

Talk

# Technical Specification: Invoice-to-Quota Synchronizati

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:
  - $Total\_Qty = Invoice\_Qty \times Recipe\_Ingredient\_Qty$

## 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.

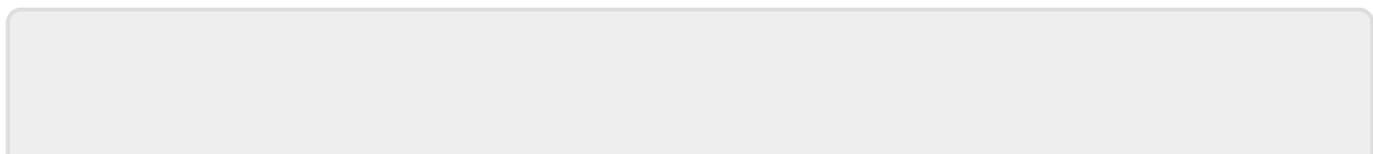
## 4. Integration Strategy (Python Task)

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	<code>dok.clan</code>	<code>gym_quota.member</code>
Why	<code>dok.id</code>	<code>gym_quota.dok</code>
When (Start)	<code>osn_zad.poc</code> (Cycle adjusted)	<code>gym_quota.sdate</code>
When (End)	<code>osn_zad</code> ( <code>dm/frek</code> calculation)	<code>gym_quota.edate</code>
What (Items)	<code>sif_art</code> (exploded via <code>osn_sastavke</code> )	<code>gym_quotad.art</code>



From:

<https://wiki.micro-process.hr/> -

Permanent link:

<https://wiki.micro-process.hr/brix/hr/members/pretplate/techspec>

Last update: **12/02/2026 21:01**

