ChatGPT

Stage Velocity Report – Final Implementation

This document provides the implementation details for the Stage Velocity report. It defines the storage table, a SQL query to compute velocity metrics, and example Python code to run the report.

A) Report Storage DDL

```
-- Fact table for Stage Velocity metrics
CREATE TABLE IF NOT EXISTS dyno_crm.report_stage_velocity_fact (
    run_id UUID NOT NULL,
    tenant_id UUID NOT NULL,
    pipeline_id UUID NOT NULL,
    stage_id UUID NOT NULL,
    completed_count BIGINT NOT NULL,
    avg_duration_days NUMERIC(10,4),
    median_duration_days NUMERIC(10,4),
    current_count BIGINT NOT NULL,
    avg_current_age_days NUMERIC(10,4),
    CONSTRAINT fk_rsv_run FOREIGN KEY (run_id)
        REFERENCES dyno_crm.report_run (id) ON DELETE CASCADE,
    CONSTRAINT fk_rsv_tenant FOREIGN KEY (tenant_id)
        REFERENCES dyno_crm.report_run (tenant_id) ON DELETE CASCADE,
    CONSTRAINT ux_rsv_run_stage UNIQUE (run_id, pipeline_id, stage_id)
);

-- Indexes for fast lookups
CREATE INDEX IF NOT EXISTS ix_rsv_tenant_pipeline_stage
    ON dyno_crm.report_stage_velocity_fact (tenant_id, pipeline_id, stage_id);
```

B) Report Generation Query

The query below computes velocity metrics for each stage in a pipeline. It can restrict to a specific stage (`:stage_id`) and period window (`:period_start`, `:period_end`) when provided.

```
--
Parameters: :run_id, :tenant_id, :pipeline_id, :period_start, :period_end, :stage_id

WITH completed_instances AS (
    SELECT
        dpss.pipeline_stage_id AS stage_id,
        EXTRACT(EPOCH FROM (dpss.exited_at - dpss.entered_at)) / 86400.0 AS
```

```
duration_days
    FROM dyno_crm.deal_pipeline_stage_state dpss
    WHERE dpss.tenant_id = :tenant_id
      AND dpss.pipeline_id = :pipeline_id
      AND dpss.exited_at IS NOT NULL
      AND ( :stage_id IS NULL OR dpss.pipeline_stage_id = :stage_id )
      AND ( :period_start IS NULL OR dpss.entered_at >= :period_start )
      AND ( :period_end IS NULL OR dpss.entered_at < :period_end )
),
current_instances AS (
    SELECT
        dpss.pipeline_stage_id AS stage_id,
        EXTRACT(EPOCH FROM (NOW() - dpss.entered_at)) / 86400.0 AS age_days
    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
      AND ( :stage_id IS NULL OR dpss.pipeline_stage_id = :stage_id )
)
SELECT
    :run_id AS run_id,
    :tenant_id AS tenant_id,
    :pipeline_id AS pipeline_id,
    s.stage_id,
    COUNT(ci.stage_id) AS completed_count,
    AVG(ci.duration_days) AS avg_duration_days,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY ci.duration_days) AS
median_duration_days,
    COUNT(ci2.stage_id) AS current_count,
    AVG(ci2.age_days) AS avg_current_age_days
FROM (
    -- Generate a row for each stage to ensure stages with no data are included
    SELECT ps.id AS stage_id
    FROM dyno_crm.pipeline_stage ps
    WHERE ps.pipeline_id = :pipeline_id
      AND ( :stage_id IS NULL OR ps.id = :stage_id )
) s
LEFT JOIN completed_instances ci ON ci.stage_id = s.stage_id
LEFT JOIN current_instances ci2 ON ci2.stage_id = s.stage_id
GROUP BY s.stage_id
ORDER BY s.stage_id;
```

Notes:

- The `completed_instances` CTE filters stage instances that have an `exited_at` timestamp and optionally applies a period window. The duration is computed in days by converting seconds to days.

- The `current_instances` CTE selects rows where `is_current` is true and calculates the age of the stage instance in days.
- A subquery `s` enumerates all stages in the pipeline. This ensures that stages with no completed or current instances still produce a row with counts of 0 and null averages.
- The median duration is computed using Postgres's `PERCENTILE_CONT(0.5)` function; if performance is a concern, this column can be omitted or replaced with another statistic.

C) Python Execution Code

```
import json
import psycopg2
from datetime import datetime

def run_stage_velocity_report(conn, tenant_id, pipeline_id, period_start=None,
period_end=None, stage_id=None):
    """
    Execute the Stage Velocity report and persist results.
    Parameters:
        conn:        psycopg2 connection
        tenant_id:   UUID
        pipeline_id: UUID
        period_start: datetime or None
        period_end:   datetime or None
        stage_id:     UUID or None to analyse all stages
    """
    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,
                'STAGE_VELOCITY',
                period_start,
                period_end,
                json.dumps({'pipeline_id': str(pipeline_id), 'stage_id':
str(stage_id) if stage_id else None})
            )
        )
        run_id = cur.fetchone()[0]

        # Load SQL from file or embed directly
        sql = open('stage_velocity_query.sql').read()
```

```
        cur.execute(
            sql,
            {
                'run_id': run_id,
                'tenant_id': tenant_id,
                'pipeline_id': pipeline_id,
                'period_start': period_start,
                'period_end': period_end,
                'stage_id': stage_id
            }
        )
        rows = cur.fetchall()

        # Insert into fact table
        insert_sql = """
            INSERT INTO dyno_crm.report_stage_velocity_fact
                (run_id, tenant_id, pipeline_id, stage_id,
                 completed_count, avg_duration_days, median_duration_days,
                 current_count, avg_current_age_days)
            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. A report run header is inserted with type `'STAGE_VELOCITY'`. The parameters include the pipeline and optional stage.
2. The SQL query (stored separately in `stage_velocity_query.sql` for clarity) is executed with run and filter parameters.
3. Results are inserted into `report_stage_velocity_fact` with the run ID. Stages with no completed or current instances will have 0 counts and null durations.
4. The transaction is committed to make the report persistent.
