ChatGPT

Pipeline Coverage Report – Final Implementation

This document specifies the implementation for the Pipeline Coverage report, including storage schema, SQL computation, and Python execution logic. It assumes the existence of a `sales_target` table to provide quota values.

A) Report Storage DDL

```
-- Fact table for Pipeline Coverage metrics
CREATE TABLE IF NOT EXISTS dyno_crm.report_pipeline_coverage_fact (
    run_id UUID NOT NULL,
    tenant_id UUID NOT NULL,
    period_start DATE NOT NULL,
    period_end DATE NOT NULL,
    principal_type VARCHAR(20) NOT NULL,
    principal_id UUID NOT NULL,
    pipeline_id UUID,
    target_amount NUMERIC(14,2),
    open_amount NUMERIC(14,2) NOT NULL,
    coverage_ratio NUMERIC(10,4),
    CONSTRAINT fk_rpc_run FOREIGN KEY (run_id)
        REFERENCES dyno_crm.report_run (id) ON DELETE CASCADE,
    CONSTRAINT fk_rpc_tenant FOREIGN KEY (tenant_id)
        REFERENCES dyno_crm.report_run (tenant_id) ON DELETE CASCADE,
    CONSTRAINT ux_rpc_unique_row UNIQUE (run_id)
);

-- Indexes for report querying
CREATE INDEX IF NOT EXISTS ix_rpc_tenant_principal
    ON dyno_crm.report_pipeline_coverage_fact (tenant_id, principal_type,
principal_id);

CREATE INDEX IF NOT EXISTS ix_rpc_period
    ON dyno_crm.report_pipeline_coverage_fact (period_start, period_end);
```

B) Report Generation Query

The query below computes the coverage metrics for a single principal and period. It retrieves the target amount from `sales_target` and sums the amounts of open deals expected to close within the period. If no target row exists, the result includes a null target and coverage.

```
--
Parameters: :run_id, :tenant_id, :period_start, :period_end, :principal_type, :principal_id, :pip

WITH target AS (
    SELECT SUM(st.target_amount) AS target_amount
    FROM dyno_crm.sales_target st
    WHERE st.tenant_id = :tenant_id
      AND st.principal_type = :principal_type
      AND st.principal_id = :principal_id
      AND st.period_start = :period_start
      AND st.period_end = :period_end
      AND ( :pipeline_id IS NULL OR st.pipeline_id = :pipeline_id )
),
open_deals AS (
    SELECT
        d.amount
    FROM dyno_crm.deal d
    JOIN dyno_crm.pipeline_stage ps ON ps.id = d.stage_id
    WHERE d.tenant_id = :tenant_id
      AND ( :pipeline_id IS NULL OR d.pipeline_id = :pipeline_id )
      AND ps.stage_state NOT IN ('DONE_SUCCESS', 'DONE_FAILED')
      AND (
        COALESCE(d.expected_close_date, d.close_date) >= :period_start
        AND COALESCE(d.expected_close_date, d.close_date) < :period_end
      )
      AND (
        (
            :principal_type = 'USER' AND (
                d.owned_by_user_id = :principal_id OR d.assigned_user_id
= :principal_id
            )
        )
        OR (
            :principal_type = 'GROUP' AND (
                d.owned_by_group_id = :principal_id OR d.assigned_group_id
= :principal_id
            )
        )
      )
)
SELECT
    :run_id AS run_id,
    :tenant_id AS tenant_id,
    :period_start AS period_start,
    :period_end AS period_end,
    :principal_type AS principal_type,
    :principal_id AS principal_id,
```

```
    :pipeline_id AS pipeline_id,
    t.target_amount,
    COALESCE(SUM(COALESCE(od.amount, 0)), 0) AS open_amount,
    CASE WHEN t.target_amount IS NULL OR t.target_amount = 0 THEN NULL
         ELSE ROUND((COALESCE(SUM(COALESCE(od.amount, 0)), 0) /
t.target_amount), 4)
    END AS coverage_ratio
FROM target t
LEFT JOIN open_deals od ON true;
```

Notes:

- `sales_target` may have multiple rows (e.g. separate targets per pipeline); summing allows them to accumulate. When `:pipeline_id` is null, targets across pipelines are aggregated.
- The `open_deals` CTE filters deals by expected/actual close date within the period, current stage state, pipeline and principal. Deals without a close date are excluded when the period is defined; if desired, modify the logic to include them.
- The final query joins `target` to `open_deals` using a `LEFT JOIN` to ensure a row is returned even if there are no open deals. It computes `coverage_ratio` only when `target_amount` is present and nonzero.

C) Python Execution Code

```
import json
import psycopg2
from datetime import date

def run_pipeline_coverage_report(conn, tenant_id, period_start, period_end,
                                 principal_type, principal_id,
pipeline_id=None):
    """
    Run the Pipeline Coverage report and persist a snapshot row.
    Parameters:
        conn: psycopg2 connection
        tenant_id: UUID
        period_start: date
        period_end: date
        principal_type: 'USER' or 'GROUP'
        principal_id: UUID
        pipeline_id: optional UUID
    """
    with conn.cursor() as cur:
        # Insert run header
        cur.execute(
            """
            INSERT INTO dyno_crm.report_run
```

```
                (tenant_id, report_type, generated_at, period_start, period_end,
parameters)
            VALUES (%s, %s, NOW(), %s, %s, %s)
            RETURNING id
            """,
            (
                tenant_id,
                'PIPELINE_COVERAGE',
                period_start,
                period_end,
                json.dumps({
                    'principal_type': principal_type,
                    'principal_id': str(principal_id),
                    'pipeline_id': str(pipeline_id) if pipeline_id else None
                })
            )
        )
        run_id = cur.fetchone()[0]

        # Execute coverage SQL
        sql = open('pipeline_coverage_query.sql').read()
        cur.execute(
            sql,
            {
                'run_id': run_id,
                'tenant_id': tenant_id,
                'period_start': period_start,
                'period_end': period_end,
                'principal_type': principal_type,
                'principal_id': principal_id,
                'pipeline_id': pipeline_id
            }
        )
        result = cur.fetchone()

        # Insert fact row
        insert_sql = """
            INSERT INTO dyno_crm.report_pipeline_coverage_fact
                (run_id, tenant_id, period_start, period_end,
                 principal_type, principal_id, pipeline_id,
                 target_amount, open_amount, coverage_ratio)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        cur.execute(insert_sql, result)

        conn.commit()

    return run_id
```

Explanation:

1. The function inserts a report run with type `'PIPELINE_COVERAGE'` and serializes the parameters.
2. It executes the coverage query, passing all relevant parameters. The query returns a single row with target and open amounts and the computed ratio.
3. The row is inserted into `report_pipeline_coverage_fact` with the run ID.
4. The transaction is committed, finalizing the snapshot.
