es. please provide a revised sql forecast function that guarantees no NULLs

**ChatGPT said:**

Absolutely — here's a **revised version** of your SQL function for Supabase that **guarantees no NULL values** during a 90-day cash forecast. It correctly initializes the forecast, fills in every required value (opening\_balance, income, expenses, closing\_balance), and ensures that each day's **closing balance becomes the next day's opening balance**.

**✅ Revised SQL Function: initialize\_cashflow\_data**

sql

Copy code

CREATE OR REPLACE FUNCTION public.initialize\_cashflow\_data(

user\_id UUID,

initial\_balance NUMERIC DEFAULT 0

)

RETURNS void

LANGUAGE plpgsql

AS $$

DECLARE

current\_date DATE := CURRENT\_DATE;

rolling\_balance NUMERIC := initial\_balance;

i INTEGER;

BEGIN

-- Optional: Clear any existing forecast from today onward

DELETE FROM public.daily\_cashflow

WHERE user\_id = initialize\_cashflow\_data.user\_id AND date >= current\_date;

-- Forecast for today + next 89 days

FOR i IN 0..89 LOOP

-- Compute each day's values

-- Income and expenses default to 0; can be updated later

-- Closing balance = opening + income - expenses = rolling\_balance (since income/expenses = 0)

INSERT INTO public.daily\_cashflow (

date,

user\_id,

opening\_balance,

income,

expenses,

closing\_balance

)

VALUES (

current\_date + i,

initialize\_cashflow\_data.user\_id,

rolling\_balance,

0,

0,

rolling\_balance -- same as opening for now

)

ON CONFLICT (date, user\_id) DO UPDATE

SET

opening\_balance = EXCLUDED.opening\_balance,

income = 0,

expenses = 0,

closing\_balance = EXCLUDED.opening\_balance;

-- Carry forward balance to next day

rolling\_balance := rolling\_balance; -- no change yet

END LOOP;

END;

$$;

**✅ Features of This Version**

| **Feature** | **Benefit** |
| --- | --- |
| DELETE existing forecast | Prevents duplicates or dirty forecasts |
| income and expenses set to 0 | Prevents null math errors |
| closing\_balance = opening\_balance | Placeholder until income/expenses updated |
| Uses ON CONFLICT (date, user\_id) | Safe to rerun forecast generation without duplication |
| Uses rolling\_balance consistently | Maintains clean carryover between days |

**🔁 When You Update a Day's Income/Expenses**

You’ll need a separate recalculation function, like:

sql

Copy code

CREATE OR REPLACE FUNCTION public.recalculate\_cashflow(user\_id UUID)

RETURNS void

LANGUAGE plpgsql

AS $$

DECLARE

prev\_balance NUMERIC;

rec RECORD;

BEGIN

-- Step 1: Get the opening balance of day 1

SELECT opening\_balance INTO prev\_balance

FROM public.daily\_cashflow

WHERE user\_id = recalculate\_cashflow.user\_id

AND date = CURRENT\_DATE

LIMIT 1;

-- Step 2: Iterate and recalculate all 90 days

FOR rec IN

SELECT \* FROM public.daily\_cashflow

WHERE user\_id = recalculate\_cashflow.user\_id

AND date >= CURRENT\_DATE

ORDER BY date

LOOP

-- Update row with new balance

UPDATE public.daily\_cashflow

SET

opening\_balance = prev\_balance,

closing\_balance = prev\_balance + COALESCE(rec.income, 0) - COALESCE(rec.expenses, 0)

WHERE id = rec.id;

-- Update carry-forward

prev\_balance := prev\_balance + COALESCE(rec.income, 0) - COALESCE(rec.expenses, 0);

END LOOP;

END;

$$;

**📦 Next Steps**

1. **Deploy the function** using Supabase SQL Editor
2. **Call initialize\_cashflow\_data(user\_id, starting\_balance)** from your app
3. Use a **button or trigger** to call recalculate\_cashflow(user\_id) after a user updates income/expense values

Would you like the accompanying **Supabase table schema for daily\_cashflow** and a **Lovable screen definition** to trigger recalculate\_cashflow after edits?