🔧 Supabase RLS Policy Migration Guide

⚠️ PROBLEM

Your code has the fixed RLS policies, but your **live Supabase database still has the old problematic policies** that cause infinite recursion errors. Code changes alone don't update the live database - you need to run the migration script on your actual Supabase instance.

✅ SOLUTION

Apply the corrected RLS policies from your migration file to your live Supabase database.

🚀 STEP-BY-STEP INSTRUCTIONS

Step 1: Access Your Supabase Dashboard

1. Go to https://app.supabase.com (https://app.supabase.com)
2. Sign in to your account
3. Select your project from the dashboard

Step 2: Open SQL Editor

1. In the left sidebar, click on **"SQL Editor"**
2. Click **"New Query"** to create a fresh SQL script

Step 3: Drop Old Problematic Policies

First, we need to remove the existing policies that cause infinite recursion. Copy and paste this SQL block:

```sql
-- Drop existing problematic policies
DROP POLICY IF EXISTS "Users can view their tenants" ON public.tenants;
DROP POLICY IF EXISTS "Users manage own tenant_user records" ON public.tenant_users;
DROP POLICY IF EXISTS "Users can view same tenant users" ON public.tenant_users;
DROP POLICY IF EXISTS "Users manage own chat sessions" ON public.chat_sessions;
DROP POLICY IF EXISTS "Users can view tenant knowledge base" ON public.knowledge_base;
DROP POLICY IF EXISTS "Service role manages knowledge base writes" ON public.know-
ledge_base;
DROP POLICY IF EXISTS "Service role manages knowledge base updates" ON public.know-
ledge_base;
DROP POLICY IF EXISTS "Service role manages knowledge base deletes" ON public.know-
ledge_base;

-- Also drop any service role policies to recreate them properly
DROP POLICY IF EXISTS "Service role full access tenants" ON public.tenants;
DROP POLICY IF EXISTS "Service role full access tenant_users" ON public.tenant_users;
DROP POLICY IF EXISTS "Service role full access chat_sessions" ON public.chat_sessions;
DROP POLICY IF EXISTS "Service role full access knowledge_base" ON public.know-
ledge_base;
```

⚡ **Run this block first** (click the "Run" button or use Ctrl+Enter)

Step 4: Create Fixed Policies

Now create the corrected policies that avoid circular references. Copy and paste this SQL block:

```sql
-- ========================================================================
-- FIXED RLS POLICIES - NON-CIRCULAR WITH PROPER TENANT ISOLATION
-- ========================================================================

-- Service role policies (complete access for service operations)
CREATE POLICY "Service role full access tenants" ON public.tenants
  FOR ALL USING (auth.role() = 'service_role');

CREATE POLICY "Service role full access tenant_users" ON public.tenant_users
  FOR ALL USING (auth.role() = 'service_role');

CREATE POLICY "Service role full access chat_sessions" ON public.chat_sessions
  FOR ALL USING (auth.role() = 'service_role');

CREATE POLICY "Service role full access knowledge_base" ON public.knowledge_base
  FOR ALL USING (auth.role() = 'service_role');

-- ========================================================================
-- USER POLICIES - DIRECT AUTH CHECKS (NO CIRCULAR REFERENCES)
-- ========================================================================

-- TENANTS: Users can only view tenants they belong to
-- Using a simple subquery that doesn't reference tenant_users in the policy check
CREATE POLICY "Users can view their tenants" ON public.tenants
  FOR SELECT USING (
    auth.role() = 'authenticated' AND (
      -- Allow if there's a direct tenant_user record (no self-reference in policy)
      id IN (
        SELECT tenant_id FROM public.tenant_users
        WHERE user_id = auth.uid()
      )
    )
  );

-- TENANT_USERS: Critical - avoid circular reference
-- Users can only manage their own tenant_user records (direct user_id check)
CREATE POLICY "Users manage own tenant_user records" ON public.tenant_users
  FOR ALL USING (auth.uid() = user_id);

-- Additional policy for tenant_users: Allow users to view other users in their tenant
-- This uses a safe approach - checking if the viewing user exists in the same tenant
CREATE POLICY "Users can view same tenant users" ON public.tenant_users
  FOR SELECT USING (
    auth.role() = 'authenticated' AND
    -- Check if the requesting user belongs to the same tenant
    -- This is safe because we're not checking roles within the policy
    EXISTS (
      SELECT 1 FROM public.tenant_users tu2
      WHERE tu2.user_id = auth.uid()
      AND tu2.tenant_id = tenant_users.tenant_id
    )
  );

-- CHAT_SESSIONS: Enhanced to respect tenant boundaries
CREATE POLICY "Users manage own chat sessions" ON public.chat_sessions
  FOR ALL USING (
    auth.uid() = user_id AND
    -- Ensure user belongs to the tenant (safe tenant check)
    EXISTS (
      SELECT 1 FROM public.tenant_users tu
      WHERE tu.user_id = auth.uid()
      AND tu.tenant_id = chat_sessions.tenant_id
```

```
    )
  );

-- KNOWLEDGE_BASE: Tenant-aware access
-- Users can view knowledge base for their tenants + global entries
CREATE POLICY "Users can view tenant knowledge base" ON public.knowledge_base
  FOR SELECT USING (
    auth.role() = 'authenticated' AND (
      -- Global knowledge base (tenant_id is NULL)
      tenant_id IS NULL
      OR
      -- Knowledge base for user's tenant
      EXISTS (
        SELECT 1 FROM public.tenant_users tu
        WHERE tu.user_id = auth.uid()
        AND tu.tenant_id = knowledge_base.tenant_id
      )
    )
  );

-- Knowledge base modification: Only service role can modify
-- This eliminates the need for complex tenant admin checks that could cause recursion
CREATE POLICY "Service role manages knowledge base writes" ON public.knowledge_base
  FOR INSERT WITH CHECK (auth.role() = 'service_role');

CREATE POLICY "Service role manages knowledge base updates" ON public.knowledge_base
  FOR UPDATE USING (auth.role() = 'service_role');

CREATE POLICY "Service role manages knowledge base deletes" ON public.knowledge_base
  FOR DELETE USING (auth.role() = 'service_role');
```

⚡ **Run this block second** (click the “Run” button or use Ctrl+Enter)

✅ VERIFICATION STEPS

Step 5: Verify Policies Were Applied

In the Supabase dashboard:

1. Go to **“Authentication” → “Policies”** in the left sidebar
2. Check that you see the new policies for each table:
   - `tenants`: “Service role full access tenants”, “Users can view their tenants”
   - `tenant_users`: “Service role full access tenant_users”, “Users manage own tenant_user records”, “Users can view same tenant users”
   - `chat_sessions`: “Service role full access chat_sessions”, “Users manage own chat sessions”
   - `knowledge_base`: Multiple policies including service role access and user view policies

Step 6: Test Database Access

Run this test query in the SQL editor to verify no more recursion:

```
-- Test query that should NOT cause infinite recursion
SELECT t.name, t.subdomain, tu.role
FROM tenants t
JOIN tenant_users tu ON t.id = tu.tenant_id
LIMIT 5;
```

This should run without errors and return results.

Step 7: Test Your Application

1. Open your chatbot application
2. Check the browser console (F12 → Console tab)
3. **The infinite recursion errors should be gone!**
4. Try logging in and using the chat functionality
5. Verify that tenant data loads properly

🎯 WHAT THESE FIXES SOLVED

Before (Problematic):

- ❌ Policies had circular references between `tenants` and `tenant_users` tables
- ❌ When checking tenant access, it would check tenant_users, which would check tenants, creating infinite loops
- ❌ Console showed "infinite recursion detected in policy for relation 'tenant_users'"

After (Fixed):

- ✅ **Direct user_id checks**: Policies now use `auth.uid() = user_id` for immediate verification
- ✅ **Safe tenant checks**: Use simple EXISTS subqueries that don't create circular references
- ✅ **Service role bypass**: Service role gets full access without complex checks
- ✅ **Proper tenant isolation**: Users only see data from their own tenants

---

🆘 TROUBLESHOOTING

If you still see errors:

1. **Clear browser cache**: Hard refresh (Ctrl+F5) to clear any cached API responses
2. **Check policy names**: In Supabase Dashboard → Authentication → Policies, make sure old policies are gone and new ones exist
3. **Verify service role**: Your application should be using the service role key for backend operations

If queries still fail:

- Make sure RLS is enabled on all tables (should already be set)
- Check that your application is using the correct Supabase keys (anon key for frontend, service key for backend)

🎉 SUCCESS INDICATORS

You'll know it worked when:
- ✅ No more "infinite recursion detected" errors in console
- ✅ Chatbot application loads without 500 errors
- ✅ Users can log in and access their tenant data

- ✅ Chat sessions load properly
- ✅ Knowledge base queries work without errors

Total estimated time: 5-10 minutes

After completing these steps, your Supabase database will have the corrected RLS policies that eliminate the infinite recursion problem while maintaining proper security and tenant isolation.