

Supplementary S1 Qualification contract controlled challenges and reproducibility

This supplement accompanies *Qualifying Observation Integrity for Selective Authorization in PostgreSQL*. It specifies the supported installation, conditional argument, experimental populations and reproduction boundaries. Bracketed reference numbers refer to the main manuscript bibliography; the full source catalogue is supplied as `references_v04.json`. Executable artifacts and raw records are supplied in Supplementary S2 (OntoCommit_ResearchSquare_S2_Reproducibility.zip) and are identified by paths relative to its artifact root. No external research implementation or independent laboratory replication is claimed.

S1. Supported installation and trust boundary

The live experiments use the row-trigger profile on ordinary PostgreSQL 16.15. Observations comprise complete rows, equality partitions and whole relations, including empty results. Descriptors encode observation kind, table, ordered field names and typed values. Domain tables have declared unique keys, non-null scalar fields and the restricted deterministic text collation. For each admitted INSERT, DELETE or UPDATE, the invalidator covers the required NEW, OLD or combined OLD/NEW identities. A partition departure and a content change within an unchanged partition both require invalidation. Conservative invalidation after a no-op update is permitted.

Qualification checks the installed program and its execution context. Following each trigger's actual function identifier, it inspects ordered argument names, input and all-argument type identities, argument modes, return type, defaults, variadic and set-returning status, routine kind, strictness, volatility, parallel-safety and security attributes, language, owner and function configuration. The epoch helper must bind (`p_kind`, `p_table`, `p_fields`, `p_values`) in that order to built-in `text`, `text`, `text[]`, `jsonb`, returning `void`. Implicit all-IN modes and explicit all-IN modes are equivalent. Checking only the type signature would miss a name permutation changing the helper body's bindings. The gate and row functions take no arguments and return trigger. These restrictions follow the executable profile, rather than claiming that every excluded attribute is intrinsically unsafe [5].

All six control relations—`epochs`, `global_clock`, `registry`, `tickets`, `effects` and `ledger`—must be permanent ordinary nonpartitioned tables owned by the non-login `oc_owner`. Both RLS and FORCE RLS are refused, including configurations whose predicates happen to allow the current executor. PostgreSQL's default-deny behavior can hide existing epoch rows despite a table SELECT grant [6]. The profile checks exact columns, order, types, non-nullness, defaults, generated/identity attributes, collation and required constraints. Expected primary keys, checks and the ledger foreign key must be validated and non-deferrable. The singleton global clock must exist. A configured active registry remains a separate service requirement.

Effective table and column privileges are checked. `oc_writer` cannot access control tables; `oc_executor` can read them and insert only tickets, effects and ledger entries. Runtime roles cannot alter retained control rows, truncate protected tables, change triggers, acquire owner membership, set replication bypass or gain elevated role flags. The qualifier also restricts rewrite rules, inheritance, extra triggers, function inventory and SECURITY DEFINER entry points. Unsupported foreign storage, partition routing, DDL, restore and other mutation paths are not made safe merely by naming them unsupported: the admitted role boundary must exclude them.

`qualify_live(..., require_authenticated=True)` requires a genuine `oc_executor` login with matching `session_user` and `current_user`. Role switching within an administrative session is insufficient. Catalog-only checking and body-only helper checks are explicitly weaker diagnostics. Qualification is a deployment

checkpoint under a trusted administrator, SQL engine, extractor, restricted parser and executor protocol. It does not protect against a hostile administrator changing the installation afterward, and a rejection does not prove that every rejected installation admits an exploit.

S2. Conditional observation and execution argument

Let S be a committed database state, d an observation descriptor, $O_d(S)$ its semantic contents, and $E_d(S)$ its retained epoch. Let $R_u(d, S, s)$ be the value observed by executor role u through snapshot S . A certificate retains a finite descriptor set D and captured epoch values. That set is **sufficient** when equality of its observation contents, together with equal policy inputs and bindings, preserves the captured policy result. Sufficiency is an evaluator obligation; the installed invalidator checker does not discover arbitrary policy dependencies.

The current implementation is scoped to one fixed, nonrestored database lifetime. It has no explicit database-incarnation field. Restoring old marker history, cloning a database or importing evidence from another history requires a separately justified invalidation/generation lifecycle and is outside the supported scope. Crash recovery preserving the same committed history is conceptually different from restoring an earlier history.

Proposition S1: indistinguishable guard inputs. Suppose a safe and an unsafe admission history give a deterministic guard the same complete input: certificate, parameters, versions, visible markers and every other value actually used. The guard returns the same result in both histories. It cannot both accept the safe history and reject the unsafe history. The corresponding randomized guard has the same acceptance distribution for identical inputs. This follows directly from functional equality. A sound guard can conservatively reject both, obtain a distinguishing observation or reevaluate sufficient current state. The proposition does not apply when an omitted global clock or fresh policy read actually distinguishes the histories.

Proposition S2: conditional reuse and local composition. Assume:

1. The original decision is correct, its observations are sufficient, and mutable non-database inputs are independently checked or versioned.
2. Every admitted committed mutation changing a recorded observation atomically advances the retained epoch under that exact descriptor. Installed helper bindings, trigger attachments and admitted event semantics agree with the checked program.
3. Epochs do not decrease, disappear, reset or wrap during the supported database lifetime. Overflow aborts the transaction rather than permitting wraparound. Domain changes and corresponding epochs commit atomically.
4. Capture and admission faithfully read each true retained epoch. A missing row means genuine absence, not a row hidden by security, a view or another unsupported read path.
5. Relevant writes and admissions acquire the same transaction gate. Admission reads from a snapshot established after obtaining the gate and after preceding invalidators finish; the gate remains held through the guarded local effect and commit. The qualified installation remains stable.
6. The executor checks subject/action binding, current authority, parameters, policy and handler identifiers, expiry and prospective bounds. One-use consumption, budget accounting and the local effect commit atomically under the same ordering discipline.

Then equality of all captured and admission epochs implies equality of every recorded observation at admission. Consequently, the captured true policy decision remains applicable under its bound inputs. The sixth premise connects this observation result to the specified local protected operation.

Proof. If any recorded observation differed, at least one committed intervening mutation changed it. Exact producer cover advanced its true marker. Retention prevents that marker from returning to the captured value. Faithful reading from the post-gate snapshot exposes the advancement, contradicting equality. Sufficient observations therefore preserve the policy decision. Holding the gate excludes a relevant mutation

between validation and the protected commit; separate authority and atomic-effect checks enforce the remaining obligations. Changes later restored to their original contents may still cause conservative invalidation. This does not contradict the implication.

The result is sufficient rather than necessary: full fresh evaluation can be correct without epochs. It is a written conditional argument, not a proof-assistant development or a verification of PostgreSQL, the Python parser or arbitrary policy code. Policy configuration fingerprints and manually maintained handler generations bind trusted deployment identifiers; they do not attest arbitrary loaded executable semantics. A local transaction does not prove commitment by an external payment or notification provider.

S3. Behavioral oracle and its finite limit

A cross-role challenge compares a controlled semantic change with the marker visible to the actual executor. A committed change to O_d with unchanged $R_u(d)$ falsifies observation preservation for that trace. A marker change without semantic change may be harmless conservatism. When the policy changes from true to false, an unchanged marker demonstrates unsafe evidence reuse **if** all other guard inputs remain equal; it is not an unauthorized effect unless a protected effect is actually admitted and committed.

The challenge uses separate authenticated writer and executor sessions on a disposable installation. Writer mutations commit before the after-read. The executor observes policy truth and the relevant partition marker; an administrative oracle separately records stored epochs. Uncommitted rollback-only writer probes would not establish another session's committed observation. Fixture resets are trusted experimental preparation between trials; no certificate crosses a reset.

Proposition S3: finite positive probes do not certify unrestricted visibility. Let a terminating campaign read a finite identity set Q , with some supported descriptor d^* outside that set. An unrestricted visibility rule can reveal every epoch in Q while hiding d^* . It produces the same campaign results as an all-visible rule. Starting with no epoch for d^* , a mutation can change its semantic observation and create epoch one while its consumer-visible value remains zero. Thus every positive probe can pass while the general contract fails.

The construction also applies to a deterministic adaptive campaign: first obtain its finite transcript against all-visible storage, then reveal precisely the queried identities. For randomized probing this describes a fixed realized transcript, not a guaranteed adaptive attack against every precommitted random campaign. The limit assumes a rule language able to distinguish unprobed identities. It does not defeat a static restriction excluding RLS, a proof of the visibility predicate, or exhaustive coverage of a genuinely closed finite domain. It is a standard testing-completeness limitation specialized to this observation contract, not a novel general impossibility theorem. Related database and security metamorphic testing predates this study [16, 17, 18, 19].

The executed decoy exposes epoch descriptors only when $dep \rightarrow 3 \rightarrow 0 = \text{'PROBE'}$. INSERT, UPDATE and DELETE probes in PROBE all pass. Policy reads for target tenant T remain visible, but their epoch rows are hidden. For the hold-absence case, the executor observes true/zero before insertion and false/zero afterward while the administrator sees stored epoch one. Both the static qualifier and structural lint reject this configuration; behavioral probing alone accepts it. The static restriction therefore remains essential in this experiment.

S4. Frozen policy population, configurations and outcomes

The challenge specification was developed by a separate AI agent within the same project before it inspected the repaired qualifier. It could inspect the prior implementation and supplied review. This is a design separation, not external independent validation, public preregistration or a random enterprise sample. Two configurations are explicitly named regressions. All policies read the T partition of one typed facts relation, giving a conservative complete dependency without claiming minimal precision.

Policy identifier	Family
P01_membership_exists	Positive existence
P02_hold_absence	Negative existence / phantom insert
P03_prospective_budget	Aggregate sum with prospective debit
P04_inventory_quantity	Aggregate sum lower bound
P05_nested_manager_join	Self join and referenced-row deletion
P06_required_consent	Distinct-value coverage
P07_unique_reservation	Count upper bound
P08_latest_deployment	Maximum aggregate and extremal deletion
P09_two_distinct_approvers	Separation of duties via distinct identity
P10_regional_exclusion	Negative predicate after status update
P11_invoice_version_limit	Conjunctive row-level equality and inequality
P12_credential_minimum	Minimum aggregate with freshness threshold

These P01-style identifiers name fixtures, not bibliography entries. The 12 families crossed with nine configurations produce 108 pairs, not 108 independent deployments. Every pair reached its expected true-to-false policy transition. No setup or comparator errors occurred, and the checked sources remained unchanged during execution.

Configuration	Class	Lint	Probes	Static	Stale pairs / 12
C01_valid_default	Supported safe	Accept	Accept	Accept	0
C02_valid_extra_index	Supported safe	Accept	Accept	Accept	0
C03_safe_when_true	Unsupported safe	Accept	Accept	Refuse	0
C04_safe_allow_all_rls	Unsupported safe	Refuse	Accept	Refuse	0
C05_regression_R1_argument_binding	Unsafe	Accept	Refuse	Refuse	12
C06_regression_R2_hidden_epochs	Unsafe	Refuse	Refuse	Refuse	12
C07_probe_decoy_visibility	Unsafe	Refuse	Accept	Refuse	12
C08_row_markers_only	Unsafe	Accept	Refuse	Refuse	12
C09_epoch_update_suppression	Unsafe	Refuse	Refuse	Refuse	11

The fixed structural lint checks roles, memberships, ownership, grants, permanent relation shape, RLS, types, keys, required triggers and helper structural attributes. It does not check semantic parameter binding or invalidation coverage. It is one explicit conventional-style checklist, not a proxy for every competent human review or external tool. Behavioral qualification requires all three PROBE mutations to change both their chosen count/sum observation and marker. The conjunction requires both static qualification and those probes.

Of five unsafe configurations, lint accepts two, probes accept one, and static qualification and the conjunction accept none. All methods retain 24/24 supported-safe observation pairs. Static qualification refuses both unsupported-safe forms, lint refuses one and probes refuse neither. Static and combined qualification have identical measured acceptance/refusal outcomes; the probes provide explanatory falsification evidence, not a measured incremental classification gain. The five unsafe configurations yield 59 stale-policy pairs. Suppression produces 11 rather than 12 because an initially absent epoch is inserted successfully before subsequent UPDATE suppression becomes relevant.

The sequential runner uses separate fresh autocommit policy and marker reads with no concurrent writer between them. It therefore establishes these observation witnesses, not gate ordering, concurrent purchase safety, or 12 integrated business effect handlers.

S5. Additional evidence and execution environment

The catalog development suite qualifies a fresh valid installation before each mutation, then requalifies the mutated installation. Ordinary PostgreSQL results contain **48/48 valid controls accepted and 48/48 mutants rejected**. The known binding and hidden-epoch examples are deliberately executed after rejection to retain semantic/marker witnesses. Those diagnostics do not represent service use under an accepted deployment. A separate PostgreSQL/WASM run obtains the same paired classifications but rejects strict authenticated qualification; embedded diagnostics are not counted as server authentication evidence.

The six pre-existing server tests also pass, with zero skips: executor qualification; denied writer bypass attempts; COPY FROM STDIN invalidation; local effect and exact replay; raw-writer invalidation of prepared evidence; and raw-writer waiting on the admission gate. Their unchanged test source runs against the repaired backend. They are not evidence that the unrepaired implementation passed the new contract. The source hashes and original output are retained in `results/server_gate/legacy_server_run_manifest.json` and `legacy_six_server_tests.log`.

The ordinary server is Ubuntu PostgreSQL **16.15**, running under a real non-root guest operating-system account in QEMU **8.2.2** full-system x86-64 TCG emulation, with **two virtual CPUs, 1536 MiB** guest memory and Linux **6.8.0-138-generic**. Python **3.12.14** and Psycpg **3.3.6** connect through TCP with SCRAM authentication. Writer and executor are separate non-superuser logins without BYPASSRLS, role creation, database creation or replication privileges; owner cannot log in. A wrong writer password is rejected. Runtime role identities are recorded, not inferred from a role-switch command.

PGDATA resides on ext4 over a 2 GiB sparse raw disk exposed by virtio-blk, not tmpfs. The VM uses writeback caching with flush handling. PostgreSQL reports `fsync`, `synchronous_commit` and `full_page_writes` enabled. Infrastructure initialization persisted across a VM restart; this is not a controlled power-loss durability experiment. Client process-exit tests exercise a different failure boundary. Emulation and transport limit timing interpretation; results describe this measured environment rather than native production capacity. Signed-package metadata matches and environment records are retained under `results/server_gate/` and `docs_v04/server_environment.md`.

An independently implemented finite abstraction, `experiments_v04/observation_model.py`, imports no production generator or checker. It enumerates two descriptors, Boolean contents, four starting states and 85 sequences of zero to three writes: **680 combinations per profile**. The same combinations are reused across profiles. Of 280 changed-observation combinations, the complete abstract contract has zero unsound reuses; wrong binding has 56, hidden-unprobed visibility 140, and dropped updates, hidden-all visibility, old snapshots and marker reset each 280. These are exact bounded counts, not defect probabilities, engine exploit rates or an unbounded machine proof.

S6. Connected purchase, receipt replay and recovery scope

The fixed purchase policy retains three manually specified dependencies: the account row and spend and holds partitions for (tenant, entity, period). Exact SQL checks current available budget, currency, holds and $\text{sum}(\text{current_spend}) + \text{requested_amount} \leq \text{period_cap}$; aggregates and prospective values use PostgreSQL exact numeric arithmetic. These dependencies are not inferred from arbitrary SQL. Application principal identity is supplied by trusted authentication middleware; a principal string alone is not end-user authentication. The fixed policy has no general tenant-membership ACL.

Approval and commit begin READ COMMITTED transactions, acquire the shared advisory gate, then issue policy/control reads in subsequent statements, so waiting for the gate does not reuse its earlier statement snapshot [7]. A new selective commit reuses its captured decision only with matching exact markers and bindings; otherwise it evaluates the same native policy. The matched native mode evaluates that policy on every new commit. Both perform identical budget debit, spend append, effect insertion and one-use consumption atomically. Both retain the same installed invalidators: this isolates retained-decision behavior but is not a minimum-cost native system without epoch maintenance.

An exact retry first checks principal/action binding and then returns the historical receipt if consumption already occurred. It performs no new effect and can return that receipt after later expiry, revocation or policy activation. Such a reply is evidence of the earlier committed operation, not renewed authority. Reusing an authority identifier with a different action is refused. Expiry and effect timestamps are evaluated at the post-gate guard time; `committed_at` records that timestamp, not the physical commit instant.

Disconnecting before COMMIT should roll back all four mutations and associated marker updates; losing a response after COMMIT permits receipt retrieval. These tests concern abrupt client termination, not database-process crashes, host power loss or remote-service atomicity. Raw admitted writer DML can subsequently alter facts; purchase safety does not imply that every raw writer preserves business invariants or append-only spend history.

All **22/22** authenticated purchase checks passed. Selective and native modes produced the expected atomic effect and replay behavior and both refused intervening spend of 140 followed by a proposed 80 under a cap of 150. Principal/action changes, revocation, policy and handler changes, positive-TTL expiry, suspension and intervening holds were handled as specified. An unrelated writer mutation preserved reusable evidence.

The concurrency case used separate authenticated sessions for conflicting purchases; the final state and authority ledger matched the prospective bound. A separate case observed an executor waiting for the gate before allowing the writer to commit, then checked the executor's fresh denial. Three injected pre-commit exceptions and three abrupt pre-commit client exits left no partial debit, spend, effect or consumption. A client exit after commit recovered by replaying the same receipt once. These are client interruption and transaction/replay observations, not server-power-loss or backup-restoration experiments.

The first 540-cycle campaign used a JSON-array expansion joined to the epoch table. At 10,000 rows without an intervening mutation, its selective and native cycle medians were 197.18 and 100.44 ms. That adverse result is preserved with the exact initial source. Query-plan investigations found a hash-join/sequence-scan plan at a small metadata size, but a grown-fixture diagnostic instead used indexed lookups and did not reproduce the original large slowdown. The cause of the first campaign's size-dependent excess cost therefore remains unresolved.

The revised service exposes its three fixed descriptor keys directly in a parameterized IN predicate. Only the lookup changed; native policy semantics, effects, maintained installation and the declared timing grid were retained. The revised code was frozen and all 23 purchase unit checks, 22 server checks and 540 timed cycles were rerun. These measurements are development-informed, not an independent performance holdout, and are not pooled with the initial campaign.

In the final same-run comparison at 10,000 rows, selective cycle latency was 24.0% lower for unchanged state and 32.4% lower after an unrelated write. Relevant mutations instead made selective mode slower: 99.11 versus 96.44 ms at 10,000 rows. At 100 rows with no mutation it was also slightly slower, 29.66 versus 28.53 ms. These results identify a workload-dependent regime for this implementation, without attributing all before/after differences to a proven query-plan mechanism.

A separate component experiment used ten fresh connections against a warm server. Median connection establishment was 28.91 ms and direct qualification was 215.60 ms. A service constructor measured after that qualifier in the same session took 124.05 ms; this order-biased warm measurement is not an independent cold-start cost. Eighty indexed writer operations covered 1-row and 100-row inserts, no-op updates, scope moves and deletes with all row maintenance active. For 100-row operations, the median total DML-plus-maintenance-plus-commit costs were insert 235.16 ms, noop update 461.14 ms, partition move 463.89 ms, delete 211.61 ms. These are total write costs, not epoch overhead isolated against a no-maintenance writer. Provisioning, physical cold-cache behavior and production capacity were not benchmarked.

S7. Artifact integrity and reproduction

The challenge payload was frozen at

31e1856f9494a1fa613395ba7b012717cc95e7a99660dc59f03fe9ee008906bb. The runner/comparator source hash is

4677a538768fb88269797d08759ad0389a40caaac2e2f966477fa6fe68cd5e7. Their manifests record pre-execution amendments for explicit INSERT columns and precise observation-pair terminology, with prior bytes archived. The qualifier's second freeze precedes challenge execution. Freeze files, not later prose descriptions, bind exact source versions.

Raw challenge SHA-256 is

7df8d2e428d843e7330079ae1a73c3b8977c5fe8b31de28a0c0a0e6c11b65175; raw ordinary-server catalog results hash is 015192ca5fe11ecc9c72dcbdd28c436c50301da388adc80cc07763e27eb58796. Consult experiments_v04/challenge_manifest.json, challenge_runner_freeze.json, results/v04_catalog_freeze.json, results/challenge_results.json and results/v04_catalog_regressions_postgresql.json. Hash agreement checks artifact identity; it does not establish semantic correctness, author independence or a trustworthy host.

Install the pinned Python client from requirements-server.txt. Provide an administrator DSN privately in OC_PG_TEST_ADMIN_DSN for an empty, dedicated disposable PostgreSQL cluster with the reserved roles initially absent. The reproduction driver creates random runtime credentials and unique databases, records stages and source hashes, and cleans only resources it created. It refuses reserved-role collisions and does not silently fall back to an embedded engine or count skipped server tests as success.

```
python experiments_v04/reproduce_v04_server.py --dedicated-server \
--output results/server_reproduction_new
python experiments_v04/observation_model.py \
--output results/observation_model_reproduced.json
```

The server driver runs the six-test, catalog, challenge and connected-purchase stages. --no-benchmark requests functional checks only and does not reproduce timing results. Use a fresh output directory and retain failures. Reexecution checks the functional relations and produces new measurements; concurrency winner order and timing need not reproduce byte-for-byte. Receipt replay inside the service, reexecution of experiments and hash verification of archived records are three distinct operations. Detailed commands, raw schemas and environment limitations accompany the source artifact; secrets and the private initialized database image are excluded.