# **Database Design & ER Diagrams**

## **Chat Application with Microsoft Authentication**

### **Document Information**

* **Project Name**: MSAL Chat Application - Database Design
* **Version**: 1.0.0
* **Created**: June 30, 2025
* **Database Type**: Relational Database (PostgreSQL/SQL Server recommended)
* **ORM Recommendation**: Prisma with TypeScript

## **1. Database Overview**

### **1.1 Purpose**

This database design supports a scalable chat application with Microsoft Azure AD authentication, persistent message storage, conversation management, and user analytics. The current application is stateless but this design enables future enhancements.

### **1.2 Design Principles**

* **Normalization**: 3NF compliance to reduce data redundancy
* **Scalability**: Designed for horizontal scaling and partitioning
* **Security**: Row-level security and audit trails
* **Performance**: Optimized indexes and query patterns
* **GDPR Compliance**: User data management and deletion capabilities

### **1.3 Technology Stack**

* **Primary Database**: PostgreSQL 15+ or SQL Server 2022+
* **ORM**: Prisma (recommended) or TypeORM
* **Migration Tool**: Database-native migrations
* **Backup Strategy**: Point-in-time recovery with daily snapshots

## **2. Entity Relationship Diagram**

### **2.1 High-Level ER Diagram**

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ USERS │ │ CONVERSATIONS │ │ MESSAGES │  
│─────────────────│ │─────────────────│ │─────────────────│  
│ PK: user\_id │◄────────┤ PK: conv\_id │◄────────┤ PK: message\_id │  
│ azure\_id │ 1:N │ FK: user\_id │ 1:N │ FK: conv\_id │  
│ email │ │ title │ │ FK: user\_id │  
│ display\_name │ │ created\_at │ │ content │  
│ tenant\_id │ │ updated\_at │ │ role │  
│ created\_at │ │ is\_archived │ │ token\_count │  
│ last\_login │ │ total\_msgs │ │ created\_at │  
│ is\_active │ └─────────────────┘ │ message\_type │  
└─────────────────┘ │ metadata │  
 │ └─────────────────┘  
 │ │  
 │ ┌─────────────────┐ │  
 │ │ USER\_SESSIONS │ │  
 │ │─────────────────│ │  
 └──────────────────┤ PK: session\_id │ │  
 1:N │ FK: user\_id │ │  
 │ access\_token │ │  
 │ refresh\_token│ │  
 │ expires\_at │ │  
 │ created\_at │ │  
 │ last\_used │ │  
 │ ip\_address │ │  
 │ user\_agent │ │  
 └─────────────────┘ │  
 │  
┌─────────────────┐ ┌─────────────────┐ │  
│ MESSAGE\_TAGS │ │ CONVERSATION\_ │ │  
│─────────────────│ │ ANALYTICS │ │  
│ PK: tag\_id │ │─────────────────│ │  
│ tag\_name │ │ PK: analytics\_id│ │  
│ color\_code │ │ FK: conv\_id │ │  
│ created\_by │ │ total\_tokens │ │  
└─────────────────┘ │ avg\_response │ │  
 │ │ sentiment │ │  
 │ │ category │ │  
 │ │ updated\_at │ │  
 │ └─────────────────┘ │  
 │ │  
 │ ┌─────────────────┐ │  
 │ │ MESSAGE\_TAG\_REL │ │  
 │ │─────────────────│ │  
 └──┤ PK: rel\_id │ │  
 N:M │ FK: message\_id │◄─────────────────────────────────┘  
 │ FK: tag\_id │  
 │ created\_at │  
 └─────────────────┘  
  
┌─────────────────┐ ┌─────────────────┐  
│ API\_REQUESTS │ │ AUDIT\_LOGS │  
│─────────────────│ │─────────────────│  
│ PK: request\_id │ │ PK: log\_id │  
│ FK: user\_id │ │ FK: user\_id │  
│ FK: message\_id │ │ action\_type │  
│ endpoint │ │ table\_name │  
│ status\_code │ │ record\_id │  
│ response\_time│ │ old\_values │  
│ tokens\_used │ │ new\_values │  
│ created\_at │ │ ip\_address │  
│ error\_details│ │ user\_agent │  
└─────────────────┘ │ created\_at │  
 └─────────────────┘

### **2.2 Detailed Entity Relationships**

#### **2.2.1 Core Relationships**

* **Users (1) → Conversations (N)**: One user can have multiple conversations
* **Conversations (1) → Messages (N)**: One conversation contains multiple messages
* **Users (1) → Messages (N)**: One user can create multiple messages
* **Messages (N) → Message\_Tags (M)**: Many-to-many relationship via Message\_Tag\_Rel
* **Users (1) → User\_Sessions (N)**: One user can have multiple active sessions

#### **2.2.2 Analytics Relationships**

* **Conversations (1) → Conversation\_Analytics (1)**: One-to-one analytics per conversation
* **Users (1) → API\_Requests (N)**: Track user API usage
* **Messages (1) → API\_Requests (N)**: Link requests to specific messages

## **3. Table Definitions**

### **3.1 Core Tables**

#### **3.1.1 Users Table**

CREATE TABLE users (  
 user\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 azure\_id VARCHAR(255) UNIQUE NOT NULL, -- Azure AD Object ID  
 email VARCHAR(320) UNIQUE NOT NULL,  
 display\_name VARCHAR(255) NOT NULL,  
 tenant\_id VARCHAR(255) NOT NULL,  
 preferred\_username VARCHAR(255),  
 job\_title VARCHAR(255),  
 department VARCHAR(255),  
 profile\_picture\_url TEXT,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 last\_login TIMESTAMP WITH TIME ZONE,  
 is\_active BOOLEAN DEFAULT TRUE,  
 settings JSONB DEFAULT '{}',  
   
 -- Indexes  
 CONSTRAINT users\_email\_check CHECK (email ~\* '^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$')  
);  
  
-- Indexes for Users  
CREATE INDEX idx\_users\_azure\_id ON users(azure\_id);  
CREATE INDEX idx\_users\_email ON users(email);  
CREATE INDEX idx\_users\_tenant\_id ON users(tenant\_id);  
CREATE INDEX idx\_users\_last\_login ON users(last\_login);  
CREATE INDEX idx\_users\_created\_at ON users(created\_at);

**3.1.2 Conversations Table**

CREATE TABLE conversations (  
 conv\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID NOT NULL REFERENCES users(user\_id) ON DELETE CASCADE,  
 title VARCHAR(255) NOT NULL DEFAULT 'New Conversation',  
 description TEXT,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 is\_archived BOOLEAN DEFAULT FALSE,  
 is\_deleted BOOLEAN DEFAULT FALSE,  
 total\_messages INTEGER DEFAULT 0,  
 total\_tokens INTEGER DEFAULT 0,  
 conversation\_type VARCHAR(50) DEFAULT 'chat', -- 'chat', 'analysis', 'brainstorm'  
 metadata JSONB DEFAULT '{}',  
   
 -- Constraints  
 CONSTRAINT conversations\_title\_length CHECK (LENGTH(title) >= 1),  
 CONSTRAINT conversations\_type\_check CHECK (conversation\_type IN ('chat', 'analysis', 'brainstorm', 'support'))  
);  
  
-- Indexes for Conversations  
CREATE INDEX idx\_conversations\_user\_id ON conversations(user\_id);  
CREATE INDEX idx\_conversations\_created\_at ON conversations(created\_at);  
CREATE INDEX idx\_conversations\_updated\_at ON conversations(updated\_at);  
CREATE INDEX idx\_conversations\_archived ON conversations(is\_archived, user\_id);  
CREATE INDEX idx\_conversations\_type ON conversations(conversation\_type);

**3.1.3 Messages Table**

CREATE TABLE messages (message\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 conv\_id UUID NOT NULL REFERENCES conversations(conv\_id) ON DELETE CASCADE,  
 user\_id UUID NOT NULL REFERENCES users(user\_id) ON DELETE CASCADE,  
 content TEXT NOT NULL,  
 role VARCHAR(20) NOT NULL, -- 'user', 'assistant', 'system'  
 message\_type VARCHAR(50) DEFAULT 'text', -- 'text', 'image', 'file', 'code'  
 token\_count INTEGER DEFAULT 0,  
 model\_used VARCHAR(100), -- 'gpt-35-turbo-16k', 'gpt-4', etc.  
 response\_time\_ms INTEGER, -- API response time  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 edited\_at TIMESTAMP WITH TIME ZONE,  
 is\_deleted BOOLEAN DEFAULT FALSE,  
 parent\_message\_id UUID REFERENCES messages(message\_id), -- For threaded conversations  
 metadata JSONB DEFAULT '{}', -- Additional context, files, etc.  
   
 -- Constraints  
 CONSTRAINT messages\_role\_check CHECK (role IN ('user', 'assistant', 'system')),  
 CONSTRAINT messages\_type\_check CHECK (message\_type IN ('text', 'image', 'file', 'code', 'error')),  
 CONSTRAINT messages\_content\_length CHECK (LENGTH(content) >= 1),  
 CONSTRAINT messages\_token\_count\_positive CHECK (token\_count >= 0)  
);  
  
-- Indexes for Messages  
CREATE INDEX idx\_messages\_conv\_id ON messages(conv\_id);  
CREATE INDEX idx\_messages\_user\_id ON messages(user\_id);  
CREATE INDEX idx\_messages\_created\_at ON messages(created\_at);  
CREATE INDEX idx\_messages\_role ON messages(role);  
CREATE INDEX idx\_messages\_type ON messages(message\_type);  
CREATE INDEX idx\_messages\_parent ON messages(parent\_message\_id);  
CREATE INDEX idx\_messages\_content\_search ON messages USING gin(to\_tsvector('english', content));

**3.2 Authentication & Session Tables**

**3.2.1 User Sessions Table**

CREATE TABLE user\_sessions (  
 session\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID NOT NULL REFERENCES users(user\_id) ON DELETE CASCADE,  
 access\_token\_hash VARCHAR(255) NOT NULL, -- Hashed token for security  
 refresh\_token\_hash VARCHAR(255),  
 token\_scope TEXT NOT NULL,  
 expires\_at TIMESTAMP WITH TIME ZONE NOT NULL,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 last\_used TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 ip\_address INET,  
 user\_agent TEXT,  
 device\_info JSONB DEFAULT '{}',  
 is\_active BOOLEAN DEFAULT TRUE,  
 revoked\_at TIMESTAMP WITH TIME ZONE,  
 revoke\_reason VARCHAR(100),  
   
 -- Constraints  
 CONSTRAINT sessions\_expires\_future CHECK (expires\_at > created\_at)  
);  
  
-- Indexes for User Sessions  
CREATE INDEX idx\_sessions\_user\_id ON user\_sessions(user\_id);  
CREATE INDEX idx\_sessions\_expires\_at ON user\_sessions(expires\_at);  
CREATE INDEX idx\_sessions\_active ON user\_sessions(is\_active, user\_id);  
CREATE INDEX idx\_sessions\_last\_used ON user\_sessions(last\_used);  
 **3.3 Analytics & Monitoring Tables**

#### **3.3.1 Conversation Analytics Table**

CREATE TABLE conversation\_analytics (  
 analytics\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 conv\_id UUID UNIQUE NOT NULL REFERENCES conversations(conv\_id) ON DELETE CASCADE,  
 total\_tokens INTEGER DEFAULT 0,  
 average\_response\_time\_ms FLOAT DEFAULT 0,  
 message\_count INTEGER DEFAULT 0,  
 user\_satisfaction\_score FLOAT, -- 1-5 rating  
 sentiment\_score FLOAT, -- -1 to 1 sentiment analysis  
 category VARCHAR(100), -- Auto-categorized conversation type  
 language\_detected VARCHAR(10) DEFAULT 'en',  
 complexity\_score FLOAT, -- 1-10 complexity rating  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
   
 -- Constraints  
 CONSTRAINT analytics\_satisfaction\_range CHECK (user\_satisfaction\_score BETWEEN 1 AND 5),  
 CONSTRAINT analytics\_sentiment\_range CHECK (sentiment\_score BETWEEN -1 AND 1),  
 CONSTRAINT analytics\_complexity\_range CHECK (complexity\_score BETWEEN 1 AND 10)  
);  
  
-- Indexes for Analytics  
CREATE INDEX idx\_analytics\_conv\_id ON conversation\_analytics(conv\_id);  
CREATE INDEX idx\_analytics\_category ON conversation\_analytics(category);  
CREATE INDEX idx\_analytics\_sentiment ON conversation\_analytics(sentiment\_score);  
CREATE INDEX idx\_analytics\_updated\_at ON conversation\_analytics(updated\_at);

#### **3.3.2 API Requests Table**

CREATE TABLE api\_requests (  
 request\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID REFERENCES users(user\_id) ON DELETE SET NULL,  
 message\_id UUID REFERENCES messages(message\_id) ON DELETE SET NULL,  
 endpoint VARCHAR(255) NOT NULL,  
 http\_method VARCHAR(10) NOT NULL,  
 status\_code INTEGER NOT NULL,  
 response\_time\_ms INTEGER NOT NULL,  
 tokens\_used INTEGER DEFAULT 0,  
 model\_used VARCHAR(100),  
 request\_size\_bytes INTEGER,  
 response\_size\_bytes INTEGER,  
 ip\_address INET,  
 user\_agent TEXT,  
 error\_message TEXT,  
 error\_code VARCHAR(50),  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
   
 -- Constraints  
 CONSTRAINT api\_requests\_status\_code\_valid CHECK (status\_code BETWEEN 100 AND 599),  
 CONSTRAINT api\_requests\_response\_time\_positive CHECK (response\_time\_ms >= 0)  
);  
  
-- Indexes for API Requests  
CREATE INDEX idx\_api\_requests\_user\_id ON api\_requests(user\_id);  
CREATE INDEX idx\_api\_requests\_created\_at ON api\_requests(created\_at);  
CREATE INDEX idx\_api\_requests\_status\_code ON api\_requests(status\_code);  
CREATE INDEX idx\_api\_requests\_endpoint ON api\_requests(endpoint);  
CREATE INDEX idx\_api\_requests\_error\_code ON api\_requests(error\_code) WHERE error\_code IS NOT NULL;

### **3.4 Tagging & Organization Tables**

#### **3.4.1 Message Tags Table**

CREATE TABLE message\_tags (  
 tag\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 tag\_name VARCHAR(100) UNIQUE NOT NULL,  
 description TEXT,  
 color\_code VARCHAR(7) DEFAULT '#3B82F6', -- Hex color code  
 is\_system\_tag BOOLEAN DEFAULT FALSE, -- System vs user-created tags  
 created\_by UUID REFERENCES users(user\_id) ON DELETE SET NULL,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 usage\_count INTEGER DEFAULT 0,  
   
 -- Constraints  
 CONSTRAINT tags\_name\_length CHECK (LENGTH(tag\_name) BETWEEN 1 AND 100),  
 CONSTRAINT tags\_color\_format CHECK (color\_code ~\* '^#[0-9A-Fa-f]{6}$')  
);  
  
-- Indexes for Message Tags  
CREATE INDEX idx\_tags\_name ON message\_tags(tag\_name);  
CREATE INDEX idx\_tags\_created\_by ON message\_tags(created\_by);  
CREATE INDEX idx\_tags\_system ON message\_tags(is\_system\_tag);

#### **3.4.2 Message Tag Relations Table**

CREATE TABLE message\_tag\_relations (  
 rel\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 message\_id UUID NOT NULL REFERENCES messages(message\_id) ON DELETE CASCADE,  
 tag\_id UUID NOT NULL REFERENCES message\_tags(tag\_id) ON DELETE CASCADE,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 created\_by UUID REFERENCES users(user\_id) ON DELETE SET NULL,  
   
 -- Prevent duplicate tag assignments  
 UNIQUE(message\_id, tag\_id)  
);  
  
-- Indexes for Tag Relations  
CREATE INDEX idx\_tag\_relations\_message\_id ON message\_tag\_relations(message\_id);  
CREATE INDEX idx\_tag\_relations\_tag\_id ON message\_tag\_relations(tag\_id);  
CREATE INDEX idx\_tag\_relations\_created\_at ON message\_tag\_relations(created\_at);

### **3.5 Audit & Security Tables**

#### **3.5.1 Audit Logs Table**

CREATE TABLE audit\_logs (  
 log\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 user\_id UUID REFERENCES users(user\_id) ON DELETE SET NULL,  
 action\_type VARCHAR(50) NOT NULL, -- 'CREATE', 'UPDATE', 'DELETE', 'LOGIN', 'LOGOUT'  
 table\_name VARCHAR(100) NOT NULL,  
 record\_id UUID NOT NULL,  
 old\_values JSONB,  
 new\_values JSONB,  
 ip\_address INET,  
 user\_agent TEXT,  
 session\_id UUID REFERENCES user\_sessions(session\_id) ON DELETE SET NULL,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
   
 -- Constraints  
 CONSTRAINT audit\_action\_type\_check CHECK (action\_type IN ('CREATE', 'UPDATE', 'DELETE', 'LOGIN', 'LOGOUT', 'VIEW'))  
);  
  
-- Indexes for Audit Logs  
CREATE INDEX idx\_audit\_user\_id ON audit\_logs(user\_id);  
CREATE INDEX idx\_audit\_action\_type ON audit\_logs(action\_type);  
CREATE INDEX idx\_audit\_table\_name ON audit\_logs(table\_name);  
CREATE INDEX idx\_audit\_record\_id ON audit\_logs(record\_id);  
CREATE INDEX idx\_audit\_created\_at ON audit\_logs(created\_at);

## **4. Database Views**

### **4.1 User Dashboard View**

CREATE VIEW user\_dashboard AS  
SELECT   
 u.user\_id,  
 u.display\_name,  
 u.email,  
 COUNT(DISTINCT c.conv\_id) AS total\_conversations,  
 COUNT(DISTINCT m.message\_id) AS total\_messages,  
 COALESCE(SUM(ca.total\_tokens), 0) AS total\_tokens\_used,  
 MAX(c.updated\_at) AS last\_conversation\_date,  
 COUNT(DISTINCT CASE WHEN c.created\_at >= NOW() - INTERVAL '30 days' THEN c.conv\_id END) AS conversations\_last\_30\_days,  
 AVG(ca.user\_satisfaction\_score) AS avg\_satisfaction\_score  
FROM users u  
LEFT JOIN conversations c ON u.user\_id = c.user\_id AND c.is\_deleted = FALSE  
LEFT JOIN messages m ON c.conv\_id = m.conv\_id AND m.is\_deleted = FALSE  
LEFT JOIN conversation\_analytics ca ON c.conv\_id = ca.conv\_id  
WHERE u.is\_active = TRUE  
GROUP BY u.user\_id, u.display\_name, u.email;

### **4.2 Conversation Summary View**

CREATE VIEW conversation\_summary AS  
SELECT   
 c.conv\_id,  
 c.user\_id,  
 u.display\_name AS user\_name,  
 c.title,  
 c.conversation\_type,  
 c.created\_at,  
 c.updated\_at,  
 COUNT(m.message\_id) AS message\_count,  
 MAX(m.created\_at) AS last\_message\_at,  
 ca.total\_tokens,  
 ca.average\_response\_time\_ms,  
 ca.sentiment\_score,  
 ca.category,  
 ARRAY\_AGG(DISTINCT mt.tag\_name ORDER BY mt.tag\_name) FILTER (WHERE mt.tag\_name IS NOT NULL) AS tags  
FROM conversations c  
JOIN users u ON c.user\_id = u.user\_id  
LEFT JOIN messages m ON c.conv\_id = m.conv\_id AND m.is\_deleted = FALSE  
LEFT JOIN conversation\_analytics ca ON c.conv\_id = ca.conv\_id  
LEFT JOIN message\_tag\_relations mtr ON m.message\_id = mtr.message\_id  
LEFT JOIN message\_tags mt ON mtr.tag\_id = mt.tag\_id  
WHERE c.is\_deleted = FALSE  
GROUP BY c.conv\_id, c.user\_id, u.display\_name, c.title, c.conversation\_type,   
 c.created\_at, c.updated\_at, ca.total\_tokens, ca.average\_response\_time\_ms,   
 ca.sentiment\_score, ca.category;

### **4.3 API Usage Analytics View**

CREATE VIEW api\_usage\_analytics AS  
SELECT   
 DATE\_TRUNC('day', ar.created\_at) AS date,  
 u.tenant\_id,  
 COUNT(\*) AS total\_requests,  
 COUNT(DISTINCT ar.user\_id) AS unique\_users,  
 AVG(ar.response\_time\_ms) AS avg\_response\_time,  
 SUM(ar.tokens\_used) AS total\_tokens,  
 COUNT(\*) FILTER (WHERE ar.status\_code >= 400) AS error\_count,  
 COUNT(\*) FILTER (WHERE ar.status\_code = 200) AS success\_count  
FROM api\_requests ar  
LEFT JOIN users u ON ar.user\_id = u.user\_id  
GROUP BY DATE\_TRUNC('day', ar.created\_at), u.tenant\_id  
ORDER BY date DESC;

## **5. Database Functions & Triggers**

### **5.1 Update Timestamp Function**

CREATE OR REPLACE FUNCTION update\_updated\_at\_column()  
RETURNS TRIGGER AS $$  
BEGIN  
 NEW.updated\_at = CURRENT\_TIMESTAMP;  
 RETURN NEW;  
END;  
$$ language 'plpgsql';  
  
-- Apply to relevant tables  
CREATE TRIGGER update\_users\_updated\_at BEFORE UPDATE ON users   
 FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();  
  
CREATE TRIGGER update\_conversations\_updated\_at BEFORE UPDATE ON conversations   
 FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();  
  
CREATE TRIGGER update\_conversation\_analytics\_updated\_at BEFORE UPDATE ON conversation\_analytics   
 FOR EACH ROW EXECUTE FUNCTION update\_updated\_at\_column();

### **5.2 Conversation Analytics Update Function**

CREATE OR REPLACE FUNCTION update\_conversation\_analytics()  
RETURNS TRIGGER AS $$  
BEGIN  
 -- Update analytics when a new message is added  
 IF TG\_OP = 'INSERT' THEN  
 INSERT INTO conversation\_analytics (conv\_id, total\_tokens, message\_count, updated\_at)  
 VALUES (NEW.conv\_id, NEW.token\_count, 1, CURRENT\_TIMESTAMP)  
 ON CONFLICT (conv\_id) DO UPDATE SET  
 total\_tokens = conversation\_analytics.total\_tokens + NEW.token\_count,  
 message\_count = conversation\_analytics.message\_count + 1,  
 updated\_at = CURRENT\_TIMESTAMP;  
   
 -- Update conversation's updated\_at and total\_messages  
 UPDATE conversations   
 SET updated\_at = CURRENT\_TIMESTAMP,  
 total\_messages = total\_messages + 1,  
 total\_tokens = total\_tokens + NEW.token\_count  
 WHERE conv\_id = NEW.conv\_id;  
   
 RETURN NEW;  
 END IF;  
   
 RETURN NULL;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER trigger\_update\_conversation\_analytics  
 AFTER INSERT ON messages  
 FOR EACH ROW EXECUTE FUNCTION update\_conversation\_analytics();

### **5.3 Audit Logging Function**

CREATE OR REPLACE FUNCTION audit\_table\_changes()  
RETURNS TRIGGER AS $$  
DECLARE  
 user\_id\_val UUID;  
BEGIN  
 -- Extract user\_id from the record being modified  
 user\_id\_val := COALESCE(NEW.user\_id, OLD.user\_id);  
   
 IF TG\_OP = 'INSERT' THEN  
 INSERT INTO audit\_logs (user\_id, action\_type, table\_name, record\_id, new\_values)  
 VALUES (user\_id\_val, 'CREATE', TG\_TABLE\_NAME, NEW.id, to\_jsonb(NEW));  
 RETURN NEW;  
 ELSIF TG\_OP = 'UPDATE' THEN  
 INSERT INTO audit\_logs (user\_id, action\_type, table\_name, record\_id, old\_values, new\_values)  
 VALUES (user\_id\_val, 'UPDATE', TG\_TABLE\_NAME, NEW.id, to\_jsonb(OLD), to\_jsonb(NEW));  
 RETURN NEW;  
 ELSIF TG\_OP = 'DELETE' THEN  
 INSERT INTO audit\_logs (user\_id, action\_type, table\_name, record\_id, old\_values)  
 VALUES (user\_id\_val, 'DELETE', TG\_TABLE\_NAME, OLD.id, to\_jsonb(OLD));  
 RETURN OLD;  
 END IF;  
   
 RETURN NULL;  
END;  
$$ LANGUAGE plpgsql;  
  
-- Apply audit triggers to sensitive tables  
CREATE TRIGGER audit\_users\_changes  
 AFTER INSERT OR UPDATE OR DELETE ON users  
 FOR EACH ROW EXECUTE FUNCTION audit\_table\_changes();  
  
CREATE TRIGGER audit\_conversations\_changes  
 AFTER INSERT OR UPDATE OR DELETE ON conversations  
 FOR EACH ROW EXECUTE FUNCTION audit\_table\_changes();

## **6. Prisma Schema (Recommended ORM)**

### **6.1 Complete Prisma Schema**

// schema.prisma  
generator client {  
 provider = "prisma-client-js"  
}  
  
datasource db {  
 provider = "postgresql"  
 url = env("DATABASE\_URL")  
}  
  
model User {  
 id String @id @default(uuid()) @map("user\_id") @db.Uuid  
 azureId String @unique @map("azure\_id") @db.VarChar(255)  
 email String @unique @db.VarChar(320)  
 displayName String @map("display\_name") @db.VarChar(255)  
 tenantId String @map("tenant\_id") @db.VarChar(255)  
 preferredUsername String? @map("preferred\_username") @db.VarChar(255)  
 jobTitle String? @map("job\_title") @db.VarChar(255)  
 department String? @db.VarChar(255)  
 profilePictureUrl String? @map("profile\_picture\_url")  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
 updatedAt DateTime @updatedAt @map("updated\_at") @db.Timestamptz  
 lastLogin DateTime? @map("last\_login") @db.Timestamptz  
 isActive Boolean @default(true) @map("is\_active")  
 settings Json @default("{}")  
  
 // Relations  
 conversations Conversation[]  
 messages Message[]  
 userSessions UserSession[]  
 apiRequests ApiRequest[]  
 auditLogs AuditLog[]  
 createdTags MessageTag[]  
  
 @@map("users")  
}  
  
model Conversation {  
 id String @id @default(uuid()) @map("conv\_id") @db.Uuid  
 userId String @map("user\_id") @db.Uuid  
 title String @default("New Conversation") @db.VarChar(255)  
 description String?  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
 updatedAt DateTime @updatedAt @map("updated\_at") @db.Timestamptz  
 isArchived Boolean @default(false) @map("is\_archived")  
 isDeleted Boolean @default(false) @map("is\_deleted")  
 totalMessages Int @default(0) @map("total\_messages")  
 totalTokens Int @default(0) @map("total\_tokens")  
 conversationType String @default("chat") @map("conversation\_type") @db.VarChar(50)  
 metadata Json @default("{}")  
  
 // Relations  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
 messages Message[]  
 analytics ConversationAnalytics?  
  
 @@map("conversations")  
}  
  
model Message {  
 id String @id @default(uuid()) @map("message\_id") @db.Uuid  
 convId String @map("conv\_id") @db.Uuid  
 userId String @map("user\_id") @db.Uuid  
 content String  
 role String @db.VarChar(20)  
 messageType String @default("text") @map("message\_type") @db.VarChar(50)  
 tokenCount Int @default(0) @map("token\_count")  
 modelUsed String? @map("model\_used") @db.VarChar(100)  
 responseTimeMs Int? @map("response\_time\_ms")  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
 editedAt DateTime? @map("edited\_at") @db.Timestamptz  
 isDeleted Boolean @default(false) @map("is\_deleted")  
 parentMessageId String? @map("parent\_message\_id") @db.Uuid  
 metadata Json @default("{}")  
  
 // Relations  
 conversation Conversation @relation(fields: [convId], references: [id], onDelete: Cascade)  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
 parentMessage Message? @relation("MessageThread", fields: [parentMessageId], references: [id])  
 childMessages Message[] @relation("MessageThread")  
 tags MessageTagRelation[]  
 apiRequests ApiRequest[]  
  
 @@map("messages")  
}  
  
model UserSession {  
 id String @id @default(uuid()) @map("session\_id") @db.Uuid  
 userId String @map("user\_id") @db.Uuid  
 accessTokenHash String @map("access\_token\_hash") @db.VarChar(255)  
 refreshTokenHash String? @map("refresh\_token\_hash") @db.VarChar(255)  
 tokenScope String @map("token\_scope")  
 expiresAt DateTime @map("expires\_at") @db.Timestamptz  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
 lastUsed DateTime @default(now()) @map("last\_used") @db.Timestamptz  
 ipAddress String? @map("ip\_address") @db.Inet  
 userAgent String? @map("user\_agent")  
 deviceInfo Json @default("{}") @map("device\_info")  
 isActive Boolean @default(true) @map("is\_active")  
 revokedAt DateTime? @map("revoked\_at") @db.Timestamptz  
 revokeReason String? @map("revoke\_reason") @db.VarChar(100)  
  
 // Relations  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
 auditLogs AuditLog[]  
  
 @@map("user\_sessions")  
}  
  
model ConversationAnalytics {  
 id String @id @default(uuid()) @map("analytics\_id") @db.Uuid  
 convId String @unique @map("conv\_id") @db.Uuid  
 totalTokens Int @default(0) @map("total\_tokens")  
 averageResponseTimeMs Float @default(0) @map("average\_response\_time\_ms")  
 messageCount Int @default(0) @map("message\_count")  
 userSatisfactionScore Float? @map("user\_satisfaction\_score")  
 sentimentScore Float? @map("sentiment\_score")  
 category String? @db.VarChar(100)  
 languageDetected String @default("en") @map("language\_detected") @db.VarChar(10)  
 complexityScore Float? @map("complexity\_score")  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
 updatedAt DateTime @updatedAt @map("updated\_at") @db.Timestamptz  
  
 // Relations  
 conversation Conversation @relation(fields: [convId], references: [id], onDelete: Cascade)  
  
 @@map("conversation\_analytics")  
}  
  
model ApiRequest {  
 id String @id @default(uuid()) @map("request\_id") @db.Uuid  
 userId String? @map("user\_id") @db.Uuid  
 messageId String? @map("message\_id") @db.Uuid  
 endpoint String @db.VarChar(255)  
 httpMethod String @map("http\_method") @db.VarChar(10)  
 statusCode Int @map("status\_code")  
 responseTimeMs Int @map("response\_time\_ms")  
 tokensUsed Int @default(0) @map("tokens\_used")  
 modelUsed String? @map("model\_used") @db.VarChar(100)  
 requestSizeBytes Int? @map("request\_size\_bytes")  
 responseSizeBytes Int? @map("response\_size\_bytes")  
 ipAddress String? @map("ip\_address") @db.Inet  
 userAgent String? @map("user\_agent")  
 errorMessage String? @map("error\_message")  
 errorCode String? @map("error\_code") @db.VarChar(50)  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
  
 // Relations  
 user User? @relation(fields: [userId], references: [id], onDelete: SetNull)  
 message Message? @relation(fields: [messageId], references: [id], onDelete: SetNull)  
  
 @@map("api\_requests")  
}  
  
model MessageTag {  
 id String @id @default(uuid()) @map("tag\_id") @db.Uuid  
 tagName String @unique @map("tag\_name") @db.VarChar(100)  
 description String?  
 colorCode String @default("#3B82F6") @map("color\_code") @db.VarChar(7)  
 isSystemTag Boolean @default(false) @map("is\_system\_tag")  
 createdBy String? @map("created\_by") @db.Uuid  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
 usageCount Int @default(0) @map("usage\_count")  
  
 // Relations  
 creator User? @relation(fields: [createdBy], references: [id], onDelete: SetNull)  
 messages MessageTagRelation[]  
  
 @@map("message\_tags")  
}  
  
model MessageTagRelation {  
 id String @id @default(uuid()) @map("rel\_id") @db.Uuid  
 messageId String @map("message\_id") @db.Uuid  
 tagId String @map("tag\_id") @db.Uuid  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
 createdBy String? @map("created\_by") @db.Uuid  
  
 // Relations  
 message Message @relation(fields: [messageId], references: [id], onDelete: Cascade)  
 tag MessageTag @relation(fields: [tagId], references: [id], onDelete: Cascade)  
  
 @@unique([messageId, tagId])  
 @@map("message\_tag\_relations")  
}  
  
model AuditLog {  
 id String @id @default(uuid()) @map("log\_id") @db.Uuid  
 userId String? @map("user\_id") @db.Uuid  
 actionType String @map("action\_type") @db.VarChar(50)  
 tableName String @map("table\_name") @db.VarChar(100)  
 recordId String @map("record\_id") @db.Uuid  
 oldValues Json? @map("old\_values")  
 newValues Json? @map("new\_values")  
 ipAddress String? @map("ip\_address") @db.Inet  
 userAgent String? @map("user\_agent")  
 sessionId String? @map("session\_id") @db.Uuid  
 createdAt DateTime @default(now()) @map("created\_at") @db.Timestamptz  
  
 // Relations  
 user User? @relation(fields: [userId], references: [id], onDelete: SetNull)  
 session UserSession? @relation(fields: [sessionId], references: [id], onDelete: SetNull)  
  
 @@map("audit\_logs")  
}

## **7. Data Migration Strategy**

### **7.1 Initial Migration Script**

-- Migration: 001\_initial\_schema.sql  
-- Create initial database schema  
  
-- Enable UUID extension  
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";  
CREATE EXTENSION IF NOT EXISTS "pgcrypto";  
  
-- Create custom types  
CREATE TYPE user\_role AS ENUM ('user', 'admin', 'moderator');  
CREATE TYPE message\_role AS ENUM ('user', 'assistant', 'system');  
CREATE TYPE conversation\_type AS ENUM ('chat', 'analysis', 'brainstorm', 'support');  
  
-- Create tables in order of dependencies  
-- (Tables created above in order)  
  
-- Insert default system tags  
INSERT INTO message\_tags (tag\_name, description, color\_code, is\_system\_tag) VALUES  
('Important', 'Important messages', '#EF4444', true),  
('Question', 'Questions from users', '#3B82F6', true),  
('Code', 'Code-related discussions', '#10B981', true),  
('Bug', 'Bug reports or issues', '#F59E0B', true),  
('Feature', 'Feature requests', '#8B5CF6', true);

### **7.2 Sample Data Script**

-- Migration: 002\_sample\_data.sql  
-- Insert sample data for development/testing  
  
INSERT INTO users (azure\_id, email, display\_name, tenant\_id) VALUES  
('00000000-0000-0000-0000-000000000001', '[john.doe@company.com](mailto:john.doe@company.com)', 'John Doe', '258ac4e4-146a-411e-9dc8-79a9e12fd6da'),  
('00000000-0000-0000-0000-000000000002', '[jane.smith@company.com](mailto:jane.smith@company.com)', 'Jane Smith', '258ac4e4-146a-411e-9dc8-79a9e12fd6da');  
  
INSERT INTO conversations (user\_id, title, conversation\_type) VALUES  
((SELECT user\_id FROM users WHERE email = '[john.doe@company.com](mailto:john.doe@company.com)'), 'Getting Started with AI', 'chat'),  
((SELECT user\_id FROM users WHERE email = '[jane.smith@company.com](mailto:jane.smith@company.com)'), 'Code Review Discussion', 'analysis');

## **8. Performance Optimization**

### **8.1 Indexing Strategy**

-- Additional performance indexes  
CREATE INDEX CONCURRENTLY idx\_messages\_conv\_created   
ON messages(conv\_id, created\_at DESC);  
  
CREATE INDEX CONCURRENTLY idx\_conversations\_user\_updated   
ON conversations(user\_id, updated\_at DESC) WHERE is\_deleted = FALSE;  
  
CREATE INDEX CONCURRENTLY idx\_api\_requests\_date\_user   
ON api\_requests(DATE(created\_at), user\_id);  
  
-- Partial indexes for better performance  
CREATE INDEX CONCURRENTLY idx\_sessions\_active\_user   
ON user\_sessions(user\_id, last\_used DESC) WHERE is\_active = TRUE;  
  
CREATE INDEX CONCURRENTLY idx\_messages\_assistant\_recent   
ON messages(conv\_id, created\_at DESC) WHERE role = 'assistant' AND is\_deleted = FALSE;

### **8.2 Partitioning Strategy**

-- Partition audit\_logs by month for better performance  
CREATE TABLE audit\_logs\_partitioned (  
 LIKE audit\_logs INCLUDING ALL  
) PARTITION BY RANGE (created\_at);  
  
-- Create monthly partitions  
CREATE TABLE audit\_logs\_2025\_06 PARTITION OF audit\_logs\_partitioned  
 FOR VALUES FROM ('2025-06-01') TO ('2025-07-01');  
  
CREATE TABLE audit\_logs\_2025\_07 PARTITION OF audit\_logs\_partitioned  
 FOR VALUES FROM ('2025-07-01') TO ('2025-08-01');

## **9. Security Considerations**

### **9.1 Row Level Security (RLS)**

-- Enable RLS on sensitive tables  
ALTER TABLE conversations ENABLE ROW LEVEL SECURITY;  
ALTER TABLE messages ENABLE ROW LEVEL SECURITY;  
  
-- Users can only access their own conversations  
CREATE POLICY conversations\_user\_policy ON conversations  
 FOR ALL TO application\_user  
 USING (user\_id = current\_setting('app.current\_user\_id')::uuid);  
  
-- Users can only access messages in their conversations  
CREATE POLICY messages\_user\_policy ON messages  
 FOR ALL TO application\_user  
 USING (conv\_id IN (  
 SELECT conv\_id FROM conversations   
 WHERE user\_id = current\_setting('app.current\_user\_id')::uuid  
 ));

### **9.2 Data Encryption**

-- Encrypt sensitive fields  
CREATE OR REPLACE FUNCTION encrypt\_sensitive\_data()  
RETURNS TRIGGER AS $$  
BEGIN  
 -- Encrypt access tokens before storing  
 NEW.access\_token\_hash = crypt(NEW.access\_token\_hash, gen\_salt('bf', 8));  
 RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER encrypt\_session\_tokens  
 BEFORE INSERT OR UPDATE ON user\_sessions  
 FOR EACH ROW EXECUTE FUNCTION encrypt\_sensitive\_data();

## **10. Backup & Recovery Strategy**

### **10.1 Backup Configuration**

-- Create backup schema for point-in-time recovery  
CREATE SCHEMA backup\_management;  
  
-- Backup tracking table  
CREATE TABLE backup\_management.backup\_log (  
 backup\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 backup\_type VARCHAR(50) NOT NULL, -- 'full', 'incremental', 'transaction\_log'  
 start\_time TIMESTAMP WITH TIME ZONE NOT NULL,  
 end\_time TIMESTAMP WITH TIME ZONE,  
 size\_bytes BIGINT,  
 status VARCHAR(20) DEFAULT 'running',  
 backup\_location TEXT,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP  
);

### **10.2 Data Retention Policy**

-- Create function to archive old data  
CREATE OR REPLACE FUNCTION archive\_old\_data()  
RETURNS void AS $$  
BEGIN  
 -- Archive audit logs older than 1 year  
 INSERT INTO backup\_management.archived\_audit\_logs  
 SELECT \* FROM audit\_logs   
 WHERE created\_at < NOW() - INTERVAL '1 year';  
   
 DELETE FROM audit\_logs   
 WHERE created\_at < NOW() - INTERVAL '1 year';  
   
 -- Archive old API requests (older than 6 months)  
 INSERT INTO backup\_management.archived\_api\_requests  
 SELECT \* FROM api\_requests   
 WHERE created\_at < NOW() - INTERVAL '6 months';  
   
 DELETE FROM api\_requests   
 WHERE created\_at < NOW() - INTERVAL '6 months';  
END;  
$$ LANGUAGE plpgsql;  
  
-- Schedule to run monthly  
SELECT cron.schedule('archive-old-data', '0 2 1 \* \*', 'SELECT archive\_old\_data();');

## **11. Monitoring & Observability**

### **11.1 Database Health Monitoring**

-- Create monitoring views  
CREATE VIEW database\_health AS  
SELECT   
 'table\_sizes' AS metric\_type,  
 schemaname,  
 tablename,  
 pg\_size\_pretty(pg\_total\_relation\_size(schemaname||'.'||tablename)) AS size,  
 pg\_total\_relation\_size(schemaname||'.'||tablename) AS size\_bytes  
FROM pg\_tables   
WHERE schemaname = 'public'  
ORDER BY pg\_total\_relation\_size(schemaname||'.'||tablename) DESC;  
  
-- Query performance monitoring  
CREATE VIEW slow\_queries AS  
SELECT   
 query,  
 calls,  
 total\_time,  
 mean\_time,  
 rows,  
 100.0 \* shared\_blks\_hit / nullif(shared\_blks\_hit + shared\_blks\_read, 0) AS hit\_percent  
FROM pg\_stat\_statements   
ORDER BY mean\_time DESC   
LIMIT 20;

## **12. Integration Points**

### **12.1 API Integration Schema**

// TypeScript interfaces for database integration

interface DatabaseUser {  
 userId: string;  
 azureId: string;  
 email: string;  
 displayName: string;  
 tenantId: string;  
 createdAt: Date;  
 lastLogin?: Date;  
 isActive: boolean;  
 settings: Record<string, any>;  
}  
  
interface DatabaseMessage {  
 messageId: string;  
 convId: string;  
 userId: string;  
 content: string;  
 role: 'user' | 'assistant' | 'system';  
 messageType: 'text' | 'image' | 'file' | 'code';  
 tokenCount: number;  
 modelUsed?: string;  
 responseTimeMs?: number;  
 createdAt: Date;  
 metadata: Record<string, any>;  
}  
  
interface ConversationWithMessages {  
 convId: string;  
 title: string;  
 createdAt: Date;  
 updatedAt: Date;  
 messages: DatabaseMessage[];  
 analytics: {  
 totalTokens: number;  
 averageResponseTime: number;  
 sentimentScore?: number;  
 category?: string;  
 };  
}

## **13. Conclusion**

This comprehensive database design provides a robust foundation for the MSAL Chat Application with the following key benefits:

### **13.1 Scalability Features**

* **Horizontal Scaling**: Partitioned tables for large datasets
* **Query Optimization**: Strategic indexing and materialized views
* **Caching Strategy**: Support for Redis integration
* **Load Distribution**: Read replicas support

### **13.2 Security Features**

* **Row-Level Security**: User data isolation
* **Audit Trail**: Complete change tracking
* **Token Encryption**: Secure credential storage
* **GDPR Compliance**: Data deletion and portability

### **13.3 Analytics Capabilities**

* **User Behavior**: Comprehensive usage tracking
* **Performance Monitoring**: Response time and error analysis
* **Business Intelligence**: Conversation analytics and insights
* **Cost Optimization**: Token usage and API monitoring

### **13.4 Maintenance & Operations**

* **Automated Backups**: Point-in-time recovery
* **Data Archival**: Automated old data management
* **Health Monitoring**: Database performance tracking
* **Migration Support**: Version-controlled schema changes

This design supports both the current stateless implementation and provides a migration path for enhanced features like persistent chat history, user analytics, and advanced conversation management.