Talk

Gym Membership Entitlement System: Comprehensive recap

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. Core Logic Flow

- Trigger: A Python scheduler runs daily.
- Scope: Processes only “active” entities where:
  - Company (sif_pod): Field `fl_akt` is true AND has access to Module 50 in `m_kupljeno`.
  - Department (sif_oj): Field `fl_akt` is true AND Setting `OJ_CLAN` is set to F in `osn_setoj`.
- Filtering: Only selects invoices (`dok`) that do not already have an entry in `gym_quota` (Idempotency).

2. Package Explosion (Sastavnice)

The system checks the article type (`sif_art.ru`):

- Simple Service: Stored directly in `gym_quotad`.
- Package (`ru = 'Y'`): The service is “exploded” into its components via `osn_sastavnice` and `osn_sastavke`.
- Quantity Logic:: The final quantity is calculated as `dok_items.kol * osn_sastavke.kol`

3. Temporal Logic (Date Calculation)

The validity window (`sdate` and `edate`) is derived from the subscription master table `osn_zad`.

SQL: Dynamic Date Calculation

This snippet demonstrates the logic for the PostgreSQL engine to handle the `dm` (Day/Month) and `frek` (Frequency) fields:

```
CASE /* DAILY LOGIC: Add integer days to the start date */ WHEN z.dm = 'D'
THEN (d.dat + (z.frek || ' days')::INTERVAL)::DATE

/* MONTHLY LOGIC: Add integer months to the start date */
WHEN z.dm = 'M' THEN
    (d.dat + (z.frek || ' months')::INTERVAL)::DATE
```

```
/* FALLBACK: Default to 1 month if undefined */
ELSE (d.dat + INTERVAL '1 month')::DATE

END AS calculated_edate
```

4. Table Mappings

Function	Source Field	Target Field
Member ID	`dok.clan`	`gym_quota.member`
Reference	`dok.id`	`gym_quota.dok`
Start Date	`osn_zad.poc` (Cycle Adjusted)	`gym_quota.sdate`
End Date	`osn_zad` (Calculated)	`gym_quota.edate`
Components	`sif_art` / `osn_sastavke`	`gym_quotad.art`
Quantity	`dok_items.kol` (Multiplied if Package)	`gym_quotad.qty`

5. Technical Implementation Notes

- Idempotency: Use a `NOT EXISTS` or `LEFT JOIN` check on `gym_quota.dok` to prevent duplicate processing.
- Transactions: The Python script must use a single DB transaction. If the `gym_quotad` insert fails, the `gym_quota` insert must roll back.
- Recursive Packages: Note that the current logic only supports 1 level of package depth (No packages inside packages).

Draft SQL query that handles that "Cycle Adjusted" sdate logic

To handle the “Moving Cycle” logic, we need to calculate an sdate that isn't just the invoice date, but rather the anniversary date of the membership that occurs most recently on or before the invoice date.

This requires a “Point-in-Time” calculation. We take the original start date (osn_zad.poc) and add the necessary number of intervals (Days or Months) to reach the current billing cycle.

The “Moving Cycle” Logic works as follows:

1. Calculate the number of intervals (N) that have passed between the original start (poc) and the invoice date (dat).
2. Multiply the interval (frek) by N.
3. Add that result to the original poc to get the current sdate.

The PostgreSQL Implementation

Here is the updated CTE (Common Table Expression) to replace the sdate logic in your Python script:

Python script that uses this logic to perform the batch insertion with the above specific SQL

```
...

# ("GymQuotaSync")

# Database connection settings
DB_CONFIG = {
    "user": "your_user",
    "password": "your_password",
    "database": "your_db",
    "host": "localhost"
}

async def run_sync_task():
    conn = await asyncpg.connect(**DB_CONFIG)

    # EXTRACTION QUERY WITH ANNIVERSARY CYCLE LOGIC
    extraction_sql = """
    WITH Active_Hierarchy AS (
        SELECT oj.id AS oj_id
        FROM sif_oj oj
        JOIN sif_pod pod ON oj.pod = pod.id
        JOIN osn_setoj sett ON oj.id = sett.oj
        JOIN m_kupljeno m ON pod.id = m.pod
        WHERE pod.fl_akt = true
          AND oj.fl_akt = true
          AND m.modul = 50
          AND sett.sett = 'OJ_CLAN' AND sett.val = 'F'
    ),
    Calculated_Cycles AS (
        SELECT
            d.id AS dok_id,
            d.clan AS member_id,
            di.item AS parent_art_id,
            di.kol AS invoice_qty,
            d.dat AS invoice_date,
            z.poc AS original_start,
            z.dm,
            z.frek,
            -- Calculate how many full cycles passed between original start
and invoice date
            CASE
                WHEN z.dm = 'D' THEN
                    FLOOR(EXTRACT(DAY FROM (d.dat - z.poc)) / NULLIF(z.frek,
```

```
0))
                WHEN z.dm = 'M' THEN
                    FLOOR((EXTRACT(YEAR FROM d.dat) - EXTRACT(YEAR FROM
z.poc)) * 12 +
                           (EXTRACT(MONTH FROM d.dat) - EXTRACT(MONTH FROM
z.poc))) / NULLIF(z.frek, 0)
                ELSE 0
            END AS intervals_passed
        FROM dok d
        JOIN dok_items di ON d.id = di.dok
        JOIN Active_Hierarchy ah ON d.oj = ah.oj_id
        JOIN osn_zad z ON d.clan = z.clan AND di.item = z.item
        LEFT JOIN gym_quota gq ON d.id = gq.dok
        WHERE gq.id IS NULL
    )
    SELECT
        cc.dok_id,
        cc.member_id,
        cc.parent_art_id,
        -- SDATE: Original start + (Cycles Passed * Frequency)
        CASE
            WHEN cc.dm = 'D' THEN (cc.original_start + (cc.intervals_passed
* cc.frek || ' days')::interval)::date
            WHEN cc.dm = 'M' THEN (cc.original_start + (cc.intervals_passed
* cc.frek || ' months')::interval)::date
            ELSE cc.invoice_date
        END AS sdate,
        -- EDATE: sdate + 1 full frequency interval
        CASE
            WHEN cc.dm = 'D' THEN (cc.original_start + ((cc.intervals_passed
+ 1) * cc.frek || ' days')::interval)::date
            WHEN cc.dm = 'M' THEN (cc.original_start + ((cc.intervals_passed
+ 1) * cc.frek || ' months')::interval)::date
            ELSE (cc.invoice_date + interval '1 month')::date
        END AS edate,
        -- Package Explosion Logic
        a.ru,
        COALESCE(sk.item, cc.parent_art_id) AS component_art,
        CASE WHEN a.ru = 'Y' THEN (cc.invoice_qty * sk.kol) ELSE
cc.invoice_qty END AS component_qty
    FROM Calculated_Cycles cc
    JOIN sif_art a ON cc.parent_art_id = a.id
    LEFT JOIN osn_sastavnice sn ON a.id = sn.art AND a.ru = 'Y'
    LEFT JOIN osn_sastavke sk ON sn.id = sk.sastavnica;
    """

    try:
        async with conn.transaction():
            rows = await conn.fetch(extraction_sql)

            if not rows:
```

```
                return "No new invoices found matching active criteria."

            processed_count = 0
            current_dok = None
            last_quota_id = None

            for row in rows:
                # Grouping: One gym_quota per Invoice Header
                if row['dok_id'] != current_dok:
                    last_quota_id = await conn.fetchval("""
                        INSERT INTO gym_quota (member, art, sdate, edate,
dok)
                        VALUES ($1, $2, $3, $4, $5)
                        RETURNING id;
                    """, row['member_id'], row['parent_art_id'],
row['sdate'], row['edate'], row['dok_id'])

                    current_dok = row['dok_id']
                    processed_count += 1

                # Grouping: Multiple gym_quotad per Header (Components)
                await conn.execute("""
                    INSERT INTO gym_quotad (quota, art, qty)
                    VALUES ($1, $2, $3);
                """, last_quota_id, row['component_art'],
row['component_qty'])

        return f"Successfully synchronized {processed_count} new quotas."

    except Exception as e:
        # Implicit rollback on error due to the 'async with' context
        print(f"Sync error: {e}")
        raise e
    finally:
        await conn.close()

@app.get("/sync-quotas")
async def handle_request(request):
    msg = await run_sync_task()
    return response.json({"status": "complete", "detail": msg})

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8000, access_log=True)
```