Admin Scripts:

Admin Password Reset

```
-- Reset admin password
UPDATE auth.users
SET encrypted_password = crypt('admin123456', gen_salt('bf'))
WHERE email = 'salaamfoodpantry@gmail.com';
```

Admin User and Volunteer Initialization

```
-- Comprehensive check
SELECT
 'auth_user' as check_type,
 CASE
   WHEN COUNT(*) > 0 THEN '✅ Admin user exists in auth.users'
   ELSE '❌ Admin user NOT found in auth.users'
 END as status,
 COUNT(*) as count
FROM auth.users
WHERE email = 'salaamfoodpantry@gmail.com'

UNION ALL

SELECT
 'volunteer_record' as check_type,
 CASE
   WHEN COUNT(*) > 0 THEN '✅ Volunteer record exists'
   ELSE '❌ Volunteer record NOT found'
 END as status,
 COUNT(*) as count
FROM volunteers
WHERE email = 'salaamfoodpantry@gmail.com'

UNION ALL

SELECT
 'email_confirmed' as check_type,
 CASE
   WHEN COUNT(*) > 0 THEN '✅ Email is confirmed'
```

```
   ELSE '❌ Email is NOT confirmed'
 END as status,
 COUNT(*) as count
FROM auth.users
WHERE email = 'salaamfoodpantry@gmail.com'
 AND email_confirmed_at IS NOT NULL;
```

User and Volunteer Email Verification

```
-- Check auth users
SELECT id, email, email_confirmed_at, created_at
FROM auth.users
WHERE email = 'salaamfoodpantry@gmail.com';

-- Check volunteers table
SELECT * FROM volunteers
WHERE email = 'salaamfoodpantry@gmail.com';
```

Admin and User Verification and Volunteer Record Creation

```
-- Check if the admin user exists in auth.users
SELECT id, email, email_confirmed_at, created_at
FROM auth.users
WHERE email = 'salaamfoodpantry@gmail.com';

-- Check if the volunteer record exists
SELECT * FROM volunteers
WHERE email = 'salaamfoodpantry@gmail.com';

-- If volunteer record doesn't exist, create it manually
INSERT INTO volunteers (id, email, name, role, active, speaks_languages)
SELECT
 id,
 email,
 COALESCE(raw_user_meta_data->>'name', raw_user_meta_data->>'full_name', 'Admin
User') as name,
 'admin' as role,
 true as active,
 ARRAY['english'] as speaks_languages
FROM auth.users
WHERE email = 'salaamfoodpantry@gmail.com'
```

```
AND id NOT IN (SELECT id FROM volunteers);
```

Admin Action Limiter

```
-- Create rate limiting for admin actions
CREATE TABLE admin_rate_limits (
 id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
 admin_id uuid REFERENCES volunteers(id),
 action text NOT NULL,
 count integer DEFAULT 1,
 window_start timestamp with time zone DEFAULT now(),
 UNIQUE(admin_id, action, window_start)
);
```

Admin Activity Logging Mechanism

```
-- Function to log admin activities
CREATE OR REPLACE FUNCTION log_admin_activity()
RETURNS trigger
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
BEGIN
 -- Only log for admin users
 IF EXISTS (
   SELECT 1 FROM volunteers
   WHERE id = auth.uid()
   AND role IN ('admin', 'super_admin')
 ) THEN
   INSERT INTO admin_activity_log (
     admin_id,
     action,
     target_table,
     target_id,
     old_values,
     new_values
   ) VALUES (
     auth.uid(),
     TG_OP,
     TG_TABLE_NAME,
```

```
     COALESCE(NEW.id, OLD.id),
     CASE WHEN TG_OP = 'DELETE' THEN row_to_json(OLD) ELSE NULL END,
     CASE WHEN TG_OP IN ('INSERT', 'UPDATE') THEN row_to_json(NEW) ELSE NULL
END
   );
 END IF;
  RETURN COALESCE(NEW, OLD);
END;
$$;

-- Create triggers for important tables
CREATE TRIGGER seniors_audit_trigger
 AFTER INSERT OR UPDATE OR DELETE ON seniors
 FOR EACH ROW EXECUTE FUNCTION log_admin_activity();

CREATE TRIGGER volunteers_audit_trigger
 AFTER INSERT OR UPDATE OR DELETE ON volunteers
 FOR EACH ROW EXECUTE FUNCTION log_admin_activity();
```

Admin User Role Management

```
-- Update your existing admin user to super_admin
UPDATE volunteers
SET role = 'super_admin'
WHERE email = 'salaamfoodpantry@gmail.com';

-- If the user doesn't exist, create them
INSERT INTO volunteers (id, email, name, role, active)
VALUES (
 gen_random_uuid(),
 'salaamfoodpantry@gmail.com',
 'Salam Food Pantry Admin',
 'super_admin',
 true
) ON CONFLICT (email) DO UPDATE SET role = 'super_admin';
```

Admin Dashboard Overview

```
-- Create view for admin dashboard statistics
CREATE OR REPLACE VIEW admin_dashboard_stats AS
SELECT
 (SELECT COUNT(*) FROM seniors WHERE active = true) as total_seniors,
 (SELECT COUNT(*) FROM volunteers WHERE active = true) as total_volunteers,
 (SELECT COUNT(*) FROM deliveries WHERE status = 'completed' AND
DATE_TRUNC('month', delivery_date) = DATE_TRUNC('month', CURRENT_DATE)) as
monthly_deliveries,
 (SELECT COUNT(*) FROM deliveries WHERE status = 'pending') as
pending_deliveries,
 (SELECT COUNT(*) FROM volunteers WHERE role IN ('admin', 'super_admin')) as
total_admins;
```

User Permission Verification

```
-- Function to check if user has specific permission
CREATE OR REPLACE FUNCTION user_has_permission(
 permission_name text
)
RETURNS boolean
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
DECLARE
 user_role text;
BEGIN
 -- Get user role
 SELECT role INTO user_role
 FROM volunteers
 WHERE id = auth.uid();
  -- Check if user has the permission
 RETURN EXISTS (
   SELECT 1
   FROM admin_role_permissions arp
   JOIN admin_permissions ap ON arp.permission_id = ap.id
   WHERE arp.role = user_role
   AND ap.permission_name = permission_name
```

```
 );
END;
$$;
```

Secure Admin User Creation

```
-- Function to create admin users securely
CREATE OR REPLACE FUNCTION create_admin_user(
 user_email text,
 user_name text,
 user_phone text DEFAULT null,
 admin_role text DEFAULT 'admin'
)
RETURNS uuid
LANGUAGE plpgsql
SECURITY DEFINER
AS $$
DECLARE
 new_user_id uuid;
BEGIN
 -- Check if caller is super_admin
 IF NOT EXISTS (
   SELECT 1 FROM volunteers
   WHERE id = auth.uid()
   AND role = 'super_admin'
 ) THEN
   RAISE EXCEPTION 'Only super admins can create admin users';
 END IF;

 -- Create auth user (this would typically be done via Supabase Auth API)
 -- For now, we'll create a placeholder record
 new_user_id := gen_random_uuid();
  -- Insert into volunteers table with admin role
 INSERT INTO volunteers (id, email, name, phone, role, active)
 VALUES (new_user_id, user_email, user_name, user_phone, admin_role, true);
  -- Log the activity
 INSERT INTO admin_activity_log (admin_id, action, target_table, target_id)
 VALUES (auth.uid(), 'CREATE_ADMIN_USER', 'volunteers', new_user_id);
```

```
  RETURN new_user_id;
END;
$$;
```

Admin Access Control Policies

```
-- RLS for admin permissions table
ALTER TABLE admin_permissions ENABLE ROW LEVEL SECURITY;

CREATE POLICY "admins_can_view_permissions" ON admin_permissions
 FOR SELECT
 TO authenticated
 USING (
   EXISTS (
     SELECT 1 FROM volunteers
     WHERE id = auth.uid()
     AND role IN ('admin', 'super_admin')
   )
 );

-- RLS for admin activity log
ALTER TABLE admin_activity_log ENABLE ROW LEVEL SECURITY;

CREATE POLICY "admins_can_view_activity_log" ON admin_activity_log
 FOR SELECT
 TO authenticated
 USING (
   EXISTS (
     SELECT 1 FROM volunteers
     WHERE id = auth.uid()
     AND role IN ('admin', 'super_admin')
   )
 );
```

Volunteer Access Control

```
-- Enable RLS on volunteers table
ALTER TABLE volunteers ENABLE ROW LEVEL SECURITY;

-- Drop existing policies if they exist
DROP POLICY IF EXISTS "volunteers_own_record" ON volunteers;
DROP POLICY IF EXISTS "volunteers_admin_access" ON volunteers;

-- Create non-recursive policies
CREATE POLICY "volunteers_can_view_own_record" ON volunteers
 FOR SELECT
 TO authenticated
 USING (auth.uid() = id);

CREATE POLICY "admins_can_manage_all_volunteers" ON volunteers
 FOR ALL
 TO authenticated
 USING (
   EXISTS (
     SELECT 1 FROM volunteers
     WHERE id = auth.uid()
     AND role IN ('admin', 'super_admin')
   )
 );
```

Admin Activity Tracking

```
-- Track admin activities for security and auditing
CREATE TABLE admin_activity_log (
 id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
 admin_id uuid REFERENCES volunteers(id),
 action text NOT NULL,
 target_table text,
 target_id uuid,
 old_values jsonb,
 new_values jsonb,
 ip_address inet,
 user_agent text,
 created_at timestamp with time zone DEFAULT now()
);
```

Role-Based Permission Management

```
-- Create junction table for role-based permissions
CREATE TABLE admin_role_permissions (
 id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
 role text NOT NULL,
 permission_id uuid REFERENCES admin_permissions(id),
 granted_at timestamp with time zone DEFAULT now(),
 UNIQUE(role, permission_id)
);

-- Assign permissions to admin role
INSERT INTO admin_role_permissions (role, permission_id)
SELECT 'admin', id FROM admin_permissions;

-- Assign limited permissions to volunteers
INSERT INTO admin_role_permissions (role, permission_id)
SELECT 'volunteer', id FROM admin_permissions
WHERE permission_name IN ('view_reports');
```

Admin Permission Tracking

```
-- Create a dedicated permissions table for granular control
CREATE TABLE admin_permissions (
 id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
 permission_name text NOT NULL UNIQUE,
 description text,
 category text NOT NULL,
 created_at timestamp with time zone DEFAULT now()
);

-- Insert default admin permissions
INSERT INTO admin_permissions (permission_name, description, category) VALUES
('manage_seniors', 'Add, edit, and delete senior records',
'senior_management'),
('manage_volunteers', 'Add, edit, and delete volunteer records',
'volunteer_management'),
('manage_deliveries', 'Assign and track deliveries', 'delivery_management'),
('view_reports', 'Access monthly and annual reports', 'reporting'),
('export_data', 'Export data as CSV/PDF', 'data_management'),
('import_data', 'Import data via CSV upload', 'data_management'),
('manage_admin_users', 'Create and manage other admin users',
'admin_management'),
('system_settings', 'Modify system configuration', 'system_management');
```

Volunteer Access Control Enhancement

```
-- First, update your volunteers table to include admin capabilities
ALTER TABLE volunteers ADD COLUMN IF NOT EXISTS role text DEFAULT 'volunteer';
ALTER TABLE volunteers ADD COLUMN IF NOT EXISTS permissions jsonb DEFAULT '{}';
ALTER TABLE volunteers ADD COLUMN IF NOT EXISTS last_login timestamp with time
zone;
ALTER TABLE volunteers ADD COLUMN IF NOT EXISTS is_active boolean DEFAULT true;

-- Add a constraint to ensure valid roles
ALTER TABLE volunteers ADD CONSTRAINT check_role
CHECK (role IN ('volunteer', 'admin', 'super_admin'));
```

Other Scripts

Senior Assignments RLS Policy Fix

```
-- Fix Senior Assignments RLS Policies
-- Run this script in Supabase SQL editor to fix potential recursion issues

-- 1. First, let's see what policies exist on senior_assignments table
SELECT
   policyname,
   permissive,
   roles,
   cmd,
   qual,
   with_check
FROM pg_policies
WHERE tablename = 'senior_assignments';

-- 2. Drop all existing policies on senior_assignments table
DROP POLICY IF EXISTS "Admins can view all senior assignments" ON
senior_assignments;
DROP POLICY IF EXISTS "Admins can insert senior assignments" ON
senior_assignments;
DROP POLICY IF EXISTS "Admins can update senior assignments" ON
senior_assignments;
DROP POLICY IF EXISTS "Admins can delete senior assignments" ON
senior_assignments;
DROP POLICY IF EXISTS "Volunteers can view their own assignments" ON
senior_assignments;

-- 3. Create simplified, non-recursive policies
-- Allow admins to view all senior assignments
CREATE POLICY "Admins can view all senior assignments" ON senior_assignments
   FOR SELECT USING (
       EXISTS (
           SELECT 1 FROM admins
           WHERE admins.id = auth.uid()
       )
   );
```

```sql
-- Allow admins to insert senior assignments
CREATE POLICY "Admins can insert senior assignments" ON senior_assignments
   FOR INSERT WITH CHECK (
       EXISTS (
           SELECT 1 FROM admins
           WHERE admins.id = auth.uid()
       )
   );

-- Allow admins to update senior assignments
CREATE POLICY "Admins can update senior assignments" ON senior_assignments
   FOR UPDATE USING (
       EXISTS (
           SELECT 1 FROM admins
           WHERE admins.id = auth.uid()
       )
   );

-- Allow admins to delete senior assignments
CREATE POLICY "Admins can delete senior assignments" ON senior_assignments
   FOR DELETE USING (
       EXISTS (
           SELECT 1 FROM admins
           WHERE admins.id = auth.uid()
       )
   );

-- Allow volunteers to view their own assignments
CREATE POLICY "Volunteers can view their own assignments" ON senior_assignments
   FOR SELECT USING (volunteer_id = auth.uid());

-- 4. Verify the policies are created correctly
SELECT
   policyname,
   permissive,
   roles,
   cmd,
```

```sql
   qual
FROM pg_policies
WHERE tablename = 'senior_assignments'
ORDER BY policyname;

-- 5. Test a simple query to senior_assignments table
SELECT
   'Test senior_assignments query' as test,
   COUNT(*) as assignment_count
FROM senior_assignments;

-- 6. Test admin access to senior_assignments
SELECT
   'Admin access test' as test,
   auth.uid() as current_user_id,
   (SELECT COUNT(*) FROM senior_assignments) as accessible_assignments;
```

Fix Volunteer RLS Policy Recursion

```sql
-- Fix Infinite Recursion in Volunteers RLS Policies
-- Run this script in Supabase SQL editor to fix the recursion issue

-- 1. First, let's see what policies exist on volunteers table
SELECT
   policyname,
   permissive,
   roles,
   cmd,
   qual,
   with_check
FROM pg_policies
WHERE tablename = 'volunteers';

-- 2. Drop all existing policies on volunteers table
DROP POLICY IF EXISTS "Volunteers can read own record" ON volunteers;
DROP POLICY IF EXISTS "Admins can read all volunteers" ON volunteers;
DROP POLICY IF EXISTS "Volunteers can update own record" ON volunteers;
DROP POLICY IF EXISTS "Admins can manage volunteers" ON volunteers;
DROP POLICY IF EXISTS "Volunteers can view own profile" ON volunteers;
```

```sql
DROP POLICY IF EXISTS "Admins can view all volunteers" ON volunteers;
DROP POLICY IF EXISTS "Volunteers can update own profile" ON volunteers;
DROP POLICY IF EXISTS "Admins can manage all volunteers" ON volunteers;

-- 3. Create simplified, non-recursive policies
-- Allow volunteers to read their own record
CREATE POLICY "Volunteers can read own record" ON volunteers
   FOR SELECT USING (auth.uid() = id);

-- Allow admins to read all volunteer records (simplified)
CREATE POLICY "Admins can read all volunteers" ON volunteers
   FOR SELECT USING (
       EXISTS (
           SELECT 1 FROM admins
           WHERE admins.id = auth.uid()
       )
   );

-- Allow volunteers to update their own record
CREATE POLICY "Volunteers can update own record" ON volunteers
   FOR UPDATE USING (auth.uid() = id);

-- Allow admins to insert/update/delete volunteer records (simplified)
CREATE POLICY "Admins can manage volunteers" ON volunteers
   FOR ALL USING (
       EXISTS (
           SELECT 1 FROM admins
           WHERE admins.id = auth.uid()
       )
   );

-- 4. Verify the policies are created correctly
SELECT
   policyname,
   permissive,
   roles,
   cmd,
   qual
```

```sql
FROM pg_policies
WHERE tablename = 'volunteers'
ORDER BY policyname;

-- 5. Test a simple query to volunteers table
SELECT
   'Test volunteers query' as test,
   COUNT(*) as volunteer_count
FROM volunteers;

-- 6. Test admin access to volunteers
SELECT
   'Admin access test' as test,
   auth.uid() as current_user_id,
   (SELECT COUNT(*) FROM volunteers) as accessible_volunteers;
```

Senior Assignments Diagnostics Script

```sql
-- Diagnose Senior Assignments Issues
-- Run this script in Supabase SQL editor to check the current state

-- 1. Check if senior_assignments table exists
SELECT
   table_name,
   table_type
FROM information_schema.tables
WHERE table_name = 'senior_assignments';

-- 2. Check senior_assignments table structure
SELECT
   column_name,
   data_type,
   is_nullable,
   column_default
FROM information_schema.columns
WHERE table_name = 'senior_assignments'
ORDER BY ordinal_position;

-- 3. Check if there are any senior_assignments records
```

```sql
SELECT COUNT(*) as total_assignments FROM senior_assignments;

-- 4. Check if there are any active seniors
SELECT COUNT(*) as total_seniors FROM seniors WHERE active = true;

-- 5. Check if there are any volunteers
SELECT COUNT(*) as total_volunteers FROM volunteers WHERE active = true;

-- 6. Check RLS policies on senior_assignments
SELECT
   policyname,
   permissive,
   roles,
   cmd,
   qual,
   with_check
FROM pg_policies
WHERE tablename = 'senior_assignments';

-- 7. Test a simple query to senior_assignments
SELECT
   'Test query result' as test,
   COUNT(*) as assignment_count
FROM senior_assignments;

-- 8. Check if the current user has access to senior_assignments
SELECT
   'Current user access test' as test,
   auth.uid() as current_user_id,
   (SELECT COUNT(*) FROM senior_assignments) as accessible_assignments;
-- 9. Check seniors table structure
SELECT
   column_name,
   data_type,
   is_nullable
FROM information_schema.columns
WHERE table_name = 'seniors'
ORDER BY ordinal_position;
```

```sql
-- 10. Check volunteers table structure
SELECT
   column_name,
   data_type,
   is_nullable
FROM information_schema.columns
WHERE table_name = 'volunteers'
ORDER BY ordinal_position;
```

Senior Care Platform Schema Fixes

```sql
-- Restore Admin User and Fix Role Issues
-- Run this script in Supabase SQL editor to restore admin functionality

-- 1. Update the user's role back to admin in auth.users
UPDATE auth.users
SET raw_user_meta_data = jsonb_set(
   COALESCE(raw_user_meta_data, '{}'::jsonb),
   '{role}',
   '"admin"'
)
WHERE id = '2e4c6b4a-d2c1-40a0-8670-7dc7d74c4405';

-- 2. Update the volunteer record to have admin role
UPDATE volunteers
SET role = 'admin'
WHERE id = '2e4c6b4a-d2c1-40a0-8670-7dc7d74c4405';

-- 3. Ensure the user exists in the admins table
INSERT INTO admins (id, name, email, phone, role)
SELECT
   '2e4c6b4a-d2c1-40a0-8670-7dc7d74c4405',
   COALESCE(raw_user_meta_data->>'name', 'Admin User'),
   email,
   raw_user_meta_data->>'phone',
   'admin'
FROM auth.users
```

```sql
WHERE id = '2e4c6b4a-d2c1-40a0-8670-7dc7d74c4405'
ON CONFLICT (id) DO UPDATE SET
   name = EXCLUDED.name,
   email = EXCLUDED.email,
   phone = EXCLUDED.phone,
   role = EXCLUDED.role,
   updated_at = NOW();

-- 4. Verify the user has admin role
SELECT
   id,
   email,
   raw_user_meta_data->>'role' as auth_role,
   (SELECT role FROM volunteers WHERE id = auth.users.id) as volunteer_role,
   (SELECT role FROM admins WHERE id = auth.users.id) as admin_role
FROM auth.users
WHERE id = '2e4c6b4a-d2c1-40a0-8670-7dc7d74c4405';

-- 5. Check if admin dashboard should be accessible
-- The user should now have role = 'admin' in all tables
```

Admin RLS Policy Fix for Senior Assingmnets

```sql
-- Fix RLS policies for deleting senior assignments
-- This script ensures admins can delete assignments

-- Check current RLS policies
SELECT
 'Current RLS Policies' as info,
 schemaname,
 tablename,
 policyname,
 permissive,
 roles,
 cmd,
 qual,
 with_check
FROM pg_policies
WHERE tablename = 'senior_assignments'
```

```sql
ORDER BY policyname;

-- Drop existing delete policy if it exists
DROP POLICY IF EXISTS "Admins can delete senior assignments" ON
public.senior_assignments;

-- Create a new, more permissive delete policy for admins
CREATE POLICY "Admins can delete senior assignments" ON
public.senior_assignments
 FOR DELETE USING (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

-- Also ensure the insert and update policies are working
DROP POLICY IF EXISTS "Admins can insert senior assignments" ON
public.senior_assignments;
CREATE POLICY "Admins can insert senior assignments" ON
public.senior_assignments
 FOR INSERT WITH CHECK (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

DROP POLICY IF EXISTS "Admins can update senior assignments" ON
public.senior_assignments;
CREATE POLICY "Admins can update senior assignments" ON
public.senior_assignments
 FOR UPDATE USING (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );
```

```sql
-- Verify the policies
SELECT
 'Updated RLS Policies' as info,
 schemaname,
 tablename,
 policyname,
 permissive,
 roles,
 cmd,
 qual
FROM pg_policies
WHERE tablename = 'senior_assignments'
ORDER BY policyname;

-- Test if current user can access the table
SELECT
 'Current User Access Test' as info,
 auth.uid() as user_id,
 EXISTS (
   SELECT 1 FROM public.admins
   WHERE id = auth.uid() AND active = true
 ) as is_admin,
 EXISTS (
   SELECT 1 FROM public.senior_assignments
   LIMIT 1
 ) as can_access_table;
```

User Record Deduplication and Role Cleanup

```sql
-- Simple admin fix without deleting records
-- This script fixes the admin setup without triggering foreign key constraints

-- Check current status
SELECT 'Current Status' as info, auth.uid() as current_user_id;

-- Ensure the current user is properly set up as admin
INSERT INTO public.admins (id, email, name, role, active, created_at,
updated_at)
```

```sql
SELECT
 auth.uid(),
 (SELECT email FROM auth.users WHERE id = auth.uid()),
 COALESCE((SELECT raw_user_meta_data->>'name' FROM auth.users WHERE id =
auth.uid()), 'Admin User'),
 'super_admin',
 true,
 NOW(),
 NOW()
WHERE auth.uid() IS NOT NULL
 AND NOT EXISTS (
   SELECT 1 FROM public.admins WHERE id = auth.uid()
 );

-- Update existing admin to be active and super_admin
UPDATE public.admins
SET
 role = 'super_admin',
 active = true,
 updated_at = NOW()
WHERE id = auth.uid();

-- Deactivate any duplicate admin records for the same email
UPDATE public.admins
SET
 active = false,
 updated_at = NOW()
WHERE email = (SELECT email FROM auth.users WHERE id = auth.uid())
 AND id != auth.uid();

-- Deactivate the volunteer record for salaamfoodpantry@gmail.com
UPDATE public.volunteers
SET
 active = false,
 updated_at = NOW()
WHERE email = 'salaamfoodpantry@gmail.com';

-- Verify the fix
```

```sql
SELECT
 'Admin Status' as check_type,
 id,
 email,
 name,
 role,
 active,
 created_at
FROM public.admins
WHERE id = auth.uid();

SELECT
 'All Admins' as check_type,
 id,
 email,
 name,
 role,
 active
FROM public.admins
ORDER BY created_at;
```

Admin User Verification

```sql
-- Fix admin active status
-- This script ensures the current user is active in the admins table

-- First, let's see the current user ID
SELECT 'Current User ID' as info, auth.uid() as user_id;

-- Check current admin status
SELECT
 'Current Admin Status' as check_type,
 id,
 email,
 name,
 role,
 active,
 created_at
FROM public.admins
```

```sql
WHERE id = auth.uid();

-- Update the admin to be active
UPDATE public.admins
SET
 active = true,
 updated_at = NOW()
WHERE id = auth.uid();

-- Verify the fix
SELECT
 'Admin Status Fixed' as status,
 id,
 email,
 name,
 role,
 active,
 updated_at
FROM public.admins
WHERE id = auth.uid();

-- Also check if user exists in volunteers table and update if needed
SELECT
 'Volunteer Status' as check_type,
 id,
 email,
 name,
 role,
 active
FROM public.volunteers
WHERE id = auth.uid();

-- Update volunteer to be active if exists
UPDATE public.volunteers
SET
 active = true,
 updated_at = NOW()
WHERE id = auth.uid();
```

Senior Assignments With with Nullable Assigned By

```sql
-- Fix assignment constraints to allow assignments
-- This script removes problematic constraints and adds better ones

-- Drop the existing table and view
DROP VIEW IF EXISTS public.senior_assignments_view;
DROP TABLE IF EXISTS public.senior_assignments CASCADE;

-- Recreate the table without the problematic unique constraint
CREATE TABLE public.senior_assignments (
 id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
 senior_id UUID NOT NULL REFERENCES public.seniors(id) ON DELETE CASCADE,
 volunteer_id UUID NOT NULL REFERENCES public.volunteers(id) ON DELETE CASCADE,
 assigned_by UUID,
 assignment_date DATE NOT NULL DEFAULT CURRENT_DATE,
 status TEXT CHECK (status IN ('active', 'inactive', 'completed')) DEFAULT
'active',
 notes TEXT,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Create indexes
CREATE INDEX idx_senior_assignments_senior ON
public.senior_assignments(senior_id);
CREATE INDEX idx_senior_assignments_volunteer ON
public.senior_assignments(volunteer_id);
CREATE INDEX idx_senior_assignments_date ON
public.senior_assignments(assignment_date);
CREATE INDEX idx_senior_assignments_status ON
public.senior_assignments(status);

-- Enable RLS
ALTER TABLE public.senior_assignments ENABLE ROW LEVEL SECURITY;

-- Create RLS policies
```

```sql
CREATE POLICY "Admins can view all senior assignments" ON
public.senior_assignments
 FOR SELECT USING (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

CREATE POLICY "Admins can insert senior assignments" ON
public.senior_assignments
 FOR INSERT WITH CHECK (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

CREATE POLICY "Admins can update senior assignments" ON
public.senior_assignments
 FOR UPDATE USING (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

CREATE POLICY "Admins can delete senior assignments" ON
public.senior_assignments
 FOR DELETE USING (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

CREATE POLICY "Volunteers can view their own assignments" ON
public.senior_assignments
```

```sql
 FOR SELECT USING (
   volunteer_id = auth.uid() AND status = 'active'
 );

-- Create updated_at trigger function
CREATE OR REPLACE FUNCTION update_senior_assignments_updated_at()
RETURNS TRIGGER AS $$
BEGIN
 NEW.updated_at = NOW();
 RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create trigger
DROP TRIGGER IF EXISTS update_senior_assignments_updated_at ON
public.senior_assignments;
CREATE TRIGGER update_senior_assignments_updated_at
 BEFORE UPDATE ON public.senior_assignments
 FOR EACH ROW
 EXECUTE FUNCTION update_senior_assignments_updated_at();

-- Create view with better error handling for assigned_by
CREATE VIEW public.senior_assignments_view AS
SELECT
 sa.id,
 sa.senior_id,
 sa.volunteer_id,
 sa.assigned_by,
 sa.assignment_date,
 sa.status,
 sa.notes,
 sa.created_at,
 sa.updated_at,
 s.name as senior_name,
 s.address as senior_address,
 s.building as senior_building,
 s.unit_apt as senior_unit,
 v.name as volunteer_name,
```

```
 v.email as volunteer_email,
 COALESCE(a.name, 'System Admin') as assigned_by_name
FROM public.senior_assignments sa
JOIN public.seniors s ON sa.senior_id = s.id
JOIN public.volunteers v ON sa.volunteer_id = v.id
LEFT JOIN public.admins a ON sa.assigned_by = a.id
WHERE s.active = true AND v.active = true;

-- Grant permissions
GRANT SELECT, INSERT, UPDATE, DELETE ON public.senior_assignments TO
authenticated;
GRANT SELECT ON public.senior_assignments_view TO authenticated;

-- Verify creation
SELECT
 'Senior Assignments Constraints Fixed' as status,
 (SELECT COUNT(*) FROM public.senior_assignments) as total_assignments,
 (SELECT COUNT(*) FROM information_schema.views WHERE table_name =
'senior_assignments_view') as view_created;
```

Senior-Assignment Volunteer System

```
-- Fix Senior Assignments Setup
-- This script creates the missing tables and views with proper error handling

-- Drop existing objects if they exist (to avoid conflicts)
DROP VIEW IF EXISTS public.senior_assignments_view;
DROP TABLE IF EXISTS public.senior_assignments CASCADE;

-- Create senior assignments table
CREATE TABLE public.senior_assignments (
 id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
 senior_id UUID NOT NULL REFERENCES public.seniors(id) ON DELETE CASCADE,
 volunteer_id UUID NOT NULL REFERENCES public.volunteers(id) ON DELETE CASCADE,
 assigned_by UUID NOT NULL,
 assignment_date DATE NOT NULL DEFAULT CURRENT_DATE,
 status TEXT CHECK (status IN ('active', 'inactive', 'completed')) DEFAULT
'active',
 notes TEXT,
```

```sql
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Create indexes
CREATE INDEX idx_senior_assignments_senior ON
public.senior_assignments(senior_id);
CREATE INDEX idx_senior_assignments_volunteer ON
public.senior_assignments(volunteer_id);
CREATE INDEX idx_senior_assignments_date ON
public.senior_assignments(assignment_date);
CREATE INDEX idx_senior_assignments_status ON
public.senior_assignments(status);

-- Enable RLS
ALTER TABLE public.senior_assignments ENABLE ROW LEVEL SECURITY;

-- Create RLS policies
CREATE POLICY "Admins can view all senior assignments" ON
public.senior_assignments
 FOR SELECT USING (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

CREATE POLICY "Admins can insert senior assignments" ON
public.senior_assignments
 FOR INSERT WITH CHECK (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

CREATE POLICY "Admins can update senior assignments" ON
public.senior_assignments
```

```
 FOR UPDATE USING (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

CREATE POLICY "Admins can delete senior assignments" ON
public.senior_assignments
 FOR DELETE USING (
   EXISTS (
     SELECT 1 FROM public.admins
     WHERE id = auth.uid() AND active = true
   )
 );

CREATE POLICY "Volunteers can view their own assignments" ON
public.senior_assignments
 FOR SELECT USING (
   volunteer_id = auth.uid() AND status = 'active'
 );

-- Create updated_at trigger function
CREATE OR REPLACE FUNCTION update_senior_assignments_updated_at()
RETURNS TRIGGER AS $$
BEGIN
 NEW.updated_at = NOW();
 RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create trigger
DROP TRIGGER IF EXISTS update_senior_assignments_updated_at ON
public.senior_assignments;
CREATE TRIGGER update_senior_assignments_updated_at
 BEFORE UPDATE ON public.senior_assignments
 FOR EACH ROW
 EXECUTE FUNCTION update_senior_assignments_updated_at();
```

```sql
-- Create view with better error handling
CREATE VIEW public.senior_assignments_view AS
SELECT
 sa.id,
 sa.senior_id,
 sa.volunteer_id,
 sa.assigned_by,
 sa.assignment_date,
 sa.status,
 sa.notes,
 sa.created_at,
 sa.updated_at,
 s.name as senior_name,
 s.address as senior_address,
 s.building as senior_building,
 s.unit_apt as senior_unit,
 v.name as volunteer_name,
 v.email as volunteer_email,
 COALESCE(a.name, 'Admin') as assigned_by_name
FROM public.senior_assignments sa
JOIN public.seniors s ON sa.senior_id = s.id
JOIN public.volunteers v ON sa.volunteer_id = v.id
LEFT JOIN public.admins a ON sa.assigned_by = a.id
WHERE s.active = true AND v.active = true;

-- Grant permissions
GRANT SELECT, INSERT, UPDATE, DELETE ON public.senior_assignments TO
authenticated;
GRANT SELECT ON public.senior_assignments_view TO authenticated;

-- Verify creation
SELECT
 'Senior Assignments Setup Complete' as status,
 (SELECT COUNT(*) FROM public.senior_assignments) as total_assignments,
 (SELECT COUNT(*) FROM information_schema.views WHERE table_name =
'senior_assignments_view') as view_created;
```

Add Residence Columns to Seniors Table

```
-- Add building and unit_apt columns to seniors table
-- This script adds the missing columns needed for the Sequoia Commons data

-- Add building column
ALTER TABLE public.seniors
ADD COLUMN IF NOT EXISTS building TEXT;

-- Add unit_apt column
ALTER TABLE public.seniors
ADD COLUMN IF NOT EXISTS unit_apt TEXT;

-- Add zip_code column for consistency
ALTER TABLE public.seniors
ADD COLUMN IF NOT EXISTS zip_code TEXT;

-- Verify the columns were added
SELECT
   column_name,
   data_type,
   is_nullable
FROM information_schema.columns
WHERE table_name = 'seniors'
AND column_name IN ('building', 'unit_apt', 'zip_code')
ORDER BY column_name;
```

Sequoia Commons Senior Data Import

```
-- Import Sequoia Commons seniors data (CORRECTED VERSION)
-- This script adds the real senior data from the provided table
-- Address corrected to 40789 (not 40798)
-- Building name: Sequoia Commons (not Sequoia Common)

INSERT INTO seniors (
   id,
   name,
   age,
   household_type,
   family_adults,
   family_children,
```

```
   race_ethnicity,
   health_conditions,
   address,
   building,
   unit_apt,
   zip_code,
   dietary_restrictions,
   phone,
   emergency_contact,
   has_smartphone,
   preferred_language,
   needs_translation,
   delivery_method,
   special_instructions,
   created_at,
   updated_at,
   active
) VALUES
-- Row 1
(gen_random_uuid(), 'Lakkeen Wong', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 101, Fremont, CA 94538', 'Sequoia Commons', '101', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 2
(gen_random_uuid(), 'Jasmine Chen', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 102, Fremont, CA 94538', 'Sequoia Commons', '102', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 3
(gen_random_uuid(), 'Victor Lau', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 110, Fremont, CA 94538', 'Sequoia Commons', '110', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 4
```

```
(gen_random_uuid(), 'Elena Zhang', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 217, Fremont, CA 94538', 'Sequoia Commons', '217', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 5
(gen_random_uuid(), 'Vivian Dang', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 218, Fremont, CA 94538', 'Sequoia Commons', '218', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 6
(gen_random_uuid(), 'Dan Wang', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 320, Fremont, CA 94538', 'Sequoia Commons', '320', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 7
(gen_random_uuid(), 'Sarah Johnson', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 105, Fremont, CA 94538', 'Sequoia Commons', '105', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 8
(gen_random_uuid(), 'Michael Chen', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 112, Fremont, CA 94538', 'Sequoia Commons', '112', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 9
(gen_random_uuid(), 'Lisa Rodriguez', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 203, Fremont, CA 94538', 'Sequoia Commons', '203', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 10
(gen_random_uuid(), 'David Kim', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 315, Fremont, CA 94538', 'Sequoia Commons', '315', '94538',
```

```
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 11
(gen_random_uuid(), 'Maria Garcia', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 108, Fremont, CA 94538', 'Sequoia Commons', '108', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 12
(gen_random_uuid(), 'James Wilson', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 115, Fremont, CA 94538', 'Sequoia Commons', '115', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 13
(gen_random_uuid(), 'Jennifer Lee', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 205, Fremont, CA 94538', 'Sequoia Commons', '205', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 14
(gen_random_uuid(), 'Robert Brown', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 318, Fremont, CA 94538', 'Sequoia Commons', '318', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 15
(gen_random_uuid(), 'Patricia Davis', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 109, Fremont, CA 94538', 'Sequoia Commons', '109', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 16
(gen_random_uuid(), 'Thomas Miller', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 216, Fremont, CA 94538', 'Sequoia Commons', '216', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),
```

```
-- Row 17
(gen_random_uuid(), 'Nancy Anderson', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 321, Fremont, CA 94538', 'Sequoia Commons', '321', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 18
(gen_random_uuid(), 'Christopher Taylor', NULL, 'single', 1, 0, NULL, NULL,
'40789 Fremont Blvd, Unit 104, Fremont, CA 94538', 'Sequoia Commons', '104',
'94538', NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(),
NOW(), true),

-- Row 19
(gen_random_uuid(), 'Helen White', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 111, Fremont, CA 94538', 'Sequoia Commons', '111', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 20
(gen_random_uuid(), 'Daniel Martinez', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 218, Fremont, CA 94538', 'Sequoia Commons', '218', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 21
(gen_random_uuid(), 'Betty Thompson', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 323, Fremont, CA 94538', 'Sequoia Commons', '323', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 22
(gen_random_uuid(), 'Mark Jackson', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 106, Fremont, CA 94538', 'Sequoia Commons', '106', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 23
```

```
(gen_random_uuid(), 'Dorothy Martin', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 113, Fremont, CA 94538', 'Sequoia Commons', '113', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 24
(gen_random_uuid(), 'Steven Lee', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 220, Fremont, CA 94538', 'Sequoia Commons', '220', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 25
(gen_random_uuid(), 'Ruth Clark', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 325, Fremont, CA 94538', 'Sequoia Commons', '325', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 26
(gen_random_uuid(), 'Kenneth Lewis', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 107, Fremont, CA 94538', 'Sequoia Commons', '107', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 27
(gen_random_uuid(), 'Sharon Hall', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 114, Fremont, CA 94538', 'Sequoia Commons', '114', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 28
(gen_random_uuid(), 'Joseph Young', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 221, Fremont, CA 94538', 'Sequoia Commons', '221', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 29
(gen_random_uuid(), 'Donna Allen', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 326, Fremont, CA 94538', 'Sequoia Commons', '326', '94538',
```

```sql
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 30
(gen_random_uuid(), 'Richard King', NULL, 'single', 1, 0, NULL, NULL, '40789
Fremont Blvd, Unit 103, Fremont, CA 94538', 'Sequoia Commons', '103', '94538',
NULL, NULL, NULL, false, 'english', false, 'doorstep', NULL, NOW(), NOW(),
true),

-- Row 31 (Elena - blank last name and unit)
(gen_random_uuid(), 'Elena', NULL, 'single', 1, 0, NULL, NULL, '40789 Fremont
Blvd, Fremont, CA 94538', 'Sequoia Commons', NULL, '94538', NULL, NULL, NULL,
false, 'english', false, 'doorstep', NULL, NOW(), NOW(), true),

-- Row 32 (Gerny - blank unit)
(gen_random_uuid(), 'Gerny', NULL, 'single', 1, 0, NULL, NULL, '40789 Fremont
Blvd, Fremont, CA 94538', 'Sequoia Commons', NULL, '94538', NULL, NULL, NULL,
false, 'english', false, 'doorstep', NULL, NOW(), NOW(), true);

-- Verify the import
SELECT
   COUNT(*) as total_seniors_imported,
   COUNT(CASE WHEN building = 'Sequoia Commons' THEN 1 END) as
sequoia_commons_residents,
   COUNT(CASE WHEN address LIKE '%40789%' THEN 1 END) as correct_address_count,
   COUNT(CASE WHEN active = true THEN 1 END) as active_seniors
FROM seniors
WHERE building = 'Sequoia Commons';

-- Show a sample of imported data
SELECT
   name,
   building,
   unit_apt,
   address,
   active
FROM seniors
WHERE building = 'Sequoia Commons'
```

```
ORDER BY unit_apt NULLS LAST, name
LIMIT 15;
```

Remove Admin from Volunteers

```
-- Deactivate salaamfoodpantry@gmail.com in volunteers table
-- Run this AFTER running the previous script to make them super admin
-- This version avoids foreign key constraint issues by deactivating instead of
deleting

-- First, verify they exist in admins table
DO $$
BEGIN
   IF NOT EXISTS (
       SELECT 1 FROM admins
       WHERE email = 'salaamfoodpantry@gmail.com'
   ) THEN
       RAISE EXCEPTION 'User salaamfoodpantry@gmail.com not found in admins
table. Please run the previous script first.';
   END IF;
END $$;

-- Deactivate the user in volunteers table (safer than deleting)
UPDATE volunteers
SET active = false, updated_at = NOW()
WHERE email = 'salaamfoodpantry@gmail.com';

-- Verify the deactivation
SELECT
   CASE
       WHEN COUNT(*) = 0 THEN 'User not found in volunteers table'
       WHEN COUNT(*) = 1 AND NOT EXISTS (
           SELECT 1 FROM volunteers
           WHERE email = 'salaamfoodpantry@gmail.com' AND active = true
       ) THEN 'User successfully deactivated in volunteers table'
       ELSE 'User still active in volunteers table'
   END as status
FROM volunteers
WHERE email = 'salaamfoodpantry@gmail.com';
```

```
-- Show final status
SELECT
   'Final Status' as info,
   CASE
       WHEN EXISTS (SELECT 1 FROM admins WHERE email =
'salaamfoodpantry@gmail.com' AND active = true)
       THEN 'User is active in admins table'
       ELSE 'User not active in admins table'
   END as admin_status,
   CASE
       WHEN EXISTS (SELECT 1 FROM volunteers WHERE email =
'salaamfoodpantry@gmail.com' AND active = true)
       THEN 'User still active in volunteers table'
       ELSE 'User deactivated in volunteers table'
   END as volunteer_status;

-- Show user details in both tables
SELECT 'Admin Table' as table_name, email, name, role, active FROM admins WHERE
email = 'salaamfoodpantry@gmail.com'
UNION ALL
SELECT 'Volunteer Table' as table_name, email, name, role, active FROM
volunteers WHERE email = 'salaamfoodpantry@gmail.com';
```

Promote User to Primary Admin

```
-- Make salaamfoodpantry@gmail.com the Primary Admin with Super Admin role
-- This script moves the user from volunteers table to admins table

-- First, let's check if the user exists in volunteers table
DO $$
BEGIN
   IF NOT EXISTS (
       SELECT 1 FROM volunteers
       WHERE email = 'salaamfoodpantry@gmail.com'
   ) THEN
       RAISE EXCEPTION 'User salaamfoodpantry@gmail.com not found in volunteers
table';
   END IF;
```

```sql
END $$;

-- Insert the user into admins table with super_admin role
INSERT INTO admins (
   id,
   email,
   name,
   role,
   active,
   created_at,
   updated_at
)
SELECT
   id,
   email,
   name,
   'super_admin' as role,
   active,
   created_at,
   NOW() as updated_at
FROM volunteers
WHERE email = 'salaamfoodpantry@gmail.com'
ON CONFLICT (email) DO UPDATE SET
   role = 'super_admin',
   active = true,
   updated_at = NOW();

-- Update the user's auth metadata to reflect the new role
UPDATE auth.users
SET raw_user_meta_data = jsonb_set(
   COALESCE(raw_user_meta_data, '{}'::jsonb),
   '{role}',
   '"super_admin"'
)
WHERE email = 'salaamfoodpantry@gmail.com';

-- Optionally, you can remove the user from volunteers table
```

```sql
-- Uncomment the following line if you want to remove them from volunteers
table
-- DELETE FROM volunteers WHERE email = 'salaamfoodpantry@gmail.com';

-- Verify the change
SELECT
   'Admin created/updated successfully' as status,
   email,
   name,
   role,
   active
FROM admins
WHERE email = 'salaamfoodpantry@gmail.com';
```

Admin Email Update and Verification

```sql
-- Update the original super admin email
UPDATE admins
SET email = 'postzjaabir@gmail.com'
WHERE email = 'salaamfoodpantry@gmail.com';

-- Verify the change
SELECT id, email, name, role FROM admins WHERE role = 'super_admin';
```

Admin Role Evaluation

```sql
-- Add the missing updated_at column
ALTER TABLE admins
ADD COLUMN updated_at timestamp with time zone DEFAULT now();
```

Emergency RLS Disable and Policy Cleanup

```sql
-- Temporarily Disable RLS to Fix Infinite Recursion
-- This is a nuclear option to get the system working immediately

-- Step 1: Disable RLS on all tables completely
ALTER TABLE admins DISABLE ROW LEVEL SECURITY;
ALTER TABLE volunteers DISABLE ROW LEVEL SECURITY;
ALTER TABLE seniors DISABLE ROW LEVEL SECURITY;
ALTER TABLE deliveries DISABLE ROW LEVEL SECURITY;
```

```sql
-- Step 2: Drop ALL policies to clean slate
-- Admins policies
DROP POLICY IF EXISTS "Users can view own admin record" ON admins;
DROP POLICY IF EXISTS "Super admins can view all admin records" ON admins;
DROP POLICY IF EXISTS "Super admins can insert admin records" ON admins;
DROP POLICY IF EXISTS "Super admins can update admin records" ON admins;
DROP POLICY IF EXISTS "Users can update own last_login" ON admins;
DROP POLICY IF EXISTS "Super admins can delete admin records" ON admins;
DROP POLICY IF EXISTS "admins_select_policy" ON admins;
DROP POLICY IF EXISTS "admins_insert_policy" ON admins;
DROP POLICY IF EXISTS "admins_update_policy" ON admins;
DROP POLICY IF EXISTS "admins_delete_policy" ON admins;
DROP POLICY IF EXISTS "admins_allow_all" ON admins;

-- Volunteers policies
DROP POLICY IF EXISTS "volunteers_can_read_own_data" ON volunteers;
DROP POLICY IF EXISTS "volunteers_can_insert_own_data" ON volunteers;
DROP POLICY IF EXISTS "volunteers_can_update_own_data" ON volunteers;
DROP POLICY IF EXISTS "admins_can_manage_volunteers" ON volunteers;
DROP POLICY IF EXISTS "service_role_can_manage_volunteers" ON volunteers;
DROP POLICY IF EXISTS "volunteers_select_policy" ON volunteers;
DROP POLICY IF EXISTS "volunteers_insert_policy" ON volunteers;
DROP POLICY IF EXISTS "volunteers_update_policy" ON volunteers;
DROP POLICY IF EXISTS "volunteers_delete_policy" ON volunteers;
DROP POLICY IF EXISTS "volunteers_allow_all" ON volunteers;

-- Seniors policies
DROP POLICY IF EXISTS "volunteers_can_read_seniors" ON seniors;
DROP POLICY IF EXISTS "admins_can_manage_seniors" ON seniors;
DROP POLICY IF EXISTS "seniors_allow_all" ON seniors;

-- Deliveries policies
DROP POLICY IF EXISTS "volunteers_can_read_own_deliveries" ON deliveries;
DROP POLICY IF EXISTS "volunteers_can_update_own_deliveries" ON deliveries;
DROP POLICY IF EXISTS "admins_can_manage_deliveries" ON deliveries;
DROP POLICY IF EXISTS "deliveries_allow_all" ON deliveries;
```

```sql
-- Step 3: Verify admin user exists
SELECT
   'Admin User Status (RLS Disabled)' as status,
   au.email,
   au.email_confirmed_at,
   CASE WHEN v.id IS NOT NULL THEN 'Yes' ELSE 'No' END as volunteer_record,
   CASE WHEN a.id IS NOT NULL THEN 'Yes' ELSE 'No' END as admin_record,
   a.role as admin_role
FROM auth.users au
LEFT JOIN volunteers v ON au.id = v.id
LEFT JOIN admins a ON au.id = a.id
WHERE au.email = 'salaamfoodpantry@gmail.com';

-- Step 4: Test all table access
SELECT 'Admins table accessible' as test, COUNT(*) as count FROM admins;
SELECT 'Volunteers table accessible' as test, COUNT(*) as count FROM
volunteers;
SELECT 'Seniors table accessible' as test, COUNT(*) as count FROM seniors;
SELECT 'Deliveries table accessible' as test, COUNT(*) as count FROM
deliveries;

-- Step 5: Show current RLS status
SELECT
   schemaname,
   tablename,
   rowsecurity
FROM pg_tables
WHERE tablename IN ('admins', 'volunteers', 'seniors', 'deliveries')
ORDER BY tablename;
```

Admin User Insertion

```sql
-- Complete RLS Reset - Fix Infinite Recursion Issues
-- This script completely removes all RLS policies and recreates them properly

-- Step 1: Disable RLS on all tables
ALTER TABLE admins DISABLE ROW LEVEL SECURITY;
ALTER TABLE volunteers DISABLE ROW LEVEL SECURITY;
ALTER TABLE seniors DISABLE ROW LEVEL SECURITY;
```

```sql
ALTER TABLE deliveries DISABLE ROW LEVEL SECURITY;

-- Step 2: Drop ALL existing policies that might cause recursion
-- Admins table policies
DROP POLICY IF EXISTS "Users can view own admin record" ON admins;
DROP POLICY IF EXISTS "Super admins can view all admin records" ON admins;
DROP POLICY IF EXISTS "Super admins can insert admin records" ON admins;
DROP POLICY IF EXISTS "Super admins can update admin records" ON admins;
DROP POLICY IF EXISTS "Users can update own last_login" ON admins;
DROP POLICY IF EXISTS "Super admins can delete admin records" ON admins;
DROP POLICY IF EXISTS "admins_select_policy" ON admins;
DROP POLICY IF EXISTS "admins_insert_policy" ON admins;
DROP POLICY IF EXISTS "admins_update_policy" ON admins;
DROP POLICY IF EXISTS "admins_delete_policy" ON admins;

-- Volunteers table policies
DROP POLICY IF EXISTS "volunteers_can_read_own_data" ON volunteers;
DROP POLICY IF EXISTS "volunteers_can_insert_own_data" ON volunteers;
DROP POLICY IF EXISTS "volunteers_can_update_own_data" ON volunteers;
DROP POLICY IF EXISTS "admins_can_manage_volunteers" ON volunteers;
DROP POLICY IF EXISTS "service_role_can_manage_volunteers" ON volunteers;
DROP POLICY IF EXISTS "volunteers_select_policy" ON volunteers;
DROP POLICY IF EXISTS "volunteers_insert_policy" ON volunteers;
DROP POLICY IF EXISTS "volunteers_update_policy" ON volunteers;
DROP POLICY IF EXISTS "volunteers_delete_policy" ON volunteers;

-- Seniors table policies
DROP POLICY IF EXISTS "volunteers_can_read_seniors" ON seniors;
DROP POLICY IF EXISTS "admins_can_manage_seniors" ON seniors;

-- Deliveries table policies
DROP POLICY IF EXISTS "volunteers_can_read_own_deliveries" ON deliveries;
DROP POLICY IF EXISTS "volunteers_can_update_own_deliveries" ON deliveries;
DROP POLICY IF EXISTS "admins_can_manage_deliveries" ON deliveries;

-- Step 3: Re-enable RLS
ALTER TABLE admins ENABLE ROW LEVEL SECURITY;
ALTER TABLE volunteers ENABLE ROW LEVEL SECURITY;
```

```sql
ALTER TABLE seniors ENABLE ROW LEVEL SECURITY;
ALTER TABLE deliveries ENABLE ROW LEVEL SECURITY;

-- Step 4: Create simple, non-recursive policies
-- Admins table - allow all operations for now
CREATE POLICY "admins_allow_all" ON admins FOR ALL USING (true) WITH CHECK
(true);

-- Volunteers table - allow all operations for now
CREATE POLICY "volunteers_allow_all" ON volunteers FOR ALL USING (true) WITH
CHECK (true);

-- Seniors table - allow all operations for now
CREATE POLICY "seniors_allow_all" ON seniors FOR ALL USING (true) WITH CHECK
(true);

-- Deliveries table - allow all operations for now
CREATE POLICY "deliveries_allow_all" ON deliveries FOR ALL USING (true) WITH
CHECK (true);

-- Step 5: Verify admin user exists and is accessible
SELECT
   'Admin User Verification' as check_type,
   au.email,
   au.email_confirmed_at,
   CASE WHEN v.id IS NOT NULL THEN 'Yes' ELSE 'No' END as volunteer_record,
   CASE WHEN a.id IS NOT NULL THEN 'Yes' ELSE 'No' END as admin_record,
   a.role as admin_role
FROM auth.users au
LEFT JOIN volunteers v ON au.id = v.id
LEFT JOIN admins a ON au.id = a.id
WHERE au.email = 'salaamfoodpantry@gmail.com';

-- Step 6: Test basic queries
SELECT 'Testing admins table access' as test, COUNT(*) as count FROM admins;
SELECT 'Testing volunteers table access' as test, COUNT(*) as count FROM
volunteers;
SELECT 'Testing seniors table access' as test, COUNT(*) as count FROM seniors;
```

```
SELECT 'Testing deliveries table access' as test, COUNT(*) as count FROM
deliveries;
```

Admin User Management Table

```
CREATE TABLE IF NOT EXISTS admins (
 id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
 email text UNIQUE NOT NULL,
 name text NOT NULL,
 phone text,
 password_hash text, -- Optionally use for any custom flows, otherwise managed
via Supabase Auth
 role text DEFAULT 'admin', -- You can expand roles (admin, super_admin) if
needed
 active boolean DEFAULT true,
 created_at timestamp with time zone DEFAULT now(),
 last_login timestamp with time zone
);
```

Admin User Management Table

```
CREATE TABLE IF NOT EXISTS admins (
 id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
 email text UNIQUE NOT NULL,
 name text NOT NULL,
 phone text,
 password_hash text, -- Optionally use for any custom flows, otherwise managed
via Supabase Auth
 role text DEFAULT 'admin', -- You can expand roles (admin, super_admin) if
needed
 active boolean DEFAULT true,
 created_at timestamp with time zone DEFAULT now(),
 last_login timestamp with time zone
);
```


Senior Support Service Database Schema

```
-- Insert sample seniors first
```

```sql
INSERT INTO seniors (id, name, age, household_type, family_adults,
family_children, race_ethnicity, health_conditions, address,
dietary_restrictions, phone, emergency_contact) VALUES
 ('650e8400-e29b-41d4-a716-446655440001', 'Mrs. Eleanor Johnson', 78, 'single',
1, 0, 'Caucasian', 'Hypertension, Hard of hearing', '123 Oak Street, Apt 2A',
'Low sodium', '(555) 123-4567', 'Sarah Johnson (Daughter): (555) 987-6543'),
 ('650e8400-e29b-41d4-a716-446655440002', 'Mr. Robert Smith', 82, 'single', 1,
0, 'African American', 'Diabetes Type 2', '456 Pine Avenue, Unit 5B',
'Diabetic', '(555) 234-5678', 'Maria Rodriguez (Neighbor): (555) 876-5432'),
 ('650e8400-e29b-41d4-a716-446655440003', 'Mrs. Leah Shah', 75, 'single', 1, 0,
'South Asian', 'Arthritis', '789 Elm Drive, House', 'Vegetarian', '(555)
345-6789', 'Dr. Patel (Family Doctor): (555) 765-4321'),
 ('650e8400-e29b-41d4-a716-446655440004', 'Mr. Carlos Rodriguez', 80, 'family',
2, 0, 'Hispanic', 'None', '321 Maple Lane, Apt 1A', NULL, '(555) 456-7890',
'Isabella Rodriguez (Granddaughter): (555) 654-3210'),
 ('650e8400-e29b-41d4-a716-446655440005', 'Mrs. Amy Chen', 73, 'single', 1, 0,
'Asian', 'Celiac Disease', '654 Birch Road, Unit 3C', 'Gluten-free', '(555)
567-8901', 'David Chen (Son): (555) 543-2109');

-- Note: Volunteers will be created through the signup process
-- This ensures proper Supabase Auth integration

-- Function to create sample deliveries after volunteers are created
CREATE OR REPLACE FUNCTION create_sample_deliveries()
RETURNS void AS $$
DECLARE
 volunteer_id UUID;
 senior_ids UUID[] := ARRAY[
   '650e8400-e29b-41d4-a716-446655440001',
   '650e8400-e29b-41d4-a716-446655440002',
   '650e8400-e29b-41d4-a716-446655440003',
   '650e8400-e29b-41d4-a716-446655440004',
   '650e8400-e29b-41d4-a716-446655440005'
 ];
 i INTEGER;
BEGIN
 -- Get a volunteer ID (will be available after signup)
 SELECT id INTO volunteer_id FROM volunteers LIMIT 1;
```

```
  IF volunteer_id IS NOT NULL THEN
   -- Insert sample deliveries with corrected column order
   FOR i IN 1..5 LOOP
     INSERT INTO deliveries (senior_id, volunteer_id, delivery_date, status,
notes, completed_at) VALUES
       (senior_ids[i], volunteer_id, '2024-12-15',
        CASE
          WHEN i <= 3 THEN 'delivered'
          ELSE 'pending'
        END,
        CASE
          WHEN i <= 3 THEN 'Delivered successfully'
          ELSE NULL
        END,
        CASE
          WHEN i <= 3 THEN '2024-12-15 10:30:00'::timestamp
          ELSE NULL
        END);
   END LOOP;
 END IF;
END;
$$ LANGUAGE plpgsql;
```

Dynamic SQL Execution Function

```
create or replace function execute_sql(sql_statement text)
returns void as $$
begin
execute sql_statement;
end;
$$ language plpgsql;
```

Senior Meal Delivery System Initialization

```
-- Insert sample seniors first
INSERT INTO seniors (id, name, age, household_type, family_adults,
family_children, race_ethnicity, health_conditions, address,
dietary_restrictions, phone, emergency_contact) VALUES
```

```
 ('650e8400-e29b-41d4-a716-446655440001', 'Mrs. Eleanor Johnson', 78, 'single',
1, 0, 'Caucasian', 'Hypertension, Hard of hearing', '123 Oak Street, Apt 2A',
'Low sodium', '(555) 123-4567', 'Sarah Johnson (Daughter): (555) 987-6543'),
 ('650e8400-e29b-41d4-a716-446655440002', 'Mr. Robert Smith', 82, 'single', 1,
0, 'African American', 'Diabetes Type 2', '456 Pine Avenue, Unit 5B',
'Diabetic', '(555) 234-5678', 'Maria Rodriguez (Neighbor): (555) 876-5432'),
 ('650e8400-e29b-41d4-a716-446655440003', 'Mrs. Leah Shah', 75, 'single', 1, 0,
'South Asian', 'Arthritis', '789 Elm Drive, House', 'Vegetarian', '(555)
345-6789', 'Dr. Patel (Family Doctor): (555) 765-4321'),
 ('650e8400-e29b-41d4-a716-446655440004', 'Mr. Carlos Rodriguez', 80, 'family',
2, 0, 'Hispanic', 'None', '321 Maple Lane, Apt 1A', NULL, '(555) 456-7890',
'Isabella Rodriguez (Granddaughter): (555) 654-3210'),
 ('650e8400-e29b-41d4-a716-446655440005', 'Mrs. Amy Chen', 73, 'single', 1, 0,
'Asian', 'Celiac Disease', '654 Birch Road, Unit 3C', 'Gluten-free', '(555)
567-8901', 'David Chen (Son): (555) 543-2109');

-- Note: Volunteers will be created through the signup process
-- This ensures proper Supabase Auth integration

-- Function to create sample deliveries after volunteers are created
CREATE OR REPLACE FUNCTION create_sample_deliveries()
RETURNS void AS $$
DECLARE
 volunteer_id UUID;
 senior_ids UUID[] := ARRAY[
   '650e8400-e29b-41d4-a716-446655440001',
   '650e8400-e29b-41d4-a716-446655440002',
   '650e8400-e29b-41d4-a716-446655440003',
   '650e8400-e29b-41d4-a716-446655440004',
   '650e8400-e29b-41d4-a716-446655440005'
 ];
 i INTEGER;
BEGIN
 -- Get a volunteer ID (will be available after signup)
 SELECT id INTO volunteer_id FROM volunteers LIMIT 1;
  IF volunteer_id IS NOT NULL THEN
   -- Insert sample deliveries with corrected column order
   FOR i IN 1..5 LOOP
```

```
     INSERT INTO deliveries (senior_id, volunteer_id, delivery_date, status,
notes, completed_at) VALUES
       (senior_ids[i], volunteer_id, '2024-12-15',
        CASE
          WHEN i <= 3 THEN 'delivered'
          ELSE 'pending'
        END,
        CASE
          WHEN i <= 3 THEN 'Delivered successfully'
          ELSE NULL
        END,
        CASE
          WHEN i <= 3 THEN '2024-12-15 10:30:00'::timestamp
          ELSE NULL
        END);
   END LOOP;
 END IF;
END;
$$ LANGUAGE plpgsql;
```

Senior Meal Delivery Data Setup

```
-- Insert sample seniors with accessibility features
INSERT INTO public.seniors (
 id, name, age, household_type, family_adults, family_children,
 race_ethnicity, health_conditions, address, dietary_restrictions,
 phone, emergency_contact, has_smartphone, preferred_language,
 needs_translation, delivery_method, special_instructions
) VALUES
 (
   '650e8400-e29b-41d4-a716-446655440001',
   'Mrs. Eleanor Johnson', 78, 'single', 1, 0,
   'Caucasian', 'Hypertension, Hard of hearing',
   '123 Oak Street, Apt 2A', 'Low sodium',
   '(555) 123-4567', 'Sarah Johnson (Daughter): (555) 987-6543',
   false, 'english', false, 'phone_confirmed',
   'Ring doorbell twice. Hard of hearing.'
```

```
),
(
  '650e8400-e29b-41d4-a716-446655440002',
  'Mr. Robert Smith', 82, 'single', 1, 0,
  'African American', 'Diabetes Type 2',
  '456 Pine Avenue, Unit 5B', 'Diabetic',
  '(555) 234-5678', 'Maria Rodriguez (Neighbor): (555) 876-5432',
  true, 'english', false, 'phone_confirmed',
  'Prefers morning deliveries before 10 AM.'
),
(
  '650e8400-e29b-41d4-a716-446655440003',
  'Mrs. Leah Shah', 75, 'single', 1, 0,
  'South Asian', 'Arthritis',
  '789 Elm Drive, House', 'Vegetarian',
  '(555) 345-6789', 'Dr. Patel (Family Doctor): (555) 765-4321',
  false, 'urdu', true, 'family_member',
  'Daughter speaks English. Call family first.'
),
(
  '650e8400-e29b-41d4-a716-446655440004',
  'Mr. Carlos Rodriguez', 80, 'family', 2, 0,
  'Hispanic', 'None',
  '321 Maple Lane, Apt 1A', NULL,
  '(555) 456-7890', 'Isabella Rodriguez (Granddaughter): (555) 654-3210',
  true, 'spanish', true, 'phone_confirmed',
  'Apartment entrance code: #1234. Speaks some English.'
),
(
  '650e8400-e29b-41d4-a716-446655440005',
  'Mrs. Amy Chen', 73, 'single', 1, 0,
  'Asian', 'Celiac Disease',
  '654 Birch Road, Unit 3C', 'Gluten-free',
  '(555) 567-8901', 'David Chen (Son): (555) 543-2109',
  true, 'english', false, 'doorstep',
  'Leave at door if no answer. Usually home in afternoons.'
);
```

```sql
-- Function to create sample deliveries after volunteers are created
CREATE OR REPLACE FUNCTION create_sample_deliveries()
RETURNS void AS $$
DECLARE
 volunteer_id UUID;
 senior_ids UUID[] := ARRAY[
   '650e8400-e29b-41d4-a716-446655440001',
   '650e8400-e29b-41d4-a716-446655440002',
   '650e8400-e29b-41d4-a716-446655440003',
   '650e8400-e29b-41d4-a716-446655440004',
   '650e8400-e29b-41d4-a716-446655440005'
 ];
 i INTEGER;
BEGIN
 -- Get a volunteer ID (will be available after signup)
 SELECT id INTO volunteer_id FROM public.volunteers LIMIT 1;
  IF volunteer_id IS NOT NULL THEN
   -- Insert sample deliveries
   FOR i IN 1..5 LOOP
     INSERT INTO public.deliveries (
       senior_id, volunteer_id, delivery_date, status,
       delivery_method, notes, language_barrier_encountered,
       translation_needed, completed_at
     ) VALUES (
       senior_ids[i],
       volunteer_id,
       '2024-12-15',
       CASE
         WHEN i <= 3 THEN 'delivered'
         ELSE 'pending'
       END,
       CASE
         WHEN i = 1 THEN 'phone_confirmed'
         WHEN i = 2 THEN 'phone_confirmed'
         WHEN i = 3 THEN 'family_member'
         ELSE 'doorstep'
       END,
       CASE
```

```
        WHEN i <= 3 THEN 'Delivered successfully'
        ELSE NULL
      END,
      CASE WHEN i = 3 THEN true ELSE false END,
      CASE WHEN i IN (3, 4) THEN true ELSE false END,
      CASE
        WHEN i <= 3 THEN '2024-12-15 10:30:00'::timestamp
        ELSE NULL
      END
    );
  END LOOP;
 END IF;
END;
$$ LANGUAGE plpgsql;
```

Senior Meal Delivery Schema

```
-- Enable Row Level Security
ALTER TABLE public.seniors ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.volunteers ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.deliveries ENABLE ROW LEVEL SECURITY;

-- Drop existing policies if they exist
DROP POLICY IF EXISTS "volunteers_can_read_own_data" ON public.volunteers;
DROP POLICY IF EXISTS "volunteers_can_insert_own_data" ON public.volunteers;
DROP POLICY IF EXISTS "volunteers_can_update_own_data" ON public.volunteers;
DROP POLICY IF EXISTS "admins_can_manage_volunteers" ON public.volunteers;

-- Volunteers table policies (non-recursive)
CREATE POLICY "volunteers_can_read_own_data" ON public.volunteers
 FOR SELECT USING (auth.uid() = id);

CREATE POLICY "volunteers_can_insert_own_data" ON public.volunteers
 FOR INSERT WITH CHECK (auth.uid() = id);

CREATE POLICY "volunteers_can_update_own_data" ON public.volunteers
 FOR UPDATE USING (auth.uid() = id) WITH CHECK (auth.uid() = id);
```

```sql
-- Admin access based on auth metadata (non-recursive)
CREATE POLICY "admins_can_manage_volunteers" ON public.volunteers
 FOR ALL USING (
   (auth.jwt() ->> 'email') = 'salaamfoodpantry@gmail.com' OR
   (auth.jwt() -> 'user_metadata' ->> 'role') = 'admin'
 );

-- Seniors table policies
CREATE POLICY "volunteers_can_read_seniors" ON public.seniors
 FOR SELECT USING (active = true);

CREATE POLICY "admins_can_manage_seniors" ON public.seniors
 FOR ALL USING (
   (auth.jwt() ->> 'email') = 'salaamfoodpantry@gmail.com' OR
   (auth.jwt() -> 'user_metadata' ->> 'role') = 'admin'
 );

-- Deliveries table policies
CREATE POLICY "volunteers_can_read_own_deliveries" ON public.deliveries
 FOR SELECT USING (volunteer_id = auth.uid());

CREATE POLICY "volunteers_can_update_own_deliveries" ON public.deliveries
 FOR UPDATE USING (volunteer_id = auth.uid()) WITH CHECK (volunteer_id =
auth.uid());

CREATE POLICY "admins_can_manage_deliveries" ON public.deliveries
 FOR ALL USING (
   (auth.jwt() ->> 'email') = 'salaamfoodpantry@gmail.com' OR
   (auth.jwt() -> 'user_metadata' ->> 'role') = 'admin'
 );

-- Storage policies for profile pictures
CREATE POLICY "users_can_upload_own_picture" ON storage.objects
 FOR INSERT WITH CHECK (
   bucket_id = 'profile-pictures' AND
   auth.uid()::text = (storage.foldername(name))[1]
 );
```

```
CREATE POLICY "users_can_view_pictures" ON storage.objects
 FOR SELECT USING (bucket_id = 'profile-pictures');

CREATE POLICY "users_can_update_own_picture" ON storage.objects
 FOR UPDATE USING (
   bucket_id = 'profile-pictures' AND
   auth.uid()::text = (storage.foldername(name))[1]
 );
```

Senior Meal Delivery Schema

```
-- Enable UUID extension
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Create seniors table with accessibility features
CREATE TABLE public.seniors (
 id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
 name TEXT NOT NULL,
 age INTEGER,
 household_type TEXT CHECK (household_type IN ('single', 'family')) DEFAULT
'single',
 family_adults INTEGER DEFAULT 1,
 family_children INTEGER DEFAULT 0,
 race_ethnicity TEXT,
 health_conditions TEXT,
 address TEXT NOT NULL,
 dietary_restrictions TEXT,
 phone TEXT,
 emergency_contact TEXT,
 has_smartphone BOOLEAN DEFAULT false,
 preferred_language TEXT DEFAULT 'english',
 needs_translation BOOLEAN DEFAULT false,
 delivery_method TEXT CHECK (delivery_method IN ('doorstep', 'phone_confirmed',
'family_member')) DEFAULT 'doorstep',
 special_instructions TEXT,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
```

```sql
 active BOOLEAN DEFAULT TRUE
);

-- Create volunteers table with language capabilities
CREATE TABLE public.volunteers (
 id UUID PRIMARY KEY,
 email TEXT UNIQUE NOT NULL,
 name TEXT NOT NULL,
 phone TEXT,
 role TEXT CHECK (role IN ('volunteer', 'admin')) DEFAULT 'volunteer',
 speaks_languages TEXT[] DEFAULT ARRAY['english'],
 profile_picture TEXT,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 active BOOLEAN DEFAULT TRUE
);

-- Create deliveries table with enhanced tracking
CREATE TABLE public.deliveries (
 id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
 senior_id UUID NOT NULL REFERENCES public.seniors(id) ON DELETE CASCADE,
 volunteer_id UUID NOT NULL REFERENCES public.volunteers(id) ON DELETE CASCADE,
 delivery_date DATE NOT NULL,
 status TEXT CHECK (status IN ('pending', 'delivered', 'missed', 'no_contact',
'family_confirmed')) DEFAULT 'pending',
 delivery_method TEXT CHECK (delivery_method IN ('doorstep', 'phone_confirmed',
'family_member')),
 notes TEXT,
 language_barrier_encountered BOOLEAN DEFAULT false,
 translation_needed BOOLEAN DEFAULT false,
 created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
 completed_at TIMESTAMP WITH TIME ZONE
);

-- Create profile pictures storage bucket
INSERT INTO storage.buckets (id, name, public)
VALUES ('profile-pictures', 'profile-pictures', true)
ON CONFLICT (id) DO NOTHING;
```

```
-- Create indexes for better performance
CREATE INDEX idx_seniors_active ON public.seniors(active);
CREATE INDEX idx_seniors_language ON public.seniors(preferred_language);
CREATE INDEX idx_seniors_smartphone ON public.seniors(has_smartphone);
CREATE INDEX idx_volunteers_active ON public.volunteers(active);
CREATE INDEX idx_volunteers_languages ON public.volunteers USING
GIN(speaks_languages);
CREATE INDEX idx_deliveries_date ON public.deliveries(delivery_date);
CREATE INDEX idx_deliveries_status ON public.deliveries(status);
CREATE INDEX idx_deliveries_senior ON public.deliveries(senior_id);
CREATE INDEX idx_deliveries_volunteer ON public.deliveries(volunteer_id);
```

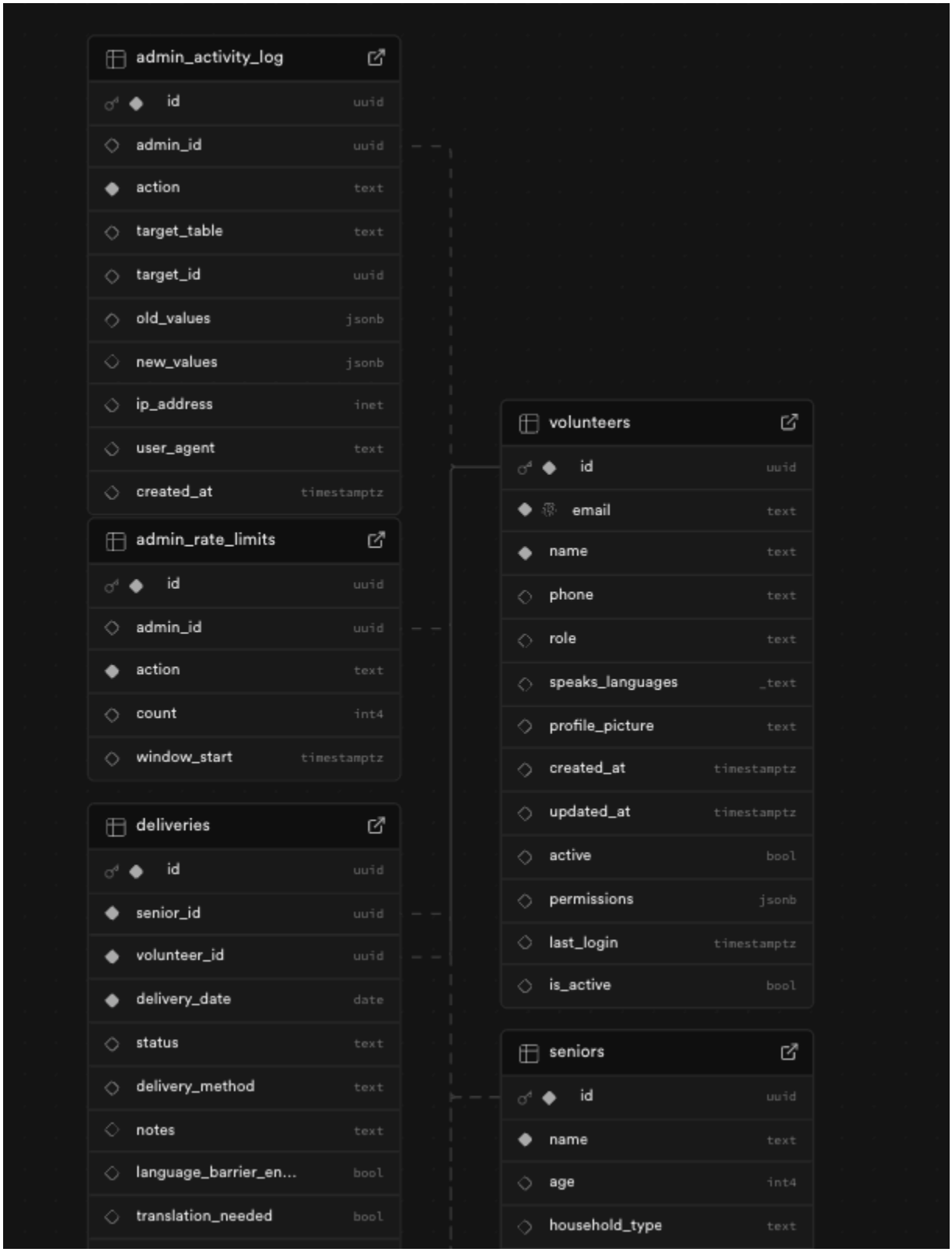
Running Queries

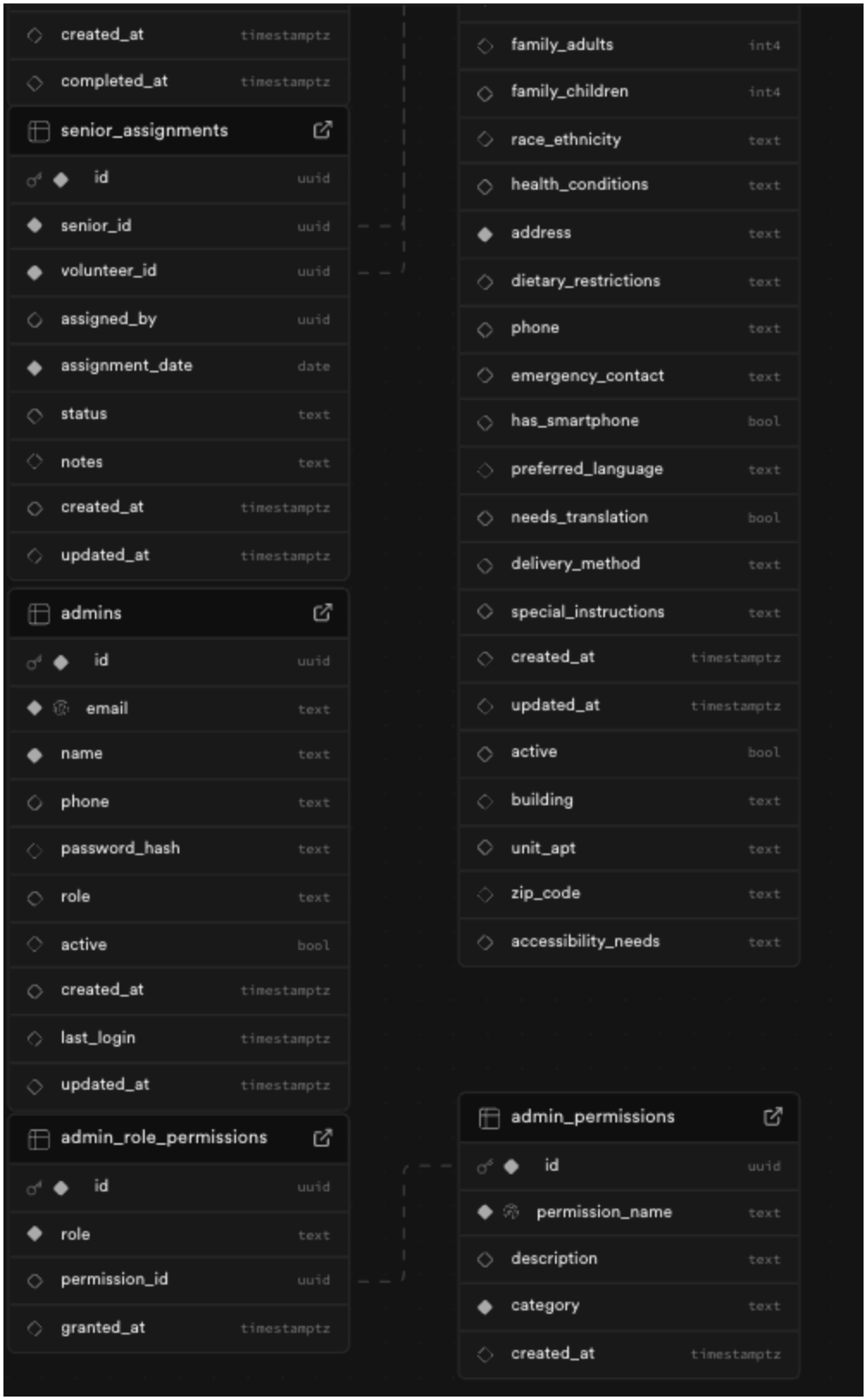
```
SET statement_timeout='58s'; SET idle_session_timeout='58s';set pg_stat_statements.track = none;
set local pgrst.db_schemas = 'public,graphql_public';
set local search_path = '';

(
with foreign_keys as (
  select
      cl.relnamespace::regnamespace::text as schema_name,
      cl.relname as table_name,
      cl.oid as table_oid,
      ct.conname as fkey_name,
      ct.conkey as col_attnums
  from
      pg_catalog.pg_constraint ct
      join pg_catalog.pg_class cl -- fkey owning table
          on ct.conrelid = cl.oid
      left join pg_catalog.pg_depend d
          on d.objid = cl.oid
          and d.deptype = 'e'
  where
      ct.contype = 'f' -- foreign key constraints
```

```
        and d.objid is null -- exclude tables that are dependencies of extensions
        and cl.relnamespace::regnamespace::text not in (
            'pg_catalog', 'information_schema', 'auth', 'storage', 'vault', 'extensions'
        )
),
index_ as (
    select
        pi.indrelid as table_oid,
        indexre
```

```
SET statement_timeout='58s'; SET idle_session_timeout='58s';select pid, query, query_start
from pg_stat_activity where state = 'active' and datname = 'postgres';

-- source: dashboard
-- user: 4f977633-89b8-4917-878c-a0f2bd808090
-- date: 2025-08-04T03:02:47.521Z
```

Schema Visualizer




created_at timestamptz
completed_at timestamptz
senior_assignments
id uuid
senior_id uuid
volunteer_id uuid
assigned_by uuid
assignment_date date
status text
notes text
created_at timestamptz
updated_at timestamptz
admins
id uuid
email text
name text
phone text
password_hash text
role text
active bool
created_at timestamptz
last_login timestamptz
updated_at timestamptz
admin_role_permissions
id uuid
role text
permission_id uuid
granted_at timestamptz
family_adults int4
family_children int4
race_ethnicity text
health_conditions text
address text
dietary_restrictions text
phone text
emergency_contact text
has_smartphone bool
preferred_language text
needs_translation bool
delivery_method text
special_instructions text
created_at timestamptz
updated_at timestamptz
active bool
building text
unit_apt text
zip_code text
accessibility_needs text
admin_permissions
id uuid
permission_name text
description text
category text
created_at timestamptz

Database Tables

Database Tables

schema public | Search for a table | + New table

Name	Description	Rows (Estimated)	Size (Estimated)	Realtime Enabled	
admin_activity_log	No description	16	72 kB	×	10 columns
admin_dashboard_stats	No description	-	-	×	5 columns
admin_permissions	No description	8	48 kB	×	5 columns
admin_rate_limits	No description	0	24 kB	×	5 columns
admin_role_permissions	No description	9	48 kB	×	4 columns
admins	No description	2	96 kB	×	10 columns
deliveries	No description	19	96 kB	×	11 columns
senior_assignments	No description	2	96 kB	×	9 columns
senior_assignments_view	No description	-	-	×	16 columns
seniors	No description	36	112 kB	×	24 columns
volunteers	No description	2	88 kB	×	13 columns

Database Tables › admin_activity_log

Filter columns

+ New column

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✗
admin_id	No description	uuid	uuid	✓
action	No description	text	text	✗
target_table	No description	text	text	✓
target_id	No description	uuid	uuid	✓
old_values	No description	jsonb	jsonb	✓
new_values	No description	jsonb	jsonb	✓
ip_address	No description	inet	inet	✓
user_agent	No description	text	text	✓
created_at	No description	timestamp with time zone	timestamptz	✓

Database Tables › admin_dashboard_stats

Filter columns

Name	Description	Data Type	Format	Nullable
total_seniors	No description	bigint	int8	✓
total_volunteers	No description	bigint	int8	✓
monthly_deliveries	No description	bigint	int8	✓
pending_deliveries	No description	bigint	int8	✓
total_admins	No description	bigint	int8	✓

Database Tables › admin_permissions

Filter columns

+ New column

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✗
permission_name	No description	text	text	✗
description	No description	text	text	✓
category	No description	text	text	✗
created_at	No description	timestamp with time zone	timestamptz	✓

Database Tables › admin_role_permissions

Filter columns

+ New column

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✗
role	No description	text	text	✗
permission_id	No description	uuid	uuid	✓
granted_at	No description	timestamp with time zone	timestamptz	✓

Database Tables › admins

Filter columns

+ New column

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✗
email	No description	text	text	✗
name	No description	text	text	✗
phone	No description	text	text	✓
password_hash	No description	text	text	✓
role	No description	text	text	✓
active	No description	boolean	bool	✓
created_at	No description	timestamp with time zone	timestamptz	✓
last_login	No description	timestamp with time zone	timestamptz	✓
updated_at	No description	timestamp with time zone	timestamptz	✓

Database Tables › deliveries

Filter columns

+ New column

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✗
senior_id	No description	uuid	uuid	✗
volunteer_id	No description	uuid	uuid	✗
delivery_date	No description	date	date	✗
status	No description	text	text	✓
delivery_method	No description	text	text	✓
notes	No description	text	text	✓
language_barrier_encountered	No description	boolean	bool	✓
translation_needed	No description	boolean	bool	✓
created_at	No description	timestamp with time zone	timestamptz	✓
completed_at	No description	timestamp with time zone	timestamptz	✓

Database Tables › senior_assignments

Filter columns

+ New column

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✗
senior_id	No description	uuid	uuid	✗
volunteer_id	No description	uuid	uuid	✗
assigned_by	No description	uuid	uuid	✓
assignment_date	No description	date	date	✗
status	No description	text	text	✓
notes	No description	text	text	✓
created_at	No description	timestamp with time zone	timestamptz	✓
updated_at	No description	timestamp with time zone	timestamptz	✓

Database Tables › senior_assignments_view

Filter columns

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✓
senior_id	No description	uuid	uuid	✓
volunteer_id	No description	uuid	uuid	✓
assigned_by	No description	uuid	uuid	✓
assignment_date	No description	date	date	✓
status	No description	text	text	✓
notes	No description	text	text	✓
created_at	No description	timestamp with time zone	timestamptz	✓
updated_at	No description	timestamp with time zone	timestamptz	✓
senior_name	No description	text	text	✓
senior_address	No description	text	text	✓
senior_building	No description	text	text	✓
senior_unit	No description	text	text	✓
volunteer_name	No description	text	text	✓
volunteer_email	No description	text	text	✓
assigned_by_name	No description	text	text	✓

Database Tables › seniors

Filter columns

+ New column

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✗
name	No description	text	text	✗
age	No description	integer	int4	✓
household_type	No description	text	text	✓
family_adults	No description	integer	int4	✓
family_children	No description	integer	int4	✓
race_ethnicity	No description	text	text	✓
health_conditions	No description	text	text	✓
address	No description	text	text	✗
dietary_restrictions	No description	text	text	✓
phone	No description	text	text	✓
emergency_contact	No description	text	text	✓
has_smartphone	No description	boolean	bool	✓
preferred_language	No description	text	text	✓
needs_translation	No description	boolean	bool	✓
delivery_method	No description	text	text	✓
special_instructions	No description	text	text	✓
created_at	No description	timestamp with time zone	timestamptz	✓
updated_at	No description	timestamp with time zone	timestamptz	✓
active	No description	boolean	bool	✓
building	No description	text	text	✓
unit_apt	No description	text	text	✓
zip_code	No description	text	text	✓
accessibility_needs	No description	text	text	✓

Database Tables > volunteers

Filter columns

New column

Name	Description	Data Type	Format	Nullable
id	No description	uuid	uuid	✗
email	No description	text	text	✗
name	No description	text	text	✗
phone	No description	text	text	✓
role	No description	text	text	✓
speaks_languages	No description	ARRAY	_text	✓
profile_picture	No description	text	text	✓
created_at	No description	timestamp with time zone	timestamptz	✓
updated_at	No description	timestamp with time zone	timestamptz	✓
active	No description	boolean	bool	✓
permissions	No description	jsonb	jsonb	✓
last_login	No description	timestamp with time zone	timestamptz	✓
is_active	No description	boolean	bool	✓

Database Functions

Database Functions

Docs

schema public

Search for a function

Create a new function

Name	Arguments	Return type	Security
create_admin_user	user_email text, user_name text...	uuid	Definer
create_sample_deliveries	-	void	Invoker
execute_sql	sql_statement text	void	Invoker
get_admin_role	user_id uuid DEFAULT auth.uid()	text	Definer
handle_new_user	-	trigger	Definer
is_admin	user_id uuid DEFAULT auth.uid()	boolean	Definer
is_super_admin	user_id uuid DEFAULT auth.uid()	boolean	Definer
log_admin_activity	-	trigger	Definer
update_admins_updated_at	-	trigger	Invoker
update_senior_assignments_update	-	trigger	Invoker
user_has_permission	permission_name text	boolean	Definer

Edit 'create_admin_user' function

Name of function

create_admin_user

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

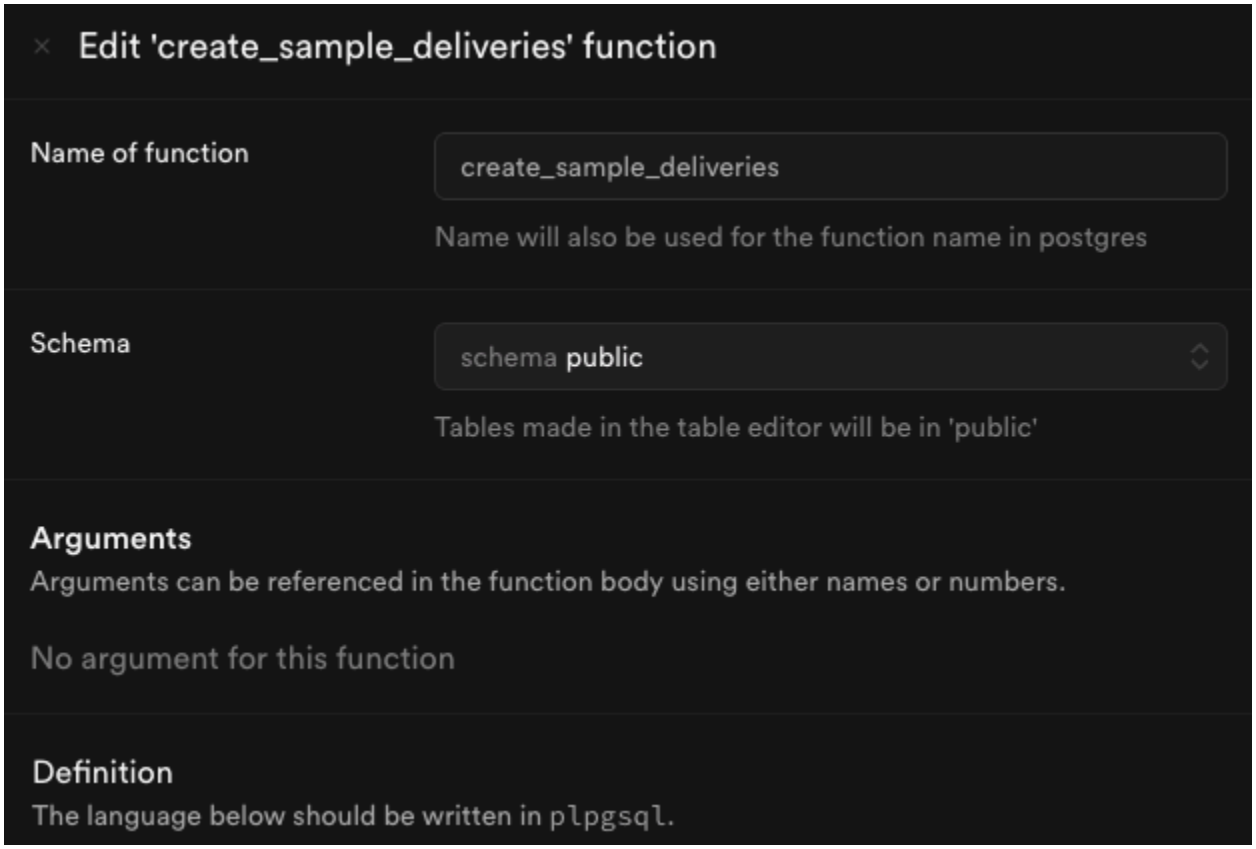
user_email	text
user_name	text
user_phone	text
admin_role	text

Definition

The language below should be written in `plpgsql`.

```
DECLARE
 new_user_id uuid;
BEGIN
 -- Check if caller is super_admin
 IF NOT EXISTS (
   SELECT 1 FROM volunteers
   WHERE id = auth.uid()
   AND role = 'super_admin'
 ) THEN
   RAISE EXCEPTION 'Only super admins can create admin users';
 END IF;
```

```
 -- Create auth user (this would typically be done via Supabase Auth API)
 -- For now, we'll create a placeholder record
 new_user_id := gen_random_uuid();
  -- Insert into volunteers table with admin role
 INSERT INTO volunteers (id, email, name, phone, role, active)
 VALUES (new_user_id, user_email, user_name, user_phone, admin_role, true);
  -- Log the activity
 INSERT INTO admin_activity_log (admin_id, action, target_table, target_id)
 VALUES (auth.uid(), 'CREATE_ADMIN_USER', 'volunteers', new_user_id);
  RETURN new_user_id;
END;
```




```
DECLARE
 volunteer_id UUID;
 senior_ids UUID[] := ARRAY[
   '650e8400-e29b-41d4-a716-446655440001',
```

```
   '650e8400-e29b-41d4-a716-446655440002',
   '650e8400-e29b-41d4-a716-446655440003',
   '650e8400-e29b-41d4-a716-446655440004',
   '650e8400-e29b-41d4-a716-446655440005'
 ];
 i INTEGER;
BEGIN
 -- Get a volunteer ID (will be available after signup)
 SELECT id INTO volunteer_id FROM public.volunteers LIMIT 1;
  IF volunteer_id IS NOT NULL THEN
   -- Insert sample deliveries
   FOR i IN 1..5 LOOP
     INSERT INTO public.deliveries (
       senior_id, volunteer_id, delivery_date, status,
       delivery_method, notes, language_barrier_encountered,
       translation_needed, completed_at
     ) VALUES (
       senior_ids[i],
       volunteer_id,
       '2024-12-15',
       CASE
         WHEN i <= 3 THEN 'delivered'
         ELSE 'pending'
       END,
       CASE
         WHEN i = 1 THEN 'phone_confirmed'
         WHEN i = 2 THEN 'phone_confirmed'
         WHEN i = 3 THEN 'family_member'
         ELSE 'doorstep'
       END,
       CASE
         WHEN i <= 3 THEN 'Delivered successfully'
         ELSE NULL
       END,
       CASE WHEN i = 3 THEN true ELSE false END,
       CASE WHEN i IN (3, 4) THEN true ELSE false END,
       CASE
         WHEN i <= 3 THEN '2024-12-15 10:30:00'::timestamp
```

```
          ELSE NULL
        END
      );
    END LOOP;
  END IF;
END;
```

Edit 'execute_sql' function

Name of function

execute_sql

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

sql_statement	text

Definition

The language below should be written in `plpgsql`.

```

begin

execute
sql_statement;

end;

```

Edit 'get_admin_role' function

Name of function

get_admin_role

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

user_id	uuid

Definition

The language below should be written in `plpgsql`.

```

BEGIN
    RETURN (
        SELECT role FROM admins
        WHERE id = user_id AND active = true
    );
END;

```

Edit 'handle_new_user' function

Name of function

handle_new_user

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

No argument for this function

Definition

The language below should be written in `plpgsql`.

```

BEGIN
    INSERT INTO volunteers (id, name, email, phone, role)
    VALUES (
        NEW.id,
        COALESCE(NEW.raw_user_meta_data->>'name', 'New User'),
        NEW.email,
        NEW.raw_user_meta_data->>'phone',
        'volunteer'
    );
    RETURN NEW;
END;

```

Edit 'is_admin' function

Name of function

is_admin

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

user_id	uuid

Definition

The language below should be written in `plpgsql`.

```
1
2   BEGIN
3       RETURN EXISTS (
4           SELECT 1 FROM admins
5           WHERE id = user_id AND active = true
6       );
7   END;
8
```

Edit 'is_super_admin' function

Name of function

is_super_admin

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

user_id	uuid

Definition

The language below should be written in `plpgsql`.

```
1
2   BEGIN
3       RETURN EXISTS (
4           SELECT 1 FROM admins
5           WHERE id = user_id AND role = 'super_admin' AND active = true
6       );
7   END;
8
```

Edit 'log_admin_activity' function

Name of function

log_admin_activity

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

No argument for this function

Definition

The language below should be written in `plpgsql`.

```
BEGIN
 -- Only log for admin users
 IF EXISTS (
   SELECT 1 FROM volunteers
   WHERE id = auth.uid()
   AND role IN ('admin', 'super_admin')
 ) THEN
   INSERT INTO admin_activity_log (
     admin_id,
     action,
     target_table,
     target_id,
     old_values,
     new_values
   ) VALUES (
     auth.uid(),
     TG_OP,
     TG_TABLE_NAME,
```

```
    COALESCE(NEW.id, OLD.id),
    CASE WHEN TG_OP = 'DELETE' THEN row_to_json(OLD) ELSE NULL END,
    CASE WHEN TG_OP IN ('INSERT', 'UPDATE') THEN row_to_json(NEW) ELSE NULL
END
  );
 END IF;
  RETURN COALESCE(NEW, OLD);
END;
```

Edit 'update_admins_updated_at' function

Name of function

update_admins_updated_at

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

No argument for this function

Definition

The language below should be written in `plpgsql`.

```
1
2  BEGIN
3      NEW.updated_at = NOW();
4      RETURN NEW;
5  END;
6
```

Edit 'update_senior_assignments_updated_at' fun...

Name of function

update_senior_assignments_updated_at

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

No argument for this function

Definition

The language below should be written in `plpgsql`.

```
1
2  BEGIN
3    NEW.updated_at = NOW();
4    RETURN NEW;
5  END;
6
```

Edit 'user_has_permission' function

Name of function

user_has_permission

Name will also be used for the function name in postgres

Schema

schema **public**

Tables made in the table editor will be in 'public'

Arguments

Arguments can be referenced in the function body using either names or numbers.

permission_name	text

Definition

The language below should be written in `plpgsql`.

```

DECLARE
  user_role text;
BEGIN
  -- Get user role
  SELECT role INTO user_role
  FROM volunteers
  WHERE id = auth.uid();

  -- Check if user has the permission
  RETURN EXISTS (
    SELECT 1
    FROM admin_role_permissions arp
    JOIN admin_permissions ap ON arp.permission_id = ap.id
```

Database Triggers

Database Triggers

Execute a set of actions automatically on specified table events

Docs

schema public | Search for a trigger | + New trigger

Name	Table	Function	Events	Orientation	Enabled
seniors_audit_trigger	seniors	log_admin_activity	AFTER UPDATE, AFTER DELETE, AFTER INSERT	ROW	✓
update_admins_updated_at	admins	update_admins_updated_at	BEFORE UPDATE	ROW	✓
update_senior_assignments...	senior_assignments	update_senior_assignments_updated_at	BEFORE UPDATE	ROW	✓
volunteers_audit_trigger	volunteers	log_admin_activity	AFTER UPDATE, AFTER DELETE, AFTER INSERT	ROW	✓

Database Policies

admin_activity_log — Disable RLS | Create policy

SELECT admins_can_view_activity_log
Applied to: authenticated role

admin_permissions — Disable RLS | Create policy

SELECT admins_can_view_permissions
Applied to: authenticated role

admin_rate_limits — Enable RLS | Create policy

Warning: Row Level Security is disabled. Your table is publicly readable and writable.

No policies created yet

admin_role_permissions — Enable RLS | Create policy

Warning: Row Level Security is disabled. Your table is publicly readable and writable.

No policies created yet

admins

Enable RLS | Create policy

Warning: Row Level Security is disabled. Your table is publicly readable and writable.

SELECT	Admin can view own record	Applied to: public role
ALL	Super admin can manage all	Applied to: public role

deliveries

Enable RLS | Create policy

Warning: Row Level Security is disabled. Your table is publicly readable and writable.

No policies created yet

senior_assignments

Disable RLS | Create policy

DELETE	Admins can delete senior assignments	Applied to: public role
INSERT	Admins can insert senior assignments	Applied to: public role
UPDATE	Admins can update senior assignments	Applied to: public role
SELECT	Admins can view all senior assignments	Applied to: public role
SELECT	Volunteers can view their own assignments	Applied to: public role

seniors

Enable RLS | Create policy

Warning: Row Level Security is disabled. Your table is publicly readable and writable.

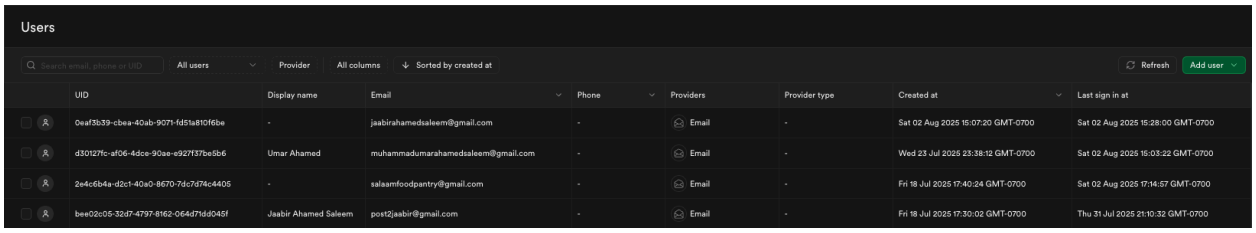
No policies created yet

volunteers

Disable RLS | Create policy

ALL	Admins can manage volunteers	Applied to: public role
SELECT	Admins can read all volunteers	Applied to: public role
ALL	admins_can_manage_all_volunteers	Applied to: authenticated role
SELECT	Volunteers can read own record	Applied to: public role
UPDATE	Volunteers can update own record	Applied to: public role
SELECT	volunteers_can_view_own_record	Applied to: authenticated role

Database Authentication

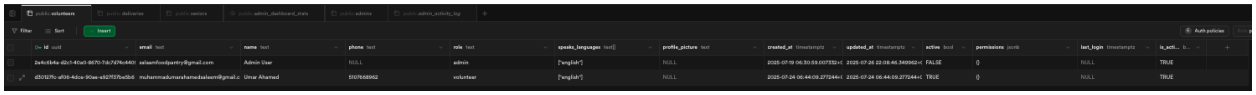


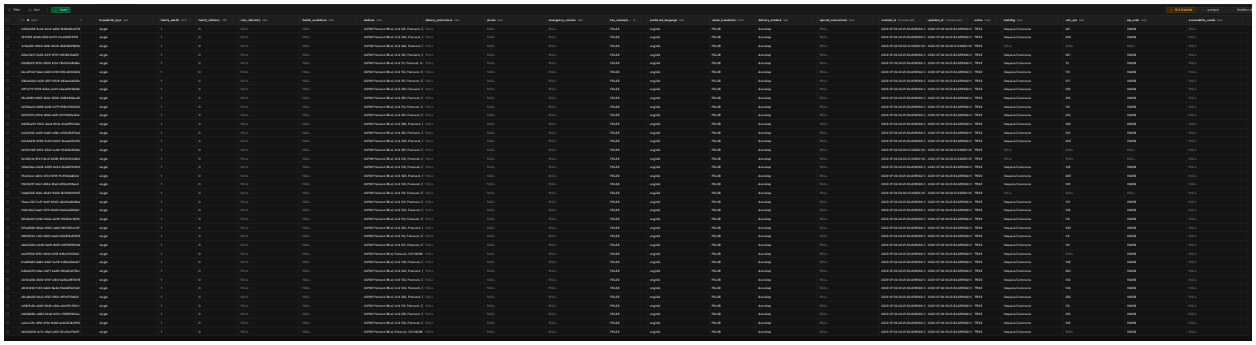
Users

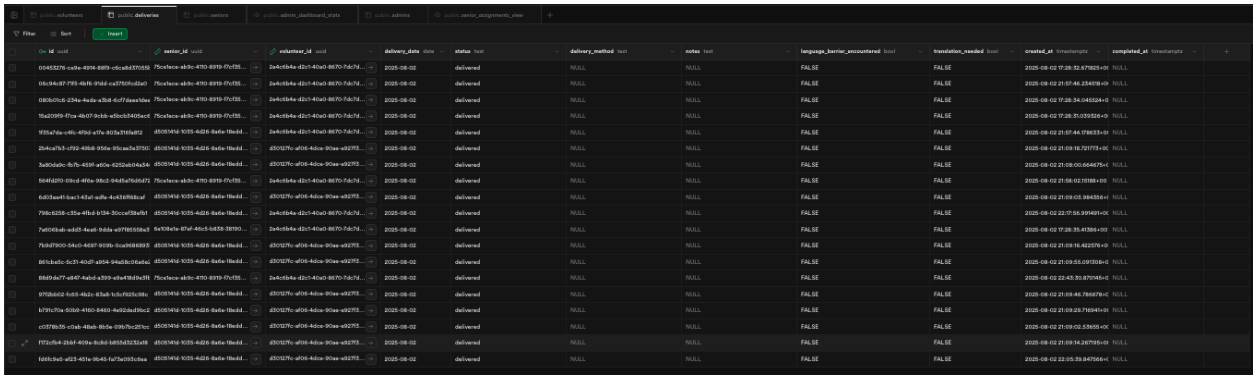
Search email, phone or UID | All users | Provider | All columns | Sorted by created at | Refresh | Add user

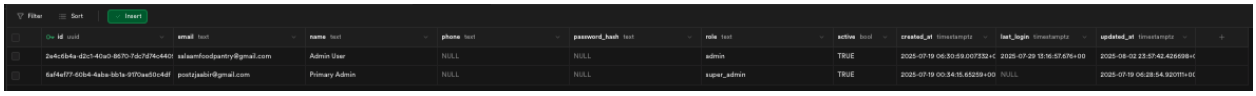
UID	Display name	Email	Phone	Providers	Provider type	Created at	Last sign in at
0eaf3b39-cbea-40ab-9071-fd51a810f6be	-	jaabirahamedsaleem@gmail.com	-	Email	-	Sat 02 Aug 2025 15:07:20 GMT-0700	Sat 02 Aug 2025 15:28:00 GMT-0700
d30127fc-af06-4dce-90ae-e927f37be5b6	Umar Ahamed	muhammadumarahamedsaleem@gmail.com	-	Email	-	Wed 23 Jul 2025 23:38:12 GMT-0700	Sat 02 Aug 2025 15:03:22 GMT-0700
2e4c6b4a-d2c1-40a0-8670-7dc7d74c4405	-	salaamfoodpantry@gmail.com	-	Email	-	Fri 18 Jul 2025 17:40:24 GMT-0700	Sat 02 Aug 2025 17:14:57 GMT-0700
bee02c05-32d7-4797-8162-064d71dd045f	Jaabir Ahamed Saleem	post2jaabir@gmail.com	-	Email	-	Fri 18 Jul 2025 17:30:02 GMT-0700	Thu 31 Jul 2025 21:10:32 GMT-0700

Database Table With Real Data









public.volunteers | public.deliveries | public.seniors | public.admin_dashboard_stats | public.admins | *public.admin_role_permissions*

Filter | Sort | Insert

id uuid	role text	permission_id uuid	granted_at timestamptz
2d873d95-030c-4c90-84b1-8b3d3538934	admin	eb4a4212-2741-484a-aadc-c8978...	2025-07-16 17:23:28.751986+00
4f3e38d4-8c76-4b15-a3d2-34d551f80fd9	admin	3cbc13b3-e341-4845-8977-3dee7...	2025-07-16 17:23:28.751986+00
69fc16ab-a3df-49b4-977a-15bf0defd21f	volunteer	64494f43-3a38-4b16-a04c-5e8e8...	2025-07-16 17:23:28.751986+00
72572187-032a-4aff-bc7a-56f504795a3d	admin	c8831f72-023f-4e0b-a4dd-8c001...	2025-07-16 17:23:28.751986+00
a2c0fa6f-14b6-4973-ab02-e5a8ce23be7b	admin	0b4cbebc-9d22-4e4d-85c3-224e...	2025-07-16 17:23:28.751986+00
c221c8ce-e409-4f38-95f0-56b450d6445b	admin	c8370de7-405e-4f4d-9264-d1a73...	2025-07-16 17:23:28.751986+00
d1d2c09b-835a-460b-9857-7e7094b7ce27	admin	64494f43-3a38-4b16-a04c-5e8e8...	2025-07-16 17:23:28.751986+00
d82447fe-73a0-45de-8187-a19c604fb66d	admin	49eeb283-a58c-43e2-880a-9326...	2025-07-16 17:23:28.751986+00
fb6fda40-e0f5-4432-a843-11e173e5c9b2	admin	be231eb9-adbe-4813-9686-b8ad...	2025-07-16 17:23:28.751986+00

Filter | Sort | Insert | Auth policy | Role postgres

id uuid	permission_name text	description text	category text	created_at timestamptz
0b4cbebc-9d22-4e4d-85c3-224efe3edf54	export_data	Export data as CSV/PDF	data_management	2025-07-16 17:23:16.86453+00
3cbc13b3-e341-4845-8977-3dee73f5e5c1	manage_volunteers	Add, edit, and delete volunteer records	volunteer_management	2025-07-16 17:23:16.86453+00
49eeb283-a58c-43e2-880a-93269a99ce6	import_data	Import data via CSV upload	data_management	2025-07-16 17:23:16.86453+00
64494f43-3a38-4b16-a04c-5e8e888dd05	view_reports	Access monthly and annual reports	reporting	2025-07-16 17:23:16.86453+00
be231eb9-adbe-4813-9686-b8ad0913d97	manage_deliveries	Assign and track deliveries	delivery_management	2025-07-16 17:23:16.86453+00
c8370de7-405e-4f4d-9264-d1a73225a047	system_settings	Modify system configuration	system_management	2025-07-16 17:23:16.86453+00
c8831f72-023f-4e0b-a4dd-8c001ae28236	manage_seniors	Add, edit, and delete senior records	senior_management	2025-07-16 17:23:16.86453+00
eb4a4212-2741-484a-aadc-c897835f0989	manage_admin_users	Create and manage other admin users	admin_management	2025-07-16 17:23:16.86453+00

total_seniors int8	total_volunteers int8	monthly_deliveries int8	pending_deliveries int8	total_admins int8
36	1	0	0	1

Filter | Sort | Insert

id uuid	admin_id uuid	action text	target_table text	target_id uuid	old_values jsonb	new_values jsonb	ip_address inet	user_agent text	created_at timestamptz
03b32ff-87f1-4f53-a283-d7d83bf5fb8	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	seniors	650e8400-e29b-41d4-a716-446655440(	{"id":"650e8400-e29b-41d4-a716-446655-	NULL	NULL	NULL	2025-07-23 23:31:45.889335+0(
9b9b6187-ed67-4bcb-b5ea-dde48b46835	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	seniors	650e8400-e29b-41d4-a716-446655440(	{"id":"650e8400-e29b-41d4-a716-446655-	NULL	NULL	NULL	2025-07-23 23:31:42.380017+0(
167ada54-7915c-4165-89a3-6acbcf6268b	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	seniors	ca387e39-e30f-42b4-a133-887b303be527	{"id":"ca387e39-e30f-42b4-a133-887b303	NULL	NULL	NULL	2025-07-23 23:44:15.069924+0(
17115d29-0074-4747-912a-9c3c41e5e786	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	seniors	650e8400-e29b-41d4-a716-446655440(	{"id":"650e8400-e29b-41d4-a716-446655-	NULL	NULL	NULL	2025-07-23 23:31:48.006938+0(
1bf47a17-5a14-40c8-a62a-8d4b5af337c	2e4c6b4a-d2c1-40a0-8670-7dc7d...	INSERT	seniors	f4fdefde-fadc-4e8b-ba03-24cc8bea549a	NULL	{"id":"f4fdefde-fadc-4e8b-ba03-24cc8bea	NULL	NULL	2025-07-31 06:21:51.164202+00
1cad2945-1c3d-407f-83b4-4eae06d1a139	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	seniors	f4fdefde-fadc-4e8b-ba03-24cc8bea549a	{"id":"f4fdefde-fadc-4e8b-ba03-24cc8bea	NULL	NULL	NULL	2025-07-31 06:22:17.801949+0(
21b2454e-14cd-4adf-8ecc-6e43bb8b0b1f	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	seniors	7ffde501-39d5-4598-afcc-c080d9f1462a4	{"id":"7ffde501-39d5-4598-afcc-c080d9f14	NULL	NULL	NULL	2025-07-24 06:35:41.561737+00
63a7077a-677e-426e-b939-46487b12d59	2e4c6b4a-d2c1-40a0-8670-7dc7d...	INSERT	seniors	ca387e39-e30f-42b4-a133-887b303be527	NULL	{"id":"ca387e39-e30f-42b4-a133-887b303	NULL	NULL	2025-07-23 23:10:03.768767+0(
75c9c01d-952b-4cde-93c2-2b3ee6134f77	2e4c6b4a-d2c1-40a0-8670-7dc7d...	INSERT	seniors	7ffde501-39d5-4598-afcc-c080d9f1462a4	NULL	{"id":"7ffde501-39d5-4598-afcc-c080d9f14	NULL	NULL	2025-07-23 23:46:45.044508+(
85457830-1aff-4ef8-9504-0d8138b2a4ce	2e4c6b4a-d2c1-40a0-8670-7dc7d...	UPDATE	seniors	650e8400-e29b-41d4-a716-446655440(	NULL	{"id":"650e8400-e29b-41d4-a716-446655-	NULL	NULL	2025-07-23 22:46:20.419542+0(
8c2e431b-45ea-47fe-baee-fa350062b9c	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	seniors	650e8400-e29b-41d4-a716-446655440(	{"id":"650e8400-e29b-41d4-a716-446655-	NULL	NULL	NULL	2025-07-23 23:31:40.739039+0(
90a4ce5f-19ae-4f92-bc7b-e18b347e4347	2e4c6b4a-d2c1-40a0-8670-7dc7d...	UPDATE	seniors	650e8400-e29b-41d4-a716-446655440(	NULL	{"id":"650e8400-e29b-41d4-a716-446655-	NULL	NULL	2025-07-23 22:46:16.677767+0(
e269bc62-767c-465e-867c-a6c0bdfb50a	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	volunteers	3af730d8-3250-41d2-b7b3-939b962dbc7f	{"id":"3af730d8-3250-41d2-b7b3-939b962	NULL	NULL	NULL	2025-07-27 18:19:27.864101+00
b2715eab-67c2-400f-809b-797c7cdc09a94	2e4c6b4a-d2c1-40a0-8670-7dc7d...	UPDATE	seniors	650e8400-e29b-41d4-a716-446655440(	NULL	{"id":"650e8400-e29b-41d4-a716-446655-	NULL	NULL	2025-07-23 22:46:22.279354+(
cce8d011-45c3-4c46-9fa3-0115d3a92fdf	2e4c6b4a-d2c1-40a0-8670-7dc7d...	DELETE	seniors	650e8400-e29b-41d4-a716-446655440(	{"id":"650e8400-e29b-41d4-a716-446655-	NULL	NULL	NULL	2025-07-23 23:31:43.914787+0(
f767c180-c3db-486d-adab-6172fa1c8cd9	2e4c6b4a-d2c1-40a0-8670-7dc7d...	INSERT	volunteers	2e4c6b4a-d2c1-40a0-8670-7dc7d7f4c440	NULL	{"id":"2e4c6b4a-d2c1-40a0-8670-7dc7d7f4	NULL	NULL	2025-07-19 06:30:59.007332+(