ChatGPT

Deal Win/Loss Analysis Report – Final Implementation Design

This document specifies the implementation-ready artefacts for the **Deal Win/Loss Analysis** report. It defines the storage schema for report results, a deterministic SQL query to generate the metrics, and example Python code for executing the report and persisting the outcome.

A) SQL DDL – Report Storage

1. `report_run` (shared header)

Ensure that the common `report_run` table exists. It is used by all reports to track execution metadata. Only one definition of this table should exist in your database:

```sql
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_deal_win_loss_fact`

This table stores the aggregated results of a deal win/loss report run. Each row corresponds to a unique combination of report dimensions for a given run:

```sql
CREATE TABLE IF NOT EXISTS dyno_crm.report_deal_win_loss_fact (
    run_id UUID NOT NULL,
    tenant_id UUID NOT NULL,
    pipeline_id UUID,
    principal_type VARCHAR(5),
-- 'USER' or 'GROUP', nullable when not filtered
```

```
    principal_id UUID,
-- matches principal_type; nullable when not filtered
    source VARCHAR(200),                -- deal source filter; nullable
    stage_id UUID,                     -- first stage entered filter; nullable
    deals_closed_won_count BIGINT NOT NULL,
    deals_closed_lost_count BIGINT NOT NULL,
    total_amount_won NUMERIC(20,2) NOT NULL,
    total_amount_lost NUMERIC(20,2) NOT NULL,
    avg_deal_size_won NUMERIC(20,2),
    avg_deal_size_lost NUMERIC(20,2),
    win_rate NUMERIC(7,4),
    PRIMARY KEY (run_id, pipeline_id, principal_type, principal_id, source,
stage_id),
    CONSTRAINT fk_rdwl_run FOREIGN KEY (run_id) REFERENCES dyno_crm.report_run
(run_id) ON DELETE CASCADE
);

CREATE INDEX IF NOT EXISTS ix_rdwl_tenant_pipeline
    ON dyno_crm.report_deal_win_loss_fact (tenant_id, pipeline_id);

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

The combination `(run_id, pipeline_id, principal_type, principal_id, source, stage_id)` uniquely identifies rows for a report run. Nullable dimension columns are permitted so that aggregated rows (e.g. across all pipelines or all sources) can be stored without conflict.

B) SQL Query – Report Generation

The following SQL computes win/loss metrics for a given tenant and optional filters. It must be executed with the parameters `:tenant_id`, `:period_start`, `:period_end`, `:pipeline_id`, `:principal_type`, `:principal_id`, `:source` and `:stage_id`. Any parameter may be null to indicate no filtering on that dimension.

```
WITH closing_instances AS (
    -- Find current stage instances that are in a terminal state and entered
within the period
    SELECT
        dpss.deal_id,
        dpss.pipeline_id,
        dpss.pipeline_stage_id,
        dpss.entered_at,
        ps.stage_state
    FROM dyno_crm.deal_pipeline_stage_state dpss
    JOIN dyno_crm.pipeline_stage ps
```

```
       ON ps.id = dpss.pipeline_stage_id
      AND ps.tenant_id = dpss.tenant_id
    WHERE dpss.tenant_id = :tenant_id
      AND dpss.is_current = TRUE
      AND ps.stage_state IN ('DONE_SUCCESS','DONE_FAILED')
      AND dpss.entered_at >= :period_start
      AND dpss.entered_at <  :period_end
),
earliest_stage AS (
    -- Determine the earliest stage instance per deal (for optional stage_id
filter)
    SELECT deal_id, MIN(entered_at) AS min_entered_at
    FROM dyno_crm.deal_pipeline_stage_state
    WHERE tenant_id = :tenant_id
    GROUP BY deal_id
),
earliest_stage_id AS (
    SELECT e.deal_id, dpss.pipeline_stage_id AS first_stage_id
    FROM earliest_stage e
    JOIN dyno_crm.deal_pipeline_stage_state dpss
      ON dpss.deal_id = e.deal_id
     AND dpss.entered_at = e.min_entered_at
     AND dpss.tenant_id = :tenant_id
)
SELECT
    COALESCE(d.pipeline_id, :pipeline_id)   AS pipeline_id,
    :principal_type                         AS principal_type,
    :principal_id                           AS principal_id,
    :source                                 AS source,
    :stage_id                               AS stage_id,
    SUM(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN 1 ELSE 0 END) AS
deals_closed_won_count,
    SUM(CASE WHEN ci.stage_state = 'DONE_FAILED'  THEN 1 ELSE 0 END) AS
deals_closed_lost_count,
    SUM(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN COALESCE(d.amount,0)
ELSE 0 END) AS total_amount_won,
    SUM(CASE WHEN ci.stage_state = 'DONE_FAILED'  THEN COALESCE(d.amount,0)
ELSE 0 END) AS total_amount_lost,
    CASE WHEN SUM(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN 1 ELSE 0 END)
> 0
         THEN AVG(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN
COALESCE(d.amount,0) END)
         END AS avg_deal_size_won,
    CASE WHEN SUM(CASE WHEN ci.stage_state = 'DONE_FAILED' THEN 1 ELSE 0 END) >
0
         THEN AVG(CASE WHEN ci.stage_state = 'DONE_FAILED' THEN
COALESCE(d.amount,0) END)
         END AS avg_deal_size_lost,
    CASE
        WHEN SUM(CASE WHEN ci.stage_state IN ('DONE_SUCCESS','DONE_FAILED')
THEN 1 ELSE 0 END) > 0
        THEN SUM(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN 1 ELSE 0 END) *
1.0
             / SUM(CASE WHEN ci.stage_state IN ('DONE_SUCCESS','DONE_FAILED')
THEN 1 ELSE 0 END)
    END AS win_rate
FROM closing_instances ci
JOIN dyno_crm.deal d
  ON d.id = ci.deal_id
 AND d.tenant_id = :tenant_id
LEFT JOIN earliest_stage_id es
  ON es.deal_id = d.id
WHERE (:pipeline_id IS NULL OR d.pipeline_id = :pipeline_id)
  AND (:source     IS NULL OR d.source = :source)
  AND (
         :principal_type IS NULL
         OR (
              :principal_type = 'USER'  AND (
                   (d.owned_by_user_id = :principal_id) OR (d.assigned_user_id
= :principal_id)
              )
             )
         OR (
              :principal_type = 'GROUP' AND (
                   (d.owned_by_group_id = :principal_id) OR
(d.assigned_group_id = :principal_id)
              )
             )
      )
  AND (
         :stage_id IS NULL
         OR es.first_stage_id = :stage_id
      )
GROUP BY
COALESCE(d.pipeline_id, :pipeline_id), :principal_type, :principal_id, :source, :stage_id;
```

Explanation:

1. `closing_instances` identifies current stage instances of deals that are in a terminal state (`DONE_SUCCESS` or `DONE_FAILED`) and entered within the reporting window. Because each deal has exactly one current stage instance (enforced by the schema), this ensures we count each closed deal once and only within the specified period.
2. `earliest_stage` **/** `earliest_stage_id` compute the first stage entered by each deal. This enables optional filtering by the initial stage (`:stage_id`). If `stage_id` is null, all deals are included.

3. The main SELECT joins deals to closing instances and applies optional filters for pipeline, principal (owner/assignee), source and initial stage. It groups by the relevant dimensions and computes counts, sums, averages and win rate.

C) Python Execution Code

The Python function below demonstrates how to run the deal win/loss analysis report. It uses `psycopg2` to connect to PostgreSQL, inserts a run header, executes the report query, inserts fact rows and marks the run as succeeded. In production, error handling should update the run status to `'FAILED'` and record the error message.

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

def run_deal_win_loss_report(conn, tenant_id, period_start, period_end,
                             pipeline_id=None, principal_type=None,
principal_id=None,
                             source=None, stage_id=None):
    """
    Executes the Deal Win/Loss Analysis report for a given tenant and optional
filters.
    Inserts a row into report_run and corresponding fact rows into
report_deal_win_loss_fact.
    Returns the run_id for audit purposes.
    """
    run_id = uuid.uuid4()
    now_ts = datetime.utcnow()
    input_params = {
        "pipeline_id": str(pipeline_id) if pipeline_id else None,
        "principal_type": principal_type,
        "principal_id": str(principal_id) if principal_id else None,
        "source": source,
        "stage_id": str(stage_id) if stage_id else None,
        "period_start": period_start.isoformat() if period_start else None,
        "period_end": period_end.isoformat() if period_end else None,
    }
    with conn.cursor() as cur:
        # Insert run header with status IN_PROGRESS
        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_win_loss', now_ts, period_start,
period_end, json.dumps(input_params))
```

```
        )
        # Execute the report query
        cur.execute(
            """
WITH closing_instances AS (
    SELECT dpss.deal_id, dpss.pipeline_id, dpss.pipeline_stage_id,
dpss.entered_at, ps.stage_state
    FROM dyno_crm.deal_pipeline_stage_state dpss
    JOIN dyno_crm.pipeline_stage ps
      ON ps.id = dpss.pipeline_stage_id AND ps.tenant_id = %s
    WHERE dpss.tenant_id = %s
      AND dpss.is_current = TRUE
      AND ps.stage_state IN ('DONE_SUCCESS','DONE_FAILED')
      AND dpss.entered_at >= %s
      AND dpss.entered_at <  %s
),
earliest_stage AS (
    SELECT deal_id, MIN(entered_at) AS min_entered_at
    FROM dyno_crm.deal_pipeline_stage_state
    WHERE tenant_id = %s
    GROUP BY deal_id
),
earliest_stage_id AS (
    SELECT e.deal_id, dpss.pipeline_stage_id AS first_stage_id
    FROM earliest_stage e
    JOIN dyno_crm.deal_pipeline_stage_state dpss
      ON dpss.deal_id = e.deal_id AND dpss.entered_at = e.min_entered_at AND
dpss.tenant_id = %s
)
SELECT
    COALESCE(d.pipeline_id, %s) AS pipeline_id,
    %s AS principal_type,
    %s AS principal_id,
    %s AS source,
    %s AS stage_id,
    SUM(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN 1 ELSE 0 END) AS
deals_closed_won_count,
    SUM(CASE WHEN ci.stage_state = 'DONE_FAILED'  THEN 1 ELSE 0 END) AS
deals_closed_lost_count,
    SUM(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN COALESCE(d.amount,0) ELSE
0 END) AS total_amount_won,
    SUM(CASE WHEN ci.stage_state = 'DONE_FAILED'  THEN COALESCE(d.amount,0) ELSE
0 END) AS total_amount_lost,
    CASE WHEN SUM(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN 1 ELSE 0 END) >
0
         THEN AVG(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN
COALESCE(d.amount,0) END)
    END AS avg_deal_size_won,
    CASE WHEN SUM(CASE WHEN ci.stage_state = 'DONE_FAILED' THEN 1 ELSE 0 END) >
0
         THEN AVG(CASE WHEN ci.stage_state = 'DONE_FAILED' THEN
COALESCE(d.amount,0) END)
    END AS avg_deal_size_lost,
    CASE
        WHEN SUM(CASE WHEN ci.stage_state IN ('DONE_SUCCESS','DONE_FAILED') THEN
1 ELSE 0 END) > 0
        THEN SUM(CASE WHEN ci.stage_state = 'DONE_SUCCESS' THEN 1 ELSE 0
END)::NUMERIC
             / SUM(CASE WHEN ci.stage_state IN ('DONE_SUCCESS','DONE_FAILED')
THEN 1 ELSE 0 END)
    END AS win_rate
FROM closing_instances ci
JOIN dyno_crm.deal d
  ON d.id = ci.deal_id AND d.tenant_id = %s
LEFT JOIN earliest_stage_id es
  ON es.deal_id = d.id
WHERE (%s IS NULL OR d.pipeline_id = %s)
  AND (%s IS NULL OR d.source = %s)
  AND (
         %s IS NULL
         OR (
              %s = 'USER'  AND ((d.owned_by_user_id = %s) OR (d.assigned_user_id
= %s))
             )
         OR (
              %s = 'GROUP' AND ((d.owned_by_group_id = %s) OR
(d.assigned_group_id = %s))
             )
      )
  AND (%s IS NULL OR es.first_stage_id = %s)
GROUP BY COALESCE(d.pipeline_id, %s), %s, %s, %s, %s
""",
            (
                tenant_id, tenant_id, period_start, period_end,
                tenant_id, tenant_id,
                pipeline_id, principal_type, principal_id, source, stage_id,
                tenant_id,
                pipeline_id, pipeline_id,
                source, source,
                principal_type,
                principal_type, principal_id, principal_id,
                principal_type, principal_id, principal_id,
                stage_id, stage_id,
                pipeline_id, principal_type, principal_id, source, stage_id
            )
        )
        rows = cur.fetchall()
        # Insert fact rows
        for (pipeline_id_val, principal_type_val, principal_id_val, source_val,
stage_id_val,
             won_count, lost_count, total_won, total_lost, avg_won, avg_lost,
win_rate) in rows:
            cur.execute(
                "INSERT INTO dyno_crm.report_deal_win_loss_fact (
                    run_id, tenant_id, pipeline_id, principal_type,
principal_id, source, stage_id,
                    deals_closed_won_count, deals_closed_lost_count,
                    total_amount_won, total_amount_lost,
                    avg_deal_size_won, avg_deal_size_lost,
                    win_rate
                ) VALUES (%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,
                    source_val, stage_id_val,
                    won_count or 0, lost_count or 0,
                    total_won or 0, total_lost or 0,
                    avg_won, avg_lost,
                    win_rate
                )
            )
        # 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 uses the same filtering and grouping logic as the SQL query. It serialises input parameters into JSON for the `report_run` row and ensures that all numeric fields are inserted even when null (defaults to zero). Real implementations should include proper exception handling to set `status = 'FAILED'` if an error occurs.
