ChatGPT

Pipeline Stage Funnel Report – Final Implementation Design

This document provides the implementation details for the **Pipeline Stage Funnel** report, including storage schema, the deterministic SQL query and sample Python code for execution.

A) SQL DDL – Report Storage

1. `report_run` (shared)

Use the common `report_run` table defined in the reporting methodology. No changes are required for this report.

2. `report_pipeline_stage_funnel_fact`

Create a fact table to store per-stage results for each run:

```
CREATE TABLE IF NOT EXISTS dyno_crm.report_pipeline_stage_funnel_fact (
    run_id UUID NOT NULL,
    tenant_id UUID NOT NULL,
    pipeline_id UUID NOT NULL,
    stage_id UUID NOT NULL,
    stage_name VARCHAR(255) NOT NULL,
    stage_state VARCHAR(20) NOT NULL,
    deal_count BIGINT NOT NULL,
    total_amount NUMERIC(20,2) NOT NULL,
    avg_age_days NUMERIC(10,2),
    PRIMARY KEY (run_id, stage_id),
    CONSTRAINT fk_rpsff_run FOREIGN KEY (run_id) REFERENCES dyno_crm.report_run
(run_id) ON DELETE CASCADE
);

CREATE INDEX IF NOT EXISTS ix_rpsff_tenant_pipeline_stage
    ON dyno_crm.report_pipeline_stage_funnel_fact (tenant_id, pipeline_id,
stage_id);

CREATE INDEX IF NOT EXISTS ix_rpsff_tenant_stage_state
    ON dyno_crm.report_pipeline_stage_funnel_fact (tenant_id, stage_state);
```

B) SQL Query – Report Generation

The query below calculates deal counts, total amounts and average age per stage. Parameters:

- `:tenant_id` – tenant UUID (required)
- `:pipeline_id` – pipeline UUID (required)
- `:period_start` – optional period start
- `:period_end` – optional period end
- `:include_closed` – boolean flag (`true` includes `DONE_*` stages)

```
WITH current_stage AS (
    SELECT
        dpss.deal_id,
        dpss.pipeline_stage_id,
        dpss.entered_at,
        dpss.exited_at
    FROM dyno_crm.deal_pipeline_stage_state dpss
    WHERE dpss.tenant_id = :tenant_id
      AND dpss.pipeline_id = :pipeline_id
      AND dpss.is_current = TRUE
),
stage_defs AS (
    SELECT
        ps.id       AS stage_id,
        ps.name     AS stage_name,
        ps.stage_state
    FROM dyno_crm.pipeline_stage ps
    WHERE ps.tenant_id = :tenant_id
      AND ps.pipeline_id = :pipeline_id
)
SELECT
    :pipeline_id AS pipeline_id,
    sd.stage_id,
    sd.stage_name,
    sd.stage_state,
    COUNT(*) AS deal_count,
    SUM(COALESCE(d.amount, 0)) AS total_amount,
    AVG(
        EXTRACT(EPOCH FROM (
            COALESCE(cs.exited_at, NOW()) - cs.entered_at
        )) / 86400.0
    ) AS avg_age_days
FROM current_stage cs
JOIN dyno_crm.deal d
  ON d.id = cs.deal_id
  AND d.tenant_id = :tenant_id
JOIN stage_defs sd
```

```
  ON sd.stage_id = cs.pipeline_stage_id
WHERE
    -- Period filtering: include if stage overlaps the window
    (:period_start IS NULL OR cs.exited_at >= :period_start OR cs.exited_at IS
NULL)
    AND (:period_end IS NULL OR cs.entered_at < :period_end)
    AND (
        :include_closed
        OR sd.stage_state NOT IN ('DONE_SUCCESS', 'DONE_FAILED')
    )
GROUP BY sd.stage_id, sd.stage_name, sd.stage_state
ORDER BY sd.stage_id;
```

Notes:

- `avg_age_days` is calculated by converting the interval between `entered_at` and either `exited_at` (for completed stages) or `NOW()` (for active stages) into days.
- Deals with NULL `amount` contribute 0 to `total_amount`.
- Period filtering ensures that any stage instance overlapping the window is included.

C) Python Execution Code

The following Python function demonstrates how to execute the pipeline stage funnel report. It uses psycopg2 for database access and follows the standard report run pattern.

```
import uuid
import json
import psycopg2
from datetime import datetime

def run_pipeline_stage_funnel_report(conn, tenant_id, pipeline_id,
period_start=None, period_end=None, include_closed=False):
    run_id = uuid.uuid4()
    now_ts = datetime.utcnow()
    input_params = {
        "pipeline_id": str(pipeline_id),
        "period_start": period_start.isoformat() if period_start else None,
        "period_end": period_end.isoformat() if period_end else None,
        "include_closed": include_closed,
    }
    with conn.cursor() as cur:
        # Insert run header
        cur.execute(
            "INSERT INTO dyno_crm.report_run (run_id, tenant_id, report_type,
generated_at, period_start, period_end, input_params, status)"
            " VALUES (%s, %s, %s, %s, %s, %s, %s, 'IN_PROGRESS')",
```

```
            (run_id, tenant_id, 'pipeline_stage_funnel', now_ts, period_start,
period_end, json.dumps(input_params))
        )
        # Run aggregation query
        cur.execute(
            """
            WITH current_stage AS (
                SELECT dpss.deal_id, dpss.pipeline_stage_id, dpss.entered_at,
dpss.exited_at
                FROM dyno_crm.deal_pipeline_stage_state dpss
                WHERE dpss.tenant_id = %s
                  AND dpss.pipeline_id = %s
                  AND dpss.is_current = TRUE
            ),
            stage_defs AS (
                SELECT ps.id AS stage_id, ps.name AS stage_name, ps.stage_state
                FROM dyno_crm.pipeline_stage ps
                WHERE ps.tenant_id = %s AND ps.pipeline_id = %s
            )
            SELECT
                %s AS pipeline_id,
                sd.stage_id,
                sd.stage_name,
                sd.stage_state,
                COUNT(*) AS deal_count,
                SUM(COALESCE(d.amount, 0)) AS total_amount,
                AVG(EXTRACT(EPOCH FROM (COALESCE(cs.exited_at, NOW()) -
cs.entered_at)) / 86400.0) AS avg_age_days
            FROM current_stage cs
            JOIN dyno_crm.deal d
              ON d.id = cs.deal_id
             AND d.tenant_id = %s
            JOIN stage_defs sd
              ON sd.stage_id = cs.pipeline_stage_id
            WHERE
                (%s IS NULL OR cs.exited_at >= %s OR cs.exited_at IS NULL)
                AND (%s IS NULL OR cs.entered_at <  %s)
                AND (%s OR sd.stage_state NOT IN ('DONE_SUCCESS',
'DONE_FAILED'))
            GROUP BY sd.stage_id, sd.stage_name, sd.stage_state
            ORDER BY sd.stage_id
            """,
            (
                tenant_id, pipeline_id, tenant_id, pipeline_id, pipeline_id,
                tenant_id,
                period_start, period_start,
                period_end, period_end,
                include_closed
```

```
            )
        )
        rows = cur.fetchall()
        # Insert fact rows
        for stage_id, stage_name, stage_state, deal_count, total_amount,
avg_age_days in rows:
            cur.execute(

"INSERT INTO dyno_crm.report_pipeline_stage_funnel_fact (run_id, tenant_id,
pipeline_id, stage_id, stage_name, stage_state, deal_count, total_amount,
avg_age_days)"
                " VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)",
                (run_id, tenant_id, pipeline_id, stage_id, stage_name,
stage_state, deal_count, total_amount, avg_age_days)
            )
        # Mark run as succeeded
        cur.execute("UPDATE dyno_crm.report_run SET status = 'SUCCEEDED' WHERE
run_id = %s", (run_id,))
    conn.commit()
    return run_id
```

Error Handling: In a production environment, wrap the execution in try/except to capture exceptions, roll back the transaction and update `report_run.status` to `'FAILED'` with an error message.
