# **PART 4: Database Engineering**

## **Database Design with Supabase**

### **Technologies**

* Database: PostgreSQL (provided by Supabase)
* Authentication: Supabase Auth
* Realtime: Supabase Realtime for live updates
* Storage: Supabase Storage for user-generated content

### **Database Schema Design**

#### **Sample Schema for LLM-powered Application**

// Users and Authentication

TABLE users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

email TEXT UNIQUE NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

last\_sign\_in TIMESTAMP WITH TIME ZONE,

metadata JSONB

);

// LLM Conversations

TABLE conversations (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES users(id),

title TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

metadata JSONB

);

// Individual Messages in Conversations

TABLE messages (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

conversation\_id UUID REFERENCES conversations(id) ON DELETE CASCADE,

role TEXT NOT NULL CHECK (role IN ('user', 'assistant', 'system')),

content TEXT NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

tokens\_used INTEGER,

model\_used TEXT

);

// User Preferences

TABLE user\_preferences (

user\_id UUID PRIMARY KEY REFERENCES users(id) ON DELETE CASCADE,

default\_model TEXT,

theme TEXT DEFAULT 'light',

settings JSONB

);

// LLM Model Performance Metrics

TABLE model\_metrics (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

model\_name TEXT NOT NULL,

timestamp TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

latency\_ms INTEGER NOT NULL,

tokens\_input INTEGER NOT NULL,

tokens\_output INTEGER NOT NULL,

success BOOLEAN NOT NULL,

error\_message TEXT

);

// Vector Embeddings

TABLE documents (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

content TEXT NOT NULL,

metadata JSONB,

user\_id UUID REFERENCES users(id),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

TABLE document\_embeddings (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

document\_id UUID REFERENCES documents(id) ON DELETE CASCADE,

embedding VECTOR(1536), -- Using pgvector extension

chunk\_index INTEGER,

chunk\_content TEXT

);

// Create index for vector similarity search

CREATE INDEX ON document\_embeddings USING ivfflat (embedding vector\_cosine\_ops);

#### **Schema Design Tools**

* Supabase Schema Editor: Visual interface for schema management
* dbdiagram.io: Create and visualize database schemas
* DrawSQL: Collaborative SQL schema visualization tool
* Schema Migration Files: Version-controlled SQL scripts
* PostgreSQL ERD Tools: pgAdmin, DBeaver with ERD visualization

### **Database Access Patterns**

1. Supabase Client Setup

// supabase.ts

import { createClient } from '@supabase/supabase-js';

const supabaseUrl = process.env.SUPABASE\_URL!;

const supabaseKey = process.env.SUPABASE\_ANON\_KEY!;

export const supabase = createClient(supabaseUrl, supabaseKey);

1. Repository Pattern

// repositories/conversationRepository.ts

import { supabase } from '../supabase';

import { Conversation, Message } from '../types';

export const ConversationRepository = {

async getByUserId(userId: string): Promise<Conversation[]> {

const { data, error } = await supabase

.from('conversations')

.select('\*')

.eq('user\_id', userId)

.order('updated\_at', { ascending: false });

if (error) throw error;

return data || [];

},

async getWithMessages(conversationId: string): Promise<Conversation & { messages: Message[] }> {

// Get conversation

const { data: conversation, error: convError } = await supabase

.from('conversations')

.select('\*')

.eq('id', conversationId)

.single();

if (convError) throw convError;

// Get messages

const { data: messages, error: msgError } = await supabase

.from('messages')

.select('\*')

.eq('conversation\_id', conversationId)

.order('created\_at', { ascending: true });

if (msgError) throw msgError;

return {

...conversation,

messages: messages || []

};

},

// Additional methods...

};

1. Row-Level Security

-- Enable RLS

ALTER TABLE conversations ENABLE ROW LEVEL SECURITY;

-- Create policies

CREATE POLICY "Users can view their own conversations"

ON conversations

FOR SELECT

USING (auth.uid() = user\_id);

CREATE POLICY "Users can insert their own conversations"

ON conversations

FOR INSERT

WITH CHECK (auth.uid() = user\_id);

CREATE POLICY "Users can update their own conversations"

ON conversations

FOR UPDATE

USING (auth.uid() = user\_id);