ChatGPT

Deal Aging / At-Risk Deals Report – Final Implementation Design

This document provides the final implementation for the **Deal Aging / At-Risk Deals** report. It defines the schema for storing report results, a deterministic SQL query to compute the metrics, and example Python code to execute the report and persist its outputs.

A) SQL DDL – Report Storage

1. `report_run` (shared header)

The `report_run` table defined in other report implementations remains the single source of truth for report execution metadata. Ensure it exists as specified in previous final designs.

2. `report_deal_aging_fact`

Create the fact table for the deal aging report. Each row corresponds to a unique combination of report dimensions for a given run. Note that `threshold_days` and `cycle_threshold_days` are stored to document the parameters used to compute the metrics.

```
CREATE TABLE IF NOT EXISTS dyno_crm.report_deal_aging_fact (
    run_id UUID NOT NULL,
    tenant_id UUID NOT NULL,
    pipeline_id UUID,
    principal_type VARCHAR(5),
    principal_id UUID,
    stage_id UUID,
    threshold_days INTEGER NOT NULL,
    cycle_threshold_days INTEGER,
    at_risk_deal_count BIGINT NOT NULL,
    at_risk_amount NUMERIC(20,2) NOT NULL,
    avg_days_in_stage NUMERIC(10,2) NOT NULL,
    median_days_in_stage NUMERIC(10,2),
    total_deal_count BIGINT NOT NULL,
    total_amount NUMERIC(20,2) NOT NULL,
    percent_at_risk NUMERIC(7,4),
    PRIMARY KEY (run_id, pipeline_id, principal_type, principal_id, stage_id),
    CONSTRAINT fk_rdfa_run FOREIGN KEY (run_id) REFERENCES dyno_crm.report_run
(run_id) ON DELETE CASCADE
);
```

```
CREATE INDEX IF NOT EXISTS ix_rdfa_tenant_pipeline
    ON dyno_crm.report_deal_aging_fact (tenant_id, pipeline_id);

CREATE INDEX IF NOT EXISTS ix_rdfa_principal
    ON dyno_crm.report_deal_aging_fact (tenant_id, principal_type,
principal_id);
```

B) SQL Query – Report Generation

The following SQL computes aging metrics for a given tenant and optional filters. It assumes that `:threshold_days` is an integer and `:cycle_threshold_days` may be null. `:period_end` is used as the "as of" timestamp when provided; otherwise, `NOW()` is used. The query uses `PERCENTILE_CONT(0.5)` to compute the median days in stage.

```
WITH params AS (
    SELECT
        :tenant_id        AS tenant_id,
        :threshold_days   AS threshold_days,
        :cycle_threshold_days AS cycle_threshold_days,
        :period_start     AS period_start,
        :period_end       AS period_end,
        :pipeline_id      AS pipeline_id,
        :principal_type   AS principal_type,
        :principal_id     AS principal_id,
        :stage_id         AS stage_id
),
as_of_ts AS (
    SELECT COALESCE(params.period_end, NOW()) AS as_of_ts FROM params
),
-- Identify current stage instances for open deals and compute durations
current_deals AS (
    SELECT
        d.id                  AS deal_id,
        d.pipeline_id         AS pipeline_id,
        d.amount              AS amount,
        d.created_at          AS deal_created_at,
        d.owned_by_user_id,
        d.owned_by_group_id,
        d.assigned_user_id,
        d.assigned_group_id,
        cs.pipeline_stage_id  AS stage_id,
        cs.entered_at         AS stage_entered_at,
        ps.stage_state,
        GREATEST(0, EXTRACT(EPOCH FROM (a.as_of_ts - cs.entered_at)) / 86400)
AS days_in_stage,
        GREATEST(0, EXTRACT(EPOCH FROM (a.as_of_ts - d.created_at)) / 86400) AS
```

```
days_in_cycle
    FROM dyno_crm.deal_pipeline_stage_state cs
    JOIN dyno_crm.pipeline_stage ps
      ON ps.id = cs.pipeline_stage_id
     AND ps.tenant_id = cs.tenant_id
    JOIN dyno_crm.deal d
      ON d.id = cs.deal_id
     AND d.tenant_id = cs.tenant_id
    CROSS JOIN as_of_ts a
    JOIN params p ON true
    WHERE cs.tenant_id = p.tenant_id
      AND cs.is_current = TRUE
      AND ps.stage_state NOT IN ('DONE_SUCCESS','DONE_FAILED')
      AND (p.period_start IS NULL OR cs.entered_at >= p.period_start)
      AND (p.period_end   IS NULL OR cs.entered_at <  p.period_end)
      AND (p.pipeline_id IS NULL OR d.pipeline_id = p.pipeline_id)
      AND (p.stage_id    IS NULL OR cs.pipeline_stage_id = p.stage_id)
      AND (
            p.principal_type IS NULL
            OR (
                 p.principal_type = 'USER'  AND (d.owned_by_user_id =
p.principal_id OR d.assigned_user_id = p.principal_id)
               )
            OR (
                 p.principal_type = 'GROUP' AND (d.owned_by_group_id =
p.principal_id OR d.assigned_group_id = p.principal_id)
               )
          )
)
SELECT
    COALESCE(cd.pipeline_id, params.pipeline_id)   AS pipeline_id,
    params.principal_type                          AS principal_type,
    params.principal_id                            AS principal_id,
    COALESCE(cd.stage_id, params.stage_id)         AS stage_id,
    params.threshold_days                          AS threshold_days,
    params.cycle_threshold_days                    AS cycle_threshold_days,
    COUNT(*) FILTER (WHERE (cd.days_in_stage > params.threshold_days) OR
(params.cycle_threshold_days IS NOT NULL AND cd.days_in_cycle >
params.cycle_threshold_days)) AS at_risk_deal_count,
    SUM(CASE WHEN (cd.days_in_stage > params.threshold_days) OR
(params.cycle_threshold_days IS NOT NULL AND cd.days_in_cycle >
params.cycle_threshold_days) THEN COALESCE(cd.amount,0) ELSE 0 END) AS
at_risk_amount,
    AVG(cd.days_in_stage) AS avg_days_in_stage,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY cd.days_in_stage) AS
median_days_in_stage,
    COUNT(*) AS total_deal_count,
    SUM(COALESCE(cd.amount,0)) AS total_amount,
```

```
    CASE WHEN COUNT(*) > 0 THEN (
        COUNT(*) FILTER (WHERE (cd.days_in_stage > params.threshold_days) OR
(params.cycle_threshold_days IS NOT NULL AND cd.days_in_cycle >
params.cycle_threshold_days))::NUMERIC
        / COUNT(*)
    ) END AS percent_at_risk
FROM current_deals cd
CROSS JOIN params
GROUP BY COALESCE(cd.pipeline_id, params.pipeline_id), params.principal_type,
params.principal_id, COALESCE(cd.stage_id, params.stage_id),
params.threshold_days, params.cycle_threshold_days;
```

Explanation:

1. The `params` CTE binds the input parameters to names that can be referenced throughout the query.
2. `as_of_ts` computes the "as of" timestamp, using `period_end` if provided or `NOW()` otherwise.
3. `current_deals` joins the current stage state to deals, computes days in stage and days in cycle, and applies filters based on pipeline, principal, stage and period. Closed deals (terminal stages) are excluded.
4. The final SELECT groups by the relevant dimensions and calculates counts, sums, averages, medians and percentages. The `FILTER` clause identifies at-risk deals according to the thresholds.

C) Python Execution Code

The Python function below illustrates how to run the deal aging report. It inserts a run header into `report_run`, executes the SQL query with the provided parameters, persists the aggregated results into `report_deal_aging_fact`, and marks the run as succeeded. Proper error handling should be added in a production environment.

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

def run_deal_aging_report(conn, tenant_id, threshold_days,
                          cycle_threshold_days=None,
                          period_start=None, period_end=None,
                          pipeline_id=None, principal_type=None,
principal_id=None,
                          stage_id=None):
    """
    Executes the Deal Aging / At-Risk report for a given tenant and optional
filters.
    Inserts a row into report_run and corresponding fact rows into
report_deal_aging_fact.
```

```
    Returns the run_id for audit purposes.
    """
    run_id = uuid.uuid4()
    now_ts = datetime.utcnow()
    input_params = {
        "threshold_days": threshold_days,
        "cycle_threshold_days": cycle_threshold_days,
        "period_start": period_start.isoformat() if period_start else None,
        "period_end": period_end.isoformat() if period_end else None,
        "pipeline_id": str(pipeline_id) if pipeline_id else None,
        "principal_type": principal_type,
        "principal_id": str(principal_id) if principal_id else None,
        "stage_id": str(stage_id) if stage_id else None,
    }
    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, 'deal_aging', now_ts, period_start, period_end,
json.dumps(input_params))
        )
        # Execute report query
        cur.execute(
            open('stage_aging_query.sql').read(),  # The SQL above should be
saved to stage_aging_query.sql
            {
                'tenant_id': tenant_id,
                'threshold_days': threshold_days,
                'cycle_threshold_days': cycle_threshold_days,
                'period_start': period_start,
                'period_end': period_end,
                'pipeline_id': pipeline_id,
                'principal_type': principal_type,
                'principal_id': principal_id,
                'stage_id': stage_id,
            }
        )
        rows = cur.fetchall()
        for (pipeline_id_val, principal_type_val, principal_id_val,
stage_id_val, threshold_val, cycle_threshold_val,
             at_risk_count, at_risk_amount, avg_days_in_stage,
median_days_in_stage,
             total_count, total_amount, percent_at_risk) in rows:
            cur.execute(
                "INSERT INTO dyno_crm.report_deal_aging_fact (
                    run_id, tenant_id, pipeline_id, principal_type,
```

```
principal_id, stage_id,
                    threshold_days, cycle_threshold_days,
                    at_risk_deal_count, at_risk_amount,
                    avg_days_in_stage, median_days_in_stage,
                    total_deal_count, total_amount,
                    percent_at_risk
                ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s)",
                (
                    run_id, tenant_id, pipeline_id_val, principal_type_val,
principal_id_val, stage_id_val,
                    threshold_val, cycle_threshold_val,
                    at_risk_count or 0, at_risk_amount or 0,
                    avg_days_in_stage or 0, median_days_in_stage,
                    total_count or 0, total_amount or 0,
                    percent_at_risk
                )
            )
        # 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
```

Note: The Python function reads the SQL from an external file `stage_aging_query.sql` for clarity. In practice, embed the query string directly or load from a resource. Ensure error handling updates `report_run.status` to `'FAILED'` if an exception is raised.
