Contents

RADIANT v5.0.2 - Source Export Part 1: Database & TypeScript

---

1. Database Schema Migration

File: `packages/infrastructure/migrations/V2026_01_09_001__v5_grimoire_governor.sql`

Purpose: Defines all database tables, Row-Level Security policies, indexes, views, and triggers for The Grimoire and Economic Governor systems.

Key Tables: - `knowledge_heuristics` - Stores procedural memory heuristics with vector embeddings - `governor_savings_log` - Tracks cost optimization decisions and savings - `decision_domain_config` - Per-domain Governor mode configuration

Security: Uses PostgreSQL Row-Level Security (RLS) with `app.current_tenant_id` for multi-tenant isolation.

```sql
-- RADIANT v5.0.2 - System Evolution
-- The Grimoire & Economic Governor Database Schema

-- Enable pgvector extension for semantic similarity search
CREATE EXTENSION IF NOT EXISTS vector;

-- ============================================================================
-- THE GRIMOIRE: Self-Optimizing Procedural Memory
-- ============================================================================

CREATE TABLE IF NOT EXISTS knowledge_heuristics (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
    domain VARCHAR(50) NOT NULL DEFAULT 'general',
    heuristic_text TEXT NOT NULL,
    context_embedding vector(1536),
    confidence_score FLOAT NOT NULL DEFAULT 0.5 CHECK (confidence_score >= 0 AND confidence_sc
    source_execution_id VARCHAR(255),
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    expires_at TIMESTAMPTZ NOT NULL DEFAULT (NOW() + INTERVAL '90 days')
);

-- RLS Policy: Tenants can only see their own heuristics
ALTER TABLE knowledge_heuristics ENABLE ROW LEVEL SECURITY;
```

```sql
CREATE POLICY knowledge_heuristics_tenant_isolation ON knowledge_heuristics
    FOR ALL
    USING (tenant_id = current_setting('app.current_tenant_id')::uuid
           OR current_setting('app.current_tenant_id') = '00000000-0000-0000-0000-000000000000
    WITH CHECK (tenant_id = current_setting('app.current_tenant_id')::uuid);

-- Indexes for query performance
CREATE INDEX idx_heuristics_tenant_domain ON knowledge_heuristics(tenant_id, domain);
CREATE INDEX idx_heuristics_confidence ON knowledge_heuristics(confidence_score DESC);
CREATE INDEX idx_heuristics_expires ON knowledge_heuristics(expires_at);
CREATE INDEX idx_heuristics_embedding ON knowledge_heuristics USING ivfflat (context_embedding

-- Trigger for updated_at
CREATE OR REPLACE FUNCTION update_heuristics_timestamp()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER heuristics_update_timestamp
    BEFORE UPDATE ON knowledge_heuristics
    FOR EACH ROW EXECUTE FUNCTION update_heuristics_timestamp();

-- Audit trail for heuristic changes
CREATE TABLE IF NOT EXISTS heuristics_audit (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    heuristic_id UUID NOT NULL,
    tenant_id UUID NOT NULL,
    action VARCHAR(20) NOT NULL,
    old_confidence FLOAT,
    new_confidence FLOAT,
    changed_by VARCHAR(255),
    changed_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE OR REPLACE FUNCTION audit_heuristic_changes()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'UPDATE' AND OLD.confidence_score != NEW.confidence_score THEN
        INSERT INTO heuristics_audit (heuristic_id, tenant_id, action, old_confidence, new_con
        VALUES (NEW.id, NEW.tenant_id, 'CONFIDENCE_CHANGE', OLD.confidence_score, NEW.confiden
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO heuristics_audit (heuristic_id, tenant_id, action, old_confidence)
        VALUES (OLD.id, OLD.tenant_id, 'DELETE', OLD.confidence_score);
    END IF;
    RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER heuristics_audit_trigger
    AFTER UPDATE OR DELETE ON knowledge_heuristics
    FOR EACH ROW EXECUTE FUNCTION audit_heuristic_changes();

-- ============================================================================
-- ECONOMIC GOVERNOR: Cost Optimization Through Intelligent Model Routing
-- ============================================================================

CREATE TABLE IF NOT EXISTS governor_savings_log (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
    execution_id VARCHAR(255) NOT NULL,
    original_model VARCHAR(100) NOT NULL,
    selected_model VARCHAR(100) NOT NULL,
    complexity_score INTEGER NOT NULL CHECK (complexity_score >= 1 AND complexity_score <= 10)
    estimated_original_cost DECIMAL(10,6) NOT NULL DEFAULT 0,
    estimated_actual_cost DECIMAL(10,6) NOT NULL DEFAULT 0,
    savings_amount DECIMAL(10,6) GENERATED ALWAYS AS (estimated_original_cost - estimated_actu
    governor_mode VARCHAR(20) NOT NULL,
    reason TEXT,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

ALTER TABLE governor_savings_log ENABLE ROW LEVEL SECURITY;

CREATE POLICY governor_savings_tenant_isolation ON governor_savings_log
    FOR ALL
    USING (tenant_id = current_setting('app.current_tenant_id')::uuid
           OR current_setting('app.current_tenant_id') = '00000000-0000-0000-0000-000000000000
    WITH CHECK (tenant_id = current_setting('app.current_tenant_id')::uuid);

CREATE INDEX idx_governor_savings_tenant ON governor_savings_log(tenant_id);
CREATE INDEX idx_governor_savings_created ON governor_savings_log(created_at DESC);
CREATE INDEX idx_governor_savings_mode ON governor_savings_log(governor_mode);

-- Add governor_mode to decision_domain_config
ALTER TABLE decision_domain_config
ADD COLUMN IF NOT EXISTS governor_mode VARCHAR(20) NOT NULL DEFAULT 'balanced';

-- View for Grimoire statistics
CREATE OR REPLACE VIEW grimoire_stats AS
SELECT
    tenant_id,
    domain,
```

```sql
    COUNT(*) as total_heuristics,
    AVG(confidence_score) as avg_confidence,
    COUNT(*) FILTER (WHERE confidence_score >= 0.8) as high_confidence_count,
    COUNT(*) FILTER (WHERE expires_at < NOW() + INTERVAL '7 days') as expiring_soon,
    MAX(created_at) as last_added
FROM knowledge_heuristics
WHERE expires_at > NOW()
GROUP BY tenant_id, domain;

-- View for Governor statistics
CREATE OR REPLACE VIEW governor_stats AS
SELECT
    tenant_id,
    governor_mode,
    COUNT(*) as decision_count,
    SUM(savings_amount) as total_savings,
    AVG(complexity_score) as avg_complexity,
    COUNT(*) FILTER (WHERE original_model != selected_model) as model_swaps
FROM governor_savings_log
WHERE created_at > NOW() - INTERVAL '30 days'
GROUP BY tenant_id, governor_mode;
```

---

2. Economic Governor Service

File: packages/infrastructure/lambda/shared/services/governor/economic-governor.ts

Purpose: Core logic for the Economic Governor. Implements "System 0" complexity classification and cost-optimized model routing.

Key Features: - Complexity scoring using cheap classifier model - Model tier mapping for cost optimization - Savings calculation and logging - Per-domain mode configuration

```typescript
/**
 * Economic Governor - Cost Optimization Through Intelligent Model Routing
 * RADIANT v5.0.2 - System Evolution
 *
 * The Economic Governor implements a "System 0" approach to cost optimization:
 * 1. A cheap classifier model scores task complexity (1-10)
 * 2. Simple tasks (1-4) are routed to efficient models
 * 3. Complex tasks (9-10) are routed to premium models
 * 4. Medium tasks (5-8) use the original requested model
 */

import { PoolClient } from 'pg';

export type GovernorMode = 'performance' | 'balanced' | 'cost_saver' | 'off';

export interface SwarmTask {
  type: string;
  prompt: string;
  context: Record<string, unknown>;
  system_prompt?: string;
}

export interface AgentConfig {
  agent_id: string;
  role: string;
  model: string;
  temperature?: number;
  max_tokens?: number;
  tools?: string[];
}

export interface GovernorDecision {
  shouldOptimize: boolean;
  selectedModel: string;
  originalModel: string;
  complexityScore: number;
  estimatedSavings: number;
  reason: string;
}

export interface GovernorConfig {
  mode: GovernorMode;
  complexityThreshold: {
    simple: number;    // Below this = simple task
    complex: number;   // Above this = complex task
  };
  modelMapping: {
    simple: string;    // Model for simple tasks
    premium: string;   // Model for complex tasks
  };
}

const DEFAULT_CONFIG: GovernorConfig = {
  mode: 'balanced',
  complexityThreshold: {
    simple: 4,
    complex: 8
  },
  modelMapping: {
    simple: 'gpt-4o-mini',
    premium: 'gpt-4o'
  }
};
```

```typescript
// Cost per 1K tokens (approximate)
const MODEL_COSTS: Record<string, { input: number; output: number }> = {
  'gpt-4o': { input: 0.005, output: 0.015 },
  'gpt-4o-mini': { input: 0.00015, output: 0.0006 },
  'gpt-4-turbo': { input: 0.01, output: 0.03 },
  'claude-3-opus': { input: 0.015, output: 0.075 },
  'claude-3-sonnet': { input: 0.003, output: 0.015 },
  'claude-3-haiku': { input: 0.00025, output: 0.00125 },
  'claude-3.5-sonnet': { input: 0.003, output: 0.015 },
};

export class EconomicGovernor {
  private config: GovernorConfig;
  private litellmUrl: string;
  private litellmApiKey: string;

  constructor(
    config: Partial<GovernorConfig> = {},
    litellmUrl?: string,
    litellmApiKey?: string
  ) {
    this.config = { ...DEFAULT_CONFIG, ...config };
    this.litellmUrl = litellmUrl || process.env.LITELLM_PROXY_URL || 'http://localhost:4000';
    this.litellmApiKey = litellmApiKey || process.env.LITELLM_API_KEY || '';
  }

  async evaluateTask(
    task: SwarmTask,
    agent: AgentConfig,
    domain: string = 'general'
  ): Promise<GovernorDecision> {
    if (this.config.mode === 'off') {
      return {
        shouldOptimize: false,
        selectedModel: agent.model,
        originalModel: agent.model,
        complexityScore: 5,
        estimatedSavings: 0,
        reason: 'Governor is disabled'
      };
    }

    if (this.config.mode === 'performance') {
      return {
        shouldOptimize: false,
        selectedModel: agent.model,
        originalModel: agent.model,
        complexityScore: 5,
        estimatedSavings: 0,
        reason: 'Performance mode - no optimization'
      };
    }

    const complexityScore = await this.classifyComplexity(task);
    const decision = this.makeRoutingDecision(complexityScore, agent.model);

    return decision;
  }

  private async classifyComplexity(task: SwarmTask): Promise<number> {
    const classifierPrompt = `Rate the complexity of this task on a scale of 1-10.
1-4 = Simple (basic Q&A, formatting, summarization)
5-8 = Medium (analysis, multi-step reasoning, code generation)
9-10 = Complex (research synthesis, creative writing, expert domain knowledge)

Task: ${task.prompt.substring(0, 500)}

Respond with ONLY a single number from 1-10.`;

    try {
      const response = await fetch(`${this.litellmUrl}/chat/completions`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${this.litellmApiKey}`
        },
        body: JSON.stringify({
          model: 'gpt-4o-mini',
          messages: [{ role: 'user', content: classifierPrompt }],
          max_tokens: 5,
          temperature: 0
        })
      });

      if (!response.ok) {
        console.warn('Complexity classification failed, defaulting to 5');
        return 5;
      }

      const data = await response.json();
      const scoreText = data.choices?.[0]?.message?.content?.trim() || '5';
      const score = parseInt(scoreText, 10);

      return isNaN(score) ? 5 : Math.max(1, Math.min(10, score));
    } catch (error) {
      console.error('Complexity classification error:', error);
      return 5;
    }
  }

  private makeRoutingDecision(complexityScore: number, originalModel: string): GovernorDecisio
    const { simple, complex } = this.config.complexityThreshold;
    const { simple: simpleModel, premium: premiumModel } = this.config.modelMapping;

    let selectedModel = originalModel;
    let shouldOptimize = false;
    let reason = 'Medium complexity - using original model';

    if (complexityScore <= simple) {
      if (this.config.mode === 'balanced' || this.config.mode === 'cost_saver') {
        selectedModel = simpleModel;
        shouldOptimize = true;
        reason = `Simple task (${complexityScore}/10) - routing to efficient model`;
      }
    } else if (complexityScore >= complex) {
      if (this.config.mode === 'cost_saver') {
        selectedModel = originalModel;
        reason = `Complex task (${complexityScore}/10) - preserving quality`;
      } else {
        if (!this.isPremiumModel(originalModel)) {
          selectedModel = premiumModel;
          shouldOptimize = true;
          reason = `Complex task (${complexityScore}/10) - upgrading to premium model`;
        }
      }
    }

    const estimatedSavings = this.calculateSavings(originalModel, selectedModel);

    return {
      shouldOptimize,
      selectedModel,
      originalModel,
      complexityScore,
      estimatedSavings,
      reason
    };
  }

  private isPremiumModel(model: string): boolean {
    const premiumModels = ['gpt-4o', 'gpt-4-turbo', 'claude-3-opus', 'claude-3.5-sonnet'];
    return premiumModels.some(pm => model.includes(pm));
  }
  private calculateSavings(originalModel: string, selectedModel: string): number {
    const originalCost = MODEL_COSTS[originalModel] || { input: 0.005, output: 0.015 };
    const selectedCost = MODEL_COSTS[selectedModel] || originalCost;
    const estimatedTokens = 2000;
    const originalTotal = (originalCost.input + originalCost.output) * estimatedTokens / 1000;
    const selectedTotal = (selectedCost.input + selectedCost.output) * estimatedTokens / 1000;
    return Math.max(0, originalTotal - selectedTotal);
  }

  async logDecision(
    client: PoolClient,
    tenantId: string,
    executionId: string,
    decision: GovernorDecision
  ): Promise<void> {
    const originalCost = this.calculateModelCost(decision.originalModel);
    const actualCost = this.calculateModelCost(decision.selectedModel);

    await client.query(`
      INSERT INTO governor_savings_log (
        tenant_id, execution_id, original_model, selected_model,
        complexity_score, estimated_original_cost, estimated_actual_cost,
        governor_mode, reason
      ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9)
    `, [
      tenantId,
      executionId,
      decision.originalModel,
      decision.selectedModel,
      decision.complexityScore,
      originalCost,
      actualCost,
      this.config.mode,
      decision.reason
    ]);
  }

  private calculateModelCost(model: string): number {
    const cost = MODEL_COSTS[model] || { input: 0.005, output: 0.015 };
    return (cost.input + cost.output) * 2;
  }
}

export function createGovernor(mode: GovernorMode = 'balanced'): EconomicGovernor {
  return new EconomicGovernor({ mode });
}
```

3. Governor Module Exports

File: `packages/infrastructure/lambda/shared/services/governor/index.ts`

Purpose: Central export point for the Economic Governor module.

```
/**
 * Economic Governor Module Exports
 * RADIANT v5.0.2 - System Evolution
 */

export {
  EconomicGovernor,
  createGovernor,
  type GovernorMode,
  type GovernorDecision,
  type GovernorConfig,
  type SwarmTask,
  type AgentConfig
} from './economic-governor';
```

Continued in GRIMOIRE-GOVERNOR-SOURCE-PART2.md