**Rewards Support Tool Enhancement - Strategy Evaluation**

**Overview**

The Rewards Support Tool fetches promotion data using the GetPromotionExecutionDetails (GPED) API, which relies on the PromotionExecutionDetails (PED) database. While it works well for event-based use cases, **it struggles with retail cases** due to multiple shipments and partial cashback issuances. This document evaluates two approaches to enhance the tool while considering backward compatibility, latency, schema changes, and API versioning.

**Approaches Considered**

|  |  |  |
| --- | --- | --- |
| **Approach** | **Description** | **Key Considerations** |
| **1: Expand PED Schema** | Modify the PED database to store granular reward issuance details, ensuring GPED API can fetch and display complete reward data. | Requires schema changes, DAO layer modifications, and API response updates. |
| **2: Aggregate Data from Multiple Databases** | Query PED + LineItemsBenefitTracker and other databases dynamically instead of modifying PED. | Requires API changes, increased query complexity, and potential caching strategies. |

**Detailed Comparison**

|  |  |  |
| --- | --- | --- |
| Factor | Approach 1: Expand PED Schema | Approach 2: Aggregate Data Dynamically |
| **Latency** | Lower (faster lookups from a single DB) | Slightly higher (fetching from multiple DBs) |
| **Backward Compatibility** | Breaks if schema is changed without versioning | Preserves existing functionality, but API response must be modified |
| **Data Storage Limits** | May hit DynamoDB **400KB item size limit** | Distributes data across multiple DBs, avoiding size constraints |
| **Schema Changes** | Requires modifications in PED and DAO layer | No schema changes, but requires enhanced API logic |
| **API Request/Response Changes** | Changes needed for GPED API and dependent services | Changes required in GPED API response model |
| **Caching Requirement** | Less necessary, as data resides in a single source | More useful to optimize multiple DB queries |
| **Complexity** | Medium (schema refactoring + backfilling) | Higher (multiple DB joins, aggregation logic) |
| **Scalability** | Limited by PED storage constraints | More scalable due to distributed data retrieval |

**Key Considerations**

**1. API Versioning to Prevent Breaking Changes**

Since other services may depend on GPED API, **API versioning** ensures backward compatibility:

|  |  |  |  |
| --- | --- | --- | --- |
| Versioning Method | Example | Pros | Cons |
| **URL Path Versioning** | /v1/gped, /v2/gped | Clear separation of versions | Requires maintaining multiple endpoints |
| **Query Parameter Versioning** | /gped?version=2 | No URL structure change, easy implementation | Can clutter API logic |

**Recommendation:** Use **query parameter versioning** since it allows smooth migration **without modifying v1 requests**.

**2. Parallel Fetching and Aggregation (For Approach 2)**

If multiple databases are queried, API responses should be sent **only after aggregating all data**:

* Implement **asynchronous parallel fetching** (e.g., Java CompletableFuture).
* Use a **fallback strategy** if one DB is slow (e.g., return partial data with a retry mechanism).

**3. Backfilling for Approach 1**

If PED schema is modified, **historical data must be backfilled** to maintain consistency:

* Extract old records from existing PED schema.
* Transform them into the new format.
* Bulk update PED.

**Final Recommendation**

* **If minimizing latency and keeping a simple schema is a priority → Approach 1 (Expand PED Schema) is preferable.**
* **If scalability, avoiding storage limits, and maintaining backward compatibility are key → Approach 2 (Aggregate from Multiple DBs) is better.**