Contents

SECTION 22: PERSISTENT MEMORY (v3.6.0)

22.1 Memory System Overview

Long-term memory storage using pgvector embeddings for semantic search.

22.2 Memory Database Schema

```sql
-- migrations/031_persistent_memory.sql

CREATE TABLE memory_stores (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    store_name VARCHAR(100) NOT NULL DEFAULT 'default',
    embedding_model VARCHAR(100) DEFAULT 'text-embedding-3-small',
    total_memories INTEGER DEFAULT 0,
    last_accessed TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(tenant_id, user_id, store_name)
);

CREATE TABLE memories (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    store_id UUID NOT NULL REFERENCES memory_stores(id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    embedding vector(1536),
    memory_type VARCHAR(50) DEFAULT 'fact',
    source VARCHAR(100),
    importance DECIMAL(3, 2) DEFAULT 0.5,
    access_count INTEGER DEFAULT 0,
    last_accessed TIMESTAMPTZ,
    expires_at TIMESTAMPTZ,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

```sql
CREATE TABLE memory_relationships (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    source_memory_id UUID NOT NULL REFERENCES memories(id) ON DELETE CASCADE,
    target_memory_id UUID NOT NULL REFERENCES memories(id) ON DELETE CASCADE,
    relationship_type VARCHAR(50) NOT NULL,
    strength DECIMAL(3, 2) DEFAULT 0.5,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(source_memory_id, target_memory_id, relationship_type)
);

CREATE INDEX idx_memories_embedding ON memories USING ivfflat (embedding vector_cosine_ops) WI
CREATE INDEX idx_memories_store ON memories(store_id);
CREATE INDEX idx_memory_relationships ON memory_relationships(source_memory_id);

ALTER TABLE memory_stores ENABLE ROW LEVEL SECURITY;
ALTER TABLE memories ENABLE ROW LEVEL SECURITY;
ALTER TABLE memory_relationships ENABLE ROW LEVEL SECURITY;

CREATE POLICY memory_stores_isolation ON memory_stores USING (tenant_id = current_setting('app
CREATE POLICY memories_isolation ON memories USING (
    store_id IN (SELECT id FROM memory_stores WHERE tenant_id = current_setting('app.current_t
);
CREATE POLICY memory_relationships_isolation ON memory_relationships USING (
    source_memory_id IN (SELECT m.id FROM memories m JOIN memory_stores ms ON m.store_id = ms.
);
```

22.3 Memory Service

```typescript
// packages/core/src/services/memory-service.ts

import { Pool } from 'pg';
import { BedrockRuntimeClient, InvokeModelCommand } from '@aws-sdk/client-bedrock-runtime';

export class MemoryService {
    private pool: Pool;
    private bedrock: BedrockRuntimeClient;

    constructor(pool: Pool) {
        this.pool = pool;
        this.bedrock = new BedrockRuntimeClient({});
    }

    async getOrCreateStore(tenantId: string, userId: string, storeName: string = 'default'): P
        const result = await this.pool.query(`
            INSERT INTO memory_stores (tenant_id, user_id, store_name)
            VALUES ($1, $2, $3)
            ON CONFLICT (tenant_id, user_id, store_name) DO UPDATE SET last_accessed = NOW()
            RETURNING id
        `, [tenantId, userId, storeName]);

        return result.rows[0].id;
    }

    async addMemory(
        storeId: string,
        content: string,
        options?: {
            type?: string;
            source?: string;
            importance?: number;
            metadata?: Record<string, any>;
        }
    ): Promise<string> {
        const embedding = await this.generateEmbedding(content);

        const result = await this.pool.query(`
            INSERT INTO memories (store_id, content, embedding, memory_type, source, importanc
            VALUES ($1, $2, $3::vector, $4, $5, $6, $7)
            RETURNING id
        `, [
            storeId,
            content,
            `[${embedding.join(',')}]`,
            options?.type || 'fact',
            options?.source,
            options?.importance || 0.5,
            JSON.stringify(options?.metadata || {})
        ]);

        // Update store count
        await this.pool.query(`
            UPDATE memory_stores SET total_memories = total_memories + 1 WHERE id = $1
        `, [storeId]);

        return result.rows[0].id;
    }

    async searchMemories(
        storeId: string,
        query: string,
        limit: number = 5,
        minSimilarity: number = 0.7
    ): Promise<any[]> {
        const embedding = await this.generateEmbedding(query);
        const result = await this.pool.query(`
            SELECT
                id, content, memory_type, source, importance, metadata,
                1 - (embedding <=> $2::vector) as similarity
            FROM memories
            WHERE store_id = $1
            AND (expires_at IS NULL OR expires_at > NOW())
            AND 1 - (embedding <=> $2::vector) >= $4
            ORDER BY embedding <=> $2::vector
            LIMIT $3
        `, [storeId, `[${embedding.join(',')}]`, limit, minSimilarity]);

        // Update access counts
        const memoryIds = result.rows.map(r => r.id);
        if (memoryIds.length > 0) {
            await this.pool.query(`
                UPDATE memories SET access_count = access_count + 1, last_accessed = NOW()
                WHERE id = ANY($1)
            `, [memoryIds]);
        }

        return result.rows;
    }

    async addRelationship(
        sourceId: string,
        targetId: string,
        relationshipType: string,
        strength: number = 0.5
    ): Promise<void> {
        await this.pool.query(`
            INSERT INTO memory_relationships (source_memory_id, target_memory_id, relationship_
            VALUES ($1, $2, $3, $4)
            ON CONFLICT (source_memory_id, target_memory_id, relationship_type)
            DO UPDATE SET strength = EXCLUDED.strength
        `, [sourceId, targetId, relationshipType, strength]);
    }

    async getRelatedMemories(memoryId: string, limit: number = 5): Promise<any[]> {
        const result = await this.pool.query(`
            SELECT m.*, mr.relationship_type, mr.strength
            FROM memory_relationships mr
            JOIN memories m ON mr.target_memory_id = m.id
            WHERE mr.source_memory_id = $1
            ORDER BY mr.strength DESC
            LIMIT $2
        `, [memoryId, limit]);
        return result.rows;
    }

    private async generateEmbedding(text: string): Promise<number[]> {
        const response = await this.bedrock.send(new InvokeModelCommand({
            modelId: 'amazon.titan-embed-text-v1',
            body: JSON.stringify({ inputText: text }),
            contentType: 'application/json'
        }));

        const result = JSON.parse(new TextDecoder().decode(response.body));
        return result.embedding;
    }
}
```