V3 Database Schema Guide

📊 TABLE OVERVIEW & IMPORTANCE

🔐 access_control

Purpose: User authentication and API access management **Importance**: Critical for security - controls who can upload/process documents **Key Use Cases**:

- API authentication via `api_key`
- User attribution for document uploads and reviews
- Audit trail tracking

```sql
-- Find who uploaded the most documents
SELECT ac."user", COUNT(*) as uploads
FROM raw_documents rd
JOIN access_control ac ON rd.uploaded_by = ac.access_id
GROUP BY ac."user" ORDER BY uploads DESC;
```

🏢 companies

Purpose: Multi-tenant support for different organizations **Importance**: Enables SaaS model - isolate data between different companies **Key Use Cases**:

- Data segregation between clients
- Company-specific analytics and reporting
- Billing and usage tracking per organization

```sql
-- Get document processing stats per company
SELECT c.company_name, COUNT(rd.*) as total_docs,
       COUNT(pd.*) as processed_docs
FROM companies c
LEFT JOIN raw_documents rd ON c.company_id = rd.company
LEFT JOIN processed_documents pd ON rd.document_id = pd.document_id
GROUP BY c.company_name;
```

🤖 model_versions

Purpose: Track different AI models and their performance over time **Importance**: Essential for model governance and A/B testing **Key Use Cases**:

- Compare accuracy between model versions
- Rollback to previous model if needed
- Track model deployment history
- Store model parameters and metrics

```sql
-- Compare model performance
SELECT mv.model_name, mv.version,
       COUNT(*) as predictions_made,
       AVG(ta.avg_score) as avg_confidence
FROM model_versions mv
JOIN processed_documents pd ON mv.model_id = pd.model_id
JOIN tag_analytics ta ON true -- Cross join for overall stats
GROUP BY mv.model_name, mv.version;
```

---

raw_documents

Purpose: Store metadata about uploaded files (the source documents) **Importance**: Foundation table - everything starts with document upload **Key Use Cases**:

- File management and organization
- Duplicate detection via `file_hash`
- Track upload progress via `status`
- Link files to companies and users

```sql
-- Find duplicate files
SELECT file_hash, COUNT(*) as duplicates,
       array_agg(document_name) as duplicate_files
FROM raw_documents
GROUP BY file_hash
HAVING COUNT(*) > 1;

-- Monitor upload status
SELECT status, COUNT(*) as count
FROM raw_documents
GROUP BY status;
```

⚙️ processed_documents (MOST IMPORTANT TABLE)

Purpose: Store AI predictions AND user confirmations - the heart of your workflow **Importance**: Contains the complete user journey from AI suggestion to final confirmation **Key Use Cases**:

- Track what AI suggested vs what users confirmed
- Identify model improvement opportunities
- User workflow management (pending reviews)
- Training data preparation

```sql
-- Documents pending user review
SELECT rd.document_name, pd.suggested_tags, pd.processing_date
FROM raw_documents rd
JOIN processed_documents pd ON rd.document_id = pd.document_id
WHERE pd.user_reviewed = false
ORDER BY pd.processing_date DESC;

-- Compare AI suggestions vs user confirmations
SELECT pd.suggested_tags, pd.confirmed_tags, pd.user_added_labels
FROM processed_documents pd
WHERE pd.user_reviewed = true;

-- Find most rejected AI suggestions (model improvement insights)
SELECT unnest(user_removed_tags) as rejected_tag, COUNT(*) as frequency
FROM processed_documents
WHERE user_removed_tags IS NOT NULL
GROUP BY rejected_tag ORDER BY frequency DESC;
```

📦 processing_batches

Purpose: Handle batch uploads and track overall processing status **Importance**: Essential for managing multiple file uploads and monitoring system health **Key Use Cases**:

- Track batch processing progress
- Handle failed batch processing
- System performance monitoring
- User experience (show progress bars)

```sql
```

```sql
-- Monitor batch processing health
SELECT status, COUNT(*) as batches,
       AVG(processed_documents::float / total_documents * 100) as avg_success_rate
FROM processing_batches
GROUP BY status;

-- Find stuck batches
SELECT * FROM processing_batches
WHERE status = 'processing'
AND created_at < NOW() - interval '1 hour';
```

📝 logs

Purpose: Audit trail for all system activities **Importance**: Critical for debugging, compliance, and security monitoring **Key Use Cases**:

- Debug processing failures
- Security audit (who did what when)
- Performance analysis
- Compliance reporting

```sql
sql

-- Recent system activity
SELECT action_type, COUNT(*) as frequency
FROM logs
WHERE action_date > NOW() - interval '24 hours'
GROUP BY action_type ORDER BY frequency DESC;

-- Failed operations
SELECT action_type, action_details, action_date
FROM logs
WHERE success = false
ORDER BY action_date DESC LIMIT 10;
```

🔍 document_access_logs (Optional)

Purpose: Track who accessed which documents when **Importance**: Security monitoring and usage analytics **Key Use Cases**:

- Security compliance
- User behavior analytics
- Access pattern analysis

🚀 INDEX USAGE GUIDE

Performance Indexes

sql

```sql
-- These queries will be FAST due to indexes:

-- ✅ Filter by document status (uses idx_raw_documents_status)
SELECT * FROM raw_documents WHERE status = 'uploaded';

-- ✅ Filter by company (uses idx_raw_documents_company)
SELECT * FROM raw_documents WHERE company = 123;

-- ✅ Recent uploads (uses idx_raw_documents_upload_date)
SELECT * FROM raw_documents WHERE upload_date > NOW() - interval '7 days';

-- ✅ Pending reviews (uses idx_processed_documents_user_reviewed)
SELECT * FROM processed_documents WHERE user_reviewed = false;

-- ✅ Documents by status (uses idx_processed_documents_status)
SELECT * FROM processed_documents WHERE status = 'user_confirmed';
```

JSONB/Array Indexes (GIN)

sql

```sql
-- These complex queries will be FAST due to GIN indexes:

-- ✅ Find documents with specific suggested tags
SELECT * FROM processed_documents
WHERE suggested_tags @> '[{"tag": "finance"}]';

-- ✅ Find documents with specific confirmed tags
SELECT * FROM processed_documents
WHERE 'urgent' = ANY(confirmed_tags);

-- ✅ Find documents with user-added labels
SELECT * FROM processed_documents
WHERE user_added_labels && ARRAY['priority', 'FY2024'];

-- ✅ Documents with specific errors
SELECT * FROM processed_documents
WHERE errors @> ARRAY['OCR failed'];
```

📈 ANALYTICS MATERIALIZED VIEW GUIDE

What is `tag_analytics`?

A pre-computed summary of all tags and their performance metrics across your entire system.

Columns Explained:

- `tag_name`: The actual tag (e.g., "finance", "legal")
- `frequency`: How many times this tag was suggested by AI
- `avg_score`: Average confidence score for this tag
- `min_score`: Lowest confidence score seen
- `max_score`: Highest confidence score seen

Key Use Cases:

1. Model Performance Analysis

```sql
-- Which tags does the AI predict most confidently?
SELECT tag_name, avg_score, frequency
FROM tag_analytics
WHERE frequency > 5  -- Only tags used frequently
ORDER BY avg_score DESC;

-- Tags with inconsistent confidence (high variance)
SELECT tag_name, frequency,
       (max_score - min_score) as score_variance
FROM tag_analytics
WHERE frequency > 3
ORDER BY score_variance DESC;
```

2. Content Analytics

```sql
```

```sql
-- Most common document types in your system
SELECT tag_name, frequency
FROM tag_analytics
ORDER BY frequency DESC
LIMIT 10;

-- Rare/unusual content (might need model training)
SELECT tag_name, frequency, avg_score
FROM tag_analytics
WHERE frequency <= 2
ORDER BY avg_score DESC;
```

3. Model Training Insights

```sql
-- Tags that might need more training data (low confidence)
SELECT tag_name, frequency, avg_score
FROM tag_analytics
WHERE avg_score < 0.7 AND frequency > 3
ORDER BY frequency DESC;

-- High-confidence tags (working well)
SELECT tag_name, frequency, avg_score
FROM tag_analytics
WHERE avg_score > 0.9
ORDER BY frequency DESC;
```

4. Business Intelligence

```sql
-- What types of documents are most common in your organization?
SELECT tag_name, frequency,
    ROUND(frequency * 100.0 / SUM(frequency) OVER(), 2) as percentage
FROM tag_analytics
ORDER BY frequency DESC;
```

Refreshing the Analytics View

The materialized view needs to be refreshed when new data is processed:

```sql
```

```sql
-- Manual refresh (run this after processing new documents)
SELECT refresh_tag_analytics();

-- Check when it was last refreshed
SELECT schemaname, matviewname, last_refresh
FROM pg_stat_user_tables
WHERE relname = 'tag_analytics';
```

Automation Tip

Consider setting up a periodic refresh:

```sql
-- Example: Refresh every hour (would need cron job or scheduler)
-- SELECT refresh_tag_analytics();
```

🎯 PRACTICAL WORKFLOWS

Daily Operations Dashboard

```sql
-- System health check
SELECT
  (SELECT COUNT(*) FROM raw_documents WHERE status = 'uploaded') as pending_docs,
  (SELECT COUNT(*) FROM processed_documents WHERE user_reviewed = false) as pending_reviews,
  (SELECT COUNT(*) FROM processing_batches WHERE status = 'processing') as active_batches,
  (SELECT COUNT(*) FROM logs WHERE action_date > NOW() - interval '24 hours' AND success = false) as recent_errors
```

Weekly Analytics Report

```sql
-- Business metrics
SELECT
  COUNT(DISTINCT rd.document_id) as total_documents_processed,
  COUNT(DISTINCT pd.user_id) as active_reviewers,
  AVG(EXTRACT(epoch FROM (pd.reviewed_at - pd.processing_date))/3600) as avg_review_time_hours,
  (SELECT COUNT(*) FROM tag_analytics) as unique_tag_types
FROM raw_documents rd
JOIN processed_documents pd ON rd.document_id = pd.document_id
WHERE rd.upload_date > NOW() - interval '7 days';
```

This structure gives you a powerful document processing and analytics system that grows with your business! 🚀