ChatGPT

Pipeline Overview Report – Final Implementation Design

This document specifies the implementation-ready artifacts for the **Pipeline Overview** report. It includes the schema for storing report results, the deterministic SQL query used to generate the data, and example Python execution code.

A) SQL DDL – Report Storage

1. `report_run` (common header)

Ensure the following table exists; it is shared across all reports:

```
CREATE TABLE IF NOT EXISTS dyno_crm.report_run (
    run_id UUID PRIMARY KEY,
    tenant_id UUID NOT NULL,
    report_type VARCHAR(100) NOT NULL,
    generated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    period_start TIMESTAMPTZ,
    period_end TIMESTAMPTZ,
    input_params JSONB,
    status VARCHAR(20) NOT NULL DEFAULT 'SUCCEEDED',
    error_message TEXT
);

CREATE INDEX IF NOT EXISTS ix_report_run_tenant_type
    ON dyno_crm.report_run (tenant_id, report_type);
```

2. `report_pipeline_overview_fact`

This table stores the aggregated results of each pipeline overview report run:

```
CREATE TABLE IF NOT EXISTS dyno_crm.report_pipeline_overview_fact (
    run_id UUID NOT NULL,
    tenant_id UUID NOT NULL,
    pipeline_id UUID,
    status VARCHAR(10) NOT NULL,
    deal_count BIGINT NOT NULL,
    total_amount NUMERIC(20,2) NOT NULL,
    PRIMARY KEY (run_id, pipeline_id, status),
```

```
    CONSTRAINT fk_rpof_run FOREIGN KEY (run_id) REFERENCES dyno_crm.report_run
(run_id) ON DELETE CASCADE
);

CREATE INDEX IF NOT EXISTS ix_rpof_tenant_pipeline
    ON dyno_crm.report_pipeline_overview_fact (tenant_id, pipeline_id);

CREATE INDEX IF NOT EXISTS ix_rpof_tenant_status
    ON dyno_crm.report_pipeline_overview_fact (tenant_id, status);
```

B) SQL Query – Report Generation

The following SQL computes the pipeline overview metrics for a given tenant, optionally restricted by period and pipeline. It should be executed with the parameters `:tenant_id`, `:period_start`, `:period_end`, and `:pipeline_id` (which may be null).

```
WITH current_stage AS (
    SELECT
        dpss.deal_id,
        dpss.pipeline_id,
        dpss.pipeline_stage_id,
        dpss.entered_at
    FROM dyno_crm.deal_pipeline_stage_state dpss
    WHERE dpss.tenant_id = :tenant_id
      AND dpss.is_current = TRUE
),
stage_state AS (
    SELECT
        cs.deal_id,
        cs.pipeline_id,
        ps.stage_state
    FROM current_stage cs
    JOIN dyno_crm.pipeline_stage ps
      ON ps.id = cs.pipeline_stage_id
     AND ps.tenant_id = :tenant_id
)

SELECT
    COALESCE(d.pipeline_id, :pipeline_id) AS pipeline_id,
    CASE
        WHEN ps.stage_state = 'DONE_SUCCESS' THEN 'WON'
        WHEN ps.stage_state = 'DONE_FAILED'  THEN 'LOST'
        ELSE 'OPEN'
    END AS status,
    COUNT(*) AS deal_count,
    SUM(COALESCE(d.amount, 0)) AS total_amount
```

```
FROM dyno_crm.deal d
JOIN stage_state ps
  ON ps.deal_id = d.id
WHERE d.tenant_id = :tenant_id
  AND (:pipeline_id IS NULL OR d.pipeline_id = :pipeline_id)
  AND (:period_start IS NULL OR d.updated_at >= :period_start)
  AND (:period_end   IS NULL OR d.updated_at <  :period_end)
GROUP BY COALESCE(d.pipeline_id, :pipeline_id), status;
```

C) Python Execution Code

The example Python function below uses `psycopg2` to run the report. It inserts a run header, executes the SQL query with the provided parameters, inserts fact rows and updates the run status. Error handling should be added in production.

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

def run_pipeline_overview_report(conn, tenant_id, period_start=None,
period_end=None, pipeline_id=None, include_inactive=True):
    """
    Executes the pipeline overview report for a given tenant and optional
filters.
    Inserts a row into report_run and corresponding fact rows.
    """
    run_id = uuid.uuid4()
    now_ts = datetime.utcnow()
    input_params = {
        "pipeline_id": str(pipeline_id) if pipeline_id else None,
        "period_start": period_start.isoformat() if period_start else None,
        "period_end": period_end.isoformat() if period_end else None,
        "include_inactive": include_inactive,
    }
    with conn.cursor() as cur:
        # Create 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_overview', now_ts, period_start,
period_end, json.dumps(input_params))
        )
        # Execute report query
        cur.execute(
```

```
            """
            WITH current_stage AS (
                SELECT dpss.deal_id, dpss.pipeline_id, dpss.pipeline_stage_id,
dpss.entered_at
                FROM dyno_crm.deal_pipeline_stage_state dpss
                WHERE dpss.tenant_id = %s
                  AND dpss.is_current = TRUE
            ),
            stage_state AS (
                SELECT cs.deal_id, cs.pipeline_id, ps.stage_state
                FROM current_stage cs
                JOIN dyno_crm.pipeline_stage ps
                  ON ps.id = cs.pipeline_stage_id
                 AND ps.tenant_id = %s
            )
            SELECT
                COALESCE(d.pipeline_id, %s) AS pipeline_id,
                CASE
                    WHEN ps.stage_state = 'DONE_SUCCESS' THEN 'WON'
                    WHEN ps.stage_state = 'DONE_FAILED'  THEN 'LOST'
                    ELSE 'OPEN'
                END AS status,
                COUNT(*) AS deal_count,
                SUM(COALESCE(d.amount, 0)) AS total_amount
            FROM dyno_crm.deal d
            JOIN stage_state ps
              ON ps.deal_id = d.id
            WHERE d.tenant_id = %s
              AND (%s IS NULL OR d.pipeline_id = %s)
              AND (%s IS NULL OR d.updated_at >= %s)
              AND (%s IS NULL OR d.updated_at <  %s)
            GROUP BY COALESCE(d.pipeline_id, %s), status
            """,
            (tenant_id, tenant_id, pipeline_id, tenant_id,
             pipeline_id, pipeline_id,
             period_start, period_start,
             period_end, period_end,
             pipeline_id)
        )
        rows = cur.fetchall()
        # Insert fact rows
        for pipeline_id_val, status, deal_count, total_amount in rows:
            cur.execute(
                "INSERT INTO dyno_crm.report_pipeline_overview_fact (run_id,
tenant_id, pipeline_id, status, deal_count, total_amount)"
                " VALUES (%s, %s, %s, %s, %s, %s)",
                (run_id, tenant_id, pipeline_id_val, status, deal_count,
total_amount)
```

```
            )
        # 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
```

This code should be wrapped with appropriate exception handling. In a failure case, it should roll back the transaction and update `report_run.status` to `'FAILED'` with an error message.
