Talk

Technical Specification: Invoice-to-Quota Synchronizationn

This document outlines the logic for the automated Invoice-to-Quota synchronization task. The system ensures that every paid service is translated into a usable member entitlement while respecting complex “Package” (BOM) structures and recurring billing cycles.

1. System Objective

The goal is to automate the creation of member “quotas” (entry entitlements) based on successful invoice generation. For every invoice issued to a member for a specific service, the system must generate a corresponding quota record that defines what they can do, how many times, and for how long.

2. The Core Data Entities

The logic draws from five distinct functional areas of your database:

- Transactions (dok, dok_items): The source of truth for what was purchased and when.
- Membership & Services (sif_clan, sif_art): The master files for who the members are and what services/packages exist.
- Recurring Logic (osn_zad): The “engine” that defines the billing frequency and the duration of the service (Days vs. Months).
- Recipes/Packages (osn_sastavnice, osn_sastavke): The breakdown logic for “bundled” services (e.g., a “VIP Package” that consists of 10 Gym entries and 2 Saunas).
- Entitlements (gym_quota, gym_quotad): The destination tables that control the member’s actual access rights.

3. Procedural Logic & Filtering

To ensure data integrity and business accuracy, the extraction process follows a strict hierarchy of filters:

A. Authorization Gatekeepers

Data is only extracted if:

1. Company Level: The company (sif_pod) is active and has purchased the specific Module 50.
2. Department Level: The department (sif_oj) is active.
3. Config Level: The specific setting OJ_CLAN = 'F' exists in osn_setoj for that department.

B. The "Package Explosion" (BOM Logic)

The system distinguishes between standalone services and packages using the sif_art.ru flag:

- Simple Service (ru != 'Y'): The item is moved directly to the quota.
- Package Service (ru = 'Y'): The system “explodes” the item into its individual components via the osn_sastavke table. The final quota quantity is calculated as:
  - $\text{Total\_Qty} = \text{Invoice\_Qty} \times \text{Recipe\_Ingredient\_Qty}$

C. Temporal Logic (Date Calculations)

Instead of using a fixed duration, the system calculates the expiry date (edate) dynamically based on the subscription master (osn_zad):

- Daily Frequency (dm = 'D'): edate = sdate + frek
- Monthly Frequency (dm = 'M'): edate = sdate + (frek * months)
- Cycle Alignment: The sdate is anchored to the membership start date (poc), moving forward in cycles to align with the current invoice period.

4. Integration Strategy (Python Task)

The task is designed to be idempotent, meaning it can run multiple times without creating duplicate records.

1. Identification: A LEFT JOIN between dok and gym_quota identifies invoices that do not yet have an assigned quota.
2. Transformation: The SQL handles the complex joining of configuration settings and package recipes in a single pass.
3. Insertion: The Python script performs a two-stage transactional insert:
   - Stage 1: Insert into gym_quota (The Header).
   - Stage 2: Use the RETURNING id from Stage 1 to populate gym_quotad (The Details) with the exploded list of services.

5. Summary of Key Mappings

Logic	Source Table	Target Field
Whom	dok.clan	gym_quota.member
Why	dok.id	gym_quota.dok
When (Start)	osn_zad.poc (Cycle adjusted)	gym_quota.sdate
When (End)	osn_zad (dm/frek calculation)	gym_quota.edate
What (Items)	sif_art (exploded via osn_sastavke)	gym_quotad.art