ChatGPT

High-Level Reporting Methodology

This document defines a **common framework** for tenant-scoped, snapshot-based reporting in the Dyno CRM Deal & Pipeline domain. All report implementations MUST conform to this methodology. The goals of this framework are:

- **Determinism** – given the same input parameters (tenant, period, options), a report run must produce identical results.
- **Isolation** – each report run is scoped to a single tenant and cannot leak data across tenants.
- **Traceability** – each report run is persisted with its inputs, outputs and timestamps to support auditing and trend analysis.
- **Reusability** – shared mechanisms (e.g. report identifiers, period handling) are defined once and reused by all reports.

Report Identification

Every report type is identified by a **stable** `report_type` **string** (e.g. `pipeline_overview`, `stage_conversion`, `weighted_pipeline`). This identifier is used in storage tables and execution code to differentiate report logic.

Report Run Header

Each execution of a report produces a single row in the `report_run` table. This header captures who, what, and when:

Column	Type	Description
`run_id`	UUID (PK)	Unique identifier for the report run
`tenant_id`	UUID	Tenant whose data is reported
`report_type`	VARCHAR	Stable identifier for the report implementation
`generated_at`	TIMESTAMPTZ	Timestamp when the report was generated
`period_start`	TIMESTAMPTZ?	Optional start of the reporting window
`period_end`	TIMESTAMPTZ?	Optional end of the reporting window
`input_params`	JSONB	JSON representation of other input parameters
`status`	VARCHAR	`SUCCEEDED` or `FAILED`; indicates run outcome
`error_message`	TEXT	Populated if `status=FAILED`

Constraints:

- (`tenant_id`, `run_id`) is unique.
- `report_type` is validated against a known list of report names.
- `generated_at` defaults to now() if not supplied.

Report Fact Tables

Each report defines its own **fact table** to store computed metrics. A fact table contains one row per logical grouping (e.g. per stage, per owner) for a specific run. Fact tables always include the following base columns:

- `run_id` – foreign key to `report_run`.
- `tenant_id` – included to allow multi-tenant partitioning and enforcement.
- Dimension columns specific to the report (e.g. `pipeline_id`, `stage_id`, `principal_id`).
- Metric columns specific to the report (e.g. counts, sums, averages).

Fact tables MUST NOT include raw transactional data (e.g. JSONB copies of deals or contacts). Instead, they store aggregated numbers suitable for UI consumption and trend calculations.

Execution Parameters

When a report is executed, it accepts the following **standard parameters**:

- `tenant_id` – UUID for which tenant the report is generated (required).
- `period_start` / `period_end` – optional timestamps defining the inclusive/exclusive reporting window. If omitted, the report is assumed to cover "all time" or a sensible default window (e.g. current week).
- `options` – optional JSON object containing report-specific filters or flags (e.g. pipeline selection, include_closed).

The combination of `tenant_id`, `report_type` and the full set of input parameters uniquely identifies a report run.

Time Windows and Periods

Reports that require time windows (e.g. weekly conversions, monthly pipeline coverage) must use a consistent definition of **period_start** and **period_end**:

- `period_start` is inclusive; `period_end` is exclusive.
- If `period_start` and `period_end` differ by exactly one calendar month, the report is considered "monthly." Similar logic applies for weekly or quarterly.
- If no period is supplied, reports SHOULD default to a configurable period (e.g. last 90 days) or treat all historical data as a single period.

Tenant Enforcement

All report queries must include a `WHERE tenant_id = :tenant_id` clause to prevent cross-tenant data leakage. Fact tables also include `tenant_id` to support partitioning and enforcement of tenant boundaries.

Naming Conventions

Report tables and files follow a predictable naming scheme:

- **Design documents**: `<report_name>_design.md`, `<report_name>_design_details.md`, `<report_name>_design_final.md`.
- **Fact table names**: `report_<report_name>_fact` (snake_case) and `report_run` for the common header.
- **Python modules**: `reports/<report_name>_executor.py` (if persisted in application codebase; not created automatically here).

Execution Flow

The high-level flow for running a report is:

1. **Start run**: insert a row into `report_run` with `status='IN_PROGRESS'` (or leave null until completion).
2. **Compute**: execute the deterministic SQL query for the report using the supplied parameters.
3. **Insert results**: insert one or more rows into the report's fact table, referencing the run_id.
4. **Complete run**: update `report_run.status` to `SUCCEEDED` and record `generated_at` if not already set.
5. **Error handling**: on failure, update `report_run.status` to `FAILED` and set `error_message`.

Python execution code must encapsulate these steps in a transaction (or explicit try/catch) to ensure atomicity; either the report run and all facts are persisted, or nothing is.

Versioning & Trends

Because each report run is persisted with a timestamp and parameters, it is trivial to compute trends by joining or aggregating across runs. Reports should not store derived trend data themselves; instead, UI or analytics layers can compute trends by comparing successive runs for the same tenant, report type and dimension values.

This methodology underpins all report designs defined in subsequent documents. Any deviations must be explicitly justified in the report's design and implementation.
