Supabase Chat Database Research

Realtime Concepts and Best Practices

Core Extensions

Supabase Realtime includes 3 core extensions:

1. **Broadcast**: Sends rapid, ephemeral messages to other connected clients. Used for tracking mouse movements, chat messages, etc.
2. **Presence**: Sends user state between connected clients. Shows "online" status, automatically removes state when user disconnects.
3. **Postgres Changes**: Receives database changes in real-time.

Key Recommendations

- **Use Broadcast by default** for chat applications
- Use "Broadcast from Database" specifically as it allows better scaling compared to Postgres Changes
- Presence is computationally heavy - use sparingly and throttle changes
- Channels are like chatrooms - participants can see who's online and send/receive messages

Authorization

- Authorization done via RLS policies against `realtime.messages` table
- Determines if user can connect to Channel and send messages
- Channels are public by default, need to explicitly set private channels

Channel Structure

- Channel is the basic building block

- Define a unique `topic` that references a channel
- Clients can bi-directionally send and receive messages over a Channel
- Example: `supabase.channel('room-one')`

Official Supabase Tutorial Schema

From the official Flutter chat tutorial, here's the recommended database schema:

Tables Structure

Profiles Table:

SQL

```
create table if not exists public.profiles (
    id uuid references auth.users on delete cascade not null primary key,
    username varchar(24) not null unique,
    created_at timestamp with time zone default timezone('utc'::text, now())
not null,

    -- username should be 3 to 24 characters long containing only a-z, 0-9,
and underscore
    constraint username_validation check (username ~* '^[a-z0-9_]{3,24}$')
);
comment on table public.profiles is 'Holds all of user profiles information';
```

Messages Table:

SQL

```
create table if not exists public.messages (
    id uuid not null primary key default gen_random_uuid(),
    profile_id uuid default auth.uid() references public.profiles(id) on
delete cascade not null,
    content varchar(500) not null,
    created_at timestamp with time zone default timezone('utc'::text, now())
not null
);
comment on table public.messages is 'Holds individual messages sent by
users';
```

Key Design Decisions

1. **UUID Primary Keys**: Both tables use UUID for primary keys
2. **Foreign Key Relationship**: Messages reference profiles via profile_id
3. **Auth Integration**: Profiles table references auth.users, messages default to auth.uid()
4. **Content Limits**: Username limited to 24 chars, message content to 500 chars
5. **Timestamps**: Both tables have created_at with timezone support
6. **Username Validation**: Regex constraint for username format
7. **Cascade Deletes**: When user/profile deleted, related data is cleaned up

Realtime Configuration

- Enable realtime on messages table for live chat updates
- Use Supabase's built-in realtime features for instant message delivery

Current Project State Analysis

User's Supabase Project: "N8N JARVIS"

- **Project Status**: Fresh project with no custom tables created
- **Current Tables**: 0 user tables (only system postgres tables exist)
- **Project URL**: https://vxnhltixxjvfhenepeyl.supabase.co
- **Environment**: Production (main branch)
- **Ready for**: Table creation and schema setup

Project Capabilities Available

- Table Editor (GUI-based table creation)
- SQL Editor (for custom SQL execution)

- Authentication system (ready for user management)
- Realtime features (ready for chat functionality)
- Storage (if needed for file attachments)

Next Steps

- Design optimal chat database schema
- Create tables using either Table Editor or SQL Editor
- Configure Row Level Security (RLS)
- Enable realtime on messages table
- Set up proper indexes for performance