SQL Highlights — Automated Data-Quality Framework

HEXA VPN app · built at **Hazel Mobile** · illustrative excerpt (read-only)

> **Portfolio excerpt.** A curated sample of a larger automated DQ suite I built for the HEXA VPN app at Hazel Mobile. Demonstrates CTEs, window functions, `FILTER` aggregates, `CASE` tiering and self-validation. Figures/logic are synthetic — **no real client, user, or production data**.

```sql
-- ============================================================================
-- SQL Highlights — Automated Data-Quality Framework
-- HEXA VPN app  ·  built at Hazel Mobile  ·  illustrative excerpt (read-only)
-- ----------------------------------------------------------------------------
-- A curated sample of an automated DQ suite that reveals the TRUE success rate
-- of a metric by reclassifying "fake" successes (flaps, self-replaces, churn).
-- Events are never deleted — a caught success is reclassified to failure.
--
-- Techniques shown:  CTEs · window functions · FILTER aggregates ·
--                    CASE tiering · self-validation (PASS/FAIL).
-- No real data — figures are synthetic.  SELECT-only.
-- Parameter :app_version  (e.g. the build being scored).
-- ============================================================================


WITH base AS (
    SELECT * FROM connection_events
    WHERE app_version = :app_version
),

-- 1) Per-row context using WINDOW FUNCTIONS ------------------------------
scan AS (
    SELECT
        b.*,
        split_part(user_token, '|', 1)                    AS account_id,   -- account = id before '|'
        lower(outcome::text)                              AS outcome_l,
        -- a >60-min telemetry gap before this row (collection-continuity check)
        (occurred_at - lag(occurred_at) OVER (ORDER BY occurred_at))
            > interval '60 minutes'                       AS gap_over_60m,
        -- the SAME account's next event time (used by the self-replace check)
        lead(occurred_at) OVER (
            PARTITION BY user_token ORDER BY occurred_at)     AS next_event_at,
        -- plausible-length floor: 14s for ad-supported users, 4s otherwise
        CASE WHEN user_type = 1 THEN 14000 ELSE 4000 END      AS min_duration_ms
    FROM base b
),

-- 2) One boolean flag per check (TRUE = the row FAILS that check) ----------
flags AS (
    SELECT
        s.*,
        coalesce(received_at < occurred_at - interval '1 second', false)  AS c1_bad_timestamp,
        coalesce(total_duration_ms = 0,   false)                          AS c5_zero_duration,
        coalesce(total_duration_ms < 100, false)                          AS c6_short_handshake,
        coalesce(total_duration_ms < min_duration_ms, false)              AS c9_implausible_len,
        coalesce(gap_over_60m, false)                                     AS c8_gap,
        -- #11 self-replace bug: session was replaced AND the same account
        -- reconnects within 60s (an app defect, not a real session)
        coalesce(
            ended_reason = 'replaced_by_other_vpn'
            AND next_event_at IS NOT NULL
            AND next_event_at - occurred_at <= interval '60 seconds',
        false)                                                            AS c11_self_replace
        -- (ISO country-code, server-IP and other row checks omitted for brevity)
    FROM scan s
```

```sql
)

-- 3) HEADLINE — success % BEFORE vs AFTER the checks (FILTER aggregates) ---
SELECT
    count(*)                                                    AS events,
    count(*) FILTER (WHERE outcome_l = 'success')               AS successes_before,
    count(*) FILTER (
        WHERE outcome_l = 'success'
          AND NOT (c1_bad_timestamp OR c5_zero_duration OR c6_short_handshake
                   OR c9_implausible_len OR c8_gap OR c11_self_replace))  AS successes_after,
    round(100.0 * count(*) FILTER (WHERE outcome_l = 'success')
          / nullif(count(*), 0), 2)                             AS success_pct_before,
    round(100.0 * count(*) FILTER (
        WHERE outcome_l = 'success'
          AND NOT (c1_bad_timestamp OR c5_zero_duration OR c6_short_handshake
                   OR c9_implausible_len OR c8_gap OR c11_self_replace))
          / nullif(count(*), 0), 2)                             AS success_pct_after
FROM flags;


-- 4) PER-CHECK SUMMARY — one row per check, with a status flag (CASE) ------
--    (the pattern repeats for every check; #11 shown as an example)
SELECT
    '#11'                                                       AS check_id,
    'Self-replace bug'                                          AS check_name,
    count(*)            FILTER (WHERE c11_self_replace)         AS flagged_rows,
    count(DISTINCT account_id) FILTER (WHERE c11_self_replace)  AS users_affected,
    round(100.0 * count(*) FILTER (WHERE c11_self_replace)
          / nullif(count(*), 0), 2)                             AS pct_of_rows,
    CASE WHEN count(*) FILTER (WHERE c11_self_replace) = 0
         THEN 'OK' ELSE 'FLAGGED' END                           AS status
FROM flags;


-- 5) SELF-VALIDATION — re-derive a number a 2nd way, assert PASS/FAIL ------
--    Guards against silent miscounts; every row must read PASS.
WITH per_user AS (
    SELECT split_part(user_token, '|', 1) AS account_id, count(*) AS sessions
    FROM connection_events
    WHERE app_version = :app_version
    GROUP BY 1
)
SELECT
    'sessions add up to total rows'                             AS self_check,
    (SELECT sum(sessions) FROM per_user)                        AS value_a,
    (SELECT count(*) FROM connection_events
        WHERE app_version = :app_version)                       AS value_b,
    CASE WHEN (SELECT sum(sessions) FROM per_user)
            = (SELECT count(*) FROM connection_events
                WHERE app_version = :app_version)
         THEN 'PASS' ELSE 'FAIL' END                            AS result;
```