ChatGPT

Weighted Pipeline Report – Final Implementation

This document defines the storage schema, SQL query, and Python execution logic for the Weighted Pipeline report. It follows the common reporting methodology and leverages the canonical CRM schema extracted in `pipeline_schema.sql`.

A) Report Storage DDL

```
-- Fact table for Weighted Pipeline metrics
CREATE TABLE IF NOT EXISTS dyno_crm.report_weighted_pipeline_fact (
    run_id UUID NOT NULL,
    tenant_id UUID NOT NULL,
    pipeline_id UUID,
    principal_type VARCHAR(20),
    principal_id UUID,
    open_deal_count BIGINT NOT NULL,
    total_amount NUMERIC(14,2) NOT NULL,
    weighted_amount NUMERIC(14,2) NOT NULL,
    avg_probability NUMERIC(5,4),
    CONSTRAINT fk_rwp_run FOREIGN KEY (run_id)
        REFERENCES dyno_crm.report_run (id) ON DELETE CASCADE,
    CONSTRAINT fk_rwp_tenant FOREIGN KEY (tenant_id)
        REFERENCES dyno_crm.report_run (tenant_id) ON DELETE CASCADE,
    CONSTRAINT ux_rwp_unique_row UNIQUE (run_id, pipeline_id, principal_type,
principal_id)
);

-- Indexes for query performance
CREATE INDEX IF NOT EXISTS ix_rwp_tenant_pipeline_principal
    ON dyno_crm.report_weighted_pipeline_fact (tenant_id, pipeline_id,
principal_type, principal_id);
```

B) Report Generation Query

The query below calculates weighted pipeline metrics. It selects open deals, computes effective probability per deal, and aggregates by pipeline and principal. Null amounts are treated as zero.

```
--
Parameters: :run_id, :tenant_id, :pipeline_id, :principal_type, :principal_id, :period_start, :pe

WITH open_deals AS (
```

```
    SELECT
        d.id AS deal_id,
        d.pipeline_id,
        d.amount,
        COALESCE(d.forecast_probability, ps.probability, 0) AS
effective_probability,
        d.owned_by_user_id,
        d.owned_by_group_id,
        d.assigned_user_id,
        d.assigned_group_id,
        d.expected_close_date
    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 ( :period_start IS NULL OR d.expected_close_date >= :period_start )
      AND ( :period_end IS NULL OR d.expected_close_date < :period_end )
),
filtered_deals AS (
    SELECT *
    FROM open_deals od
    WHERE (
        -- No principal filter
        :principal_type IS NULL OR :principal_id IS NULL
        OR (
            :principal_type = 'USER' AND (
                od.owned_by_user_id = :principal_id OR od.assigned_user_id
= :principal_id
            )
        )
        OR (
            :principal_type = 'GROUP' AND (
                od.owned_by_group_id = :principal_id OR od.assigned_group_id
= :principal_id
            )
        )
    )
)
SELECT
    :run_id AS run_id,
    :tenant_id AS tenant_id,
    CASE WHEN :pipeline_id IS NULL THEN NULL ELSE :pipeline_id END AS
pipeline_id,
    :principal_type AS principal_type,
    :principal_id AS principal_id,
    COUNT(*) AS open_deal_count,
    COALESCE(SUM(COALESCE(od.amount, 0)), 0) AS total_amount,
```

```
    COALESCE(SUM(COALESCE(od.amount, 0) * od.effective_probability), 0) AS
weighted_amount,
    CASE WHEN COUNT(*) > 0 THEN AVG(od.effective_probability) ELSE NULL END AS
avg_probability
FROM filtered_deals od;
```

Notes:

- The `open_deals` CTE selects deals that are open based on the current stage's `stage_state`. It also derives `effective_probability` by coalescing the deal's override and the stage default.
- The `filtered_deals` CTE applies the principal filter. When no principal parameters are provided, all open deals pass through.
- In the final aggregation, `pipeline_id` and principal fields are expressed using the input parameters. When `:pipeline_id` is null, the result row has a null `pipeline_id` to indicate aggregation across all pipelines.
- `avg_probability` is null when no deals match the filters to avoid confusing zero with absence of data.

C) Python Execution Code

```
import json
import psycopg2
from datetime import datetime

def run_weighted_pipeline_report(conn, tenant_id, pipeline_id=None,
                                 principal_type=None, principal_id=None,
                                 period_start=None, period_end=None):
    """
    Run the Weighted Pipeline report and persist results.
    Parameters:
        conn: psycopg2 connection
        tenant_id: UUID
        pipeline_id: optional UUID
        principal_type: 'USER' or 'GROUP' or None
        principal_id: optional UUID
        period_start: datetime or None
        period_end: datetime or None
    """
    with conn.cursor() as cur:
        # Insert report 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,
                'WEIGHTED_PIPELINE',
                period_start,
                period_end,
                json.dumps({
                    'pipeline_id': str(pipeline_id) if pipeline_id else None,
                    'principal_type': principal_type,
                    'principal_id': str(principal_id) if principal_id else None
                })
            )
        )
        run_id = cur.fetchone()[0]

        # Read SQL from file
        sql = open('weighted_pipeline_query.sql').read()
        cur.execute(
            sql,
            {
                'run_id': run_id,
                'tenant_id': tenant_id,
                'pipeline_id': pipeline_id,
                'principal_type': principal_type,
                'principal_id': principal_id,
                'period_start': period_start,
                'period_end': period_end
            }
        )
        rows = cur.fetchall()

        # Insert result row (there will be one row per combination)
        insert_sql = """
            INSERT INTO dyno_crm.report_weighted_pipeline_fact
                (run_id, tenant_id, pipeline_id, principal_type, principal_id,
                 open_deal_count, total_amount, weighted_amount,
avg_probability)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        for row in rows:
            cur.execute(insert_sql, row)

        conn.commit()

    return run_id
```

Explanation:

1. The function inserts a new row into `report_run` with type `'WEIGHTED_PIPELINE'` and captures the input parameters.
2. It executes the weighted pipeline query, which computes metrics for the selected pipeline and principal. The query returns exactly one row per combination (since aggregation uses provided parameters rather than grouping sets).
3. The result is inserted into `report_weighted_pipeline_fact` with the run ID. If multiple principal or pipeline filters are needed, the caller can run this function multiple times.
4. A commit finalizes the run.
