Phase 1: Conversation History - Migration Guide

Overview

This guide contains the database migration required to implement Phase 1 of conversation history functionality. The migration creates proper `conversations` and `messages` tables with full multi-tenant support and Row Level Security (RLS) policies.

Database Schema Changes

The migration performs the following changes:

1. **Renames `chat_sessions` to `conversations`** - Better semantic clarity
2. **Creates `messages` table** - Individual message storage for better querying and performance
3. **Migrates existing data** - Preserves all existing chat sessions and messages
4. **Updates indexes** - Optimized performance indexes for conversations and messages
5. **Updates RLS policies** - Maintains multi-tenant security

Manual Migration Steps

Option 1: Supabase Dashboard (Recommended)

1. Open your Supabase project dashboard
2. Navigate to **SQL Editor**
3. Create a new query
4. Copy and paste the contents of `/supabase/migrations/002_conversation_history.sql`
5. Execute the query

Option 2: Supabase CLI

```
# If you have Supabase CLI installed
supabase db push
```

Option 3: Direct PostgreSQL Connection

If you have direct database access:

```
psql -h your-database-host -U postgres -d your-database-name -f supabase/migrations/
002_conversation_history.sql
```

New Database Schema

Conversations Table

```
conversations (
  id UUID PRIMARY KEY,
  tenant_id UUID NOT NULL REFERENCES tenants(id),
  user_id UUID NOT NULL REFERENCES auth.users(id),
  title VARCHAR(255) NOT NULL DEFAULT 'New Conversation',
  metadata JSONB DEFAULT '{}',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
)
```

Messages Table

```
messages (
  id UUID PRIMARY KEY,
  conversation_id UUID NOT NULL REFERENCES conversations(id),
  role VARCHAR(20) CHECK (role IN ('user', 'assistant', 'system')),
  content TEXT NOT NULL,
  metadata JSONB DEFAULT '{}',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
)
```

Features Implemented

✅ **Database-Backed Conversations** - All conversations are now stored in the database
✅ **Individual Message Storage** - Each message is stored separately for better querying
✅ **Multi-Tenant Support** - Full isolation between different tenants
✅ **Row Level Security** - Proper RLS policies for data security
✅ **API Endpoints** - RESTful endpoints for conversation management
✅ **Real-time Updates** - UI updates in real-time as conversations are modified
✅ **Backward Compatibility** - Works with existing chat interface
✅ **Performance Optimized** - Proper indexing for fast queries

API Endpoints

Conversations

- `GET /api/conversations` - List user's conversations
- `POST /api/conversations` - Create new conversation
- `GET /api/conversations/[id]` - Get conversation with messages
- `PATCH /api/conversations/[id]` - Update conversation metadata
- `DELETE /api/conversations/[id]` - Delete conversation

Chat (Enhanced)

- `POST /api/chat` - Send message with automatic conversation persistence

Testing the Implementation

After running the migration:

1. **Start the development server**: `npm run dev`
2. **Open the application** in your browser
3. **Start a new conversation** - It will automatically create a database-backed conversation
4. **Send messages** - All messages are automatically saved to the database
5. **Refresh the page** - Your conversation history will persist
6. **Check conversation sidebar** - Lists all your previous conversations
7. **Switch between conversations** - Click any conversation to load its message history

Troubleshooting

Migration Issues

- Ensure your database has the `uuid-ossp` extension enabled
- Check that your service role key has sufficient permissions
- Verify all existing tables (tenants, tenant_users) exist

Authentication Issues

- Ensure you're logged in with proper tenant association
- Check that tenant_users table has records for your user
- Verify RLS policies are working correctly

Performance Issues

- The migration includes performance-optimized indexes
- Large existing datasets may take time to migrate
- Consider running during low-traffic periods

Data Migration Safety

The migration is designed to:
- ✅ Preserve all existing data
- ✅ Maintain referential integrity
- ✅ Handle edge cases (empty messages arrays, null values)
- ✅ Continue working if some steps fail
- ✅ Not break existing functionality

Next Steps

After successful migration:
1. Test all conversation functionality
2. Verify multi-tenant isolation
3. Check performance with your data volume
4. Consider implementing Phase 2 features (conversation management, search, export)

Note: This migration is part of Phase 1 implementation. The application will work with existing `chat_sessions` table but will have limited functionality until the migration is completed.