Evidencing behavior in AI-assisted database migration

Elena Marković Adrian Wójcik
Maya Srinivasan Julian Hartmann

Abstract

AI-assisted migration can preserve the appearance of SQL while changing the behavior that an application depends on. Compilation, row counts, and a plausible explanation are weak acceptance oracles for changes involving nulls, duplicate rows, rounding, time zones, transaction ownership, retries, and contention. This paper develops a behavior-evidencing method for migrations targeting Oracle AI Database 26ai. Each case connects an explicit business contract to a small executable fixture, a discriminating observation, and a retained evidence record. Seven engineering cases include purchase-order ingestion, financial reconciliation, monetary settlement, event cutoffs, atomic application work, retryable debits, and concurrent account updates.

The contribution is a reproducible SQL and PL/SQL case suite and an acceptance protocol for reviewing AI-generated candidates. The protocol separates expected outcomes, execution observations, and approval decisions. Source and candidate identities, database release, client and session settings, and complete assertion logs form the evidence boundary. The cases expose why equal summaries or successful calls cannot establish application equivalence. They also identify the limits of local evidence: a serial fixture does not prove a concurrent schedule, and a database transaction does not establish exactly-once external effects.

Keywords: database migration; behavioral evidence; Oracle; SQL; PL/SQL; idempotency; concurrency

1 Introduction

A migration changes an executable contract, not merely a database artifact. An application may distinguish an absent purchase-order reference from an explicitly blank field, or depend on line-level rounding rather than rounding an invoice total. An AI rewrite can alter either rule without introducing a syntax error. The useful question is therefore which observable would distinguish a correct migration from a plausible but incorrect candidate.

1.1 Contributions

We specify seven concrete cases, a deterministic assertion harness, two-session schedules, and a source-bound evidence manifest. The units of review are a stated contract and a candidate digest; the model does not approve its own output. Related SQL research motivates testing denotations across discriminating inputs and constraining generated syntax, while leaving application-specific acceptance to independent evidence. [16], [17]

The cases define exact expected observations. A target-specific implementation supplies measured observations by executing the illustrated logic against the declared database and release update. The publication does not substitute expected values for a completed Oracle execution record.

2 The unit of behavioral evidence

For a candidate C, define evidence as the tuple of contract, environment, fixture, ordered operations, oracle, and observation. A passing result means that C met this contract on this fixture under this environment. It is not a universal proof. The bundle must preserve enough information to repeat the decision when a driver, release update, session policy, or schema definition changes.

Implement these cases in a new dedicated scratch schema with CREATE SESSION, CREATE TABLE, CREATE PROCEDURE, and a tablespace quota. Use only BE_ objects and avoid CREATE OR REPLACE. Existing objects must cause setup to stop rather than be overwritten. DDL belongs to setup because Oracle commits before a syntactically valid DDL statement and after successful DDL. DDL must not be mixed into rollback evidence. [10]

2.1 Assertion and runner contract

A serial implementation should set autocommit off, spool the entire run, compile the package and procedures, check USER_ERRORS, capture environment values, and run C1 through C6. The assertion procedure below rejects null inputs as well as unequal numbers. This avoids the common defect in which an UNKNOWN Boolean silently bypasses a failing assertion. Define its package declaration and supporting tables during schema setup.

```
procedure eq (
  a number, e number, label varchar2) is
begin
  if a is null or e is null or a <> e then
    raise_application_error(-20001, label);
  end if;
  dbms_output.put_line('PASS ' || label);
end;
```

Run the bundle from its directory in SQL*Plus or compatible SQLcl script mode. Preserve the final completion marker and the process outcome, not selected PASS lines. SQL*Plus error codes may be truncated by an operating system; a zero process status alone does not prove that every assertion ran. Compilation errors are queried explicitly because a CREATE statement can leave an invalid PL/SQL object without producing the same client behavior as a failed anonymous block.

2.2 Isolation of observations

Serial cases roll back their fixture data. Session-changing tests restore the original settings on success and handled failure. Concurrent tests use a committed reset, two independent connections, and explicit synchronization points. Run each schedule only after both sessions are idle. Save source hashes before execution and archive the resulting logs under a new run identifier.

3 C1: preserve input presence

Purchase-order ingestion may accept a missing external reference but reject an explicitly supplied blank. In Oracle, a zero-length character value is currently treated as null. A target VARCHAR2 column alone cannot recover the distinction after both inputs have been coerced. Preserve a presence flag from the source representation or API payload before the value crosses that boundary. [1]

The BE_IN fixture stores absent, explicit blank, and populated references. The rule is not that every null is invalid; it is that a supplied field must contain a value. A proposed AI simplification to IS NULL is syntactically valid but rejects both IDs 1 and 2. The presence-aware predicate rejects only ID 2. The setup constraint prevents a non-null value from being marked absent.

```
-- C1: presence survives empty-string folding.
insert into be_in values (1, null, 'N');
insert into be_in values (2, '', 'Y');
insert into be_in values (3, 'PO-7', 'Y');
declare
  n number;
begin
  select count(*) into n from be_in
    where ext_ref = '';
  be_assert.eq(n, 0, 'C1 equality UNKNOWN');
  select count(*) into n from be_in
    where ext_ref is null;
  be_assert.eq(n, 2, 'C1 null conflation');
  select count(*) into n from be_in
    where present_flag = 'Y'
      and ext_ref is null;
  be_assert.eq(n, 1, 'C1 explicit blank');
  select min(id) into n from be_in
    where present_flag = 'Y'
      and ext_ref is null;
  be_assert.eq(n, 2, 'C1 rejected row');
end;
/
rollback;
```

3.1 The discriminating observation

Equality with an empty literal is not a null test. Its expected selected-row count is zero because the comparison cannot be TRUE for a null value. IS NULL selects two rows, which is correct as a storage observation and insufficient as the business rule. The presence-aware count and identifier jointly establish that the explicit blank was identified. Checking only the count could pass after selecting a different row.

In a heterogeneous migration, the presence bit must come from a source that still preserves the distinction. Deriving it from the already-coerced Oracle column cannot reconstruct lost information. For an Oracle-to-Oracle upgrade, this case is relevant to changed ingestion code or an AI rewrite of validation logic; it is not a new 26ai null rule. Archive the input-mapping decision as part of the contract.

4 C2: evidence must preserve multiplicity

A financial reconciliation may contain repeated equal entries that represent separate events. A migration that removes or adds a duplicate can preserve row count, total amount, and the set of distinct rows. C2 uses source amounts 10, 10, 20, 30, 30 and target amounts 10, 20, 20, 20, 30. Each side has five rows totaling 100. Both directional MINUS queries are empty because ordinary MINUS compares distinct rows. [2]

The tuple includes key, amount, and memo; the memo is null in this fixture. Compare a bag of complete projected tuples by assigning a positive contribution to every source row and a negative contribution to every target row. UNION ALL preserves contributions and GROUP BY puts equal tuples, including the null-valued memo group, together. A nonzero signed count identifies a multiplicity mismatch.

```
select k, amount, memo, sum(d) delta
from (
  select k, amount, memo, 1 d from be_src
  union all
  select k, amount, memo, -1 from be_dst)
group by k, amount, memo
having sum(d) <> 0;
```

The complete 02_bags.sql script seeds both sides and asserts five rows, total 100, zero differences in both set directions, and three mismatch groups. Its ordered diagnostic query returns the following signed deltas; positive values mean excess source multiplicity.

```
-- amount  delta
-- 10      +1
-- 20      -2
-- 30      +1
```

4.1 Scope of reconciliation

A digest is useful for locating a suspect partition but is not a collision-free proof of row equivalence. The comparison projection must include every contract-relevant attribute, with explicit type and representation rules. Rows sampled at unrelated consistency boundaries can disagree because of legitimate intervening writes. A mismatch report therefore needs its extraction marker and fixture identity as well as the query.

The example is a counterexample to summary-based assurance, not a universal production reconciliation engine. Large objects, floating-point tolerances, collation, and partitioning require additional mappings. Null normalization must not collapse a distinction the contract preserves. For 26ai-specific alternatives such as MINUS ALL, separately verify source-version support and the intended treatment of complete tuples.

5 C3: make numeric policy explicit

Settlement code must state the rounding boundary. Three exact NUMBER values of 0.005 produce 0.03 when each line is rounded to cents and then summed. Summing first and rounding the total produces 0.02. Neither expression is inherently the correct accounting policy; an AI rewrite between them changes the policy. Oracle documents rounding for NUMBER separately from effects of binary floating-point representation. [3]

```
-- C3: monetary rounding is not distributive.
declare
  a number;
  b number;
begin
  with lines(v) as (
    select 0.005 from dual union all
    select 0.005 from dual union all
    select 0.005 from dual)
  select sum(round(v, 2)), round(sum(v), 2)
    into a, b from lines;
  be_assert.eq(a, 0.03, 'C3 line rounding');
  be_assert.eq(b, 0.02, 'C3 total rounding');
end;
/
```

5.1 A decimal grammar is part of input

A second failure mode is treating a localized display string as a stable interchange representation. The suite changes NLS_NUMERIC_CHARACTERS and checks that TO_CHAR renders 1.25 as either 1.25 or 1,25. It then parses the same comma-decimal string with an explicit format and NLS argument under both session settings. The expected numeric value remains 1.25. [4], [5]

```
select to_number('1,25', '9D99',
  'NLS_NUMERIC_CHARACTERS='',.''')
from dual;
```

The full procedure saves the initial setting and restores it on success and handled failure. Any saved separator inserted into dynamic SQL must be quoted as data, including a separator that is itself an apostrophe. A restoration path is part of the test contract: a correct assertion followed by a broken session can contaminate the next case.

5.2 Evidence and exclusions

Record the column precision and scale, the NUMBER versus floating-point choice, the rounding location, and the session snapshot. Do not compare rendered numeric strings without specifying their grammar. The fixture does not establish a tax or currency policy; that policy is supplied by the application owner. Its role is to distinguish two executable implementations that otherwise look like a harmless algebraic simplification.

6 C4: repeated wall time is not one instant

Event cutoffs are common in order capture and settlement. A repeated local wall time can denote different instants when offsets differ. The fixture contains 02:30 at +02:00, 02:30 at +01:00, and 03:30 at +01:00. Their UTC instants are 00:30, 01:30, and 02:30 respectively. The fixed offsets make this fixture independent of a named-region daylight-saving lookup. Production region-based data still requires the target time-zone file and session policy to be recorded. [6]

```
-- C4: repeated wall time, distinct instants.
insert into be_evt values (1,
  to_timestamp_tz('2026-10-25 02:30 +02:00',
    'YYYY-MM-DD HH24:MI TZH:TZM'));
insert into be_evt values (2,
  to_timestamp_tz('2026-10-25 02:30 +01:00',
    'YYYY-MM-DD HH24:MI TZH:TZM'));
insert into be_evt values (3,
  to_timestamp_tz('2026-10-25 03:30 +01:00',
    'YYYY-MM-DD HH24:MI TZH:TZM'));
declare
  n number;
begin
  select count(*) into n from be_evt
    where occurred_at < to_timestamp_tz(
      '2026-10-25 01:00 +00:00',
      'YYYY-MM-DD HH24:MI TZH:TZM');
  be_assert.eq(n, 1, 'C4 instant cutoff');
  select min(id) into n from be_evt
    where occurred_at < to_timestamp_tz(
      '2026-10-25 01:00 +00:00',
      'YYYY-MM-DD HH24:MI TZH:TZM');
  be_assert.eq(n, 1, 'C4 selected event');
  select count(*) into n from be_evt
    where cast(occurred_at as timestamp)
      < timestamp '2026-10-25 01:00:00';
  be_assert.eq(n, 0, 'C4 lossy wall cutoff');
end;
/
rollback;
```

6.1 The candidate loses information

The correct instant comparison selects ID 1 before 01:00 UTC. Casting away the time-zone information and comparing the resulting wall fields to the same UTC wall fields selects no rows in this fixture. It is a deliberately discriminating input for a candidate that replaces a typed comparison with a superficially simpler timestamp expression. The two 02:30 records demonstrate why the wall fields cannot identify an instant by themselves. [7]

The acceptance result must identify the selected event, not merely a count. A boundary fixture should also cover equality with the cutoff, fractional precision, and the chosen inclusive or exclusive relation. Session-local rendering is an output concern that must not silently become the storage or comparison contract. C4 uses TIMESTAMP WITH TIME ZONE; TIMESTAMP WITH LOCAL TIME ZONE needs its own fixture because it has different normalization and display behavior.

The bundle specifies expected observations and retains explicit source values. A production claim additionally needs actual target results, driver binding behavior, and the complete run environment. A successful fixture on fixed offsets does not validate every named-region transition or historical correction.

7 C5: error propagation is not a rollback policy

An application unit should either complete its local work or undo that local work while preserving earlier caller changes. An unhandled stored-subprogram exception does not itself guarantee rollback of the subprogram changes; the invoker or host determines the outcome. A test that relies on an unspecified client error path cannot isolate the procedure contract. C5 catches the error inside an invoking PL/SQL block so that the observable behavior is explicit. [8]

```
create procedure be_work is
begin
  savepoint be_work_start;
  insert into be_mark values (101, 'partial');
  raise_application_error(-20041, 'failure');
exception
  when others then
    rollback to be_work_start;
    raise;
end;
/
```

To compare failure behavior, define BE_BAD to insert marker 100 and raise the same error without local cleanup. Verify that marker 100 is visible after the caller catches that failure. Then roll back that observation, insert caller marker 99, call BE_WORK, and catch the preserved -20041 error. Marker 101 must be absent while marker 99 remains.

```
-- Expected after the caught BE_WORK failure:
select count(*) from be_mark where id = 101;
-- 0: local partial work was undone.
select count(*) from be_mark where id = 99;
-- 1: earlier caller work remains.
rollback;
```

7.1 Reserved state and transaction ownership

A savepoint name is transaction-scoped. Reusing it replaces the earlier marker, so this small routine reserves BE_WORK_START and excludes recursive use or a caller that uses the same name. The routine does not COMMIT and does not roll back the entire caller transaction. A full ROLLBACK in a generic error handler would satisfy local cleanup while violating caller ownership. [9], [10]

Verify the original error code as well as the data state. A handler that replaces every error with a success-like return can hide the need for application recovery. DDL, autonomous transactions, external effects, and distributed work are outside this local savepoint example and require separate contracts.

8 C6: bind a request key to its payload

A lost acknowledgment can leave a caller uncertain whether its previous transaction committed. Repeating a debit without a stable request identity can apply the business effect twice. The fixture uses BE_REQ.request_id as an immediate primary key and stores account and amount beside that key. A repeat is accepted only for the same payload. The claim and debit remain in the caller transaction. [10], [15]

The following is the complete POST routine body inside package BE_API. Supply its package declaration and supporting tables during setup, and check USER_ERRORS before running it. The input must be positive, non-null, and representable at the stated cent precision. An insufficient-balance failure removes the local claim with ROLLBACK TO, rather than leaving a key that would incorrectly suppress a later valid attempt.

```
procedure post (
    p_req varchar2,
    p_account number,
    p_amount number) is
    l_account number;
    l_amount number;
  begin
    savepoint be_post_start;
    if p_req is null or p_account is null
      or p_amount is null or p_amount <= 0
      or p_amount <> round(p_amount, 2) then
      raise_application_error(-20030, 'input');
    end if;
    begin
      insert into be_req values (
        p_req, p_account, p_amount);
    exception
      when dup_val_on_index then
        select account_id, amount
          into l_account, l_amount
          from be_req where request_id = p_req;
        if l_account <> p_account
          or l_amount <> p_amount then
          raise_application_error(
            -20031, 'payload mismatch');
        end if;
        return;
    end;
    update be_acct
      set balance = balance - p_amount
      where account_id = p_account
        and balance >= p_amount;
    if sql%rowcount <> 1 then
      raise_application_error(
        -20032, 'insufficient balance');
    end if;
  exception
    when others then
      rollback to be_post_start;
      raise;
  end;
```

8.1 Serial evidence

06_idempotency.sql makes two identical R1 calls and expects one claim and balance 90 from an initial 100. Reusing R1 with amount 11 raises -20031. Request R2 for 200 raises -20032 and leaves no R2 claim. The caller then rolls back; balance returns to 100 and the claims disappear. This last observation detects an unintended internal COMMIT.

This is a database-local retry contract, not an exactly-once guarantee for messages, payments, or other external effects. Request retention must prevent premature key reuse. The example reserves BE_POST_START, assumes READ COMMITTED for the prescribed concurrent schedule, and excludes recursive or distributed use.

9 C6: test commit and rollback branches

A serial duplicate test cannot establish what happens when the first claimant is still uncommitted. Use two independent connections to the same scratch schema with autocommit off and READ COMMITTED. Reset and commit the fixtures only while both sessions are idle. Session A claims C1 and debits 7 but holds the transaction open. Session B attempts the same key and payload. Its call is expected to wait while A retains the conflicting claim. [11], [15]

```
-- Session A, connected to the SAME schema.
-- Start only after 09_reset.sql commits.
set serveroutput on
set autocommit off
whenever sqlerror exit sql.sqlcode rollback
set transaction isolation level read committed;
exec be_api.post('C1', 1, 7);
prompt A owns C1. Leave this transaction open.
prompt Start 07b in session B and observe wait.
prompt Then choose ONE: COMMIT or ROLLBACK.
-- Do not exit this session before that choice.
```

Start B after A has returned from its call and before ending A. In the first schedule, COMMIT A. B should follow the duplicate path, verify the stored payload, and return without a second debit. In a separate reset-and-repeat schedule, ROLLBACK A. B should obtain the now-available claim and perform the debit. End B with COMMIT in either branch.

```
-- Session B, same schema; A already owns C1.
set serveroutput on
set autocommit off
whenever sqlerror exit sql.sqlcode rollback
set transaction isolation level read committed;
prompt This call waits while A retains C1.
exec be_api.post('C1', 1, 7);
commit;
declare
  n number;
begin
  select balance into n from be_acct
    where account_id = 1;
  be_assert.eq(n, 93, 'C6 concurrent debit');
  select count(*) into n from be_req
    where request_id = 'C1';
  be_assert.eq(n, 1, 'C6 concurrent one claim');
end;
/
-- Both A-COMMIT and A-ROLLBACK yield 93.
-- Their traces differ: replay vs fresh work.
```

9.1 Equal final states need different traces

Both branches should leave balance 93 and exactly one C1 claim. These final values alone cannot identify which path occurred. Retain the A event log, the B invocation boundary, the point at which A committed or rolled back, and B completion. Observe the wait rather than replacing it with a guessed elapsed-time threshold.

Repeat the changed-payload case separately: a committed request key must not authorize a different account or amount. The serial suite covers the mismatch error, while an operational extension can add a concurrent mismatch trace. Record client cancellation, lock timeouts, and unexpected errors as failed or incomplete runs. Do not remove a blocked attempt from the denominator merely because the final balance later appears correct.

A new isolation level is a new experiment. Reusing the READ COMMITTED expectation under SERIALIZABLE without an amended oracle would conflate an isolation difference with a migration defect.

10 C7: put the business condition in the write

A read-then-decide pattern can allow two callers to observe the same available balance. A separate read is not evidence that the later write remains eligible. C7 places the sufficient-balance condition in the UPDATE and captures SQL%ROWCOUNT immediately. The fixture uses account 2 with balance 100 and two attempted debits of 60. [11], [13], [14]

```
-- Session A; reset fixtures before this case.
set serveroutput on
set autocommit off
whenever sqlerror exit sql.sqlcode rollback
set transaction isolation level read committed;
declare
  affected number;
begin
  update be_acct set balance = balance - 60
    where account_id = 2 and balance >= 60;
  affected := sql%rowcount;
  be_assert.eq(affected, 1, 'C7 first debit');
end;
/
prompt Keep A open. Start 08b in session B.
prompt After B reports NOWAIT rejection,
prompt B waits in its UPDATE. COMMIT in A.
```

A performs the first debit and holds its row lock. B first issues a SELECT FOR UPDATE NOWAIT probe. The expected error is -54 while A retains the row. A normal SELECT must not be confused with this locking read; ordinary read consistency does not make every reader wait. The probe is caught and checked inside B. [12]

```
-- Session B; A holds the account-2 row lock.
set serveroutput on
set autocommit off
whenever sqlerror exit sql.sqlcode rollback
set transaction isolation level read committed;
declare
  b number;
begin
  begin
    select balance into b from be_acct
      where account_id = 2 for update nowait;
    raise_application_error(-20999, 'missing');
  exception
    when others then
      be_assert.eq(sqlcode, -54,
        'C7 NOWAIT rejects held row');
  end;
end;
/
prompt Waiting UPDATE: now COMMIT session A.
declare
  affected number;
  b number;
begin
  update be_acct set balance = balance - 60
    where account_id = 2 and balance >= 60;
  affected := sql%rowcount;
  be_assert.eq(affected, 0, 'C7 rejected');
  select balance into b from be_acct
    where account_id = 2;
  be_assert.eq(b, 40, 'C7 final balance');
end;
/
rollback;
```

10.1 Acceptance trace

After B reaches the conditional UPDATE, COMMIT A. Under the specified READ COMMITTED schedule, B is expected to affect zero rows and observe balance 40. The script saves the implicit cursor value before another SQL operation can replace it. B then rolls back its own transaction; A's committed 40 remains. [14]

The schedule establishes a specific guarded-write behavior. It does not prove fairness, absence of deadlocks across multiple accounts, or a latency objective. Transfer operations that lock several rows need a consistent lock order and their own failure injection. Preserve both rowcount and final balance: either observation alone can miss a faulty implementation or an unintended competing write.

11 From an AI proposal to an evidence decision

The assistant should produce a bounded change, its claimed contract, applicable documentation, and the tests that could disprove the claim. Approval attaches to a candidate digest, not to the model or to a conversational explanation. C1 offers a compact review: the broad null predicate is a plausible candidate, but it fails the stated presence-aware rejection rule.

```
-- Candidate: valid syntax, wrong business rule.
select id from be_in
  where ext_ref is null
  order by id;

-- Contract: reject explicitly supplied blanks.
select id from be_in
  where present_flag = 'Y'
    and ext_ref is null
  order by id;
```

With the three C1 rows present, the first query returns IDs 1 and 2; the second returns only ID 2. These queries depend on the seeded fixture and must run before 01_nulls.sql rolls it back. An evaluator should retain the candidate SQL, the contract, and the exact row identifiers, rather than record only a model-generated verdict.

11.1 Separate specification from observation

```
paper_id: RN-2026-002
suite_revision: 1.0
source_sha256: <archived source digest>
candidate_sha256: <candidate digest>
target_version_full: <captured version>
client_version: <captured client>
isolation_level: READ COMMITTED
autocommit: false
expected_before_ids: [1, 2]
expected_after_ids: [2]
actual_result_path: <immutable run log>
status: <execution and review outcome>
```

An implementation manifest should contain SHA-256 values for its SQL sources and leave execution fields unpopulated until a target run supplies them. Recompute content identities after each reviewed change without asserting execution. Archive each completed run separately; do not overwrite the evidence specification with a later result. Hashes establish artifact identity, not semantic correctness.

11.2 An independent oracle

A model-generated assertion can repeat the misunderstanding in a model-generated patch. Review the business rule and fixture before examining candidate success. Include counterexamples and withheld boundary cases; preserve rejected candidates and failed runs. Related work on constrained generation and semantic test suites supports these distinctions without validating this particular migration workflow. [16], [17], [18]

Treat schema comments and retrieved text as untrusted data. Tool permissions and approval transitions belong in conventional enforcement code. A recommendation cannot expand its own execution scope or mark its own evidence as accepted.

12 An acceptance protocol for migration behavior

Begin with the application contract and classify the change: release upgrade, platform relocation, or heterogeneous transformation. Record exactly which behavior may change. Select fixtures that distinguish the original rule from plausible incorrect candidates, including missing data, duplicated rows, boundary values, failure paths, and concurrent schedules.

Run setup and serial assertions in a fresh scratch schema. Check compilation and complete logs before scheduling two-session tests. Capture target version and patch identity, client version, NLS values, time-zone settings, character sets, isolation, autocommit, and object definitions. An environment label such as "26ai" is not a sufficient identity for reproducing a run.

Family	Required observation
Data semantics	Selected IDs and complete mismatch tuples
Numeric/time	Typed values, settings, and boundary outcomes
Transactions	Error code and local/caller state
Retries	Claim identity, payload, and retained balance
Concurrency	Ordered session trace and final assertions

Table 1: Acceptance evidence by case family.

12.1 Coverage and reporting

Report all assigned cases, attempted runs, completed assertions, failures, and exclusions. A completion marker cannot establish that the right scripts ran unless the manifest binds their hashes. Conversely, matching hashes cannot establish that a run reached its final assertion. These observations are complementary parts of the evidence chain.

For an empirical comparison of assisted and conventional workflows, use matched tasks and independently reviewed acceptance criteria. Record discovery, review, repair, inference, and infrastructure cost. Include failed attempts and avoid counting multiple attempts on one task as independent observations. This protocol specifies how to measure benefit; its case expectations are not measured performance results.

12.2 Limits and conclusion

The suite is intentionally small. It does not certify an entire application, another engine, a different isolation level, distributed transactions, external side effects, or production performance. Some behaviors also depend on driver binding and application retry policy. Add those layers to the acceptance boundary instead of assuming database-local success covers them.

Behavior evidencing turns a migration proposal into a reviewable claim with an executable counterexample and a retained observation. The central control is traceability from business intent through source and candidate identity to independent assertions. AI assistance can accelerate proposal work; it does not replace the evidence needed to accept the change.

References I

Oracle sources below are the primary authorities for database semantics used in C1-C5. Documentation was checked on 25 October 2026. A production evidence bundle should archive the applicable revision and retrieval date because a stable URL can later contain revised release guidance.

Source applicability

C1 depends on Oracle character-null semantics and on retaining presence before target coercion. C2 distinguishes ordinary set comparison from duplicate-sensitive reconciliation. C3 uses exact NUMBER literals and explicit numeric formats; it does not generalize the rounding observation to every binary floating-point representation. C4 uses fixed offsets to make its expected instants explicit. Region-based transitions require additional time-zone-file evidence.

The SQL and PL/SQL examples specify engineering fixtures and expected outputs. Implement their supporting schema, packages, and session setup before executing them. Actual compilation, errors, selected rows, and session traces must be retained from the declared target. Each case isolates behavior so that a failure can be diagnosed without a production dataset. Its reduced inputs intentionally expose a rule, not the complete complexity of a deployed application.

[1] Oracle. 2026. Nulls. Character values of zero length and three-valued comparison behavior.

[2] Oracle. 2026. Set Operators. Distinct and duplicate-preserving operators.

[3] Oracle. 2026. ROUND (number). NUMBER rounding and representation-sensitive differences.

[4] Oracle. 2026. TO_NUMBER. Explicit numeric conversion formats and NLS parameters.

[5] Oracle. 2026. TO_CHAR (number). Session-sensitive rendering and explicit decimal/group characters.

[6] Oracle. 2026. Datetime Data Types and Time Zone Support. Datetime types, offsets, session settings, and time-zone files.

[7] Oracle. 2026. CAST. Built-in type conversions and their boundaries.

[8] Oracle. 2026. Unhandled Exceptions. Stored-subprogram exception propagation and rollback responsibility.

[9] Oracle. 2026. SAVEPOINT. Transaction-scoped savepoint naming and replacement.

References II

The following sources support transaction, locking, cursor, constraint, and related AI-evaluation discussions. Research on text-to-SQL and retrieval informs the method; it does not constitute Oracle migration validation.

Artifact map

Begin an implementation with the schema, assertion package, local-work procedures, and retry API. Keep serial fixtures and assertions separate from owner and contender schedules. Establish a committed reset before concurrent cases, capture environment evidence, and preserve the serial transcript. Record before and after candidate queries alongside their behavioral contract so that later reviewers can identify precisely which transformation was evaluated.

For each target run, document execution order and expected observations. Record content hashes and keep expected fields separate from measured results. Recompute source identities after reviewed changes and retain the earlier run records. The code listings in this paper describe the cases; teams should adapt their execution harness to the declared target, privileges, release update, and application contract before collecting acceptance evidence.

Run only in a new dedicated scratch schema. Keep serial and concurrent evidence as separate run artifacts. Preserve failed and cancelled runs as well as completed assertions, and reconcile the source manifest with the exact SQL submitted to the target. This artifact boundary makes the review repeatable while leaving production-specific deployment and acceptance decisions with the responsible application and database owners.

[10] Oracle. 2026. COMMIT. Durability, caller transaction ownership, DDL commits, and uncertain acknowledgments.

[11] Oracle. 2026. Data Concurrency and Consistency. Read consistency, isolation, and automatic locking.

[12] Oracle. 2026. SELECT. FOR UPDATE and NOWAIT locking-read semantics.

[13] Oracle. 2026. UPDATE. Conditional DML and statement behavior.

[14] Oracle. 2026. Cursors Overview. Implicit cursor attributes and immediate capture of SQL%ROWCOUNT.

[15] Oracle. 2026. Integrity Constraints. Primary keys, uniqueness, foreign keys, and check constraints.

[16] Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. Related work on denotation testing across discriminating database instances.

[17] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. Related work on syntactic constraints for generated SQL.

[18] Patrick Lewis et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Related work on conditioning generation on retrieved evidence.