis assumes the central-DP model (trusted curator) rather than local DP, since relational synthetic-data generation for bug reproducibility requires preserving cross-table structure and optimizer-sensitive distributions that are typically degraded under local perturbation.

Historical evidence shows that de-identification alone is often insufficient against linkage and background-knowledge attacks. In high-dimensional preference data, the Netflix Prize release was shown to be vulnerable to robust de-anonymization [20], while linkage-style re-identification risks are also documented in public health microdata release practice (including the Massachusetts hospital discharge context) [1]. Related re-identification concerns have likewise been demonstrated for genomic data [19]. These examples motivate using DP as the formal privacy mechanism in this thesis.

## 1.3 Research Gap and Opportunity

Existing DP-synthesis evaluations typically emphasize aggregate utility metrics or generic workload fidelity, while DBMS bug reproduction requires behavior-level retention of specific failure signatures (e.g., timeout asymmetry, index-path sensitivity, or semi-join pathologies). The methodological gap is therefore not only privacy-preserving synthesis, but synthesis configuration and validation under bug-specific criteria. In current practice, this process remains expert-driven: manual candidate filtering, schema adaptation, and iterative hyperparameter search under memory/runtime constraints. This thesis addresses that gap with a reproducible, criteria-gated workflow and explicitly logged campaign decisions.

## 1.4 Thesis Focus and Scope

This thesis investigates the following central question: *to what extent can known, data-dependent DBMS bug signatures be retained after DP-relational synthesis across privacy budgets and data scales?* We operationalize this question through deterministic source-vs-synthetic replay, bug-specific acceptance criteria, and campaign-level analysis of reproduction boundaries and operating-point trade-offs. The formal research questions are listed in Section 3.3.

## 1.5 Contributions

- A scope-defined threat model for bug-report privacy, distinguishing primary mitigation targets (data-value leakage) from contextual risks (schema and presence leakage).
- A two-stage candidate-to-benchmark pipeline with explicit Stage-2 adaptation criteria (trigger preservation, deterministic observability, subsystem compatibility, and per-bug granularity).
- Reproducibility evidence for two MySQL bug families (#111669 and #74602), including source/synthetic outcomes, epsilon-boundary analysis, and campaign-level final selected operating-point decisions.
- A reproducibility package with run metadata, configuration IDs, and iterative experiment matrices to support auditability and extension.

## 1.6 Thesis Structure

The remainder of this thesis is organized as follows. The Background section summarizes differential privacy and relational synthesis context. The Methodology and Benchmark Design section defines dataset selection, candidate filtering, adaptation criteria, and execution protocol. The Results section reports source/synthetic reproduction outcomes and campaign analyses for MySQL #111669 and #74602. The Discussion and Threats to Validity section covers internal, construct, external, and conclusion validity limits. The Conclusion section summarizes findings, and the Limitations and Future Work section outlines key constraints and next-step directions. Appendices provide detailed artifacts, prompt transcript, and configuration tables.

# 2 Background

## 2.1 Differential Privacy

Differential privacy (DP) is a formal privacy framework that bounds the influence of any single record on a released output. Intuitively, for neighboring datasets $D$ and $D'$ that differ in one individual's record, a randomized mechanism $M$ is differentially private if the output distributions induced by $M(D)$ and $M(D')$ remain close. This guarantee is parameterized by $(\epsilon, \delta)$, where smaller values correspond to stronger privacy.

Historically, DP was introduced in statistical database settings, especially for privacy-preserving query release (including counting and other aggregate-style analyses), and was later generalized to broader analysis workflows [9]. More recent research on DP synthetic data generation, including relational synthesis methods, expands the design space from releasing individual noisy answers to releasing privacy-protected datasets that can support downstream tasks [2, 11]. This shift opens applications such as the one studied in this thesis: privacy-aware DBMS bug reporting where execution-level bug behavior must remain testable without sharing raw production data.

In this thesis, DP is central because bug-reproduction artifacts (queries, execution traces, and supporting data extracts) can leak information even after direct identifiers are removed. DP provides a principled guarantee that limits this leakage while preserving enough statistical utility for downstream analysis and experimentation.

Two properties of DP are especially relevant to our workflow. First, post-processing invariance: once a DP output is released, downstream transformations and bug-reproduction runs on that synthetic output do not consume additional privacy budget. Second, composition: privacy loss accumulates across all DP mechanisms that access raw source data during synthesis (e.g., single-table synthesis and relational learning), so budget allocation across those synthesis stages must be explicit.

## 2.2 DP Synthesis for Relational Databases

Most traditional DP synthesizers target a single table. However, real DBMS workloads operate over relational schemas with foreign keys, join paths, and cross-table correlations. Flattening relational data can lose structural signals that are critical for realistic optimizer behavior and query-plan selection [2].

Because our objective is bug reproducibility (not only aggregate accuracy), preserving relational structure is essential. In particular, key/foreign-key distributions, relationship cardinalities, and cross-table dependency patterns can directly affect whether a bug trigger remains observable after synthesis.

## 2.3 DP-Relational and Kamino

**DP-relational** introduces a framework for differentially private synthetic relational databases by combining table-level synthesis with iterative relational refinement to improve cross-table consistency and low-order marginal fidelity while maintaining referential integrity [2, 3]. This aligns well with our thesis requirements, where join behavior and relationship structure are part of the bug-trigger mechanism.

**Kamino** is a constraint-aware DP synthesis system that incorporates schema and integrity constraints during synthesis to better preserve structural properties [10, 11]. This is relevant for bug reproduction because many DBMS failures and performance pathologies are structure-sensitive rather than purely value-distribution-sensitive.

Accordingly, this thesis uses DP-relational as the synthesis approach in the executed experiments. Kamino is retained as a potential comparative/stretch baseline for future evaluation, but was not executed in this thesis phase due to available time. The experimental chapters therefore evaluate whether DP-relational preserves behavior-level signatures needed to reproduce known DBMS bugs at realistic scales.

## 2.4 Related Work and Positioning

Related work most directly connected to this thesis spans four lines. First, DP data release and synthesis work defines the formal privacy basis and evaluation framing [9, 23]. Second, relational DP synthesis methods address foreign-key structure and cross-table consistency under privacy constraints, including PrivLava and PrivPetal, and more recently DP-relational/Kamino-style approaches [2, 5, 6, 11].

Third, DBMS bug-testing research provides strong automation for bug discovery and diagnosis (e.g., optimization-bug detection and query-partitioning-based testing), but these methods are not designed as privacy-preserving bug-report-data release pipelines [15, 21, 22]. In particular, APOLLO explicitly identifies confidential dataset sharing as an important unresolved issue and points to differential privacy as a promising future direction rather than an implemented solution [15].

Fourth, adjacent software-engineering privacy studies (e.g., privacy-preserving bug severity prediction) show that privacy constraints are operationally important in bug-related workflows, but focus on predictive modeling tasks rather than execution-level DBMS bug reproducibility [7].

For context, we use SQLStorm-derived StackOverflow splits as a realistic relational data substrate, but our objective differs from benchmark generation: we evaluate retention of known bug-trigger behavior under DP synthesis [24].

In the literature reviewed for this thesis, we are not aware of prior work that directly studies privacy-aware DBMS bug reporting via differentially private relational synthesis with execution-level reproducibility criteria and campaign-level operating-point selection. We therefore position this thesis as an initial, scoped case-study contribution at that boundary.

## 3 Methodology and Benchmark Design

For each bug and dataset size (1GB, 12GB, 222GB), we first execute the source-data reproduction harness; synthetic reproduction at a given size is attempted only when source reproduction at the same size succeeds, and each synthetic run is linked to a bug-specific dp-relational configuration ID (Table 4, Appendix Table 10).

## 3.1 Hardware Configuration

To improve reproducibility and make resource assumptions explicit, we separate the hardware setup into three execution contexts. Fill in the fields below with the exact machine and runtime details used in your experiments.

*3.1.1 Synthesis Hardware (DP-relational).* Experiments were conducted on a Vast.ai rented GPU instance using the PyTorch (Vast) template.

- CPU: AMD Ryzen Threadripper PRO 5955WX (16 cores)
- GPU: 1× NVIDIA RTX PRO 6000 Blackwell Workstation Edition
- RAM: 257.6 GB (provider-reported)
- Storage: 300 GB
- OS and runtime stack: Ubuntu 22.04.3 LTS (Jammy Jellyfish), Docker (Vast PyTorch template), CUDA 13.0, PyTorch 2.10.0+cu128

*3.1.2 Source-Data Bug Reproduction Hardware.*

- CPU: 8-vCPU VM class (Google Cloud `n2-highmem-8` or AWS `i4i.2xlarge`)
- RAM: approximately 64 GB
- Storage: local NVMe-backed lab storage (GCE: 4×375 GB Local SSD + 50 GB boot disk; AWS: local NVMe instance store + 64 GB encrypted gp3 root volume)
- DBMS and execution environment: containerized MySQL via Docker Compose (`db1`/`db2`); MySQL 8.0.x images
- OS and runtime stack: Ubuntu 22.04 LTS (Jammy) VM images (GCE and AWS), Docker Compose

*3.1.3 Synthetic-Data Bug Reproduction Hardware.*

- CPU: Intel Core Ultra 7 255H (16 vCPUs)
- RAM: 30 GiB
- Storage: 476.9 GB NVMe SSD
- DBMS and execution environment: containerized MySQL via Docker Compose (`db1`/`db2`) and bug-specific load/reproduction scripts
- OS and runtime stack: Ubuntu 24.04.4 LTS (Noble Numbat), Docker Compose

## 3.2 Dataset and Scale Selection Rationale

We use the StackOverflow schema and public data dumps released by the SQLStorm authors because they provide a realistic, production-like relational setting while remaining fully reproducible in an academic workflow.[1] The schema is multi-table, join-centric, and large enough to expose optimizer behavior that is often invisible in toy datasets. This is important for our objective, which is behavior-level bug reproduction rather than only aggregate statistical fidelity.

At a high level, the StackOverflow dumps are composed of core interaction tables (`Posts`, `PostHistory`, `Comments`, `Votes`)

[1] `https://github.com/SQL-Storm/SQLStorm/tree/schmidt/v2.0`

and supporting entity/link tables (e.g., `Users`, `Badges`, `Tags`, and `PostLinks`). In this thesis, different bug cases exercise different table subsets: bug #111669 primarily uses `Posts–PostHistory`, while bug #74602 is reproduced on source data using `Votes`. This case-dependent table usage is expected to extend to additional bugs, while the broader schema context remains relevant for cross-case benchmark realism and transferability.

We specifically use three scale points that are already standardized in that ecosystem:

- 1GB: StackOverflow DBA split;
- 12GB: StackOverflow Math split;
- 222GB: full StackOverflow split.

This gives two advantages: (i) consistent cross-scale comparisons under the same schema, and (ii) direct alignment with a publicly documented benchmark pipeline [24]. In other words, scale variation in this thesis reflects controlled data growth, not schema drift.

Importantly, as noted by the SQLStorm authors, these StackOverflow splits are not perfectly linear rescalings [24]. Different relation groups grow at different rates across scales (e.g., users vs. posts/-comments), which changes both cardinalities and distributions. For this thesis, that property is desirable: optimizer behavior and bug triggers are often sensitive to non-uniform growth and shifting selectivity, so evaluating reproducibility across these splits is closer to real-world workload evolution than synthetic linear upsampling.

Alternative public datasets (e.g., IMDb non-commercial datasets [14], MovieLens [12], and Netflix Prize-derived data [4]) are relevant but were deferred. In this thesis phase, we prioritized one schema family with established multi-scale dumps and existing harness compatibility, to reduce confounding factors from schema translation and dataset-specific preprocessing. Extending the same bug-reproducibility protocol to additional public schemas is a clear next step.

## 3.3 Research Questions

To guide this study, the following research question has been formulated:

| | |
|---|---|
| RQ1 | To what extent do selected data-dependent DBMS bug-triggering query patterns remain reproducible after DP-relational synthesis across epsilon values? |

In this thesis phase, RQ1 is evaluated on two selected MySQL optimizer bugs (#111669 and #74602).

## 3.4 Stage 1: LLM-Assisted Candidate Discovery

We used LLM-assisted discovery to identify an initial pool of DBMS bug candidates for reproducibility experiments on StackOverflow-derived datasets.

*Candidate bug discovery.* Using web-based LLM interfaces (ChatGPT and Grok), we prompted for confirmed relational DBMS bugs with the following constraints: (i) confirmed in official trackers or equivalent public channels, (ii) data-dependent behavior, (iii) non-toy or realistic data scale, and (iv) available reproduction details. We explicitly requested both correctness bugs and performance bugs with equal priority, while preferring MySQL and PostgreSQL.

The exact prompt is reproduced in Appendix B.5, Prompt 1.

*ChatGPT discovery output (candidate pool).* Table 1 summarizes the bug candidates returned by ChatGPT in this discovery step (conversation log used in this thesis). At this stage, all items are treated as candidate hypotheses, not accepted benchmark cases. We also obtained a candidate list from Grok using a comparable prompt; we did not focus on it as much as the ChatGPT output in this thesis, but include it in the appendix (Table 11) as it may support future research.

We then manually filtered this pool by requiring tracker-level confirmation quality, compatibility with our StackOverflow-schema adaptation workflow, and deterministic replay feasibility under our time/resource constraints. This filtering retained MySQL #111669 and MySQL #74602 for execution in the present thesis phase.

## 3.5 Stage 2: Schema-Constrained Adaptation and Validation

*Schema-constrained adaptation.* For selected candidates, we adapted reproduction queries to our StackOverflow schema, allowing realistic index additions when necessary. We used available scale factors (1GB, 12GB, 222GB) and table/index statistics to constrain each adaptation toward plausible production conditions.

The operational adaptation requirements are formalized below as the Stage-2 acceptance gate.

*Stage-2 adaptation criteria (acceptance gate).* To move from a discovered candidate to an executable benchmark case, an adapted case had to satisfy all of the following:

- **Trigger preservation:** preserve the original bug mechanism (failure condition), not merely the original query text.
- **Schema-realistic implementation:** map the case to StackOverflow tables with only production-plausible index changes, each with explicit justification.
- **Minimal reproducible package:** provide runnable setup (DDL/index steps if needed), query under test, expected observable symptom, and optional control/forced-plan variant.
- **Deterministic observability:** symptom must be repeatedly verifiable in our harness with measurable evidence (e.g., runtime/timeout, EXPLAIN shape, counters, or result mismatch).
- **Subsystem compatibility:** trigger must be reproducible in our single-node query-execution harness; cases requiring replication, logical decoding pipelines, failover orchestration, or distributed-cluster state are out of scope in this thesis phase.
- **Explicit mismatch handling:** if an exact mapping is impossible, the proxy and mismatch must be explicit; if the mismatch weakens causal interpretation, the case is excluded.
- **Per-bug granularity:** each accepted case is specified and validated independently per bug ID.

PostgreSQL #5673 was excluded at this gate because we could not construct a StackOverflow-schema adaptation that simultaneously preserved the reported trigger semantics and yielded a deterministic minimal symptom package without a substantial proxy mismatch. PostgreSQL #13725 was excluded at this gate under subsystem compatibility because its trigger relies on logical decoding/replication behavior outside the single-node harness scope.

**Table 1: ChatGPT-assisted candidate bug list from Stage-1 discovery.**

| DBMS | Bug ID | Class | Data-dependent trigger summary from ChatGPT output | Status in this thesis |
|---|---|---|---|---|
| MySQL | #111669 | Performance | Two individually fast `IN` subqueries (fulltext predicates) become slow/-timeout when combined with `AND`, attributed to semijoin/plan-choice behavior on larger history data. | Selected and adapted |
| MySQL | #74602 | Performance | `ORDER BY ... LIMIT` (`low_limit`) case where optimizer may prefer sort-order index over a more selective predicate index, causing large unnecessary scans. | Selected and adapted |
| MySQL | #36513 | Performance | Large-table InnoDB statistics under heavy skew reported severe cardinality misestimation and resulting poor index choice. | Candidate only |
| MySQL | #91139 | Performance | Large `IN (...)` lists reported planning-time overhead growth from many index dives under default optimizer settings. | Candidate only |
| MySQL | #102037 | Performance | Large `IN (...)` lists reported excess CPU overhead per list element during execution in MySQL 8.0.22. | Candidate only |
| MySQL | #40974 | Correctness | Complex query plans (including `NOT EXISTS` paths) reportedly returned different result sets depending on index presence. | Candidate only |
| MySQL | #105975 | Correctness | DuplicateWeedout/semijoin strategy on a specific self-join data pattern reportedly produced incorrect row counts. | Candidate only |
| MySQL | #21059 | Crash | Joins on very large NDB tables reported server crash behavior. | Candidate only |
| PostgreSQL | #5673 | Correctness | Planner/index-condition interactions (e.g., PK presence) reportedly produced duplicate result rows. | Excluded by Stage-2 adaptation criteria |
| PostgreSQL | #13725 | Correctness | Logical decoding on very large transactions reportedly repeated or replayed rows in output. | Excluded by Stage-2 adaptation criteria |
| PostgreSQL | #16905 | Performance | Large-table runtime regression after table/index recreation was linked to statistics sampling and plan-choice sensitivity. | Candidate only |
| PostgreSQL | #16993 | Performance | Extreme skew (near-constant join key) reportedly led the planner toward costly sequential-scan behavior. | Candidate only |

*Execution and validation protocol.* Each adapted case was treated as a hypothesis and accepted only after deterministic validation in MySQL using scripted harnesses. For each run, we logged:

- bug ID, DBMS version, and query variant;
- dataset scale and synthetic-generation configuration;
- runtime and timeout outcomes;
- optimizer-relevant evidence (e.g., EXPLAIN output, Handler_read_next, Sort counters);
- reproduction label (reproduced / partially reproduced / not reproduced).

*DP reproducibility experiments.* We evaluated whether bug-triggering behavior is preserved after DP-relational synthesis across epsilon settings and scale factors. For bug #111669, reproduction evidence includes preserved asymmetry between fast subqueries and failing/timeout combined predicates under a fixed execution-time budget. For bug #74602, evidence includes index-choice-sensitive runtime and counter behavior under ORDER BY ... LIMIT conditions.

## 3.6 Synthetic Experiment Design and Configuration

*From baseline method to bug-focused drivers.* For bug #111669, we used a dedicated `dp-relational` experiment driver (`experiment_pgd_stackoverflow_bug111669.py`) adapted from the project's reference experiment scripts. We retained the original three-stage synthesis structure: (i) DP single-table synthesis (`aim` or `mst`), (ii) cross-table workload construction with `QueryManagerTorch`, and (iii) Projected Gradient Descent (PGD)-based relationship learning for the synthetic linking table. Our adaptation is therefore implementation-focused (schema and bug signal targeting), not a change to the underlying relational DP optimization procedure. The adaptation targets the fulltext interaction signal on `PostHistory–Posts` (fast individual subqueries versus slow combined predicates) by preserving token-bearing overlap structure under the same DP-relational training workflow.

For bug #74602, we use an analogous dedicated driver (`experiment_pgd_stackoverflow_bug74602.py`) with the same backend synthesis pipeline and optimizer.

The adaptation targets the low-`LIMIT` access-path signal on `Votes` (default vs forced selective-index behavior) by preserving the relational shape between `Votes` and `Posts` under the same DP-relational training workflow.

*Attribution boundary.* In this work, `dp-relational` is treated as a fixed synthesis backend. We do not re-derive or modify its optimization objective or convergence guarantees. Accordingly, equations in this section formalize only our experimental protocol (run gating, configuration identity, and implemented budget split).

*Bug-specific adaptation choices.* For bug #111669, the script is specialized to the minimal `PostHistory–Posts` setting used by the reproduction harness. Instead of preserving raw free text directly, it preserves bug-relevant signals through engineered features (e.g., token indicators for `Text`/`Comment`, SQL-token indicators, and length buckets), then exports minimal MySQL-loadable CSV files (`Posts.csv`, `PostHistory.csv`) for deterministic replay.

Preprocessing caps are exposed to control memory/runtime on large sources while preserving token-bearing rows. `max-posthistory-rows` caps how many source `PostHistory` rows enter synthesis; in our loader, token-bearing rows are retained first and background rows are sampled second. `max-posts-rows` caps source `Posts` and keeps posts with richer history coverage, then filters `PostHistory` to those post IDs.

For bug #74602, the script is specialized to a minimal `Votes–Posts` setting aligned with the reproduction harness. It preserves the bug-relevant selector signal around `PostId`/`VoteTypeId` and exports deterministic replay artifacts (`Votes.csv`, `Posts.csv`). Here, `max-votes-rows` and `max-posts-rows` define the source subset used to fit marginals and relational structure before synthesis; therefore, as with #111669, they are source-selection controls rather than post-synthesis scaling knobs.

*Implementation blocker and remediation.* During early bug #74602 synthesis runs, we encountered a runtime failure in `dp-relational` during `QueryManager.get_offsets()`. Although `PostId` is categorical after preprocessing, synthesized table outputs could emit category-coded values as `float64` (e.g., 769.0) rather than integer dtype. Before remediation, offsets were accumulated directly into an integer array, triggering NumPy's in-place casting error (`UFuncTypeError: Cannot cast ufunc 'add' output from dtype('float64') to dtype('int64') with casting rule 'same_kind'`). We fixed this by normalizing each synthesized category column before offset accumulation: integer dtypes are cast to `np.int_`, float dtypes are rounded then cast, and values are clipped to the valid per-dimension domain $[0, \text{dim\_count}-1]$. This removed the crash and hardened workload evaluation against out-of-domain synthetic category values. The exact patch artifact is included in Appendix B.3 (Listing 5).

*Execution gating rule.* A synthetic run for bug $b$, scale $s$, and configuration $c$ is attempted only if source-data reproduction succeeds at the same $(b, s)$:

$$1_{\text{synth}}(b, s, c) = 1\{\text{source\_ok}(b, s) = 1\}.$$

*Parameter groups and interpretation.* At run time, parameters are grouped as follows:

- **Data I/O and bug signal:** `–source-dir`, `–output-dir`, and bug-signal selector (`–token` for #111669; `–target-votetype` for #74602).
- **Privacy and single-table synthesis:** `–epsilon`, `–synth`.
- **Synthetic table sizes and structure:** bug-specific primary size knob (`–n-syn-posthistory` for #111669 or `–n-syn-votes` for #74602), plus shared size/structure knobs (`–n-syn-posts`, `–dmax`, `–k`).
- **Relational PGD controls:**
  - `–T`, `–subtable-size`, `–queries-to-reuse`
  - `–k-new-queries`, `–exp-alpha`, `–guaranteed-rels`
- **Execution and reproducibility:**
  - `–device`, `–seed`
  - preprocessing caps (`–max-posthistory-rows` for #111669 or `–max-votes-rows` for #74602), and `–max-posts-rows`

**Table 2: Interpretive scale for selected $\epsilon$ values ($e^\epsilon$).**

| $\epsilon$ | $e^\epsilon$ |
|---|---|
| 0.1 | 1.11 |
| 0.5 | 1.65 |
| 1 | 2.72 |
| 2 | 7.39 |
| 3 | 20.09 |
| 6 | 403.43 |

*Privacy-budget split used in experiments.* The script applies a fixed split:

$$\epsilon_1 = \epsilon_2 = \frac{\epsilon}{3}, \qquad \epsilon_{\text{rel}} = \epsilon - \epsilon_1 - \epsilon_2 = \frac{\epsilon}{3},$$

where $\epsilon_1$ and $\epsilon_2$ are allocated to single-table synthesis and $\epsilon_{\text{rel}}$ to relational learning. In this thesis, this accounting boundary is attached to the synthesis stages. Bug-specific preprocessing and postprocessing (e.g., source capping, subset selection, bucketization, and reconstruction heuristics) are data-dependent wrapper steps and are not presented as part of a formal end-to-end DP proof over raw source data.

*Configuration traceability.* Each run is represented by

$$r_j = (b, s, \epsilon, c, \texttt{seed}_j), \qquad j = 1, \ldots, N,$$

where $b$ is bug ID, $s$ is dataset scale, $\epsilon$ is total privacy budget, $c$ is the bug-specific configuration ID, and $\texttt{seed}_j$ is the random seed of repetition $j$. Each synthetic run is tied to this configuration identity (Table 4, Appendix Table 10), enabling direct comparison across scales, epsilon settings, and synthesis strategies.

*Reporting strategy across iterative-design subsections.* In this chapter, we report protocol intent, decision criteria, and representative turning points. Exhaustive per-run details (all attempted configurations, seeds, statuses, and metrics) are maintained in iterative experiment matrix artifacts, which serve as traceability ledgers for each bug case; campaign labels are introduced in the Results section for consistency across bug cases.

## 3.7 Epsilon Interpretation and Bug-Reproduction Sweep Design

Differential privacy (DP) provides a multiplicative bound on how much output probabilities can change between neighboring datasets. For any output event $S$:

$$\Pr[M(D) \in S] \leq e^\epsilon \Pr[M(D') \in S].$$

Thus, $\epsilon$ directly controls the privacy-utility tradeoff: smaller $\epsilon$ implies stronger privacy and typically lower utility, while larger $\epsilon$ weakens privacy and typically improves utility [9]. For interpretability, Table 2 reports $e^\epsilon$, the multiplicative probability-ratio bound from the $\epsilon$-term (exact when $\delta = 0$; under $(\epsilon, \delta)$-DP, an additive $\delta$ slack also applies).

Choosing $\epsilon$ is known to be application-dependent; there is no universal consensus for a single "correct" value across tasks and domains [8, 13]. Accordingly, we treat $\epsilon$-selection as an empirical

**Table 3: Working $\epsilon$-range hypothesis used to initialize the experiment sweep.**

| Bug type | Initial $\epsilon$ range |
|---|---|
| Performance regression at scale | 1–2 |
| Optimizer chooses poor plan | 2–3 |
| Cardinality-estimation-sensitive bug | 3–4 |
| Exact-value pattern dependence | 6+ |

model-selection step tied to bug-reproduction objectives, while keeping the privacy interpretation explicit.

Our post-data-synthesis scaling step is handled under the same accounting logic. This is in line with the guarantees stated by the DP-relational authors: once the synthetic relational database is released as the output of the DP mechanism, any downstream transformation or querying that uses only this released output (e.g., post-synthesis upscaling for stress/reproduction runs) is post-processing and does not consume additional privacy budget, provided there is no renewed access to the original private data [2]. However, this does not imply utility preservation: post-synthesis scaling can change cardinalities, selectivities, and runtime behavior, so we treat it as a utility/reproducibility control and report it separately from $\epsilon$.

We hypothesize that bug classes requiring finer-grained value fidelity may need larger $\epsilon$ than bugs primarily driven by aggregate cardinalities. Motivated by prior evidence that cardinality/estimation errors strongly affect optimizer behavior [16], we use the bug-type ranges in Table 3 as an initial sweep heuristic rather than as established theory.

In this thesis, we treat $\epsilon \in [1, 3]$ as the preferred evaluation region. When needed to establish a reproducible anchor or stable operating profile, we temporarily escalate to higher $\epsilon$ values. We then report both the minimum reproducible $\epsilon$ and the final operating $\epsilon$ selected under reproducibility and utility constraints, preserving comparability across bug cases and scales.

## 3.8 Benchmark Case Specification

### 3.8.1 *MySQL Bug #111669.*

Bug #111669 is an optimizer-related performance issue in MySQL where two individually fast `MATCH ... AGAINST` subqueries become unexpectedly slow when combined with `AND` through `IN` predicates in an outer query. The bug was later marked Verified by the MySQL verification team.[2]

*Official query pattern (source report).* The original report uses a `person_main` outer table and two fulltext-filtered subqueries on `person_history_work`:

**Listing 1: Official bug pattern (simplified) from Bug #111669**

```sql
SELECT pm.ID
FROM person_main pm
WHERE pm.ID IN (
  SELECT phw.person_main_ref_id
  FROM person_history_work phw
  WHERE MATCH(phw.work_summary)
        AGAINST('finance')
)
AND pm.ID IN (
  SELECT phw.person_main_ref_id
  FROM person_history_work phw
  WHERE MATCH(phw.work_title)
        AGAINST('software')
);
```

[2]https://bugs.mysql.com/bug.php?id=111669

*Adapted StackOverflow-schema trigger query.* For our benchmark, we map the same logical pattern onto `Posts` and `PostHistory`, using two fulltext predicates over different text fields, then intersecting their ID sets:

**Listing 2: Adapted Bug #111669 query used in our benchmark (`Posts`/`PostHistory`)**

```sql
-- each subquery is fast on its own
SELECT DISTINCT PostId
FROM PostHistory
WHERE MATCH(Text) AGAINST('error' IN BOOLEAN MODE);

SELECT DISTINCT PostId
FROM PostHistory
WHERE MATCH(Comment) AGAINST('error' IN BOOLEAN MODE);

-- combined form can time out
SET SESSION max_execution_time = 60000;

SELECT p.Id
FROM Posts p
WHERE p.Id IN (
  SELECT DISTINCT ph.PostId
  FROM PostHistory ph
  WHERE MATCH(ph.Text)
        AGAINST('error' IN BOOLEAN MODE)
)
AND p.Id IN (
  SELECT DISTINCT ph.PostId
  FROM PostHistory ph
  WHERE MATCH(ph.Comment)
        AGAINST('error' IN BOOLEAN MODE)
);
```

*Reproduction criterion in this thesis.* We consider the case reproduced when both component subqueries complete within the time budget, while the combined `AND`-intersection form exhibits severe slowdown or timeout under the same budget. This criterion preserves the behavioral asymmetry reported in the official bug thread.

*Auxiliary support metric for #111669 (`OverlapDistinct`).* In addition to runtime/timeout outcomes, we log an intersection-support metric over `PostHistory`:

$$\texttt{OverlapDistinct} = |A \cap B|,$$

where $A$ is the set of distinct `PostId` values matched by `MATCH(Text) AGAINST('error' IN BOOLEAN MODE)` and $B$ is the set of distinct `PostId` values matched by `MATCH(Comment) AGAINST('error' IN BOOLEAN MODE)`. This metric indicates whether the combined-predicate interaction remains populated, but it is supportive evidence only and not the reproduction criterion.

### 3.8.2 *MySQL Bug #74602.*

Bug #74602 concerns an optimizer access-path selection bias in low-`LIMIT` ordering queries. Under `ORDER BY ... LIMIT 1`, MySQL may prefer an index that satisfies ordering while providing weaker predicate selectivity, increasing row-access work relative to a selective-index alternative. The bug was reported against MySQL 5.6.x and later marked fixed in MySQL 5.7 [3].

[3]https://bugs.mysql.com/bug.php?id=74602

*Official query pattern (source report).* The original report contrasts the default plan with a forced selective-index plan for the same `ORDER BY ... LIMIT 1` shape, showing that the default choice can incur substantially higher row reads despite returning the same tuple:

**Listing 3: Official Bug #74602 pattern (simplified)**

```
-- default plan can prefer the ordering index
EXPLAIN
SELECT *
FROM t1
WHERE b = @b
  AND c <= NOW()
  AND (d IS NULL OR d >= NOW() - INTERVAL 2 DAY)
  AND e = 'BBB'
ORDER BY c ASC
LIMIT 1;

-- selective alternative for comparison
EXPLAIN
SELECT *
FROM t1 FORCE INDEX (b)
WHERE b = @b
  AND c <= NOW()
  AND (d IS NULL OR d >= NOW() - INTERVAL 2 DAY)
  AND e = 'BBB'
ORDER BY c ASC
LIMIT 1;
```

*Adapted StackOverflow-schema trigger query.* In our benchmark, we instantiate the same mechanism on `Votes` using a competing ordering index (`idx_votes_creationdate`) and a selective filter index (`idx_votes_post_votetype`). For fixed `PostId`/`VoteTypeId` values, we compare default and forced variants under identical predicates and ordering:

**Listing 4: Adapted Bug #74602 query used in our benchmark (Votes)**

```
-- default
FLUSH STATUS;
SELECT Id, PostId, VoteTypeId, CreationDate
FROM Votes
WHERE PostId = @post
  AND VoteTypeId = @vote_type
ORDER BY CreationDate ASC
LIMIT 1;
SHOW STATUS LIKE 'Handler_read%';
SHOW STATUS LIKE 'Sort%';

-- forced selective
FLUSH STATUS;
SELECT Id, PostId, VoteTypeId, CreationDate
FROM Votes FORCE INDEX (idx_votes_post_votetype)
WHERE PostId = @post
  AND VoteTypeId = @vote_type
ORDER BY CreationDate ASC
LIMIT 1;
SHOW STATUS LIKE 'Handler_read%';
SHOW STATUS LIKE 'Sort%';
```

*Reproduction criterion in this thesis.* We classify bug #74602 as reproduced when: (i) default and forced variants return the same row, (ii) the default plan selects the ordering-favoring access path (or an equivalent non-selective alternative), and (iii) the default variant exhibits materially higher execution-cost signals than the forced selective-index variant, most notably elevated `Handler_read_next`, with supporting `Sort%`/runtime differences.

Because bug-trigger mechanisms differ across cases, iterative configuration design is reported separately for each bug.

## 3.9 Iterative Configuration Design: MySQL #111669

For MySQL bug #111669, parameter tuning was conducted as an iterative, constraint-aware process rather than a single fixed design choice. The target behavior is not generic utility alone, but reproduction of a specific asymmetry: two individually fast full-text subqueries and a slow/timeout combined intersection form. In this setting, the practical signal is concentrated in `PostHistory`-driven fulltext matches and their overlap over `PostId`. In the source `PostHistory` data (11,795,244 rows, token `error`), overlap is sparse: 981 rows contain the token in both `Text` and `Comment`, versus 115,955 `Text`-token rows and 16,607 `Comment`-token rows (about 0.85% of `Text`-token rows, and about 0.008% of all `PostHistory` rows). Consequently, absolute overlap in synthetic outputs depends both on correlation preservation and on emitted synthetic row counts (especially `n-syn-posthistory`): lower synthetic sizes reduce expected overlap even when rates are approximately preserved.

*Why source table-ratio matching was not the primary objective.* Although source-scale ratios between `Posts` and `PostHistory` are informative, directly preserving that global ratio at larger synthetic scales can make the cross-table learning stage prohibitively expensive in memory and runtime. In our implementation, the dominant resource pressure is tied to the synthetic cross-space size induced by `n-syn-posthistory` × `n-syn-posts`, not only to each table size in isolation. Therefore, for bug #111669, preserving the source global table ratio was treated as secondary to preserving enough overlap rows to maintain the bug-triggering predicate interaction under hardware limits.

*Iterative tuning protocol.* We first fixed memory-stable PGD controls (notably `–subtable-size`) and a constant seed for diagnostic comparability, then varied synthetic table sizes and relational controls incrementally. Candidate settings were retained only if they (i) completed without out-of-memory (OOM) failures, (ii) preserved non-trivial token-overlap counts in exported `PostHistory.csv`, and (iii) reproduced the expected execution asymmetry in MySQL. This produced a two-level search strategy: overlap-preserving configuration search in synthesis space, followed by execution-level validation in DBMS space.

## 3.10 Iterative Configuration Design: MySQL #74602

For MySQL bug #74602, parameter tuning was likewise conducted as an iterative, constraint-aware process rather than as a one-shot configuration choice. The target behavior is preservation of an access-path asymmetry in low-`LIMIT` queries: under identical predicates and ordering, the default variant should exhibit substantially higher scan effort than a forced selective-index variant. Therefore, the tuning objective is behavior preservation under DP synthesis, not only aggregate error minimization.

*Signal definition and run acceptance.* Candidate synthetic configurations were evaluated using execution-level signals from the reproduction harness. A candidate was retained only when default and forced query forms returned the same row, and the default form showed materially higher scan-cost evidence (primarily `Handler_read_next`, with supporting runtime/`Sort` signals). This kept selection tied to the bug mechanism itself rather than to distributional fit alone.

*Three-phase search protocol.* To separate concerns and maintain comparability, the search followed three phases. Phase 1 performed structure and memory-safety exploration to obtain an initial reproducible anchor. Phase 2 fixed that structural profile and swept $\epsilon$ to map the reproducibility boundary with controlled seed comparability. Phase 3 treated reproducibility as a hard gate and utility (`average_error`) as a secondary objective, first varying preprocessing caps and then PGD controls to seek a practical operating point.

## 4 Results

### 4.1 Hardware Resource Envelope (12-Run Representative Sample)

To characterize practical synthesis hardware requirements, we ran a representative 12-run subset and logged peak memory metrics in `outputs/memory_summary_12runs.csv`. The subset was selected by stratified coverage (not random sampling): for each bug (#111669, #74602) and each campaign (A, B, C), we selected two runs that cover anchor/early structure search behavior, fixed-profile epsilon-sweep behavior, and operating-point or cap-variation behavior. This yields $2 \times 3 \times 2 = 12$ runs.

Peak memory was recorded from instrumented `dp-relational` experiment drivers as (i) peak process RSS (system RAM) and (ii) CUDA peak allocated and reserved memory during the run window that includes preprocessing and synthesis. Table 5 summarizes the observed ranges.

### 4.2 MySQL #111669 Campaign Outcomes

We evaluated Bug #111669 using three experimental campaigns. Campaign A (`C-111669-A`) is the legacy, full-source-cap setup. Campaign B (`C-111669-B`) uses the default-capped regime (250000 PostHistory rows, 100000 Posts rows) and an unscaled, bug-focused profile. Campaign C (`C-111669-C`) is a ratio-preserving reduced-size ablation where synthetic table sizes maintain the source cardinality ratio while reducing absolute size for memory safety.

Campaign A also included post-synthesis $\times 10$ scaling controls to stress-test trigger recovery under larger execution loads. In these runs, timeout behavior was often restored even when the corresponding unscaled synthetic output did not time out, indicating that post-processing can materially change execution conditions (cardinality, selectivity, and runtime) without changing privacy accounting. Therefore, we treat scaled outcomes as auxiliary stress evidence and base boundary and operating-point conclusions on unscaled campaigns (`C-111669-B`, `C-111669-C`).

Campaign B provided the strongest practical result: unscaled timeout reproduction was retained across the full tested $\epsilon$-range from 2.0 down to 0.2. Table 6 reports the exact values, and Figure 1

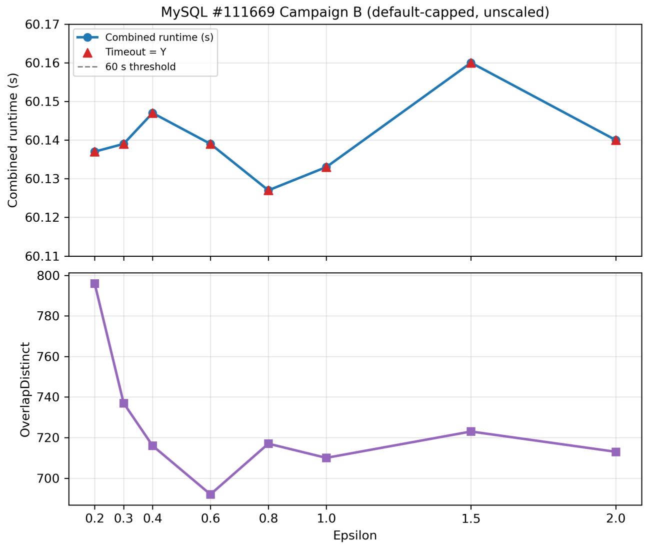

**Figure 1: Campaign B (default-capped) trends for MySQL #111669: combined runtime with timeout markers (top) and overlap cardinality (bottom).**

visualizes the observed trends. Across Campaigns B and C, overlap counts remain non-trivial at their respective synthetic scales, which is consistent with retaining the bug-relevant interaction signal needed for timeout reproduction. We report `OverlapDistinct` as an auxiliary intersection-signal metric. Here, `Timeout 3024` denotes MySQL error code 3024 (query execution was interrupted because `max_execution_time` was exceeded). Reproduction decisions are based on execution behavior (`Combined sec`, `Timeout 3024`), so `OverlapDistinct` is interpreted as supportive evidence and is not expected to vary monotonically with $\epsilon$.

Campaign C confirms that ratio-preserving synthesis can still reproduce the bug without post-synthesis scaling, but with a sharper utility threshold. In this setup, timeout reproduction is observed at $\epsilon \in \{1.0, 0.8, 0.7\}$, while $\epsilon = 0.6$ remains suboptimal-plan-positive but falls below timeout. Table 7 reports the exact values, and Figure 2 highlights the threshold transition.

For feasibility interpretation, memory pressure is better approximated by cross-space size ($n_{\text{syn,PH}} \times n_{\text{syn,Posts}}$) together with PGD slice settings than by $\epsilon$ alone. If source ratio preservation were applied at $n_{\text{syn,PH}} = 240000$, then $n_{\text{syn,Posts}} \approx 78336$ and cross-space would be $\approx$ 18.8B cells, which is impractical for our hardware envelope. Campaign C instead uses $(36000, 11752)$, giving $\approx$ 423M cross cells, close to Campaign B's $\approx$ 432M, enabling a fairer utility comparison.

### 4.3 MySQL #74602 Campaign Outcomes

We evaluated Bug #74602 using three experimental campaigns. Campaign A (`C-74602-A`) performs initial structure search to obtain the first reproducing anchor; Campaign B (`C-74602-B`) freezes that structure and maps the reproducibility boundary over $\epsilon$; Campaign C (`C-74602-C`) selects a practical operating point under a hard reproducibility gate with `average_error` as a secondary objective.

**Table 4: Bug reproduction outcomes by dataset size on source and synthetic data. Synthetic runs are attempted only when source reproduction at the same size succeeds.**

| Bug ID | Config ID | Source data | | | Synthetic data | | |
|---|---|---|---|---|---|---|---|
| | | 1GB | 12GB | 222GB | 1GB | 12GB | 222GB |
| MySQL #111669 | C-111669-A | – | ✓ | – | – | ✓ | – |
| MySQL #111669 | C-111669-B | – | ✓ | – | – | ✓ | – |
| MySQL #111669 | C-111669-C | – | ✓ | – | – | ✓ | – |
| MySQL #74602 | C-74602-A | – | – | ✓ | – | – | ✓ |
| MySQL #74602 | C-74602-B | – | – | ✓ | – | – | ✓ |
| MySQL #74602 | C-74602-C | – | – | ✓ | – | – | ✓ |

Legend: ✓= reproduced; ✗= not reproduced; – = not attempted because source reproduction failed at the same dataset size or not attempted. For MySQL #111669 at 222GB, source reproduction was not attempted due to practical index-build setup time at that scale.

**Table 5: Peak memory envelope from the 12-run representative synthesis subset.**

| Group | Peak system RAM (GiB) | Peak GPU allocated (GiB) | Peak GPU reserved (GiB) |
|---|---|---|---|
| MySQL #111669 | 17.433–17.673 | 31.643–53.131 | 35.512–59.609 |
| MySQL #74602 | 6.656–6.657 | 11.621–25.334 | 13.023–28.438 |
| Overall max | 17.673 | 53.131 | 59.609 |

**Table 6: Campaign B (default-capped) unscaled $\epsilon$-sweep for MySQL #111669.**

| $\epsilon$ | Combined sec | Timeout 3024 | OverlapDistinct |
|---|---|---|---|
| 2.0 | 60.140 | Y | 713 |
| 1.5 | 60.160 | Y | 723 |
| 1.0 | 60.133 | Y | 710 |
| 0.8 | 60.127 | Y | 717 |
| 0.6 | 60.139 | Y | 692 |
| 0.4 | 60.147 | Y | 716 |
| 0.3 | 60.139 | Y | 737 |
| 0.2 | 60.137 | Y | 796 |

**Table 7: Campaign C (ratio-preserving reduced-size) unscaled $\epsilon$-sweep for MySQL #111669.**

| $\epsilon$ | Combined sec | Timeout 3024 | OverlapDistinct |
|---|---|---|---|
| 1.0 | 60.124 | Y | 403 |
| 0.8 | 60.147 | Y | 409 |
| 0.7 | 60.137 | Y | 424 |
| 0.6 | 57.582 | N | 419 |

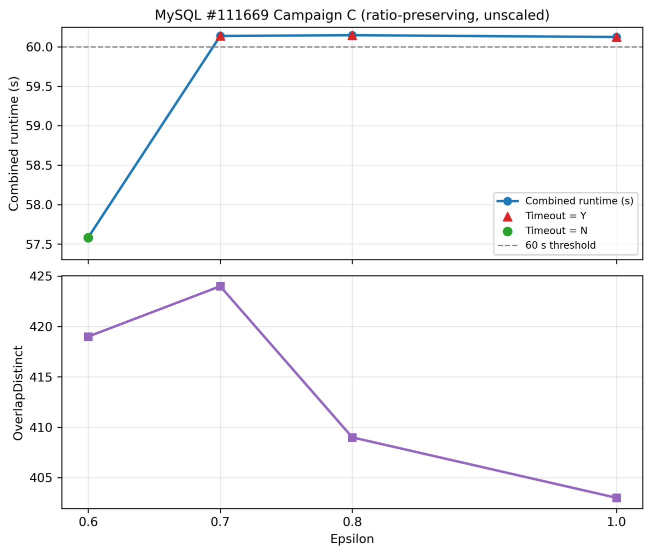

**Figure 2: Campaign C (ratio-preserving reduced-size) trends for MySQL #111669: combined runtime and timeout transition (top) and overlap cardinality (bottom).**

For robustness reporting, we use `Meets(Y)` over a fixed set of 150 candidate probes per run. A probe is counted as `Y` when it satisfies the same behavioral reproduction checks used by the harness (same-row agreement between default and forced variants, plus required execution-cost asymmetry). Accordingly, `Meets(Y)/150` is reported as a robustness ratio: the number of probe checks meeting the criterion out of 150. Early diagnostic runs used smaller candidate limits (e.g., 30), but boundary mapping and operating-point selection use a fixed 150-probe budget to reduce sensitivity to single-candidate effects and improve cross-run stability.

In Campaign A (`C-74602-A`), we established the first reproducible anchor by iterating structural and memory-safety settings. The first stable reproducing profile used n-syn-votes = 360000, n-syn-posts = 400, dmax = 2000, max-votes-rows = 500000, max-posts-rows =

**Table 8: Campaign B boundary points for MySQL #74602 (`C-74602-B`, fixed-structure sweep).**

| $\epsilon$ | Reproduced | Meets(Y)/150 | Avg. error | Note |
|---|---|---|---|---|
| 1.8 | Y | 47 | 56.2368 | lowest passing point |
| 1.7 | N | 0 | 58.9097 | first failing point |
| 1.6 | N | 0 | 65.1300 | last runnable failing point |
| $\leq 1.5$ | – | – | – | synthesis not runnable (budget floor) |

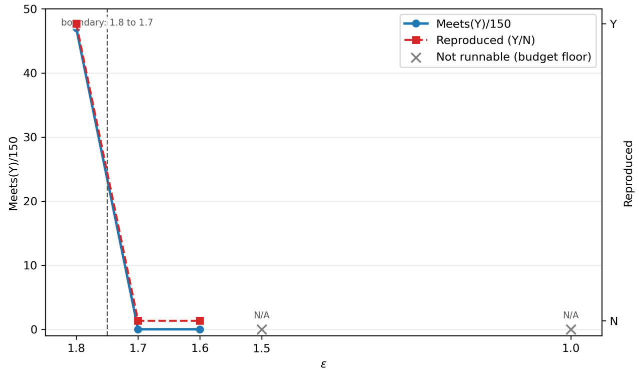

**Figure 3: Campaign B boundary profile for MySQL #74602 (`C-74602-B`): Meets(Y)/150 on the left axis and reproduced status (Y/N) on the right axis. Points at $\epsilon = 1.5$ and $\epsilon = 1.0$ are marked as not runnable due to the synthesis-time budget floor.**

1000, and guaranteed-rels = 0.97. PGD controls were T = 8, queries-to-reuse = 16, and k-new-queries = 12.

In Campaign B (`C-74602-B`), we froze this profile and swept $\epsilon$ to map the reproducibility boundary. Reproduction was retained across the tested interval from $\epsilon = 9.0$ down to $\epsilon = 1.8$, and was lost at $\epsilon = 1.7$ and $\epsilon = 1.6$. Additional probes at $\epsilon = 1.5$ and $\epsilon = 1.0$ did not produce synthetic outputs because the implemented budget split left insufficient training budget after preprocessing. Table 8 summarizes the boundary points, and Figure 3 visualizes the same boundary using dual y-axes for robustness and binary reproduction status. We also observed near-boundary sensitivity: a Campaign C control rerun at $\epsilon = 1.8$ with the same nominal selector-profile settings did not pass the reproduction gate (`Meets(Y)=0/150`). We therefore interpret $\epsilon = 1.8$ as the lowest observed passing point rather than a guaranteed pass for every rerun.

In Campaign C (`C-74602-C`), reproducibility was treated as a hard gate and `average_error` as a secondary optimization objective. In the cap-sweep phase at $\epsilon = 1.8$, broader `Posts`-cap settings reduced error but failed reproduction (all Meets(Y) = 0/150), reaching an average error of 48.1287 at `max-posts-rows=120000`. Raising $\epsilon$ to 5.0 under the same high-cap setting further reduced error to the lowest observed value (47.7692), but reproduction still failed (Meets(Y) = 0/150). Reproducibility was restored only after returning to the cap-1k profile and deepening PGD controls

**Table 9: Campaign C3 operating-profile sweep for MySQL #74602 (`C-74602-C`; `max-posts-rows=1000`, `T=10`, `queries-to-reuse=20`, `k-new-queries=14`).**

| $\epsilon$ | Avg. error | Meets(Y)/150 | Best read_next_ratio | Reproduced |
|---|---|---|---|---|
| 5.0 | 50.2762 | 26 | 12.63 | Y |
| 4.0 | 50.8373 | 35 | 13.72 | Y |
| 3.5 | 51.0617 | 31 | 13.29 | Y |
| 3.0 | 52.1300 | 28 | 13.52 | Y |

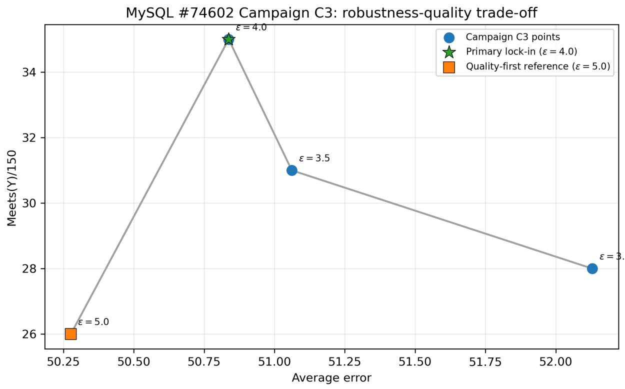

**Figure 4: Campaign C3 trade-off for MySQL #74602 (`C-74602-C`): robustness (Meets(Y)/150) versus `average_error`. The primary final selected operating point (lock-in) is $\epsilon = 4.0$; $\epsilon = 5.0$ is retained as a quality-first reference.**

(`T=10`, `queries-to-reuse=20`, `k-new-queries=14`); this profile remained reproducible through $\epsilon = 5.0, 4.0, 3.5, 3.0$, with gradual robustness decline as $\epsilon$ decreased (Table 9). Figure 4 shows the resulting robustness-quality trade-off and highlights the selected operating points.

For reporting clarity, Campaign B (`C-74602-B`) is used to state the boundary result ($\epsilon = 1.8$, lowest passing point under fixed structure), while the Campaign C (`C-74602-C`) is used to state the final operating-point decision. We select $\epsilon = 4.0$ as the primary #74602 operating point because it provides the best robustness among low-error operating-profile passes (35/150) with only a small error increase relative to $\epsilon = 5.0$. We keep $\epsilon = 5.0$ as a quality-first reference due to its lower `average_error`. Thus, Campaign C is reported as constrained optimization: the global minimum-error run was not feasible under the reproducibility criterion, and final selection is based on the best reproducible trade-off.

## 5 Discussion and Threats to Validity

*Internal validity.* LLM outputs are stochastic and web interfaces do not expose decoding parameters (e.g., temperature, top-p, system prompt). To mitigate discovery-stage variability, we archived prompts and model identifiers and applied predefined inclusion/exclusion criteria during candidate selection. Bug reproduction was then evaluated separately using deterministic script-based harnesses and fixed decision rules.

*Construct validity.* Adapting a historical bug report to a different schema may alter trigger conditions. We therefore required behavior-level evidence (plan shape, counters, timeout profile, or wrong-result signature) rather than textual similarity to original reports.

*Privacy-scope validity.* Formal DP accounting in this work applies to synthesis stages (single-table and relational synthesis). Data-dependent wrapper transformations outside those stages can affect strict end-to-end guarantee interpretation. Accordingly, privacy claims are scoped to DP-enabled synthesis stages rather than full pipeline guarantees over raw source data.

*External validity.* Our case-study set (e.g., MySQL #111669 and #74602) may not represent all optimizer or engine bug classes. Results should be interpreted as evidence for reproducibility-retention in selected bug families, not universal DBMS behavior.

*Conclusion validity.* Although the thesis project spanned roughly three months, the final experiment window was constrained after substantial upfront effort on research/scoping, infrastructure setup (hardware/software orchestration), DBMS provisioning, data loading/indexing, and source-data bug reproduction. These resource and time constraints limit the number of seeds, epsilon points, and full 222GB runs. We report these limits explicitly and prioritize transparent logging and rerunnable scripts over breadth.

## 6 Conclusion

This thesis investigated whether data-dependent DBMS bug behavior can be retained under differentially private relational synthesis. We used a two-stage workflow: LLM-assisted candidate discovery followed by schema-constrained adaptation and deterministic validation in a MySQL harness. Within this setup, we reproduced MySQL bugs #111669 and #74602 on source data and evaluated reproducibility retention on synthetic data across epsilon settings and campaign configurations.

Results show that retention is feasible but strongly bug- and configuration-dependent. For #111669, timeout behavior without post-synthesis scaling remained reproducible across the tested epsilon range in the default-capped campaign, while ratio-preserving reduced-size settings showed a sharper threshold. For #74602, fixed-structure boundary mapping retained reproduction to $\epsilon = 1.8$ and failed at 1.7 and 1.6, and the selected operating point at $\epsilon = 4.0$ balanced reproducibility robustness and error. Overall, the study provides practical evidence that privacy-aware synthetic bug-reproduction workflows can preserve bug-trigger behavior when accompanied by explicit adaptation criteria, execution-level validation, and clear privacy-scope accounting boundaries.

## 7 Limitations and Future Work

### 7.1 Current Limitations

This thesis provides evidence from two selected MySQL optimizer bugs and should therefore be interpreted as a scoped case study rather than a universal claim across bug classes, DBMS families, or deployment settings. Resource constraints also limited full-factorial exploration (seeds, epsilon points, and full-scale runs). In addition, formal privacy accounting is scoped to synthesis stages; data-dependent wrapper preprocessing/postprocessing steps are outside a strict end-to-end DP proof over raw source data. Comparative Kamino experiments were not executed in this thesis phase due to available time and are deferred to future work.

### 7.2 Optimizing Bug-Targeted Synthesis Configuration

A key bottleneck in the current workflow is that bug-targeted synthetic reproduction still requires substantial manual effort. For each bug, we must first review the original report and identify the behavior that must be preserved (e.g., predicate interaction, index-choice sensitivity, join/cardinality effects, timeout asymmetry). We then manually design the synthetic experiment (feature selection, table pairing, export format) and tune a high-dimensional parameter space (privacy budget, synthesizer choice, relational-learning controls, row caps, and synthetic table sizes). In practice, this process is iterative and expensive, especially when the goal is not only utility preservation but bug-trigger preservation.

Future work should treat this step as an optimization problem rather than manual tuning. Given a fixed bug and source-scale baseline, the system could automatically search for configurations that maximize a bug-specific reproduction score under resource and privacy constraints. Candidate approaches include Bayesian optimization, multi-fidelity sweeps, and adaptive early stopping over parameter trials. This would reduce dependence on expert intuition and improve repeatability across bugs and DBMS versions.

### 7.3 End-to-End Privacy Accounting

A parallel direction is to strengthen privacy guarantees from synthesis-stage accounting to end-to-end pipeline accounting. This includes replacing global data-dependent selection heuristics with DP-aware or stability-bounded alternatives, explicitly budgeting preprocessing and postprocessing stages, and evaluating reproducibility under full composition-based privacy accounting.

### 7.4 Generalization Across Bug Classes and DBMSs

Future validation should include additional bug classes (e.g., correctness and engine-level behaviors) and additional systems beyond the current MySQL-focused scope. A broader corpus would help distinguish bug-specific from method-level effects, improve external validity, and clarify how well bug-targeted configuration strategies transfer across schemas and DBMS implementations.

### 7.5 Resource-Aware and Multi-Seed Evaluation

An additional direction is adaptive scaling and stronger robustness profiling. Instead of evaluating only fixed scales, the pipeline could progressively increase synthetic scale and identify the minimum scale at which the target bug behavior reappears. Such threshold-oriented scaling would reduce compute cost while improving interpretability of when and why reproduction fails. Coupling this with larger multi-seed evaluations and explicit hardware-envelope reporting would improve confidence in reproducibility claims and support resource-aware operating-point selection.

## 8 Use of Generative AI

Generative AI tools were used to help with finding of performance and correctness bugs for further investigation and implementation into the pipeline, as mentioned in the body of the paper. Beyond that, OpenAI's Codex 5.3 was used to assist in the code implementation.

## References

[1] 2012. Survey of Publicly Available State Health Databases. ResearchGate. https://www.researchgate.net/publication/237092249_Survey_of_Publicly_Available_State_Health_Databases Accessed: 2026-02-21.

[2] Kaveh Alimohammadi, Hao Wang, Ojas Gulati, Akash Srivastava, and Navid Azizan. 2025. Differentially Private Synthetic Data Generation for Relational Databases. arXiv preprint arXiv:2405.18670 (2025). https://arxiv.org/abs/2405.18670

[3] Azizan Lab. 2026. azizanlab/dp-relational: Code for "Adapting Differentially Private Synthetic Data to Relational Databases". https://github.com/azizanlab/dp-relational. GitHub repository, accessed 2026-02-16.

[4] James Bennett and Stan Lanning. 2007. The Netflix Prize. Proceedings of KDD Cup and Workshop 2007. https://www.cs.uic.edu/~liub/KDD-cup-2007/proceedings/The-Netflix-Prize-Bennett.pdf

[5] Kuntai Cai, Xiaokui Xiao, and Graham Cormode. 2023. PrivLava: Synthesizing Relational Data with Foreign Keys under Differential Privacy. Proceedings of the ACM on Management of Data 1, 2 (2023). https://doi.org/10.1145/3589287

[6] Kuntai Cai, Xiaokui Xiao, and Yin Yang. 2025. PrivPetal: Relational Data Synthesis via Permutation Relations. arXiv preprint arXiv:2503.22970 (2025). https://arxiv.org/abs/2503.22970

[7] Huseyin Dervisoglu, Ramazan Halepmollasi, and Emre Eyvaz. 2025. Privacy-Preserving Methods for Bug Severity Prediction. arXiv preprint arXiv:2506.22752 (2025). https://arxiv.org/abs/2506.22752

[8] Cynthia Dwork, Nitin Kohli, and Deirdre Mulligan. 2019. Differential Privacy in Practice: Expose Your Epsilons! Journal of Privacy and Confidentiality 9, 2 (2019). https://doi.org/10.29012/jpc.689

[9] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. Foundations and Trends in Theoretical Computer Science 9, 3–4 (2014), 211–407. https://doi.org/10.1561/0400000042

[10] Chang Ge and contributors. 2026. cgebest/kamino: Source code for the VLDB 2021 paper. https://github.com/cgebest/kamino. GitHub repository, accessed 2026-02-16.

[11] Chang Ge, Shubhankar Mohapatra, Xi He, and Ihab F. Ilyas. 2021. Kamino: Constraint-Aware Differentially Private Data Synthesis. arXiv preprint arXiv:2012.15713 (2021). https://arxiv.org/abs/2012.15713

[12] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems 5, 4 (2015), 19:1–19:19. https://doi.org/10.1145/2827872

[13] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, and Aaron Roth. 2014. Differential Privacy: An Economic Method for Choosing Epsilon. In 2014 IEEE 27th Computer Security Foundations Symposium (CSF). 398–410. https://doi.org/10.1109/CSF.2014.35

[14] IMDb. 2026. IMDb Non-Commercial Datasets. https://developer.imdb.com/non-commercial-datasets. Accessed 2026-02-23.

[15] Jinho Jung, Hong Hu, Joy Arulraj, Taesoo Kim, and Woon-Hak Kang. 2019. APOLLO: Automatic Detection and Diagnosis of Performance Regressions in Database Systems. Proceedings of the VLDB Endowment 13, 1 (2019), 57–70. https://doi.org/10.14778/3357377.3357382

[16] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? Proceedings of the VLDB Endowment 9, 3 (2015), 204–215. https://doi.org/10.14778/2850583.2850594

[17] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. 2007. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In Proceedings of the 23rd International Conference on Data Engineering (ICDE). 106–115. https://doi.org/10.1109/ICDE.2007.367856

[18] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. 2007. l-Diversity: Privacy Beyond k-Anonymity. ACM Transactions on Knowledge Discovery from Data 1, 1 (2007). https://doi.org/10.1145/1217299.1217302

[19] Bradley Malin and Latanya Sweeney. 2004. How (Not) to Protect Genomic Data Privacy in a Distributed Network: Using Trail Re-identification to Evaluate and Design Anonymity Protection Systems. Journal of Biomedical Informatics 37, 3 (2004), 179–192. https://doi.org/10.1016/j.jbi.2004.04.005

[20] Arvind Narayanan and Vitaly Shmatikov. 2006. Robust De-anonymization of Large Datasets (How to Break Anonymity of the Netflix Prize Dataset). arXiv:cs/0610105. https://arxiv.org/pdf/cs/0610105 Accessed: 2026-02-21.

[21] Manuel Rigger and Zhendong Su. 2020. Detecting Optimization Bugs in Database Engines via Non-Optimizing Reference Engine Construction. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). https://doi.org/10.1145/3368089.3409710

[22] Manuel Rigger and Zhendong Su. 2020. Finding Bugs in Database Systems via Query Partitioning. Proceedings of the ACM on Programming Languages 4, OOPSLA (2020). https://doi.org/10.1145/3428279

[23] Lucas Rosenblatt, Bernease Herman, Anastasia Holovenko, Wonkwon Lee, Joshua Loftus, Elizabeth McKinnie, Taras Rumezhak, Andrii Stadnik, Bill Howe, and Julia Stoyanovich. 2023. Epistemic Parity: Reproducibility as an Evaluation Metric for Differential Privacy. Proceedings of the VLDB Endowment 16, 11 (2023), 3178–3191. https://doi.org/10.14778/3611479.3611517

[24] Tobias Schmidt, Viktor Leis, Peter Boncz, and Thomas Neumann. 2025. SQLStorm: Taking Database Benchmarking into the LLM Era. Proceedings of the VLDB Endowment 18, 11 (2025), 4144–4157. https://doi.org/10.14778/3749646.3749683

[25] Latanya Sweeney. 1997. Weaving Technology and Policy Together to Maintain Confidentiality. Journal of Law, Medicine & Ethics 25, 2-3 (1997), 98–110. https://doi.org/10.1111/j.1748-720X.1997.tb01885.x

[26] Latanya Sweeney. 2002. k-Anonymity: A Model for Protecting Privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10, 5 (2002), 557–570. https://doi.org/10.1142/S0218488502001648

## A DP-Relational Parameter Configuration

Table 10 lists the bug-specific dp-relational configurations used in synthetic-data runs.

## B Reproducibility Package and Prompt Protocol

### B.1 Grok-Assisted Candidate Bug List

We also collected a Grok candidate list using a prompt comparable to Prompt 1. Because this thesis focuses on the ChatGPT stream, the Grok list is included for completeness; SQLancer-derived toy-dataset cases are excluded. Table 11 summarizes the remaining candidates.

### B.2 Artifact Inventory

The reproducibility materials are distributed with this thesis as `thesis_attachments`, organized into four groups:

- **DP-relational adaptation:** bug-specific synthesis scripts, adapter modules, dependencies, and the `QueryManager.get_offsets()` crash-fix patch (Appendix B.3, Listing 5).
- **MySQL harness:** environment template, Docker Compose configuration, loader/reproduction scripts, and minimal schemas.
- **Source-bug SQL:** source-data reproduction SQL for MySQL #111669 and #74602.
- **Results:** iterative matrices, consolidated CSV/JSON datasets for figure regeneration, and representative resource-profile logs.

### B.3 Patch Artifact: QueryManager Float-Category Normalization

Listing 5 reproduces the exact code diff applied to resolve the synthesized-float offset crash during bug #74602 experiments.

**Listing 5: Patch artifact for `QueryManager.get_offsets()`: float-category normalization and clipping.**

```
diff --git a/dp_relational/lib/qm.py b/dp_relational/lib/qm.py
index 30723e8..5a5100f 100644
--- a/dp_relational/lib/qm.py
+++ b/dp_relational/lib/qm.py
@@ -210,10 +210,21 @@ class QueryManager:
         data_sizes = [self.n_syn1, self.n_syn2] if is_synth \
             else [self.rel_dataset.table1.df.shape[0],self.rel_dataset.table2.
      df.shape[0]]
         data_size = data_sizes[table_num]
+        dim_counts = self.workload_dict[workload][["dim_1", "dim_2"][table_num
      ]]

         offsets = np.zeros(shape=data_size, dtype=np.int_)
         for col in range(len(dimsizes)):
-            offsets += table[col] * dimsizes[col]
+            # Synthesizers can emit category-coded columns as float dtype (e.g.
       3.0).
+            # Normalize to integer category indices and clamp to workload
      domain.
+            values = np.asarray(table[col])
+            if np.issubdtype(values.dtype, np.integer):
+                values = values.astype(np.int_, copy=False)
+            else:
+                values = np.rint(values).astype(np.int_, copy=False)
+
+            max_idx = max(int(dim_counts[col]) - 1, 0)
+            values = np.clip(values, 0, max_idx)
+            offsets += values * dimsizes[col]

         return offsets
```

### B.4 LLM Prompting Protocol

We used web-based LLM interfaces (ChatGPT and Grok) for candidate-bug discovery, specifically to identify confirmed, data-dependent DBMS bugs with realistic data assumptions and available reproduction details.
For each prompting session, we logged:

- model name and access mode (web app),
- date/time of interaction,
- full user prompts and model responses,
- selected bug IDs and exclusion reasons.

### B.5 Exact Prompt Transcript (ChatGPT Web App)

*Prompt 1 - initial bug discovery request*

*P1*
```
Task: Identify confirmed, data-dependent bugs in relational DBMSs. I am compiling a corpus of confirmed DBMS bugs for empirical evaluation. Please return bug cases that satisfy all of the following. (1) Bug type: either correctness bugs (e.g., wrong results, crashes) or performance bugs/regressions (e.g., severe slowdown, optimizer mis-plans). (2) Data dependence: the bug must depend on data characteristics such as skew, NULL distribution, cardinality, data volume, correlation, join structure, or partitioning. (3) Realistic scale: reproduction must involve non-trivial data volume or realistic schema/workload, not toy examples with only a few rows. (4) Confirmation status: include only confirmed bugs from official sources (issue trackers, release notes, mailing lists, or vendor/developer forums). (5) Reproducibility evidence: include reproduction SQL or steps, or sufficient details to reconstruct a failing test. Scope: prioritize PostgreSQL and MySQL; include other RDBMSs if they meet the same criteria; no publication-year restriction. For each bug provide: DBMS and version(s) affected, bug ID and title, bug class (correctness or performance), data-dependent trigger condition, evidence of realistic scale, reproduction query or steps (or closest available), confirmation evidence (status, fix, or release note if available), and direct source links (primary source preferred). Quality constraints: do not invent bug IDs or links; if evidence is ambiguous, label as uncertain and exclude from the final recommended list.
```

*Model access note.* Prompt 1 was submitted to both ChatGPT and Grok on 24 January 2025 at 22:35. On ChatGPT the run used GPT-5.2 with Deep Research mode; on Grok, Expert mode was used. Decoding parameters such as temperature, top-p, and system prompt were not user-visible in the interface and are therefore reported as unknown.

### B.6 Execution and Labeling Protocol

Each candidate bug was validated via script-based execution and labeled as:

- **Reproduced:** expected failure/slowdown signature observed.
- **Partially reproduced:** related behavior observed, but not full signature.
- **Not reproduced:** expected behavior not observed.

Logged evidence includes:

- runtime and timeout outcome,
- query plan evidence (`EXPLAIN` or equivalent),
- optimizer/runtime counters (e.g., `Handler_read_next`, `Sort %`),
- dataset scale and synthesis configuration.

## B.7 Threats Specific to LLM-Assisted Candidate Mining

- LLM outputs are stochastic and may include inaccuracies; therefore, all candidates require execution-based confirmation.
- Web-app usage does not expose full decoding configuration; reproducibility is enforced through prompt logging.
- Schema adaptation can alter trigger conditions; conclusions are based on observed behavior-level signatures, not textual similarity to original reports.

**Table 10: Bug-specific dp-relational configurations (keyed by config ID).**

| Config ID | Target Bug | Parameter Group | Values Used |
|---|---|---|---|
| C-111669-A | MySQL #111669 | Campaign role | Legacy baseline campaign used before default-capped and ratio-preserving refinements; included unscaled runs plus post-synthesis $\times10$ controls. |
| C-111669-A | MySQL #111669 | Representative configuration | `–synth aim`; `–device cuda:0`; `–n-syn-posthistory 220000`; `–n-syn-posts 2000`; `–dmax 15`; `–k 4`; `–T 10`; `–subtable-size 120000`; `–queries-to-reuse 20`; `–k-new-queries 20`; `–guaranteed-rels 0.8`. |
| C-111669-A | MySQL #111669 | Cap policy | Explicit full-source caps used in this legacy campaign (`–max-posthistory-rows PH_ROWS`, `–max-posts-rows P_ROWS`). |
| C-111669-B | MySQL #111669 | Campaign role | Default-capped unscaled sweep optimized for direct timeout reproduction without post-synthesis scaling. |
| C-111669-B | MySQL #111669 | Core parameters | `–synth aim`; `–device cuda:0`; `–n-syn-posthistory 240000`; `–n-syn-posts 1800`; `–dmax 10`; `–k 3`; `–T 6`; `–subtable-size 120000`; `–queries-to-reuse 12`; `–k-new-queries 8`; `–guaranteed-rels 0.9`. |
| C-111669-B | MySQL #111669 | Cap policy and outcome | Explicit default caps (`–max-posthistory-rows 250000`, `–max-posts-rows 100000`); reproduced unscaled timeout for all tested $\epsilon$ values from 2.0 down to 0.2. |
| C-111669-C | MySQL #111669 | Campaign role | Ratio-preserving reduced-size ablation to test whether source cardinality ratio can be maintained under memory-safe synthetic sizes. |
| C-111669-C | MySQL #111669 | Core parameters | `–n-syn-posthistory 36000`; `–n-syn-posts 11752` (ratio $\approx 0.3264$); same optimizer/query knobs as `C-111669-B`; caps fixed at 250000/100000. |
| C-111669-C | MySQL #111669 | Outcome | Unscaled timeout reproduced at $\epsilon = 1.0, 0.8, 0.7$; not reproduced at $\epsilon = 0.6$ (combined runtime below timeout threshold). |
| C-74602-A | MySQL #74602 | Campaign role | Initial structure and memory-safe search (anchor $\rightarrow$ deskew $\rightarrow$ density $\rightarrow$ selector tuning) to obtain the first reproducing synthetic profile. |
| C-74602-A | MySQL #74602 | Representative configuration | `–synth aim`; `–device cuda:0`; `–n-syn-votes 360000`; `–n-syn-posts 400`; `–dmax 2000`; `–k 3`; `–T 8`; `–subtable-size 120000`; `–queries-to-reuse 16`; `–k-new-queries 12`; `–guaranteed-rels 0.97`; `–max-votes-rows 500000`; `–max-posts-rows 1000`. |
| C-74602-A | MySQL #74602 | Outcome | First criteria-meeting synthetic reproduction obtained at $\epsilon = 10.0$ (selectorB anchor), establishing the fixed structural profile used in subsequent campaigns. |
| C-74602-B | MySQL #74602 | Campaign role | Fixed-structure epsilon sweep to map the reproducibility boundary after anchoring the selectorB profile. |
| C-74602-B | MySQL #74602 | Core parameters | Fixed structural/cap settings from `C-74602-A`; epsilon sweep over 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.5, 3.0, 2.5, 2.0, 1.8, 1.7, 1.6 with additional floor probes at 1.5 and 1.0. |
| C-74602-B | MySQL #74602 | Outcome | Reproduction retained down to $\epsilon = 1.8$; not reproduced at $\epsilon = 1.7$ and $\epsilon = 1.6$; runs at $\epsilon \leq 1.5$ were not runnable due to synthesis-time epsilon floor. |
| C-74602-C | MySQL #74602 | Campaign role | Operating-point optimization under a hard reproducibility gate, treating `average_error` as a secondary objective. |
| C-74602-C | MySQL #74602 | Core parameters | Same structural/cap settings as `C-74602-A` with deeper PGD controls (`–T 10`; `–queries-to-reuse 20`; `–k-new-queries 14`); evaluated at $\epsilon = 5.0, 4.0, 3.5, 3.0$. |
| C-74602-C | MySQL #74602 | Outcome | Final selected operating point at $\epsilon = 4.0$ (lock-in; Reproduced=Y, Meets(Y)=35/150); $\epsilon = 5.0$ retained as quality-first reference. |

**Table 11: Grok-assisted candidate bug list from Stage-1 discovery (SQLancer-originated bugs omitted).**

| DBMS | Bug ID | Class | Data-dependent trigger summary from Grok output | Status in this thesis |
|---|---|---|---|---|
| MySQL | #114996 | Performance | Subquery with `ORDER BY` much slower in 8.0 vs. 5.7; semijoin/temporary/filesort; triggered by stale statistics after data distribution changes (e.g., post-INSERT). | Candidate only |
| MySQL | #87641 | Performance | Execution plan differs after upgrade to 5.7 (1s to 30s); optimizer mischooses access method, scans millions more rows; sensitive to data distribution in correlated subquery with joins. | Candidate only |
| MySQL | #103225 | Logical/Perf. | "Out of Sort Memory" inconsistent with buffer size; large variable-length columns (JSON/TEXT/GEOMETRY); non-deterministic sort chunk merging and row sizes. | Candidate only |
| MySQL | #97552 | Performance | Slow `LEFT JOIN` with impossible `ON` (e.g. 1=0) leads to full scan; join cardinality sensitive. Fixed in 8.0.20. | Candidate only |
| PostgreSQL | #15160 | Performance | Planner overestimates rows in join with CTE when CTE outputs > 200 rows; falls back to default selectivity, causing poor plans. | Candidate only |
| PostgreSQL | #19332 | Performance | Sudden 330× slowdown in `SELECT` amid `INSERT`s; autovacuum does not update stats promptly, leading to misestimation of table sizes/cardinality. | Candidate only |
| PostgreSQL | —[a] | Performance | Fix cardinality estimates for parallel joins; misestimates per-worker rows in parallel paths, data-dependent on join selectivity and distribution. | Candidate only |
| Oracle | —[b] | Performance | Inaccurate join cardinality (e.g. nested loops on large skew); skew in join key (e.g. one value 22k matches) prevents accurate estimates without runtime info. | Does not meet prompt criteria (source: Ask TOM, not official issue tracker) |
| SQL Server | —[c] | Logical/Perf. | Wrong estimated rows on Key/RID Lookups with filters when data skewed; underestimates can cause spills to TempDB. | Does not meet prompt criteria (source: personal blog, not official issue tracker). |
| SQL Server | —[d] | Performance | Uses first date in range for estimates, ignoring skew; slow with date expr vs. literals on uneven date distributions. | Does not meet prompt criteria (source: Stack Overflow, not official issue tracker). |
| SQL Server | —[e] | Performance | Skewed/missing stats cause wild overestimates (e.g. 100M rows, 8GB memory grant) on procedures over large skewed tables. | Does not meet prompt criteria (source: DBA Stack Exchange, not official issue tracker). |

[a] PostgreSQL commit log: https://www.postgresql.org/message-id/E1cS6je-0004eL-Id@gemulon.postgresql.org.
[b] https://asktom.oracle.com/ords/f?p=100:11:::::P11_QUESTION_ID:9543288700346979555
[c] https://www.sql.kiwi/2012/10/cardinality-estimation-bug-with-lookups-in-sql-server-2008-onward/
[d] https://stackoverflow.com/questions/18241977/query-runs-slow-with-date-expression-but-fast-with-string-literal
[e] https://dba.stackexchange.com/questions/259550/funky-cardinality-estimate-skews-high-memory-grants-and-missing-statistics