**Job Security Scoring Methodology & Implementation Prompts**

**Core Methodology: Multi-Factor Job Security Analysis**

**1. Data Source Integration Framework**

**Primary Data Sources (Free/Publicly Available):**

1. **SEC EDGAR Database** - Company financial health
2. **State WARN Databases** - Local layoff notifications
3. **Bureau of Labor Statistics** - Industry employment trends
4. **Federal Reserve Economic Data (FRED)** - Economic indicators
5. **Company Glassdoor/Indeed Reviews** - Employee sentiment patterns
6. **User Self-Assessment** - Personal job security perceptions

**Secondary Data Sources (Enhanced Tiers):**

1. **LinkedIn API** - Professional network changes
2. **News API** - Company-specific news sentiment
3. **Industry-specific databases** - Sector risk assessments

**2. Enhanced Career Stress Tracking Implementation**

**Prompt 1: Weekly Career Check-in Questions Design**

Create a comprehensive weekly career check-in questionnaire that:

- Builds on existing job stress level question (1-10 scale)

- Adds job security feeling assessment (1-10 scale)

- Captures employer warning signals (layoffs, budget cuts, restructuring)

- Records company name for financial analysis

- Identifies industry sector for risk assessment

- Includes conditional questions based on stress/security levels

- Maintains cultural sensitivity for African American professionals aged 25-35

- Takes no more than 3-4 minutes to complete

- Provides immediate feedback and reassurance

Format the questions to be mobile-friendly and include:

- Clear, non-anxiety-inducing language

- Progress indicators

- Optional explanatory text for each question

- Skip logic for relevant follow-ups

**Prompt 2: Job Security Scoring Algorithm Enhancement**

Design a comprehensive job security scoring algorithm that:

- Weighs user perception (60%) vs external data (40%)

- Incorporates stress level impact on security perception

- Accounts for employer warning signals with appropriate penalties

- Integrates external financial data when available

- Provides fallback scoring when external data is unavailable

- Generates confidence levels for each score component

- Creates risk factor identification

- Produces actionable recommendations based on score ranges

- Maintains scores between 0-100 for easy understanding

- Updates dynamically as new data becomes available

Include specific formulas for:

- User perception calculations

- External risk factor weighting

- Confidence level determination

- Recommendation triggers

**3. External Data Integration Prompts**

**Prompt 3: SEC EDGAR Financial Health Analyzer**

Create a Python class that:

- Searches SEC EDGAR database by company name

- Retrieves latest 10-K and 10-Q filings

- Extracts key financial metrics (revenue, debt, cash, profitability)

- Calculates financial health score (0-100)

- Identifies specific risk indicators

- Handles private companies gracefully

- Provides data freshness timestamps

- Returns structured JSON with confidence levels

- Implements rate limiting for API compliance

- Includes error handling for missing data

Key financial indicators to analyze:

- Revenue trend (3-year analysis)

- Debt-to-equity ratios

- Cash flow stability

- Profit margin trends

- Working capital position

- Recent filing frequency and timeliness

**Prompt 4: Local Labor Market Risk Assessment**

Design a system to analyze local job market conditions that:

- Integrates state WARN databases for layoff notifications

- Analyzes geographic proximity to user location (25-mile radius)

- Weights recent layoffs by company size and industry relevance

- Incorporates BLS local unemployment data

- Tracks industry-specific employment trends in user's area

- Provides comparative analysis to national averages

- Generates local risk scores (0-100)

- Updates weekly with fresh data

- Handles data gaps gracefully

Include specific calculations for:

- Geographic proximity weighting

- Temporal relevance (recent vs older layoffs)

- Industry correlation factors

- Company size impact multipliers

**4. Enhanced Dashboard Integration**

**Prompt 5: Job Security Dashboard Component**

Create a responsive web component for job security analysis that:

- Displays overall security score prominently (0-100)

- Shows component breakdowns (user perception, external data)

- Uses color coding (green/yellow/red) for risk levels

- Provides trend arrows for score changes over time

- Lists specific risk factors and positive indicators

- Generates personalized action recommendations

- Includes "What changed?" explanations for score updates

- Offers drill-down into component details

- Maintains mobile-first responsive design

- Integrates seamlessly with existing health correlation cards

Visual elements to include:

- Circular progress indicator for main score

- Factor contribution bars

- Trend line for 12-week score history

- Risk level badges

- Recommendation priority indicators

**5. Industry-Specific Risk Modeling**

**Prompt 6: Industry Risk Assessment Framework**

Develop an industry risk assessment system that:

- Maps user's industry to standard NAICS codes

- Analyzes industry-wide employment trends

- Incorporates automation/AI replacement risk scores

- Tracks industry growth/decline patterns

- Considers economic cycle sensitivity

- Includes geographic concentration risks

- Weighs remote work impact on industry stability

- Provides industry-specific career advancement insights

- Updates monthly with fresh economic data

Create risk profiles for key industries serving your target demographic:

- Technology/Software

- Healthcare

- Financial Services

- Education

- Government/Public Sector

- Retail/Hospitality

- Manufacturing

- Professional Services

**6. Actionable Recommendations Engine**

**Prompt 7: Personalized Recommendation System**

Build a recommendation engine that:

- Generates specific actions based on job security scores

- Personalizes advice for African American professionals

- Considers user's income level ($40K-$100K range)

- Accounts for geographic location and local job markets

- Provides timeline-based action plans (immediate, 30-day, 90-day)

- Includes skill development suggestions relevant to user's industry

- Offers networking and career advancement strategies

- Recommends emergency fund targets based on risk level

- Suggests side income opportunities when appropriate

- Connects to relevant resources and professional development platforms

Recommendation categories by score range:

- High Risk (0-40): Emergency preparation focus

- Moderate Risk (41-60): Skill building and networking

- Low Risk (61-80): Career advancement opportunities

- Secure (81-100): Long-term growth strategies

**7. Cultural Sensitivity & User Experience**

**Prompt 8: Culturally Responsive Interface Design**

Design user interface elements that:

- Use encouraging, non-alarmist language for risk communication

- Acknowledge unique challenges facing African American professionals

- Provide context for systemic vs individual factors

- Offer community-focused solutions and networking opportunities

- Include success stories and positive role models

- Avoid stigmatizing language around financial stress

- Emphasize empowerment and control over available factors

- Provide clear explanations of how external factors are measured

- Include options to connect with culturally competent career coaches

- Offer privacy controls for sensitive career information

Language guidelines:

- Focus on "opportunity awareness" rather than "job insecurity"

- Use "career resilience building" instead of "risk mitigation"

- Emphasize "strategic planning" over "emergency preparation"

- Frame recommendations as "career optimization strategies"

**8. Implementation Testing**

**Prompt 9: Phased Rollout Strategy**

Testing priorities:

- User comprehension of scores and recommendations

- Emotional response to risk level communications

- Actionability and usefulness of generated advice

- Data accuracy and freshness verification

- Performance under various data availability scenarios

**9. Integration with Existing Mingus Features**

**Prompt 10: Holistic Financial Impact Modeling**

Design integration points that connect job security analysis with:

- Cash flow forecasting (adjust emergency fund recommendations)

- Health correlation analysis (stress spending patterns)

- Milestone planning (career transition timing)

- Special events budgeting (professional development costs)

- Relationship health tracking (career stress impact on relationships)

Create specific algorithms that:

- Adjust emergency fund targets based on job security scores

- Modify spending recommendations during high-risk periods

- Suggest career investment timing based on financial stability

- Correlate job stress with health metrics and spending patterns

- Provide integrated life planning that considers career transitions

**Implementation Priority Matrix**

**Future Enhancements:**

1. Advanced predictive modeling
2. Professional network analysis
3. Skills gap assessment tools
4. Career transition planning features

**Success Metrics**

**User Engagement:**

* Weekly check-in completion rates
* Time spent on job security dashboard
* Recommendation click-through rates
* User feedback scores

**Accuracy Validation:**

* Prediction accuracy for users who experience job changes
* Correlation between scores and actual employment outcomes
* External data source reliability metrics
* User perception vs reality alignment

**Business Impact:**

* Feature adoption rates across pricing tiers
* User retention correlation with job security features
* Premium tier conversion driven by career insights
* Customer satisfaction scores for career-related features

**IMPLEMENTATION CODE:**

# Job Security Analysis System - Enhanced Implementation

# Building on existing Mingus career stress tracking

import requests

import json

import datetime

from typing import Dict, List, Optional, Tuple

from dataclasses import dataclass

import sqlite3

import logging

@dataclass

class JobSecurityMetrics:

overall\_score: int

user\_perception\_score: int

external\_data\_score: Optional[int]

confidence\_level: str

risk\_factors: List[str]

recommendations: List[str]

last\_updated: datetime.datetime

class JobSecurityAnalyzer:

"""Enhanced job security analysis integrating multiple data sources"""

def \_\_init\_\_(self, db\_connection):

self.db = db\_connection

self.sec\_base\_url = "https://data.sec.gov/api/xbrl/"

self.headers = {'User-Agent': 'Mingus Financial App (contact@mingus.app)'}

self.weight\_user\_perception = 0.6

self.weight\_external\_data = 0.4

def enhanced\_career\_checkin\_questions(self) -> Dict:

"""Enhanced career check-in building on existing stress question"""

return {

'section': 'career\_wellness',

'questions': [

{

'id': 'job\_stress\_level',

'question': 'How stressful has work been this week?',

'type': 'scale',

'scale': '1-10 (1=No stress, 10=Extremely stressful)',

'existing': True,

'required': True

},

{

'id': 'job\_security\_feeling',

'question': 'How secure do you feel in your current position?',

'type': 'scale',

'scale': '1-10 (1=Very uncertain, 10=Very secure)',

'new': True,

'required': True,

'help\_text': 'Consider your role stability, company health, and industry trends'

},

{

'id': 'employer\_warnings',

'question': 'Have you heard any concerning news about your workplace recently?',

'type': 'multiple\_choice',

'options': [

'Yes, layoff discussions or rumors',

'Budget cuts or hiring freezes mentioned',

'Management changes or restructuring',

'No concerning news',

'Positive news (growth, hiring, etc.)'

],

'new': True,

'required': True

},

{

'id': 'work\_environment\_changes',

'question': 'Any significant changes in your work environment?',

'type': 'multiple\_choice',

'options': [

'Increased workload for same pay',

'Team members leaving',

'New demanding management',

'Technology/process changes',

'No significant changes',

'Positive improvements'

],

'conditional': True,

'show\_if': 'job\_stress\_level > 6 OR job\_security\_feeling < 6',

'new': True

},

{

'id': 'employer\_name',

'question': 'Current employer name (for financial analysis)',

'type': 'text',

'required': False,

'privacy\_note': 'Used only for public company financial health analysis',

'new': True

},

{

'id': 'industry\_sector',

'question': 'What industry do you work in?',

'type': 'dropdown',

'options': [

'Technology/Software',

'Healthcare',

'Financial Services',

'Education',

'Government/Public Service',

'Retail/Consumer Services',

'Manufacturing',

'Professional Services',

'Media/Communications',

'Non-profit',

'Other'

],

'new': True,

'required': True

}

],

'conditional\_questions': {

'trigger\_condition': 'job\_stress\_level > 7 OR job\_security\_feeling < 5',

'additional\_questions': [

{

'id': 'stress\_primary\_source',

'question': 'What\'s your biggest work concern right now?',

'type': 'multiple\_choice',

'options': [

'Job security/layoff risk',

'Overwhelming workload',

'Difficult management',

'Limited career growth',

'Work-life balance',

'Compensation concerns',

'Workplace culture issues'

]

},

{

'id': 'coping\_strategies',

'question': 'How are you currently managing work stress?',

'type': 'multiple\_choice',

'multiple\_select': True,

'options': [

'Talking to friends/family',

'Exercise or physical activity',

'Meditation/mindfulness',

'Professional counseling',

'Looking for new job opportunities',

'Skill building/education',

'Not actively managing it'

]

}

]

}

}

def calculate\_job\_security\_score(self, user\_data: Dict, external\_data: Optional[Dict] = None) -> JobSecurityMetrics:

"""Enhanced scoring combining user input with external data"""

# Calculate user perception score (60% weight)

user\_score = self.\_calculate\_user\_perception\_score(user\_data)

# Calculate external data score (40% weight) if available

external\_score = None

if external\_data:

external\_score = self.\_calculate\_external\_risk\_score(external\_data)

final\_score = (user\_score \* self.weight\_user\_perception +

external\_score \* self.weight\_external\_data)

else:

final\_score = user\_score

# Generate risk factors and recommendations

risk\_factors = self.\_identify\_risk\_factors(user\_data, external\_data)

recommendations = self.\_generate\_recommendations(final\_score, user\_data, risk\_factors)

confidence\_level = self.\_calculate\_confidence\_level(user\_data, external\_data)

return JobSecurityMetrics(

overall\_score=max(0, min(100, int(final\_score))),

user\_perception\_score=int(user\_score),

external\_data\_score=int(external\_score) if external\_score else None,

confidence\_level=confidence\_level,

risk\_factors=risk\_factors,

recommendations=recommendations,

last\_updated=datetime.datetime.now()

)

def \_calculate\_user\_perception\_score(self, user\_data: Dict) -> float:

"""Calculate score based on user's own assessment"""

base\_score = 100

# Job stress impact (higher stress typically correlates with lower security)

stress\_level = user\_data.get('job\_stress\_level', 5)

stress\_penalty = (stress\_level - 1) \* 3 # 0-27 point penalty

# Security feeling (direct correlation)

security\_feeling = user\_data.get('job\_security\_feeling', 5)

security\_bonus = (security\_feeling - 1) \* 7 # 0-63 point bonus

# Warning signs penalty

warnings = user\_data.get('employer\_warnings', 'No concerning news')

warning\_penalty = {

'Yes, layoff discussions or rumors': 30,

'Budget cuts or hiring freezes mentioned': 20,

'Management changes or restructuring': 15,

'No concerning news': 0,

'Positive news (growth, hiring, etc.)': -10 # Bonus for positive news

}.get(warnings, 10)

# Work environment changes impact

env\_changes = user\_data.get('work\_environment\_changes', 'No significant changes')

env\_penalty = {

'Increased workload for same pay': 10,

'Team members leaving': 15,

'New demanding management': 10,

'Technology/process changes': 5,

'No significant changes': 0,

'Positive improvements': -5

}.get(env\_changes, 0)

user\_score = base\_score - stress\_penalty + security\_bonus - warning\_penalty - env\_penalty

return max(0, min(100, user\_score))

def \_calculate\_external\_risk\_score(self, external\_data: Dict) -> float:

"""Calculate score based on external risk factors"""

base\_score = 100

# Employer financial health (if available)

employer\_health = external\_data.get('employer\_financial\_health', 50)

health\_adjustment = (employer\_health - 50) \* 0.6 # ±30 point impact

# Local layoff activity

local\_layoffs = external\_data.get('recent\_local\_layoffs', 0)

layoff\_penalty = min(local\_layoffs \* 3, 30) # Up to 30 point penalty

# Industry trends

industry\_risk = external\_data.get('industry\_risk\_score', 50)

industry\_adjustment = (industry\_risk - 50) \* 0.4 # ±20 point impact

# Economic indicators

economic\_health = external\_data.get('local\_economic\_indicators', 50)

economic\_adjustment = (economic\_health - 50) \* 0.2 # ±10 point impact

external\_score = (base\_score + health\_adjustment - layoff\_penalty +

industry\_adjustment + economic\_adjustment)

return max(0, min(100, external\_score))

def \_identify\_risk\_factors(self, user\_data: Dict, external\_data: Optional[Dict]) -> List[str]:

"""Identify specific risk factors based on data"""

risk\_factors = []

# User-reported risks

if user\_data.get('job\_stress\_level', 0) > 7:

risk\_factors.append("High reported job stress levels")

if user\_data.get('job\_security\_feeling', 10) < 5:

risk\_factors.append("Low confidence in job security")

warnings = user\_data.get('employer\_warnings', '')

if 'layoff' in warnings.lower():

risk\_factors.append("Recent layoff discussions at workplace")

elif 'budget cuts' in warnings.lower():

risk\_factors.append("Budget constraints at employer")

# External data risks

if external\_data:

if external\_data.get('employer\_financial\_health', 50) < 40:

risk\_factors.append("Employer showing financial stress indicators")

if external\_data.get('recent\_local\_layoffs', 0) > 2:

risk\_factors.append("Increased layoff activity in local area")

if external\_data.get('industry\_risk\_score', 50) < 40:

risk\_factors.append("Industry facing headwinds")

return risk\_factors

def \_generate\_recommendations(self, score: float, user\_data: Dict, risk\_factors: List[str]) -> List[str]:

"""Generate personalized recommendations based on score and risk factors"""

recommendations = []

if score < 40: # High risk

recommendations.extend([

"💰 Prioritize building 6-month emergency fund immediately",

"📄 Update resume and LinkedIn profile this week",

"🔍 Begin discreet job search and networking",

"📚 Identify and develop in-demand skills in your field",

"🤝 Strengthen professional network through industry events",

"📊 Research salary benchmarks for your role in other companies"

])

elif score < 60: # Moderate risk

recommendations.extend([

"💡 Increase emergency fund to 4-6 months expenses",

"📈 Monitor industry trends and company news regularly",

"🎯 Consider additional income streams or side projects",

"📱 Keep professional network active with regular check-ins",

"🏆 Document your achievements for performance reviews"

])

elif score < 80: # Low-moderate risk

recommendations.extend([

"✅ Maintain 3-4 month emergency fund",

"📊 Stay informed about company performance and industry trends",

"🚀 Focus on skill development and career advancement",

"🎯 Set clear professional development goals"

])

else: # Low risk

recommendations.extend([

"🎯 Explore long-term career advancement opportunities",

"💰 Focus on retirement planning and investment goals",

"🌟 Consider mentoring others and building leadership skills",

"🔍 Look for stretch assignments and new challenges"

])

# Add specific recommendations based on user responses

if 'job security/layoff risk' in user\_data.get('stress\_primary\_source', ''):

recommendations.insert(0, "🚨 Create 30-day action plan for job search preparation")

if user\_data.get('industry\_sector') in ['Retail/Consumer Services', 'Media/Communications']:

recommendations.append("📱 Consider skills in growing sectors like healthcare or technology")

return recommendations[:6] # Limit to top 6 recommendations

def \_calculate\_confidence\_level(self, user\_data: Dict, external\_data: Optional[Dict]) -> str:

"""Calculate confidence level for the analysis"""

confidence\_factors = 0

# User data completeness

if user\_data.get('job\_stress\_level'):

confidence\_factors += 1

if user\_data.get('job\_security\_feeling'):

confidence\_factors += 1

if user\_data.get('employer\_warnings'):

confidence\_factors += 1

if user\_data.get('industry\_sector'):

confidence\_factors += 1

# External data availability

if external\_data:

if external\_data.get('employer\_financial\_health'):

confidence\_factors += 2

if external\_data.get('recent\_local\_layoffs') is not None:

confidence\_factors += 1

if external\_data.get('industry\_risk\_score'):

confidence\_factors += 1

if confidence\_factors >= 7:

return "High"

elif confidence\_factors >= 5:

return "Medium"

else:

return "Low"

class EmployerFinancialAnalyzer:

"""Analyze employer financial health using SEC EDGAR data"""

def \_\_init\_\_(self):

self.sec\_base\_url = "https://data.sec.gov/api/xbrl/"

self.headers = {'User-Agent': 'Mingus Financial App (contact@mingus.app)'}

def analyze\_employer\_financial\_health(self, employer\_name: str) -> Dict:

"""Analyze employer's financial health using SEC filings"""

try:

# Find company CIK

cik = self.\_find\_company\_cik(employer\_name)

if not cik:

return self.\_handle\_private\_company(employer\_name)

# Get latest financial data

financial\_data = self.\_get\_company\_financial\_facts(cik)

# Calculate health score

health\_score = self.\_calculate\_financial\_health\_score(financial\_data)

return {

'status': 'success',

'company\_name': financial\_data.get('company\_name'),

'cik': cik,

'financial\_health\_score': health\_score['overall\_score'],

'risk\_indicators': health\_score['risk\_indicators'],

'last\_filing\_date': financial\_data.get('last\_filing\_date'),

'analysis\_date': datetime.datetime.now().isoformat(),

'confidence': 'high'

}

except Exception as e:

return {

'status': 'error',

'error\_message': str(e),

'fallback\_score': 50,

'confidence': 'low'

}

def \_find\_company\_cik(self, employer\_name: str) -> Optional[str]:

"""Find company's CIK using SEC company database"""

# Implementation for CIK lookup

# This would search the SEC company database

# For brevity, returning None to trigger private company handling

return None

def \_handle\_private\_company(self, employer\_name: str) -> Dict:

"""Handle analysis for private companies"""

return {

'status': 'private\_company',

'company\_name': employer\_name,

'financial\_health\_score': 50, # Neutral score

'risk\_indicators': ['Limited public financial data available'],

'confidence': 'low',

'note': 'Private company - using industry and local market data only'

}

def \_get\_company\_financial\_facts(self, cik: str) -> Dict:

"""Retrieve company financial facts from SEC EDGAR"""

# Implementation for SEC API calls

pass

def \_calculate\_financial\_health\_score(self, financial\_data: Dict) -> Dict:

"""Calculate financial health score from SEC data"""

# Implementation for financial health scoring

pass

class LocalLaborMarketAnalyzer:

"""Analyze local labor market conditions and layoff activity"""

def \_\_init\_\_(self):

self.warn\_databases = {

'CA': 'https://edd.ca.gov/en/jobs\_and\_training/layoff\_services\_warn',

'WA': 'https://esd.wa.gov/newsroom/warn-list',

'MD': 'https://www.dllr.state.md.us/employment/warn.shtml',

'IL': 'https://www2.illinois.gov/dceo/WorkforceDevelopment/warn',

# Add more states as needed

}

def analyze\_local\_risk(self, user\_zip\_code: str, industry: str) -> Dict:

"""Analyze local labor market risk factors"""

try:

state = self.\_get\_state\_from\_zip(user\_zip\_code)

# Get recent WARN notices

local\_layoffs = self.\_get\_local\_warn\_activity(state, user\_zip\_code)

# Get BLS local unemployment data

unemployment\_data = self.\_get\_local\_unemployment\_data(user\_zip\_code)

# Calculate local risk score

risk\_score = self.\_calculate\_local\_risk\_score(

local\_layoffs, unemployment\_data, industry

)

return {

'status': 'success',

'recent\_local\_layoffs': len(local\_layoffs),

'local\_unemployment\_rate': unemployment\_data.get('rate', 5.0),

'local\_risk\_score': risk\_score,

'analysis\_date': datetime.datetime.now().isoformat()

}

except Exception as e:

return {

'status': 'error',

'recent\_local\_layoffs': 0,

'local\_risk\_score': 50, # Neutral

'error\_message': str(e)

}

def \_get\_state\_from\_zip(self, zip\_code: str) -> str:

"""Get state abbreviation from ZIP code"""

# Implementation for ZIP to state lookup

pass

def \_get\_local\_warn\_activity(self, state: str, zip\_code: str) -> List[Dict]:

"""Get recent WARN notices for local area"""

# Implementation for WARN database scraping/API calls

pass

def \_get\_local\_unemployment\_data(self, zip\_code: str) -> Dict:

"""Get local unemployment data from BLS"""

# Implementation for BLS API calls

pass

def \_calculate\_local\_risk\_score(self, layoffs: List[Dict],

unemployment: Dict, industry: str) -> int:

"""Calculate local labor market risk score"""

# Implementation for local risk scoring

pass

# Database schema updates for job security features

def create\_job\_security\_tables(db\_connection):

"""Create necessary tables for job security analysis"""

cursor = db\_connection.cursor()

# Enhanced career check-ins table

cursor.execute('''

ALTER TABLE health\_checkins

ADD COLUMN job\_security\_feeling INTEGER CHECK (job\_security\_feeling BETWEEN 1 AND 10),

ADD COLUMN employer\_warnings TEXT,

ADD COLUMN work\_environment\_changes TEXT,

ADD COLUMN employer\_name TEXT,

ADD COLUMN industry\_sector TEXT,

ADD COLUMN stress\_primary\_source TEXT,

ADD COLUMN coping\_strategies TEXT

''')

# Job security analysis results table

cursor.execute('''

CREATE TABLE IF NOT EXISTS job\_security\_analysis (

id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id TEXT NOT NULL,

analysis\_date DATE NOT NULL,

overall\_score INTEGER CHECK (overall\_score BETWEEN 0 AND 100),

user\_perception\_score INTEGER,

external\_data\_score INTEGER,

confidence\_level TEXT,

risk\_factors TEXT, -- JSON array

recommendations TEXT, -- JSON array

employer\_financial\_score INTEGER,

local\_risk\_score INTEGER,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(id)

)

''')

# Employer analysis cache table

cursor.execute('''

CREATE TABLE IF NOT EXISTS employer\_analysis\_cache (

id INTEGER PRIMARY KEY AUTOINCREMENT,

employer\_name TEXT UNIQUE NOT NULL,

cik TEXT,

financial\_health\_score INTEGER,

risk\_indicators TEXT, -- JSON array

last\_updated TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

data\_source TEXT -- 'sec\_edgar', 'private\_