Contents

SECTION 24: RESULT MERGING & AI SYNTHESIS (v3.6.0)

24.1 Result Merging Overview

Combine responses from multiple AI models with intelligent synthesis.

24.2 Merge Database Schema

```sql
-- migrations/033_result_merging.sql

CREATE TABLE merge_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id),
    user_id UUID NOT NULL REFERENCES users(id),
    prompt TEXT NOT NULL,
    merge_strategy VARCHAR(50) NOT NULL DEFAULT 'consensus',
    models_used TEXT[] NOT NULL,
    final_result TEXT,
    quality_score DECIMAL(3, 2),
    total_cost DECIMAL(10, 6),
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE merge_responses (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    session_id UUID NOT NULL REFERENCES merge_sessions(id) ON DELETE CASCADE,
    model VARCHAR(100) NOT NULL,
    response TEXT NOT NULL,
    tokens_used INTEGER,
    latency_ms INTEGER,
    contribution_weight DECIMAL(3, 2) DEFAULT 1.0,
    created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_merge_sessions_user ON merge_sessions(tenant_id, user_id);
CREATE INDEX idx_merge_responses_session ON merge_responses(session_id);
```

```
ALTER TABLE merge_sessions ENABLE ROW LEVEL SECURITY;
ALTER TABLE merge_responses ENABLE ROW LEVEL SECURITY;

CREATE POLICY merge_sessions_isolation ON merge_sessions USING (tenant_id = current_setting('a
CREATE POLICY merge_responses_isolation ON merge_responses USING (
    session_id IN (SELECT id FROM merge_sessions WHERE tenant_id = current_setting('app.curren
);
```

24.3 Result Merger Service

```
// packages/core/src/services/result-merger.ts

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

type MergeStrategy = 'consensus' | 'best_of' | 'synthesize' | 'debate';

interface MergeResult {
    sessionId: string;
    mergedResponse: string;
    qualityScore: number;
    contributions: Array<{ model: string; weight: number }>;
}

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

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

    async merge(
        tenantId: string,
        userId: string,
        prompt: string,
        responses: Array<{ model: string; response: string; tokens?: number; latencyMs?: numbe
        strategy: MergeStrategy = 'consensus'
    ): Promise<MergeResult> {
        // Create session
        const models = responses.map(r => r.model);
        const sessionResult = await this.pool.query(`
            INSERT INTO merge_sessions (tenant_id, user_id, prompt, merge_strategy, models_use
            VALUES ($1, $2, $3, $4, $5)
            RETURNING id
        `, [tenantId, userId, prompt, strategy, models]);
```

```
    const sessionId = sessionResult.rows[0].id;

    // Store individual responses
    for (const response of responses) {
        await this.pool.query(`
            INSERT INTO merge_responses (session_id, model, response, tokens_used, latency_
            VALUES ($1, $2, $3, $4, $5)
        `, [sessionId, response.model, response.response, response.tokens, response.latenc
    }

    // Perform merge based on strategy
    const mergeResult = await this.performMerge(prompt, responses, strategy);

    // Update session with result
    await this.pool.query(`
        UPDATE merge_sessions SET final_result = $2, quality_score = $3 WHERE id = $1
    `, [sessionId, mergeResult.mergedResponse, mergeResult.qualityScore]);

    return {
        sessionId,
        ...mergeResult
    };
}

private async performMerge(
    prompt: string,
    responses: Array<{ model: string; response: string }>,
    strategy: MergeStrategy
): Promise<{ mergedResponse: string; qualityScore: number; contributions: Array<{ model: s
    const strategyHandlers: Record<MergeStrategy, () => Promise<any>> = {
        consensus: () => this.consensusMerge(responses),
        best_of: () => this.bestOfMerge(prompt, responses),
        synthesize: () => this.synthesizeMerge(prompt, responses),
        debate: () => this.debateMerge(prompt, responses)
    };

    return strategyHandlers[strategy]();
}

private async consensusMerge(responses: Array<{ model: string; response: string }>) {
    const responsesText = responses.map((r, i) => `Response ${i + 1} (${r.model}):\n${r.res

    const synthesis = await this.invokeModel(`
        Analyze these AI responses and create a consensus response that incorporates the b

        ${responsesText}
```

```
            Create a unified response that represents the consensus view. Return JSON: { "resp
        `);

        return {
            mergedResponse: synthesis.response,
            qualityScore: 0.85,
            contributions: synthesis.contributions || responses.map(r => ({ model: r.model, we
        };
    }

    private async bestOfMerge(prompt: string, responses: Array<{ model: string; response: stri
        const responsesText = responses.map((r, i) => `Response ${i + 1} (${r.model}):\n${r.re

        const evaluation = await this.invokeModel(`
            Original question: ${prompt}

            Evaluate these responses and select the best one:

            ${responsesText}

            Return JSON: { "bestIndex": 0-${responses.length - 1}, "reason": "...", "score": 0
        `);

        const bestResponse = responses[evaluation.bestIndex || 0];

        return {
            mergedResponse: bestResponse.response,
            qualityScore: evaluation.score || 0.8,
            contributions: [{ model: bestResponse.model, weight: 1.0 }]
        };
    }

    private async synthesizeMerge(prompt: string, responses: Array<{ model: string; response: 
        const responsesText = responses.map((r, i) => `Response ${i + 1} (${r.model}):\n${r.re

        const synthesis = await this.invokeModel(`
            Original question: ${prompt}

            Create a comprehensive synthesis that combines insights from all these responses:

            ${responsesText}

            Synthesize into a single, well-structured response that incorporates unique insigh
        `);

        return {
            mergedResponse: synthesis,
            qualityScore: 0.9,
```

```
            contributions: responses.map(r => ({ model: r.model, weight: 1 / responses.length
        };
    }

    private async debateMerge(prompt: string, responses: Array<{ model: string; response: stri
        // Multi-round debate simulation
        const responsesText = responses.map((r, i) => `${r.model}:\n${r.response}`).join('\n\n

        const debate = await this.invokeModel(`
            Simulate a debate between these AI perspectives on: ${prompt}

            Initial positions:
            ${responsesText}

            Identify points of agreement and disagreement, then synthesize a conclusion that ac
        `);

        return {
            mergedResponse: debate,
            qualityScore: 0.85,
            contributions: responses.map(r => ({ model: r.model, weight: 1 / responses.length
        };
    }

    private async invokeModel(prompt: string): Promise<any> {
        const response = await this.bedrock.send(new InvokeModelCommand({
            modelId: 'anthropic.claude-3-sonnet-20240229-v1:0',
            body: JSON.stringify({
                anthropic_version: 'bedrock-2023-05-31',
                max_tokens: 4096,
                messages: [{ role: 'user', content: prompt }]
            }),
            contentType: 'application/json'
        }));

        const result = JSON.parse(new TextDecoder().decode(response.body));
        const text = result.content[0].text;

        try {
            const jsonMatch = text.match(/\{[\s\S]*\}/);
            return jsonMatch ? JSON.parse(jsonMatch[0]) : text;
        } catch {
            return text;
        }
    }
}
```