A real compatibility transition

Question and test setting

Could a bounded application repair restore required behavior after a provider-library change, and could review detect a defect that successful query results concealed? The study used pandas 1.5.3, SQLAlchemy 1.4.46 and 2.0.0, and SQLite. The reporting application and checks were authored for the investigation; the provider transition and public incompatibility were real. [A1]

Comparison design

The comparisons included the original old stack, the provider upgrade alone, an initial application repair, an ordinary pandas upgrade, and a revised repair on both stacks. Keeping the ordinary upgrade as a baseline matters: custom code generation should earn its place against a simpler available remedy.

Condition	Selected primary checks passed
Original application / old stack	11 / 12
Original application / upgraded provider	1 / 12
First application repair	11 / 12
Ordinary pandas upgrade	11 / 12
Revised repair / upgraded provider	12 / 12
Revised repair / old stack	12 / 12

Interpretation

The old-stack rollback defect preceded the upgrade. It must remain a preexisting application problem. These six conditions are engineering alternatives on one transition, not independent model trials or a population-level repair success rate. The figures above were checked against retained result records during this expansion. No study tests were rerun.

What the behavioral checks examined

Evidence group	Selected observation in revised result
Q01-Q06: query behavior	Rows, types, parameters and query forms in selected fixtures
Q07: empty result	Expected columns retained with zero rows
Q08: values	Unicode and SQL null preserved
Q09: error	Expected database error propagated
T01: write persistence	A new connection could see the written row
T02: rollback	A failed operation left no partial row
R01: cleanup	Zero checked-out connections after completion

Why successful execution was insufficient

SQLAlchemy’s migration changed connectionless execution and accepted query handling. The repair needed an explicit connection and suitable executable query representation. Write persistence required a separate application transaction policy. Repairing one interface cannot silently decide when the customer’s operation should commit. [A1, A2]

Concrete repair pattern

For reads, the study moved from an implicit integration call to explicit connection ownership and executable query handling. For writes, it used a declared unit of work that commits or rolls back as a unit. Positional driver parameters needed their own selected path. These are scoped engineering choices, not a universal migration recipe.

What the counts mean

The twelve checks share an application, fixtures and requirements. Passing them establishes the observed behavior of that candidate under those conditions. It does not show that change discovery was complete, that every affected call was identified, or that unseen provider changes can be repaired.

The defect hidden by correct-looking rows

Initial candidate

The first candidate returned a chunk iterator from inside a connection context. The function exited the context before the consumer finished iterating. SQLite could still return expected rows, so a row-only test could pass while resource ownership was wrong.

Additional challenge

The review checked the pool while iteration was in progress. The first candidate had zero checked-out connections when the test required one. This was a direct observation of the lifetime defect, not an inference from a final answer. The revision kept the connection active until exhaustion, explicit close or error. [A3]

Challenge	First candidate	Revised candidate
Connection retained during iteration	FAIL	PASS
Explicit early close returns connection	PASS	PASS
Error propagates and connection returns	PASS	PASS
Parameters remain data; table unchanged	PASS	PASS

Mechanism of the change

The defective pattern returned the iterator and let the context end. The revised generator yielded chunks while still inside that context. This moved ownership to the lifetime of consumption. Review changed the implementation and then checked both the specific failure and neighboring cleanup/error behaviors.

Transfer to a second consumer

The study also exercised the real pandasql module. Four authored query checks failed before the scoped migration rule and passed afterward. The rule and tests were exposed to the author. The result demonstrates reuse of a known lesson within this transition; it does not establish unseen-change adaptation.

Execution, recovery and the next proof

A recorded orchestration failure

An early research adapter ran old-stack checks but omitted required effect-status fields. Its behavioral observation remained 11/12 while orchestration acceptance stayed uncertain with a reservation. The grant was revoked and the evidence retained. The original operation was not blindly redispatched. A linked successor completed nine remaining bounded database stages. [A4]

Replay

Replaying a completed successor stage returned its historical completion without another dispatch. This matters because maintenance has separate effects: compute a patch, write a branch, send a pull request and obtain acceptance. Retrying an uncertain external effect can duplicate work. This experiment establishes selected local accounting and replay behavior, not recovery of a customer pull request.

What HSDLG and the assistant each supplied

Existing HSDLG controls supplied bounded local execution, retained source/effect records and replay/accounting behaviors. The assistant supplied semantic interpretation and repair code. The study did not isolate their causal contributions. EMSM’s selected native scope did not admit an API-risk calculation, so no API reliability advantage is attributed to it.

Next qualification

Freeze diverse historical provider changes and both affected and unaffected consumers. Measure detection and impact precision/recall before repair quality. Compare a competent direct workflow with the reviewed workflow under equal inputs and budgets. A consented customer pilot should measure rejected patches, engineering review effort, time to acceptance and recurring use.

Sources and diligence

[A1] Making Self Maintaining APIs Real, study design and results; supplied project archive.
[A2] SQLAlchemy 2.0 migration guide: https://docs.sqlalchemy.org/en/20/changelog/migration_20.html
[A3] Retained challenges_v1.json and challenges_v2.json; read for this expansion.
[A4] Methods Companion and retained execution records. Primary result counts cross-checked against summary.json and repair_v2.json. Same-assistant production and review; exposed tests; one known transition.

Additional technical documentation, standards, test records and implementation details are available for scoped diligence under an appropriate NDA or contract.